1 2 9 0

UNIVERSIDADE Ð
COIMBRA

Dinis Silva Costa Carvalho

Management tools of the re-commerce system LoopOS

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, developed in The Loop co. under the supervision of Eng. João Rodrigues and advised by Professor António Dourado from DEI and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

January 2024

1 2 9 0

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Đ
COIMBRA

Dinis Silva Costa Carvalho

Management tools of the re-commerce system LoopOS

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, developed in The Loop co. under the supervision of Eng. João Rodrigues and advised by Professor António Dourado from DEI and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

January 2024

1 2 9 0

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Đ
COIMBRA

Dinis Silva Costa Carvalho

Ferramentas de gestão do sistema de re-commerce LoopOS

Dissertação no âmbito do Mestrado em Engenharia Informática, com especialização em Engenharia de Software, desenvolvida na The Loop co., orientada pelo Eng. João Rodrigues e pelo Professor Antonio Dourado do DEI e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Janeiro 2024

Acknowledgements

First, I would like to thank my family, especially my parents, and brothers, for supporting me throughout my academic journey and consistently believing in me.

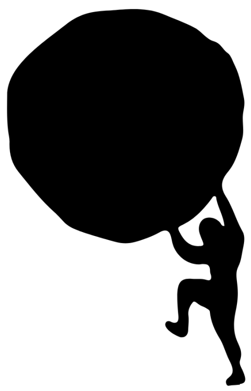
I want to thank Inês Silva, whose unwavering belief in my abilities has consistently pushed me to strive and overcome the greatest challenges. I am eternally grateful for her support and presence in my life.

I am immensely grateful to João Rodrigues, Marta Mercier, and the entire LoopOS team for allowing me to participate in this enriching project. Their belief in my capabilities and support throughout my involvement has been instrumental in my personal and professional growth.

I extend my heartfelt appreciation to my teammates in Team R, whose dedication, hard work, and camaraderie have made this journey truly rewarding. Working alongside them has been an enjoyable experience filled with intellectual stimulation and shared camaraderie.

I wish to express my sincere gratitude to Professor António Dourado for his availability and valuable guidance throughout this project. His expertise and insights have significantly shaped the project's direction and ensured its success.

Last but certainly not least, I cannot thank enough my lifelong friends: João Teixeira, Frederico Macedo, Alexandre Santos, Abdellahi Brahim, Filip van der Kroft, and Iggy Kepka. The best friendship has to offer. Without your presence, I would not be the person I am today.



Abstract

Existing retail channels have been optimized to push items through the supply chain and deliver them to the customers. However, the rise of e-commerce and its associated benefits, such as easy returns and circular economy opportunities, have created a problem for retailers who have not adapted to this reverse flow. As a result, the process of recovering items from customers is often manual and inefficient.

The Loop co.[15] is developing LoopOS, a circular economy framework, to solve this problem. LoopOS integrates different aspects of the circular economy, such as product submission, validation, delivery, certification, refurbishment, and re-entry into the market, into a single, decentralized, and collaborative ecosystem.

This thesis explores the development process and implementation of two critical components within the LoopOS environment: LoopOS Onboarding, which focuses on simplifying the onboarding process for new users within the LoopOS platform while also automating their environment creation within LoopOS, and LoopOS UI, designed to standardize and enhance front-end development across various LoopOS applications. The former addresses challenges in asset pipelines and front-end development. It ensures seamless integration and modularity for LoopOS developers, enabling them to focus on core functionality rather than front-end complexities. The LoopOS UI significantly contributes to the modular architecture, providing a solid foundation for implementing a new user interface in LoopOS applications.

In conclusion, this thesis achieves its stated objectives and showcases a comprehensive approach to software development, from planning and execution to innovation and future vision. The implemented solutions contribute to the ongoing development of the LoopOS platform, highlighting the adaptability, collaboration, and problem-solving skills crucial in a dynamic software development environment.

Keywords

Circular Economy, Web Application, Application Deployment & Management, Partner Onboarding, Front-end Development

Resumo

Os canais de venda a retalho existentes foram otimizados para levar os itens ao longo da cadeia de fornecimento e entregá-los aos clientes. No entanto, o aumento do comércio eletrónico e dos seus benefícios associados, como devoluções fáceis e oportunidades de economia circular, criaram um problema para os retalhistas que não se adaptaram a esse fluxo reverso. Como resultado, o processo de recuperação de itens dos clientes é frequentemente manual e ineficiente.

A Loop co.[15] está a desenvolver o LoopOS, um sistema de economia circular, para resolver este problema. O LoopOS integra diferentes aspectos da economia circular, como a submissão de produtos, validação, entrega, certificação, renovação e reentrada no mercado, num ecossistema único, descentralizado e colaborativo.

Esta tese explora o processo de desenvolvimento e instanciação de dois componentes fundamentais no ambiente LoopOS: o LoopOS Onboarding, que se concentra na simplificação do processo de onboarding para novos usuários tal como a automatização da criação dos seus ambientes dentro do LoopOS, e o LoopOS UI projetado para padronizar e aperfeiçoar o desenvolvimento front-end em vários aplicações do LoopOS. Este aborda desafios em “asset pipeline” e desenvolvimento front-end. Ele garante integração e modularidade, permitindo que os desenvolvedores da LoopOS se concentrem na funcionalidade principal em vez das complexidades do front-end. O LoopOS UI contribui significativamente para a arquitetura modular, fornecendo uma base sólida para implementar uma nova interface do usuário em aplicações do LoopOS.

Em conclusão, esta tese atinge os seus objetivos declarados e mostra uma abordagem detalhada ao desenvolvimento de software, do planeamento à execução, da inovação à visão do futuro. As soluções implementadas contribuem para o desenvolvimento contínuo da plataforma LoopOS, destacando a adaptabilidade, colaboração e habilidades de solução de problemas num ambiente de desenvolvimento de software dinâmico.

Palavras-Chave

Economia Circular, Aplicação Web, Autenticação & Autorização, Instanciação e Gestão de Aplicações, Integração de Parceiros, Desenvolvimento Front-end, Pipeline de Recursos

Contents

Acronyms

API	Application Programming Interface.
CERN	European Organization for Nuclear Research.
CI/CD	Continuous Integration/Continuous Delivery.
CTO	Chief Technology Officer.
DEI	Departamento de Engenharia Informática.
ESA	European Space Agency.
HR	Human Resources.
NIF	Fiscal Identification Number.
PM	Product Manager.
PO	Project Owner.
QA	Quality Assurance.
R&D	Research and Development.
RCE	Remote Code Execution.
SSO	Single Sign-On.
UI	User Experience.

List of Figures

List of Tables

Chapter 1

Introduction

The present dissertation was carried out in an academic-enterprise environment as part of the Dissertation/Internship in Software Engineering, with a Master's degree in Informatics Engineering and a specialization in Software Engineering at the Faculty of Sciences and Technology of the University of Coimbra.

The internship took place in the company The Loop co.[15] under the supervision of Eng. João Rodrigues and Professor António Dourado from Departamento de Engenharia Informática (DEI), where it initially consisted of the improvement of the management application in LoopOS[16] (LoopOS Network Manager), but became two unique products within LoopOS: LoopOS Onboarding and LoopOS UI.

The motivation (Section 1.1), the objectives (Section 1.2), and the structure of this thesis (Section 1.3) will be presented in this chapter.

1.1 Motivation

The Loop Co. is a technology company founded in 2016 with the circular economy project Book in Loop [14], a pioneering project in reusing school textbooks in Portugal. Since then, The Loop Co. has expanded its areas of activity, now dedicating itself to two fundamental areas: Circular Economy and Research and Development (R&D).

Currently, the company works with major national and international partners, such as Sonae, Portuguese Red Cross, European Organization for Nuclear Research (CERN), and European Space Agency (ESA) on technological innovation projects and business development, focusing on green tech.

Retailers face challenges when managing returns and entering the resale market because traditional retail channels are designed for a uni-directional flow of goods. With the rise of e-commerce and circular economy, The Loop co. recognizes the need for a plug-and-play solution that simplifies refurbishing and reselling. To achieve this, it created a centralized system that automates the management of all returns, enabling each item to be directed to its optimal destination - LoopOS. This will connect customers, retailers, and warehouses on a single platform, streamlining the return process and simplifying entry into the resale market.

LoopOS is a valuable tool for retailers looking to streamline their return process and those looking to enter the resale market. For retailers already engaged in circular economy initiatives, it will make expanding their operations easier. Although there may be variations in how clients use the platform, LoopOS strives to design a system that minimizes these differences.

Therefore, by standardizing the process of buying second-hand items, LoopOS intends to achieve several benefits:

- ❖ A **positive environmental impact** by reducing the number of items that end up in landfills.
- ❖ A **social impact** by providing more options for people to receive a fair price for their used items.
- ❖ A **business impact** by enabling retailers to establish a profitable niche market.

LoopOS addresses these challenges through the development of multiple applications, such as LoopOS Submission (for customers to submit the items) and LoopOS Validation (for workers of the Partner using LoopOS to validate these items), each managing specific steps in the item return and refurbishment process. However, as multiple applications are instantiated per partners, a management issue arises in handling instances, partners, users, and other aspects of this technological framework.

To tackle this problem, LoopOS introduced the Network Manager, serving as the orchestrator and manager of the entire LoopOS system. While the initial version of the Network Manager successfully allowed manual user, partner, and application instantiation from a centralized location, it encountered two significant issues as the client base expanded:

- The manual instantiation of environments for partners became impractical and unfeasible.
- The prioritizing functionality over user-friendliness in the Network Manager to speed its completion, posed difficulties for users unfamiliar with the system.

This thesis aims to address these challenges.

1.2 Objectives

As stated before, my goal for this thesis is to solve two major issues with the Network Manager by streamlining partner onboarding and environment instantiation and enhancing the user-friendliness of the Network Manager.

The following are the established objectives:

- Automated deployment and management of applications, clusters, and database servers through the Network Manager.
- Creation of templates for easy configuration and setup of various environments.
- Development of a self-onboarding feature for Partners to create their online platform.
- Implementation of new UI.

1.3 Structure

The thesis is structured in several chapters.

2. **Background:**
   This chapter provides an introduction to the LoopOS project, its objectives, and the context in which it operates. It outlines the fundamental concepts and motivations that form the basis of this thesis.

3. **State of the Art:**
   In this chapter, we explore the current state of technologies and practices in the domain of LoopOS. It provides an in-depth analysis of existing solutions, frameworks, and methodologies relevant to the project.

4. **Requirements & Risks Analysis:**
   This chapter delves into a detailed analysis of the project's requirements, including functionalities and features crucial for the successful implementation of LoopOS. It also addresses potential risks and challenges associated with the project.

5. **Framework:**
   The chosen framework for LoopOS is presented in this chapter. It outlines the structural approach and methodology guiding the development process of LoopOS.

6. **Development:**
   The central chapter documents the development process of LoopOS Onboarding and LoopOS UI. It highlights key milestones, challenges, and decision-making processes. Technical aspects and coding practices are discussed to provide insight into their construction.

8. **Testing:**
   Focused on the testing phase, this chapter explores the strategies and methodologies employed to ensure the proper functionality, security, and performance of the developed applications.

8. **Conclusion:**
   The final chapter presents a comprehensive overview of the project, and potential future developments, concluding this thesis on a reflective note.

Chapter 2

Background

Firstly, to grasp the context of this thesis thoroughly, it is essential to understand the principles of circularity within a circular economy. This understanding will then serve as a foundation for comprehending the architectural choices made in LoopOS and the requisite features it encompasses.

Section 2.1 will, therefore, examine the drawbacks of the traditional linear resource consumption model and explore the advantages of transitioning to a circular economy model. Moreover, it will establish the relevance of this discussion to LoopOS and how its implementation takes advantage of the principles of circularity.

Section 2.2 will give context on the methodology used in the project and, consequently, in this thesis.

Section 2.4 will give insight into the LoopOS overall architecture and a couple of use cases to better understand how users interact with the system.

Section 2.4 will conclude this chapter by delving into the architectural complexities and operational nuances of the Network Manager, defining its current state, in order for an analysis of how it can be improved by comparing it to other technological solutions can be conducted in the State of the Art chapter.

2.1 Circular Economy

Our industrial economy remains rooted in a linear model despite its evolution and diversification. Materials are extracted from the Earth, manufactured into products, sold to consumers, and eventually thrown away as waste when they no longer serve their purpose [33].

This linear resource consumption model, established in the early days of industrialization, has exposed many companies to risks, including higher prices and supply chain disruptions, leading to an increasing number of businesses recognizing the need for change. These companies feel the pressure of unpredictable costs, heightened competition, and stagnant demand. This trend towards higher resource prices started around the turn of the millennium, effectively reversing a century-long decline in the real costs of natural resources as observed in the graph in Figure 2.1.

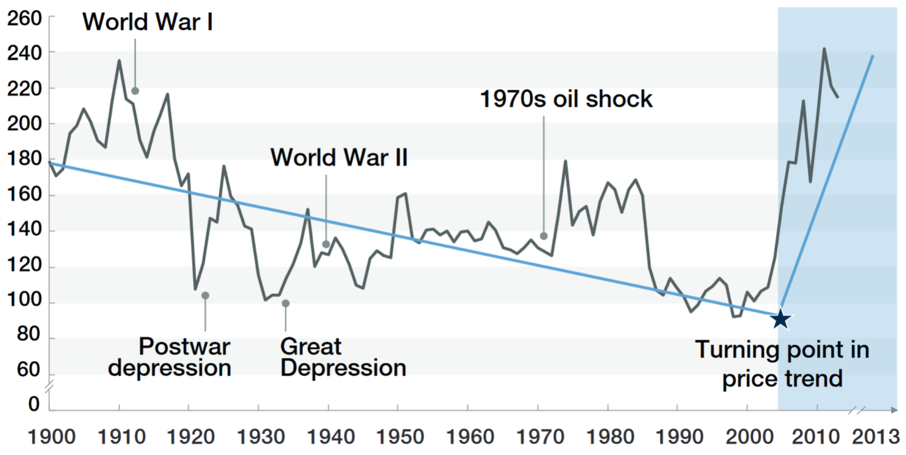


Figure 2.1: Sharp price increases in commodities since 2000 have erased all the real price declines of the 20th century [33].

Resource extraction and utilization, heavily reliant on fossil fuels, contributes to high energy consumption and greenhouse gas emissions, posing a global challenge [2]. The European Commission reports that resource extraction and processing account for 90% of biodiversity loss, the design phase determines up to 80% of product environmental impact, and only 11.7% of all materials used in the EU came from recycled waste [21]. Furthermore, according to the US Environmental Protection Agency, approximately 60 million metric tons deposited in landfills each year of E-waste pose significant risks to humans, wildlife, and ecosystems [25].

Scholarly literature offers several definitions of circular economy [45]. However, for this thesis, I have decided to reference the description provided by the Ellen MacArthur Foundation. This UK-based charity organization aims to hasten the transition towards a circular economy. They define the circular economy as an industrial system that aims to be restorative or regenerative by design and estimates material savings worth over one trillion dollars [33]. The foundation has developed the butterfly diagram to illustrate the two main cycles of the circular economy. Figure 2.2 showcases these two cycles: the technical cycle, which focuses on reusing, repairing, remanufacturing, and recycling, and the biological cycle, which emphasizes returning nutrients from biodegradable materials to regenerate nature.

The adoption of the circular economy model is rapidly increasing globally due to its various benefits. These advantages include its ability to impact the environment positively, the availability of government incentives, the promotion of economic stability through the creation of resilient value chains, and the generation of new business opportunities and high-quality jobs resulting in improved social welfare [21]. For example, China enacted a Circular Economy Promotion Law in 2008 [2], and the European Union established a Circular Economy Strategy in 2015 to transition towards a more resource-efficient economy. As of 2023, the European Union has developed a comprehensive Circular Economy Framework, which includes the European Green Deal [18], the Circular Economy Action Plan [19], and the Sustainable Product Initiative [20].

Additionally, several capital investments, like the Blackrock Global Fund, the Circu-

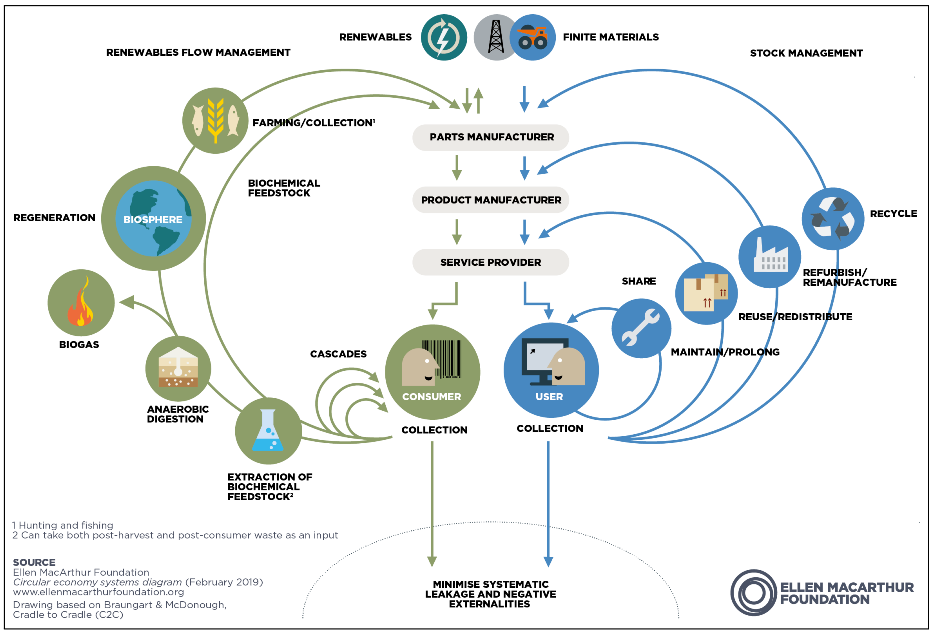

RENEWABLES
FINITE MATERIALS
RENEWABLES FLOW MANAGEMENT
STOCK MANAGEMENT
FARMING/COLLECTION¹
PARTS MANUFACTURER
BIOCHEMICAL FEEDSTOCK
PRODUCT MANUFACTURER
REGENERATION
BIOSPHERE
RECYCLE
SERVICE PROVIDER
REFURBISH/ REMANUFACTURE
SHARE
REUSE/REDISTRIBUTE
BIOGAS
MAINTAIN/PROLONG
CASCADES
CONSUMER
USER
ANAEROBIC DIGESTION
COLLECTION
COLLECTION
EXTRACTION OF BIOCHEMICAL FEEDSTOCK²
1 Hunting and fishing
2 Can take both post-harvest and post-consumer waste as an input
SOURCE
Ellen MacArthur Foundation
Circular economy systems diagram (February 2019)
www.ellenmacarthurfoundation.org
Drawing based on Braungart & McDonough, Cradle to Cradle (C2C)
MINIMISE SYSTEMATIC LEAKAGE AND NEGATIVE EXTERNALITIES
ELLEN MACARTHUR FOUNDATION

Figure 2.2: Butterfly diagram of the circular economy [33].

lar Innovation Fund, and the Alphabet Corporate Bond, have aided the shift toward a circular economy [2].

In conclusion, by developing a plug-and-play centralized system that allows retailers to access the second-hand market and handle returns, reducing waste and ensuring that materials are reused and recycled effectively, LoopOS benefits the circular economy. This approach promotes sustainability and responsibility in the retail industry by assisting retailers in reducing their environmental impact while also promoting new jobs [25]. By facilitating the expansion of the second-hand market, LoopOS encourages the development of new jobs in the retail sector, such as refurbishment, quality control, and customer support, contributing to economic growth and social well-being.

2.2 Methodology

It is essential to delve into the methodology underpinning this project since it significantly influenced decisions on problem-solving and planning timelines.

The Agile methodology was used to develop all applications in LoopOS. Agile methodologies are characterized by an iterative and incremental process, simplicity and ease of adoption, collaboration between all parties involved, and the ability to produce high-quality software within given requirements, budgets, and timelines [43]. In 2001, the Agile Manifesto was created to provide a set of principles and values for software development teams to follow to improve their final products. It emphasizes customer satisfaction through early and continuous delivery of valuable software, welcomes changing requirements even late in development, and delivers working software frequently with a preference for a shorter timescale [10].

The closest agile-based software development method used was Scrum, a widely used and practical approach to software development that emphasizes flexibility and addresses common development challenges.[43] It follows a development lifecycle that includes a series of month-long iterations called Sprints, making it ideal for projects with emergent requirements. The framework outlines specific roles for team members and provides guidelines for how these roles should function in a Scrum environment. While Scrum is a lightweight methodology, it includes more rules than Agile [58]. For example, it recommends having daily meetings to improve communication and dividing work into Sprints to improve project efficiency [58].

The three primary roles in Scrum are the Product Owner, Scrum Master, and Developers. The Product Owner manages and prioritizes the Product Backlog, while the Scrum Master ensures the project runs smoothly and enforces Scrum practices and rules. Developers are responsible for developing the product and achieving the Sprint Goal. The Customer also creates the Product Backlog and provides ideas and information for system features [43] [58].

It is important to note that the chosen methodology in LoopOS introduces an element of unpredictability. The nature of Agile and Scrum makes it challenging to predict all aspects of the project in advance. The methodology's volatility means that objectives and priorities could undergo changes sprint by sprint, allowing for adaptability, new requirements, and evolving project dynamics. This was the case with this thesis, where the objectives became more precise with the course of the project, and solutions to accomplish them changed. While initially, the sole focus was on improving the Network Manager, this gave rise to the creation of LoopOS Onboarding, an application within LoopOS specifically designed to address partner onboarding and environment management, and LoopOS UI, a Rails Engine (an application inside another application to extent its functionalities) to enhance the UI across all LoopOS applications.

2.3 LoopOS

LoopOS is not a single piece of software. Multiple applications created separately by The Loop Co. work together to fulfill the system's goals. These apps consist of:

- **LoopOS Submission:** allows the customer to introduce their product into the system.
- **LoopOS Validation:** assists in determining whether the product meets acceptance criteria.
- **LoopOS Hubs:** serves as a mediator for physical interactions so the client can deliver their product if accepted.
- **LoopOS Handling:** for managing reverse logistics and circular economy processes.
- **LoopOS Exits:** offers options to reintroduce products into the market.

- **LoopOS Core:** acts as a central control panel for configuring and parameterizing the entire product journey within the system, regarding product categories, submission processes, validation criteria, pricing rules, and logistics options.

The aforementioned applications can be deployed individually for each specific use case, catering to the unique requirements of each partner (the companies that desire to use the system). For instance, a large corporation might seek to integrate all available applications into its business operations, enabling comprehensive support for buying and selling used items. Alternatively, a local shop might utilize only the Handling and Core applications, focusing on efficient stock management.

At the center of the entire LoopOS ecosystem is the LoopOS Network Manager. In contrast with the other applications, the Network Manager is only deployed once. It plays a crucial role in the infrastructure deployment and management of all applications across the various partners. Additionally, it ensures the authentication of both users and applications and allows the custom configuration of the applications to cater to each partner's needs.

Moreover, although the Network Manager is not yet available for production (which means only developers use it), it will adapt its functionality based on the user's role. For example, Partners will use the app to oversee and manage their applications and have authority over their users. This includes grouping users and assigning specific roles to regulate access to applications. While users will use the Network Manager to edit their profile

Figure 2.3 illustrates the LoopOS architecture to better visualize the relation between apps.

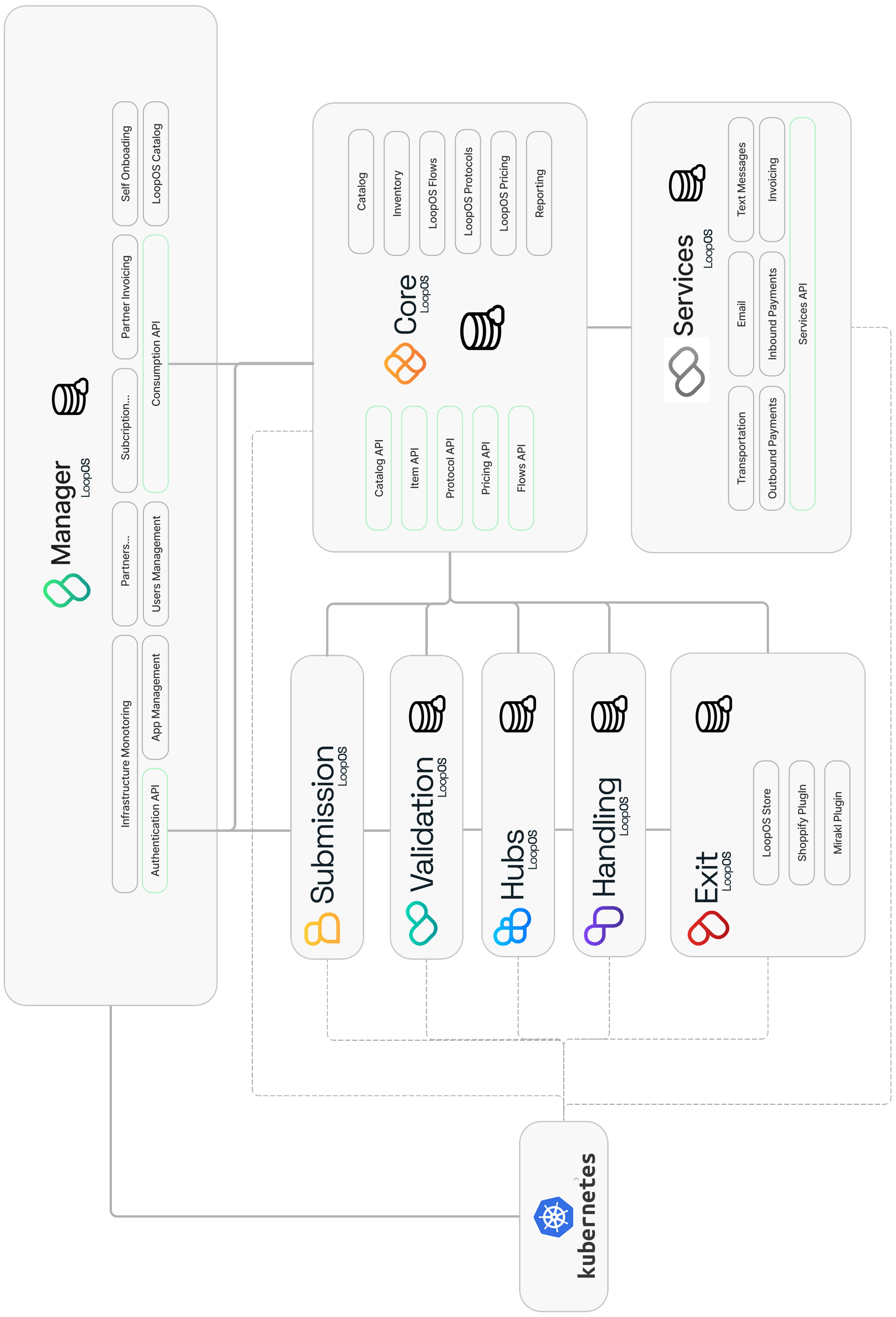


Figure 2.3: LoopOS Architecture.

To gain a deeper understanding of how a product operates within the LoopOS ecosystem, it is crucial to comprehend its journey within a system that uses all of the applications:

1. **Getting a product into the system:**

   The **LoopOS Submission App** allows consumers to input information about a product they want to sell or return. It streamlines the process by requesting product information and condition details, enabling faster processing.

2. **Validate that a product can get into the system:**

   The **LoopOS Validation App** assists in determining whether a product meets the criteria for acceptance into the system. It guides workers through item identification, condition and price evaluation, making it user-friendly even for those with minimal training.

3. **Deliver a product to the system:**

   Once a product is accepted, it can be sent through a LoopOS transport service or taken to one of the LoopOS Hubs. The **LoopOS Hubs App** serves as a mediator for any physical interaction between consumers and the LoopOS network, offering options such as item pickup points, LoopOS Partners Drop Shipment, and LoopOS Delivery.

4. **Certify, Refurbish, and Store:**

   The **LoopOS Handling App** manages and guides operations related to reverse logistics and circular economy processes. It facilitates step-by-step product certification, refurbishment, tracking of costs, upload of professional photos, and integration with Warehouse Management Systems (WMS) for storage. The app allows non-specialized workers to efficiently carry out these processes across various product categories.

5. **Get the product out in the market again:**

   The **LoopOS Exits** offers multiple options to reintroduce products into the market. Either through headless e-commerce API with platforms like Shopify, through its own store and renting services (**LoopOS Store** & **LoopOS Renting**), or through the Partners own store, ensuring the product has another chance to be sold or utilized.

To integrate all the apps, LoopOS relies on three key components:

1. **LoopOS Core App:** Serving as the central control panel for the circular business, enabling the configuration and parameterization of the product journey within the system. It enables the user to make crucial decisions regarding the categories and types of products they want to circulate, the submission process for consumers, the validation criteria, pricing and depreciation rules, collection points and transport services, physical checks by hub workers, handling, etc. To quickly set up the circular business, the Loop co. devised **LoopOS Flows**, a no-code logistics solution designed specifically for reverse logistics.

Users can effortlessly create personalized flows by dragging and dropping various components like apps and services. Figure 2.4 shows an example of the flow.

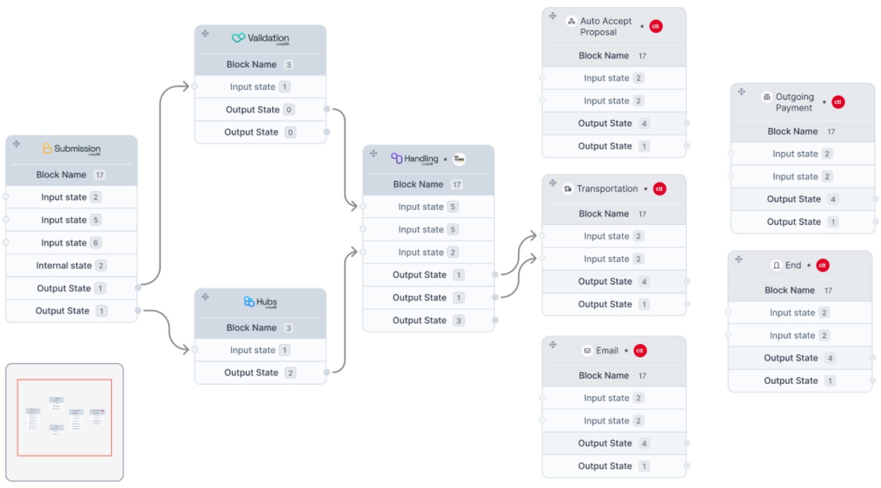


Figure 2.4: LoopOS Flow example [16].

2. **LoopOS Network Manager:** The focus of this thesis and central orchestrator of the LoopOS ecosystem, it is the main app that addresses the challenges of an ecosystem with such a distributed architecture.

3. **LoopOS Services:** Simplifies service management by including Service Providers for services such as Transports, Email, SMS, Payments In, and Payments Out, where each provider is associated with a specific service provision, cost method and configuration to be able to access these services API. These services can be chosen by the Partner and applied in the chosen Applications by the Network Manager, in order to further customize each Partners LoopOS system.

Finally, the presented use cases offer a glimpse into the real-world applications of LoopOS to showcase its functionality and impact on different stakeholders:

Customer Putting a Product into the System:

Scenario: A customer, Alice, wants to sell her used smartphone through LoopOS. She uses the LoopOS Submission App to input details about the product, such as model, condition, and any additional information. The app streamlines the process, making it easy for Alice to provide the necessary information.

Steps:

1. Alice opens the LoopOS Submission App and enters details about her smartphone.

2. The app guides her through the submission process, ensuring all required information is provided.

3. Once submitted, the product information is sent to the LoopOS Core App for processing.

Outcome: The product details are now in the LoopOS ecosystem, awaiting validation.

| Worker Using LoopOS Apps to Handle Items:

Scenario: John, a worker at a LoopOS Hub, receives a set of products that have passed validation and need further processing. He needs to certify, refurbish, and store these items efficiently.

Steps:

1. John uses the LoopOS Validation App to confirm the acceptance of the products into the system.
2. The LoopOS Handling App guides him through the certification and refurbishment processes, ensuring each step is completed.
3. John uses the app to upload professional photos of the refurbished products and updates their status accordingly.

Outcome: The certified and refurbished products are now ready for reintroduction, with detailed records of their condition and certification process.

| LoopOS Workers Deploying Applications:

Scenario: LoopOS Workers are responsible for deploying applications to enhance the system's functionality based on partner requirements.

Steps:

1. LoopOS Workers access the Network Manager and deploy the applications the Partner desires manually.
2. LoopOS Workers access the Network Manager and configures these applications according to the the partner's business requirements manually.

Outcome: The LoopOS system is enriched with new applications tailored to the specific requirements of Company XYZ.

| Partners Managing Employees and Permissions Through Network Manager:

Scenario: A Partner, Company XYZ, uses the LoopOS Network Manager to manage employees, assign permissions, and create user groups for streamlined access control within the customized LoopOS system.

Steps:

1. Partners log into the LoopOS Network Manager with their administrative credentials.

2. They navigate to the "User Management" section to create user groups based on the organizational structure or specific roles within Company XYZ.

3. Partners assign appropriate permissions to each user or user group, defining access levels to various features and modules of the LoopOS system.

4. Partners set up and manage employee profiles, including adding new users and modifying existing ones as the organizational structure evolves.

Outcome: Company XYZ effectively manages its employees through the LoopOS Network Manager. Partners have established user groups, defined permissions, and maintained control over access to LoopOS services and applications. This results in a well-organized and secure system where employees have the necessary access to perform their roles efficiently.

These use cases highlight the flexibility and scalability of the LoopOS ecosystem, catering to both end-users and partners with specific business requirements.

In conclusion, LoopOS is a multifaceted ecosystem of interconnected applications designed to facilitate and streamline the circular economy processes by working together to seamlessly meet the diverse needs of partner companies, ranging from large corporations to local businesses. Nevertheless, conducting a more comprehensive analysis of the Network Manager is essential, which we will delve into in the next chapter.

2.4 Network Manager

Before delving into the State of the Art, it is essential to fully comprehend the existing Network Manager implementation, application deployment and management processes, and authentication and authorization mechanisms. Automation and improvement efforts are futile without understanding the underlying systems.

As mentioned earlier, due to its urgent nature, the Network Manager was initially developed with a focus on the back-end. The first iteration of this product was exclusively intended for developers to manually create environments, resulting in suboptimal usability and user experience. For example, the only permission levels applied here are in terms of access to this application, where acess to individual pages is not restricted.

More specifically, this initial version of the Network Manager was created leveraging AVO [4], a Content Management System for Ruby on Rails. It facilitates the construction of a practical front-end, freeing up the backend's development time. Figure 2.5 showcases the welcome page of the Network Manager.

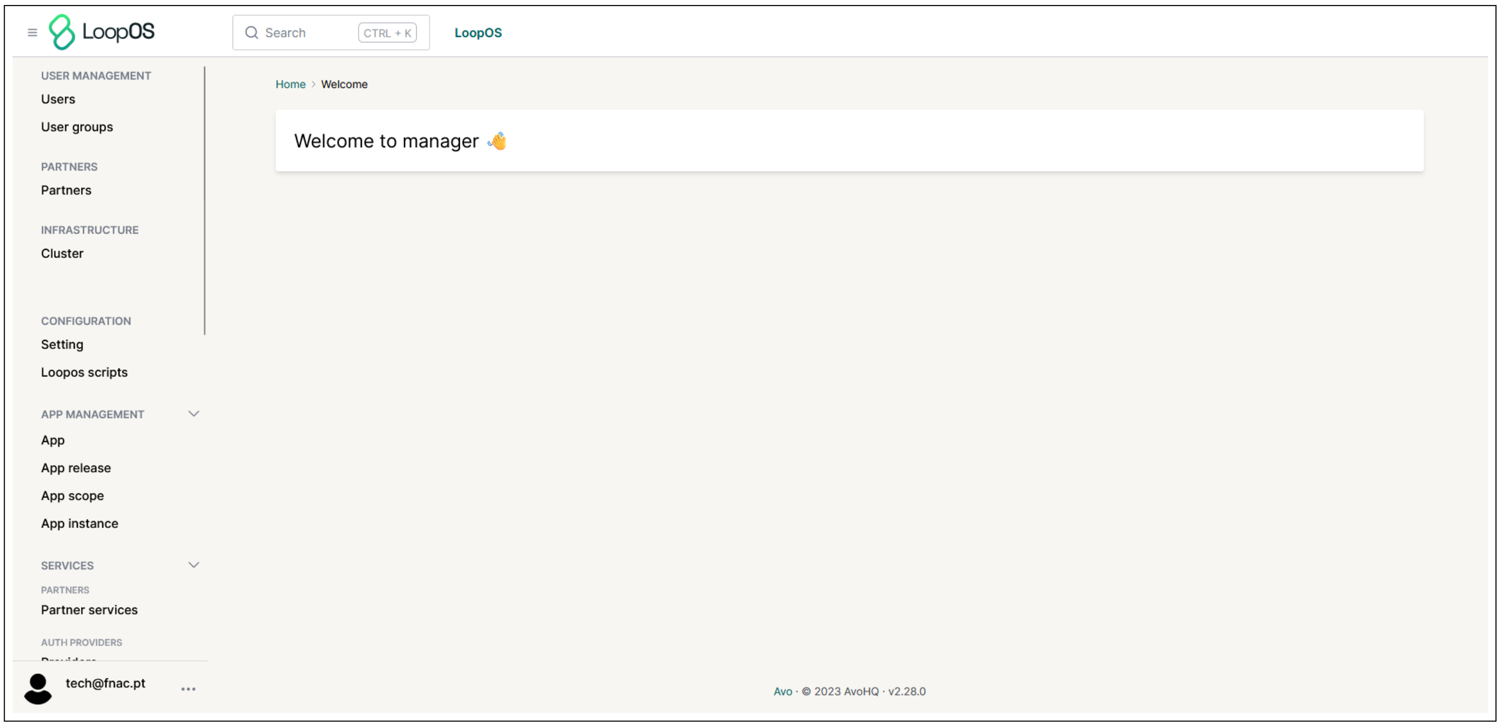
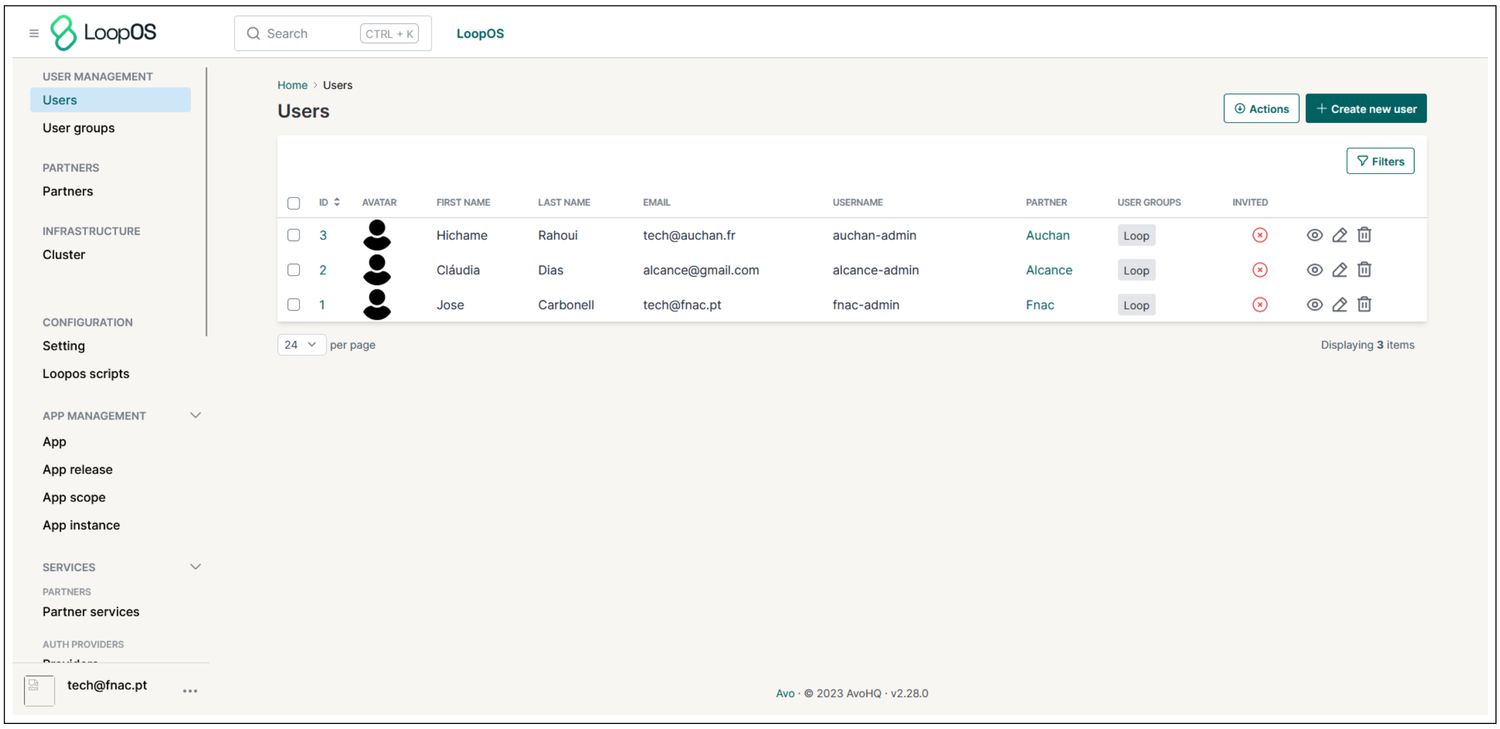


Figure 2.5: Network Manager: Welcome Page.

Along with thoroughly examining the Network Manager's inner workings, this section will also present visual representations of these components to highlight the need for a better user experience. It should be noted the data presented in these depictions is purely fictional since they have been collected from development environments.

User Management: *User, User Group*

Users represent individuals who interact with the LoopOS ecosystem. Each user is associated with a Partner, reflecting their affiliated company. The Network Manager allows the edit of user basic profile information, including email, name, and avatar. Figure 2.6 showcases the user index page presenting user details and its affiliation.



Users

ID	AVATAR	FIRST NAME	LAST NAME	EMAIL	USERNAME	PARTNER	USER GROUPS	INVITED
3	●	Hichame	Rahoui	tech@auchan.fr	auchan-admin	Auchan	Loop	✗
2	●	Cláudia	Dias	alcance@gmail.com	alcance-admin	Alcance	Loop	✗
1	●	Jose	Carbonell	tech@fnac.pt	fnac-admin	Fnac	Loop	✗

24 per page

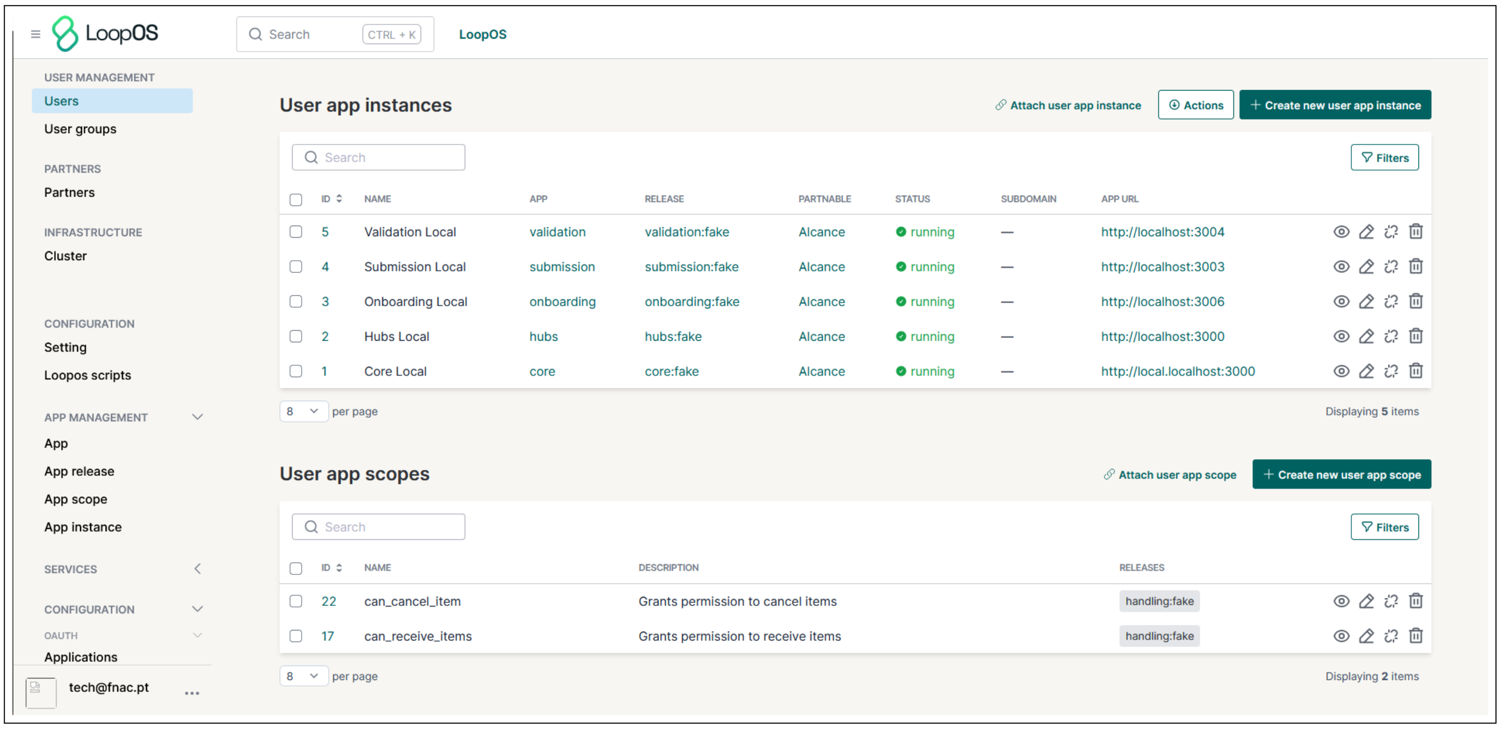
Displaying 3 items

Avo · © 2023 AvoHQ · v2.28.0

Figure 2.6: Network Manager: User Index Page.

Users can be linked to App Instances, representing specific instances of applications that belong to a Partner. This allows for granular control over user access to individual application instances. For example, a User connected only to a Core instance belonging to a “CTT” partner can not access a Handling app instance belonging to the same partner. Additionally, the user can not access another instance belonging to a different Partner.

Moreover, users can be linked to App Scopes, which define their permissions within specific applications. These permissions determine the level of access and actions a user can perform within an application. For example, using the same example as before, if the user has only the App Scope *can_view_items*, this is the only action he can perform inside the assigned Core instance. Figure 2.7 showcases an example of a user associated with multiple App Instances and multiple App Scopes.



User app instances

ID	NAME	APP	RELEASE	PARTNABLE	STATUS	SUBDOMAIN	APP URL
5	Validation Local	validation	validation:fake	Alcance	running	—	http://localhost:3004
4	Submission Local	submission	submission:fake	Alcance	running	—	http://localhost:3003
3	Onboarding Local	onboarding	onboarding:fake	Alcance	running	—	http://localhost:3006
2	Hubs Local	hubs	hubs:fake	Alcance	running	—	http://localhost:3000
1	Core Local	core	core:fake	Alcance	running	—	http://local.localhost:3000

8 per page Displaying **5** items

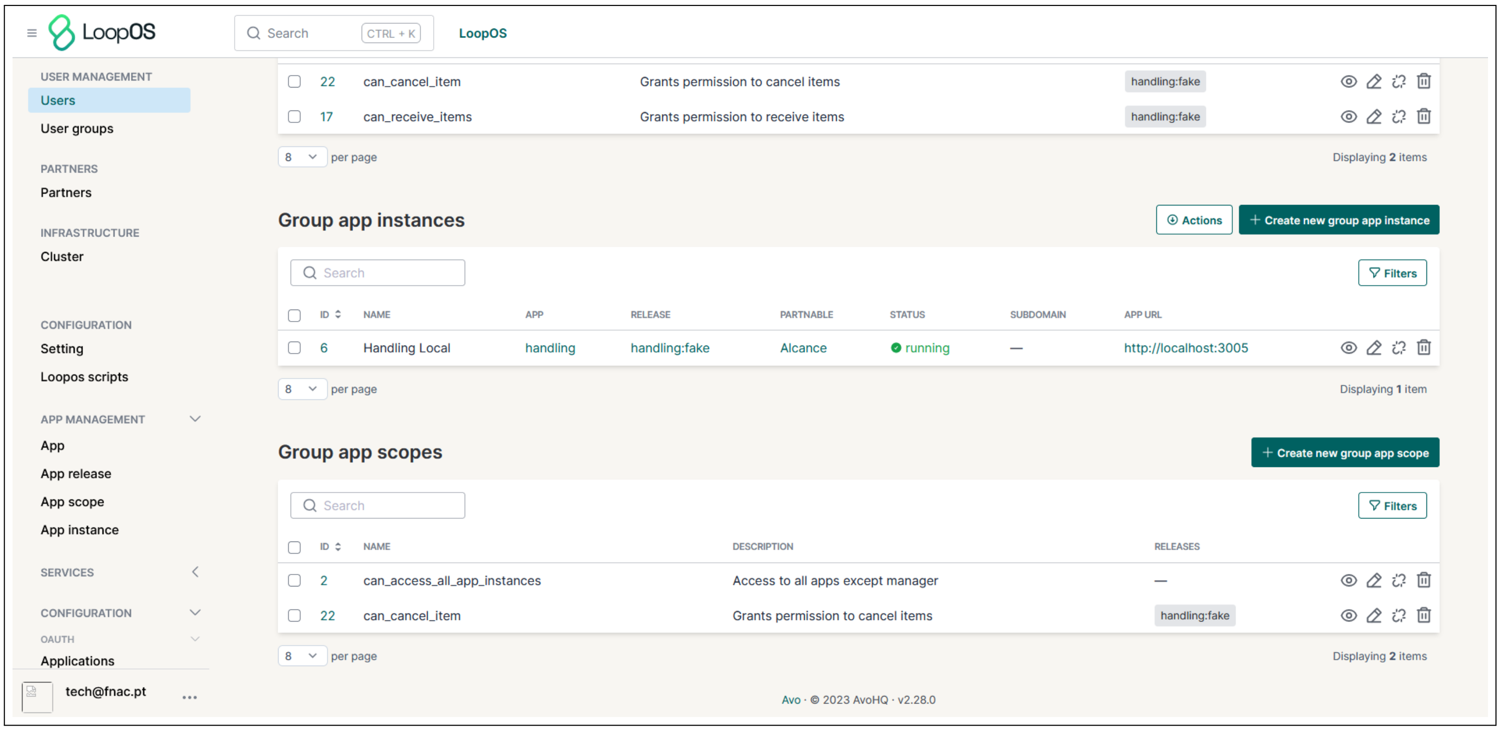
User app scopes

ID	NAME	DESCRIPTION	RELEASES
22	can_cancel_item	Grants permission to cancel items	handling:fake
17	can_receive_items	Grants permission to receive items	handling:fake

8 per page Displaying **2** items

Figure 2.7: Network Manager: User Show Page.

On the other hand, **User Groups** provides an efficient mechanism for assigning App Scopes and App Instances to multiple users simultaneously. This eliminates the need to manage permissions and access for individual users one by one, streamlining the user administration process. Figure 2.8 showcases App Instances and App Scopes belonging to a User that were inherited from the affiliated User Group.



22	can_cancel_item	Grants permission to cancel items	handling:fake
17	can_receive_items	Grants permission to receive items	handling:fake

8 per page — Displaying 2 items

Group app instances

ID	NAME	APP	RELEASE	PARTNABLE	STATUS	SUBDOMAIN	APP URL
6	Handling Local	handling	handling:fake	Alcance	running	—	http://localhost:3005

8 per page — Displaying 1 item

Group app scopes

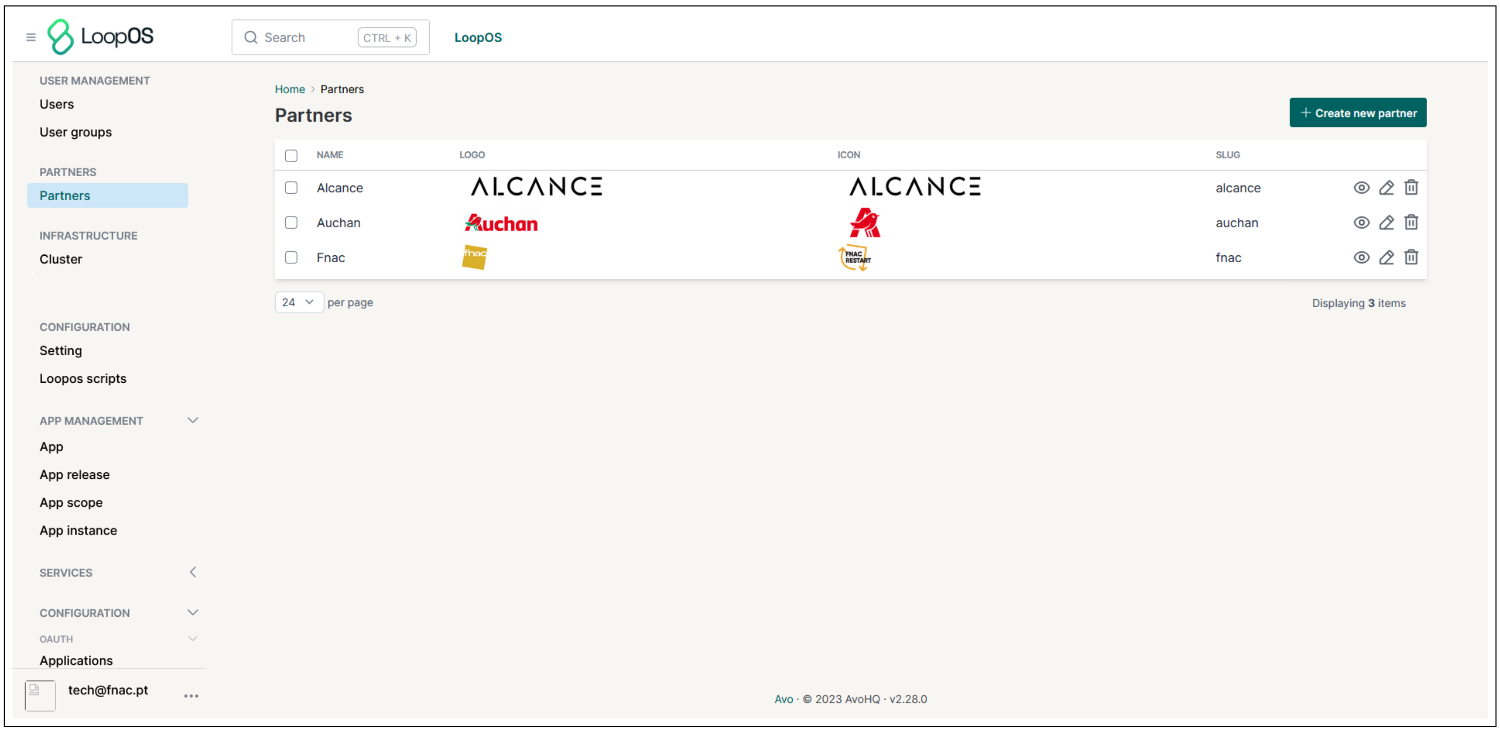
ID	NAME	DESCRIPTION	RELEASES
2	can_access_all_app_instances	Access to all apps except manager	—
22	can_cancel_item	Grants permission to cancel items	handling:fake

8 per page — Displaying 2 items

Figure 2.8: Network Manager: User Show Page - Inherited elements from affiliated User Group.

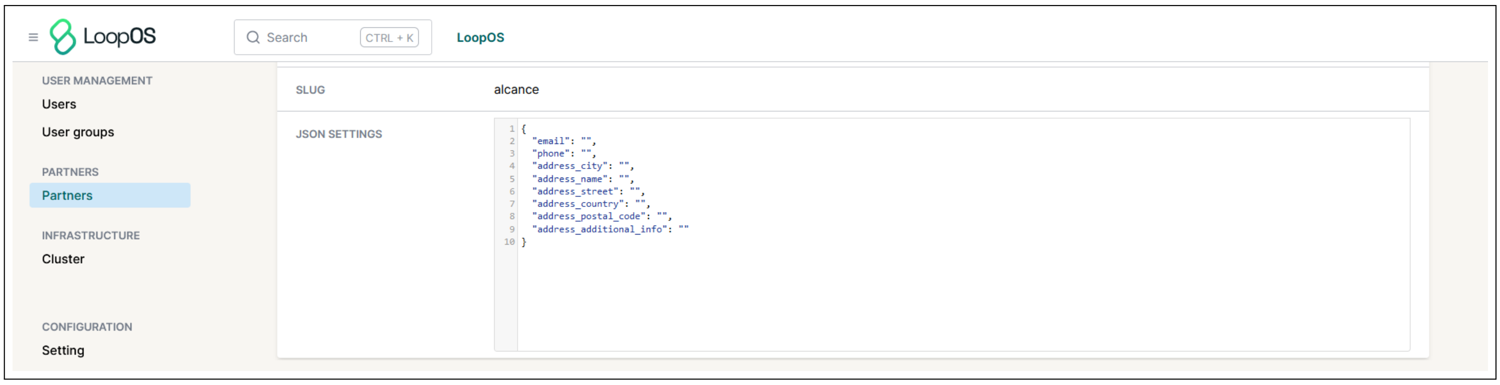
Partner Management: *Partner*

Partners are any individual organization that interact with LoopOS. They are the highest business logic aggregator in LoopOS and can be linked to Users, App Instances, Partner Services, and Email Message Templates. Figure 2.9 showcases the partner index page presenting partner details, and Figure 2.10 illustrates how settings can be created and edited. The Network Manager also enables adding providers from multiple services while creating a way to log all access to these services. Figure 2.11 illustrates an example of Services and how these services are presented in the Partner Show Page.



NAME	LOGO	ICON	SLUG
Alcance	ALCANCE	ALCANCE	alcance
Auchan	Auchan		auchan
Fnac	fnac	FNAC RESTART	fnac

Figure 2.9: Network Manager: Partner Index Page.

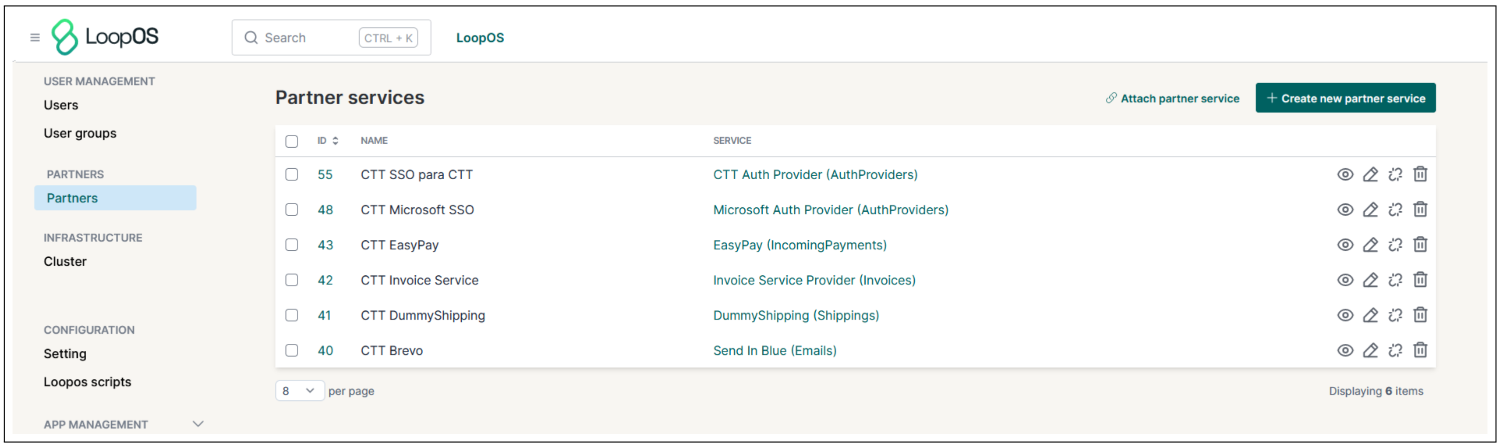


SLUG alcance

JSON SETTINGS

```
{
  "email": "",
  "phone": "",
  "address_city": "",
  "address_name": "",
  "address_street": "",
  "address_country": "",
  "address_postal_code": "",
  "address_additional_info": ""
}
```

Figure 2.10: Network Manager: Partner Show Page - Example implementation of settings.



Partner services

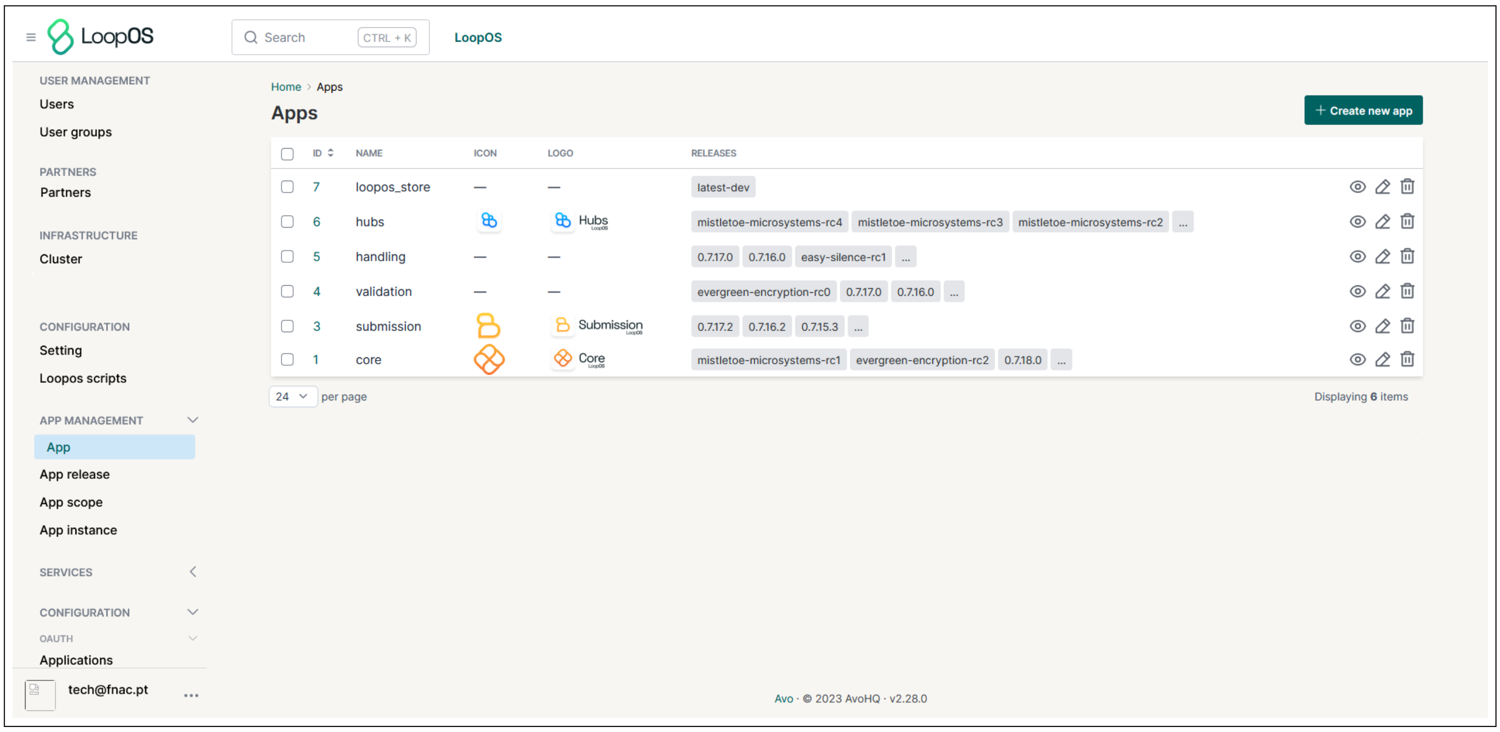
ID	NAME	SERVICE
55	CTT SSO para CTT	CTT Auth Provider (AuthProviders)
48	CTT Microsoft SSO	Microsoft Auth Provider (AuthProviders)
43	CTT EasyPay	EasyPay (IncomingPayments)
42	CTT Invoice Service	Invoice Service Provider (Invoices)
41	CTT DummyShipping	DummyShipping (Shippings)
40	CTT Brevo	Send In Blue (Emails)

8 per page Displaying 6 items

Figure 2.11: Network Manager: Partner Show Page - Example implementation of associated Services to a Partner.

App Management: *App, App Release, App Instance, App Scope*

Apps are the central control panel for configuring and parameterizing the entire product journey, defining which apps are available to the LoopOS system showcased in figure 2.12. Each App can be linked to many App Releases. Figure 2.13 illustrates a LoopOS Exists App and its associated releases and the header style values to configure its header.



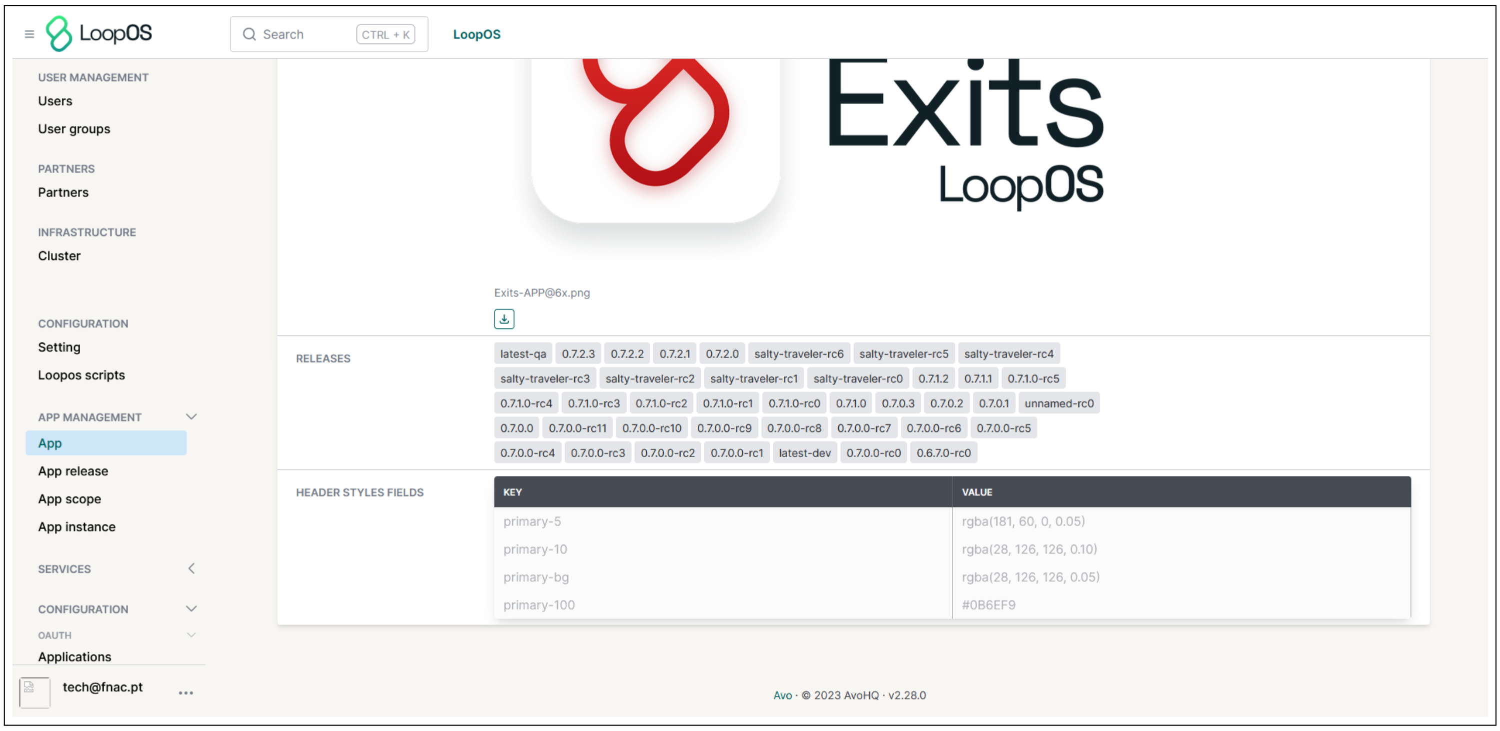
Apps

ID	NAME	ICON	LOGO	RELEASES
7	loopos_store	—	—	latest-dev
6	hubs		Hubs	mistletoe-microsystems-rc4, mistletoe-microsystems-rc3, mistletoe-microsystems-rc2, ...
5	handling	—	—	0.7.17.0, 0.7.16.0, easy-silence-rc1, ...
4	validation	—	—	evergreen-encryption-rc0, 0.7.17.0, 0.7.16.0, ...
3	submission		Submission	0.7.17.2, 0.7.16.2, 0.7.15.3, ...
1	core		Core	mistletoe-microsystems-rc1, evergreen-encryption-rc2, 0.7.18.0, ...

24 per page

Displaying **6** items

Figure 2.12: Network Manager: App Index Page.



Exits-APP@6x.png

RELEASES

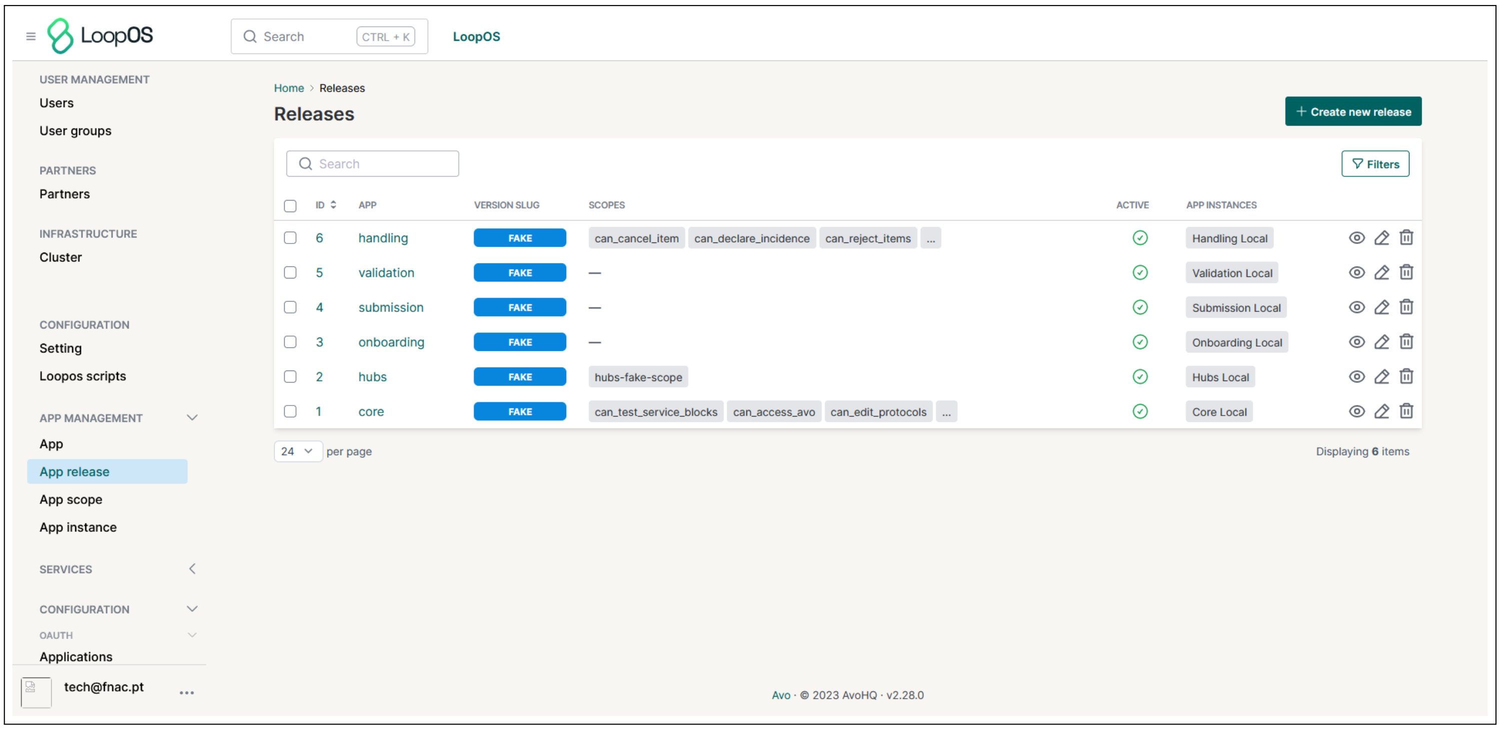
latest-qa 0.7.2.3 0.7.2.2 0.7.2.1 0.7.2.0 salty-traveler-rc6 salty-traveler-rc5 salty-traveler-rc4
salty-traveler-rc3 salty-traveler-rc2 salty-traveler-rc1 salty-traveler-rc0 0.7.1.2 0.7.1.1 0.7.1.0-rc5
0.7.1.0-rc4 0.7.1.0-rc3 0.7.1.0-rc2 0.7.1.0-rc1 0.7.1.0-rc0 0.7.1.0 0.7.0.3 0.7.0.2 0.7.0.1 unnamed-rc0
0.7.0.0 0.7.0.0-rc11 0.7.0.0-rc10 0.7.0.0-rc9 0.7.0.0-rc8 0.7.0.0-rc7 0.7.0.0-rc6 0.7.0.0-rc5
0.7.0.0-rc4 0.7.0.0-rc3 0.7.0.0-rc2 0.7.0.0-rc1 latest-dev 0.7.0.0-rc0 0.6.7.0-rc0

HEADER STYLES FIELDS

KEY	VALUE
primary-5	rgba(181, 60, 0, 0.05)
primary-10	rgba(28, 126, 126, 0.10)
primary-bg	rgba(28, 126, 126, 0.05)
primary-100	#0B6EF9

Figure 2.13: Network Manager: App Show Page - Associated Releases and Header styles settings.

App Releases is a specific app version from GitLab [73] linked to a docker image, belonging to an App and having one or many App Instances. Figure 2.14 showcases the App Release Index page. It has public, private, and infrastructure setting schema in order to define which settings the associated instances must have, as seen in Figure 2.15. In addition, they have many **App Scopes** that represent user scopes supported by each app. Figure 2.16 illustrates the associated App Scopes and App Instances. A scope typically represents the ability to perform a specific or group of specific actions in the app.



Releases

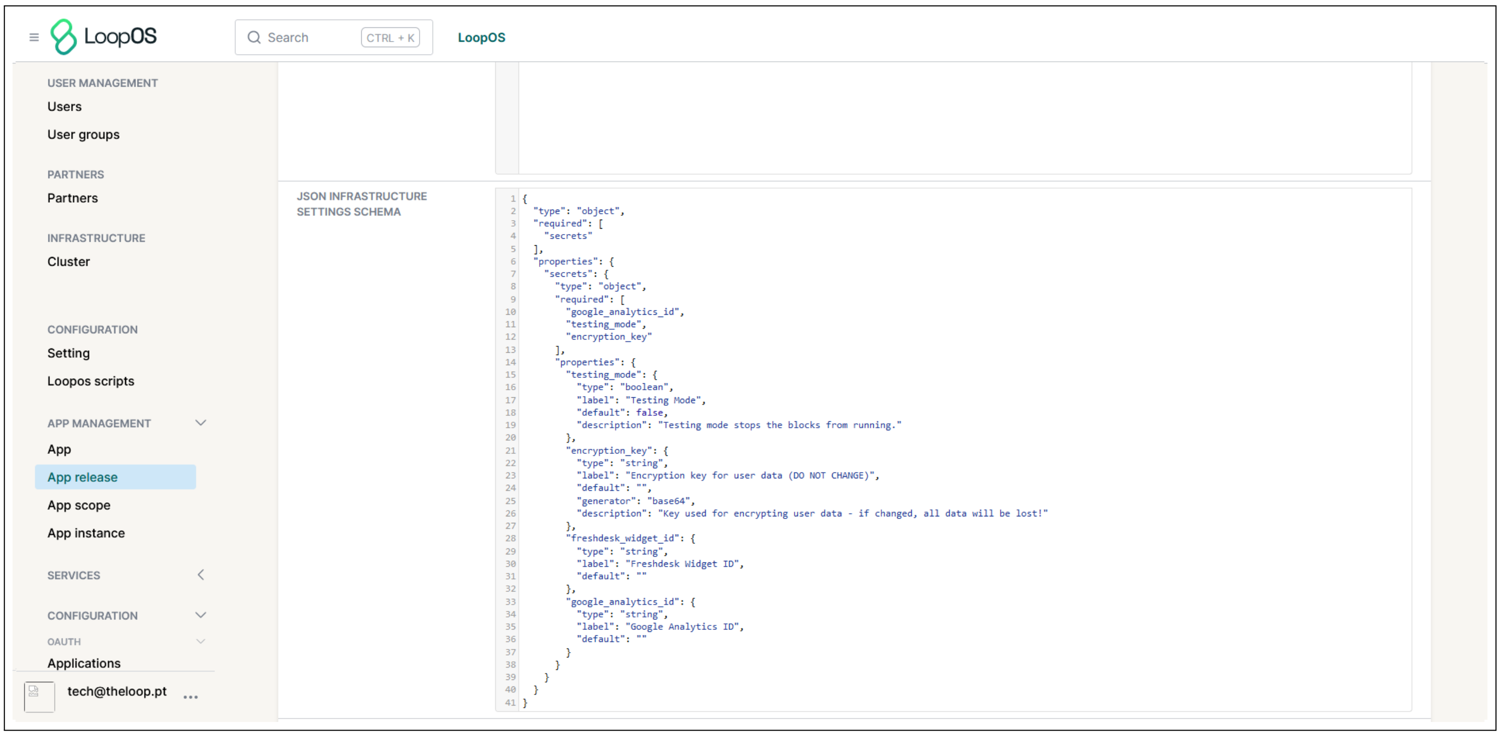
	ID	APP	VERSION SLUG	SCOPES	ACTIVE	APP INSTANCES
☐	6	handling	FAKE	can_cancel_item can_declare_incidence can_reject_items ...	✓	Handling Local
☐	5	validation	FAKE	—	✓	Validation Local
☐	4	submission	FAKE	—	✓	Submission Local
☐	3	onboarding	FAKE	—	✓	Onboarding Local
☐	2	hubs	FAKE	hubs-fake-scope	✓	Hubs Local
☐	1	core	FAKE	can_test_service_blocks can_access_avo can_edit_protocols ...	✓	Core Local

24 per page

Displaying **6** items

Avo · © 2023 AvoHQ · v2.28.0

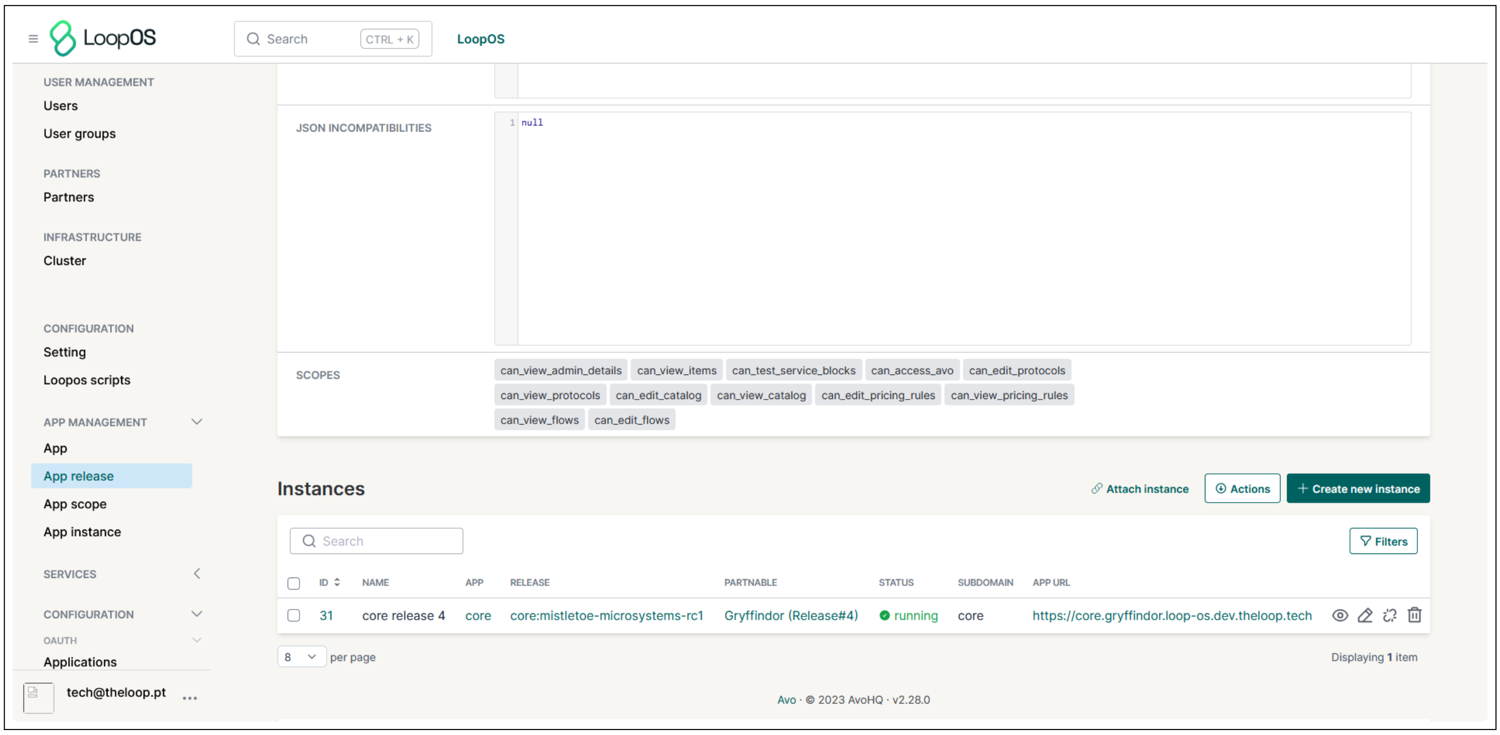
Figure 2.14: Network Manager: App Release Index Page.



JSON INFRASTRUCTURE SETTINGS SCHEMA

```
{
  "type": "object",
  "required": [
    "secrets"
  ],
  "properties": {
    "secrets": {
      "type": "object",
      "required": [
        "google_analytics_id",
        "testing_mode",
        "encryption_key"
      ],
      "properties": {
        "testing_mode": {
          "type": "boolean",
          "label": "Testing Mode",
          "default": false,
          "description": "Testing mode stops the blocks from running."
        },
        "encryption_key": {
          "type": "string",
          "label": "Encryption key for user data (DO NOT CHANGE)",
          "default": "",
          "generator": "base64",
          "description": "Key used for encrypting user data - if changed, all data will be lost!"
        },
        "freshdesk_widget_id": {
          "type": "string",
          "label": "Freshdesk Widget ID",
          "default": ""
        },
        "google_analytics_id": {
          "type": "string",
          "label": "Google Analytics ID",
          "default": ""
        }
      }
    }
  }
}
```

Figure 2.15: Network Manager: App Release Show Page - Example implementation of infrastructure settings schema.



JSON INCOMPATIBILITIES

```
null
```

SCOPES: can_view_admin_details, can_view_items, can_test_service_blocks, can_access_avo, can_edit_protocols, can_view_protocols, can_edit_catalog, can_view_catalog, can_edit_pricing_rules, can_view_pricing_rules, can_view_flows, can_edit_flows

Instances

ID	NAME	APP	RELEASE	PARTNABLE	STATUS	SUBDOMAIN	APP URL
31	core release 4	core	core:mistletoe-microsystems-rc1	Gryffindor (Release#4)	● running	core	https://core.gryffindor.loop-os.dev.theloop.tech

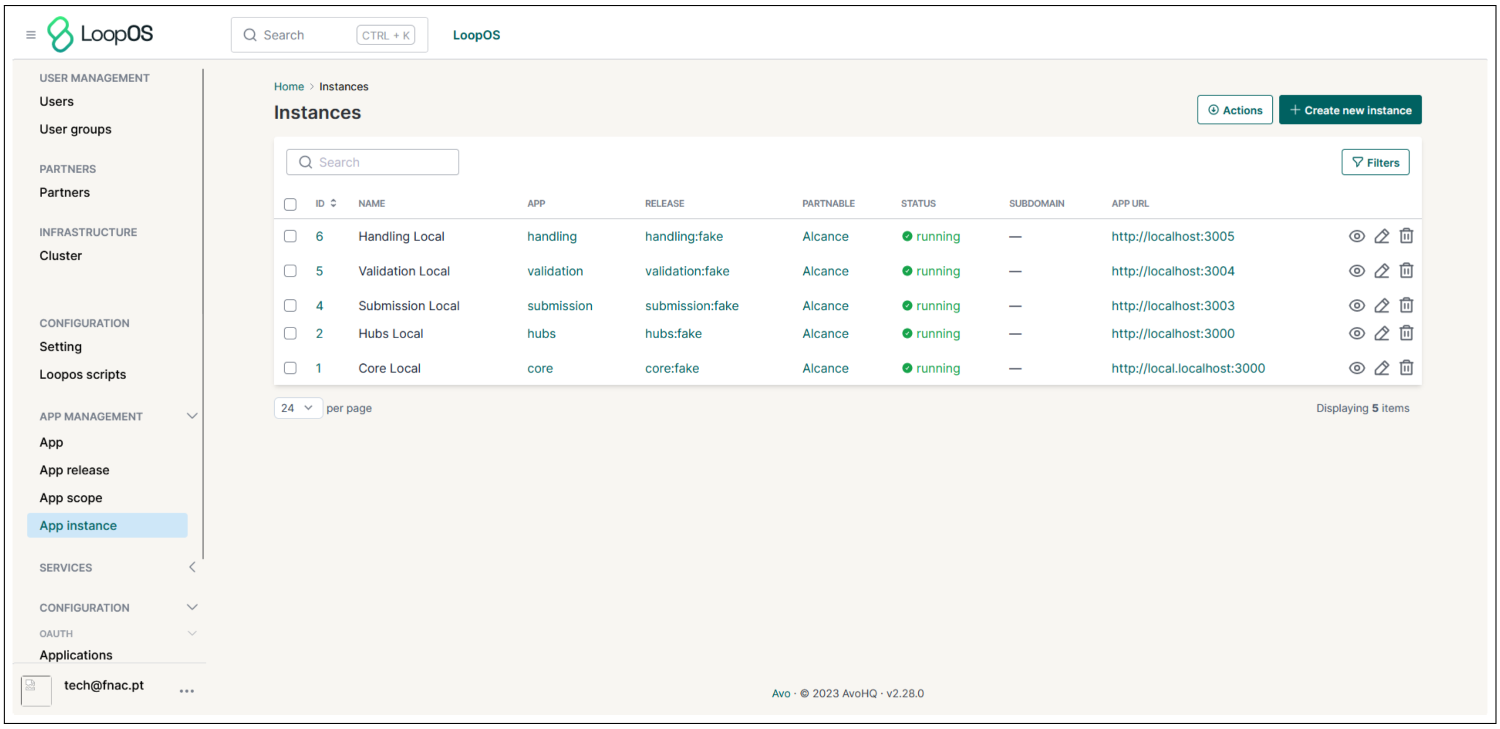
8 per page — Displaying 1 item

Figure 2.16: Network Manager: App Release Show Page - Associated App Scopes and App Instances.

Finally, **App Instances** represents a specific deployment of an App to be used by a specific partner. App instances have infrastructure settings (DevOps settings such as secrets), public settings accessible by all parties (for now, accessed by other App Instances by requesting this data through an endpoint open in the Network Manager), and private settings that can only be changed by LoopOS personnel (currently, accessed only by the App Instance itself, requesting this data through an endpoint open in the Network Manager). These settings must obey the corresponding setting schema from the associated App Release.

In addition, App Instances are associated with an **Oauth Application** to manage their security. These contain the essential data required for an App Instance to connect to the Network Manager.

Most notably, App Instances have two key features: they can be deployed and kept in sync with their corresponding Gitlab image with the *"SyncInfrastructureJob"* and can have code executed on them, commanded from the Network Manager. This will be explained in Sub-section 2.4.1 - Application Deployment & Management.



Instances

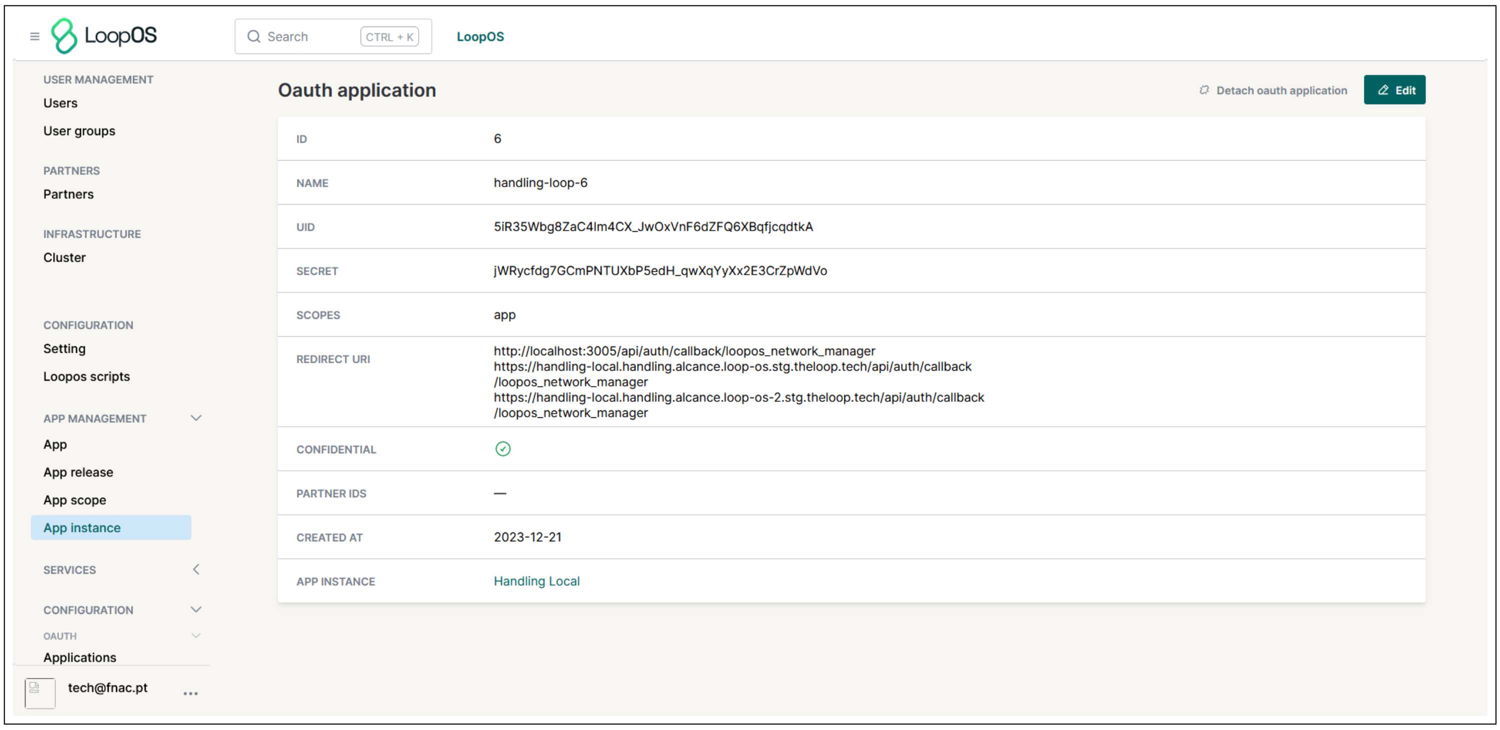
ID	NAME	APP	RELEASE	PARTNABLE	STATUS	SUBDOMAIN	APP URL
6	Handling Local	handling	handling:fake	Alcance	running	—	http://localhost:3005
5	Validation Local	validation	validation:fake	Alcance	running	—	http://localhost:3004
4	Submission Local	submission	submission:fake	Alcance	running	—	http://localhost:3003
2	Hubs Local	hubs	hubs:fake	Alcance	running	—	http://localhost:3000
1	Core Local	core	core:fake	Alcance	running	—	http://local.localhost:3000

24 per page

Displaying 5 items

Avo · © 2023 AvoHQ · v2.28.0

Figure 2.17: Network Manager: App Instances Index Page.



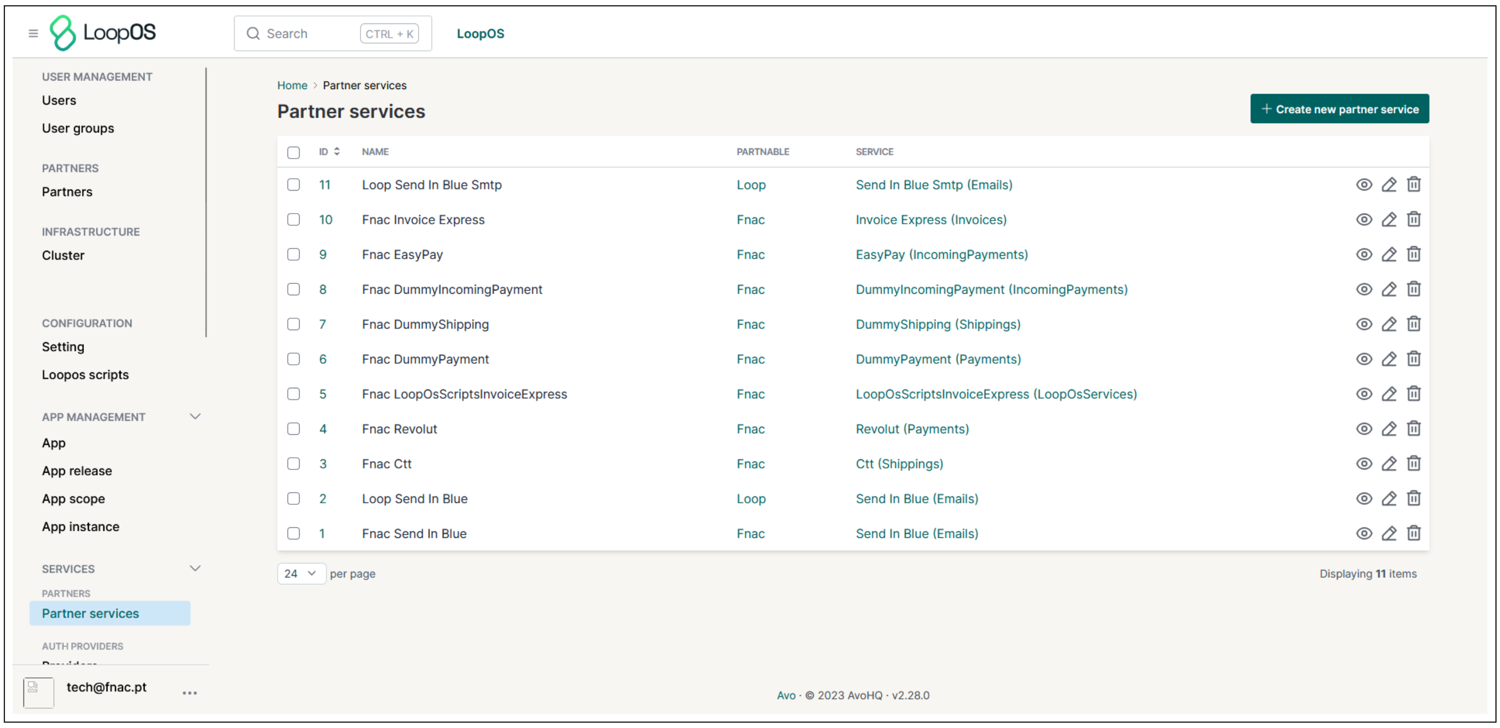
Oauth application

ID	6
NAME	handling-loop-6
UID	5iR35Wbg8ZaC4Im4CX_JwOxVnF6dZFQ6XBqfjcqdtkA
SECRET	jWRycfdg7GCmPNTUXbP5edH_qwXqYyXx2E3CrZpWdVo
SCOPES	app
REDIRECT URI	http://localhost:3005/api/auth/callback/loopos_network_manager https://handling-local.handling.alcance.loop-os.stg.theloop.tech/api/auth/callback /loopos_network_manager https://handling-local.handling.alcance.loop-os-2.stg.theloop.tech/api/auth/callback /loopos_network_manager
CONFIDENTIAL	✓
PARTNER IDS	—
CREATED AT	2023-12-21
APP INSTANCE	Handling Local

Figure 2.18: Network Manager: App Instances Show Page - OAuth Application associated.

Service Management: *Authentication, Email, Invoice, Shipping, Payments, Incoming Payments*

The Network Manager facilitates the addition of Services, enabling LoopOS to expand its capabilities and address a wider range of business needs. This includes services such as authentication, email, invoice management, shipping, and payment processing. Figure 2.19 illustrates the Service index page.

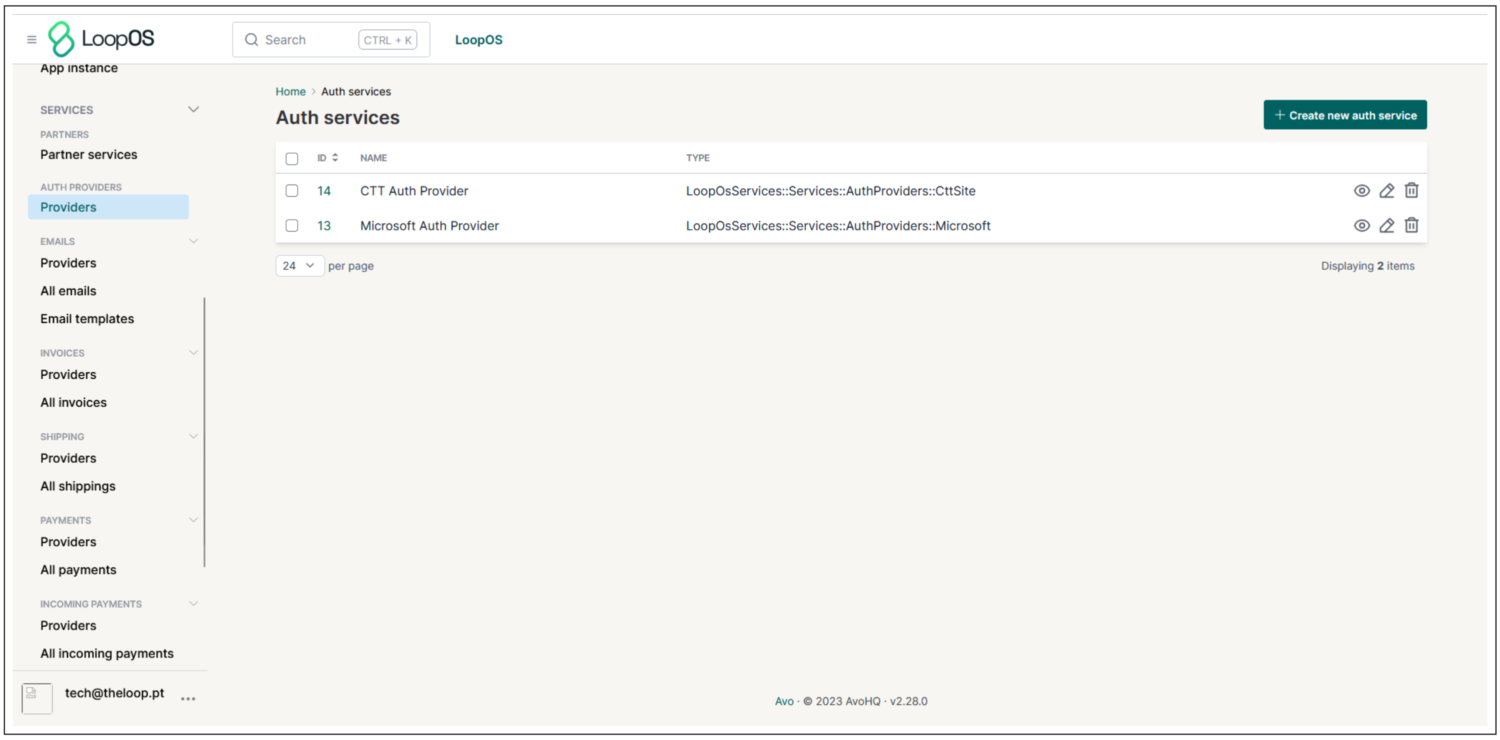


Partner services

ID	NAME	PARTNABLE	SERVICE
11	Loop Send In Blue Smtp	Loop	Send In Blue Smtp (Emails)
10	Fnac Invoice Express	Fnac	Invoice Express (Invoices)
9	Fnac EasyPay	Fnac	EasyPay (IncomingPayments)
8	Fnac DummyIncomingPayment	Fnac	DummyIncomingPayment (IncomingPayments)
7	Fnac DummyShipping	Fnac	DummyShipping (Shippings)
6	Fnac DummyPayment	Fnac	DummyPayment (Payments)
5	Fnac LoopOsScriptsInvoiceExpress	Fnac	LoopOsScriptsInvoiceExpress (LoopOsServices)
4	Fnac Revolut	Fnac	Revolut (Payments)
3	Fnac Ctt	Fnac	Ctt (Shippings)
2	Loop Send In Blue	Loop	Send In Blue (Emails)
1	Fnac Send In Blue	Fnac	Send In Blue (Emails)

Figure 2.19: Network Manager: Service Index Page.

Authentication: Holds external authentication providers available for sign up and sign in, in the LoopOS ecosystem. Figure 2.20 illustrates the current authentication providers, apart from its own, present in the Network Manager.

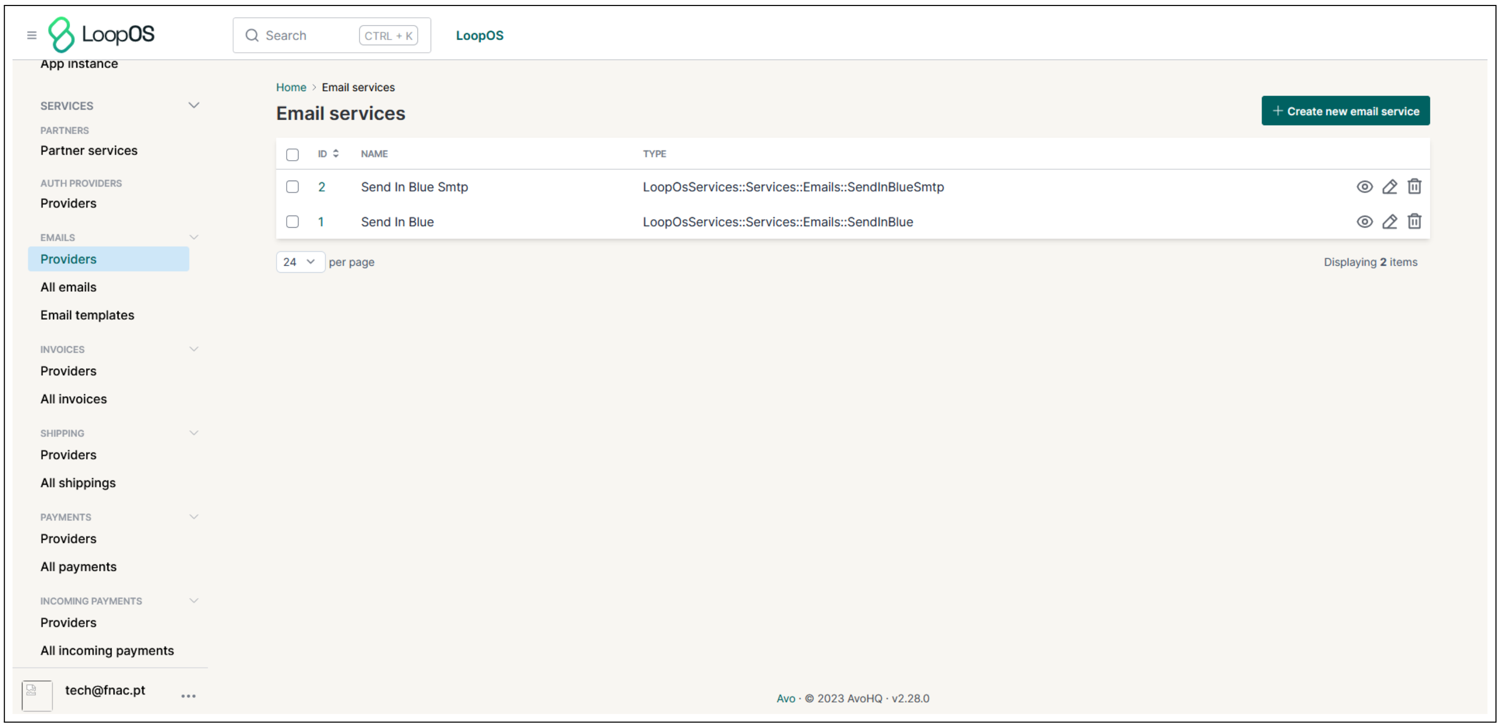


Auth services

ID	NAME	TYPE
14	CTT Auth Provider	LoopOsServices::Services::AuthProviders::CttSite
13	Microsoft Auth Provider	LoopOsServices::Services::AuthProviders::Microsoft

Figure 2.20: Network Manager: Authentication Service Provider Index Page.

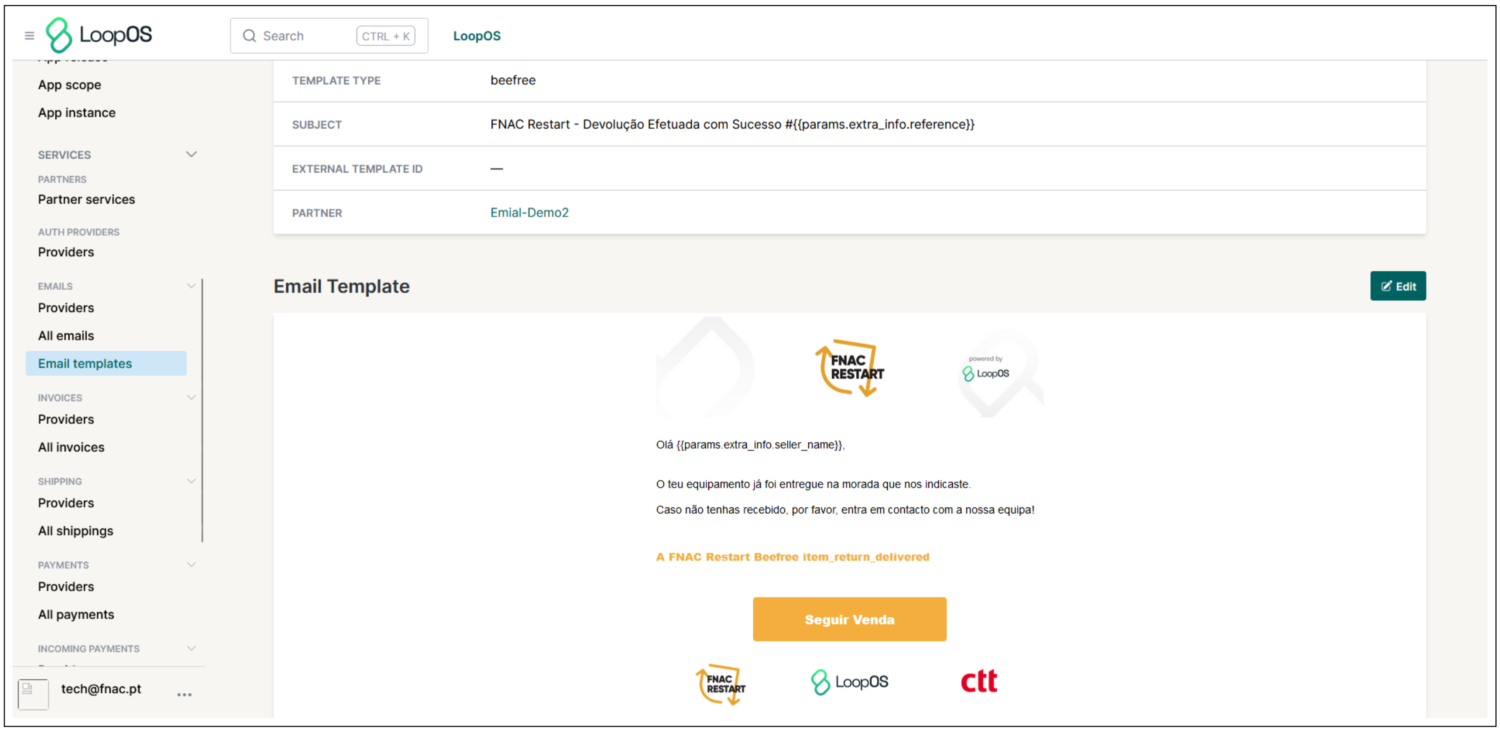
Email: Holds the email providers which will send the email (illustrated in Figure 2.21), the email templates from which the emails will be created from (illustrated in Figure 2.22) and all the emails sent by that provider (illustrated in Figure 2.23). Each email has logs saving the success of actions at specific timestamps. Also, since most of the emails sent will be in regards to items operating on the other apps, these emails can also save a URL to that specific item in the corresponding Core app.



Email services

ID	NAME	TYPE
2	Send In Blue Smtp	LoopOsServices::Services::Emails::SendInBlueSmtp
1	Send In Blue	LoopOsServices::Services::Emails::SendInBlue

Figure 2.21: Network Manager: Email Service Providers Index Page.



TEMPLATE TYPE	beefree
SUBJECT	FNAC Restart - Devolução Efetuada com Sucesso #{{params.extra_info.reference}}
EXTERNAL TEMPLATE ID	—
PARTNER	Emial-Demo2

Email Template

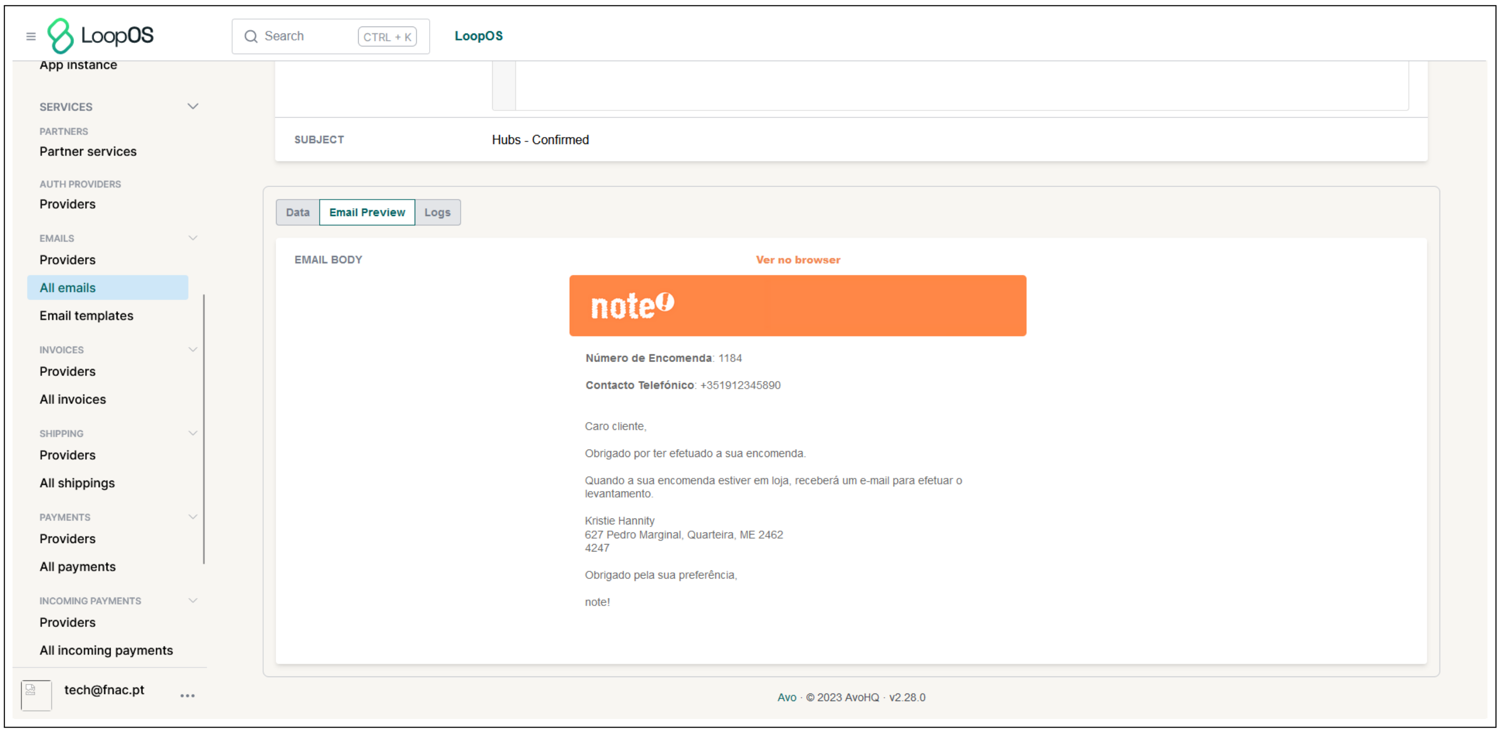
Olá {{params.extra_info.seller_name}},

O teu equipamento já foi entregue na morada que nos indicaste.

Caso não tenhas recebido, por favor, entra em contacto com a nossa equipa!

A FNAC Restart Beefree item_return_delivered

Figure 2.22: Network Manager: Email Template Page - Example of an email template for a successful return with the Fnac Partner.



SUBJECT	Hubs - Confirmed

EMAIL BODY

Número de Encomenda: 1184

Contacto Telefónico: +351912345890

Caro cliente,

Obrigado por ter efetuado a sua encomenda.

Quando a sua encomenda estiver em loja, receberá um e-mail para efetuar o levantamento.

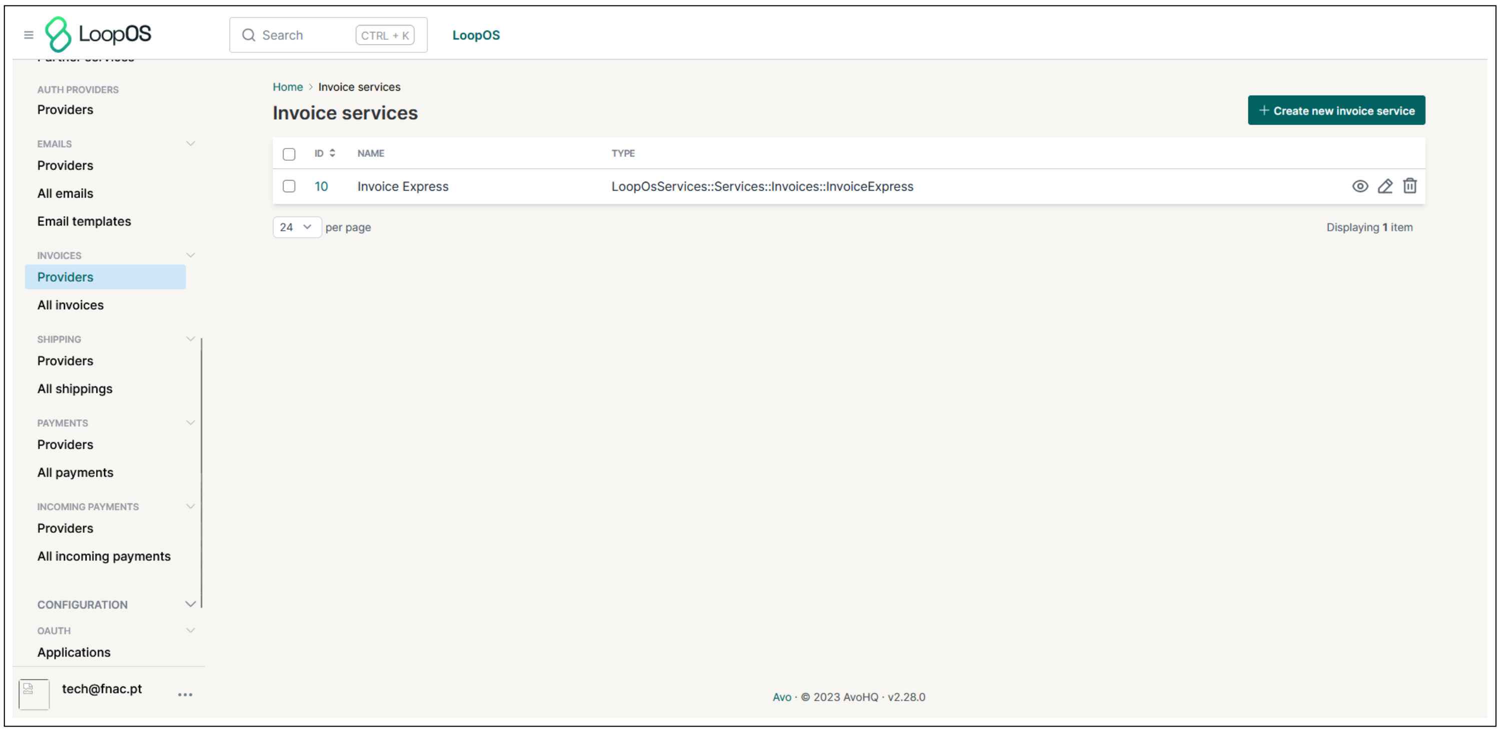
Kristie Hannity
627 Pedro Marginal, Quarteira, ME 2462
4247

Obrigado pela sua preferência,

note!

Figure 2.23: Network Manager: Email Show Page - Example of an email sent by the Network Manager.

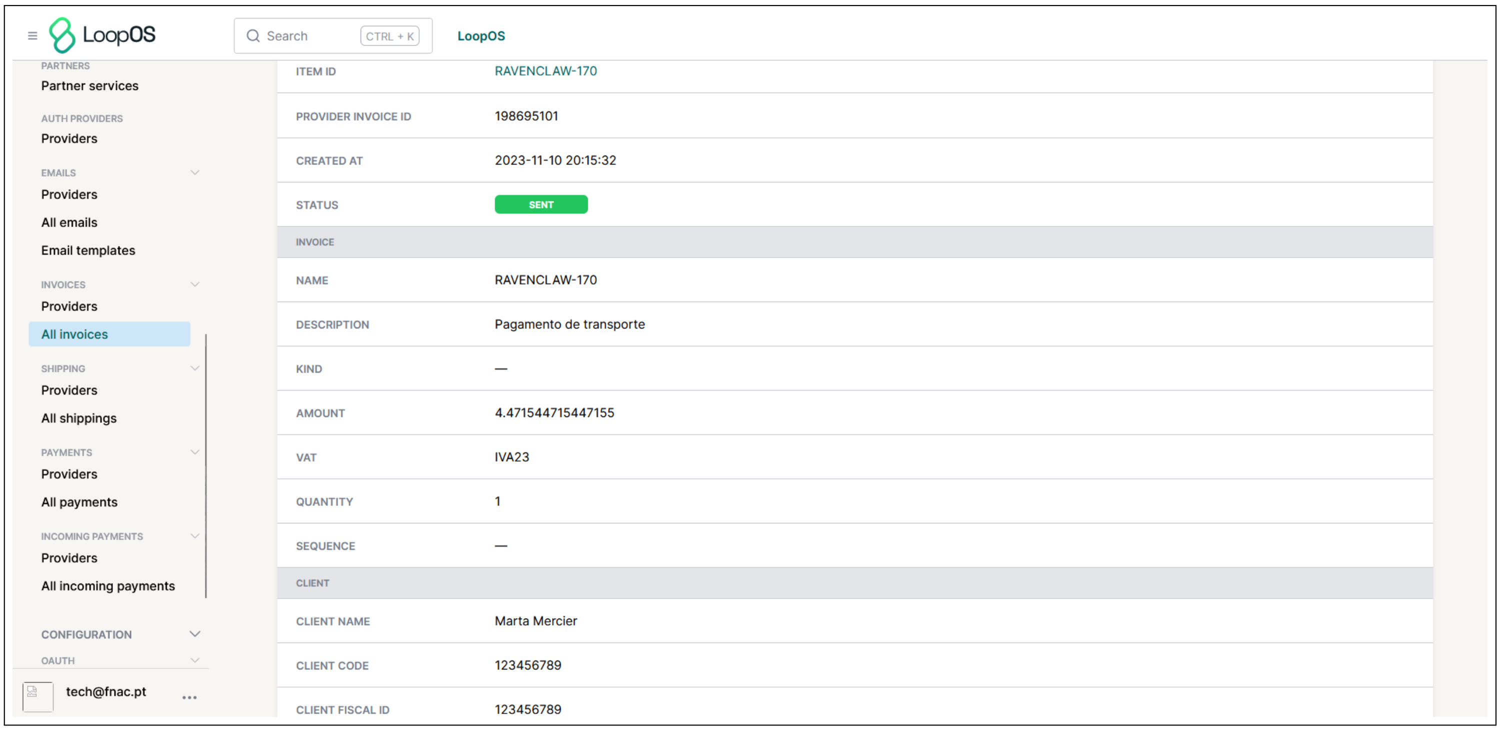
Invoices: Holds the invoice providers (showcased in Figure 2.24) which will create the invoices and all the invoices created by each provider. Each invoice contains essential information such as General details, including the item id and its corresponding URL in the Core App, creation timestamp, and status to track the transaction's progress, also containing more specifics of the invoice, such as its name, description, amount, VAT, and quantity, to provide a more comprehensive overview and client data, such as the client's data for identifying the involved party. In addition, extra data offers additional context such as the application id. Figure 2.25 illustrates how some of these details are presented to the user.



Invoice services

ID	NAME	TYPE
10	Invoice Express	LoopOsServices::Services::Invoices::InvoiceExpress

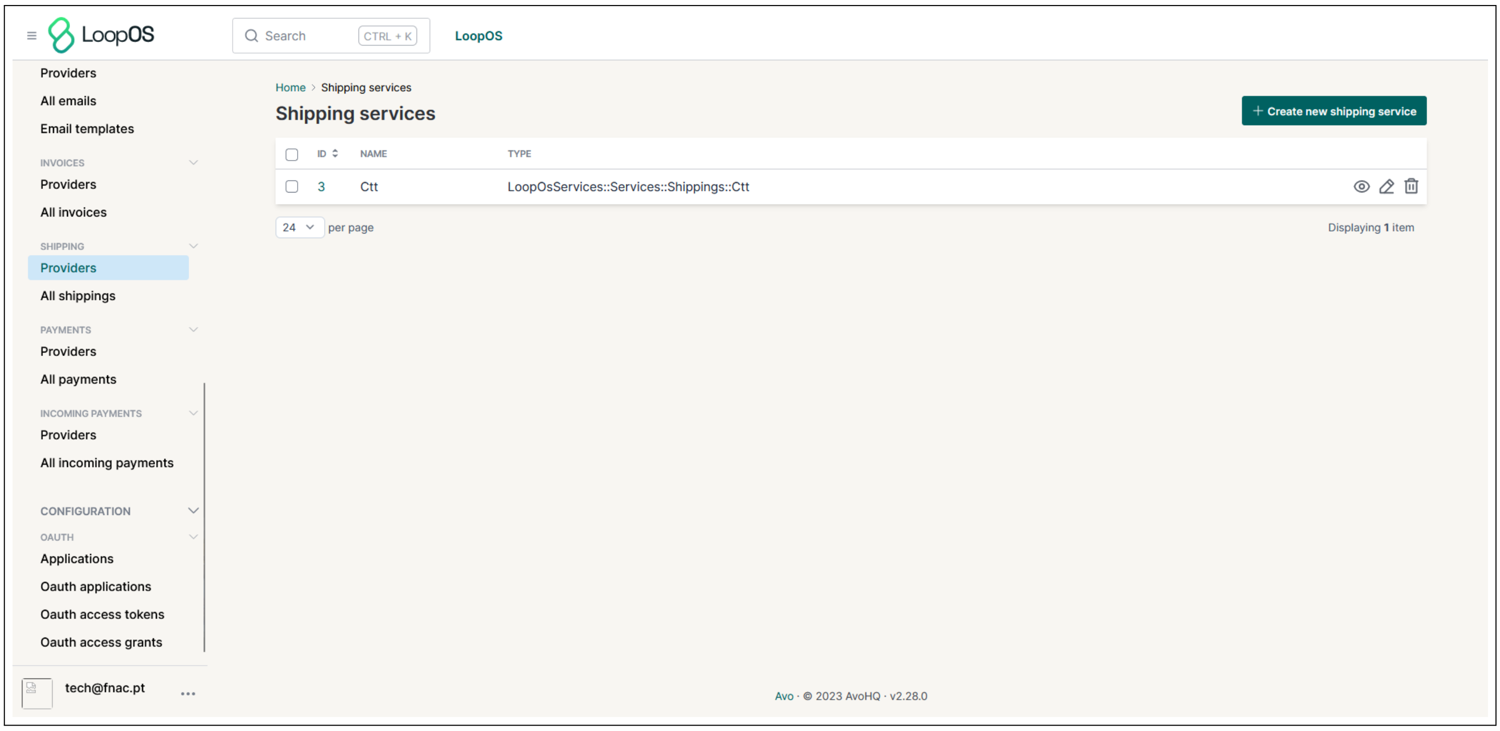
Figure 2.24: Network Manager: Invoice Service Providers Index Page.



ITEM ID	RAVENCLAW-170
PROVIDER INVOICE ID	198695101
CREATED AT	2023-11-10 20:15:32
STATUS	SENT
INVOICE	
NAME	RAVENCLAW-170
DESCRIPTION	Pagamento de transporte
KIND	—
AMOUNT	4.471544715447155
VAT	IVA23
QUANTITY	1
SEQUENCE	—
CLIENT	
CLIENT NAME	Marta Mercier
CLIENT CODE	123456789
CLIENT FISCAL ID	123456789

Figure 2.25: Network Manager: Invoice Show Page - Partial example of the invoice details of an item.

Shipping: Holds the shipping providers (illustrated in Figure 2.26) which will handle and track the shipping of the products and all the shippings being tracked by provider. The general information includes item id and its corresponding URL in the Core App, its reference, creation timestamp, status, shipping type, and request ID. Specific shipping details cover the name, sender and recipient details, as well as, extra data, provides additional context such as the application id. Each shipping is linked to one or many shipping package that saves the package status, tracking code, weight, and shipping reference. Logs document actions upon these objects, such as sending notifications and creating return guides. Additionally, each shipping as a shipping return guide pdf attached with all this information. Figure 2.27 illustrates how some of these details are presented to the user.



Providers
All emails
Email templates

INVOICES
Providers
All invoices

SHIPPING
Providers
All shippings

PAYMENTS
Providers
All payments

INCOMING PAYMENTS
Providers
All incoming payments

CONFIGURATION
OAUTH
Applications
Oauth applications
Oauth access tokens
Oauth access grants

tech@fnac.pt ...

Home > Shipping services

Shipping services

+ Create new shipping service

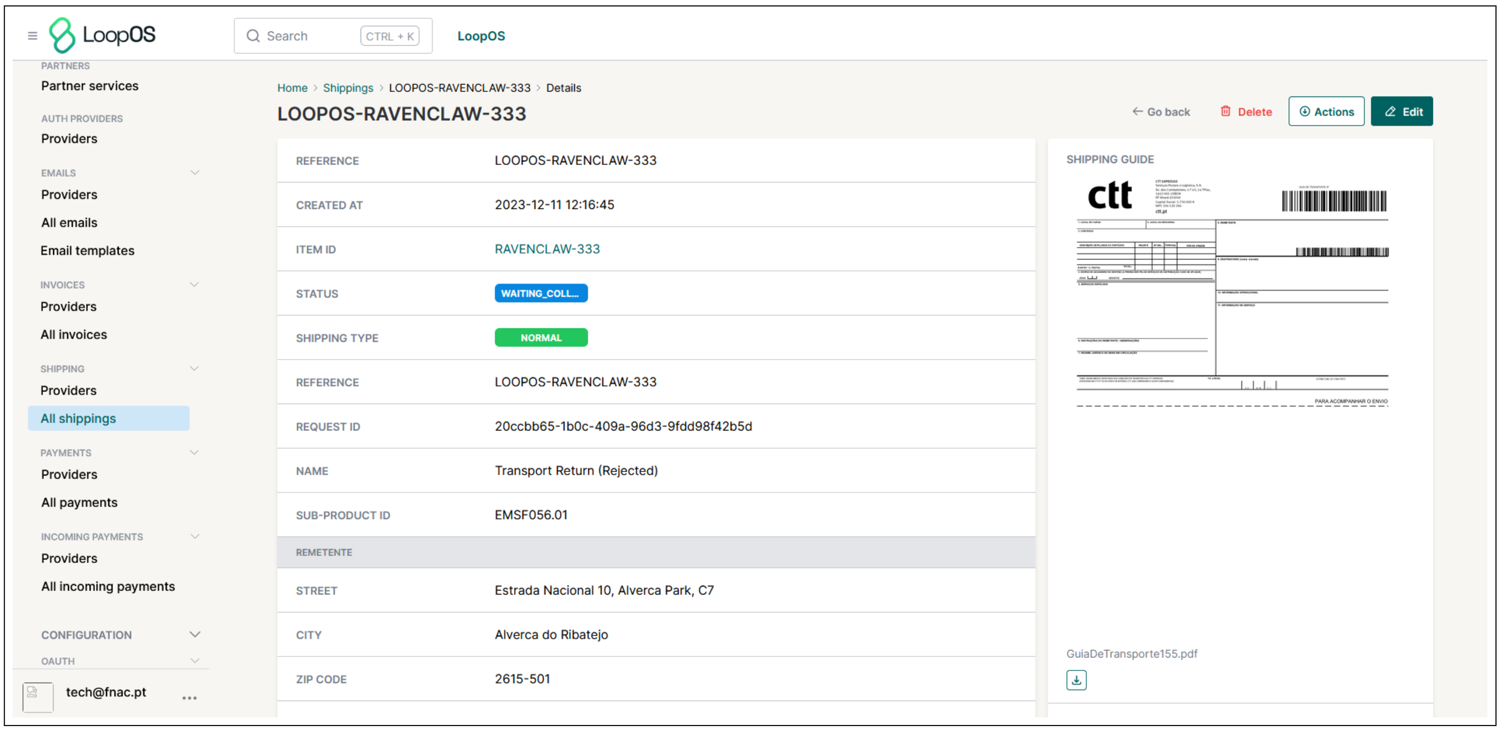
ID	NAME	TYPE
3	Ctt	LoopOsServices::Services::Shippings::Ctt

24 per page

Displaying 1 item

Avo · © 2023 AvoHQ · v2.28.0

Figure 2.26: Network Manager: Shipping Service Providers Index Page.



PARTNERS
Partner services

AUTH PROVIDERS
Providers

EMAILS
Providers
All emails
Email templates

INVOICES
Providers
All invoices

SHIPPING
Providers
All shippings

PAYMENTS
Providers
All payments

INCOMING PAYMENTS
Providers
All incoming payments

CONFIGURATION
OAUTH

tech@fnac.pt ...

Home > Shippings > LOOPOS-RAVENCLAW-333 > Details

LOOPOS-RAVENCLAW-333

← Go back   Delete   Actions   Edit

REFERENCE	LOOPOS-RAVENCLAW-333
CREATED AT	2023-12-11 12:16:45
ITEM ID	RAVENCLAW-333
STATUS	WAITING_COLL...
SHIPPING TYPE	NORMAL
REFERENCE	LOOPOS-RAVENCLAW-333
REQUEST ID	20ccbb65-1b0c-409a-96d3-9fdd98f42b5d
NAME	Transport Return (Rejected)
SUB-PRODUCT ID	EMSF056.01
REMETENTE	
STREET	Estrada Nacional 10, Alverca Park, C7
CITY	Alverca do Ribatejo
ZIP CODE	2615-501

SHIPPING GUIDE

ctt

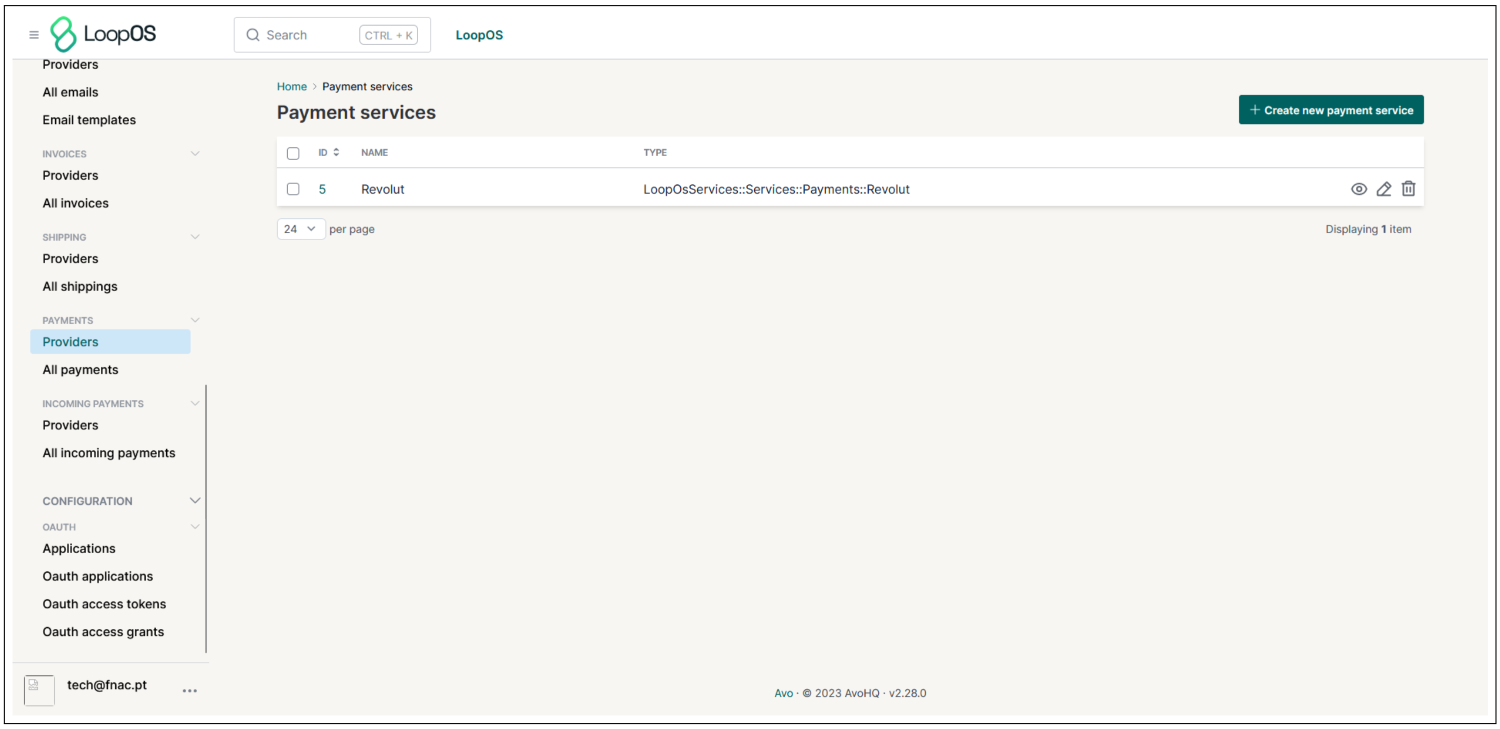
PARA ACOMPANHAR O ENVIO

GuiaDeTransporte155.pdf

Figure 2.27: Network Manager: Shipping Show Page - Partial example of the shipping details of an item.

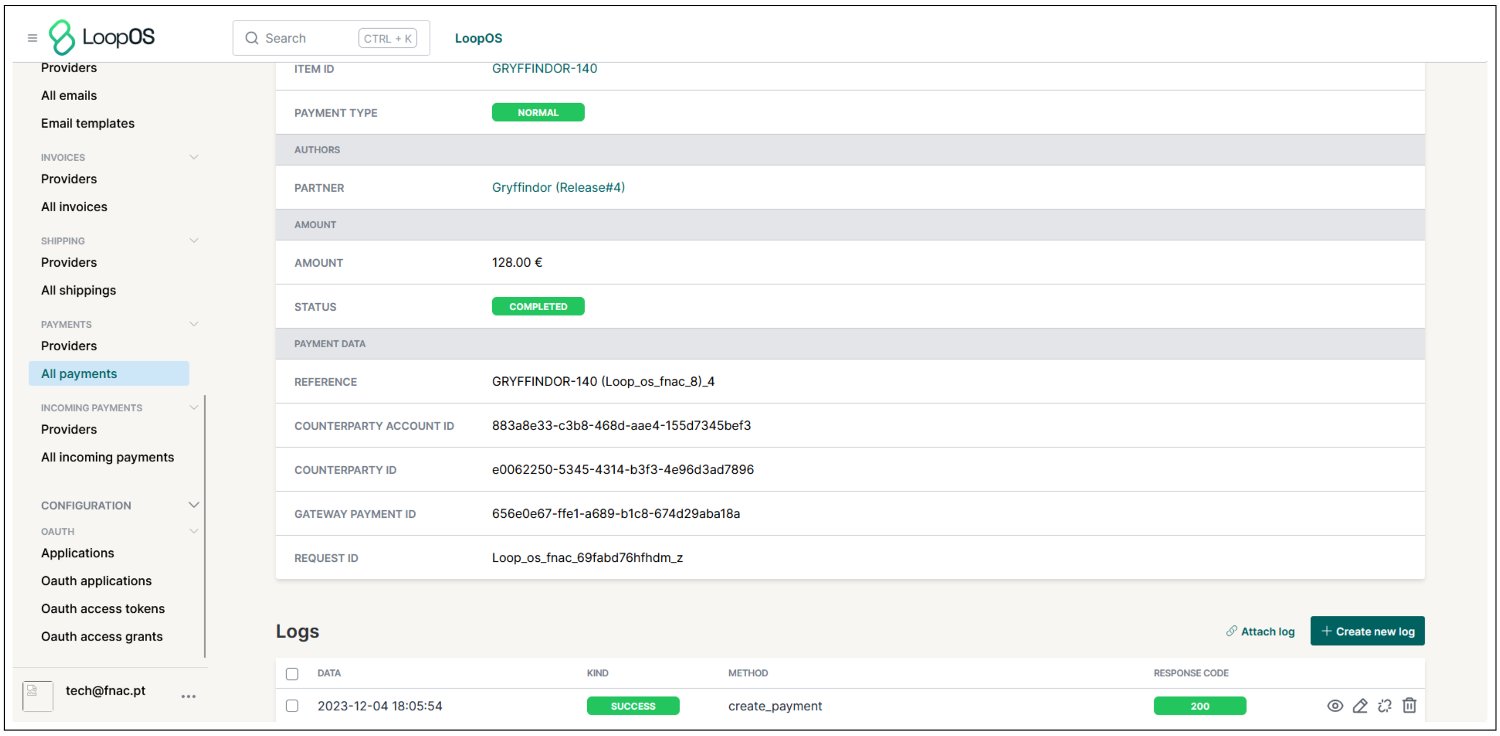
Payments: Hold the payment provides (illustrated in Figure 2.28), which will handle the payments to pay the customer for an item and all the payments by each provider. The general information includes the item ID and its corresponding

URL in the Core App, the reference, creation timestamp, payment type, authors, amount, and status (since there is the possibility the payments must be approved first). Payment data covers reference details and other useful information in order to complete the operation. Logs document actions upon these payments. Figure 2.29 illustrates how some of these details are presented to the user.



ID	NAME	TYPE
5	Revolut	LoopOsServices::Services::Payments::Revolut

Figure 2.28: Network Manager: Payment Service Providers Index Page.



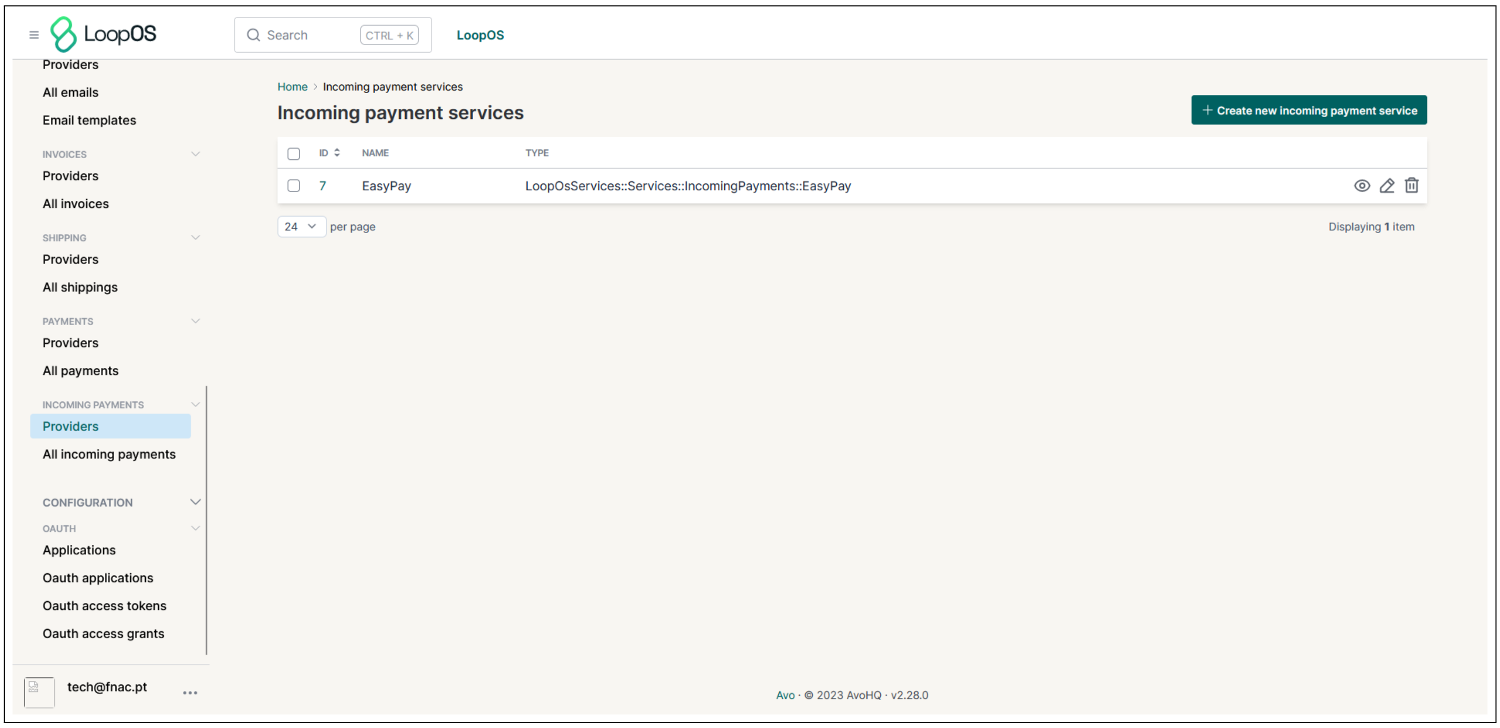
ITEM ID	GRYFFINDOR-140
PAYMENT TYPE	NORMAL
AUTHORS	
PARTNER	Gryffindor (Release#4)
AMOUNT	
AMOUNT	128.00 €
STATUS	COMPLETED
PAYMENT DATA	
REFERENCE	GRYFFINDOR-140 (Loop_os_fnac_8)_4
COUNTERPARTY ACCOUNT ID	883a8e33-c3b8-468d-aae4-155d7345bef3
COUNTERPARTY ID	e0062250-5345-4314-b3f3-4e96d3ad7896
GATEWAY PAYMENT ID	656e0e67-ffe1-a689-b1c8-674d29aba18a
REQUEST ID	Loop_os_fnac_69fabd76hfhdm_z

Logs

DATA	KIND	METHOD	RESPONSE CODE
2023-12-04 18:05:54	SUCCESS	create_payment	200

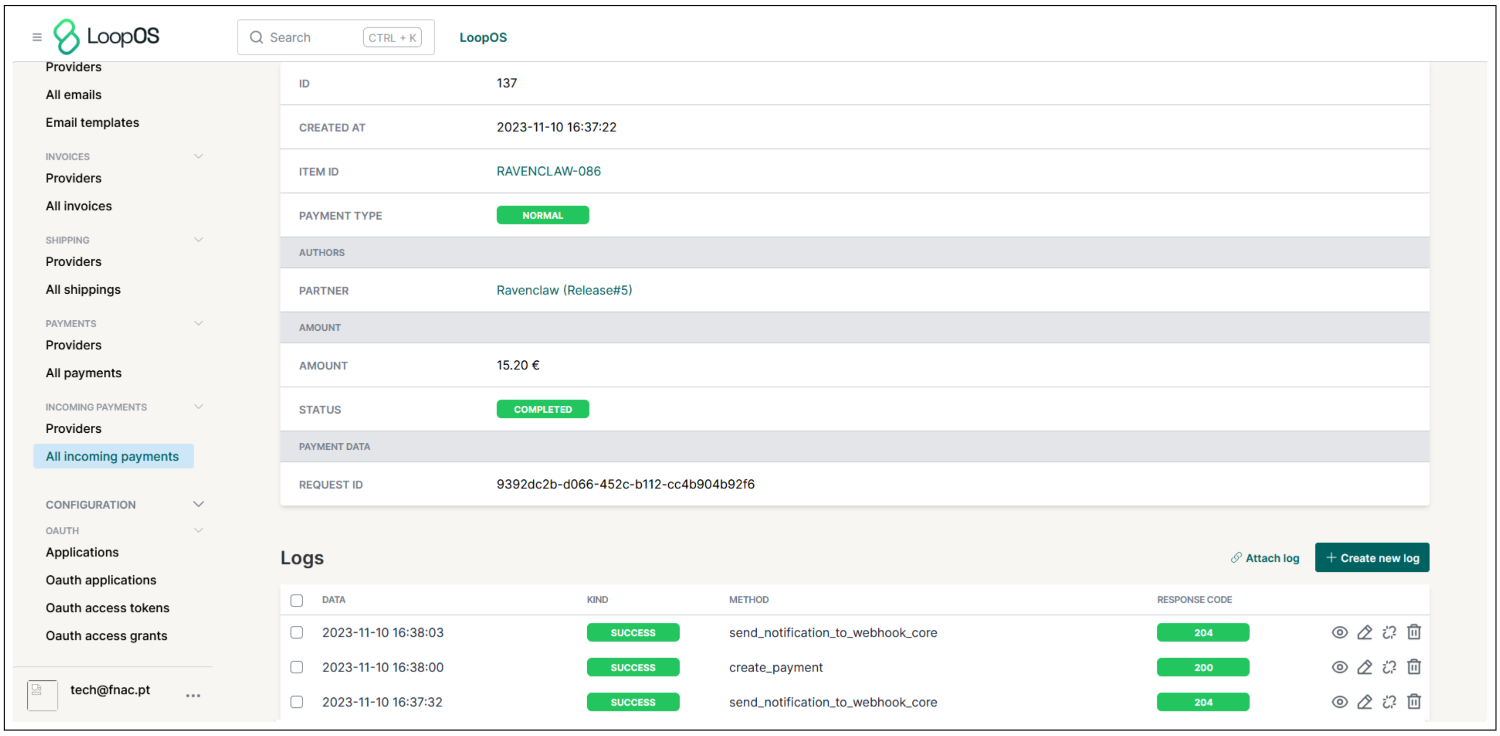
Figure 2.29: Network Manager: Payments Show Page - Partial example of the shipping details of an item.

Incoming Payments: Hold the incoming payment providers (illustrated in Figure 2.30), which will handle the payments coming from the customers and the incoming_payments per provider. The general information includes item id and its corresponding URL in the Core App, created timestamp payment type, the Partner, the amount, and status (since, for example, payments can expire), and payment-specific details, such as the request ID. Logs document actions upon these incoming payments. Figure 2.31 illustrates how some of these details are presented to the user.



ID	NAME	TYPE
7	EasyPay	LoopOsServices::Services::IncomingPayments::EasyPay

Figure 2.30: Network Manager: Incoming Payments Service Providers Index Page.



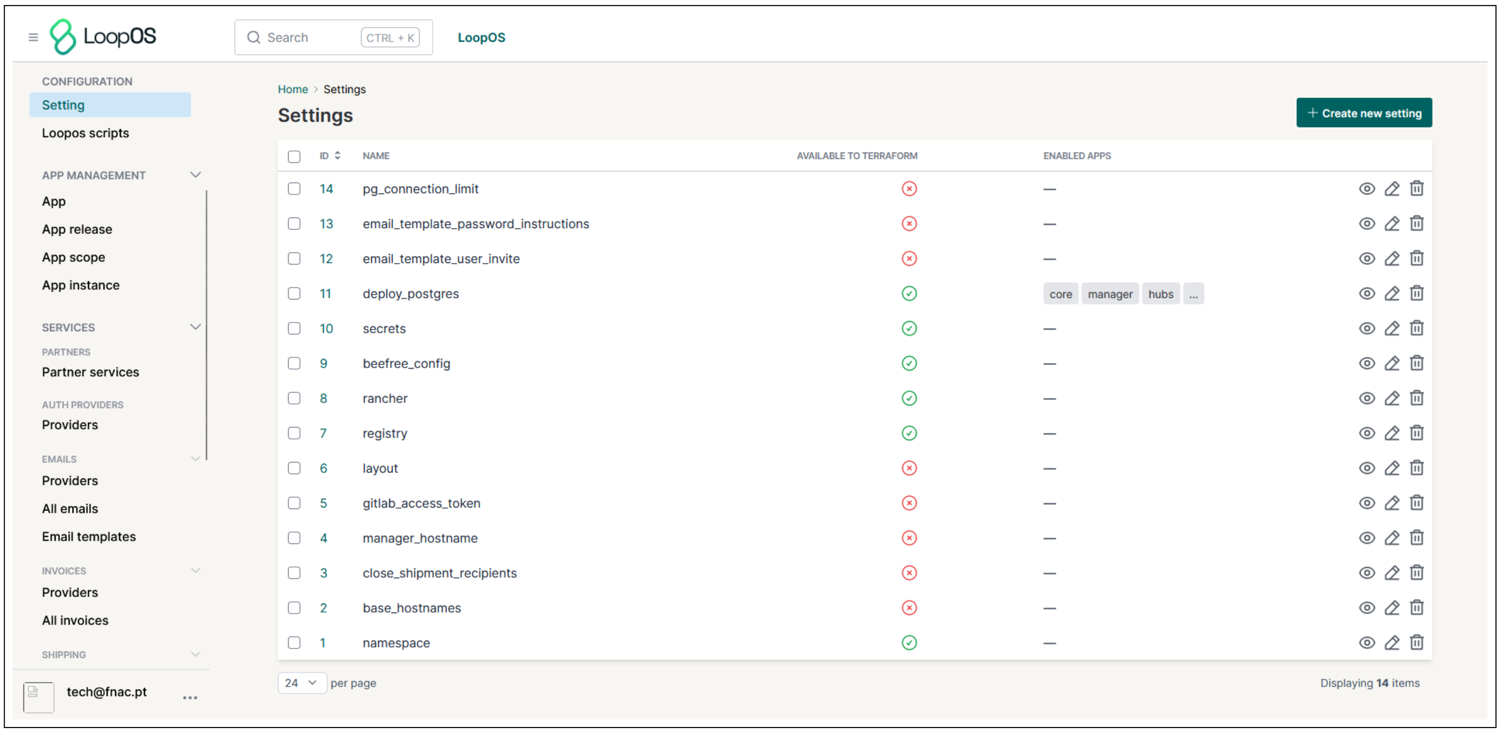
ID	137
CREATED AT	2023-11-10 16:37:22
ITEM ID	RAVENCLAW-086
PAYMENT TYPE	NORMAL
AUTHORS	
PARTNER	Ravenclaw (Release#5)
AMOUNT	
AMOUNT	15.20 €
STATUS	COMPLETED
PAYMENT DATA	
REQUEST ID	9392dc2b-d066-452c-b112-cc4b904b92f6

DATA	KIND	METHOD	RESPONSE CODE
2023-11-10 16:38:03	SUCCESS	send_notification_to_webhook_core	204
2023-11-10 16:38:00	SUCCESS	create_payment	200
2023-11-10 16:37:32	SUCCESS	send_notification_to_webhook_core	204

Figure 2.31: Network Manager: Payments Show Page - Example of the incoming payment details of an item.

Most notably, any of these services created can be immediately linked to a Partner, giving it access to all its features. Hence, each provider added is an additional possible service for the Partner to use, thus enhancing the customization of LoopOS.

Finally, the Network Manager allows to configure general settings for the Manager. Figure 2.32 showcases the current implemented settings. Each setting has five properties: name, its description, the data with the content of the setting, in which App it is enabled and a boolean field if it is available to use in Terraform. The last field is crucial to the deployment of LoopOS, since its used to configure aspects of the Terraform deployment, from Kubernetes secrets, Gitlab credentials to Rancher cluster and project id, in order to deploy the App Instances dynamically. In addition, other settings that are not available to Terraform can vary from Email Template will be used to invite the user, to settings necessary to use certain services, for example, the key and secret to allow the Network Manager to make call to certain Email Provider API's. Figure 2.33 illustrates how the settings of the beefree email service provider can be set.

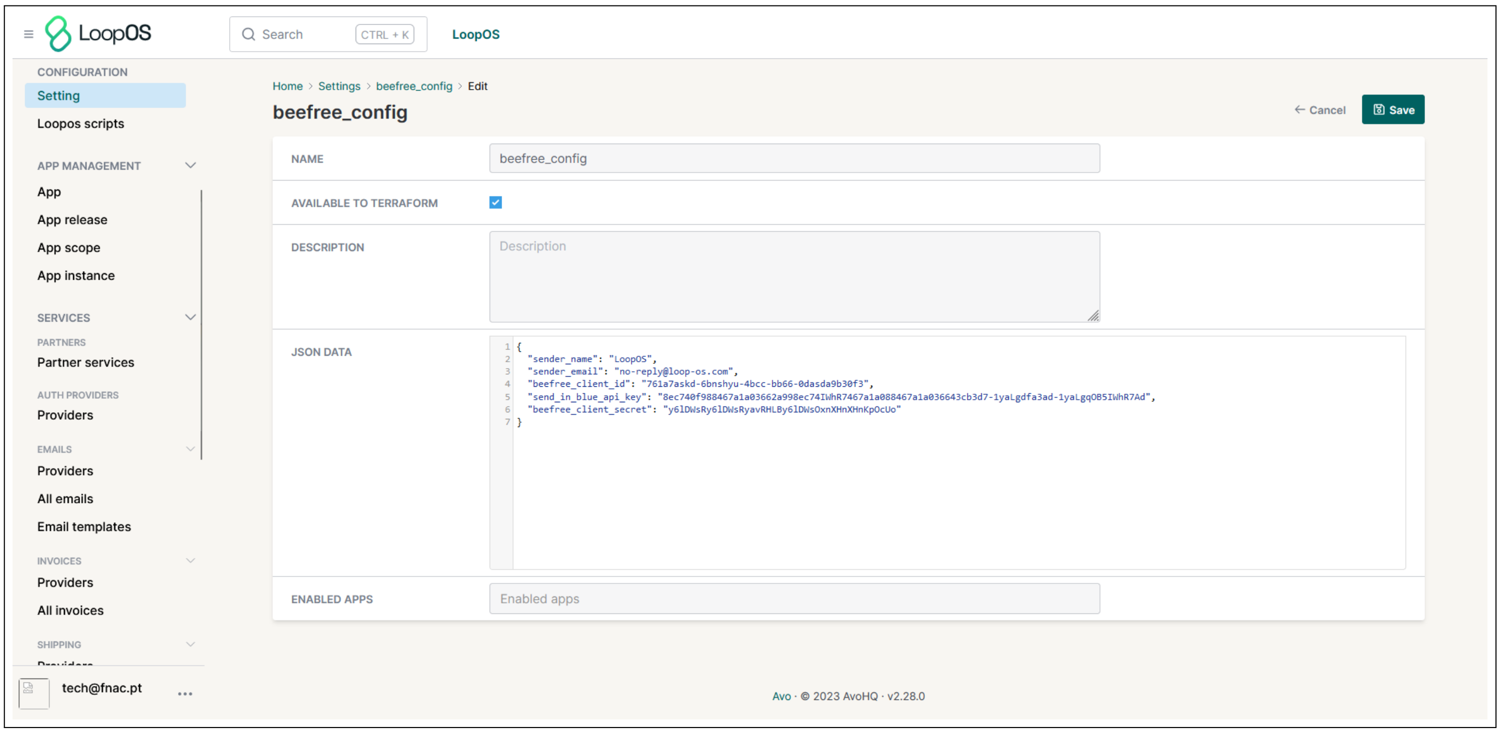


Settings

ID	NAME	AVAILABLE TO TERRAFORM	ENABLED APPS
14	pg_connection_limit	✗	—
13	email_template_password_instructions	✗	—
12	email_template_user_invite	✗	—
11	deploy_postgres	✓	core manager hubs ...
10	secrets	✓	—
9	beefree_config	✓	—
8	rancher	✓	—
7	registry	✓	—
6	layout	✗	—
5	gitlab_access_token	✗	—
4	manager_hostname	✗	—
3	close_shipment_recipients	✗	—
2	base_hostnames	✗	—
1	namespace	✓	—

Displaying **14** items

Figure 2.32: Network Manager: Settings Index Page.



beefree_config

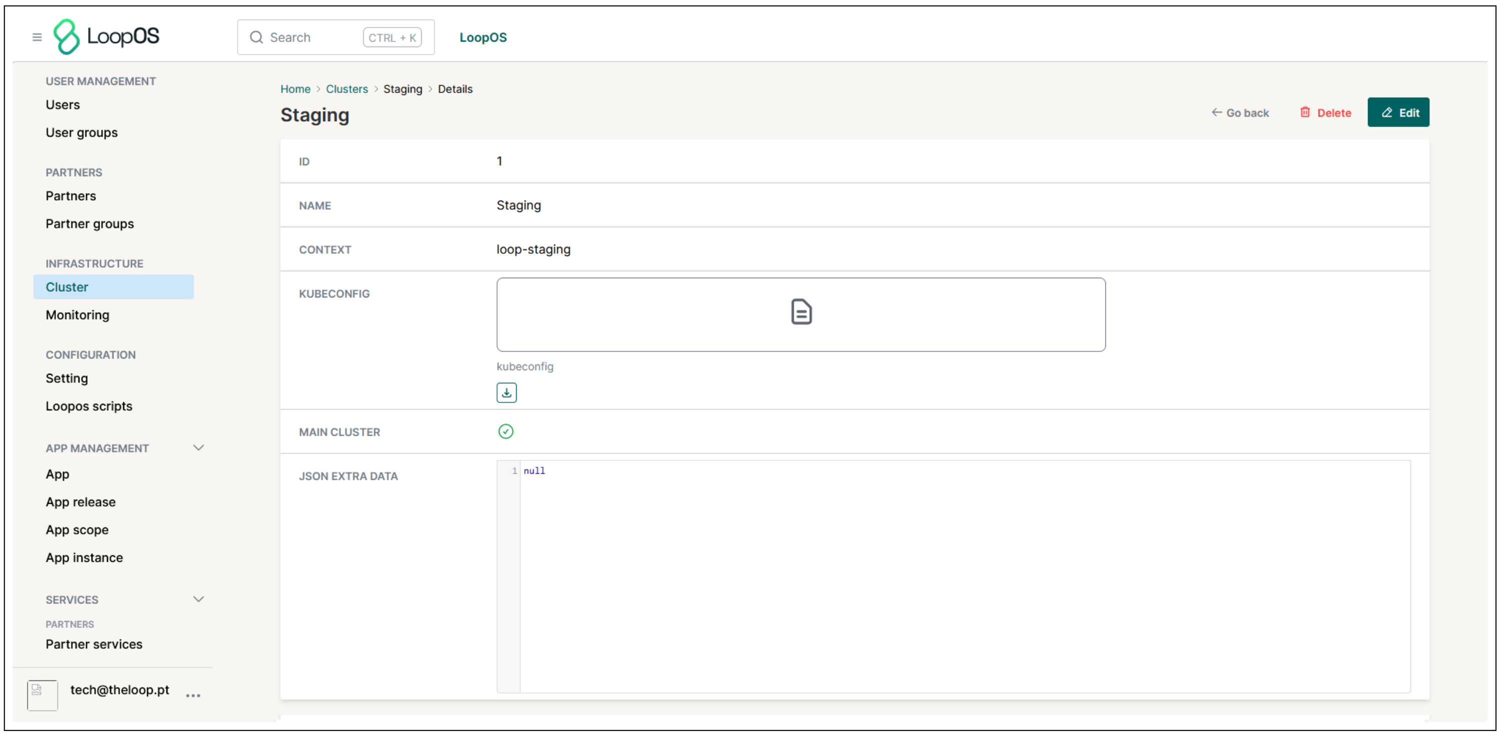
NAME	beefree_config
AVAILABLE TO TERRAFORM	✓
DESCRIPTION	Description
JSON DATA	see below
ENABLED APPS	Enabled apps

```
{
  "sender_name": "LoopOS",
  "sender_email": "no-reply@loop-os.com",
  "beefree_client_id": "761a7askd-6bnshyu-4bcc-bb66-0dasda9b30f3",
  "send_in_blue_api_key": "8ec740f988467a1a03662a998ec74IWhR7467a1a088467a1a036643cb3d7-1yaLgdfa3ad-1yaLgqOB5IWhR7Ad",
  "beefree_client_secret": "y6lDWsRy6lDWsRyavRHLBy6lDWsOxnXHnXHnKpOcUo"
}
```

Figure 2.33: Network Manager: Setting Show Page - Example of a settings implementation.

Clusters Management: *Cluster*

Currently, Kubernetes configurations and additional data are managed for all instances within a single cluster. However, each partner will have their dedicated cluster in the future. Most notably, inside each **Cluster** is a kubeconfig that contains the configuration information for connecting to a Kubernetes cluster. It includes the cluster's name, the server URL, and the authentication information for a user account. It also defines a context, which is a combination of a cluster and a user, and it is used to specify which cluster and user to employ when connecting to Kubernetes. Figure 2.34 illustrates how these details are presented to the user.



Staging

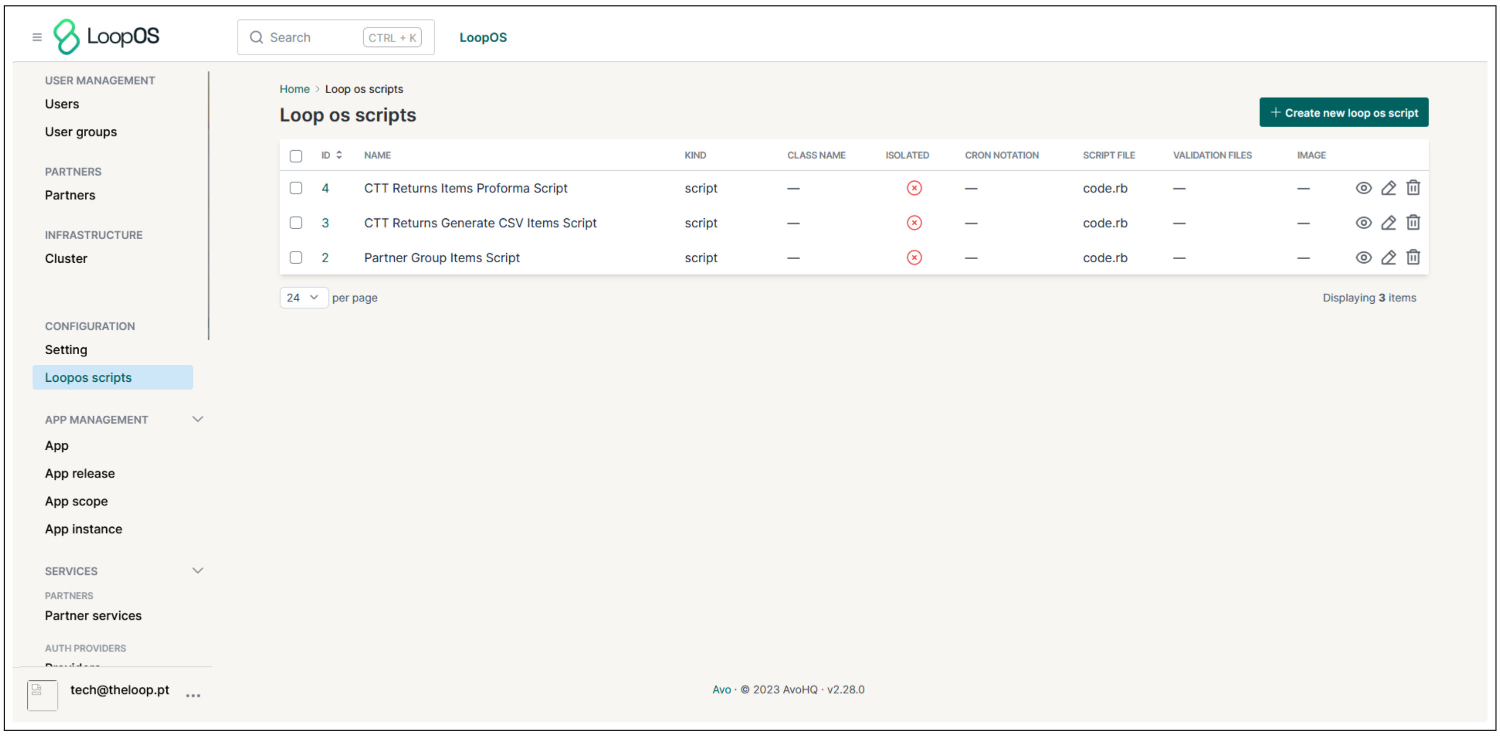
ID	1
NAME	Staging
CONTEXT	loop-staging
KUBECONFIG	kubeconfig
MAIN CLUSTER	✓
JSON EXTRA DATA	null

Figure 2.34: Network Manager: Cluster Show Page.

LoopOS Scripts: *Script, Validator, Boilerplate*

LoopOS Scripts, can be of 3 types: Script, Validator & Boilerplate. They can be executed in two modes: shared mode where script is executed with access to all the resources of the application it runs on (like models/services etc), or isolated mode (default), where it will run outside the context of rails so it won't have access to any resources within the application. This can be set by checking the isolated checkbox while creating/updating the scripts.

The Script type is used for code that can be identified as scripts. That is, a piece of code that can be executed successfully or failed due to some error. The success or error message can be shown using successful or failed methods. Figure 2.35 showcases the index page of LoopOS Scripts with examples of scripts already being used in the Network Manager.



Loop os scripts

ID	NAME	KIND	CLASS NAME	ISOLATED	CRON NOTATION	SCRIPT FILE	VALIDATION FILES	IMAGE
4	CTT Returns Items Proforma Script	script	—	⊗	—	code.rb	—	—
3	CTT Returns Generate CSV Items Script	script	—	⊗	—	code.rb	—	—
2	Partner Group Items Script	script	—	⊗	—	code.rb	—	—

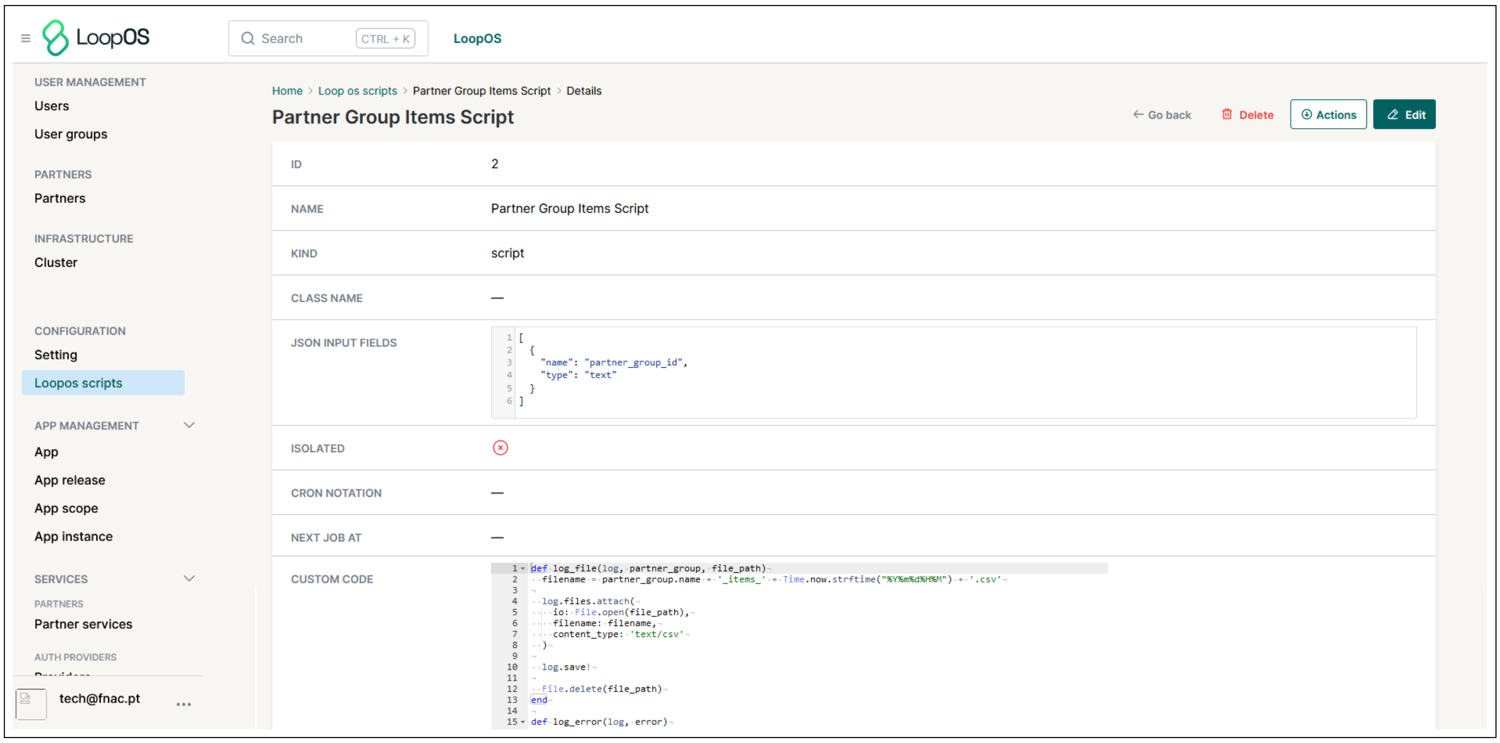
24 per page

Displaying 3 items

Avo · © 2023 AvoHQ · v2.28.0

Figure 2.35: Network Manager: LoopOS Scripts Index Page.

For example, using the script type, i aided in the development of the "Partner Group Items Script", as can be see in Figure 2.36. This script exports items from a chosen partner group (illustrated in Figure 2.37) to a CSV file by retrieving item data from all partners in the group, combining the data into a single CSV file, and logging the export process. These logs save the script's outcome and the generated file, showcased in Figure 2.38.



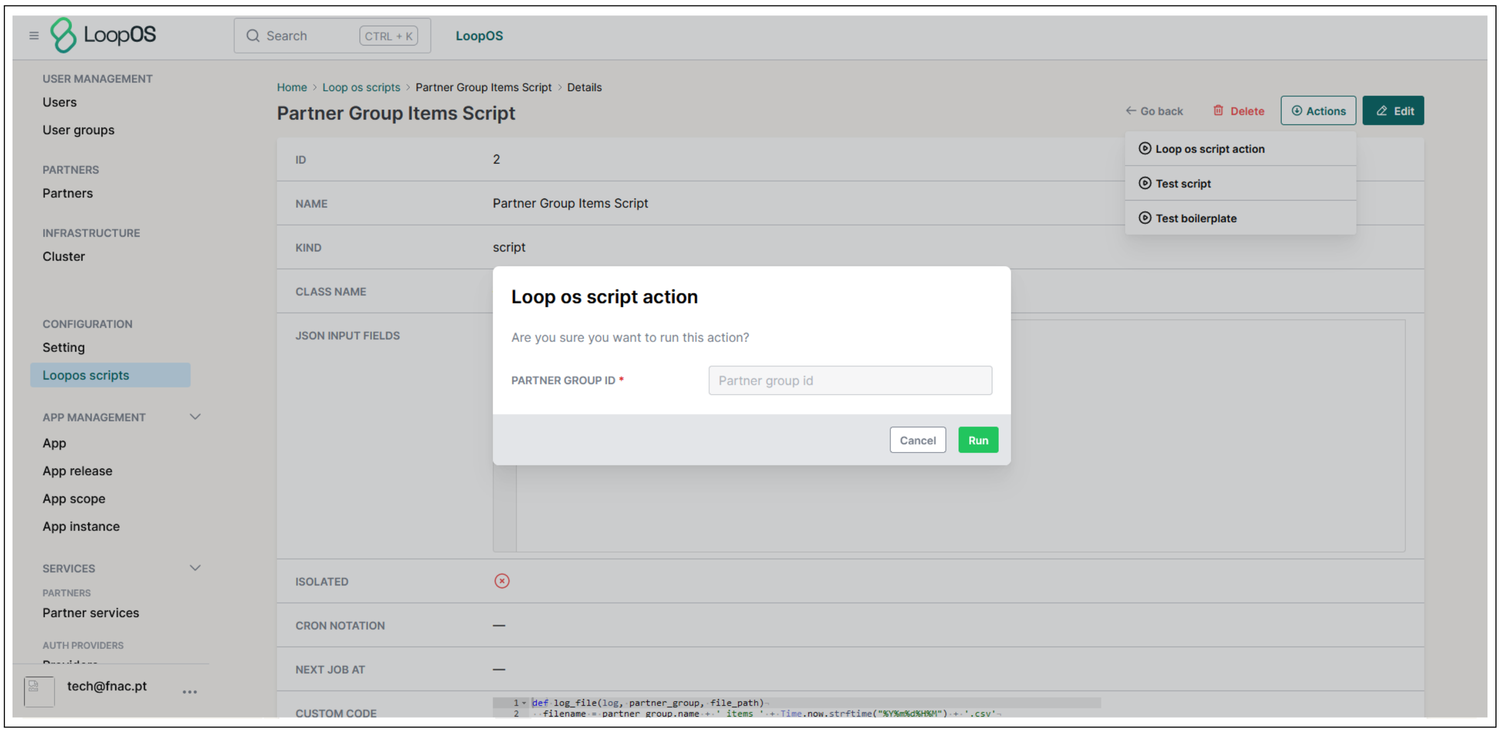
Partner Group Items Script

ID	2
NAME	Partner Group Items Script
KIND	script
CLASS NAME	—
JSON INPUT FIELDS	see below
ISOLATED	⊗
CRON NOTATION	—
NEXT JOB AT	—
CUSTOM CODE	see below

```
[
  {
    "name": "partner_group_id",
    "type": "text"
  }
]
```

```
def log_file(log, partner_group, file_path)
  filename = partner_group.name + '_items_' + Time.now.strftime("%Y%m%d%H%M") + '.csv'

  log.files.attach(
    io: File.open(file_path),
    filename: filename,
    content_type: 'text/csv'
  )

  log.save!

  File.delete(file_path)
end

def log_error(log, error)
```

Figure 2.36: Network Manager: LoopOS Scripts Show Page - Partner Group Items Script.



Partner Group Items Script

Loop os script action

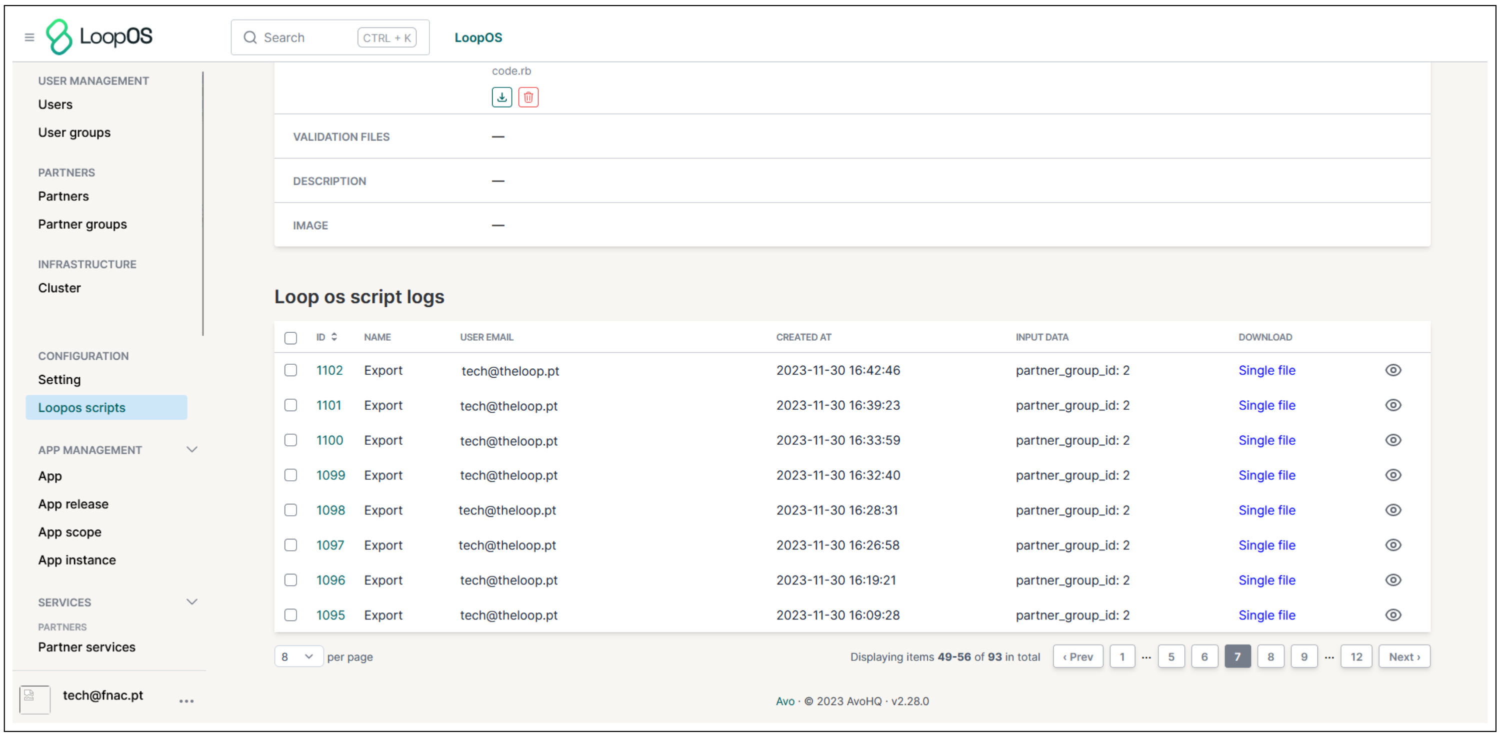
Are you sure you want to run this action?

PARTNER GROUP ID * : Partner group id

ID	2
NAME	Partner Group Items Script
KIND	script
CLASS NAME	
JSON INPUT FIELDS	
ISOLATED	⊗
CRON NOTATION	—
NEXT JOB AT	—
CUSTOM CODE	

```
def log_file(log, partner_group, file_path)
  filename = partner_group.name + '_items_' + Time.now.strftime("%Y%m%d%H%M") + '.csv'
```

Figure 2.37: Network Manager: LoopOS Scripts Show Page - Partner Group Items Script input section.



code.rb

VALIDATION FILES	—
DESCRIPTION	—
IMAGE	—

Loop os script logs

ID	NAME	USER EMAIL	CREATED AT	INPUT DATA	DOWNLOAD
1102	Export	tech@theloop.pt	2023-11-30 16:42:46	partner_group_id: 2	Single file
1101	Export	tech@theloop.pt	2023-11-30 16:39:23	partner_group_id: 2	Single file
1100	Export	tech@theloop.pt	2023-11-30 16:33:59	partner_group_id: 2	Single file
1099	Export	tech@theloop.pt	2023-11-30 16:32:40	partner_group_id: 2	Single file
1098	Export	tech@theloop.pt	2023-11-30 16:28:31	partner_group_id: 2	Single file
1097	Export	tech@theloop.pt	2023-11-30 16:26:58	partner_group_id: 2	Single file
1096	Export	tech@theloop.pt	2023-11-30 16:19:21	partner_group_id: 2	Single file
1095	Export	tech@theloop.pt	2023-11-30 16:09:28	partner_group_id: 2	Single file

8 per page   Displaying items **49-56** of **93** in total   ‹ Prev 1 … 5 6 7 8 9 … 12 Next ›

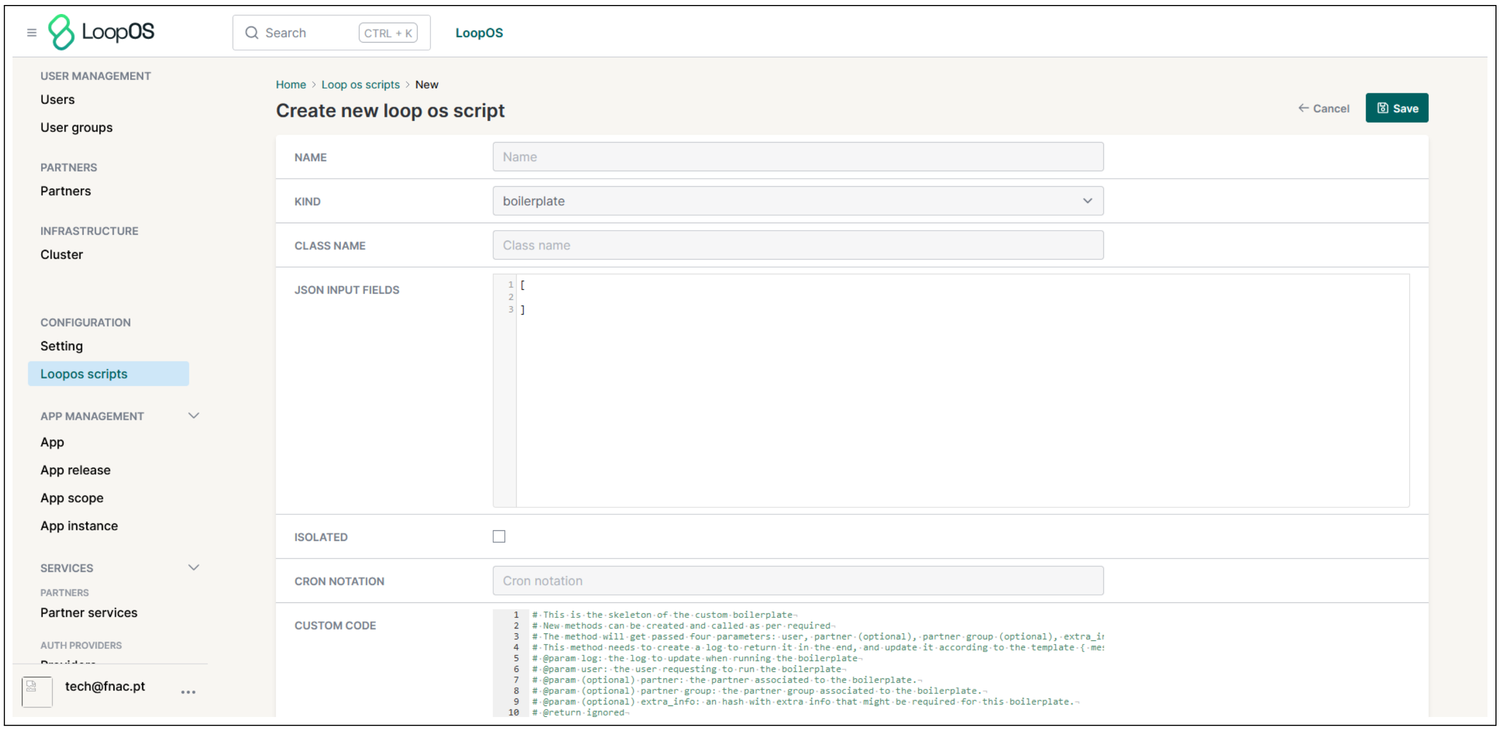
Avo · © 2023 AvoHQ · v2.28.0

Figure 2.38: Network Manager: LoopOS Scripts Show Page - Partner Group Items Script generated logs.

Validator kind is used for code that can be identified as a validator. For example, a code that is used to validate a zip code. If a zip code is valid, it will return true, else, it will return some error. Similar to the script type, success or error messages can be shown using successful or failed methods. Moreover, in validator, files can also sent to parameters for validation.

Most notably, the Boilerplate type was created to provide a customizable structure for creating and invoking methods to be used for creating a Partner environment in LoopOS. The method takes in four arguments: user, optionally partner, partner group, and extra_info. Its primary objective is to generate and update a log based on a predefined template containing message, percentage . Boilerplate is used to define custom methods tailored to specific requirements while ensuring consistent log generation and updates based on the provided parameters.

Chapter 6 - Development will delve into my creation and implementation of a boilerplate-type script (illustrated in Figure 2.39) that, by exploiting all these mechanics implemented before, significantly contributed to the successful automation of partner environment deployment.



Create new loop os script

NAME	Name
KIND	boilerplate
CLASS NAME	Class name
JSON INPUT FIELDS	[ ]
ISOLATED	☐
CRON NOTATION	Cron notation
CUSTOM CODE	

```
# This is the skeleton of the custom boilerplate
# New methods can be created and called as per required
# The method will get passed four parameters: user, partner (optional), partner group (optional), extra_i
# This method needs to create a log to return it in the end, and update it according to the template { me
# @param log: the log to update when running the boilerplate
# @param user: the user requesting to run the boilerplate
# @param (optional) partner: the partner associated to the boilerplate.
# @param (optional) partner group: the partner group associated to the boilerplate.
# @param (optional) extra_info: an hash with extra info that might be required for this boilerplate.
# @return ignored
```

Figure 2.39: Network Manager: LoopOS Scripts Show Page - Empty Boilerplate Script.

In the subsequent sections, I explore two primary domains overseen by the Network Manager: Application Deployment & Management and Security. A thorough comprehension of the Network Manager's role within the LoopOS system is established by building upon the earlier context and delving into these areas. This comprehensive understanding sets the stage for presenting the decisions and developments undertaken to achieve the objectives outlined in this thesis.

2.4.1 Application Deployment & Management

In recent years, microservice architecture has become popular due to its numerous advantages, including improved availability, fault tolerance, and horizontal scalability (increasing the capacity of a system by adding additional machines). Furthermore, the adoption of container technologies, such as Kubernetes [3] and Docker [29], have further boosted the momentum of microservices, especially in cloud-based environments. As a result, many prominent companies like Amazon, Bestbuy.com, Coca-Cola, eBay, Etsy, Gilt.com, Netflix, Spotify, Uber, and Zalando have successfully adopted this architecture [28].

In LoopOS, although the applications serve a distinct purpose in the ecosystem, they are not strictly microservices since there is a degree of centralized control and interdependence between them. However, they do not follow a strict monolith architecture since they are different applications with different responsibilities.

This system follows a Service-oriented architecture (SOA) [30] pattern, where applications are built as a collection of services. Each service encapsulates a specific business capability and communicates with other services through a well-defined interface, typically over a network protocol, using the HTTPS network protocol in this particular case. Figure 2.40 demonstrates the differences in isolation between Monolithic, SOA, and Microservice architecture.

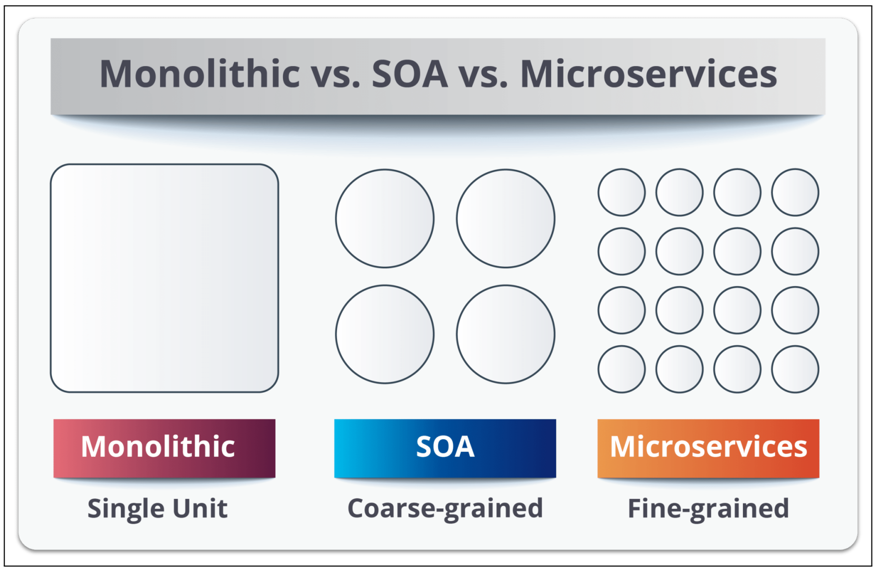


Figure 2.40: Differences in isolation: Monolithic vs SOA vs Microservice.

In the scientific literature, plenty of methods exist to deploy these services [31]. The Network Manager, the sole application that deploys, manages, and coordinates the various other applications, is thus considered an Orchestrator.

The LoopOS technological ecosystem is deployed with Rancher [55], a container management platform. Rancher provides a centralized management interface for multiple Kubernetes [3] clusters, enabling the orchestration of containers across different environments. In the case of LoopOS, within Rancher, there is only one Kubernetes cluster, which has multiple pods. Each pod represents either a specific application instance or a containerized tool that supports that application instance. For example, a PostgreSQL [35] database container could be deployed within a pod to provide data storage for an application.

Rancher uses Terraform [46], an Infrastructure as Code (IaC) tool, to provision and manage the infrastructure resources required by the LoopOS applications. Terraform allows for declarative infrastructure provisioning, meaning that the desired state of the infrastructure is defined in a configuration file. Terraform automatically creates or updates the necessary resources to match the desired state.

The Network Manager interacts with Rancher and Terraform to automate the deployment process for LoopOS applications.

Applications are created in the Network Manager interface and linked to a GitLab image via the creation of an App Release. The following breakdown and the accompanying Figure 2.41 provide a comprehensive explanation of the deployment process:

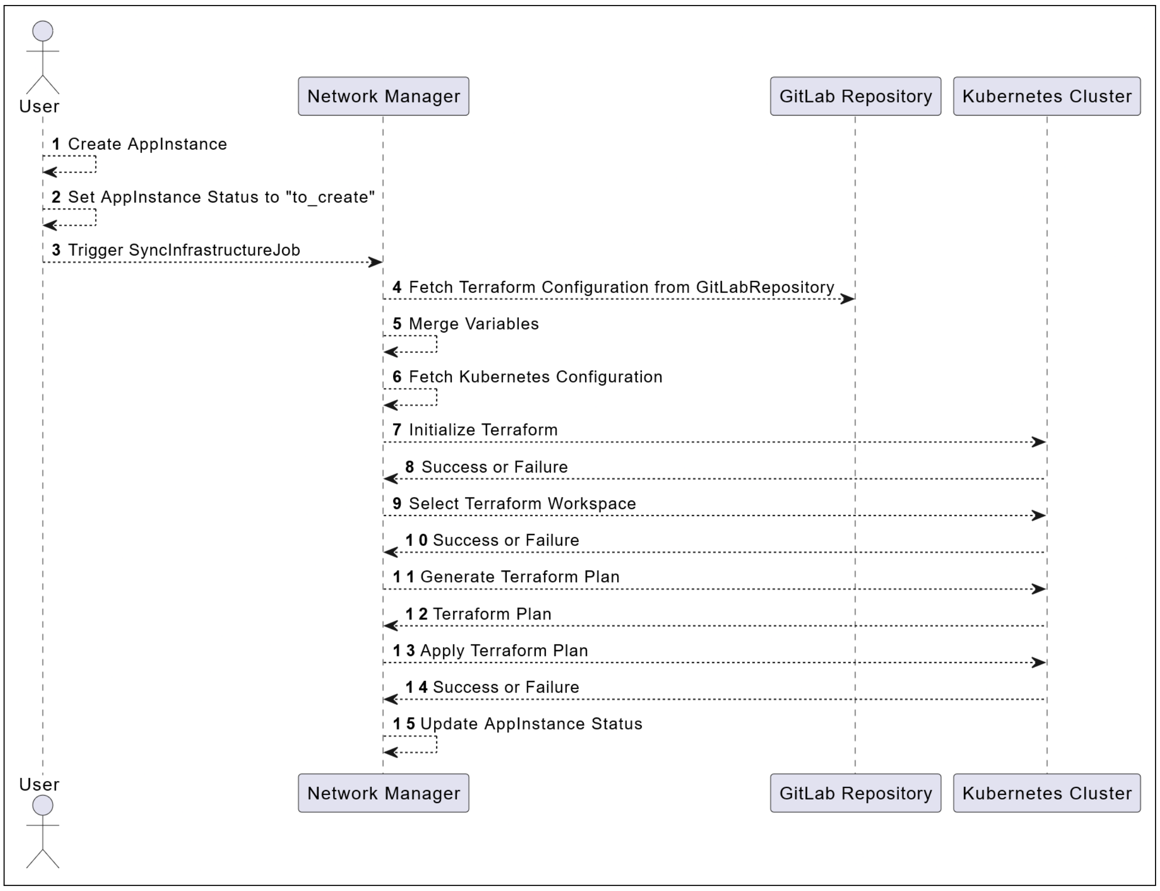


Figure 2.41: UML Diagram: App Instance Infrastructure Deployment.

1. **Create an App Instance:** A new App Instance is created in the Network Manager with the status set to to_create. This indicates that the infrastructure for the app instance still needs to be provisioned.

2. **Trigger SyncInfrastructureJob:** The SyncInfrastructureJob is manually triggered or scheduled to run. This job fetches the latest Terraform configuration file from the GitLab repository associated with the app release.

3. **Merge Terraform Variables:** The job merges the necessary Terraform variables from various sources:

   - App Instance Infrastructure Settings: These settings are specific to the app instance and define the infrastructure requirements, such as the deployment type, resource configuration, and networking setup.
   - Env-Wide Settings: These settings apply to all app instances within a particular environment and may include overarching configurations like region, availability zone, and external dependencies.
   - Infrastructure Generated Data: These data elements are generated during the infrastructure provisioning process and may include instance IDs, IP addresses, and other unique identifiers.
   - Container Image and Hostnames: These values specify the container image to deploy and the hostnames for the application instances.

4. **Fetch Kubernetes Configuration:** The job fetches the Kubernetes configuration (kubeconfig file) for the cluster associated with the app instance. This file is necessary to connect to the cluster and manage the application resources.

5. **Initialize Terraform:** Terraform is initialized, which sets up the working directory and configures Terraform to use the backend configuration. The backend configuration specifies where Terraform stores state information about the infrastructure.

6. **Select Terraform Workspace:** The job selects the Terraform workspace corresponding to the app instance's ID. This ensures that changes are applied to the correct environment and avoid conflicts with other deployments.

7. **Generate Terraform Plan:** A Terraform plan is generated, which describes the changes that will be made to the infrastructure. This plan includes the resources that will be created, updated, or destroyed.

8. **Apply Terraform Plan:** The generated Terraform plan is applied to the infrastructure. This process makes the actual changes to the cluster, creating or updating the necessary resources.

9. **Update App Instance Status:** Upon successful completion of the Terraform plan application, the app instance's status is updated to reflect the successful creation or update of the infrastructure. This indicates that the app is ready for use.

10. **Handle GitLab Repository Changes:** If there is a change in the GitLab repository associated with the app release, the SyncInfrastructureJob is triggered again to re-fetch the updated Terraform configuration and apply the changes to the infrastructure. This ensures that the infrastructure remains consistent with the latest code changes.

Hence, the deployment process within the LoopOS ecosystem is seamlessly orchestrated by the Network Manager, which serves as the central hub for creating, managing, and coordinating various applications by leveraging the integration with Rancher and GitLab, enabling a robust Continuous Integration/Continuous Deployment (CI/CD) pipeline.

In addition, as stated before, the Network Manager can execute code directly to these instances - Remote Code Execution (RCE). This is achieve due to App Instance model having a Rails Concern (extends the methods of certain class) entitled *KubeDeploymentInstanceConcern*, that extends the App Instance methods to handle its corresponding Pod. One specific method, is called *run_code* which executes the chosen code on the chosen LoopOS Core inside its Kubernetes Pod.

The following detailed breakdown and the accompanying Figure 2.42 provide a comprehensive explanation of RCE:

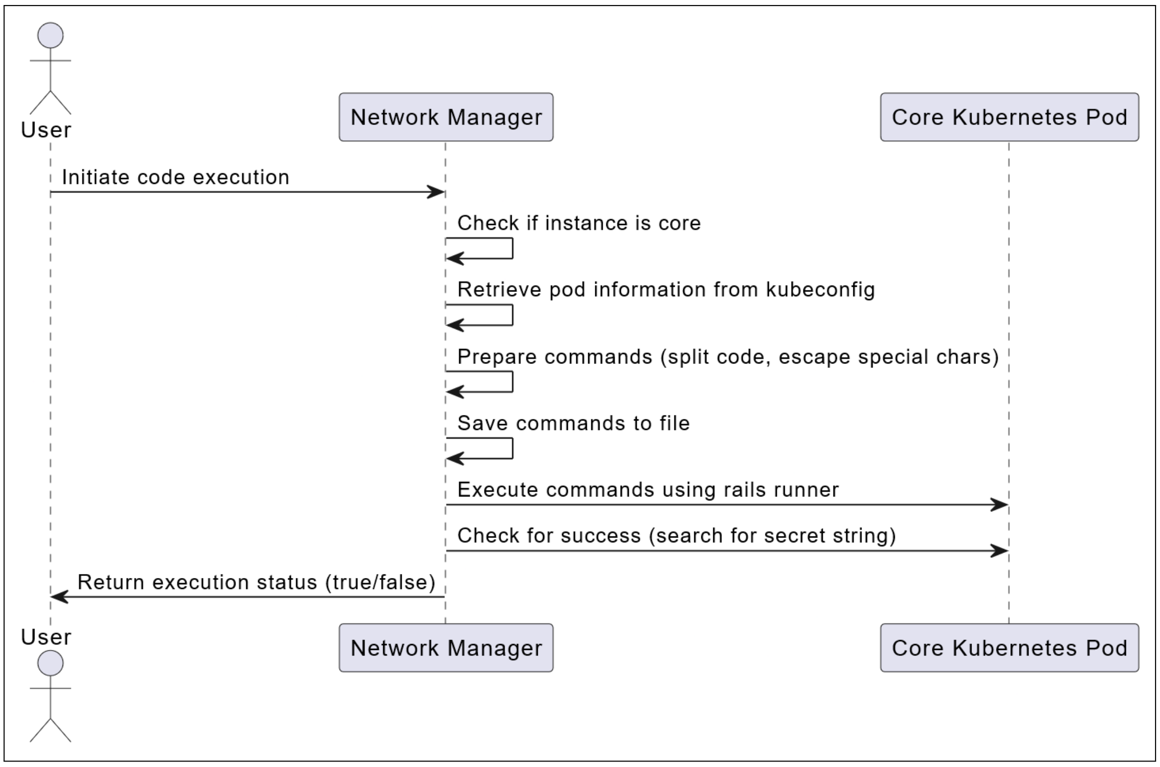

User
Network Manager
Core Kubernetes Pod
Initiate code execution
Check if instance is core
Retrieve pod information from kubeconfig
Prepare commands (split code, escape special chars)
Save commands to file
Execute commands using rails runner
Check for success (search for secret string)
Return execution status (true/false)
User
Network Manager
Core Kubernetes Pod

Figure 2.42: UML Diagram: Secure Code Execution Flow

1. **Retrieve the pod information:** The method retrieves the pod information for the core instance which returns the Kubernetes pod object for the core instance.

2. **Prepare the commands:** The method splits the provided code string into individual commands and escapes any special characters that could cause issues when running the commands in the Kubernetes pod.

3. Execute the commands: The method executes the commands by first storing them in a file to prevent command injection attacks and then using the rails runner command to execute the commands stored in the file.

4. **Check for success:** The method checks if the execution of the commands was successful by searching for a special secret string that was included in the code. If the secret string is found, the method returns true, indicating that the code execution was successful. Otherwise, it returns false.

In conclusion, not also is it possible to manage and deploy App Instances to Kubernetes it is also possible to execute code in any LoopOS Core instances, while also employing mitigation strategies to minimize the risk of command inject attacks by storing commands in a file to prevent direct execution by the Kubernetes pod, using rails runner to execute the commands in a controlled Rails application environment, escaping special characters before storing the commands to prevent misinterpretations and checking for success using a secret string in the output to ensure command integrity.

2.4.2 Security

Authentication involves verifying the identity of a person or entity to ensure that they are who they claim to be. Authorization, conversely, entails granting permission to a person or entity to perform a specific action or access a particular resource [32].

Different mechanisms exist for authenticating applications and individual users within the LoopOS ecosystem. This is a common practice in systems where both applications and users need to access resources, and different authentication flows are tailored to the specific needs of each entity.

It is important to note that both processes involve the validation of credentials by the Network Manager before granting access tokens. The access tokens serve as a form of authorization for the client application to access specific resources within LoopOS.

LoopOS utilizes two distinct authentication flows: one for Apps Authentication and the other for Users Authentication. LoopOS implements the Client Credentials Flow Pattern for Apps Authentication, a standard OAuth2.0 flow for authenticating client applications [24]. An illustration of this process is present in Figure 2.43.

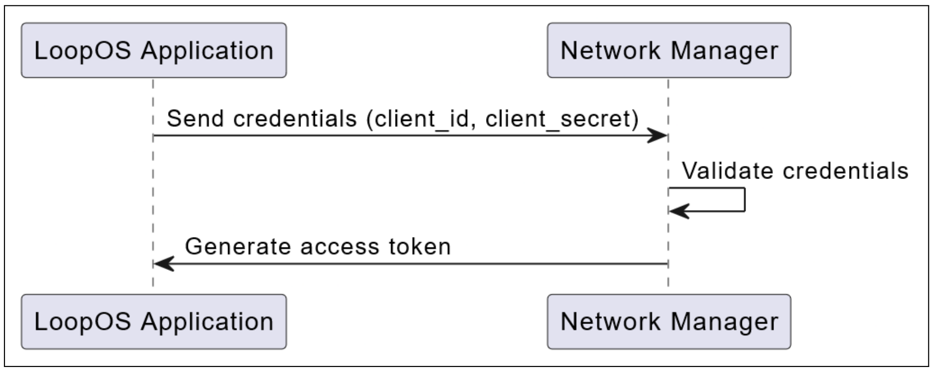


Figure 2.43: Authentication Flow in LoopOS App Authentication [16].

1. When an application wants to authenticate with LoopOS for accessing resources, it sends its credentials, including the *client_id* and *client_secret*, directly to the authorization server (Network Manager) over a secure channel (HTTPS).
2. The authorization server validates the client credentials and generates an access token if they are valid.
3. Once the access token is generated, the authorization server sends it back to the client application.

The client application then includes this access token in its requests to LoopOS to access protected resources as proof of the client application's identity and permissions.

LoopOS verifies the access token in incoming requests from client applications. It checks the integrity and authenticity of the token and decodes the token to extract information about the client application and its permissions.

The Network Manager lists all applications and their respective *client_id* and *client_secret*, so when it receives them, it can validate the applications. Assignment of the *client_id* and *client_secret* is done manually. However, in the future, this will be done automatically by the Network Manager.

On the other hand, LoopOS employs the Password Grant Type Pattern, a standard OAuth2.0 flow for Users Authentication[23]. The following are the steps required for user authentication, and an illustration of this process is present in Figure 2.44.

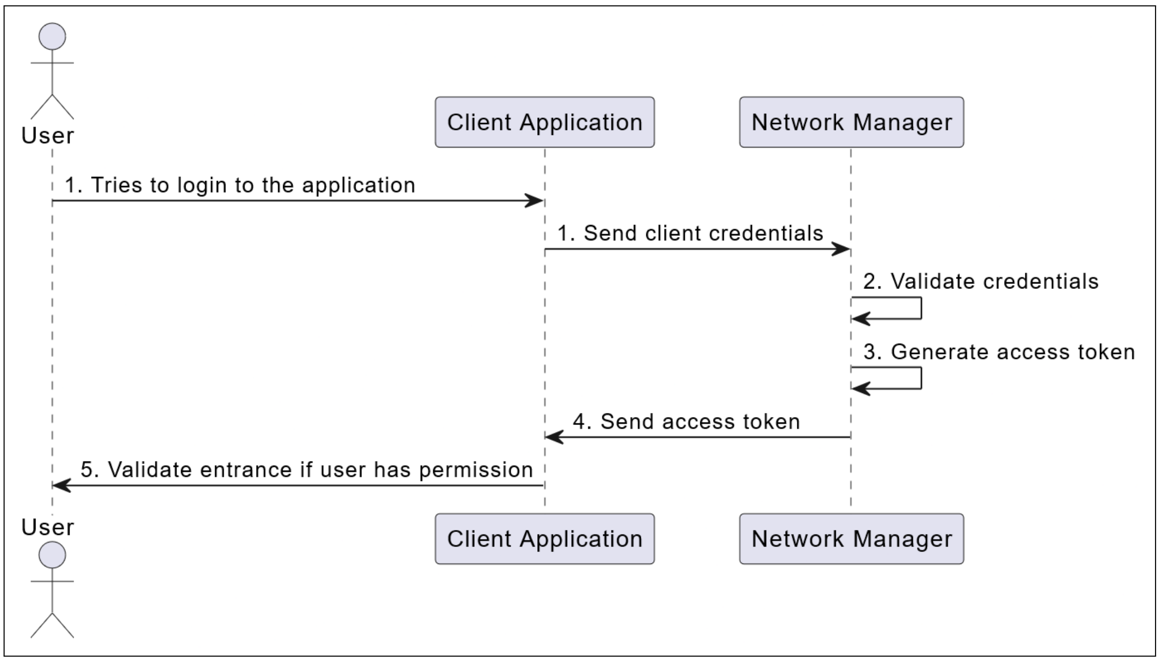


Figure 2.44: Authentication Flow in LoopOS User Authentication [16].

1. When a user wants to authenticate with LoopOS, the client application collects the user's credentials, which include a username and password.

2. The client application then securely sends the user's credentials to the authorization server (Network Manager) over a protected channel.

3. The authorization server validates the user's credentials. The authorization server issues an access token to the client application if the credentials are valid. The access token contains information about the authenticated user and their permissions.

Chapter 3

State of the Art

Effectively orchestrating and managing users, partners, and applications is essential for success in the LoopOS ecosystem. The Network Manager plays a pivotal role in this orchestration. This State of the Art delves into the user, partner, and application management, drawing comparisons between the Network Manager and other leading platforms. By identifying strengths and areas for improvement, I aim to comprehensively outline the Network Manager's capabilities and establish a foundation for enhancing them to meet this thesis objectives.

3.1 User Management

Comprehensive user management is crucial to orchestrate an ecosystem with such a distributed architecture and diverse user base, as many different users and applications need to interact with the system.

The Network Manager achieves this by providing a structured and efficient framework for managing user access, permissions, and settings using Users, User Groups, and App Scopes.

However, to implement a comprehensive solution for user management, it is crucial to analyze the Network Manager's user management capabilities by comparing them to other software solutions.

For this analysis, I chose Human Resources (HR) management software solutions. While these management tools primarily focus on employee management, the capabilities embedded within these solutions provide a relevant benchmark for evaluating the Network Manager user management system, access control, user monitoring, and self-service portals.

The following HR management software solutions have been selected for comparison:

- **BambooHR [7]:** HR platform offering a comprehensive suite of features, including access control, performance management, and engagement tools.
- **Gusto [37]:** Payroll, benefits, and human resource management software com-

pany that provides businesses with a comprehensive suite of tools to manage their employees.

- **Leapsome [48]:** A versatile people management platform encompassing performance management, employee engagement, development and recruiting.

- **15Five [1]:** A goal-setting and progress-tracking platform that facilitates goal setting, progress tracking, feedback exchange, team alignment, and communication.

An analysis of these software solutions compared to the Network Manager revealed the following:

In terms of access control, all software solutions have access control features that are simpler than the Network Manager's. The Network Manager has a more comprehensive granular access control system based on roles, permissions, and user states and includes user-specific and group-based access. This system grants permissions to the apps a user can access and their actions within those apps. However, a gap in its access control system is the lack of any granular access control to the pages on the Network Manager itself.

Concerning user management, all software solutions offer user creation, modification, and termination. However, most software solutions provide a survey for users to complete when their accounts are terminated.

Regarding self-service portals, the only features standard to the other software solutions missing from the Network Manager are a place in the user profile to leave feedback for their Partner and/or PartnerGroup to review. This feature could be beneficial to implement in the Network Manager.

Finally, all these platforms provide user analytics. In the Network Manager case, this feature could provide insights into user behavior, such as which applications are frequently used and where the user encounters most difficulties.

In conclusion, all five platforms offer customizable user management features to adapt to the unique needs of different organizations. However, the Network Manager's ability to define and assign permissions across multiple applications and its support for user groups and settings inheritance provides a more granular and flexible approach to user management. However, the Network Manager lacks access control in its pages, user feedback collection, and user analytics. These additional features could further enhance the Network Manager's user management capabilities and make it an even more comprehensive solution.

3.2 Partner Management

Partner Management is pivotal to ensure the seamless operation of the LoopOS ecosystem. With so many Partners, it is essential to implement an effective Partner management that covers all their needs.

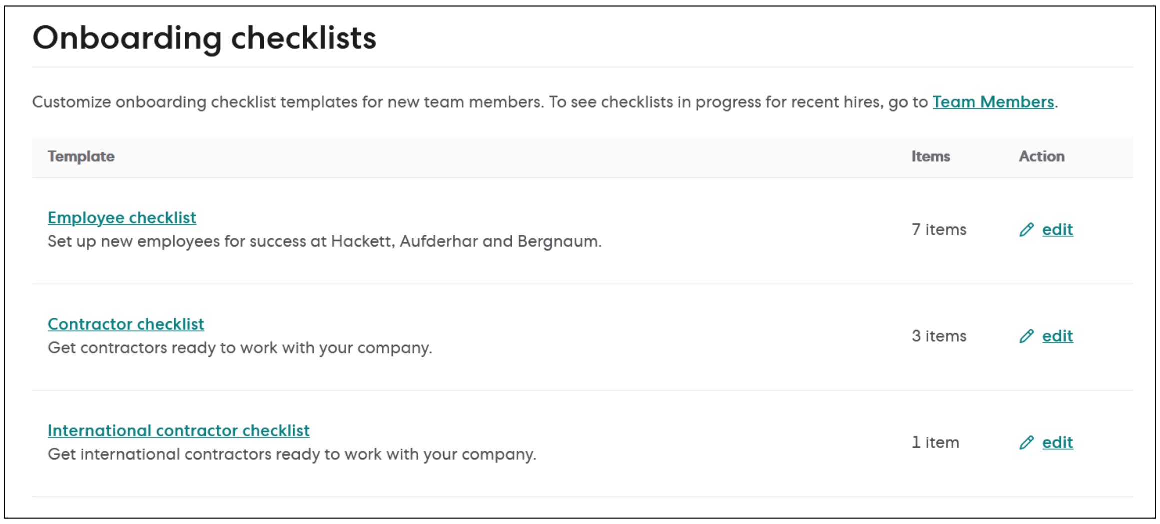
However, similar to user management, to implement a comprehensive solution for partner management, it is crucial to analyze the Network Manager's approach by comparing them to other software solutions.

For this analysis, I chose partner management software solutions to support businesses in building and managing successful partner ecosystems, considering these benchmarks: partner onboarding, partner collaboration, partner performance tracking, and service-linking. The chosen management solutions were:

- **Salesforce Partner Management** [61]**:** A specialized platform for managing partner relationships, encompassing deal registration, lead sharing, and co-marketing initiatives.
- **Zoho Partner Portal** [74]**:** A platform designed for onboarding and managing partners, fostering collaboration and communication through a dedicated portal.
- **PartnerStack** [50]**:** A platform that streamlines the management and automation of partner programs, covering recruitment, onboarding, and compensation.
- **Impact** [41] **:** A platform for managing and tracking affiliate marketing programs, enabling campaign optimization and performance analysis.

An analysis of these software solutions compared to the Network Manager revealed the following:

Partner Onboarding is currently not implemented in the Network Manager. This feature would allow partners to create their own accounts and begin using the system without waiting for an administrator to approve their requests. While the aforementioned platforms offer different onboarding approaches, the Network Manager could implement standard onboarding features, such as a task list for partners to complete and open-ended feedback collection through polls. Figure 3.1, available in Gusto's, a solution from the previous section, is an example of this implementation.



Onboarding checklists

Customize onboarding checklist templates for new team members. To see checklists in progress for recent hires, go to Team Members.

Template	Items	Action
Employee checklist Set up new employees for success at Hackett, Aufderhar and Bergnaum.	7 items	edit
Contractor checklist Get contractors ready to work with your company.	3 items	edit
International contractor checklist Get international contractors ready to work with your company.	1 item	edit

Figure 3.1: Gusto's customizable onboarding tasks [36].

Partner collaboration is a key feature of all four platforms. However, the Network Manager should not implement all partner collaboration features available on other platforms, as this is not its main goal. Instead, the Network Manager should focus on implementing the ability for partners to share information with each other. Partners should be able to decide which other partners and/or partner groups can access their information, including shipping and invoice documentation.

Partner performance tracking is a complex feature implemented in all four platforms. However, the Network Manager should take a more straightforward, focused approach to tracking partner performance. The Network Manager can provide valuable insights into partner performance without overwhelming partners with too much data by focusing on trends such as user acquisition, application usage, revenue, and customer satisfaction. Partners should be able to control which data is shared with their partner groups.

The Partner should be able to see it consumptions by tracking and logging the usage of various services, items, or resources offered by LoopOS to the partner, its Payments for availing LoopOS services or any other payment-related activities. This could include subscription fees, usage charges, service fees, or any other financial transactions between the partner and LoopOS. The Partner should also be able to add and see any relevant documents or files associated with a partner within the LoopOS platform. These documents could include contracts, agreements, legal documents, certificates, or other files specific to the partner.

Regarding Service Linking, the Network Manager also provides a unique ability to provide services from various areas and apply them to available applications immediately. This makes it a powerful tool for businesses that need to rapidly deploy and manage various services. Once a service has been provisioned, it can be immediately applied to any application in the LoopOS ecosystem. It provides a comprehensive framework for service linking, which includes Auth, Email, Shipping, Invoice, Incoming and Outgoing Payments. This automation eliminates the need for manual integration, however, services can be added per request of the partners. For example, the CTT - Correios de Portugal [22] authentication service was added to the available authentication services as per the request of this company.

Finally, although the aforementioned platform offers a complete list of possibilities within these services, the Network Manager will expand its service-linking capabilities with more services in the future.

3.3 App Management

App Management is a critical component of the Network Manager, essential for the seamless operation of the LoopOS technological ecosystem, enabling a robust and reliable environment for LoopOS applications.

For this analysis, I chose cloud-based software platforms to deploy and manage applications, considering these benchmarks: centralized management, deployment, monitoring, scaling, and performance. The chosen platforms were:

- **Heroku [39]:** A cloud-based platform tailored for building, deploying, and managing apps, offering a broad range of language support and flexible scaling options.
- **Azure App Service [6]:** Microsoft's platform for building and managing web apps and services, providing integrated development tools and automated scaling capabilities.
- **Google App Engine [13]:** Google's platform for building and running web apps on Google's infrastructure, leveraging automatic scaling and load balancing.
- **Amazon Web Services Elastic Beanstalk [62]:** A platform designed for deploying and managing web apps on AWS, simplifying resource provisioning and scaling.

An analysis of these software solutions compared to the Network Manager revealed the following:

The Network Manager employs Rancher and Kubernetes for deploying and overseeing its applications, ensuring high performance, dynamic scaling, and automated deployment, aligning with the capabilities of other platforms. Nevertheless, in terms of monitoring, the Network Manager currently lacks more advanced monitoring features.

The key distinction lies in the primary objective of the Network Manager's application deployment compared to these platforms. Instead of facilitating user-desired applications, the Network Manager focuses on deploying predefined applications (LoopOS Handling, LoopOS Validation, etc.). This deployment is tailored to partners' specific needs, emphasizing inter-communication and security among these applications.

In conclusion, the Network Manager stands out as a robust system, leveraging Rancher and Kubernetes to offer application management similar to other platforms. While there is room for enhancement in monitoring features, the Network Manager's unique strength lies in its specialized approach to application deployment. It prioritizes deploying pre-defined applications that are custom-tailored to meet partners' specific needs.

3.4 Conclusion

The Network Manager distinguishes itself as a comprehensive and specialized solution within the LoopOS ecosystem. While other platforms excel in specific aspects of user management, partner management, and app management, none fully encompass the unique combination of features provided by the Network Manager. Its granular user management capabilities, focused partner collaboration features, and specialized approach to deploying predefined applications set it apart. Table 3.1 summarizes this unique combination.

Table 3.1: State of the Art conclusion comparison.

Platform	**User Management**	**Partner Management**	**App Management**
Network Manager	✔	✔	✔
BambooHR	✔	✖	✖
Gusti	✔	✖	✖
15Five	✔	✖	✖
Salesforce Partner Management	✖	✔	✖
Zoho Partner Portal	✖	✔	✖
Partner-Stack	✖	✔	✖
Heroku	✖	✖	✔
Azure App Service	✖	✖	✔
Google App Engine	✖	✖	✔
Amazon Web Services Elastic Beanstalk	✖	✖	✔

Analyzing the Network Manager's capabilities compared to relevant software solutions has provided valuable insights into its strengths and areas for potential improvement.

The Network Manager provides a granular and flexible approach to access control, permissions, and settings using Users, User Groups, and App Scopes in user management. While it outperforms HR management software solutions in certain aspects, it can benefit from incorporating access control in its pages, user feedback collection, and user analytics to enhance its overall user management capabilities.

Partner management within the Network Manager also demonstrates strong capabilities, particularly in service-linking, where it offers a comprehensive framework for linking various services to applications seamlessly. However, further enhancing its partner features, such as seeing its relevant documents, partner consumption, adding partner onboarding, and simplified partner performance tracking, would further elevate the Network Manager's effectiveness in managing its extensive partner ecosystem.

Regarding app management, although the Network Manager's unique strength lies in its specialized approach to deploying predefined applications tailored to the specific needs of partners, its monitoring features should be enhanced. For example, receiving resource alerts and seeing real-time CPU and memory usage per App Instance.

In conclusion, the Network Manager is a robust and flexible system, offering a distinct set of capabilities contributing to the LoopOS ecosystem's seamless operation. However, certain areas demand improvement, particularly in Partner Onboarding, Application Monitoring, and overall User Experience. While not explicitly addressed in a dedicated section, the inability of Partners and Users to intuitively navigate the Network Manager renders the provision of new features largely ineffective. The quick development of the Network Manager through the use of AVO came at the expense of usability. This approach may have sufficed during the initial developer-centric phase, but as Partner and User adoption increases, the application must prioritize user-friendliness, intuitiveness, and ease of use.

Chapter 4 - Requirements and Risks will address the identified areas for improvement. It will define the actors involved, translate the necessary improvements into user stories and requirements, and prioritize them based on urgency and feasibility within the scope and time frame of this thesis.

Chapter 4

Requirements & Risk Analysis

Unlike the other LoopOS applications, only one instance of the Network Manager runs at any given time. This instance is responsible for coordinating all the other instances of different applications. This uniqueness requires careful planning of new releases and almost complete backward compatibility. in the first semester of this thesis occured an initial elicitation of actors, user stories, and requirements that was split into two cycles - version one (v1) & version two (v2) - encompassing planning, developing, and testing. However, due to the nature of the Agile methodology implemented, the project's objectives became more clear during the second semester of this thesis. Actors, User Stories, and Functional Requirements identified in the first semester of this thesis will be presented in the Appendix to minimize verbosity while Risks and Non-Functional Requirements have been completely changed.

It should be noted that some requirements, such as creating Partner Documents, Partner Consumptions, and Partner Payments, could not be met. Other, such as Templates and Self-Onboarding, were achieved by creating the LoopOS Onboarding Application and the creation of a Boilerplate Script.

As such, I decided it was necessary to create new Functional and Non-Functional Requirements, with new ways to be tested, new User Stories and new Risks. I also divided these into two: LoopOS Onboarding and LoopOS UI, since these were the two main focuses of development.

Starting with LoopOS Onboarding, its primary objective was to facilitate the onboarding of partners, encompassing individual partners with no established connections to other companies, who would be presented with LoopOS's default settings and companies onboarding to use the services of a company they were already collaborating with. A real-world example of this application was CTT - Correios de Portugal [22], who sought to employ LoopOS to manage its packages. To accomplish this, they aimed to onboard partners who would employ their transportation service (e.g., clothing stores handling returns through CTT) by having them complete specific settings, such as the address of the comapny the product would be returned to.

As the Onboarding App development neared completion, the Network Manager project took a new direction, prioritizing a full-scale User Experience (UI) overhaul.

This shift prompted the creation of a Rails Engine (LoopOS UI) to manage the front-end aspects of not only the Network Manager but also all LoopOS applications. This Engine centralizes common styles, scripts, and assets, thus becoming a single source of truth in front-end development and allowing individual apps to focus on their unique components. Each app instantiates this Engine, a central hub for accessing any required component by requesting its corresponding builder. For example, the Network Manager can swiftly obtain a sidebar using the provided code snippet in Listing 4.1. Here, *top_menu* and *bottom_menu* are arrays of hashes containing sidebar items and their associated subitems:

```
<%= render LooposUi::SidebarLayoutComponent.new do |
   component| %>
  <% component.top_sidebar(menu: top_menu) %>
  <% component.bottom_sidebar(menu: bottom_menu) %>
<% end %>
```

Listing 4.1: Network Manager Sidebar using LoopOS UI sidebar builder.

Any changes to the overall UI can be implemented directly in LoopOS UI, ensuring consistency across all applications. This streamlined approach is enabled by adopting Continuous Integration/Continuous Delivery (CI/CD) practices, which automatically propagate code updates from the Git repository into the Engines within each app.

Hence, the subsequent sections will detail the Actors, User Stories, and Functional and Non-Functional Requirements implemented, Restrictions and Risks for both the LoopOS Onboarding and LoopOS UI.

4.1 Actors

The following are the actors identified for the LoopOS Onboarding and LoopOS UI.

❙ LoopOS Onboarding

- ❖ **Partner Group:** A company which will manage its Partner through LoopOS.
- ❖ **Partner:** A company that will use LoopOS to manage its returns using the services belonging to its affiliated PartnerGroup.

❙ LoopOS UI

- ❖ **App Developer:** A developer that will use LoopOS UI to implement layouts and components in their respective applications.

Similar to the following sections, the actors elicited in the intermediate defense are present in the appendix.

4.2 User Stories

Understanding the motivations and needs of various actors is crucial for the successful development of any project. User stories thus provide a valuable framework for capturing requirements and understanding and communicating the intended outcomes.

Based on the actors mentioned in the previous section, the following user stories will be structured: "**As a** [**actor**], I [**want to**], [**so that**]" [17]. As before, the user stories planned for the intermediate defense are split into two versions (v1 & v2) and are in the appendix to demonstrate how the project and the thesis evolved. The following will present the User Stories for the applications I developed during this thesis:

❙ LoopOS Onboarding

- **As a** Partner Group, **I want to** customize the Onboarding landing page with my brand's logo, colors, and styles, **so that** I can provide a branded and engaging onboarding experience for my Partners.
- **As a** Partner Group **I want to** define which settings I want my Partners to fill in during its Onboarding process, **so that** I can streamline the onboarding experience and ensure that essential information is captured,
- **As a** Partner Group, **I want to** have my services available in the applications by Partners create, **so that** they can deploy their business logic with my services

- **As a** Partner, **I want to** have the option to access the Onboarding app in multiple languages, including English and Portuguese, **so that** I can my experience can be tailored to my preferences.
- **As a** Partner, **I want to** register on the Onboarding platform by providing necessary information such as my brand's name, logo, and user-specific details, **so that** I can have my own identity within the system.
- **As a** Partner, **I want to** select from a list of tailored boilerplates representing specific business logic, **so that** I can ensure the apps chosen align with my brand's requirements.
- **As a** Partner, **I want to** be informed about the progress of app creation through a dedicated wait page, **so that** I know my environment is being created.

- **As a** Partner, **I want to** receive an email notification upon successfully completing the onboarding process, **so that** I can have a seamless transition to using the newly created apps.
- **As a** Partner, **I want to** access a list of newly created apps after completing the onboarding process, **so that** I can easily navigate and use my new applications.

LoopOS UI

- **As an** App Developer, **I want to** seamlessly integrate the LoopOS UI Engine into my application, **so that** I can leverage its features to enhance the user interface of my application without extensive development effort.
- **As an** App Developer, **I want to** support a progressive implementation approach for rendering components, **so that** I can gradually integrate LoopOS UI into my application, making the adoption process more flexible and manageable.
- **As an** App Developer, **I want to** dynamically implement sidebars using LoopOS UI, **so that** I can customize the layout and navigation of my application based on specific requirements and user preferences.
- **As an** App Developer, **I want to** dynamically implement the header of show pages using LoopOS UI, **so that** I can create a consistent and aesthetically pleasing user interface for displaying the header of individual pages.
- **As an** App Developer, **I want to** dynamically implement the content of show pages using LoopOS UI, **so that** I can have flexibility in presenting and arranging content on individual pages.
- **As an** App Developer, **I want to** dynamically implement the header of index pages using LoopOS UI, **so that** I can ensure a uniform and engaging design across pages displaying lists or collections.
- **As an** App Developer, **I want to** dynamically implement the content of index pages using LoopOS UI, **so that** I can efficiently design and structure the content presentation on pages displaying lists or collections.

4.3 Functional Requirements

The literature on requirements has many definitions of functional requirements or quality attributes. This thesis will focus on the definition of “Software Engineering” by Software Engineer Professor Ian Sommerville, which states: “statements of the services that the system must provide or are descriptions of how some computations must be carried out.” [63].

Additionally, prioritization is crucial regarding requirements, so when it comes time to implement them, it is essential to understand which ones take precedence. Prioritization helps ensure that the most critical requirements are addressed first, allowing the development to focus the efforts and resources effectively. The MoSCoW method based on the book “DSDM, Dynamic Systems Development Method” by Jennifer Stapleton [64] will be used to accomplish this.

MoSCoW is a simple method of prioritization that can be used to divide the requirements into four categories:

- **Must:** Non-negotiable needs for the project. Mandatory.
- **Should:** Essential to the product, project, or release, but they are not vital. If left out, the product or project still functions. However, the initiatives may add significant value.
- **Could:** Nice to have, but not necessary to the core function of the product.
- **Will not have (this time):** Not a priority for the project’s current timeframe.

The requirements are split into the features that each actor can use. A unique ID identifies each requirement, its actor, and its version. The following are the fundamental functional requirements identified during the requirements elicitation phase, which in turn shaped how these features were planned and developed:

LoopOS Onboarding

PartnerGroup Requirements:

FR1 (Should Have): The PartnerGroup should be able to customize the Onboarding landing page with their brand’s logo, colors, and styles.

FR2 (Must Have): The PartnerGroup should be able to define which settings they want their Partners to fill in during the onboarding process.

FR3 (Must Have): The PartnerGroup should be able to provide their Partners with their services created in LoopOS.

Partner Requirements:

FR4 (Must Have): The Onboarding app should be accessible in multiple languages, including English and Portuguese.

FR5 (Must Have): Partners should be able to register on the Onboarding platform by providing necessary information such as their brand's name, logo, and user-specific details.

FR6 (Should Have): Partners should be able to select from a list of tailored boilerplates representing specific business logic, generating from App Instances to LoopOS Flows.

FR7 (Could Have): Partners should be informed about the progress of app creation through a dedicated wait page.

FR8 (Must Have): Partners should receive an email notification upon successfully completing the onboarding process.

FR9 (Should Have): Partners should be able to access a list of newly created apps after completing the onboarding process.

LoopOS UI

App Developer Requirements:

FR10 (Must Have): Enable App Developers to seamlessly integrate the LoopOS UI Engine into their applications.

FR11 (Must Have): Support a progressive implementation approach for rendering components.

FR12 (Must Have): Provide App Developers with the ability to dynamically implement sidebars using LoopOS UI.

FR13 (Should Have): Provide App Developers with the ability to dynamically implement the header of show pages using LoopOS UI.

FR14 (Could Have): Provide App Developers with the ability to dynamically implement the content of show pages using LoopOS UI.

FR15 (Could Have): Provide App Developers with the ability to dynamically implement the header of index pages using LoopOS UI.

FR16 (Could Have): Provide App Developers with the ability to dynamically implement the content of index pages using LoopOS UI.

Similar to the previous sections, to demonstrate how the project and the thesis evolved, the requirements planned for the intermediate defense are present in the appendix.

4.4 Non-Functional Requirements

The literature on requirements has many definitions of non-functional requirements or quality attributes. As before, this thesis will focus on the definition in "Software Engineering" (2016) by Professor Ian Sommerville, which states: Non-Functional Requirements "often constrain the system being developed and the development process used. These might be product requirements, organizational requirements, or external requirements. They often relate to the system's emergent properties and apply to the system as a whole" [63].

This section will outline the non-functional requirements identified for the LoopOS UI and LoopOS Onboarding, following the structure presented in Figure 4.1:

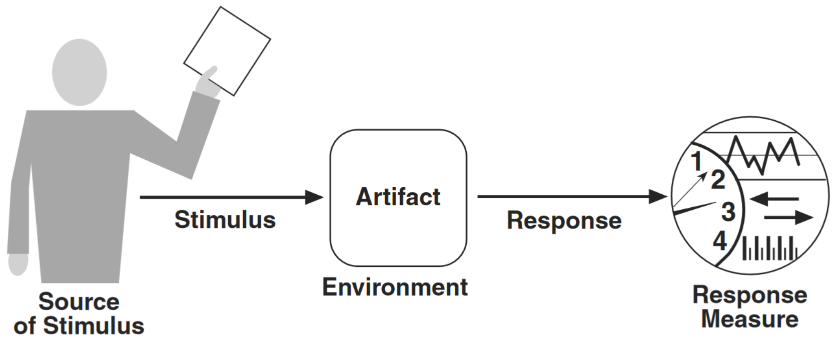


Figure 4.1: Non-Functional Requirements Scenario Diagram [8].

- **Source of stimulus:** This is some entity (a human, a computer system, or any other actuator) that generated the stimulus.
- **Stimulus:** The stimulus is a condition that needs to be considered when it arrives at a system.
- **Environment:** The stimulus occurs within certain conditions. The system may be overloaded or running when the stimulus occurs, or another condition may be true.
- **Artifact:** Some artifact is stimulated. This may be the whole system or some pieces of it.
- **Response:** The response is the activity undertaken after the arrival of the stimulus.
- **Response measure:** When the response occurs, it should be measurable in some fashion so that the requirement can be tested.

This section, will only present the non-functional requirements identified as the highest priority and crucial for adequately implementing the desired system within the scope of this Thesis.

While some non-functional requirements may overlap, similar to the previous sections, I will divide them into LoopOS UI and LoopOS Onboarding due to being two separate pieces of software with distinct testing approaches.

LoopOS Onboarding

Availability: the state in which a system remains free from failure, ensuring its users, be it humans or other systems, experience uninterrupted functionality and avoid the associated consequences.

- The LoopOS Onboarding should be resilient to failures, with measures in place to handle errors gracefully and prevent data loss or corruption.

Table 4.1: Availability - Resilience to Failures.

Source of stimulus	System event (e.g. software error)
Stimulus	A failure occurs within the LoopOS Onboarding.
Environment	The system is running and serving users.
Artifact	The LoopOS Onboarding system components (software).
Response	The LoopOS Onboarding should be resilient to failures, with measures in place to handle errors gracefully and prevent data loss or corruption.
Response measure	Application downtime.

Akin to other LoopOS applications, LoopOS Onboarding is deployed using Rancher. This orchestration platform employs Prometheus and Grafana to monitor its clusters [54].

- **Prometheus:**
  An open-source monitoring platform specifically designed for containerized workloads. Records real-time metrics in a time series database, enabling flexible queries and real-time alerting. It is commonly used for monitoring Kubernetes clusters and offers powerful features for managing and analyzing containerized environments [52].
- **Grafana:**
  Allows the creation of observability dashboards using various data sources, including metrics from Prometheus. It provides a flexible and customizable interface for visualizing and analyzing data, making it suitable for monitoring and tracking user and application behavior, including error types and frequencies [34].

DigitalOcean [27], a cloud computing platform that hosts LoopOS, recommends for critical workloads that require high availability to have an uptime of 99.95% [26]. This translates to allowed downtime/unavailability periods with an average of 22 minutes per month. After completing the LoopOS Onboarding, this requirement will be tested passively by using Prometheus for the following months until the delivery of this thesis.

The success case for this requirement will be an average monthly downtime equal to or lower than 22 minutes.

Performance: the timely response of a system to various events arising from multiple sources, where resource consumption fulfills service requests amidst concurrent servicing.

- The LoopOS Onboarding should have fast response times, providing a seamless user experience and minimize delays in loading pages or executing tasks.

Table 4.2: Performance - Fast Response Times.

Source of stimulus	User
Stimulus	The user interacts with the LoopOS Onboarding, such as loading pages or executing tasks.
Environment	The system may have varying levels of user concurrency and data loads.
Artifact	The LoopOS Onboarding interface and underlying system components (e.g., api calls).
Response	The LoopOS Onboarding should have fast response times, providing a seamless user experience and minimizing delays in loading pages or executing tasks.
Response measure	Page load time

The LoopOS Onboarding will depend highly on Application Programming Interface (API) calls to obtain and transmit the information required from and to the Network Manager. These API calls can be highly costly if not performed correctly for user wait time and the Network Manager resource consumption. Hence, the LoopOS Onboarding should implement measures such as effective caching strategies to reduce the load on the underlying infrastructure and improve response times.

The average page load time in 2023 is 2.5 seconds on desktop [67] [49] [60]. However, this number can vary widely depending on the type of website, the user's location, and the internet connection speed.

Since Rancher employs optimized load balancing, these tests can be performed without additional traffic (load) on the system.

To assess the loading times of the web pages within the platform, the Development Tools of the Firefox browser can be employed, effectively capturing the overall loading duration of each page. This testing will be conducted on all LoopOS Onboarding pages post-completion, encompassing 20 measurements per page. The final outcome will be derived by averaging these measurements.

The success case for this requirement will be an average page load equal to or lower than 2.5 seconds.

LoopOS UI

Performance: the timely response of a system to various events arising from multiple sources, where resource consumption fulfills service requests amidst concurrent servicing.

- The LoopOS UI should have fast response times, providing a seamless user experience and minimize delays in loading pages or executing tasks.

Table 4.3: Performance - Fast Response Times.

Source of stimulus	User
Stimulus	The user interacts with the LoopOS UI, such as loading pages or executing tasks.
Environment	The system may have varying levels of user concurrency and data loads.
Artifact	The LoopOS UI interface and underlying system components (e.g., components and assets).
Response	The LoopOS UI should have fast response times, providing a seamless user experience and minimizing delays in loading pages or executing tasks.
Response measure	Page load time

LoopOS UI will change how the application is deployed to handle components, but it should not significantly add add more load time to the pages.

The testing of this requirement will follow a similar approach of LoopOS Onboarding. The testing will be conducted on all implemented Network Manager pages, encompassing 20 measurements per page. The final outcome will be derived by averaging these measurements.

The success case for this requirement will be an average page load equal to or lower than 2.5 seconds.

Security: encompasses the system's capacity to withstand unauthorized usage while ensuring uninterrupted service delivery to authorized users.

- The LoopOS UI should have role-based access controls.

Table 4.4: Security - Role-Based Access Controls.

Source of stimulus	User (administrator)
Stimulus	The administrator configures access permissions for different user roles within the LoopOS UI system.
Environment	The system is running, and administrative actions are being performed.
Artifact	The access control mechanisms within theLoopOS UI system.
Response	The LoopOS UI should have role-based access controls, allowing administrators to define fine-grained permissions for different roles.
Response measure	User Scopes.

LoopOS UI should add the ability to control which pages the user can access depending on its role. Users should be able to access its partner profile page, for example.

One possible way to implement this requirement is by using Pundit [53]. A minimal authorization through object-oriented design and pure Ruby classes, that would allow LoopOS UI to verify whether a user is authorized to perform specific actions, such as creating, editing, or deleting resources, and restrict user access to specific views or components based on their permissions.

The success case for this requirement will be that users can access the proper pages and perform authorized actions based on their roles, enhancing the overall security and usability of the LoopOS UI.

4.5 Restrictions

The Loop Co. placed certain restrictions upon my work to follow company policies and better integrate with the system already in place.

This means that all my work will be conditioned to the use of Ruby on Rails and the company's desired gems (libraries like Rubocop [9] to enforce consistent code style and detect potential issues) as well as the use of Clickup for management, Microsoft Teams for communication and Visual Studio Code due to also having extensions used by the company.

4.6 Risk Analysis

As stated in "Risk analysis and management: a vital key to effective project management" [47]: "While we can never predict the future with certainty, we can apply a simple and streamlined risk management process to predict the uncertainties in the projects and minimize the occurrence or impact of these uncertainties. (...) This improves the chance of successful project completion and reduces the consequences of those risks."

In this section, by applying robust risk management processes, we delve into the fundamental aspects of risk management, focusing on Risk Identification (Subsection 4.6.1), Risk Analysis (Subsection 4.6.2) and Risk Planning (Subsection 4.6.3) as can be seen in figure 4.2.

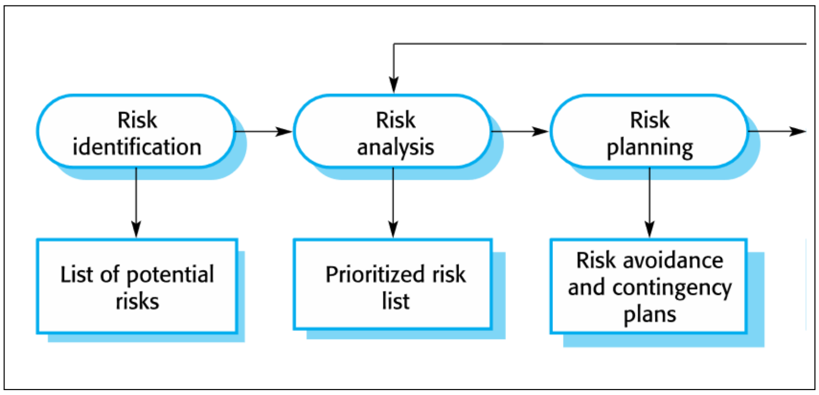


Figure 4.2: Risk Management Process [63].

4.6.1 Risk Identification

The Risk Identification present in this thesis will focus on the paper "Taxonomy-Based Risk Identification" [11] since most of the literature uses it as a basis for risk management models.

The paper groups different risk sources into diverse categories and provides a questionnaire called TBQ (Taxonomy-Based Questionnaire) that allows for performing a systematic identification process. This involves breaking down the system, being analyzed into its parts, and examining each piece for potential risks. Presenting this questionnaire to multiple project members facilitates the identification of risks.

However, the full implementation of this questionnaire would be out of the scope of this thesis. Hence, the risk identification will consider only the main aspects of the taxonomy defined by the authors:

- **Product Engineering:** The technical aspects of the work to be accomplished.
- **Development Environment:** The methods, procedures, and tools used to produce the product.
- **Program Constraints:** The contractual, organizational, and operational factors within which the software is developed but which are generally outside of the direct control of the local management.

To be easily identified in further sections, each risk presented will have a code that will follow the pattern: R - #risk_number:

[R-1] **Requirements:** Incomplete or unclear requirements leading to incorrect implementation or unsatisfied customer needs.

[R-2] **Requirements:** Infeasible requirements that cannot be implemented within the available resources or technology constraints.

[R-3] **Code and Unit Test:** Poor coding practices that make the code difficult to maintain or modify.

[R-4] **Development Process:** Insufficient product control leading to deviations from the original project scope.

[R-5] **Development System:** Lack of experience.

[R-6] **Work Environment:** Communication breakdowns leading to misunderstandings and missed deadlines.

[R-7] **Resources:** Tight schedule leading to potential quality issues or delays due to balancing the internship and university work.

[R-8] **Development Halt:** Company's need to allocate resources to other parts of LoopOS may lead to delays.

[R-9] **Changes in Planning:** Changes in the way of accomplishing objectives may lead to overall changes in the thesis (Objectives, State of the Art, Requirements, etc.).

4.6.2 Risk Analysis

Once risks are identified, it is possible to conduct a rigorous risk analysis to assess each risk's likelihood and potential impact. This analysis will allow us to make informed decisions about risk treatment strategies, considering factors such as cost-benefit.

This thesis follows the proposed metrics from “Risk analysis and management: a vital key to effective project management” [47], which classify risks based on their rating and probability. The rating of a risk reflects its potential impact on the project, while the probability reflects the likelihood of the risk occurring. There are three risk ratings or impact: C (Marginal), B (Moderate) and A (Critical). There are also four probability levels: low (0% < x < 30%), medium-low (30% x < 60%), medium-high (60% x < 80%) and high (80% x 100%). These metrics can be used to assess a project’s overall risk and prioritize risks for mitigation. Its implementation to the risks identified in the previous section can be observed in figure 4.3.

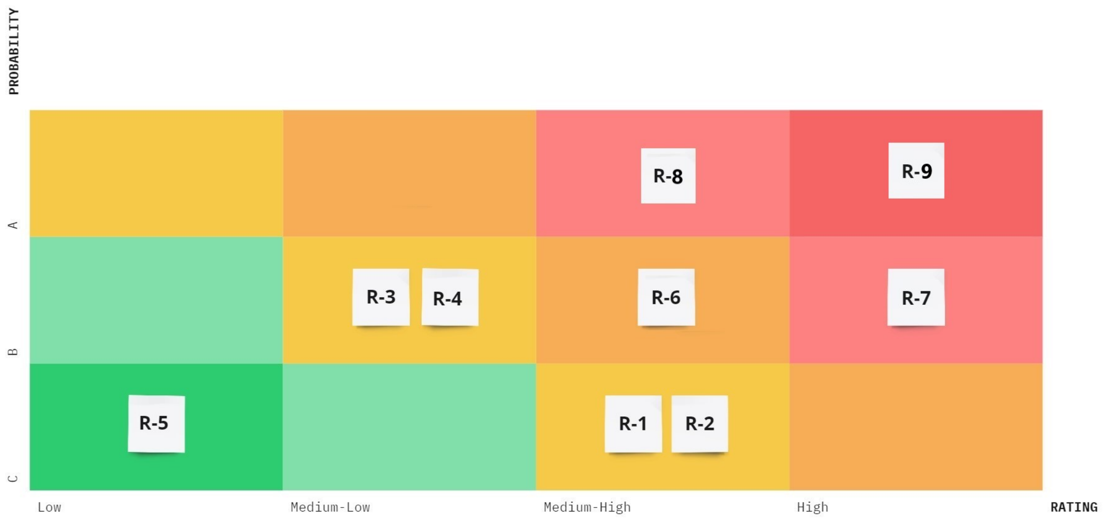


Figure 4.3: Risk Matrix [47].

4.6.3 Risk Planning

However, risk analysis alone is insufficient to safeguard against potential threats. Risk planning is crucial to implementing measures to reduce the probability and impact of identified risks. It is possible to minimize exposure to this project’s various risks by effectively mitigating risks.

The mitigation strategies for the risks identified in the first section of this chapter can be viewed in table 4.5.

Table 4.5: Mitigation strategies for the previous identified risks.

Risk ID	**Risk Mitigation Strategy**
R-1	Perform detailed requirements gathering and analysis, and ensure clear communication with stakeholders throughout the project.
R-2	Communicate if a requirement seems unlikely to be completed on time, to be reassigned or be given help.
R-3	Adopt good coding practices and conduct thorough unit testing to ensure code maintainability and ease of modification.
R-4	Establish a well-defined development process that includes robust project control measures.
R-5	Training on required tools, and leverage experienced team members to mentor.
R-6	Foster a culture of open and effective communication among team members, establish clear expectations and timelines for deliverables and use communication software or conduct in-person meetings.
R-7	Prioritize tasks based on their level of importance and urgency and promptly communicate when these deadlines seem infeasible.
R-8	Aid in the development of other project areas so the development may processed.
R-9	Maintain thorough documentation of the initial thesis focus and any subsequent modifications. This documentation serves as a reference point, ensuring clarity on its evolution and the reasons behind each adjustment.

Chapter 5

Framework

This chapter will start by outlining the team organization, followed by exploring the delivery lifecycle employed throughout the development. This chapter will conclude by presenting the scheduled and achieved timeline of the first semester and second semester of this thesis.

5.1 Team Organization

A well-structured and coordinated team is essential to execute the project and achieve its objectives successfully. This section overviews the team members involved in the LoopOS project and their respective roles and responsibilities.

- **Project Owner (PO)** - *Joana Ribeiro & Ricardo Morgado*:
  They bear the business responsibility for successful project implementation and are responsible for gathering user requirements, managing stakeholder expectations, and ensuring that the developed system aligns with the business goals.

- **Chief Technology Officer (CTO)** - *João Rodrigues*:
  Plays a crucial role in guiding the technical direction of the project by providing technical expertise and by leading the decision-making process related to the system's architecture, technology stack, and overall development strategy.

- **Product Manager (PM)** - *Marta Mercier*:
  Responsible for coordinating the activities of the development team, managing project timelines and deliverables, and ensuring effective communication among team members.

- **Backend Tech Lead** - *Daniel Gonçalves*:
  Leads the development and implementation of the backend infrastructure of the system, ensuring the delivery of high-quality, scalable, and robust software systems.

- **Frontend Tech Lead** - *Pedro Reis*:
  Leads the development of the user interface and the client-side functionality, ensuring a cohesive user experience and the successful integration of frontend and backend components.

- **Design Lead** - *Rafaela Silva*:
  Focuses on overseeing the design aspects of the project, being responsible for creating an intuitive and appealing user interface that aligns with user needs and business requirements, by also working closely with the frontend tech lead to ensure the design is implemented accurately and effectively.

- **Backend Developers:**
  Under the guidance of the Backend Tech Lead, they are Responsible for implementing server-side logic, database management, and integration with external systems to provide reliable and scalable backend solutions.

- **Frontend Developers:**
  Under the guidance of the Frontend Tech Lead, they are responsible for implementing the user interface, implementing responsive designs, and ensuring smooth user experiences in web applications.

- **Designers:**
  Under the guidance of the Design Lead, they are responsible for developing visually appealing, consistent, and intuitive user-friendly interfaces.

- **Intern:**
  As an intern in this company, my main responsibilities revolve around completing the objectives of my thesis by the development of the LoopOS Onboarding and LoopOS UI while also participating in the improvement and testing of other LoopOS applications, in accordance with the company's needs.

Collaboration and synergy among team members are vital to the project's success. Regular communication, feedback exchange, and shared decision-making enable the team to deliver a high-quality system that meets the requirements of all stakeholders.

5.2 Delivery Lifecycle & Technologies

Weekly sprint planning is a crucial step in the agile development process, ensuring that the team is focused on the right tasks and working collaboratively and efficiently. This section outlines each phase's phases, task states, weekly overview, and end-state conditions.

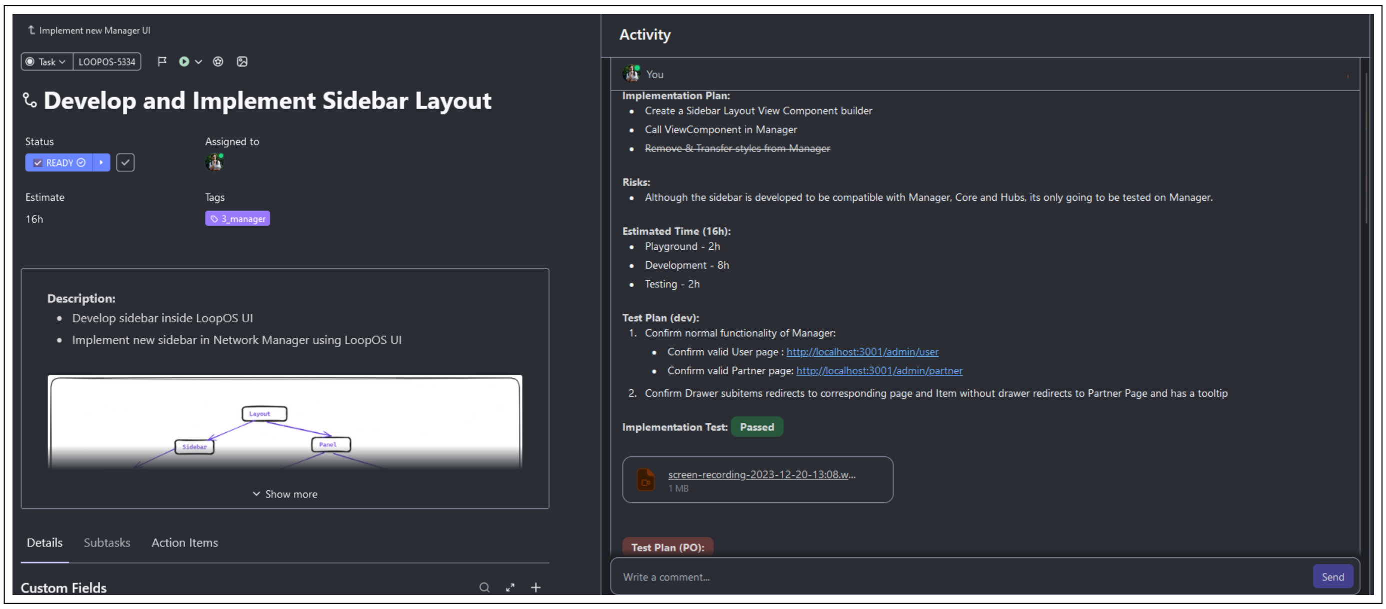
To facilitate this process, the company employed two essential tools: ClickUp [12], a comprehensive productivity platform used to manage and track tasks for each sprint, and GitLab[73], a robust software repository management platform that hosted each application repository while enabling the creation of Merge Requests. These facilitate the deployment of branches created for each task into the main branch of each repository.

It is important to acknowledge that the product lifecycle evolved throughout the thesis development process. Several modifications were implemented to enhance productivity and overall predictability of the product. One significant change involved forming cross-functional teams, which I was part of. This structural shift enabled the optimization of various phases, including the Planning phase. Team members now convene in person to review and refine each other's planning efforts collaboratively. Additionally, the Testing phase was improved by incorporating a "Team review" step, where team members gathered to review each other's tasks before they proceeded, enabling earlier detection of potential issues.

Thus, the current weekly sprint planning process is divided into five phases, encompassing all company roles. However, there may be slight variations in the execution of these phases. For instance, some senior Backend Developers may be responsible for deploying the merge requests to the main branch of the corresponding application repository if approved during the Execution phase. Hence, the following will detail these phases from my perspective:

1. **Backlog Definition:** This phase entails identifying and prioritizing the tasks. At the beginning of developing LoopOS Onboarding and LoopOS UI, I divided the work into manageable tasks and assigned them appropriate priorities after devising a plan for each application. These tasks were then split into each sprint.

2. **Planning:** This phase entails a detailed planning and estimation of all tasks assigned to that sprint, encompassing the creation of sub-tasks, defining an execution plan, identifying potential risks, and formulating two test plans: one for the developers who will conduct the 1QA testing and another for the Project Owners who will evaluate it upon its arrival at the Staging review. Since the Project Owners belong to the business team, their test plan requires less technical planning. Figure 5.1 illustrates the Sidebar Creation task for LoopOS UI with these elements.


Implement new Manager UI

Task | LOOPOS-5334

Develop and Implement Sidebar Layout

Status: READY

Assigned to

Estimate: 16h

Tags: 3_manager

Description:
- Develop sidebar inside LoopOS UI
- Implement new sidebar in Network Manager using LoopOS UI

Show more

Details | Subtasks | Action Items

Custom Fields

Activity

You

Implementation Plan:
- Create a Sidebar Layout View Component builder
- Call ViewComponent in Manager
- ~~Remove & Transfer styles from Manager~~

Risks:
- Although the sidebar is developed to be compatible with Manager, Core and Hubs, its only going to be tested on Manager.

Estimated Time (16h):
- Playground - 2h
- Development - 8h
- Testing - 2h

Test Plan (dev):
1. Confirm normal functionality of Manager:
   - Confirm valid User page : http://localhost:3001/admin/user
   - Confirm valid Partner page: http://localhost:3001/admin/partner
2. Confirm Drawer subitems redirects to corresponding page and Item without drawer redirects to Partner Page and has a tooltip

Implementation Test: Passed

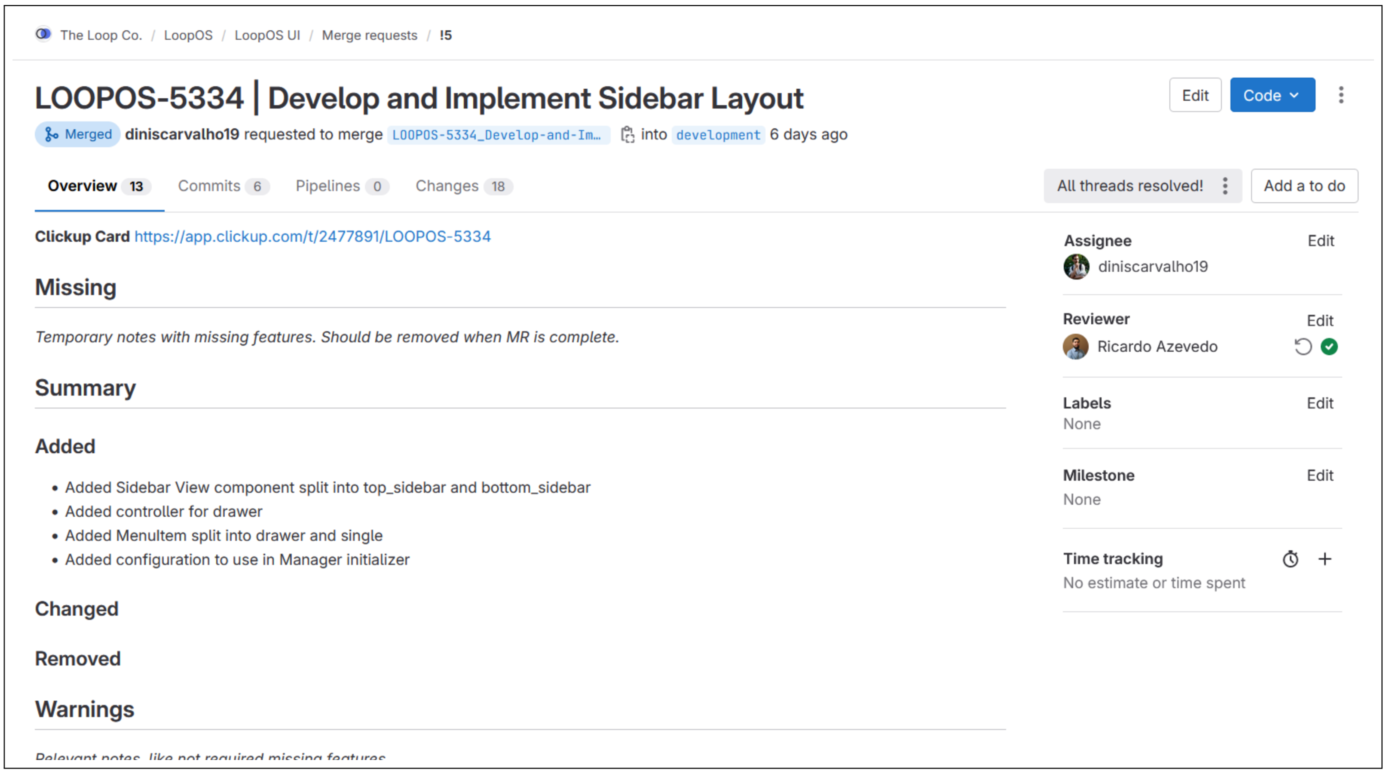
screen-recording-2023-12-20-13:08.w...
1 MB

Test Plan (PO):

Write a comment... Send

Figure 5.1: LoopOS UI - Sidebar Creation: Task in Clickup.

3. **Execution:** This phase involves implementing tasks, creating Merge Requests, reviewing team members' pending merges, and addressing feedback or issues/bugs that might appear. Figure 5.2 showcases a merge request for the Sidebar Creation task of LoopOS UI.



The Loop Co. / LoopOS / LoopOS UI / Merge requests / !5

LOOPOS-5334 | Develop and Implement Sidebar Layout

Merged diniscarvalho19 requested to merge `LOOPOS-5334_Develop-and-Im...` into `development` 6 days ago

Clickup Card https://app.clickup.com/t/2477891/LOOPOS-5334

Missing

Temporary notes with missing features. Should be removed when MR is complete.

Summary

Added

- Added Sidebar View component split into top_sidebar and bottom_sidebar
- Added controller for drawer
- Added MenuItem split into drawer and single
- Added configuration to use in Manager initializer

Changed

Removed

Warnings

Assignee: diniscarvalho19

Reviewer: Ricardo Azevedo

Labels: None

Milestone: None

Time tracking: No estimate or time spent

Figure 5.2: LoopOS UI - Sidebar Creation: Merge Request in Gitlab.

4. **Review:** This phase involves performing first-tier quality assurance (1QA) testing and updating the state to “Team Review” if testing passes or updating the state to ”Issue/Bug” with details if testing fails. In “Team Review” the task is tested again with the team.

5. **Closing:** This phase involves launching the final release versions, updating local development, double-checking release stability by the Project Oweners, and reverting back if issues arise. This phase is usually assigned to Project Managers and Project Owners.

Tasks are tracked through different states as they move through the sprint phases. Table 5.1 these states in the order they should be followed.

Table 5.1: Task states through sprint lifecycle.

State	Description
Backlog	In the backlog awaiting prioritization and definition
Planning	Being planned and estimated
Pending	Waiting for approval or feedback
To Do	Ready for implementation
In Progress	Actively being implemented
Issue/Bug	Task has encountered an issue or bug that needs to be resolved
Pending	Waiting for review or approval
MR Review	Merge request is awaiting review
MR Ready	Merge request is ready for merging
Deploying	Being deployed to the test environment
1st QA	In first-tier QA testing
1st QA Testing	Being tested in the first-tier QA environment
Team review	Awaiting review by the team
Ready	Ready for deployment to production
Ready for release	Ready to be handed over for release
Ready for production	Ready for deployment to production
Stg review	in staging environment for review
Ready for production	Ready for deployment to production

Additionally, each phase has a predefined time slot within the week, as can be seen in Figure 5.3.

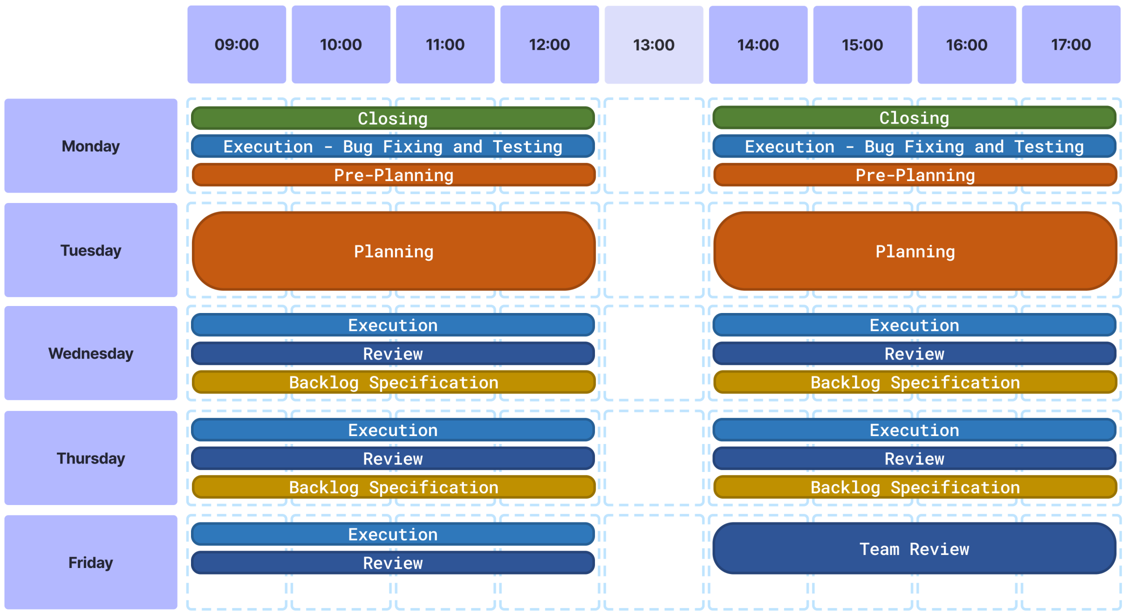


Figure 5.3: Weekly sprint timeline overview.

Finally, to determine the success of each phase in the spring, the following conditions must be met:

- **Planning:** All tasks are defined and in a "to-do" state.
- **Execution:** All tasks in the "1st QA" state.

- **Review:** Firstly, all tasks in the “team review” state. After “Team Review,” all tasks are in the “ready” state.
- **Closing:** All tasks in the “ready for production” state.

With the delivery lifecycle of the product outline, it is possible to advance to the Work Plan section.

5.3 Work Plan

This section presents the expected and actual plans for the first and second semesters, taking into account the methodology used, the delivery lifecycle, and the estimated time required to complete the work.

5.3.1 First Semester

The planning for the first semester of this thesis was as follows:

1. Survey of state of the art in terms of microservices and authentication/authorization solutions and multi-application system management.
2. Identification and analysis of project requirements and constraints.
3. Study of application development in Ruby on Rails.
4. Drafting of the development plan.
5. Monitoring the UI/UX team on the development of UI mockups for the modules to be developed.
6. Preparation for the intermediate defense.

Out of all the mentioned points, the only unfinished task was the collaboration with the UI/UX team.

There was also some development of the Network Manager. The focus was creating and updating models, databases, and visualization components using Avo [4], a Ruby on Rails gem. Improvements were made to the Partner Model and User Model, including introducing new AVO resources, controllers, and policies.

Additionally, the UserGroup Model was implemented alongside its corresponding avo components. Substantial progress was also made in developing the Cluster module and AppManagement module, which involved creating essential models such as AppScope, AppRelease, and AppInstance, all supported by avo resources, controllers, and policies. Collectively, these updates significantly improved the overall functionality of the Network Manager.

Finally, the combination of both charts for a better comparison can be viewed in Figure 5.4.

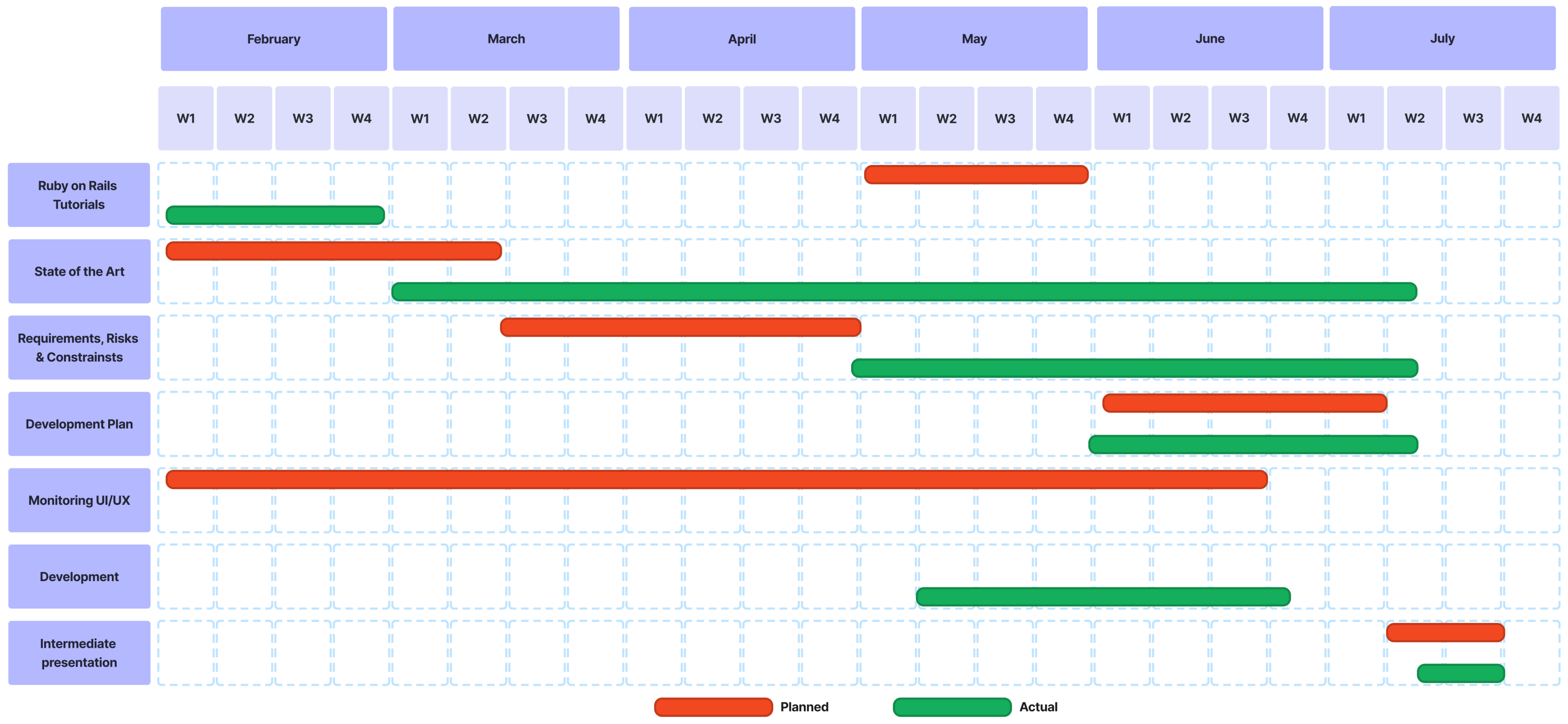


Figure 5.4: Gantt chart depicting the execution and planning combined of the first semester of this thesis.

5.3.2 Second Semester

The second semester was planned as follows:

1. Development: Continue working on the project's development phase, implementing the required features and functionalities.
2. Test Plan: Start creating a comprehensive test plan that outlines the testing strategies, methodologies, and test cases to be executed, testing functional and non-functional requirements.
3. Functionality Testing: Begin executing the functionality testing as per the test plan. Test each feature and functionality of the project to ensure they work as intended.
4. Final Report: Encompasses developments throughout the semester, including details of the implemented components and conducted tests and their corresponding results. Moreover, it incorporates the necessary modifications to the thesis based on the intermediate revisions.
5. Final Presentation: Showcase the project's development process, testing outcomes, and key findings.

However, as was described in previous sections, the objectives of this thesis persited through out its process, but the way to achieve them changed. Instead of spliting the work in two versions of the Network Manager, two applications were created: LoopOS Onboarding and LoopOS UI.

In the initial phase and between working on applications, I contributed to minor feature development, bug fixes, and testing across various applications within the LoopOS. For instance, I ensured that App Scopes were unique by name, made App Releases searchable by their version, generated PDF documents with shipment details and sent them by email, and performed tests on the Network Manager API. However, these contributions lie outside the scope of this thesis and will not be discussed further.

The planned and implemented Gantt chart of the second thesis semester can be seen in Figure 5.5.

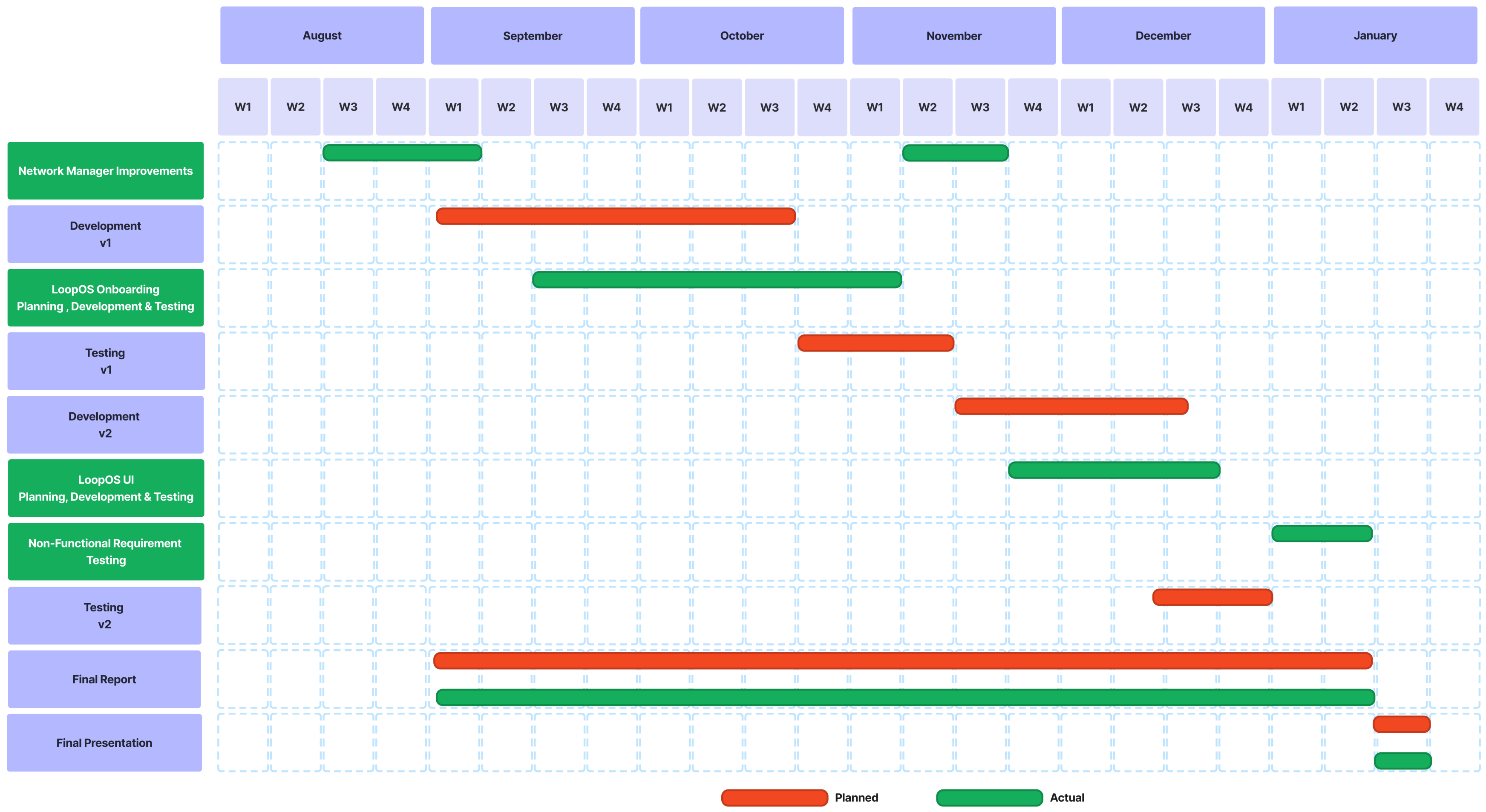


Figure 5.5: Gantt chart depicting the planning and execution of the second semester of this thesis.

As discussed in the previous section, each task underwent multiple testing phases, including those I conducted. For example, when developing endpoints for LoopOS Onboarding, I employed a Ruby gem called Rspec [59] to create automated tests, which will be detailed in Chapter 7 - Testing. Similarly, in the case of LoopOS UI, the View Components [69], the framework employed to create the new User Experience (UI), were also tested using its own methods.

Consequently, Functional Requirement testing was performed throughout the development process. Furthermore, as will be elaborated upon in the development of LoopOS Onboarding, an additional testing phase was conducted at the end.

It is worth noting that due to the company's focus on enhancing the UI of the Network Manager, Non-Functional Requirement (NFR) received less emphasis, and in return, the development of LoopOS UI took priority. However, NFR testing did occur in the first weeks of January, although not in the ideal depth, and can also be found in Chapter 7 - Testing.

Therefore, the following chapter will delve into the execution of LoopOS Onboarding and LoopOS UI and conduct a risk analysis post-development, addressing the risks identified in the previous chapter.

Chapter 6

Development

The second semester focused on implementing two prominent features: LoopOS Onboarding and LoopOS UI. The planning and development will be discussed in detail in the following sections.

6.1 LoopOS Onboarding

To enable partners to create LoopOS accounts and to automate the deployment and management of the LoopOS applications they wish to use within their system, LoopOS Onboarding was the first application development.

◆ Planning

The inaugural client to implement LoopOS Onboarding was CTT Returns, a novel solution offered by CTT. Its goal was to empower its partner brands and other users to effortlessly manage the return process using the LoopOS platform. Hence, there was an immediate need to create an aggregator entity higher in hierarchy to the Partner: the Partner Group.

The Partner Group's main function is representing organizations that work with multiple partners and desire to offer their services through LoopOS. Partner Groups can be linked to Partners, Users, Partner Services, and App Instances while also having settings, user settings schema in order to control the associated user's settings, and partner settings schema to control the associated partner settings.

In addition, in planning with the Design Team and then properly verified by the Project Owners, it was decided the LoopOS Onboarding would have four distinct pages that and would be tightly integrated with the Network Manager role in creating new partners and deploying and managing LoopOS applications. The entire onboarding process would implemented in Ruby on Rails to ensure smooth and efficient compatibility with other applications. Figure 6.1 presents the planned Partner journey on LoopOS Onboarding

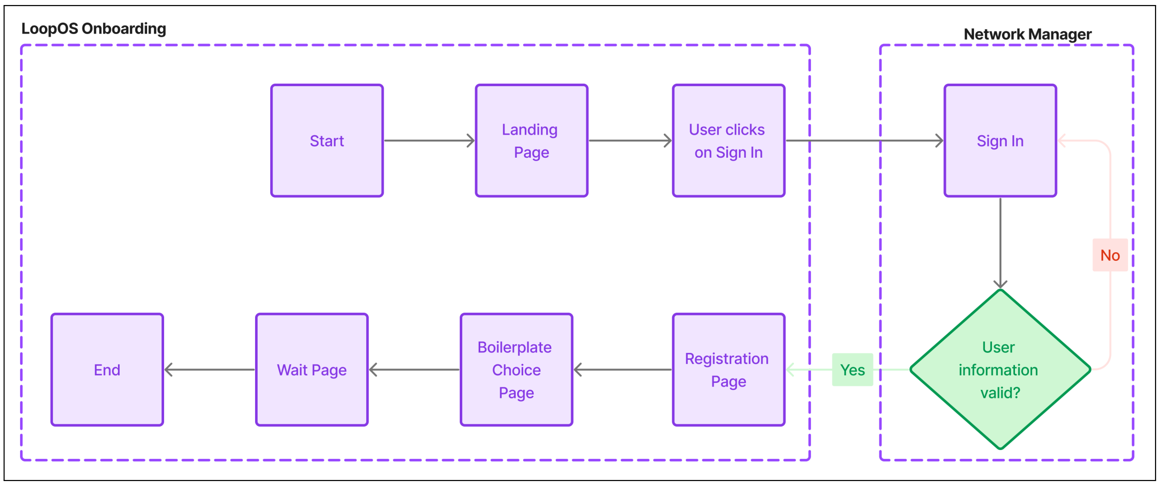


Figure 6.1: Partner planned flow in LoopOS Onboarding.

The following is a brief description of the four pages:

❚ Page 1 - Landing Page:

The Partner starts by creating an account in LoopOS. In the use case of CTT, there should be a Single Sign-On (SSO) so Partners can use their CTT account to link to a new account in the Network Manager. The Partner begins their journey on the LoopOS Onboarding Landing page, a central information hub and a gateway for signing in. This page should offer Partner Groups extensive customization options, allowing them to tailor the onboarding experience to their unique branding and preferences. The Partner then decides to sign in, and he is redirected to the Network Manager to handle authentication. If approved, the Partner is redirected to the Registration Page in LoopOS Onboarding.

❙ Page 2 - Registration Page:

The Partner Registration page serves as the heart of the onboarding process. The Partner provides partner-specific data and settings, including crucial information like the package return address. This information is essential for establishing a functional and personalized onboarding experience. Figures 6.2 and 6.3 are mockups of this page.

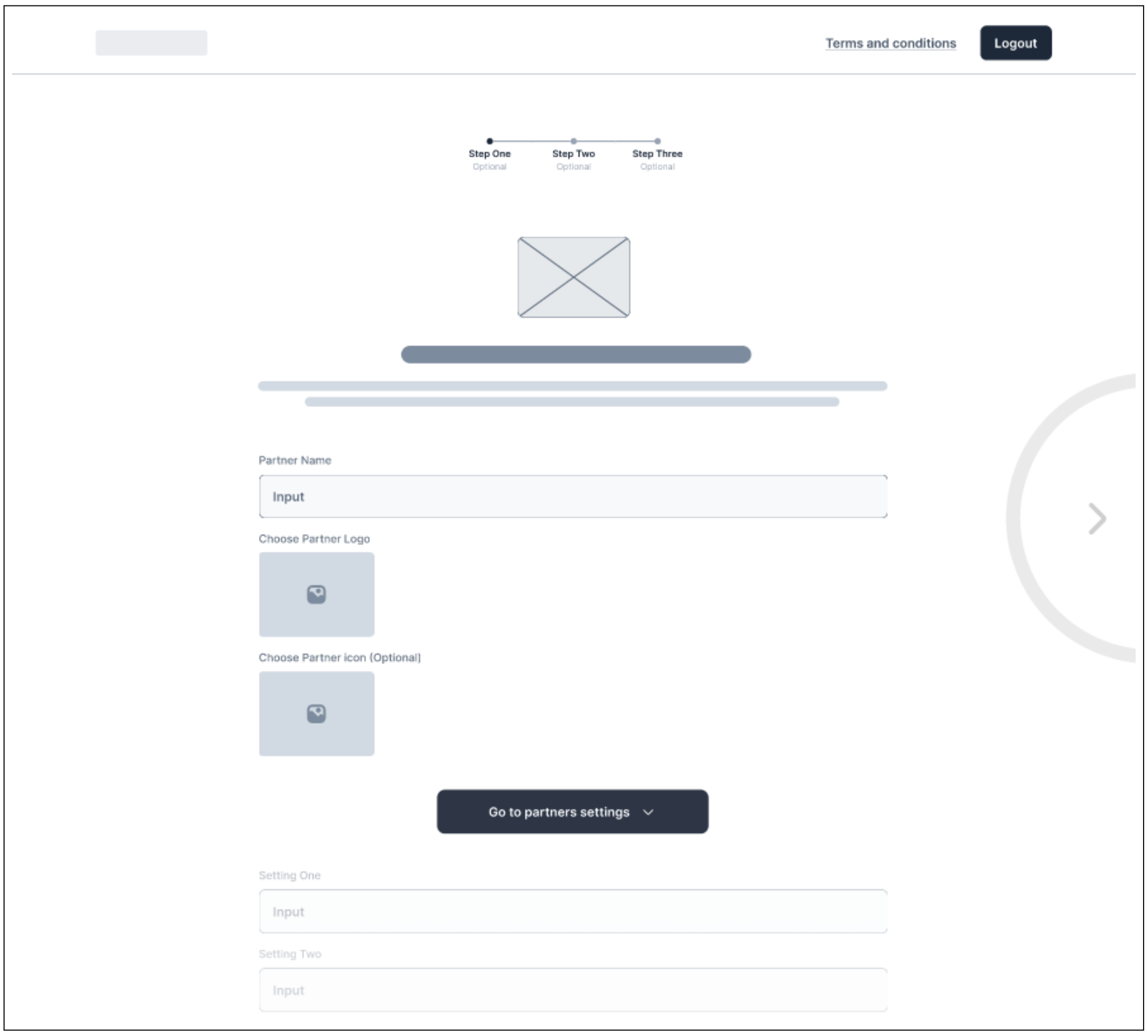


Figure 6.2: Mockup of the Partner Registration Page with default information that will be presented to all Partners independent of Partner Group

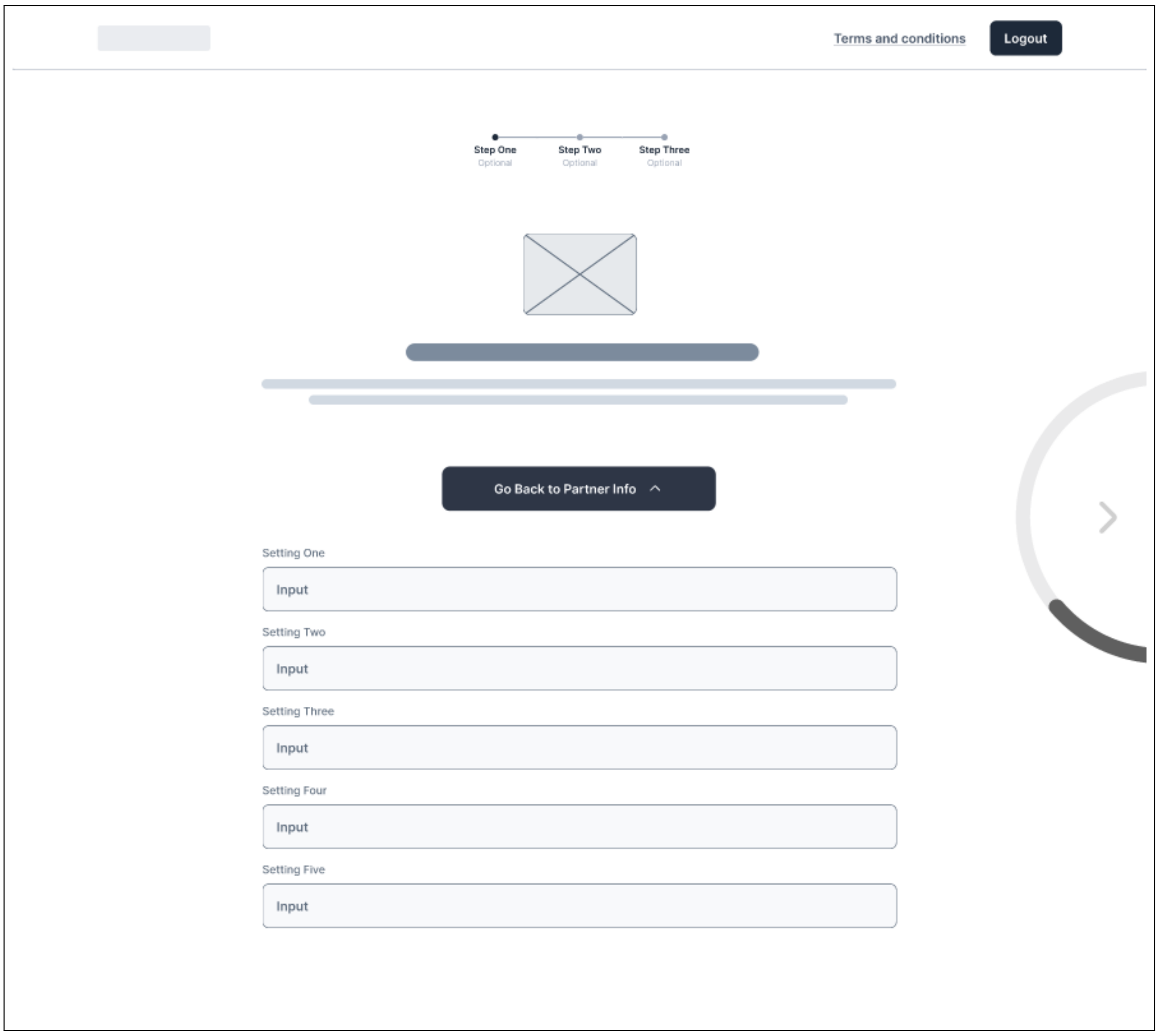


Figure 6.3: Mockup of the Partner Registration Page after filling default information and being presented with specific settings defined by the Partner Group to be filled.

Page 3 - Boilerplate Choice Page:

To cater to the diverse needs of the Partner Groups, this page presents a selection of boilerplates tailored to the Partner Groups, each representing specific business logic according to their needs. For example, for CTT, is a Return boilerplates that deploys and configures the LoopOS applications to handle the return process. After choosing the Partner is redirected to the Wait Page. The mockup for this step can be seen in Figure 6.4.

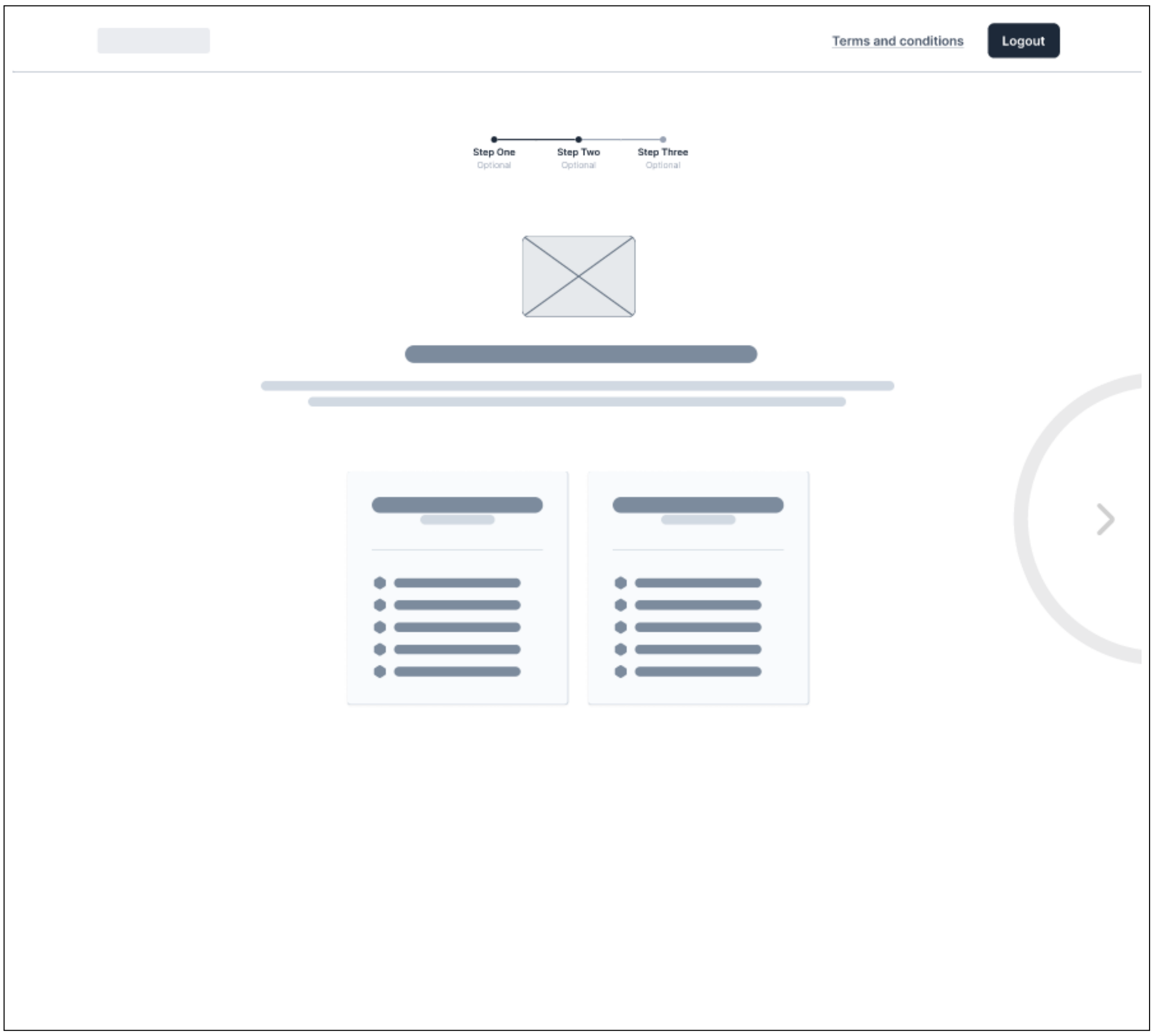


Figure 6.4: Mockup of the Onboarding Choice Page.

Page 4 - Wait Page:

The Wait Page provides real-time updates on the LoopOS applications creation and configuration progress, keeping Partners informed about the status of their newly deployed applications. A notification is displayed upon successful app creation, ensuring transparency and a sense of accomplishment. Once complete, Partners are presented with a list of their newly deployed applications, providing easy access and navigation. This streamlined approach ensures they can instantly use the needed applications to streamline operations and achieve business goals. The mockup for this step can be seen in Figure 6.5.

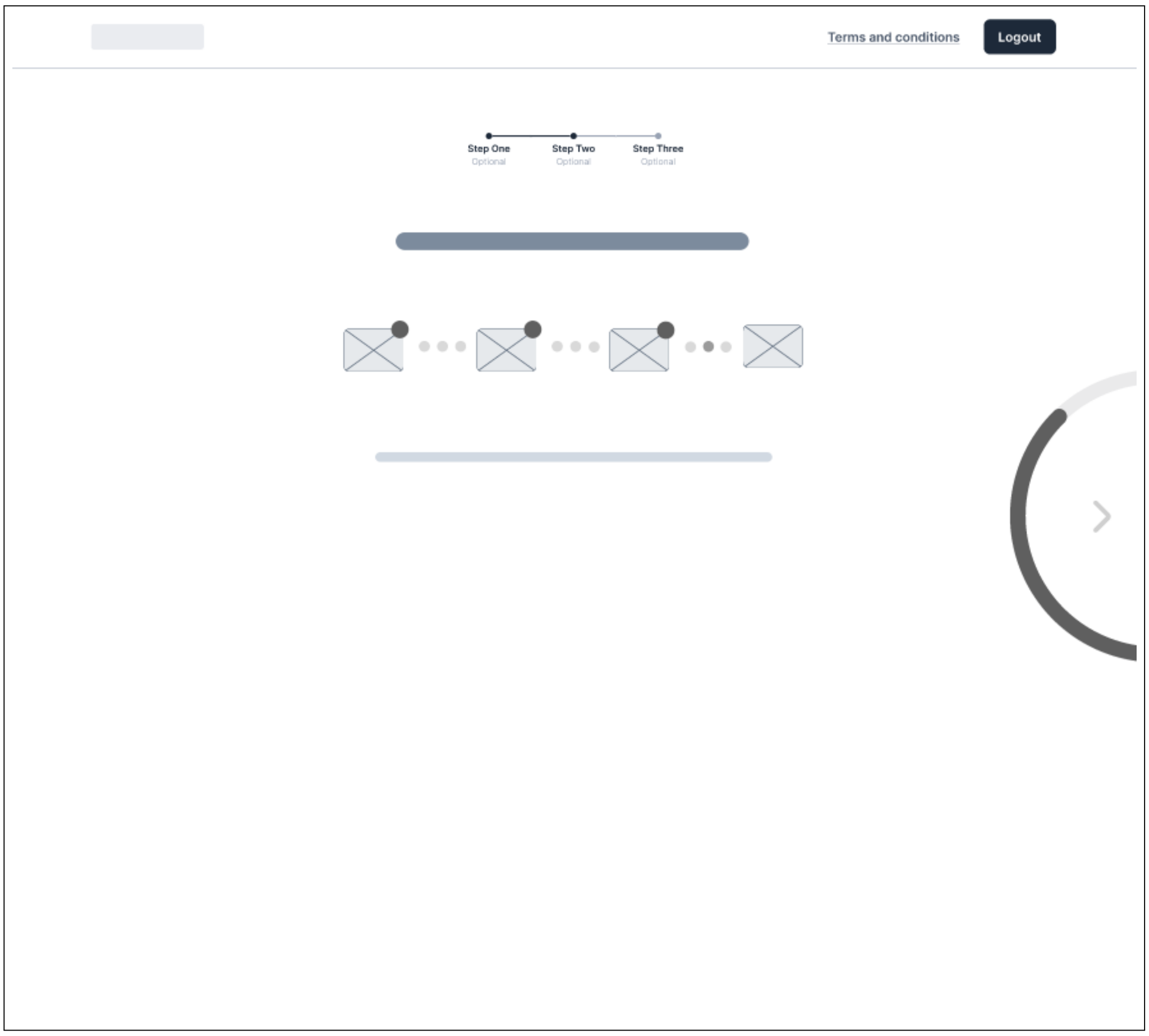


Figure 6.5: Mockup of the Wait Page.

◆ Development

With this initial planning done, my development started with creating the Partner Group in the Network Manager. Several steps were taken to implement the necessary changes and establish relationships within the existing data model.

Firstly, I created the new model, following the Model-View-Controller pattern. Most notably, the Partner Group includes name, logo, partner settings schema, and user settings schema, which can be associated with Partners, Users, App Instances, and Partner Services.

Then, I modified the Partner model by adding a new optional *belongs_to partner_group* relation. Furthermore, settings and *user_settings_schema* fields were incorporated, and subsequent logic was implemented to ensure that this new setting field adhered to the schema of Partner Group if the relation existed.

For example, if a Partner belongs to a Partner Group, and the Partner Group schema is as presented in Listing 6.1:

```
{
  "required": [
    "return_address",
    "valid_return_days"
  ],
  "properties": {
    "return_address": {
      "type": "string"
    },
    "valid_return_days": {
      "type": "integer"
    }
  }
}
```

Listing 6.1: Partner Groups' partner_setting_schema

Then the Partner settings must obey it so, for example, as seen in Listing 6.2:

```
{
  "return_address" : "1234 Main Street",
  "valid_return_days" : 15
}
```

Listing 6.2: Partners' setting

Simultaneously, the User model was modified to include a relationship with both Partner and Partner Group, with the logic in place to respect the schema of these entities, depending on which one is present or if both are.

I applied a similar logic to the User and the Partner model regarding obeying *settings_schema*. It covers two event hooks with the following steps:

Before Creation:

1. **Populate Settings:** The *populate_settings* method is invoked to populate the settings field with the default values for the schema. This ensures that the object has some initial settings, even if they are not explicitly provided in the creation request.

2. **Parse JSON Fields:** The *parse_json_fields* method is called to parse the settings field into a hash format if it is not already a hash. This ensures that the settings are consistent and structured before saving the object to the database.

Before Save:

1. **Parse JSON Fields:** Repeatedly, the *parse_json_fields* method is invoked to ensure that the settings field is always in a consistent hash format. This happens before creating and saving the object, ensuring the settings are properly parsed and formatted.

2. **Check Settings Changes:** The *check_settings_changes* method is executed to check if the settings have changed since the last save operation. If any changes occur, the settings are validated against the schema to ensure they adhere to the expected structure and values.

Finally, due to the User having the possibility of obeying two different schemas, a method was implemented to combine the schemas of the Partner Group and the Partner, if any are associated, before either saving or creating the object. This ensures that the schema is always up-to-date with the current associations of the object. Figure 6.6 showcases this logic.

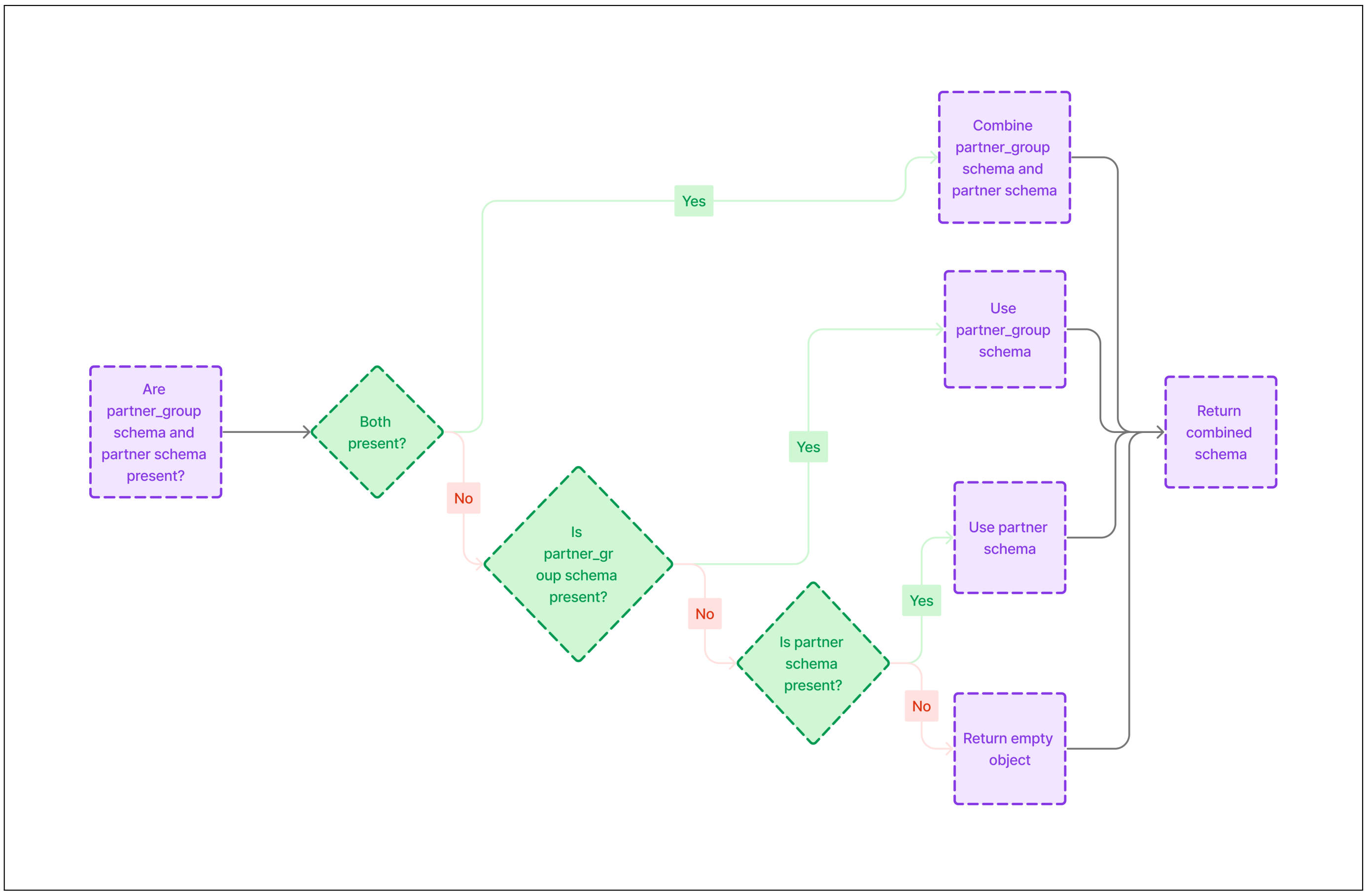
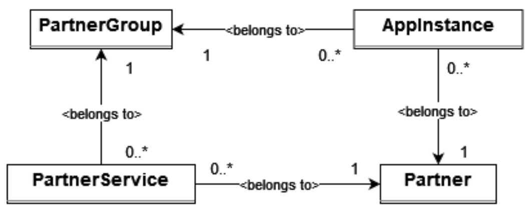

Are partner_group schema and partner schema present?
Both present?
Yes
No
Is partner_group schema present?
Yes
No
Is partner schema present?
Yes
No
Combine partner_group schema and partner schema
Use partner_group schema
Use partner schema
Return empty object
Return combined schema

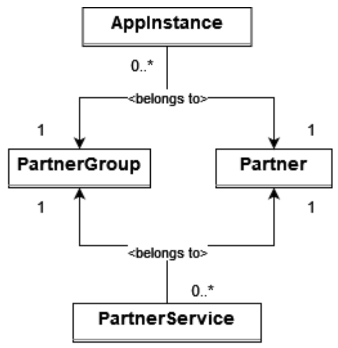
Figure 6.6: Flow chart diagram representing logic that keeps User settings valid with its Partner or Partner Group association.

Furthermore, a crucial step involved establishing a one-to-many relationship between Partner Group and App Instances, as well as between Partner Group and Partner Service. However, these models could only have a relationship with either Partner Group or Partner.

To achieve this relationship, I implemented polymorphism by introducing a new Partnable class, thereby promoting code reusability and flexibility. With this approach, App Instance and Partner Service objects became Partnable, eliminating the need to determine if the object belonged to a Partner or a Partner Group. Figures 6.7 depict two Entity-Relationship diagrams illustrating the difference between using and not using polymorphism, respectively.



(a) Without Polymorphism.



(b) With Polymorphism.

Figure 6.7: Relation between Partner Service and App Instance with Partner/Partner Group with and without polymorphism.

To streamline code and facilitate the sharing of fields, methods, and relationships between Partner and Partner Group, a Concern was introduced. A concern in this context refers to a modular and reusable code component that encapsulates shared functionality for these two entities.

To ensure data integrity and a smooth transition, a migration was created. This migration transferred data from the existing *AppInstance.partner* and *PartnerService.partner* columns to the new partnable column in each table. This was a crucial step to maintain data continuity and consistency.

Listing 6.3 demonstrates the implementation of polymorphism and the Partnable concern in Ruby on Rails to establish fully functional relationships between Partner, Partner Group, App Instance, and Partner Service models.

```ruby
class Partner < ApplicationRecord
  #this is to include the methods in the Partnable
     Concern
  include Partnable
end

class PartnerGroup < ApplicationRecord
  #this is to include the methods in the Partnable
     Concern
```

```
    include Partnable
  end

  class AppInstance < ApplicationRecord
    belongs_to :partnable, polymorphic: true
  end


  class PartnerService < ApplicationRecord
    belongs_to :partnable, polymorphic: true
  end

  module Partnable
    extend ActiveSupport::Concern
    included do
      has_many :partner_services, dependent: :destroy, as:
         :partnable
      has_many :app_management_app_instances, dependent: :
         destroy, as: :partnable
      #...
    end
  end
```

Listing 6.3: Ruby on Rails Polymorphic Associations and “Partnable” Concern Implementation.

Finally, to reflect the changes, all calls to Partner Service and App Instance objects and methods were updated to reference the new partnable column instead of the old partner column. This required an in-depth analysis of the entire project code, as all these models needed to access the new partnable column.

With the new Partner Group model, several significant modifications were implemented to facilitate a more seamless user experience and correctly guide users through the onboarding process.

First and foremost, a new “state” field was incorporated into the User model within the Network Manager. This field is pivotal in tracking the progression of users and ensuring they are directed through the appropriate stages of their onboarding journey.

In addition to introducing the “state” field, the user flows have been restructured in the Network Manager to align with four possible distinct states. These states are as follows:

- **Onboarding:** Users entering this state are those who sign up via Single Sign-On (SSO) and currently have no associations, such as partners or app instances. This state signifies the initial phase of the onboarding process.
- **Invited:** Users categorized as “invited” fall into this state. They are individuals created within the system by a manager and subsequently received an

invitation to join the platform.

- **Active:** The "active" state applies to users who have completed the onboarding process. The Onboarding module will oversee this phase and encompasses users who have finalized the invite flow.
- **Inactive:** While currently not in active use, the "inactive" state has been reserved for future functionalities. This provides flexibility for anticipated developments in the system.

Hence, when a user in LoopOS Onboarding clicks Login, they are redirected to Network Manager. If the user chooses to use CTT SSO, they are redirected to CTT and logged in. If the login is accepted, the user is redirected to Onboarding. If the user creates an account with CTT, they are also created in Network Manager with state onboarding. If the user uses the default login, they are logged in with Network Manager and redirected to Onboarding. In Onboarding, if the user's state is onboarding or invited, they are redirected to the Onboarding Registration Page. If the user's state is active, they are redirected to Network Manager. Figure 6.8 illustrates this flow.

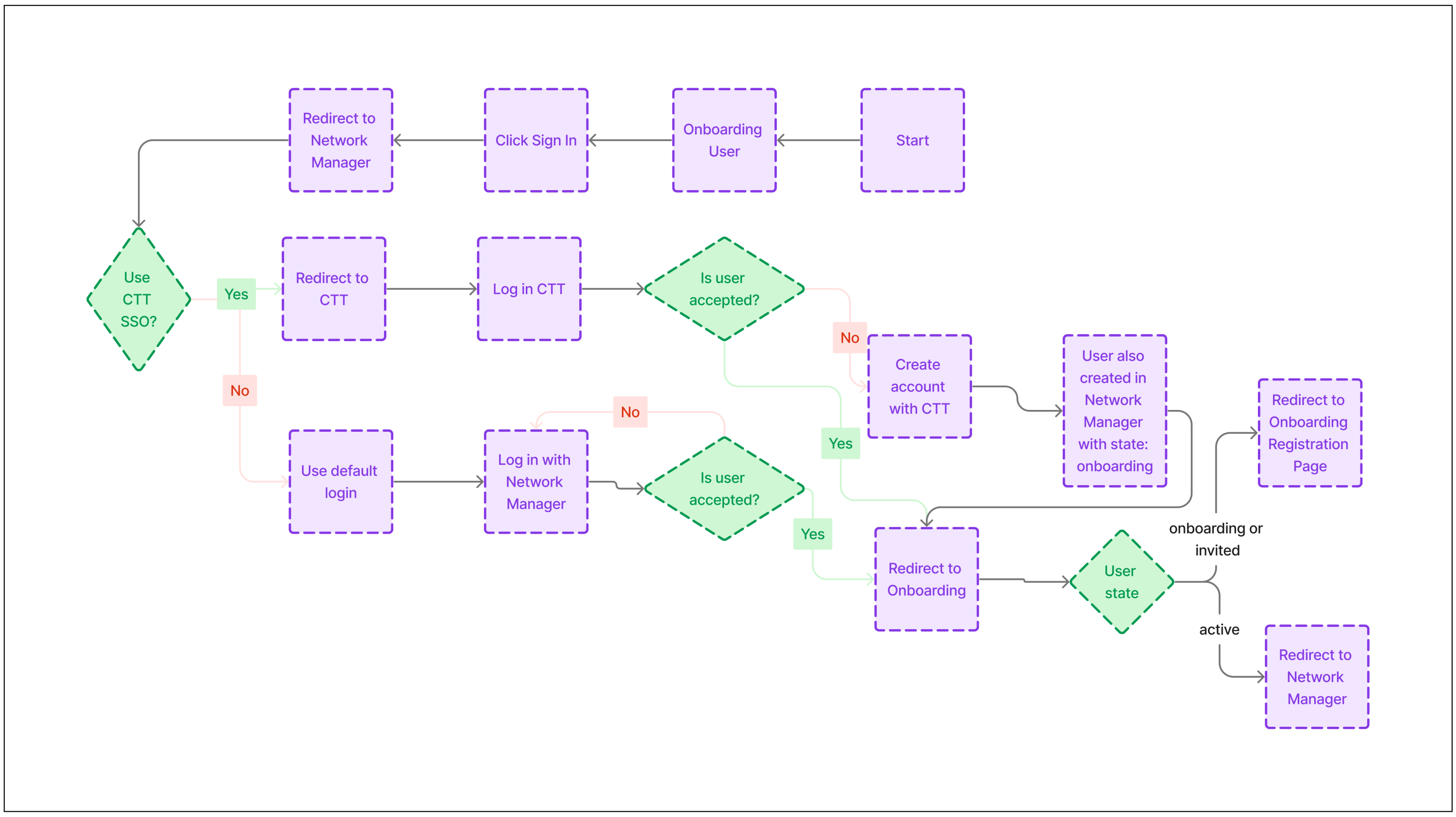


Figure 6.8: Diagram representing planned User authentication flow.

Then I started developing the LoopOS Onboarding application, creating a simple Rails Application and a Landing Page. It should be clear that my objective here was to develop the backend logic. In the case of this page, I would need to implement the logic behind obtaining the App Instance settings from the Network Manager so later it would be possible to stylize the page, adding a "sign up" button that redirects to the manager with a callback to come back to this app, and when the user came back to check the user state, implementing the previously planned authentication flow. However, implementing the SSO required collaboration with CTT, such as enabling access to their SSO through our domain. However, due to delays on their side, the integration of the CTT SSO was postponed. This ensured that the development of the LoopOS Onboarding remained on track.

There were two key features implemented in this process. The first was to implement the logic so that, from the LoopOS Onboarding, it would be possible to trigger an endpoint in the Network Manager and, in turn, to send the needed App Instance settings.

To achieve this, I first copied the security logic from other applications, as explained in Chapter 2, in Sub-section 2.4.2. This meant, in order to make requests to the Netowrk Manager, i would first have to authenticate the application. The ID and Secret of each application is stored in two places: In the corresponding App Instance object in the Network Manager and in the ".env" file that stays with the application. Hence, in LoopOS Onboarding, i created the logic to send its ID and Secret to the Network Manager, which in turn creates and sends an *access_token* so I can add it to the Onboarding requests. Figure 6.9 illustrates this logic.

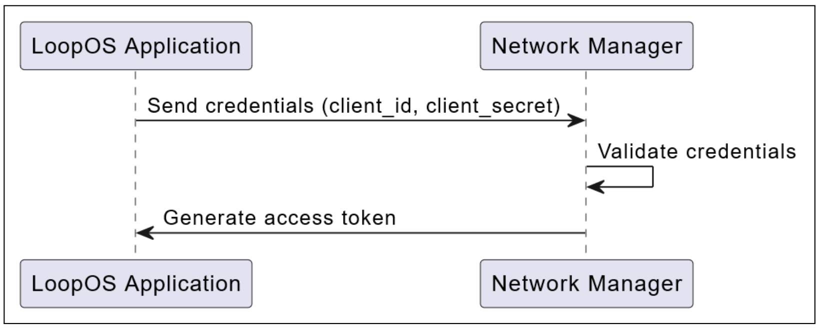


Figure 6.9: Authentication Flow in LoopOS App Authentication [16].

Creating endpoints in the Network Manager is straightforward since Ruby on Rails eases the process. First, in the *routes.rb* config file, which is used to map URL requests to application actions and requests, I declared a new endpoint, as seen in the code snippet present in listing 6.4. The application ID is the same ID used in the authentication.

```
namespace :api do
   namespace :v1 do
     get "applications/:application_id/settings", to: "
        applications#settings"
```

Listing 6.4: Application settings endpoint.

Then I created the following logic in the controller handling the endpoint:

1. **Doorkeeper authorization:** Ensure that the user can access the application settings. This is done by checking if the user has a valid access token for the

application. This token is received in the request's header, as was explained before.

2. **Setting retrieval:** Retrieve the App Instance that corresponds to the provided application_id.

3. **Application data gathering:** Since I wanted this endpoint to be used by other applications, I decided to send all relevant data connected to the requested instance, sending a JSON object with:

   - The instance id, private and public settings;
   - The associated Partner or Partner Group id, slug, name, and logo;
   - An hash with all URLs and logo of all App Instances connected to the associated Partner or Partner Group.

In case the application instance was not found, a 404 - Not Found error is returned. Then, in the LoopOS Onboarding, I sent a GET request to the corresponding URL with the token in the header following these steps:

1. **Check for cached settings:** Check if the settings for the specified application ID are already cached. If they are, it returns the cached settings.

2. **Fetch OAuth token:** If the settings are not cached, the class first sends a request to the Network Manager to get an OAuth token from the Network Manager API.

3. **Make API request:** Once the OAuth token is obtained, make an API request to the Network Manager to get the settings for the specified application ID.

4. **Cache settings:** If the API request is successful, the class caches the retrieved settings for the specified application ID. This ensures that the settings are not fetched from the API whenever needed.

Figure 6.10 illustrates this flow.

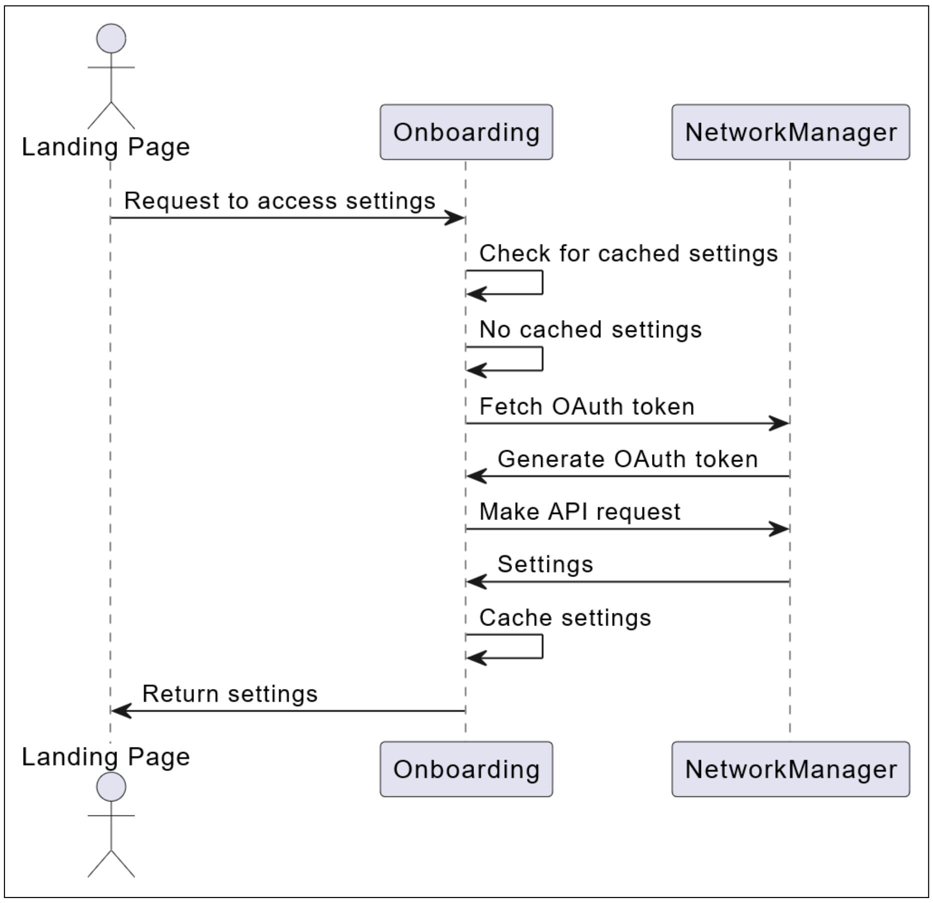


Figure 6.10: UML diagram: Fetch settings endpoint example where settings are not yet cached so the full request must be executed.

The second key feature was the way to obtain the user state. Obtaining the user state was a crucial aspect of the onboarding process. Initially, three potential solutions were considered. The first option was establishing a dedicated endpoint solely for retrieving user information. However, relying heavily on endpoints for data retrieval can strain both applications, particularly the Network Manager, which operates as a single instance.

The second approach would use the previously mentioned endpoint for fetching application data. However, I created the aforementioned endpoint so it would be usable by other Applications, hence an advantage to have in the Network Manager. Re-purposing it exclusively for user state retrieval, primarily required only for onboarding, would render its original purpose obsolete.

The final solution, which was ultimately adopted, involved modifying an existing call that was already necessary. As mentioned, two types of authentication are implemented in LoopOS: User and Application.

Chapter 2, Sub-section 2.4.2 explains this process, however Figure 6.11 revisits it with an example.

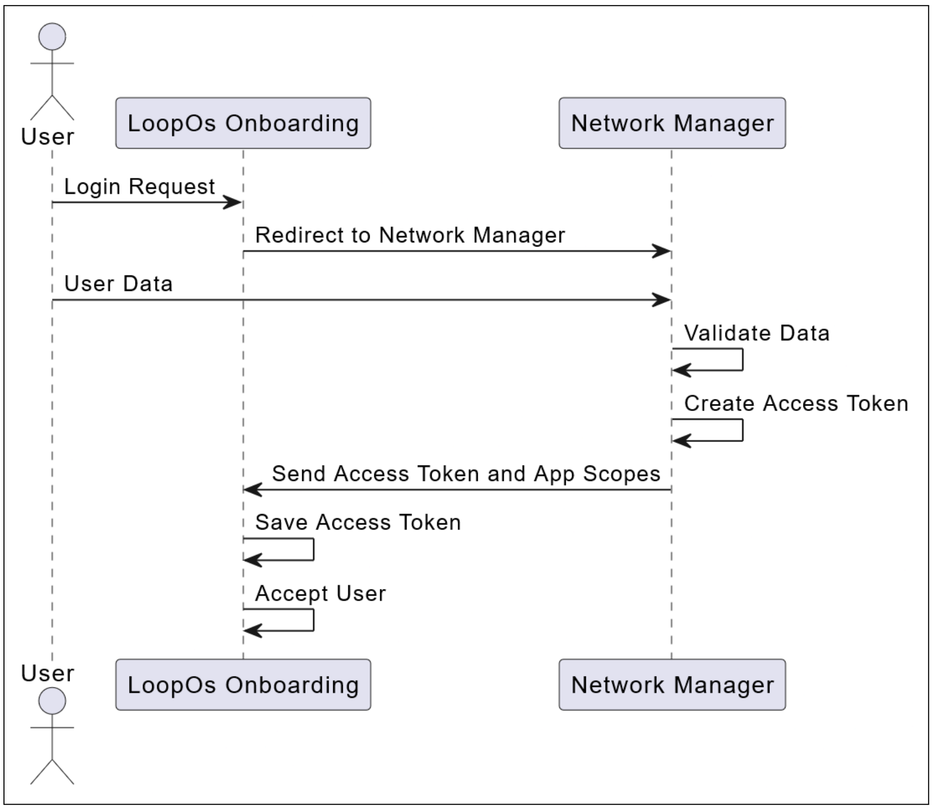


Figure 6.11: UML diagram: Example of a User login attempt to LoopOS Onboarding.

Additionally, as explained in this subsection, this authentication is aided by the use of Doorkeeper and Devise. In a simplified explanation, while Devise is a user authentication framework that simplifies the process of user registration, login, and password management, Doorkeeper is an OAuth 2.0 server that manages access tokens, authorization grants, and scopes.

In this case, the Doorkeeper is the one aiding in creating the token, where the Network Manager has applied a custom token response to also send the user App Scopes. Devise handles the authentication, from checking if the password is correct in the Network Manager to creating a new User with the Doorkeeper data on the Onboarding app.

With this knowledge, the solution was straightforward. First, change the doorkeeper custom token response in the Network Manager as can be seen in the code snippet presented by listing 6.5.

```
module Overrides
  module Doorkeeper
    module CustomTokenResponse
      def body
        additional_data = {}
        if @token.user?
```

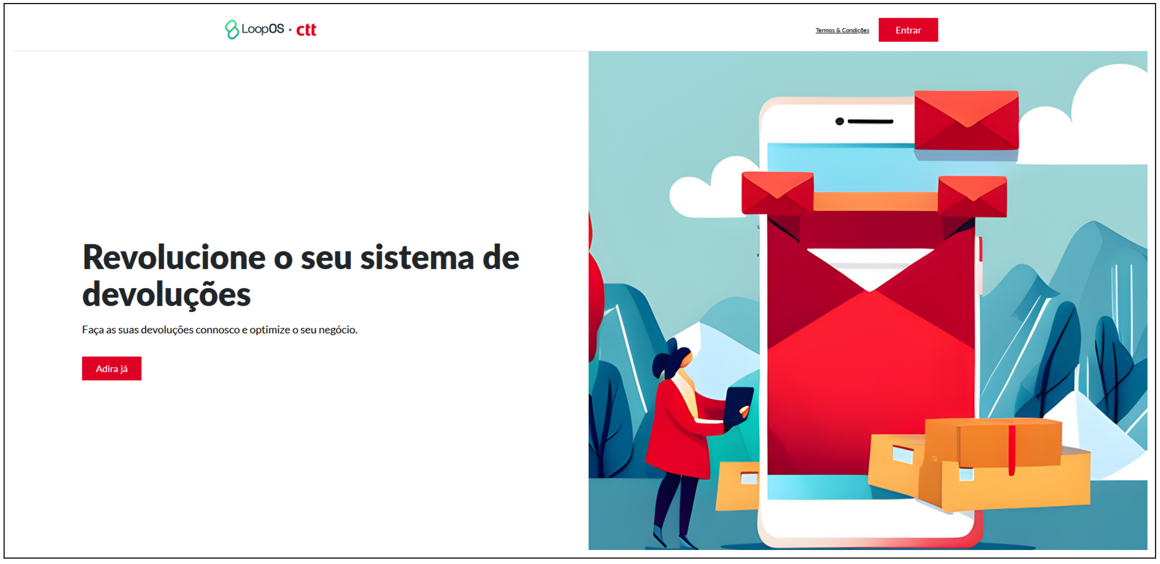
```
              #...
              additional_data[:app_scopes] = user_scopes.
                 pluck(:name).join(" ")

              #new code that adds state to token
              additional_data[:user] = { state: @token.user
                 .state, id: @token.user.id }
            end

            super.merge(additional_data)
          end
        end
      end
    end
```

Listing 6.5: Custom token response override in Network Manager.

Then, to ensure that the state attribute of the User model in the Onboarding application is always set to the one from the Doorkeeper token, I added the attribute to the model, created a migration to add the column to the database, and implemented the logic to set the state attribute when the user is updated. This update will always occur when the user logs in.

In summary, the solution to the problem of determining the user's state for the Onboarding application involved modifying the Doorkeeper token response and ensuring the User model's state attribute was properly synced. Hence, while the implementation was straightforward, it required a thorough understanding of the underlying components and their interaction. The same can be said for creating the logic between the communication of the Network Manager and the LoopOS Onboarding.

Now that the landing page had the necessary information, the design team could begin implementing the front end. Figure 6.12 showcases the final look of the Partner Landing Page in LoopOS Onboarding for CTT Returns.



Revolucione o seu sistema de devoluções

Faça as suas devoluções connosco e optimize o seu negócio.

Figure 6.12: LoopOS Onboarding: Final look of the Partner Landing Page.

Moving on with the user flow, after a successful login, the user is then redirected to the second page: Partner Registration page.

The Partner Registration Page should serve two primary functions: dynamically populate the page with settings specific to the Partner Group, allowing potential partners to provide the required information, and upon successful completion of the registration process, seamlessly integrate the newly registered Partner into the Network Manager's system.

Starting the population of the page, I first created an endpoint in the Network Manager to send the Partner Group data belonging to a specific ID. Then, I called this endpoint using the Partner Group ID, fecthed from the previous endpoint, on the Registration Page Controller. Using the same logic of the previous endpoint, I requested an access token, sending the application ID and secret to the Network Manager if the token was not saved or had expired. Afterwards, I performed a GET request using the token in the header. Consequently, obtaining the *partner_settings_schema* from the Partner Group.

With this data, I implemented a basic front-end. Firstly, there were some hardcoded default fields in the beginning, such as the required name and logo, and the optional icon. In the end of the page, added the mandatory terms and conditions that were populated from the *public_settings* of the application settings endpoint.

In the middle, I generated the dynamic fields according to the schema. The following JSON object present in listing 6.6 is an example schema to better visualize the rules I applied to create this page.

```
{
  "required": [
    "address_city",
    "address_street"
  ],
  "properties": {
    "address_city": {
      "type": "string"
      "order": 2
    },
    "want_notifications": {
      "type": "boolean"
      "order": 5
    },
    "valid_day_return_days":{
      "type": "integer"
      "minimum": 1,
      "maximum": 10
    },
    "address_street":{
      "order": 3
    }
  }
}
```

Listing 6.6: Example Partner Setting Schema

The rules applied during the generation were as follows:

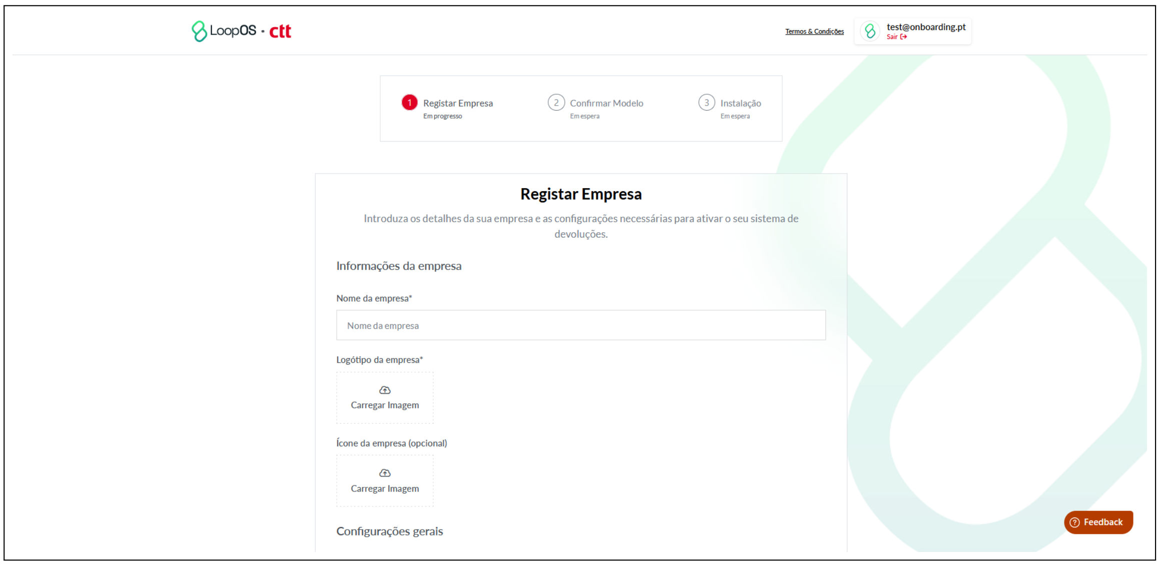
- All settings listed in the required section of the schema must be included.
- The input field type must match the corresponding type specified in the schema. If the schema omits the type, the input becomes a string.
- If present, all values must adhere to the specified minimum and maximum boundaries. This applies to both strings and integers.
- The order property acts as a weight, with lower numbers indicating higher priority placement. Input fields with no order value are positioned at the bottom. In the case of ties, the order is determined alphabetically by the key name.

It was essential for the settings filled by the Partner to follow the schema. If in the Network Manager a Partner was created disobeying the schema, for example, if the schema mandated for a field to be of type integer and arrived as a string, it would deny its creation. This was the logic I created when implementing the new settings and schemas in the User/Partner/Partner Group. Hence the strict following of these rules.

With the data filled in, I created a button on this page that called a POST endpoint to the Network Manager. I created this endpoint with logic similar to the other endpoints. In this case, the Onboarding would send the access token in the header and the Partner data in its content. However, since the Doorkeeper token contains which application performed the request, it would not be necessary to also send the Partner Group ID in the request since, knowing the application, The Network Manager would know the Partner Group who owned it. Additionally, since Devise saved the current user email on login, I would not need to send it in the body of the request. The Network Manager would then create a new Partner with the submitted data, while also ensuring that it belongs to the specified Partner Group and is associated with the correct user.

However. during testing, the Partner Registration page, which adhered to the Partner Group schema, failed to process the POST requests. This was due to the fact that, even if the settings were of the appropriate type in the LoopOS Onboarding, they would always arrive as strings in the Network Manager. To successfully create a partner, I implemented logic to type-cast these settings to the proper format in the endpoint.

After creating the Partner in the Network Manager, the endpoint then sends a response with the status created, and thus, the User could be redirected to the Boilerplate Choice Page. Figure 6.13 showcases the final look of the Partner Registration Page in LoopOS Onboarding for CTT Returns.



Registar Empresa

Introduza os detalhes da sua empresa e as configurações necessárias para ativar o seu sistema de devoluções.

Informações da empresa

Nome da empresa*

Nome da empresa

Logótipo da empresa*

Ícone da empresa (opcional)

Configurações gerais

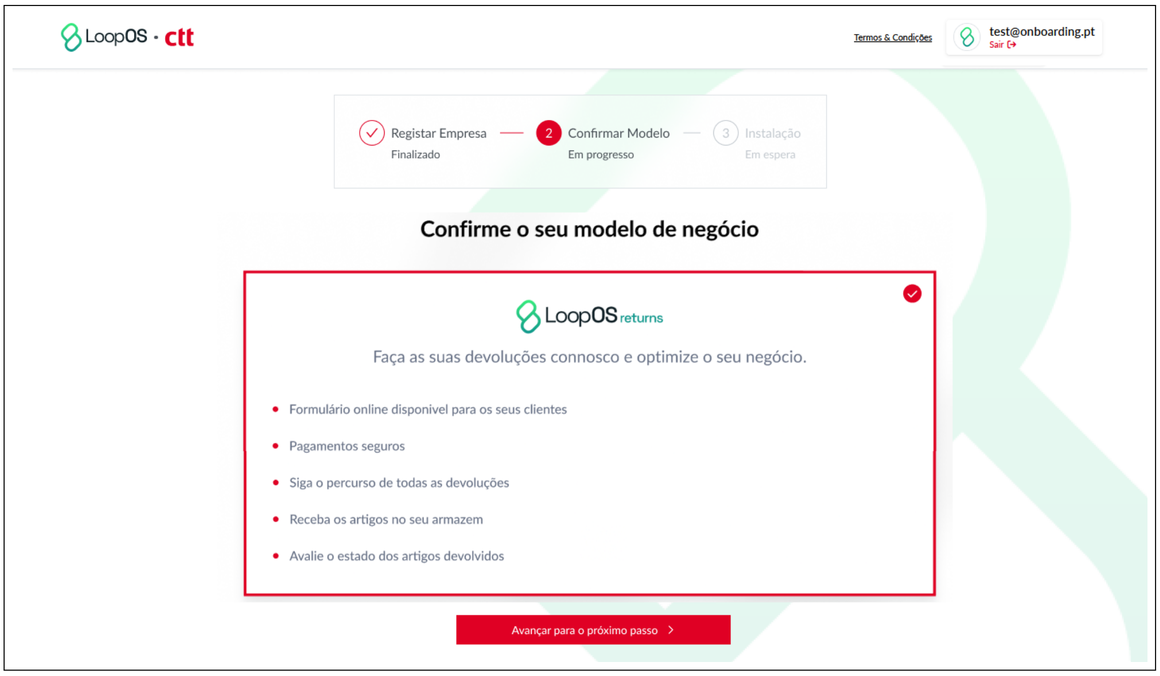
Figure 6.13: LoopOS Onboarding: Final look of the Partner Registration Page.

Having worked on a wide range of LoopOS applications and possessing a comprehensive understanding of the LoopOS technological ecosystem, I believe this page to be the most impactful creation I have yet achieved. Partners can now generate a fully functional LoopOS environment from scratch with a single button press.

Hence, to streamline the onboarding process for Partners, I would display a list of boilerplate on this page, each accompanied by an icon and a description outlining its deployment capabilities. When a partner selects a boilerplate, it would trigger an endpoint that executes the chosen script.

First, I modified the existing LoopOS Script model in the Network Manager to allow icons and descriptions for better presentation to partners. Anticipating a growing number of boilerplate scripts for various business models in the future, I incorporated the boilerplate script IDs into the App Instance's private settings so as to only present a select number of boilerplates.

Then, I created a controller in the Network Manager to handle a new endpoint with two distinct tasks. First, the GET "/boilerplate" that, when triggered, would check which application made the request, check its private settings, get the boilerplate IDs, and send back the details of these scripts so I could display them on the Boilerplate Choice page. Figure 6.14 showcases the final look of the Boilerplate Choice Page in LoopOS Onboarding for CTT Returns. Currently, with only one boilerplate script available, Partners are limited to selecting one for their onboarding needs. However, as the LoopOS expands to accommodate more Onboarding apps for different Partner Groups with unique requirements, this page will evolve to accommodate various business logics.



Confirme o seu modelo de negócio

LoopOS returns

Faça as suas devoluções connosco e optimize o seu negócio.

- Formulário online disponivel para os seus clientes
- Pagamentos seguros
- Siga o percurso de todas as devoluções
- Receba os artigos no seu armazem
- Avalie o estado dos artigos devolvidos

Figure 6.14: LoopOS Onboarding: Final look of the Boilerplate Choice Page.

The second endpoint that I created in the Network Manager was the POST "/trigger_boilerplate", which would need to receive the chosen boilerplate ID. Due to the Network Manager knowing the user that sent the request, it was able to obtain its associated Partner and Partner Group, and thus not need to include them in the body of the request. Figure 6.15 illustrates this flow.

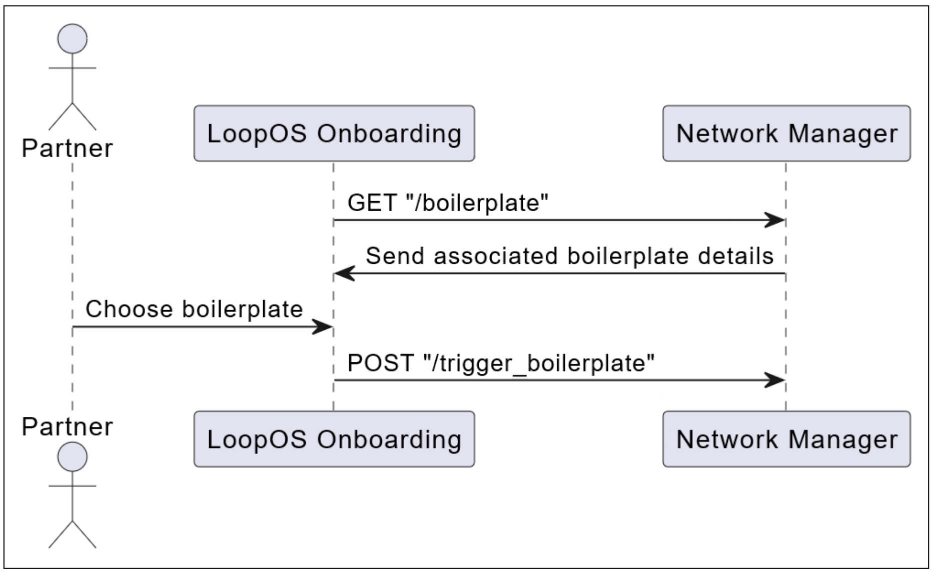


Figure 6.15: UML diagram: Choose and trigger boilerplate.

As discussed in Chapter 2, Section 2.4, LoopOS scripts come in three forms: Scripts, Boilerplates, and Validators. Each script type has a predefined skeleton that establishes its mandatory methods. This is crucial because the Network Manager, when executing these scripts, must know the order to run these methods. Listing 6.7 demonstrates the boilerplate script skeleton.

```
def run_boilerplate(log, user, partner, partner_group,
   extra_info: {})
    # Write code here
end
```

Listing 6.7: Skeleton of boilerplate type script.

The main method, *run_boilerplate*, gets passed four parameters: the log to update when running the boilerplate, the user requesting to run the boilerplate, the partner (optional) associated with the boilerplate, the partner group (optional) associated with the boilerplate, and the extra_info (optional) a hash that might be required for the boilerplate.

It is important to understand here that CTT was the first client to use this product, so I had restrictions in the creation of this script since I needed to create it to their needs. LoopOS Returns, as the boilerplate script would be called, would need to allow the corresponding flow:

1. Client submits and pays for item return.

2. Client receives shipping guide by email.
3. Client adds shipping guide to package and sends it.
4. CTT delivers package to Partner
5. Partner receives Package

The minimum number of LoopOS applications to implement this flow is three. One Submission for the clients to submit the product and for CTT to know which items it needs to deliver, one Handling per partner to receive the product, and one Core per Partner to manage these applications. Figure 6.16 showcases the connection between these applications.

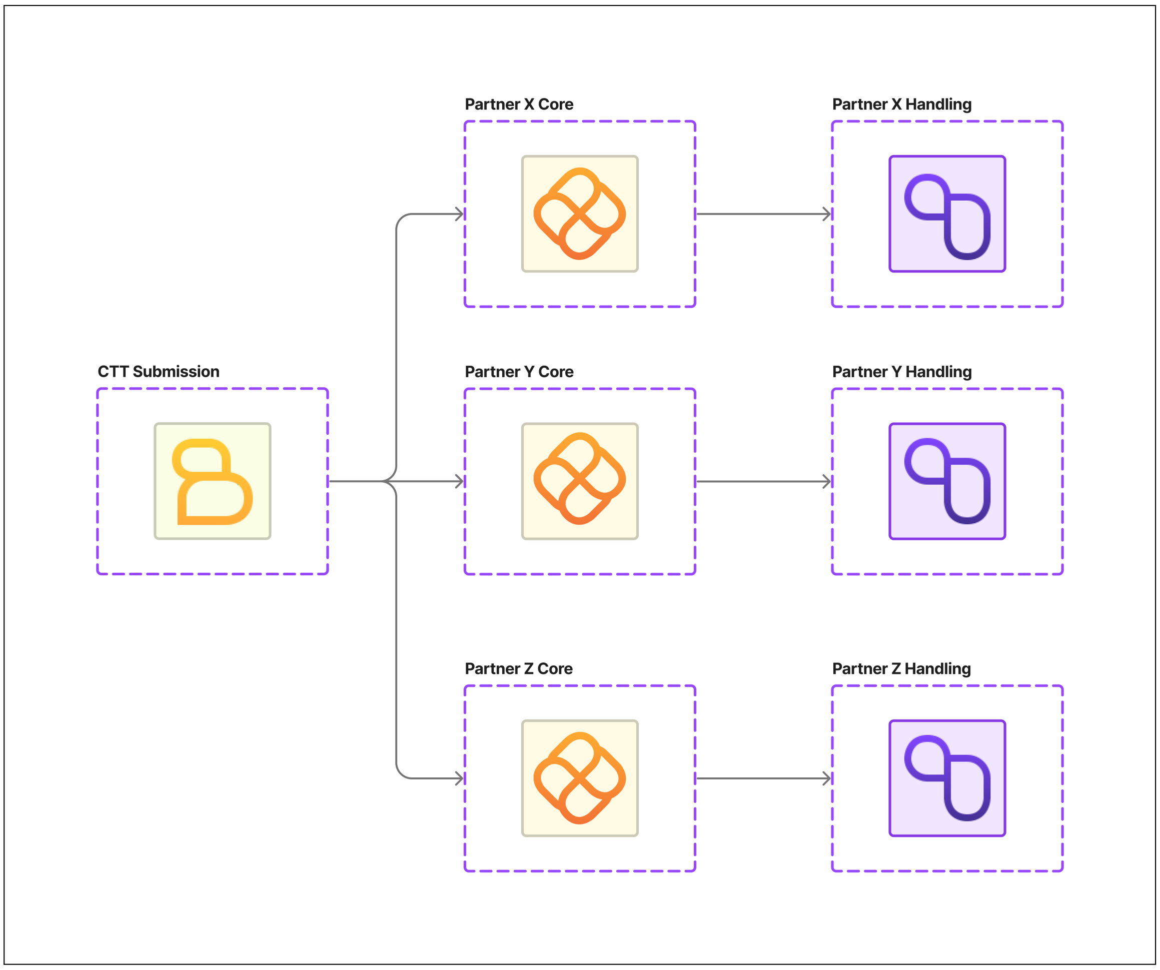


Figure 6.16: Connection between CTT applications and Partner applications.

Hence, the following was the agreed journey an item would take using the aforementioned applications. It will be split into three: Client submission, Client payment and item delivery, and Partner item acceptance. This is to ease the explanation:

The client begins by submitting the product information and client details in the CTT Submission. This action prompts the creation of an item object in the Partner Core. The Partner Core then requests the Network Manager to send an email notification to the client upon completion of the submission and to generate an Incoming Payment. Figure 6.17 outlines this process.

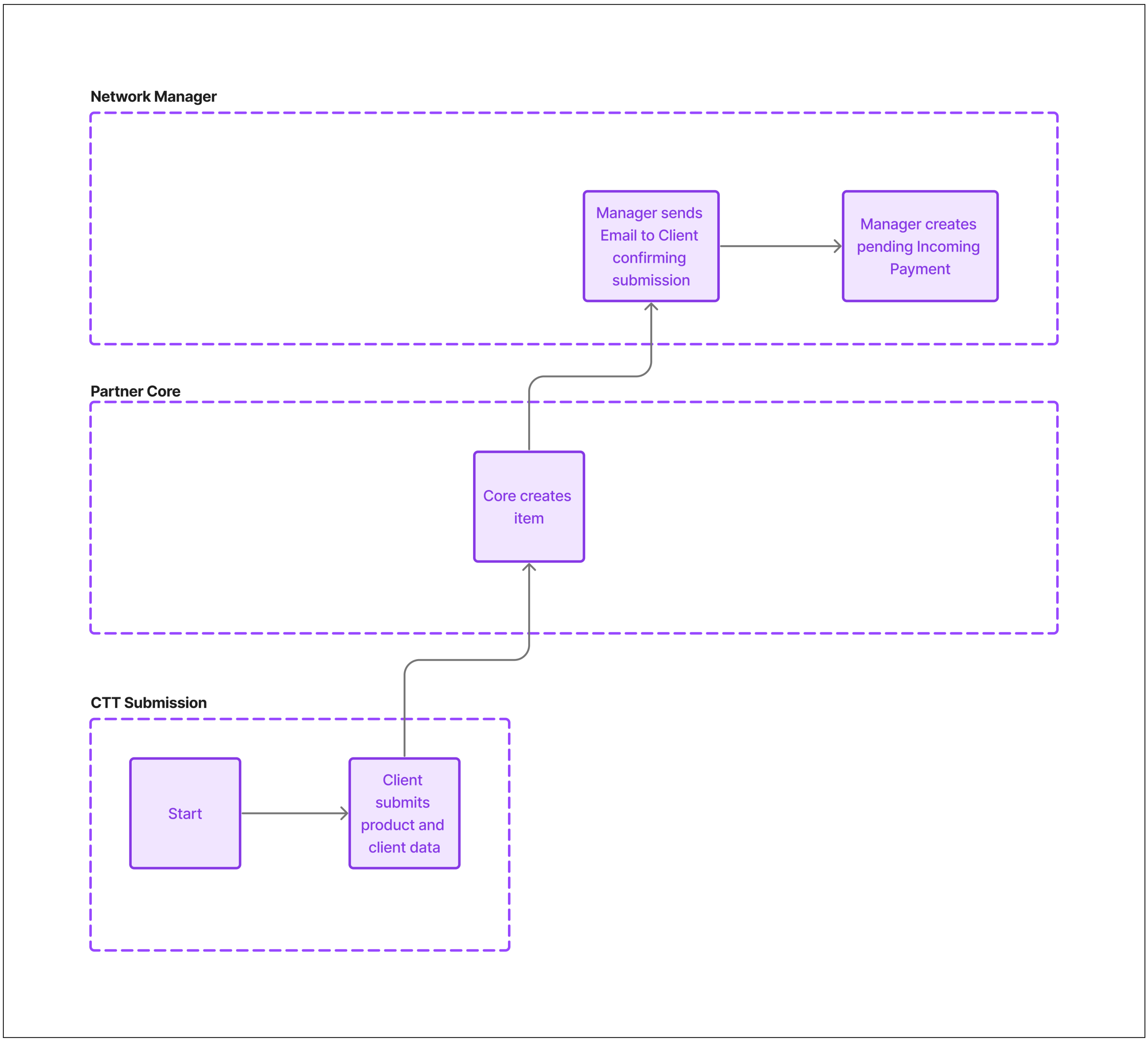

Network Manager
Manager sends Email to Client confirming submission
Manager creates pending Incoming Payment
Partner Core
Core creates item
CTT Submission
Start
Client submits product and client data

Figure 6.17: Client submits the item(s) it wants to return.

Once the client has paid for the return, the Payment API will communicate with the Network Manager, updating the Incoming Payment's status to complete. An email notification will be sent to the client informing them of this change. The Partner Core will then update the item's status to reflect the payment completion. Next, the Partner Core will request a shipping guide from the Network Manager. The Network Manager will create the shipping guide and send it to the client via email so it can add it to its package. The Partner Core will then mark the item as shipped. Figure 6.18 illustrates this process.

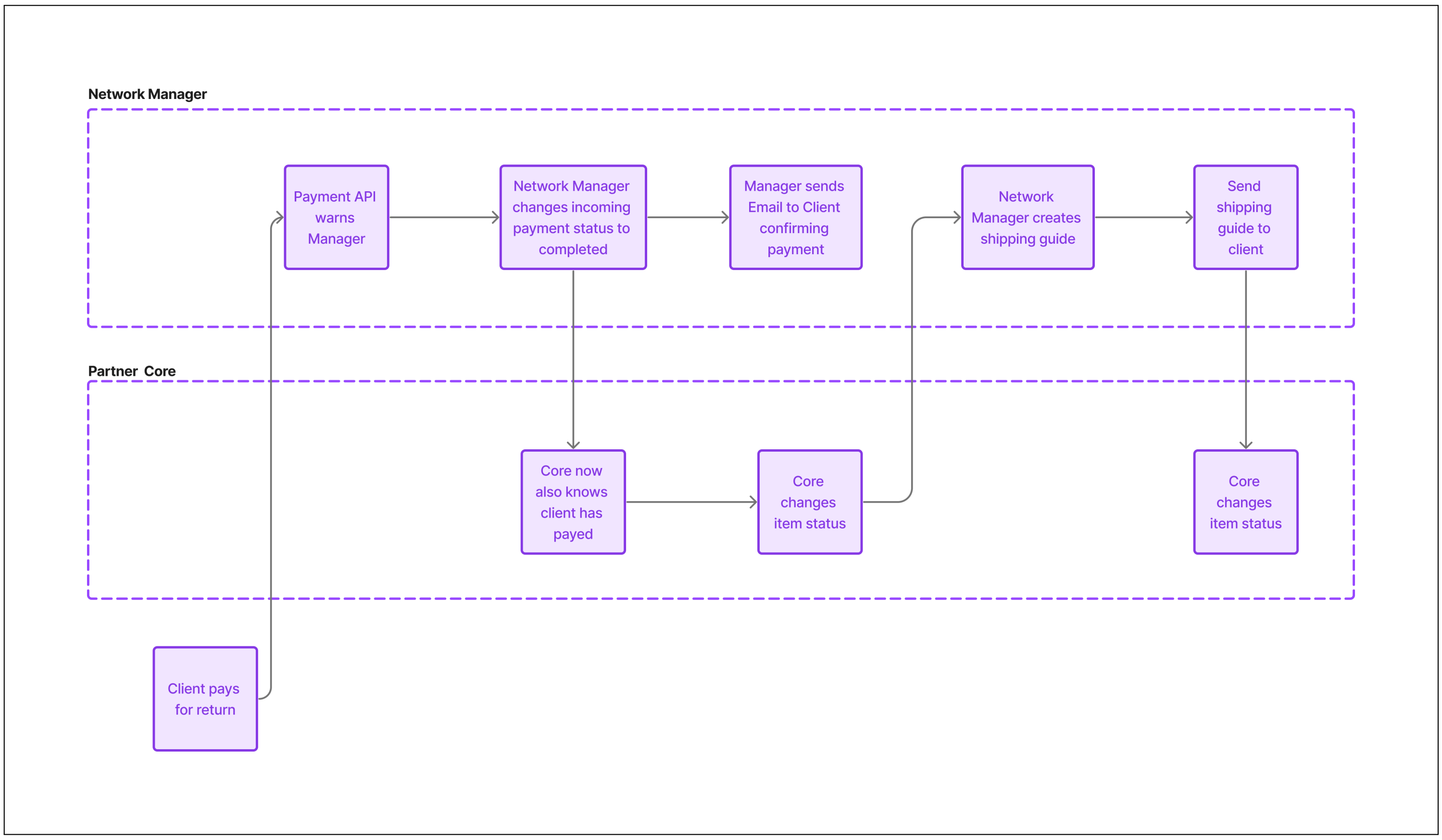

Network Manager
Payment API warns Manager
Network Manager changes incoming payment status to completed
Manager sends Email to Client confirming payment
Network Manager creates shipping guide
Send shipping guide to client
Partner Core
Core now also knows client has payed
Core changes item status
Core changes item status
Client pays for return

Figure 6.18: Client pays for item return.

Once the client has shipped the item, the CTT will handle the delivery. The Partner will then receive confirmation of the item's receipt in its Handling application, which will promptly notify its Core and request the Network Manager to inform the client of the item's arrival. Figure 6.19 depicts this process.

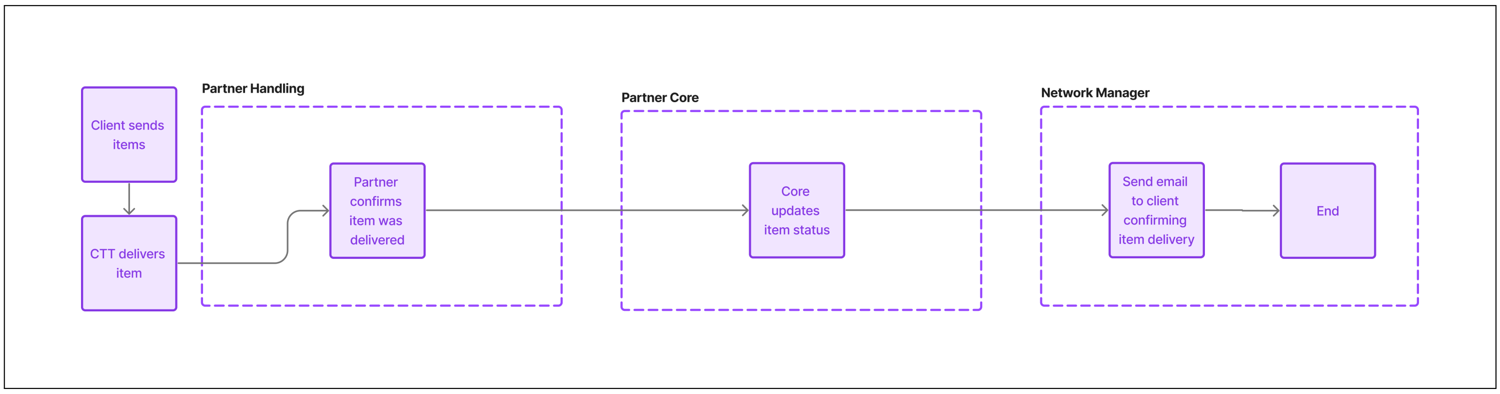


Figure 6.19: Partner receives item.

Finally, there are only two more concepts I want to detail before outlining how the script was developed.

Most notably, LoopOS Flows. As stated, it is a no-code logistics solution designed specifically for reverse logistics. Users can effortlessly create personalized flows by dragging and dropping various blocks like apps and services. This is how the Core knows the correct flow of the system.

To illustrate, Figure 6.20 depicts a segment of a LoopOS Flow comprising three blocks. In this instance, upon insertion of an item through Submission, its status transitions to “Submitted” which aligns with the exit condition of this block. Consequently, it attempts to transition to the next block. It succeeds because the entry condition for that block is the absence of email delivery. Thus, an email is dispatched to the user notifying them that the item has been submitted. Subsequently, the email’s delivery triggers the block’s exit condition, prompting it to exit and proceed to the subsequent block.

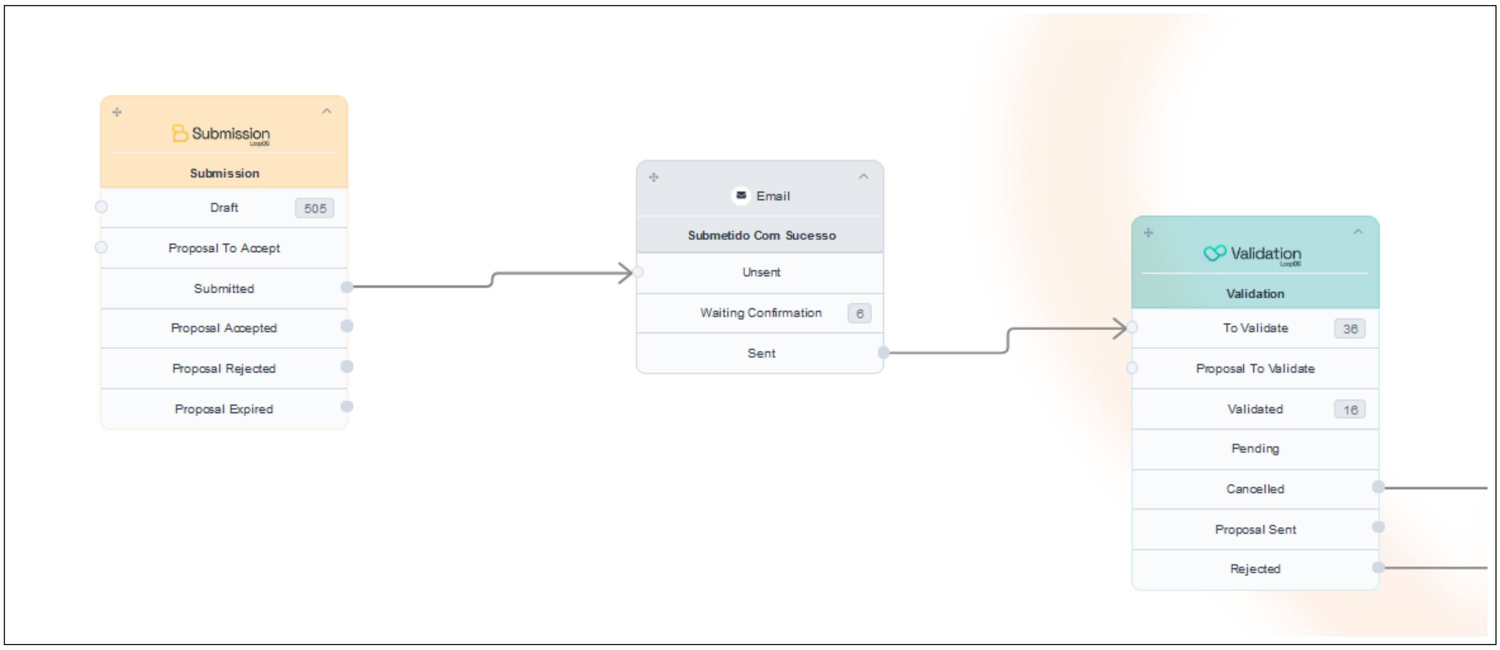
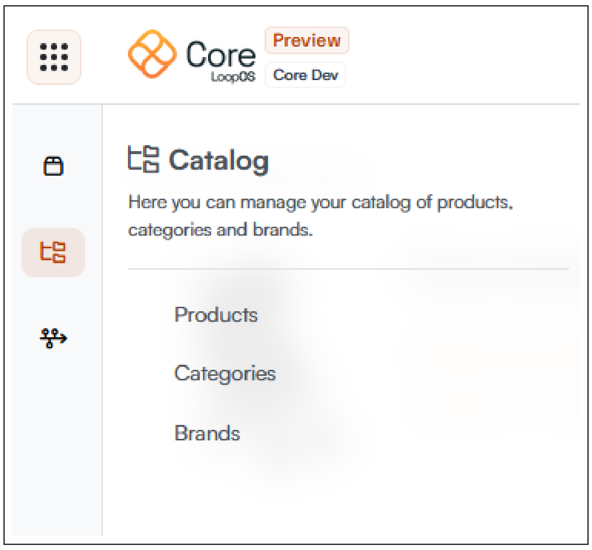


Figure 6.20: Partial LoopOS Flow example.

In the same way a user can create a LoopOS Flow through UI, I can create it through code according to my needs.

Finally, the last concept I wanted to detail is Catalogs, which reside in each Core and help manage the items in their LoopOS applications. They can have Categories, Brands, and Products. Figure 6.21 showcases the Core sidebar with these items.



Here you can manage your catalog of products, categories and brands.

Figure 6.21: LoopsOS Core sidebar.

FNAC, a multinational consumer electronics and entertainment retailer, uses LoopOS to buy used iPhones. For instance, in their Core Application, they have the Category “Smartphone” with the Brand ”Apple” and the Product “IPhone X” so their Submission application recognizes which products they are searching for.

Additionally, each Category and Product may have numerous Protocols. These are sets of questions created by the Partner, so the other linked LoopOS applications (Submission, Validation, Handling, and Hubs) can present them to the user to be filled out.

In the instance of Submission, since it is the customer submitting the item, the protocol could have questions such as “Does the product function properly?” and in the case of Validation, with a worker validating a product, it could have questions such as “Does the item description correspond to the item sent?”. Finally, there are also User Protocols, which also exist in LoopOS Core and are then presented in Submission for the customer submitting the product to fill out, such as name, email, address, and so on.

As we move forward, it is crucial to recognize that LoopOS is a team effort. While I was the developer of the LoopOS Onboarding application, it wouldn’t have been feasible without the extensive groundwork laid by others before. From establishing methods within App Instances that paved the way for me to deploy the application to the creation of LoopOS Flows to link them all.

Thus, with all this context, it is finally possible to understand how and why I created the boilerplate script in the following way:

The *run_boilerplate* method serves as the starting point for the script, it begins by receiving various parameters including the log object, user information, partner details, and partner group information. Then, retrieves the Submission App Instance belonging to the CTT Partner Group.

Next, the script identifies the latest stable App Release associated with LoopOS Core and uses it to create an App Instance linked to the Partner. The instance's state is set to ":to_create", initiating the *SyncInfrastructureJob* workflow. As described in 2, Sub-section 2.4.1, this job deploys the instance in a Kubernetes Pod and transitions its state to "active". This process is replicated for the creation of a LoopOS Handling App Instance linked to the Partner.

The script then creates a User Group specifically for admin users. Any user associated with this group gains admin privileges for the partner apps. This ensures authorized individuals can manage and maintain the partner's applications effectively.

Once both applications are fully deployed, the script initiates the preparation of the Core. This process involves remotely executing code, as outlined in 2, Sub-section 2.4.1. Initially, all services associated with the Partner Group are retrieved, encompassing email providers, email templates, and incoming payment providers. This ensures that the necessary components are readily available for creating the desired flow. The script then encapsulates each code intended for execution by the Core into a string variable, running them one by one.

Firstly, the *flow_code* variable encapsulates the code that constructs the desired LoopOS Flow. Starting with the block formation, the script devises, defines and connects the following blocks in this order:

1. **Submission:** Handles the initial item submission by the client.
2. **Submitted Email:** Sends a confirmation email to the client acknowledging their submission.
3. **Incoming Payment:** Waits for the return payment to be made.
4. **Email Paid**: Informs the client that their payment was received.
5. **Transport:** Handles the process from shipping instructions to updating the item status for display on Handling.
6. **Handling:** Facilitates item acceptance by the Partner.
7. **Email Handling:** Informs the client that their item was returned and accepted.

Upon successful execution of the *flow_code* variable, the *catalog_code* variable is created. This variable contains simple code that establishes a default category and a default product. After this code's successful execution, the *protocol_code* variable is created, encompassing a questionnaire that the user will submit to provide details about their return. These specific questions were agreed upon in a prior meeting with CTT:

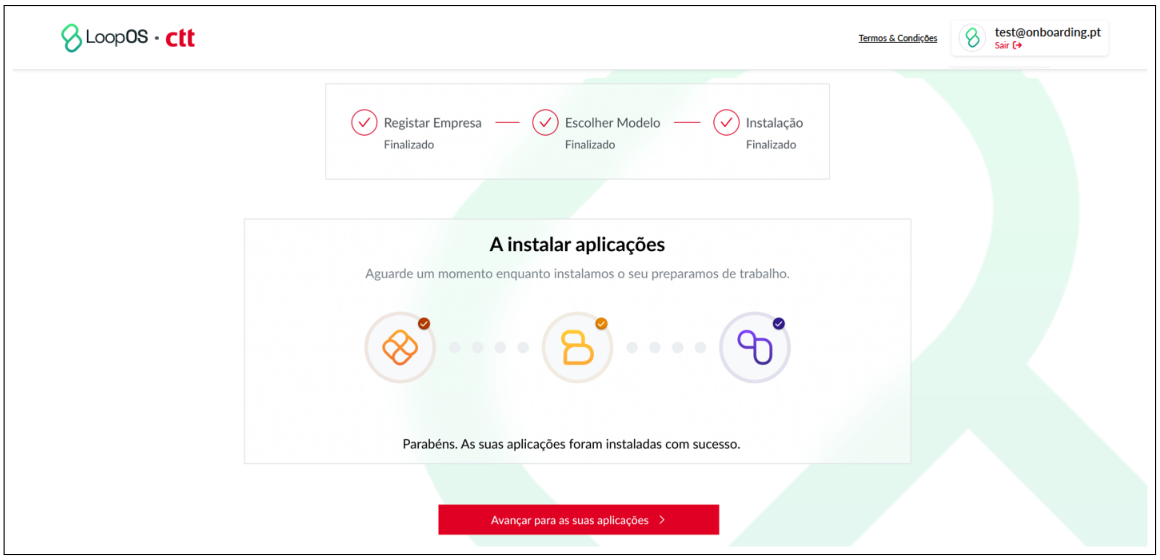
1. What is the order number?
2. When was the order placed?
3. Upload a copy of the receipt.
4. Please describe the items being returned.
5. Select the reason for the return. The possible reasons are:
    - Incorrect size
    - Defective product
    - Delayed delivery
    - Product different from website image
    - Incorrect product received
    - Damaged product during delivery
    - Purchase of the same item in different sizes

Afterward was the creation of *user_protocol_code*. Following a similar logic as before, the agreed-upon details the user had to fill in were the full name, the email address, the phone number, the email address, its Fiscal Identification Number (NIF), and two fields for the user to agree to share their personal data with LoopOS and CTT to return their item and to agree to the terms and conditions of the return service.

Finally, after successfully remotely executing the *user_protocol_code*, the script changed the user state to "active", and so, the Partner has his environment setup complete, from just choosing, at the beginning of the Boilerplate Choice Page, which boilerplate he wanted to execute.

However, when the user chooses a boilerplate, he does not remain on the Boilerplate Choice Page while he waits for the boilerplate completion. Instead, he is directed to the Wait Page, upon executing the endpoint to trigger the boilerplate, a Log object is created and sent along with the other attributes to the boilerplate and kept updated throughout its execution. The log's ID is also sent in the request to trigger the endpoint in the response to LoopOS Onboarding.

To give the user a sense of the boilerplate's progress, a job on LoopOS Onboarding triggers an endpoint in the Network Manager every 5 seconds. This endpoint requests the log's information by sending the aforementioned ID and retrieving the corresponding message, such as "Your applications are being deployed..." and the percentage to completion. The Onboarding Application then dynamically updates a progress bar on the Wait Page to reflect the script's progress. Figure 6.22 illustrates this behavior by showcasing the final look of the Wait Page in LoopOS Onboarding for CTT Returns.



A instalar aplicações

Aguarde um momento enquanto instalamos o seu preparamos de trabalho.

Parabéns. As suas aplicações foram instaladas com sucesso.

Figure 6.22: LoopOS Onboarding: Final look of the Wait Page.

Once the boilerplate is complete and the Wait Page reaches 100% completion, a button is displayed for the Partner. Upon clicking this button, the Partner is seamlessly guided to their User profile page within the Network Manager. From here, they can easily access and navigate their newly created applications.

◆ Conclusion

In conclusion, the LoopOS Onboarding application, with a focus on the CTT client, was developed with a meticulous and well-considered approach to ensure seamless integration of various functionalities. From user authentication and application deployment to the creation of dynamic pages and the creation and execution of a boilerplate script, each step was carefully designed and implemented to streamline the onboarding process for Partners.

This new application can generate a fully functional LoopOS environment with a single button press. It stands as a significant milestone in aiding in solving the problem of the manual onboarding of partners as well as the manual creation of their environments.

The implementation of the boilerplate script, as described in the provided sections, demonstrates a thoughtful understanding of the inner workings of the LoopOS framework, the partner needs, and strategic execution to fulfill agreed-upon requirements.

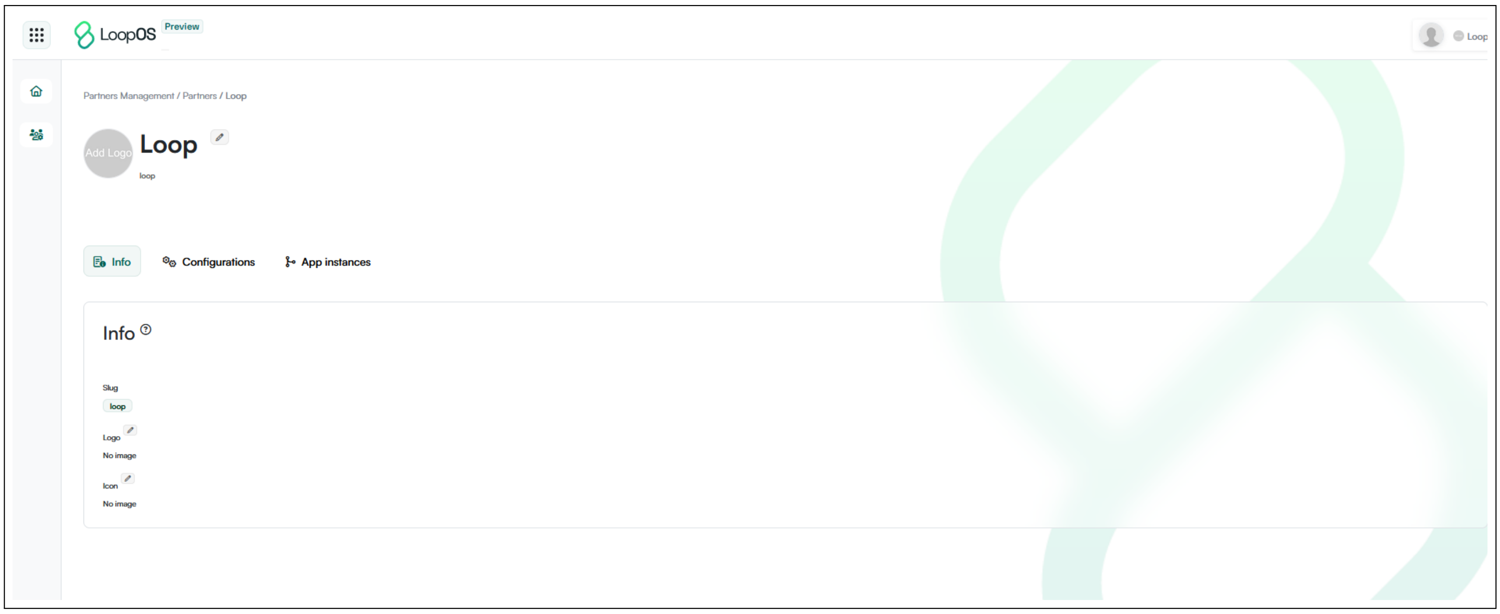
For a comprehensive evaluation of the LoopOS Onboarding application's performance and functionality, including testing conducted and an analysis of requirement completion, please refer to Chapter 7 - Testing.

The subsequent section will delve into the planning and development of the LoopOS UI, providing insights into the new User Experience (UI) design and considerations that contribute to an enhanced overall user experience.

6.2 LoopOS UI

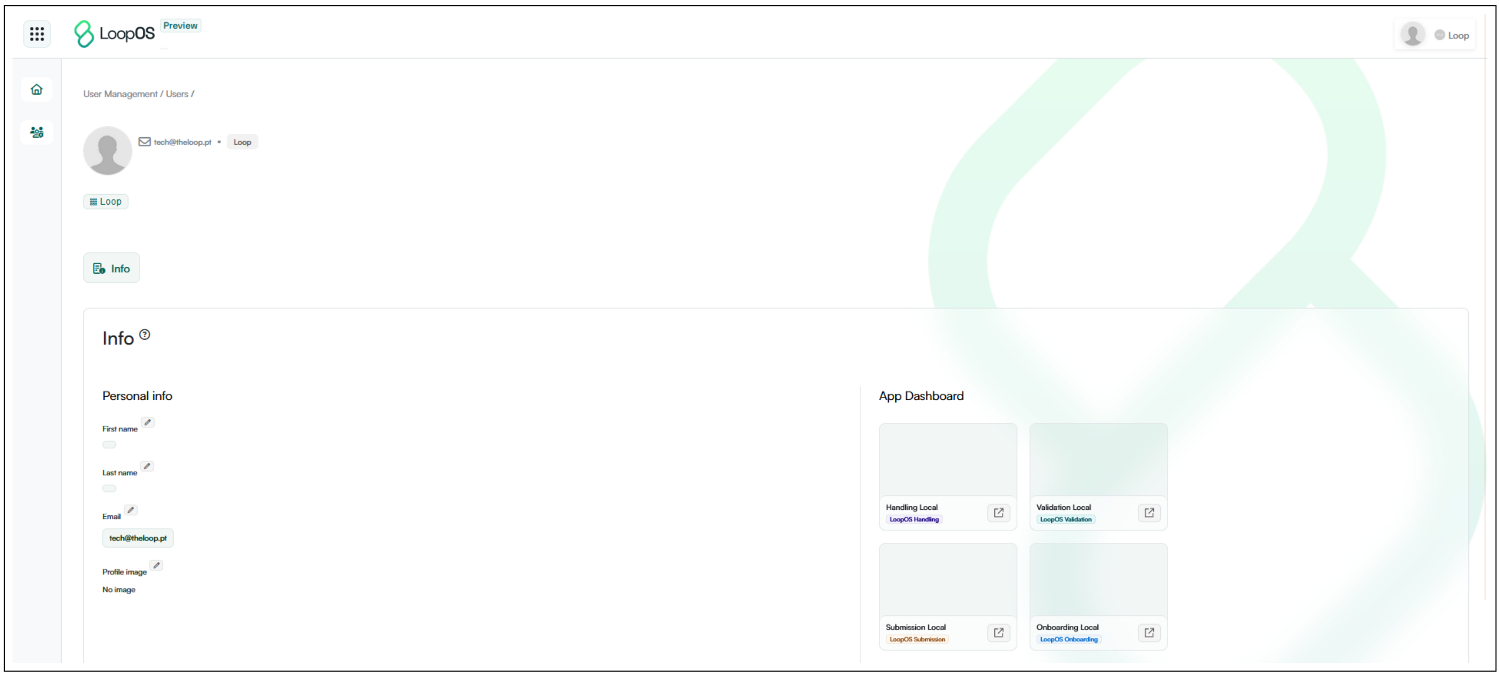
Upon successfully completing the onboarding process, the Partner is redirected to the Network Manager, where they can access their applications. As depicted in the screenshots in Chapter 2, the User Experience (UI) was developed entirely using AVO [4], a powerful and flexible admin interface gem for Ruby on Rails applications. AVO enables developers to focus on the project functionalities rather than spending excessive time on front-end development. Given the project's Agile methodology, AVO's ease of implementation proved instrumental in hastening the front-end development for all LoopOS apps. However, AVO was always considered a temporary solution, as The Loop Co. envisioned a unique visual identity and user experience for the LoopOS ecosystem.

While working on the LoopOS Onboarding, efforts were directed toward creating simple User and Partner pages to minimize the impact of AVO's usability limitations on the overall user experience. Thus ensuring that the Partners were not overwhelmed by the Network Manager's interface upon initial access. Figures 6.23 and 6.24 are screenshots of the Partner and User profile pages, respectively.



Slug
loop
Logo
No image
Icon
No image

Figure 6.23: Network Manager: Partner profile page.



Personal info
First name
Last name
Email
tech@theloop.pt
Profile image
No image

Figure 6.24: Network Manager: User profile page.

However, since these pages were the only pages implemented and had only basic features, the company decided the development of the new Network Manager UI took priority. Hence, after the LoopOS Onboarding Application was concluded, the planning and development of the new front-end for the Network Manager started.

◆ Planning

The planning process started with thoroughly examining the current and future designs of LoopOS applications by analyzing the mockups of each app. Upon recognizing that many applications shared common components or entire layouts, the objective of this thesis was expanded beyond simply developing the new UI for the Network Manager. For instance, Figures 6.25 and 6.26 illustrate the resemblances between the mockups for the Network Manager Show user page and the Core show item page, respectively.

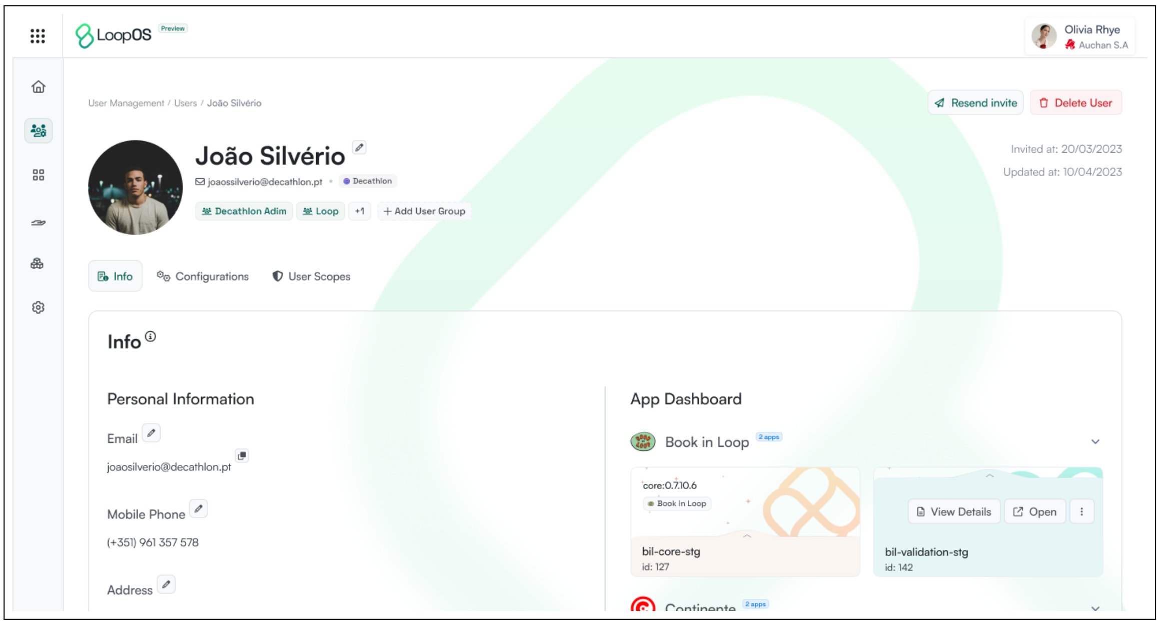
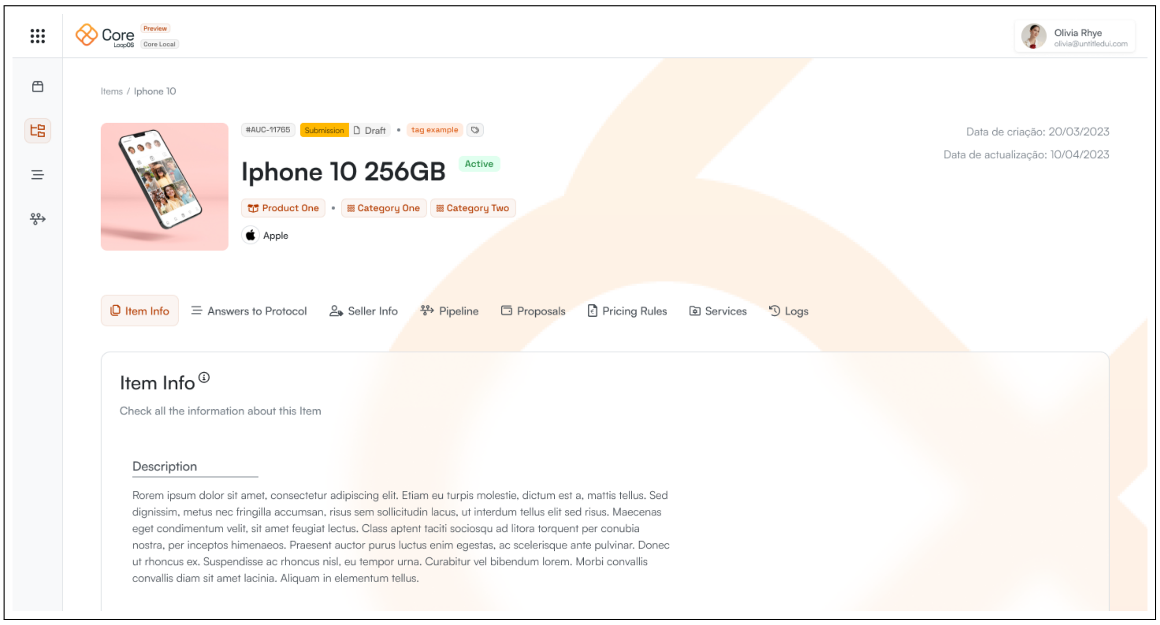


Figure 6.25: Network Manager: Show User Page - Mockup.



Rorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eu turpis molestie, dictum est a, mattis tellus. Sed dignissim, metus nec fringilla accumsan, risus sem sollicitudin lacus, ut interdum tellus elit sed risus. Maecenas eget condimentum velit, sit amet feugiat lectus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Praesent auctor purus luctus enim egestas, ac scelerisque ante pulvinar. Donec ut rhoncus ex. Suspendisse ac rhoncus nisl, eu tempor urna. Curabitur vel bibendum lorem. Morbi convallis convallis diam sit amet lacinia. Aliquam in elementum tellus.

Figure 6.26: LoopOS Core: Show Item Page - Mockup.

Thus, instead of just implementing the new UI for the Network Manager, I would create a Rails Engine (LoopOS UI) responsible for maintaining the layout and loading the required assets implemented in all LoopOS applications starting with the Network Manager.

Before outlining this engine's planning, setup, and development, providing context on Rails Engines and Rails Gems is essential.

- **Rails Engines** are modular and reusable components that encapsulate specific features or functionality within a Rails application. They are Rails applications that can be mounted within another Rails application. A way to package and share functionality across different Rails projects. Like Rails applications, they share the same Model-View-Controller pattern. LoopOS UI will, therefore, be able to be mounted inside all LoopOS applications, and by following Continuous Integration/Continuous Delivery using tools such as GitLab, when LoopOS UI is implemented in all apps, any change in the front-end can be done in one place.

- **Rails Gems** encapsulates reusable code that provides additional functionality or features to a Rails application. Gems can include various Ruby elements like classes, modules, and libraries. They are commonly used to distribute reusable code that doesn't necessarily rely on the Rails framework itself. Compared to Rails engines, Gems are generally lighter and more generic, not specifically tailored for mounting within a Rails application.

An example of a Gem developed for and by LoopOS is LoopOS SDK. It is a valuable tool for maintaining consistent UI components across all LoopOS projects. It leverages Storybook, an open-source tool for developing, inspecting, and testing UI components in isolation. When a component is refined in Storybook, it is seamlessly integrated into LoopOS SDK through GitLab CI/CD, ensuring that all applications use the latest version of the gem. With the ongoing development of LoopOS UI, this gem will reside exclusively within the new engine, which will dynamically generate the front end for the app it is coupled with.

The development of the LoopOS UI was split into three crucial phases: Planning, Setup, and Development.

I began by developing a high-level plan to comprehensively grasp what needed to be implemented. One of the initial requirements identified was that this engine would need to seamlessly integrate with other LoopOS applications. This prompted me to meticulously analyze the LoopOS Onboarding, Submission, Handling, Validation, Core, Hubs, and Network Manager mockups.

After a thorough examination, I concluded that the LoopOS Onboarding, Submission, Handling, and Validation mockups were either outdated or diverged in fundamental aspects. However, they did share certain components like buttons, tags, and input elements, which were sourced from the LoopOS SDK. Hence, to alleviate computational resources from each application, instead of these components being managed by the SDK and the application, they could be exclusively handled by the LoopOS UI.

In contrast, the Network Manager, the Core, and the Hubs mockups exhibited enough resemblance for me to create Macro and Micro layouts. Macro layouts describe the larger, page-wide organization of the application. Micro layouts, however, deal with the specific placement and arrangement of individual elements within these larger sections. It is possible to split the Network Manager, the Core, and the Hubs into three Macro layouts: Topbar, Sidebar, and Panel. Figures 6.27, 6.28 and 6.29 illustrate the Macro layouts in each of these applications.

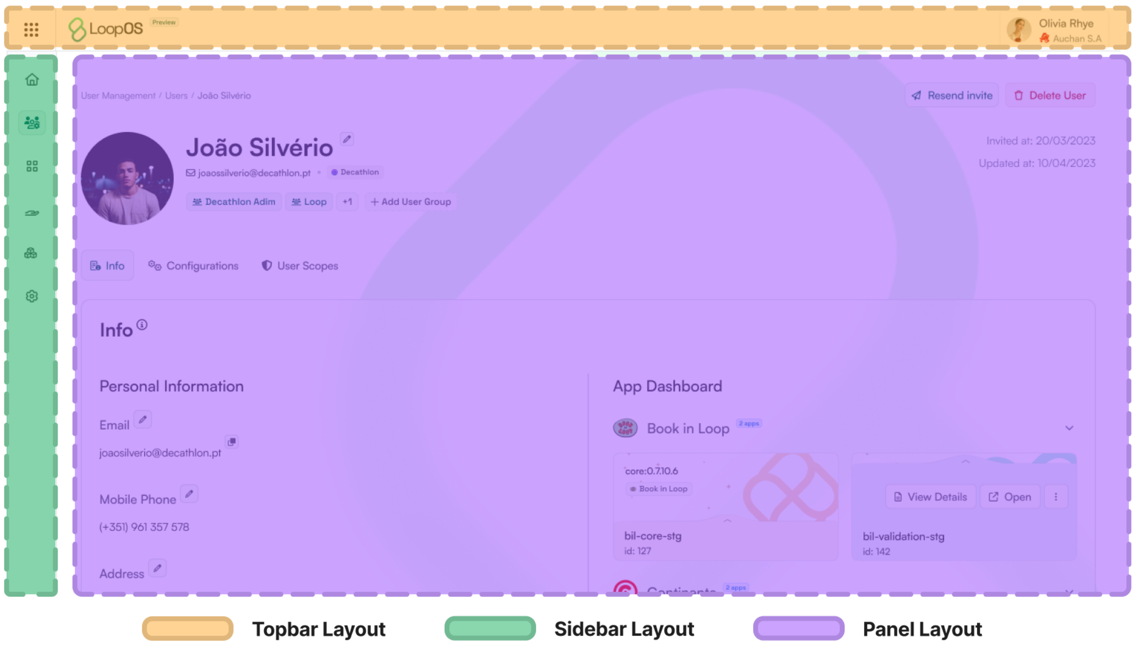


Figure 6.27: Network Manager: Show User Page - Macro Layouts.

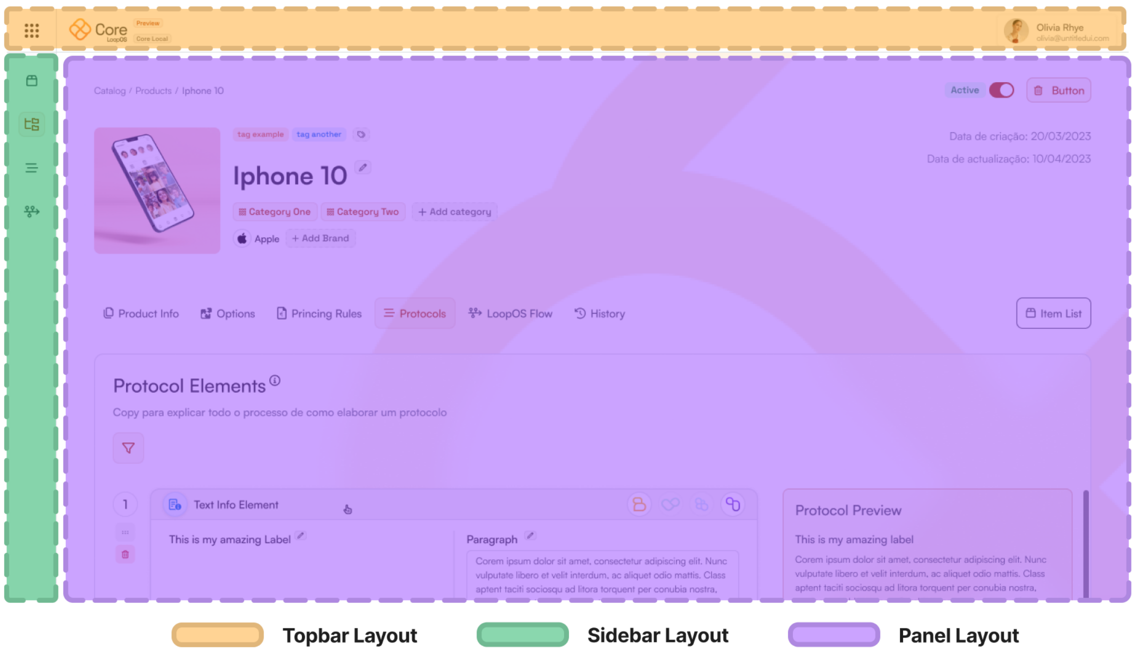


Figure 6.28: LoopOS Core: Show Item Page - Macro Layouts.

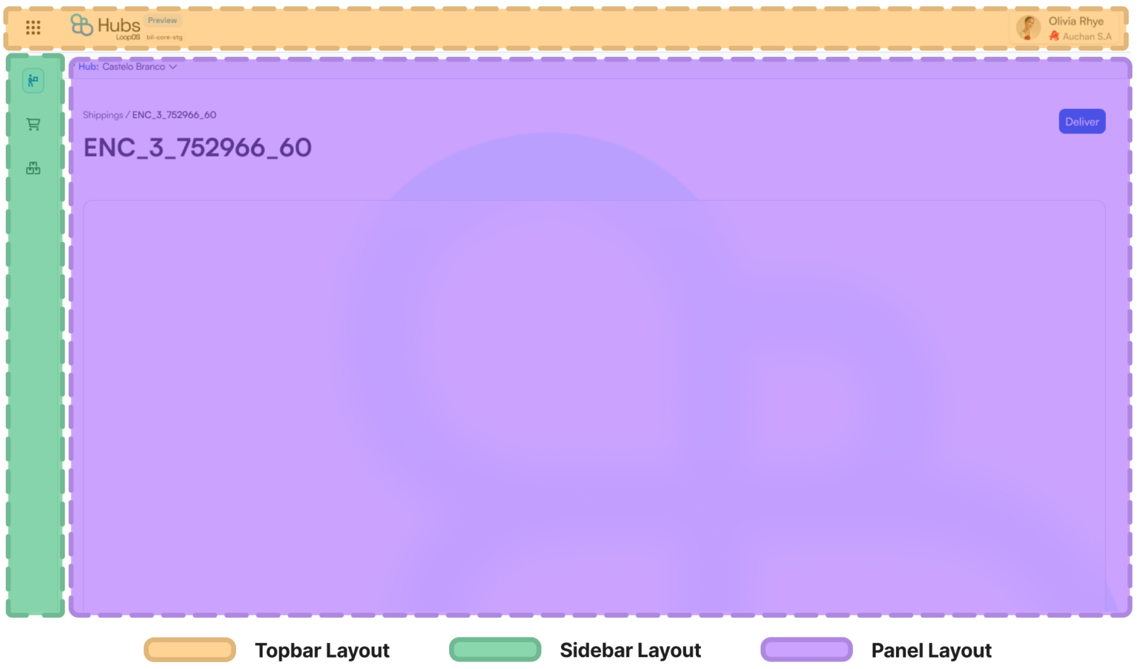
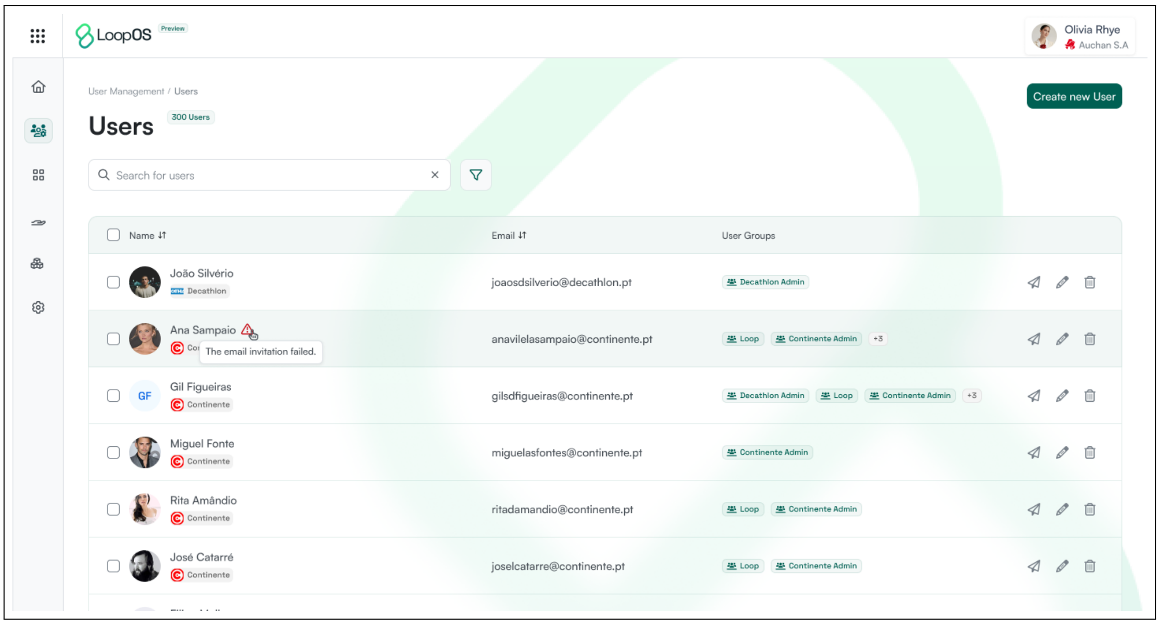


Figure 6.29: LoopOS Hubs: Show Shipping Page - Macro Layouts.

As for the **Topbar Layout**, since it is hosted by the Network Manager for other applications, it should remain outside the new Engine and continue to be maintained by the Network Manager. This is to ensure easier maintenance and enable quick rectification of authentication-related issues. The Engine will, therefore, have a Topbar section, but it will only render what the Network Manager sends.

The **Sidebar Layout**, the primary navigation component for all LoopOS applications, should be implemented using a builder method that takes a hash of items as input. Each item in the hash represents a section in the sidebar, and each section can encompass multiple menu items or sub-sections (dropdowns/drawers). The Core and the Hubs each already have a different implementation of the sidebar layout, so this needs to be considered when creating this builder.

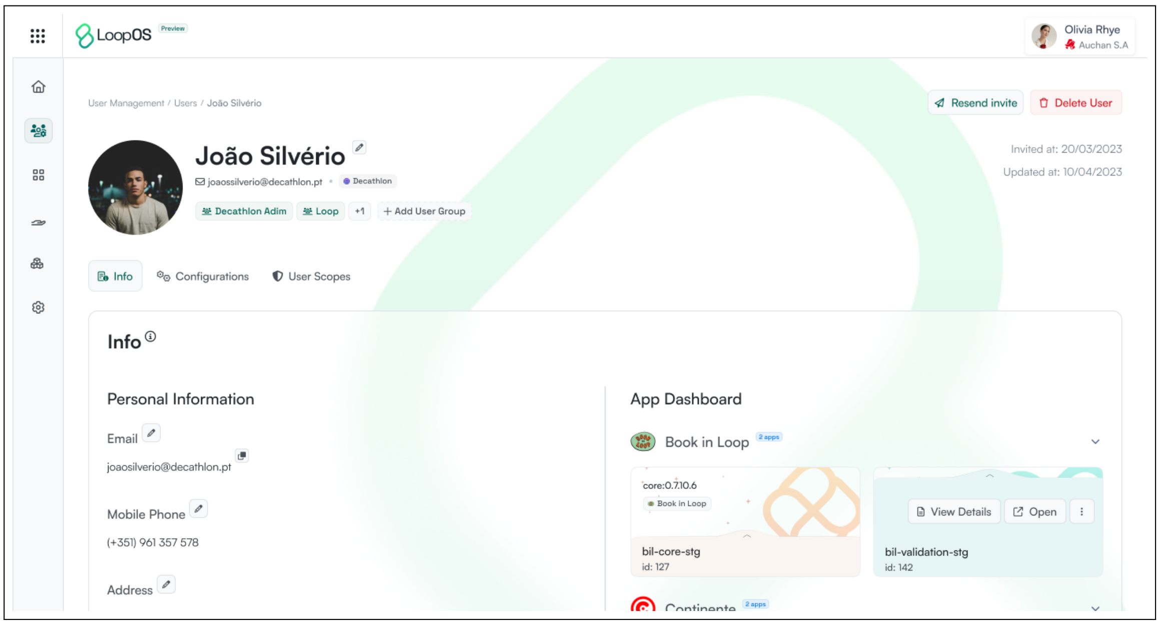
In contrast to the other two layouts, the **Panel Layout** can be further divided, thus giving rise to what I refer to as micro layouts (layouts within layouts): the Index Layout, which displays all the items belonging to that page, and the Show Layout, which showcases a specifically selected item. Figures 6.31 and 6.30 showcase the mockups of the user show and index page of the Network Manager.



Users 300 Users

Name	Email	User Groups
João Silvério Decathlon	joaosdsilverio@decathlon.pt	Decathlon Admin
Ana Sampaio Co[The email invitation failed.]	anavilelasampaio@continente.pt	Loop, Continente Admin, +3
Gil Figueiras Continente	gilsdfigueiras@continente.pt	Decathlon Admin, Loop, Continente Admin, +3
Miguel Fonte Continente	miguelasfontes@continente.pt	Continente Admin
Rita Amândio Continente	ritadamandio@continente.pt	Loop, Continente Admin
José Catarré Continente	joselcatarre@continente.pt	Loop, Continente Admin

Figure 6.30: Network Manager: User Index Page - Mockup.



João Silvério
joaossilverio@decathlon.pt • Decathlon

Invited at: 20/03/2023
Updated at: 10/04/2023

Info

Personal Information

Email
joaosilverio@decathlon.pt

Mobile Phone
(+351) 961 357 578

Address

App Dashboard

Book in Loop 2 apps

core:0.7.10.6
Book in Loop
bil-core-stg
id: 127

bil-validation-stg
id: 142

Continente 2 apps

Figure 6.31: Network Manager: User Show Page - Mockup.

These layouts must be crafted in the LoopOS UI first and implemented within the Network Manager second, following a phased development analogous to the LoopOS Onboarding process to align with the agile approach adopted for this project. However, instead of progressing page by page, this time, the focus will be on implementing layout by layout, component by component. The development and implementation were then divided into five distinct phases. Each phase is connected with a diagram illustrating the progress of the engine:

1. Macro Layout implementation:

Create Macro Layouts with yields inside the LoopOS UI, have the Network Manager call these layouts, and send the full content for each part - Figure 6.32.

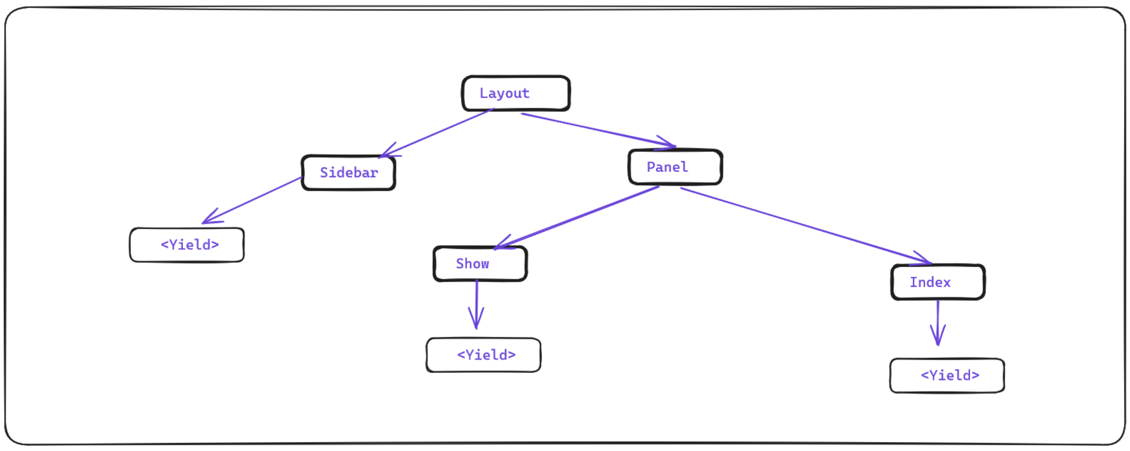


Figure 6.32: LoopOS UI progressive development planning: Macro Layout implementation

2. Sidebar Layout Implementation & Refinement:

Improve Sidebar Layout inside LoopOS UI, so it generates the sidebar depending on the attribute hash it is sent and have the Network Manager call this new builder - Figure 6.33.

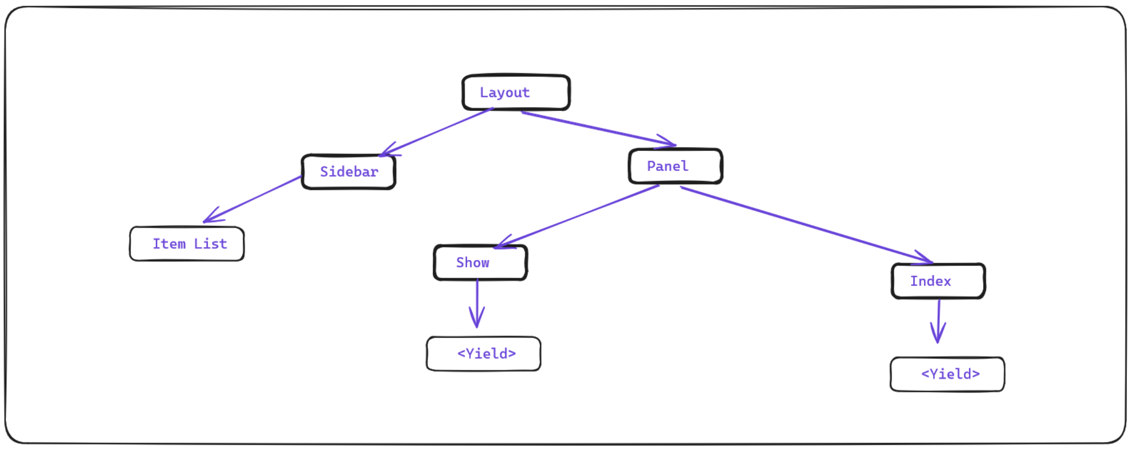


Figure 6.33: LoopOS UI progressive development planning: Sidebar Layout Implementation & Refinement.

▍3. Show Layout - Micro Layout Implementation:

In this phase, I will split the Show Layout layout into two Header and Content as illustrated by Figure 6.34, which is common in all Show Pages across the three applications mentioned before.

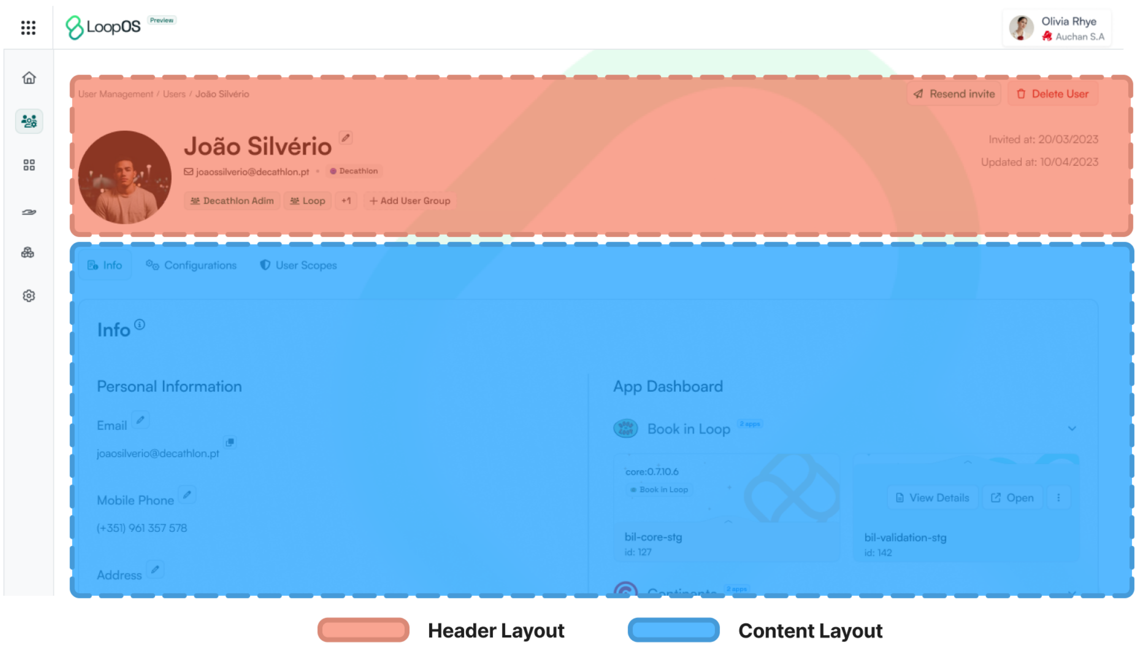
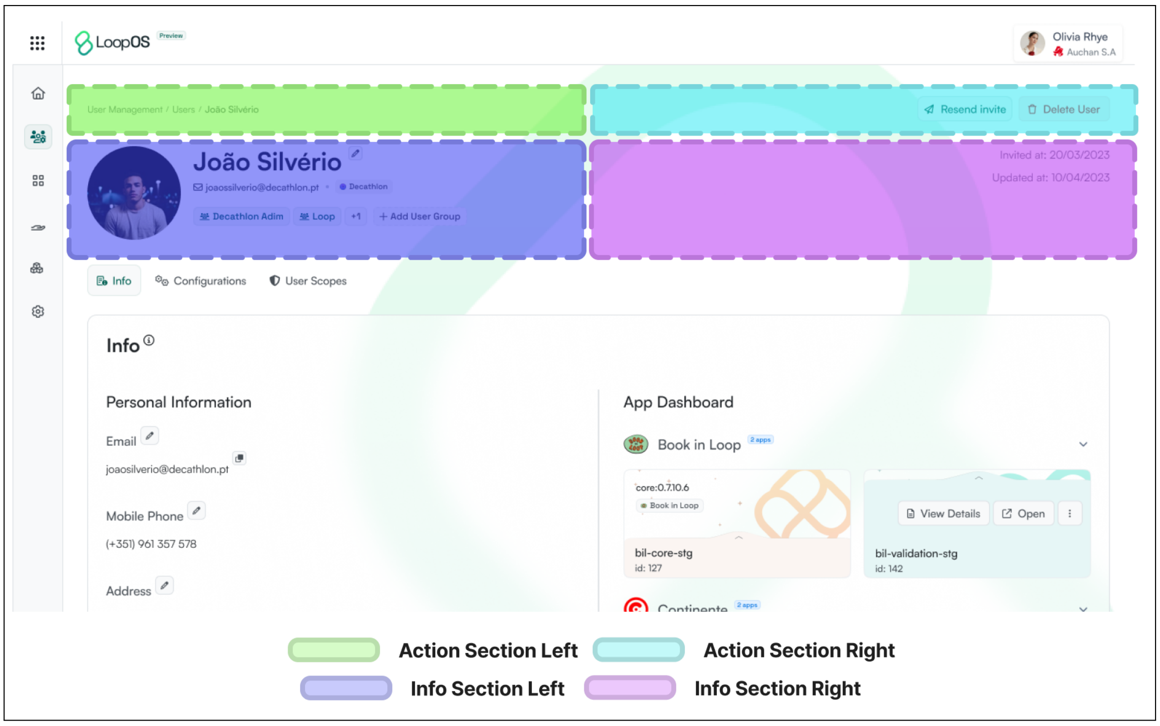


Figure 6.34: Network Manager: Show User Page split into Content and Header.

However, I will further split the Header into four: Action Section Left, Action Section Right, Info Section Left, and Info Section Right. Figure 6.35 visually represents how the header is divided in the Show User Page. The Action Sections are designed to accommodate actionable elements, while the Info Sections are intended for informational components. The “left” and “right” designations indicate their respective positions within the header. It is worth mentioning that this division was carefully considered in conjunction with the Show Pages of other applications, ensuring a consistent header design across apps while allowing flexibility in the specific components within each section.



Info

Personal Information

Email

joaosilverio@decathlon.pt

Mobile Phone

(+351) 961 357 578

Address

App Dashboard

Book in Loop 2 apps

core:0.7.10.6

Book in Loop

bil-core-stg
id: 127

View Details Open

bil-validation-stg
id: 142

Continente 2 apps

Figure 6.35: Network Manager: Show User Page Header split into Action Section and Info Section.

Hence, this phase will entail the creation of these layouts in LoopOS UI with yields and have the Network Manager send in the full content - Figure 6.36.

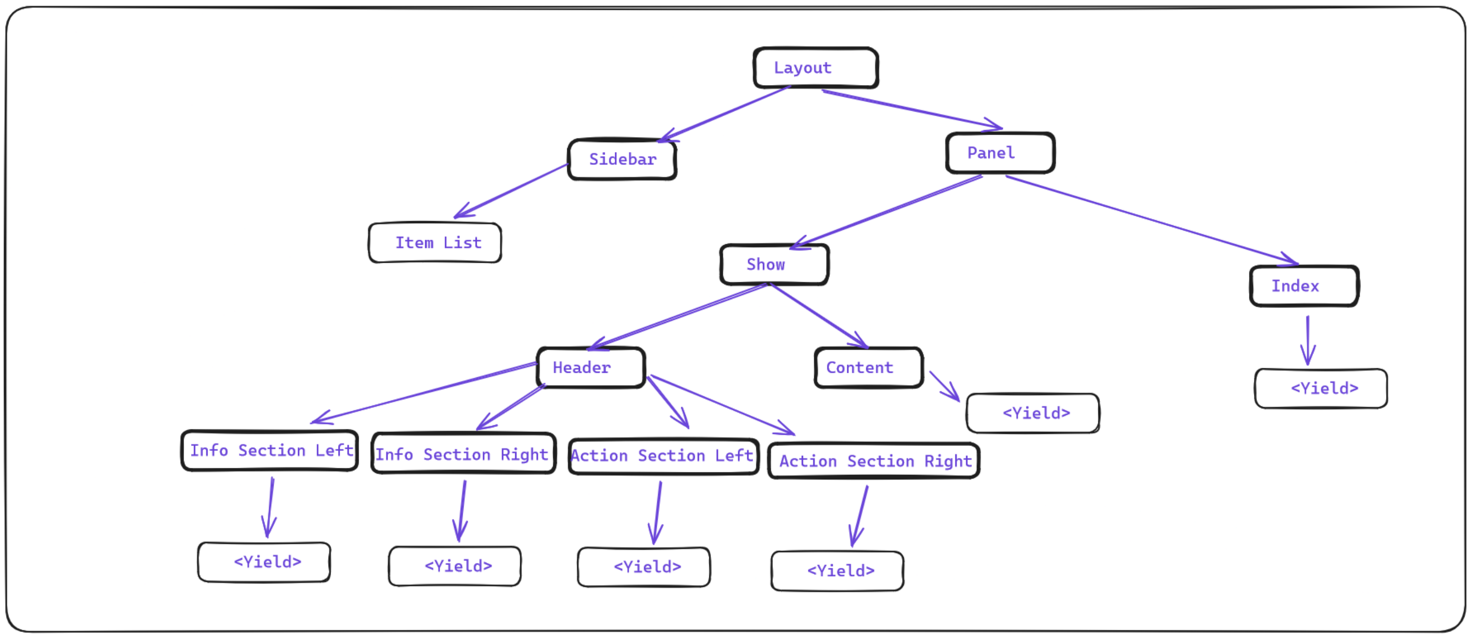


Figure 6.36: LoopOS UI progressive development planning: Micro layout implementation of Show Layout.

4. Show Layout - Micro Layout Refinement:

This phase entails populating the Header and the Content with builders to generate each section. This meant a study of the previously mentioned application common elements within each of the previous sections. Figure 6.37 illustrates an example of this analysis and the common components found on the different show pages of Core, Hubs, and Network Manager mockups.

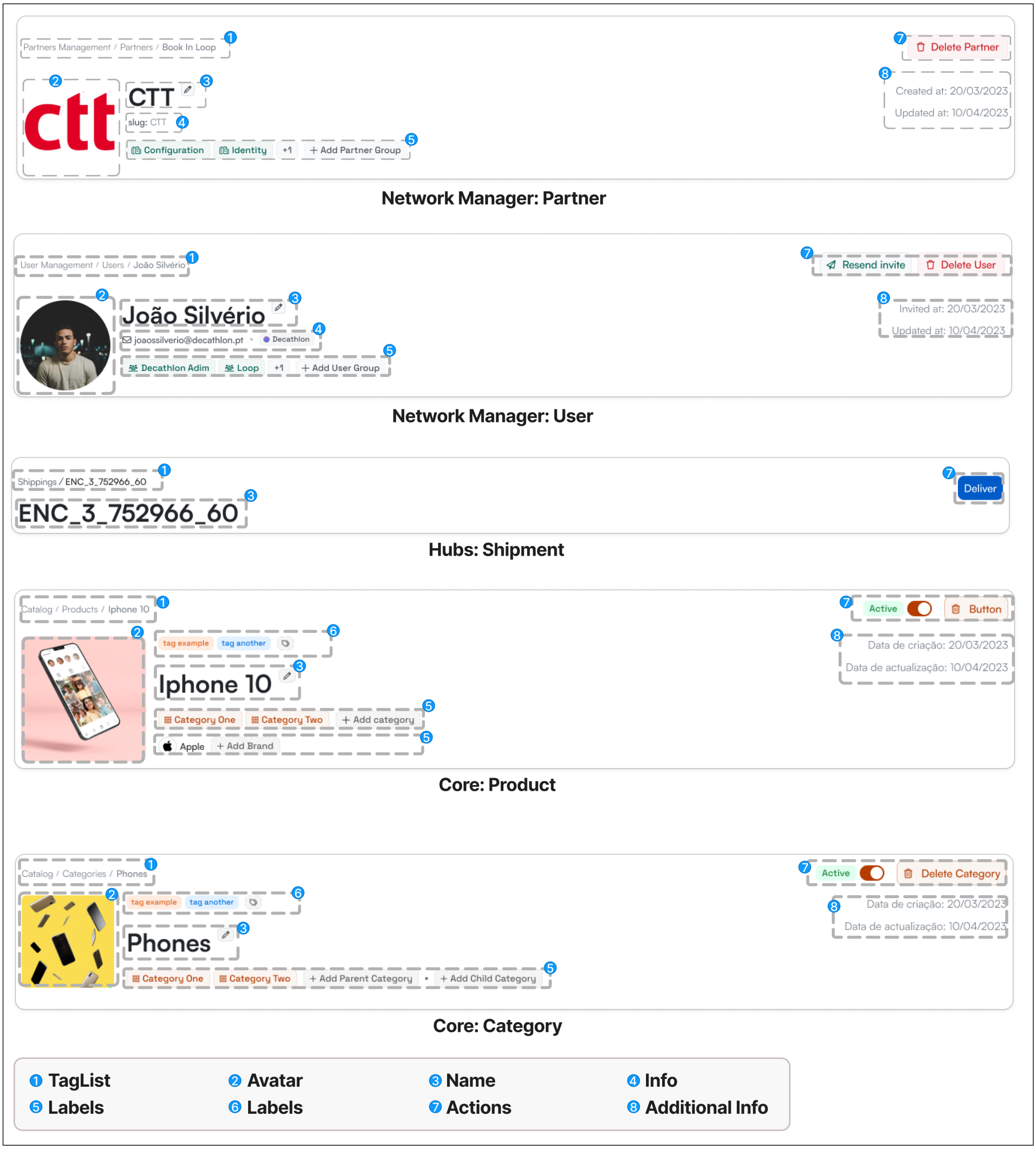


Figure 6.37: Header components across applications.

Consequently, these were the conclusions for the current mockups:

Actions Section Left: will only have breadcrumbs. Breadcrumbs are a navigational aid that helps users understand their location within a website or application. These can be constructed with a builder taking in a hash variable, as seen in Listing 6.8.

It should be noted that this code snippet serves to make the reader accustomed to the logic behind the LoopOS UI implementation. In this case, the Network Manager sends an array with the breadcrumbs hash containing its item to the *action_section_left*. This will create a **View Component** of *ShowLayoutComponent* type, with a *ActionSectionLeftComponent* View Component inside.

In turn, Listing 6.9 showcases a code snippet, where it will receive the payload, iterate it, check breadcrumbs inside the hash, and render it correspondingly. This way, since each object has its corresponding View Component, each object will know how it is rendered.

Although it may sound confusing now, later on, I will explain how View Components work and how applying them properly allowed for smart and quick development of the LoopOS UI.

```
action_section_left_payload = [
  {
    breadcrumbs: [
      { name: "Partners Management", path: "", },
      { name: "Partners", path: partner_index_path, },
      { name: "#{@partner.name}", path: partner_show_path
        , },
     ],
  },
]
<%= render LooposUi::ShowLayoutComponent.new do |
   component| %>
  <% component.header_action_section_left(payload:
     action_section_left_payload)
  ...
<% end %>
```

Listing 6.8: Network Manager: Possible Breadcrumb Implementation.

```
module LooposUi
  class ShowLayoutComponent < ViewComponent::Base
    renders_one :action_section_left, "
       HeaderActionSectionLeftComponent"
    #rendering of other View Components

    def initialize()
    end
```

```
      class HeaderActionSectionLeftComponent <
         ViewComponent::Base
        #class public and private methods
       end

      #class declare of other View Components

    end
  end
```

Listing 6.9: LoopOS UI: Possible Breadcrumb Implementation.

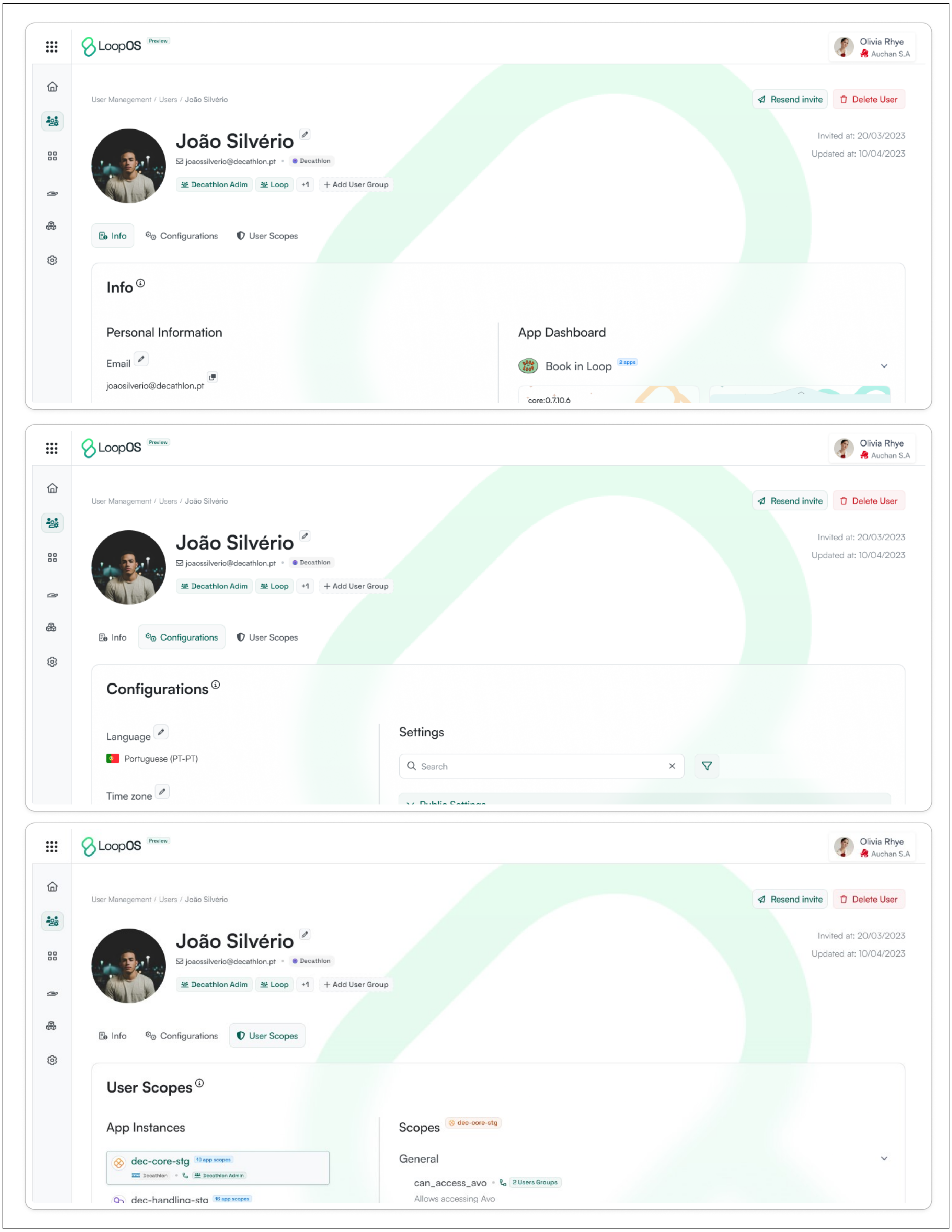
Actions Section Right: will have actions such as buttons and toggles.

Info Section Right: will have simple text with the desired information. For now, containing the information for when it was created and when it was updated.

Info Section Left: The most informative part of the header, typically containing the following elements:

- **Avatar:** An image representing the entity, either in a circular or square format, allowing each application to customize its appearance.
- **Top Tag:** An optional tag that can be displayed above the title, specifically supported by the Core application but not applicable to the Network Manager.
- **Title:** A descriptive title of the entity, with an optional editable property for applications that require it.
- **Additional Details:** Supplementary text providing specific information relevant to the entity, particularly useful for the Network Manager.
- **Taglines:** A group of taglines representing associations or relationships to other entities. For instance, a User in the Network Manager could have taglines indicating their associated User Groups, while a Product in the Core could have taglines for Brands and Categories. While the Network Manager only supports a single tagline, the Core allows for two, which should be considered during implementation.

The Content section of the Show Page incorporates tabs to enhance the presentation and organization of information. These tabs encapsulate specific details and actions related to the corresponding model the page represents. Figure 6.38 showcases the potential tabs that a User Show page mockup might feature, including the "Info" tab for displaying user information that can be edited, along with connected applications, and the "User Scopes" tab for displaying the user's associated App Scopes.



User Management / Users / João Silvério

João Silvério
joaossilverio@decathlon.pt • Decathlon
Decathlon Adim | Loop | +1 | + Add User Group

Invited at: 20/03/2023
Updated at: 10/04/2023

Info

Personal Information
Email
joaosilverio@decathlon.pt

App Dashboard
Book in Loop 2 apps
core:0.7.10.6

Configurations

Language
Portuguese (PT-PT)
Time zone

Settings
Search
Public Settings

User Scopes

App Instances
dec-core-stg 10 app scopes
Decathlon • Decathlon Admin
dec-handling-stg 16 app scopes

Scopes dec-core-stg
General
can_access_avo • 2 Users Groups
Allows accessing Avo

Figure 6.38: Network Manager: User Show Page Tabs Mockups.

Possible tabs differ greatly not only from Application to Application but also from within. For example, Partner tabs and User tabs are not the same. Hence, the LoopOS UI will take the content of each page and render it. This could be implemented by the View Component responsible for the tabs, which receives an array of hashes for each tab. An example of a possible implementation is in the code snippet Listing 6.10, where each hash represents a tab and includes its name, icon, and code content (partial).

```
tabs_payload: [
  {name:  "Info", icon: "fa-info", partial: user_info},
  {name:  "Configurations", icon: "fa-gears", partial:
     user_configurations},
  {name:  "User Scope", icon: "fa-shield", partial:
     user_scopes},
],

<%= render LooposUi::ShowLayoutComponent.new do |
   component| %>
  ...
  <% component.header_action_section_right ....
  <% component.content_tabs(payload: tabs_payload)%>
<% end %>
```

Listing 6.10: Network Manager: Possible Tabs Implementation.

To conclude the planning of the Show Layout, Figure 6.39 illustrates a diagram of the progress at the end of this step.

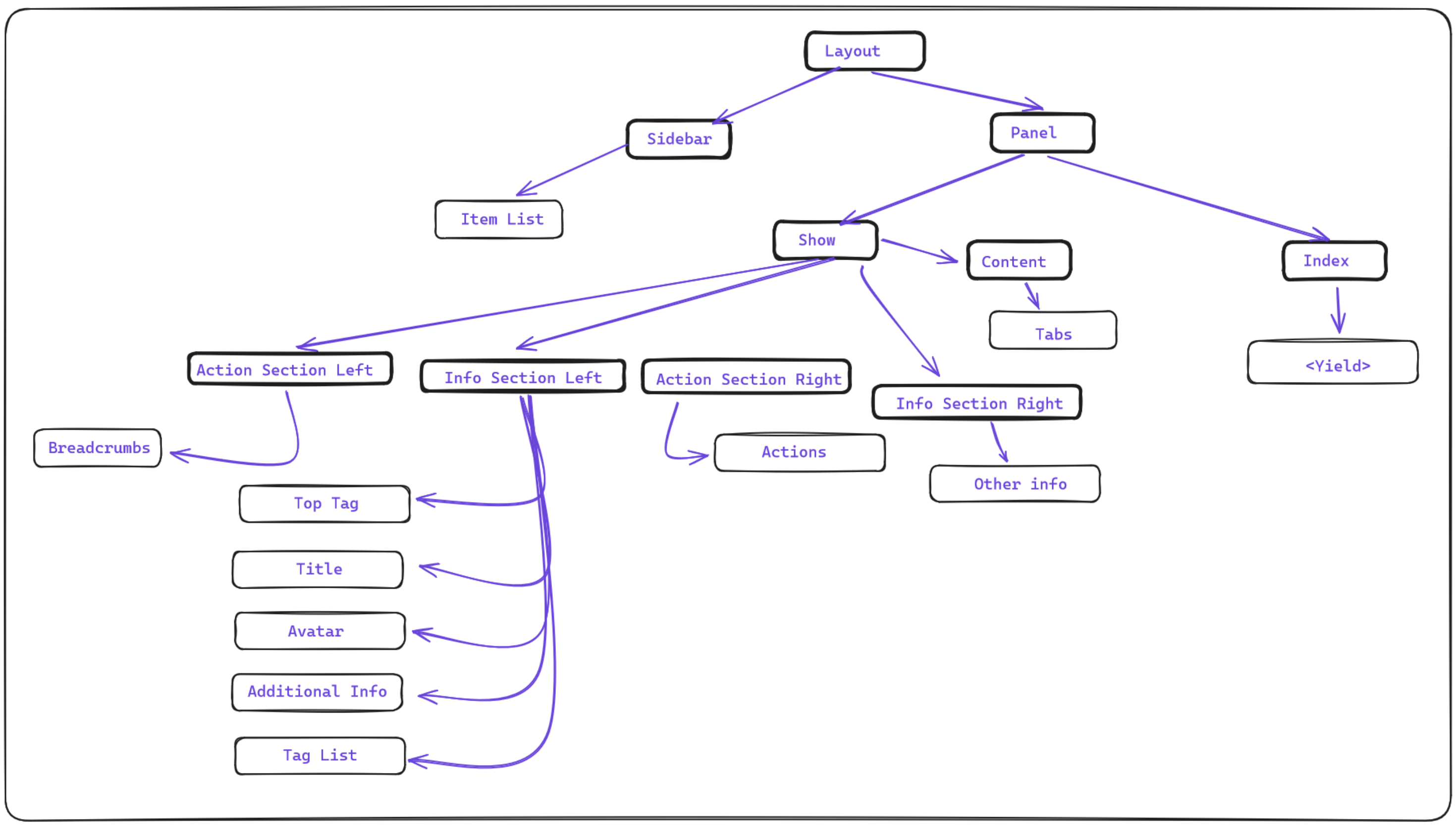
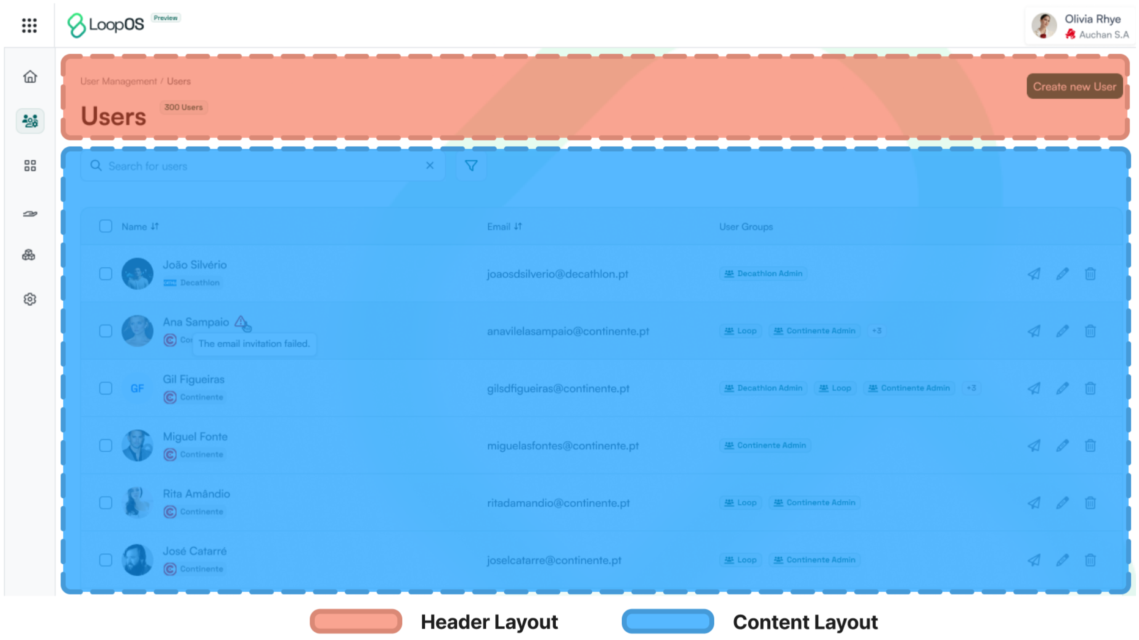


Figure 6.39: LoopOS UI progressive development planning: Micro layout refinement of Show Layout.

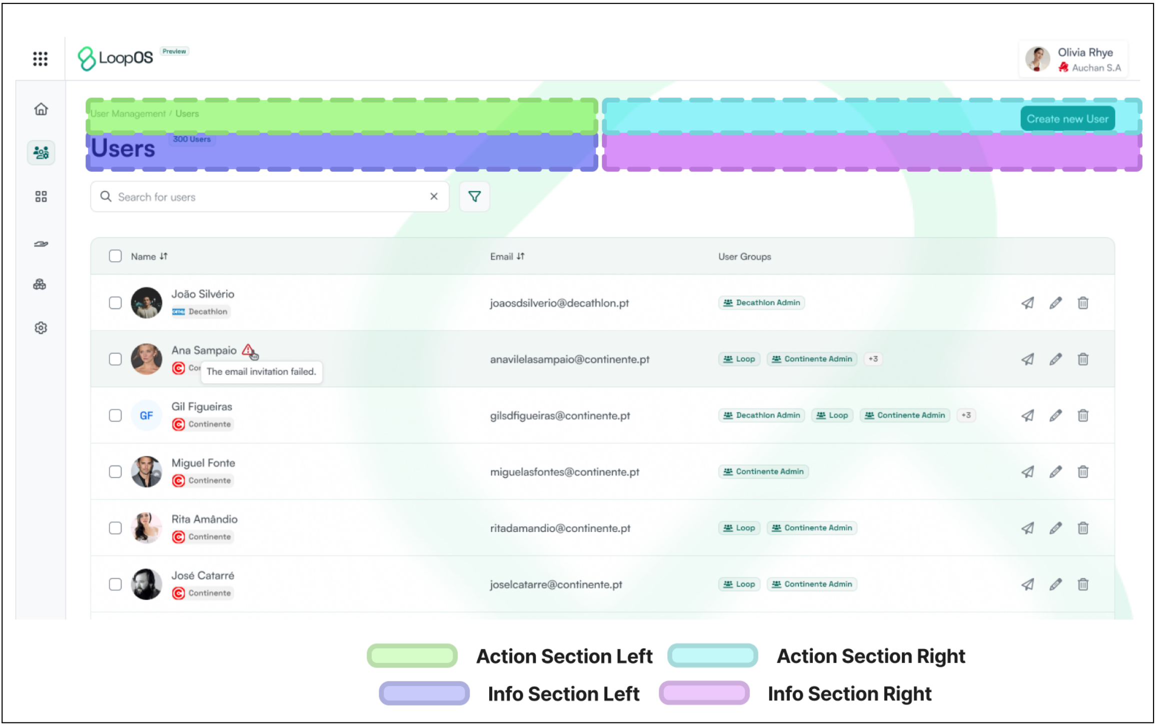
5. Index Layout - Implementation & Refinement: Similar to Step 3 (Show Layout - Micro Layout Implementation), the same process was put into practice by first conducting a study on the Index page mockups for the three apps. I concluded that similarly to the Show page, it could be split in two: Header and Content, as is exemplified in Figure 6.40.



Name	Email	User Groups
João Silvério Decathlon	joaosdsilverio@decathlon.pt	Decathlon Admin
Ana Sampaio The email invitation failed.	anavilelasampaio@continente.pt	Loop, Continente Admin, +3
Gil Figueiras Continente	gilsdfigueiras@continente.pt	Decathlon Admin, Loop, Continente Admin, +3
Miguel Fonte Continente	miguelasfontes@continente.pt	Continente Admin
Rita Amândio Continente	ritadamandio@continente.pt	Loop, Continente Admin
José Catarré Continente	joselcatarre@continente.pt	Loop, Continente Admin

Figure 6.40: Network Manager: Index User Page split into Content and Header.

Thus, the process would be the same. Furthermore, similarly to the Show Page, the header could be split in the exact same way. Since I could use the same developed components from before, i could immediately apply the header using LoopOS UI. Figure 6.41 showcases the header split similar to the Show page.



Name	Email	User Groups
João Silvério Decathlon	joaosdsilverio@decathlon.pt	Decathlon Admin
Ana Sampaio Co The email invitation failed.	anavilelasampaio@continente.pt	Loop, Continente Admin, +3
Gil Figueiras Continente	gilsdfigueiras@continente.pt	Decathlon Admin, Loop, Continente Admin, +3
Miguel Fonte Continente	miguelasfontes@continente.pt	Continente Admin
Rita Amândio Continente	ritadamandio@continente.pt	Loop, Continente Admin
José Catarré Continente	joselcatarre@continente.pt	Loop, Continente Admin

Figure 6.41: Network Manager: Index User Page Header split into Action Section and Info Section.

The Content section varies on the Index Page. Not only does it contain a table with all the data of that model, but it also contains a search input to filter the table. Figure 6.42 illustrates a diagram of the progress at the end of this step, where Index reuses Header components from the Show Page and creates a new Content Component which will create a Table with its corresponding Search Bar.

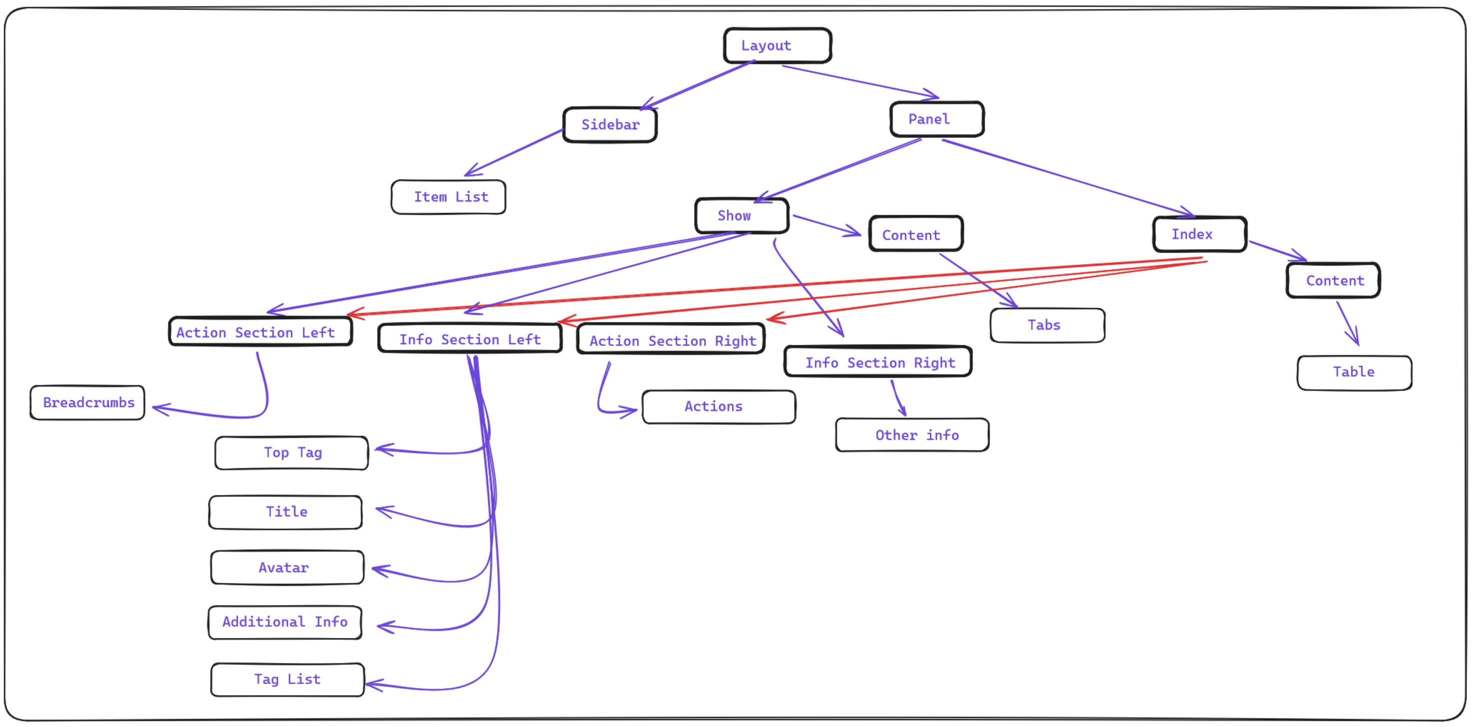


Figure 6.42: LoopOS UI progressive development planning: Micro layout refinement of Show Layout.

Thus, while preserving the distinct Header and Content sections, the Show and Index pages could be combined into a single Panel Component. Each page would employ its specialized header and content components within the overarching Panel structure.

This approach maintains code reusability and simplifies component management. By employing a single Panel component, developers can effectively encapsulate various content types, enhancing code modularity and reducing redundancy. Simultaneously, this strategy eliminates the need for separate components for Show and Index, streamlining the overall structure and promoting code clarity. Figure 6.43 illustrates a diagram of the progress at the end of this step, where Index and Show are merged.

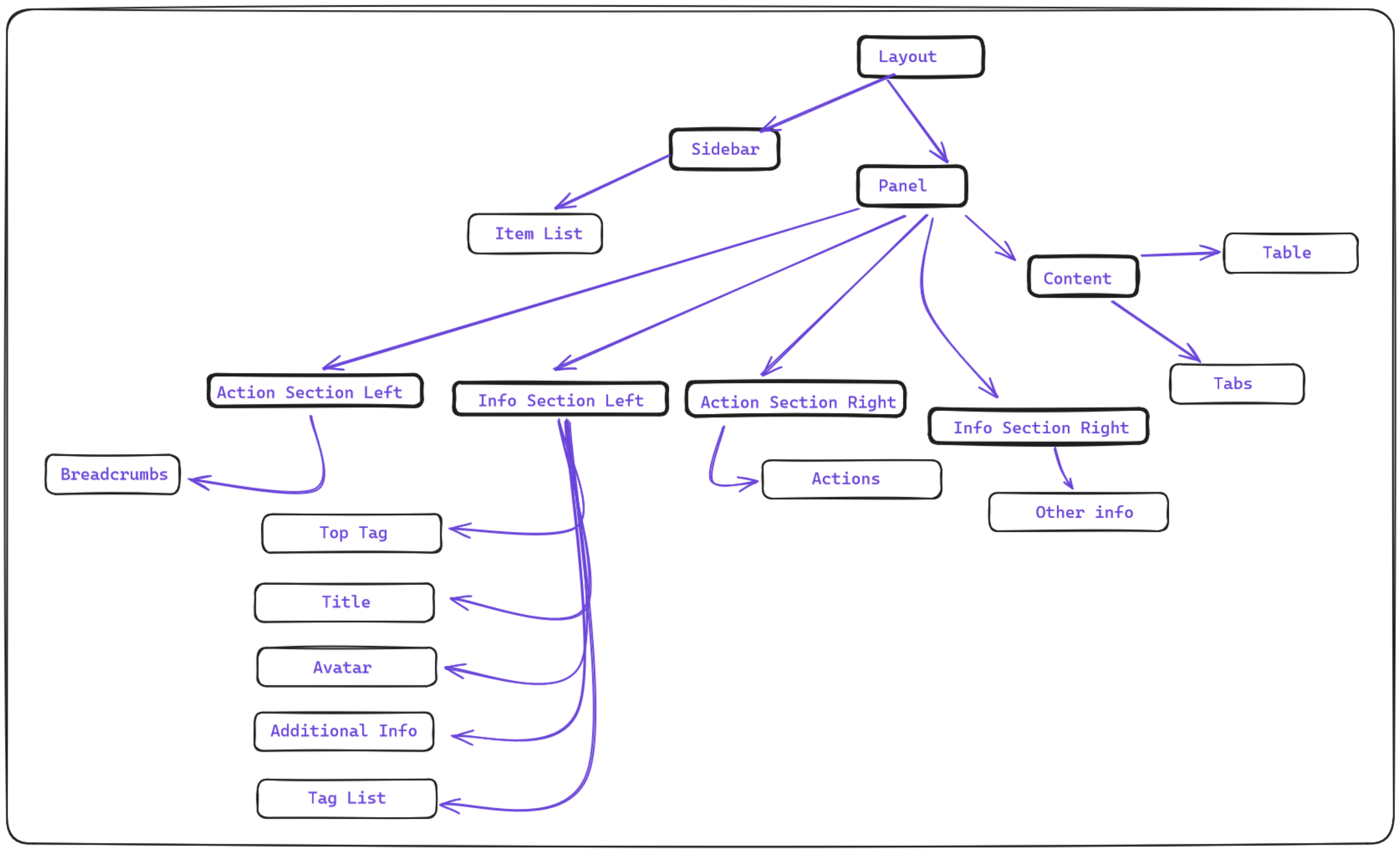


Figure 6.43: LoopOS UI progressive development planning - final architecture.

To conclude this Planning phase, Figure 6.44 and 6.45 will exemplify, while using the previous diagram, which components the User Show Page and User Index Page, respectively, use.

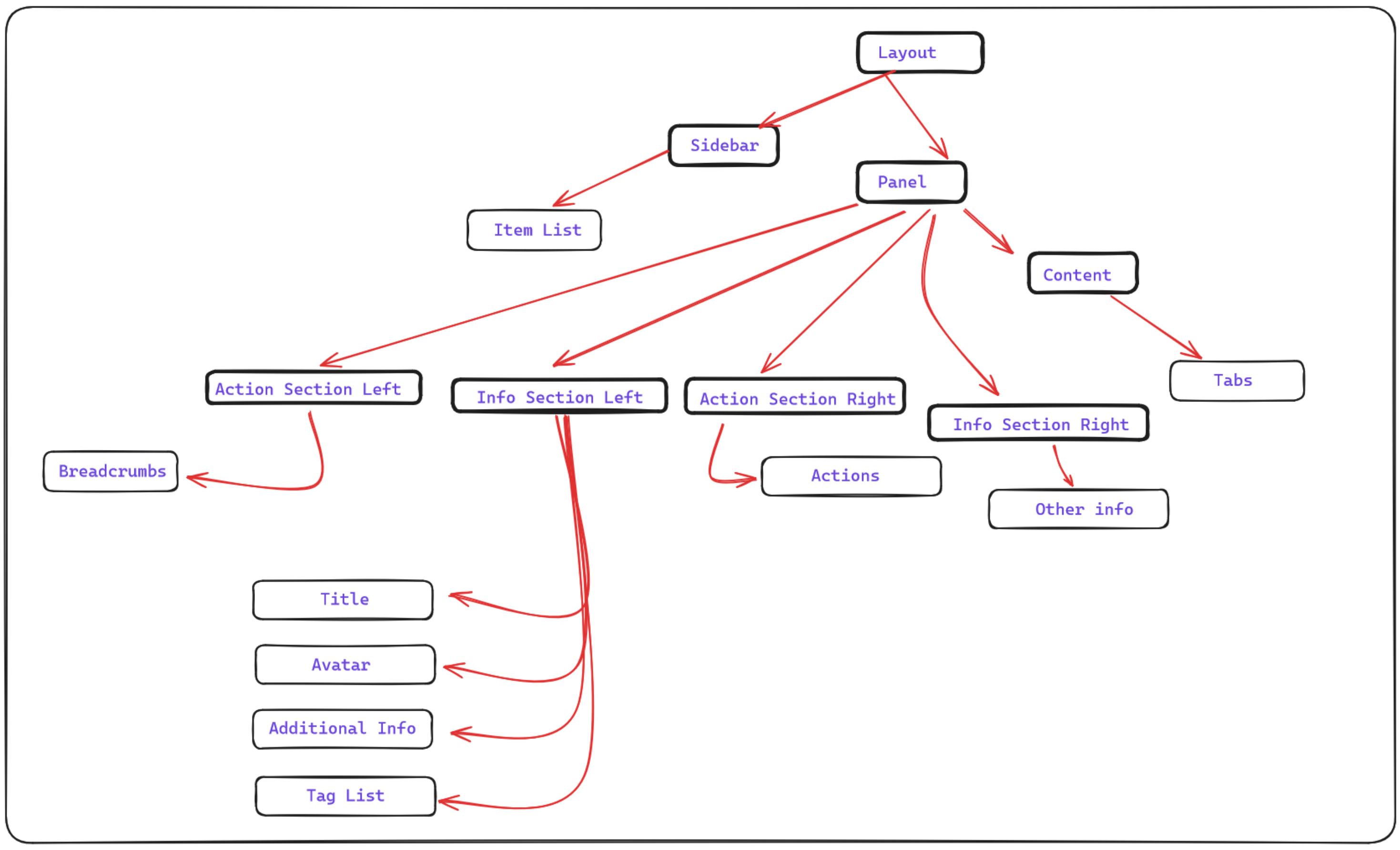


Figure 6.44: Network Manager implementation of User Show Page using LoopOS UI.

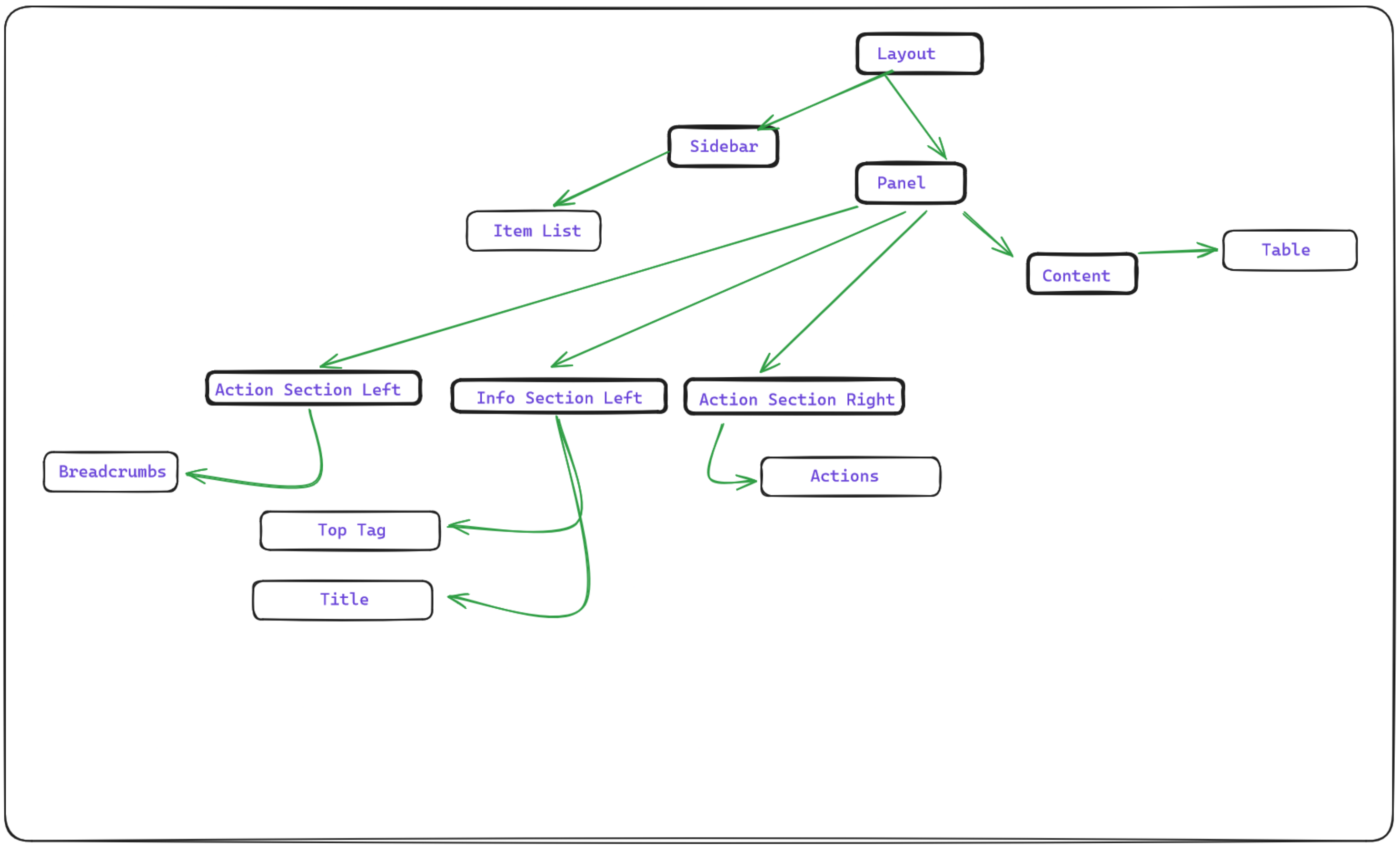


Figure 6.45: Network Manager implementation of Index Show Page using LoopOS UI.

With the overall planning concluded, moving on to sprint planning was possible. Adhering to the development lifecycle outlined in Chapter 5, Section 5.2, I carefully created a realistic roadmap for each sprint, outlining the features to be developed and implemented in both the Network Manager and the LoopOS UI. Furthermore, I conducted a detailed low-level planning exercise at the beginning of each sprint, detailing the precise tasks to be addressed, their execution plan, and comprehensive test plans. This meticulous approach ensured adherence to the company's established sprint lifecycle and facilitated efficient feature development.

◆ Setup

As mentioned before, the planning and development of LoopOS UI was conducted immediately after the development of LoopOS Onboarding. Consequently, with only four weeks left, the proposed plan followed.

With no prior experience in creating and setting up Rails Engines and their necessary dependencies, including configuring asset pipelines, two weeks were allocated solely to this setup. Due to their complex nature, a detailed explanation of the purpose and configuration of asset pipelines will be provided later. The subsequent sprints were dedicated to developing and implementing the Sidebar in the third week, with the progressive implementation of Partner and User Show pages planned for the fourth week.

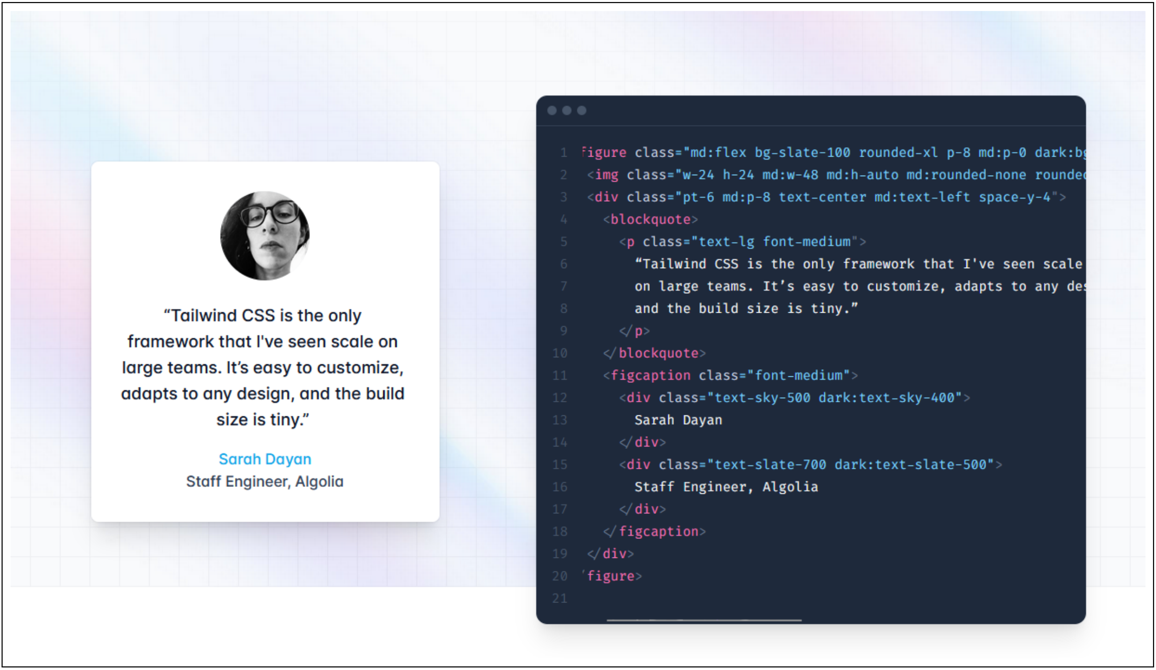
The setup started with creating a self-contained Rails Engine in a distinct Git repository, enabling applications to simply clone it upon instantiation, accompanied by a dummy application for testing. This approach enabled the implementation of essential dependencies and asset pipeline configuration in a separate environment, without directly impacting the Network Manager. Given the existence of a single Network Manager instance, directly modifying its code could have been problematic.

Hence, with the Rails Engine in place, I started by analyzing the necessary dependencies in the Network Manager and other LoopOS applications that needed to exist on the LoopOS UI side.

Starting with **Turbo** [68]. It enhances the user experience by enabling seamless, fast, and efficient page updates without full page reloads. Turbo optimizes the front-end interaction by updating only the necessary parts of a page, reducing the need for full page reloads. This results in a smoother and more responsive user interface.

For example, a user has a list of tabs on a page similar to Figure 6.38, and they want to be able to click on a tab to show the content for that tab without having to reload the page. By using Turbo, when a user clicks on a tab, it is possible to replace just the content of the corresponding view. This will happen without reloading the page, improving the user experience.

Tailwind CSS [66], is a CSS framework that is also employed by LoopOS applications. It provides a set of pre-defined utility classes that can be used to style HTML elements. Tailwind is used for styling and layout purposes in all LoopOS applications. Figure 6.46 presents an example from the Tailwind website highlighting its ease of use and customization.



“Tailwind CSS is the only framework that I've seen scale on large teams. It’s easy to customize, adapts to any design, and the build size is tiny.”

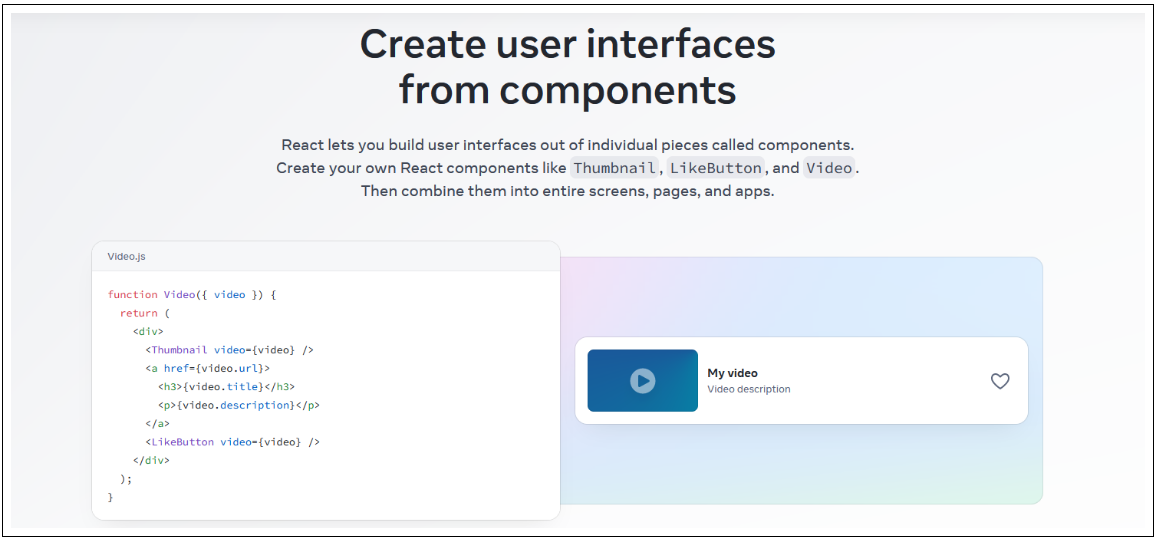
Sarah Dayan
Staff Engineer, Algolia

```
igure class="md:flex bg-slate-100 rounded-xl p-8 md:p-0 dark:b
 <img class="w-24 h-24 md:w-48 md:h-auto md:rounded-none rounde
 <div class="pt-6 md:p-8 text-center md:text-left space-y-4">
   <blockquote>
     <p class="text-lg font-medium">
       “Tailwind CSS is the only framework that I've seen scale
       on large teams. It’s easy to customize, adapts to any de
       and the build size is tiny.”
     </p>
   </blockquote>
   <figcaption class="font-medium">
     <div class="text-sky-500 dark:text-sky-400">
       Sarah Dayan
     </div>
     <div class="text-slate-700 dark:text-slate-500">
       Staff Engineer, Algolia
     </div>
   </figcaption>
 </div>
igure>
```

Figure 6.46: Implementation example from Tailwind Website [66].

React [57] is a JavaScript library for building user interfaces that enables the creation of reusable UI components while efficiently managing the application’s state. It is mostly employed in LoopOS due to the expertise in React from the front-end team and being the core of LoopOS SDK, the reusable library components explained before.

Figure 6.47 shows a React component called Video, which displays a video thumbnail, title, and description. The component also includes a LikeButton component, which allows users to like the video. This is a simple example of how React components can be used to create complex user interfaces.



Create user interfaces from components

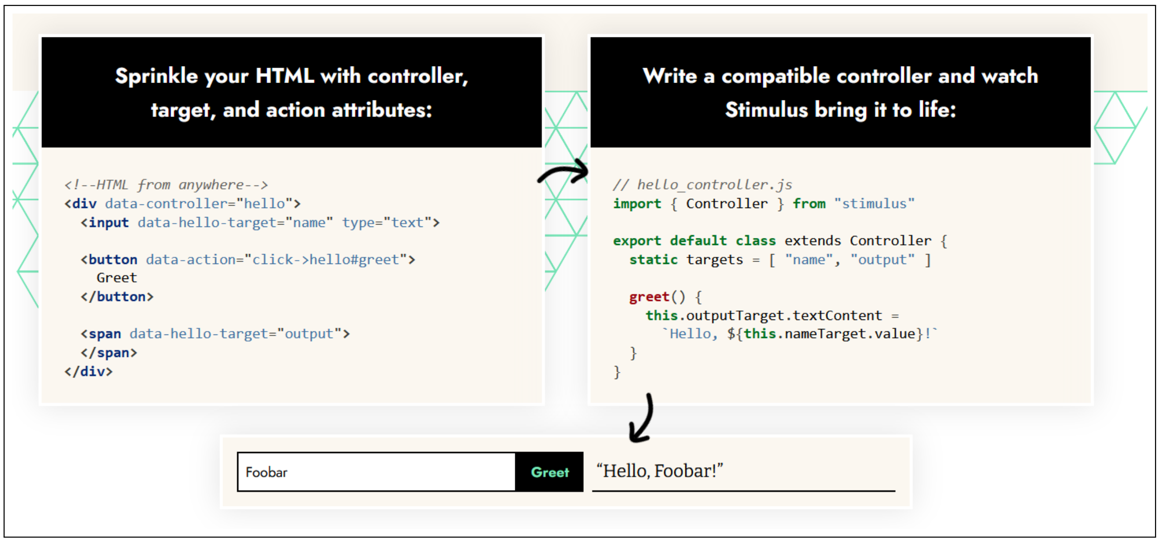
React lets you build user interfaces out of individual pieces called components.
Create your own React components like `Thumbnail`, `LikeButton`, and `Video`.
Then combine them into entire screens, pages, and apps.

Video.js

```
function Video({ video }) {
  return (
    <div>
      <Thumbnail video={video} />
      <a href={video.url}>
        <h3>{video.title}</h3>
        <p>{video.description}</p>
      </a>
      <LikeButton video={video} />
    </div>
  );
}
```

Figure 6.47: Example from React Website[57].

In addition, **Stimulus**[65] is built to work with Turbo and Tailwind CSS, providing several features that make building interactive and responsive UIs easier. For example, Stimulus allows to define custom events that elements in the UI can trigger, and it provides a templating engine that makes it easy to create reusable components. It is mostly used by the back-end team in LoopOS to provide actions to components created by the front-end team. Figure 6.48 showcases an example where an HTML document uses a Stimulus controller to dynamically change its content.



Sprinkle your HTML with controller, target, and action attributes:

```html
<!--HTML from anywhere-->
<div data-controller="hello">
  <input data-hello-target="name" type="text">

  <button data-action="click->hello#greet">
    Greet
  </button>

  <span data-hello-target="output">
  </span>
</div>
```

Write a compatible controller and watch Stimulus bring it to life:

```javascript
// hello_controller.js
import { Controller } from "stimulus"

export default class extends Controller {
  static targets = [ "name", "output" ]

  greet() {
    this.outputTarget.textContent =
      `Hello, ${this.nameTarget.value}!`
  }
}
```

Figure 6.48: Example from Stimulus Website **Stimulus**.

To exemplify, imagine you want to create a button that changes its color when clicked. With React, you can create a component for the button and import the corresponding Tailwind CSS file to style it. Within the React component of the button, it will call a Stimulus controller to handle its on-click event and manage the color-changing process.

Hence, React will trigger its on-click event, triggering the Stimulus controller to update the box's color when the button is clicked. Since the button is within a Turbo frame, this will happen without reloading the page.

However, to ensure consistent UI components across all LoopOS projects, most applications use the LoopOS SDK. The SDK acts as a centralized repository for UI components, enabling developers to easily import and customize the desired elements. Instead of creating custom button components for each application, developers can simply import the pre-built button component from the SDK and specify the necessary attributes, such as the desired icon, size, and Stimulus action.

This approach simplifies the development process and promotes consistency across the platform. The CSS styles and parent React Components remain within the LoopOS SDK, while the application-specific React Component that imports the button component and the Stimulus controller resides in the Network Manager.

This allows the Network Manager to control the button's behavior without concerning itself with the underlying CSS or controller logic.

Thus, all these dependencies and LoopOS SDK must exist in LoopOS UI to aid in populating the aforementioned layouts with interactive components.

Moving on to the layouts themselves, these will be created using **View Components** [69], a framework-agnostic library that promotes a modular and reusable approach to building user interfaces. It provides a structured way to organize and encapsulate front-end code, enabling the LoopOS UI to become the single source of truth for page layout in LoopOS Applications.

While other dependencies may be essential, View Components form the foundation of this new Engine. Therefore, it is crucial to grasp their core concepts. A ViewComponent is essentially a Ruby object that renders using a template. Listings 6.11 and 6.12 showcase the creation of a ViewComponent that accepts a name and renders it within the template.

```ruby
class MessageComponent < ViewComponent::Base
  def initialize(name:)
    @name = name
  end
end
```

Listing 6.11: View Component Class - Basic Example.

```erb
<h1>Hello, <%= @name %>!</h1>
```

Listing 6.12: View Component Template - Basic Example.

The View Component is then embedded in a View, as shown in Listing 6.13.

```erb
<%= render(MessageComponent.new(name: "World")) %>
```

Listing 6.13: View Component Render - Basic Example.

Depending on the request, the Network Manager can render the View Component with either the respective user or the partner information. This allows for the creation of both the User Show Page and the Partner Show Page using the same View Component.

In addition, View Components can accept content through slots, making it possible to render multiple blocks of content, including other components. Slots can be defined with *renders_one* and *renders_many*, where *renders_one* specifies a slot that will be rendered at most once per component and *renders_many* specifies a slot that can be rendered multiple times per component. Listing 6.14 illustrates how a Blog class can declare another View Component.

```ruby
class BlogComponent < ViewComponent::Base
  renders_one :header

  class HeaderComponent < ViewComponent::Base

    def initialize(name:)
      @name = name
    end
  end
```

```
end
```

Listing 6.14: View Component Class - Slots Example.

Then, Listing 6.15 presents the template of the Blog component, which renders its header, while Listing 6.16 shows the template of the Header Component.

```
<%= header %>
```

Listing 6.15: View Component Blog Template - Slots Example.

```
<h1>Hello, <%= @name %>!</h1>
```

Listing 6.16: View Component Header Template - Slots Example.

Finally, the call to the Blog component with a Header component is realized in a view, as can be seen in Listing 6.17.

```
<%= render BlogComponent.new do |component| %>
  <% component.header(name: "World")%>
<% end %>
```

Listing 6.17: View Component Render - Basic Example.

In conclusion, LoopOS UI can use View Components to create complex and hierarchical UIs. These components can be nested within each other to build layouts, and they can accept content through slots, making it possible to reuse and customize components.

The core objective of implementing this new engine in LoopOS applications is to centralize all front-end functionality within the Engine. To achieve this, I integrated all the mentioned dependencies into the LoopOS UI and, in the dummy app, only the ones that did not extend beyond front-end capabilities. Specifically, I implemented React, Tailwind CSS, Stimulus, Turbo, View Components, and the LoopOS SDK within the Engine. I only implemented Turbo and Stimulus in the dummy app to simulate a LoopOS application.

After this, it was possible to move on to the configuration of the asset pipeline.

In the dynamic world of web development, asset pipelines play a crucial role in tackling the intricate challenges that arise as websites evolve. While the basic HTML, CSS, and JavaScript toolkit form a solid foundation, there is often the integration or replacement of these core elements, as seen before with React and Tailwind.

Website development encompasses a wide range of challenges. From managing dependencies, which becomes critical as third-party modules are incorporated, often leading to compatibility issues, to optimizing resources by combining multiple files into one, often mentioned as efficient bundling. Additionally, compatibility with older browsers requires creating distinct JavaScript versions.

To solve these issues comes the creation of module bundlers, more specifically Webpack [71], the one I employed due to aleady being used in other LoopOS applications. Its primary function involves resolving imports and effectively managing dependencies to generate a unified, optimized bundle.

Through code bundling, Webpack seamlessly combines multiple JavaScript files, along with their dependencies, into a single file for efficient loading in the browser. Loaders within webpack facilitate the processing of files beyond JavaScript, such as stylesheets (CSS/Sass), allowing diverse file types to seamlessly integrate into the application. Figure 6.49 illustrates the different files bundled into one.

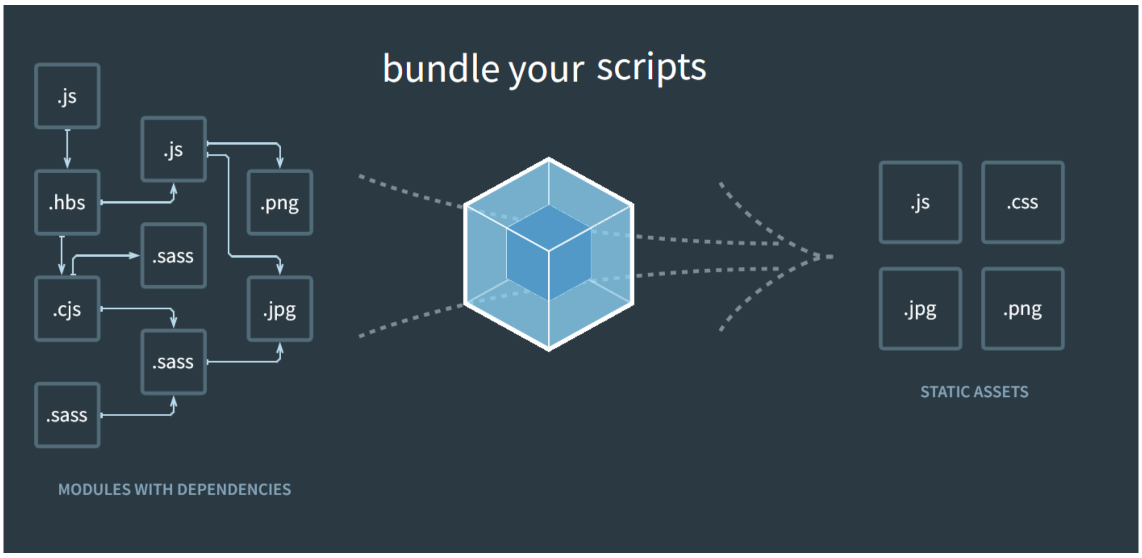
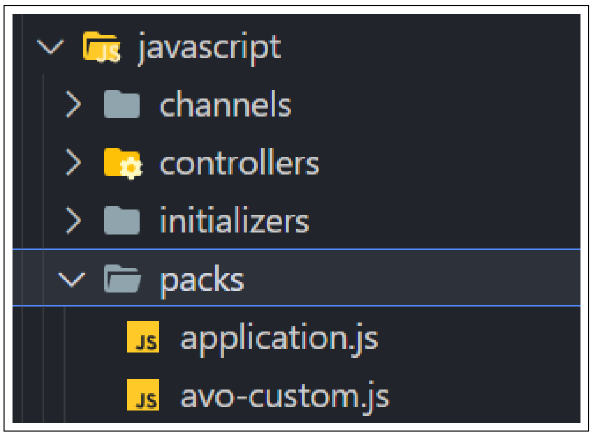


Figure 6.49: Bundling diagram from Webpack website [71].

Webpack offers a suite of optimizations, including minification [42] (removing unnecessary or redundant data: removing whitespace, shortening variables, etc), code splitting [56] (creating multiple bundles that can be dynamically loaded at runtime, so the server only load the assets it requires), and tree-shaking [72] (removing unused code from bundles), to reduce file size and enhance performance.

Webpack is the JavaScript bundler used in all LoopOS applications. Within the javascript folder, there is a packs directory that holds multiple entrypoint in the form of JavaScript files. These files import all the code from the controllers to the components required for their specific functionality.

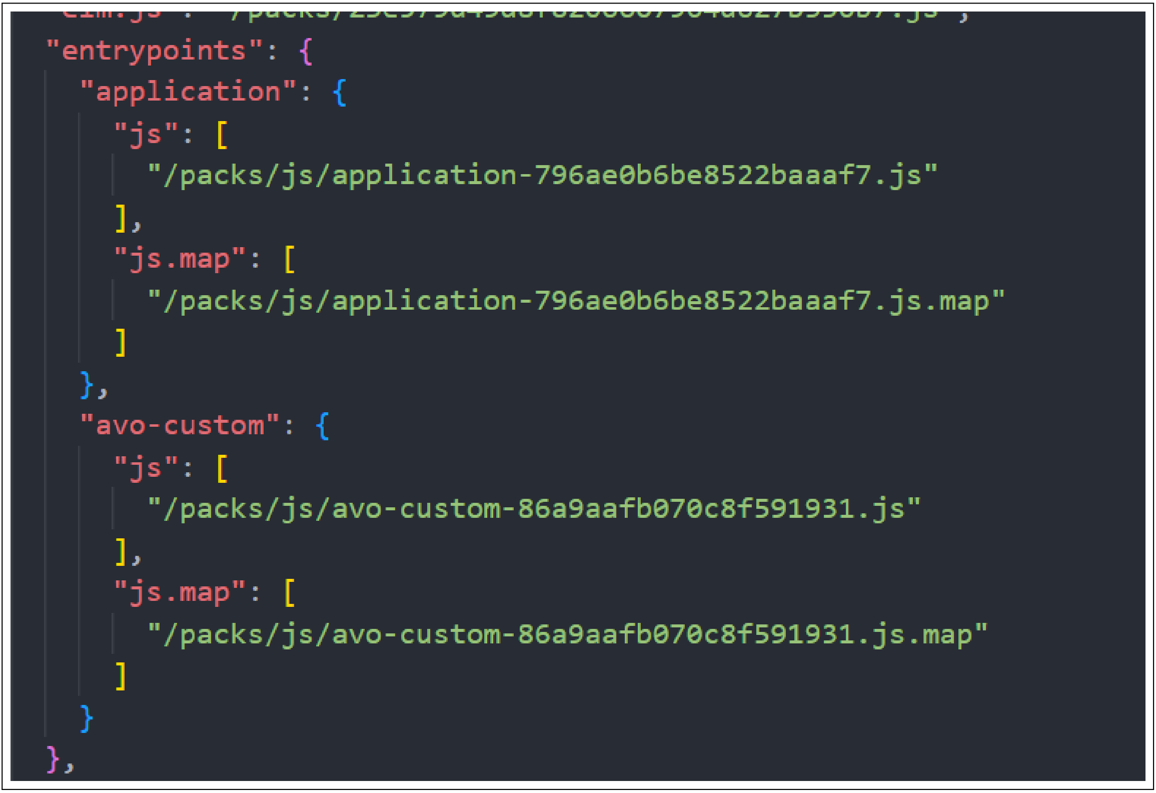
As shown in Figure 6.50, the Network Manager application has several packs, each with its own entrypoint file.



```
javascript
  channels
  controllers
  initializers
  packs
    application.js
    avo-custom.js
```

Figure 6.50: Network Manager packs.

When Webpack initializes or detects changes in the JavaScript files, it bundles all the code and creates a manifest.json file in the application's public folder. This file contains entrypoints to the bundled JavaScript code. Figure 6.51 depicts a portion of the Network Manager manifest file, where the previously mentioned packs have been split into separate bundles, ensuring code splitting. This allows for loading specific bundles only when needed, as demonstrated in Listing 6.18.



```
"entrypoints": {
  "application": {
    "js": [
      "/packs/js/application-796ae0b6be8522baaaf7.js"
    ],
    "js.map": [
      "/packs/js/application-796ae0b6be8522baaaf7.js.map"
    ]
  },
  "avo-custom": {
    "js": [
      "/packs/js/avo-custom-86a9aafb070c8f591931.js"
    ],
    "js.map": [
      "/packs/js/avo-custom-86a9aafb070c8f591931.js.map"
    ]
  }
},
```

Figure 6.51: Network Manager manifest file.

```
<%= javascript_pack_tag "application" %>
```

Listing 6.18: Network Manager - Application pack tag.

This approach provides efficient code loading and improves the overall performance of LoopOS applications.

One of the more challenging aspects of this setup was the need to manually install and configure Webpack within the Rails Engine. Additionally, due to the presence of Webpack in each LoopOS application, there was a need for multiple Webpack instances, which Webpack is not prepared for, it introduced compatibility issues. This required careful coordination and configuration to ensure that all Webpack instances were operating correctly and that code could be shared seamlessly between the Rails Engine and the applications.

To address this challenge, a two-stage setup was implemented. First, the dummy app's Webpack was run, generating the required files in the public folder and creating a manifest file with those entrypoints. Then, the engine's Webpack was executed, and its files were outputted to the applications' public folder as well, generating another manifest file merging the entrypoints. This ensured that the dummy app had access to its own Webpack content and the engine's Webpack content. This arrangement enabled the engine to bundle its React and Tailwind components, add them to the dummy app, and allow the dummy app to use them as regular JavaScript files without needing React or Tailwind installed. This approach effectively created a shared asset pipeline between the engine and the dummy app. Listing 6.19 illustrates the dummy app calling in its HTML file the application pack-tag which contains all the controllers and components of the application and the loopos-ui pack-tag which contains all the controllers and components of the Engine.

```
<%= javascript_pack_tag "application" %>
<%= javascript_pack_tag "loopos-ui" %>
```

Listing 6.19: Dummy App - Application pack tag.

Thus, after two weeks of setup and successfully testing the Engine in this isolated environment, only two weeks were left to integrate the LoopOS UI in the Network Manager and develop the layouts.

Despite the initial success of the two-stage setup, a critical issue emerged upon implementing the LoopOS UI in the Network Manager. It became apparent that the asset pipeline was not functioning as expected. This issue arose from a fundamental flaw in the testing procedure. During the initial testing of the asset pipeline in the dummy app, no JavaScript files were created within the dummy app itself. As a result, the assumption that the asset pipeline had successfully merged both manifest files was incorrect. The Engines manifest file inadvertently replaced the dummy app's manifest file, resulting in the dummy app's JavaScript bundle being overridden.

After a week of troubleshooting and exploring alternative solutions, I devised a workaround. Instead of maintaining two separate Webpacks, a single Webpack instance was employed for the entire application. Within the main application pack, a

JavaScript file from the Engine was imported, loading the LoopOS UI components and controllers. While this workaround provided a functional solution, it lacked the efficiency of code splitting. To access the LoopOS UI's controllers and components, the application pack-tag (Listing 6.18) needed to be invoked, loading all the code from both the application and the LoopOS UI, potentially affecting performance.

Whoever, this allowed, in the one week I had left, to proceed with the layout implementation in the Network Manager.

◆ Layout Implementation

The development process began with creating the main layout View Component. This component, named LooposUi::LayoutComponent, serves as a reusable building block for structuring web pages. It encapsulates the three key sections of a web page: the topbar, the sidebar, and the app content region. Passing these sections as arguments to the component seamlessly combines them into a cohesive layout structure.

In the main layout of the Network Manager application, the LooposUi::LayoutComponent is rendered to generate the overall page structure. The component receives the topbar and sidebar sections rendered using partials, which are reusable chunks of Ruby code. It also receives the app content section which is dynamically generated using the yield keyword, allowing any specific page content to be included. Finally, it receives *user_signed_in* and *current_user* parameters provide information about the current user's authentication status and user information. Listings 6.20 and 6.21 showcase the code to implement the LooposUi::LayoutComponent component in the Network Manager and its creation in the LoopOS UI Engine respectively. Should be noted, the LooposUi::LayoutComponent has template file to handle this rendering

```
<%= render LooposUi::LayoutComponent.new(
  topbar: render(partial: "admin/navigation/topbar"),
  sidebar: render(partial: "admin/navigation/sidebar"),
  app_content: yield,
  user_signed_in: user_signed_in?,
  current_user: current_user,
  ) %>
```

Listing 6.20: Network Manager - LooposUi::LayoutComponent is used to rendered the layout structure.

```
module LooposUi
  class LayoutComponent < ViewComponent::Base
    include Turbo::StreamsHelper

    def initialize(topbar:, sidebar:, app_content:,
       user_signed_in:, current_user:)
      @topbar = topbar
      @sidebar = sidebar
      @app_content = app_content
```

```
        @user_signed_in = user_signed_in
        @current_user = current_user
      end
    end
  end
```

Listing 6.21: LoopOS UI - LooposUi::LayoutComponent class.

◆ Show Layout Implementation

The development in this phase started with the creation of panel component to render the header and the content. As explained in Setup phase, View Components allows for slots making it possible to render multiple blocks of content, including other components. Thus, the Panel Component also creates and renders the Header and the Content. Listing 6.22 showcases the creation of this class.

```
module LooposUi
  class PanelLayoutComponent < ViewComponent::Base
    include Turbo::StreamsHelper

    renders_one :header_layout, "HeaderLayoutComponent"
    renders_one :content_layout, "ContentLayoutComponent"

    def initialize()
    end

    class HeaderLayoutComponent < ViewComponent::Base
      def initialize(layout:)
        @layout = layout
      end
    end

    class ContentLayoutComponent < ViewComponent::Base
      def initialize(layout:)
        @layout = layout
      end
    end
  end
end
```

Listing 6.22: LoopOS UI - LooposUi::PanelLayoutComponent class.

Then, each of these classes has a template. While the Panel component template renders the header and the content (as seen in Listing 6.23) the Header and the Content components template render their layout (as seen in Listing 6.24).

```
<div id="app_layout_frame" class="form-layout">
  <%= header_layout %>
  <%= content_layout %>
  </div>
```

Listing 6.23: LoopOS UI - LooposUi::PanelLayoutComponent Template.

```
  <%= @layout %>
```

Listing 6.24: LoopOS UI - Header and Content Template.

Following, the Network Manager uses this View Component in the User and Partner Show page to render them. Listing 6.25 showcases this implementation in the User Show Page.

```

  <%= render LooposUi::PanelLayoutComponent.new do |
     component| %>
    <% component.header_layout(layout: render(
                                      partial: "admin/users/
                                         header",
                                      locals: { user: @user},
                                    )) %>
    <% component.content_layout(layout: render(
                                       partial: "admin/users/
                                          content",
                                       locals: { user: @user },
                                     )) %>
  <% end %>
```

Listing 6.25: LoopOS UI - Header and Content Template.

Although simple at first, this is the objective of employing a progressive implementation. In this first phase, the View Components will receive the entire content of each partial and render it as seen before. However, when at the end of the development the Network Manager will only call for what it desires and the LoopOS UI will built it. Listing 6.26 illustrates a snippet on how the User Show Page in the Network Manager might request LoopOS UI to built its page, where the left header action section has breadcrumbs that show the current location in the application hierarchy. The right header action section has two actions: Edit and Delete. The left header info section displays the user's name, email, partner, and user groups. The right header info section displays the user's invitation and updated timestamps. The content section has three tabs: Info, Configurations, and User Scope.

```
<%= render LooposUi::PanelLayoutComponent.new do |component
   | %>
  <% component.left_header_action_section(
      breadcrumbs: [
      { "User Management" => "" },
      { "Users" => "" },
      { "#{@user.full_name}" => admin_user_path },
    ]
  )%>
  <% component.right_header_action_section(
      actions: [
        { label: "Edit", url: edit_admin_user_path(
           current_user) },
        { label: "Delete", url: admin_user_path(
           current_user), method: :delete, data: { confirm:
            "Are you sure?" } },
      ]
  )%>
  <% component.left_header_info_section(
      title: {name: "User", is_editable: true},
      email: current_user.email,
      partner: current_user.partner,
      tag_list: [
        { user_groups: current_user.user_groups },
      ]
  )%>

  <% component.right_header_info_section(
      additional_info: [
        { label: "Invited At", value: current_user.
           invited_at },
        { label: "Updated At", value: current_user.
           updated_at },
      ]
  )%>

  <% component.content_section(
      tabs: [
        {name:  "Info", icon: "fa-info", partial: user_info
           },
        {name:  "Configurations", icon: "fa-gears", partial
           : user_configurations},
        {name:  "User Scope", icon: "fa-shield", partial:
           user_scopes},
      ]
  )%>
<% end %>
```

Listing 6.26: LoopOS UI - User Show Page possible implementation.

◆ Sidebar Layout Implementation & Refine

Due to the time constraint, the final component developed was the Sidebar. Although appearing simple, it was a complex component since it was the first View Component builder.

In the absence of final sidebar mockups from the Network Manager, I analyzed the mockups from Core and Hubs to establish the sidebar's structure. As depicted in Figure 6.52, the sidebar can be divided into two sections: the Top Sidebar and the Bottom Sidebar. The Top Sidebar houses all primary navigation elements, while the Bottom Sidebar contains helper items and a home page icon. Despite the incomplete development of the Bottom Sidebar elements due to the absence of finalized mockups, their implementation will be straightforward in the future due to the logical approach employed during the creation of the Sidebar Layout View Component.

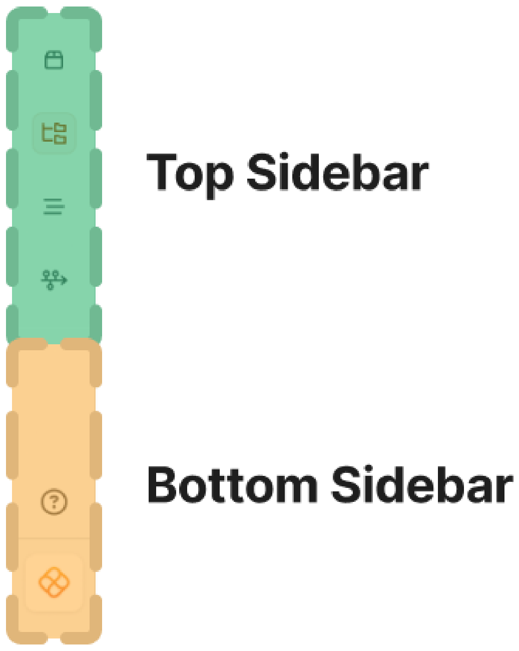


Figure 6.52: Sidebar Main Anatomy.

Secondly, the primary navigation items identified before can be categorized into two groups: Drawer items and Single items.

Drawer items, depicted in Figure 6.53, do not immediately navigate to a specific page but expand to reveal multiple subitems upon hovering. These drawers consist of an icon, a title, a subtitle, and corresponding subitems, each associated with a distinct redirect.

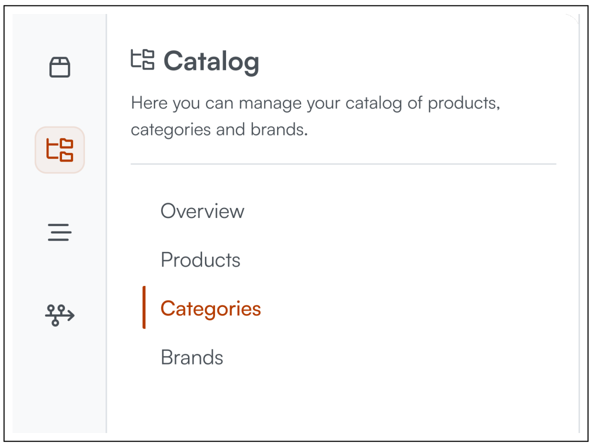

Catalog
Here you can manage your catalog of products, categories and brands.
Overview
Products
Categories
Brands

Figure 6.53: Core: Sidebar Mockups - Item with drawer open.

Single items, illustrated in Figure 6.54, redirect to specific pages upon clicking and display a tooltip when hovered. These items contain an icon, a name, and a redirect.

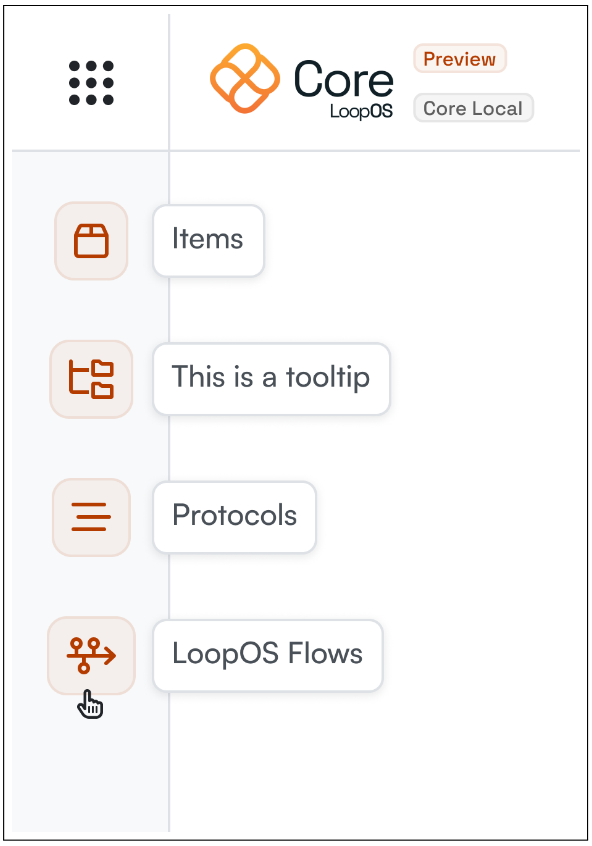


Figure 6.54: Core: Sidebar Mockups - Item without drawer.

Moreover, Drawer or Single items icons should be dynamically highlighted to indicate the current page the user is on. As shown in Figure 6.55, the "Items" icon should be highlighted since the user is currently viewing the "Items" page. Thus, when building the items, they should receive that page's controller (or controllers in the case of Drawers).

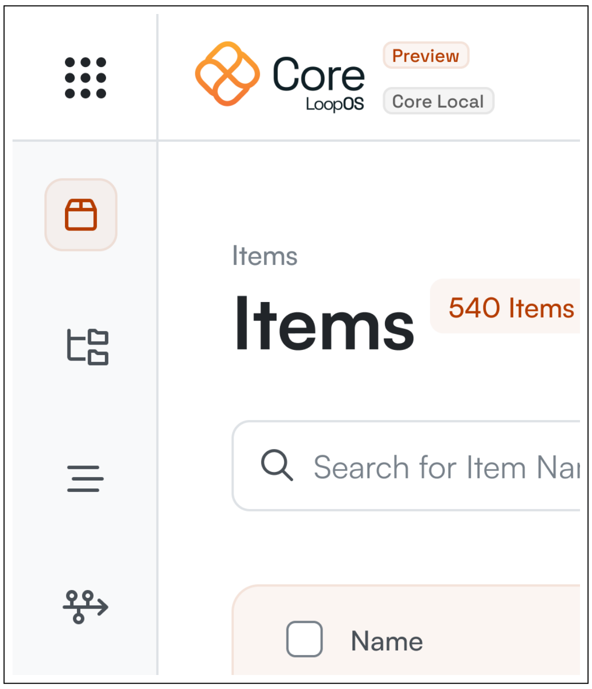


Figure 6.55: Core: Sidebar Mockups - Item highlight.

To ensure compliance with the non-functional security requirement, each application should implement a mechanism to control item visibility. This can be achieved by incorporating a lambda function (an anonymous function that takes any number of arguments but can only have one expression), *can_view*, within each item and sub-items. Each application can then use this variable to determine whether a particular item should be rendered based on user permissions. For instance, the Hubs employs Pundit (a tool referenced in Section 4.4) for role-based access control, while Core and Network Manager do not currently implement such mechanisms, thus granting them the liberty to implement this requirement further on.

In addition, all item paths are provided to the corresponding View Component through helper methods defined in the routes file. For instance, the User Show page in the routes file is defined as *admin_user_path*. These helper methods are sent to the View Component as lambda functions. These lambda functions, such as the *can_view* function, must be triggered by a call method to be executed.

All LoopOS UI components also leverage the View Components framework to maintain modularity and self-sufficiency. This entails each component being separate and capable of rendering itself, ensuring each component possesses its own rendering method.

Furthermore, given the substantial overlap in components between Drawer and Single items, creating a parent class establishes an aggregation relationship, promoting code reusability and maintainability. This parent class encapsulates common functionalities, allowing Drawer, Single, and any future items (such as the helper item for the Bottom Sidebar) to inherit and extend these features and rendering methods. Figure 6.56 presents a Class Diagram with these items.

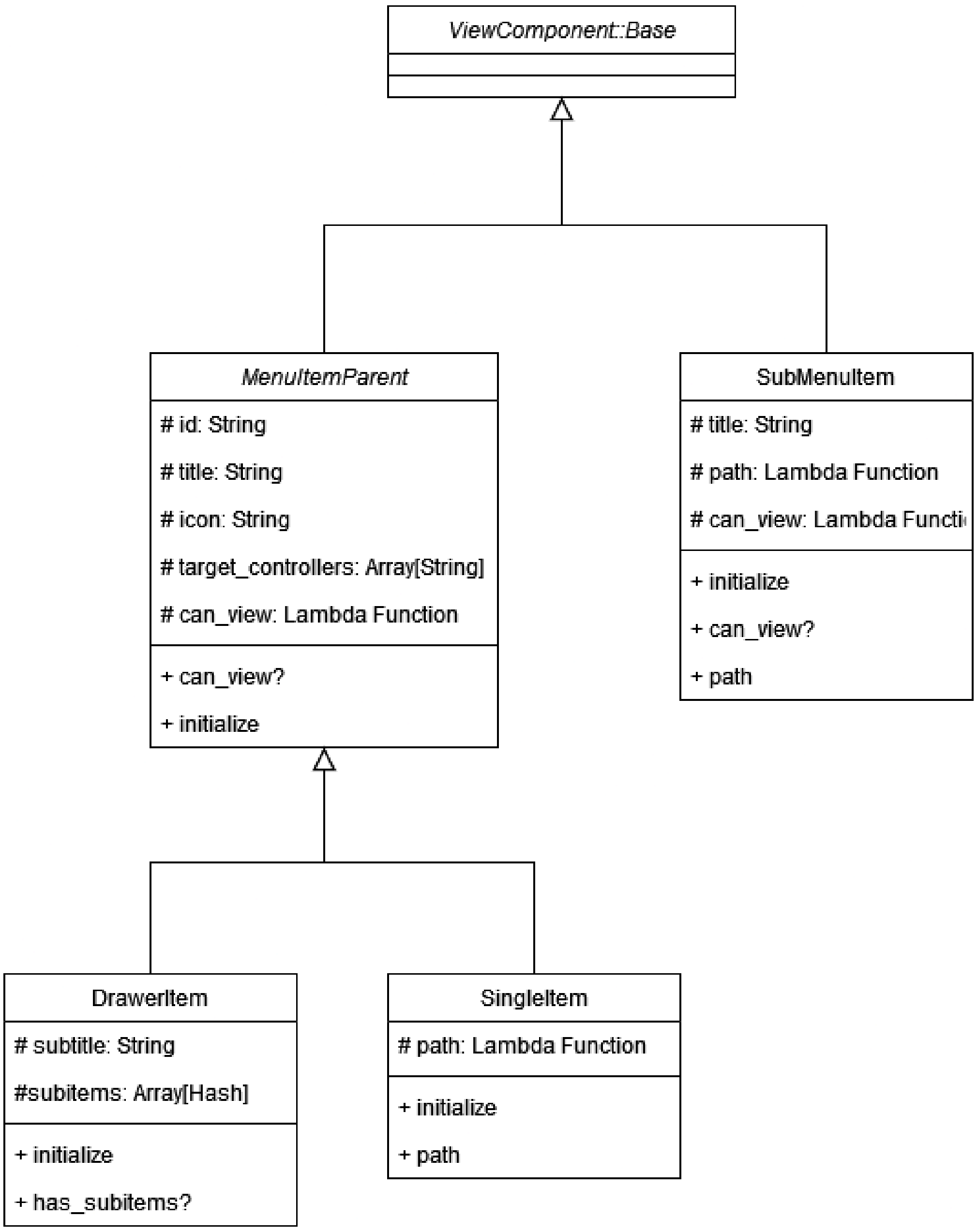


Figure 6.56: Sidebar Items Class Diagram.

The Drawer item and the Single item each have their distinct rendering mechanisms. This is achieved by having separate files with the same name of the class file but with an *.html.erb* extension.

Starting with the functionality of the Drawer Item, its template initially checks if the current controller operating on the page is included in the *target_controllers* array. If it is, the button is highlighted.

Next, the template links the button's ID to the corresponding modal element. It also specifies that hovering over the button should trigger the opening of the drawer. This is where the Stimulus JavaScript framework comes into play. The command in Listing 6.27 activates the Engine Bar controller JavaScript file, which holds the event handlers for displaying and hiding the drawer. In this case, the drawer with the specified ID will be opened.

```
data-target-modal="<%= @id %>"
data-action="click->enginebar#show mouseover->enginebar#
    show">
```

Listing 6.27: LoopOS UI - Drawer item Template

The sidebar icon is then displayed even when the drawer is closed, ensuring its presence. The icon, title, and subtitle are presented inside the drawer, which is initially hidden. Next, the drawer checks if there are subitems. If so, it iterates through the array of Subitems View Components and renders their title along with a link to their respective path if the *can_view* property returns true.

Thus achieving the mechanics seen in Figure 6.53.

The Single item template employs an approach similar to its Drawer item counterpart to identify the current active controller. It examines whether the current controller operating on the page is included in the *target_controllers* array. If it is, the button associated with that item is highlighted.

Next, the template implements a feature that hides any open drawer when hovering over the button. This behavior is triggered by the code snippet in Listing 6.28, which activates the hide method in the Engine Bar controller JavaScript file. Upon hovering over the button, any drawer is hidden.

```
data-action="mouseover->enginebar#hide">
```

Listing 6.28: LoopOS UI - Single item Template

In contrast to the Drawer item template, the Single item template directly links the icon to the appropriate path and adds a tooltip with the title. Thus achieving the mechanics seen in Figure 6.54.

Moving on to the *SidebarLayoutComponent* itelf, it has two View Component slots, one for the Top Sidebar and another for the Bottom Sidebar, although this one was not fully implemented due to the reasons described before.

The *TopSidebarComponent* receives an array of hashes, each representing a menu item. If the array is not empty, it calls the *map_menu_items* method to iterate through each item and create corresponding View Components based on the presence of a *:subitems* key.

The *map_menu_items* method iterates through the array of hashes and calls the *create_menu_item* method for each hash. The *create_menu_item* method checks for the presence of the *:subitems* key in the hash. If the key is present, it creates a Drawer item. Otherwise, it creates a Single item.

This approach allows the *TopSidebarComponent* to create a *menu* variable populated with the desired View Components based on the structure of the received data. Listing 6.29 illustrates the implementation of these methods.

```
class TopSidebarComponent < ViewComponent::Base
  def initialize(menu: [])
    @menu = \textit{map\_menu\_items}(menu) unless menu.
       empty?
  end

  private

  def \textit{map\_menu\_items}(menu)
    menu.map { |item_hash| \textit{create\_menu\_item}(
       item_hash) }
  end

  def \textit{create\_menu\_item}(item_hash)
    if item_hash[\textit{:subitems}].present?
      LooposUi::MenuItemComponent::DrawerItem.new(
         item_hash: item_hash)
    else
      LooposUi::MenuItemComponent::SingleItem.new(
         item_hash: item_hash)
    end
  end
end
```

Listing 6.29: LoopOS UI - Top Sidebar Class.

Subsequently, the Topsidebar template iterates through its *menu* variable and renders each item only if its *can_view* function returns true. This approach not only safeguards the possibility of role-based access within the sidebar but also enables rendering each item according to its respective template, facilitating the seamless addition of future items if needed. This practice should be consistently adhered to throughout the development of the LoopOS UI. Listing 6.30 demonstrates the code used to render the Topsidebar.

```
<% @menu.each do |item| %>
    <%= render item if item.can_view? %>
<% end %>
```

Listing 6.30: LoopOS UI - Top Sidebar Template.

Finally, the Network Manager implements its Sidebar by calling the *LooposUi::Sidebar* View Component as seen in Listing 6.31.

```
<%= render LooposUi::SidebarLayoutComponent.new do |
   component| %>
  <% component.top_sidebar(menu: LooposUi::Sidebar.
     configuration.top_items) %>
```

```
    <% component.bottom_sidebar(menu: LooposUi::Sidebar.
       configuration.bottom_items) %>
  <% end %>
```

Listing 6.31: Network Manager - Sidebar.

The code provided in Listing 6.31 is the only code required in the sidebar's view file for its construction in the Network Manager. This approach ensures a clean and organized development experience for LoopOS UI users. The content sent to the *top_sidebar* and *bottom_sidebar* elements is neatly managed due to its origin in the Sidebar configuration class within LoopOS UI, as seen in Listing 6.32. This class holds the variables responsible for the sidebar content.

```
module LooposUi
  module Sidebar
    class Configuration
      attr_accessor :top_items, :bottom_items
    end

    class << self
      attr_writer :configuration

      def configuration
        @configuration ||= Configuration.new
      end

      def configure
        yield configuration
      end
    end
  end
end
```

Listing 6.32: LoopOS UI - Sidebar Configure.

The configuration class is populated in an initializer file that runs during the initialization of the Network Manager. This ensures that the sidebar content is properly defined and ready for use when the Network Manager starts while simplifying the development process and enhancing the overall user experience for LoopOS UI developers.

With the completion of the Sidebar View Component, the development process comes to an end. Figure 6.57 provides a Class Diagram illustrating the relationships between all View Components within the Sidebar.

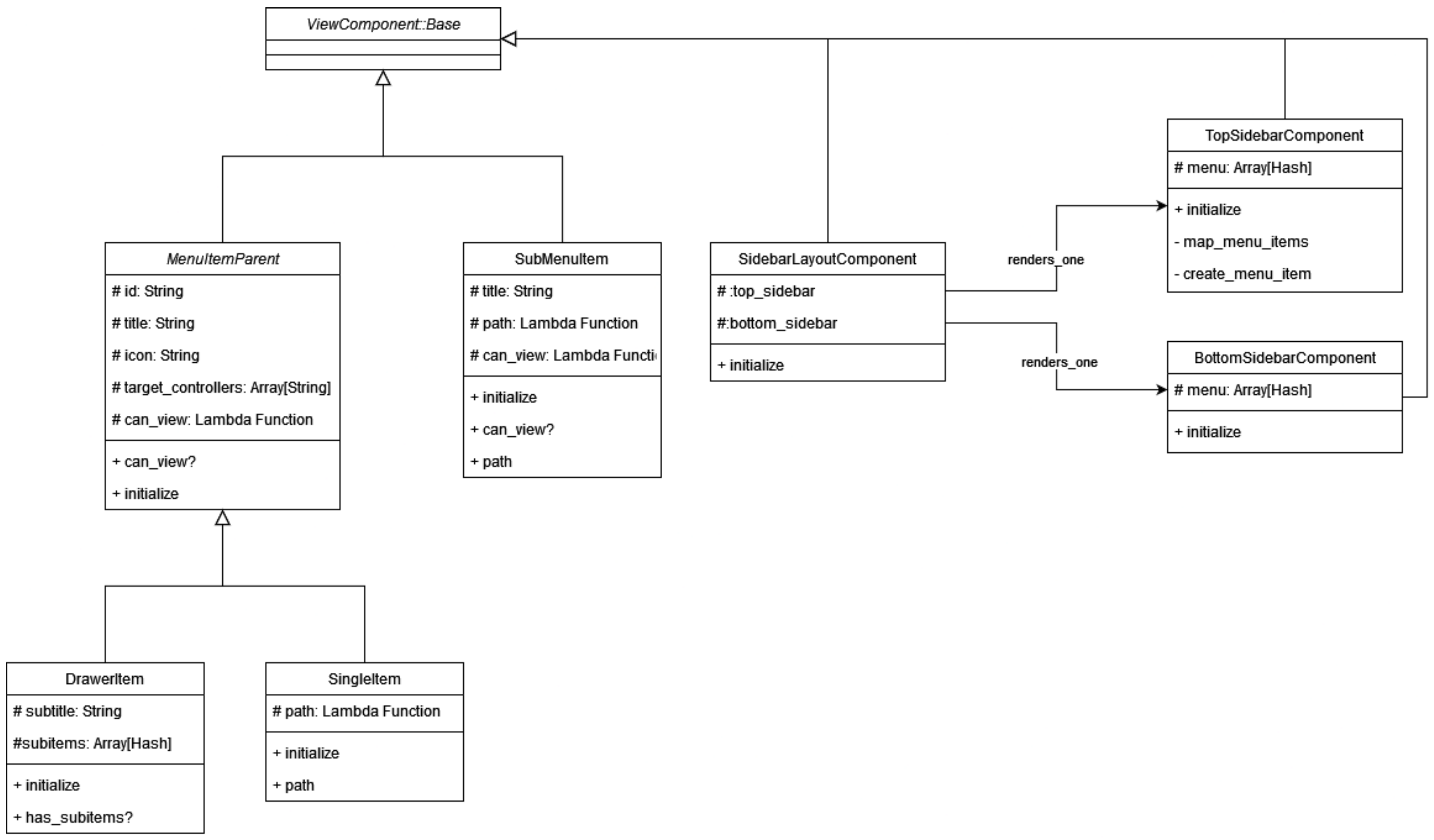


Figure 6.57: LoopOS UI: Sidebar View Component Class Diagran.

◆ Conclusion

In summary, Figure 6.58 provides a comprehensive Class Diagram depicting the structure and relationships of all View Components developed for the LoopOS UI. Additionally, Figure 6.59 illustrates the organization and hierarchy of the View Components source files, revealing the underlying structure that enabled the creation of the layouts and their components.

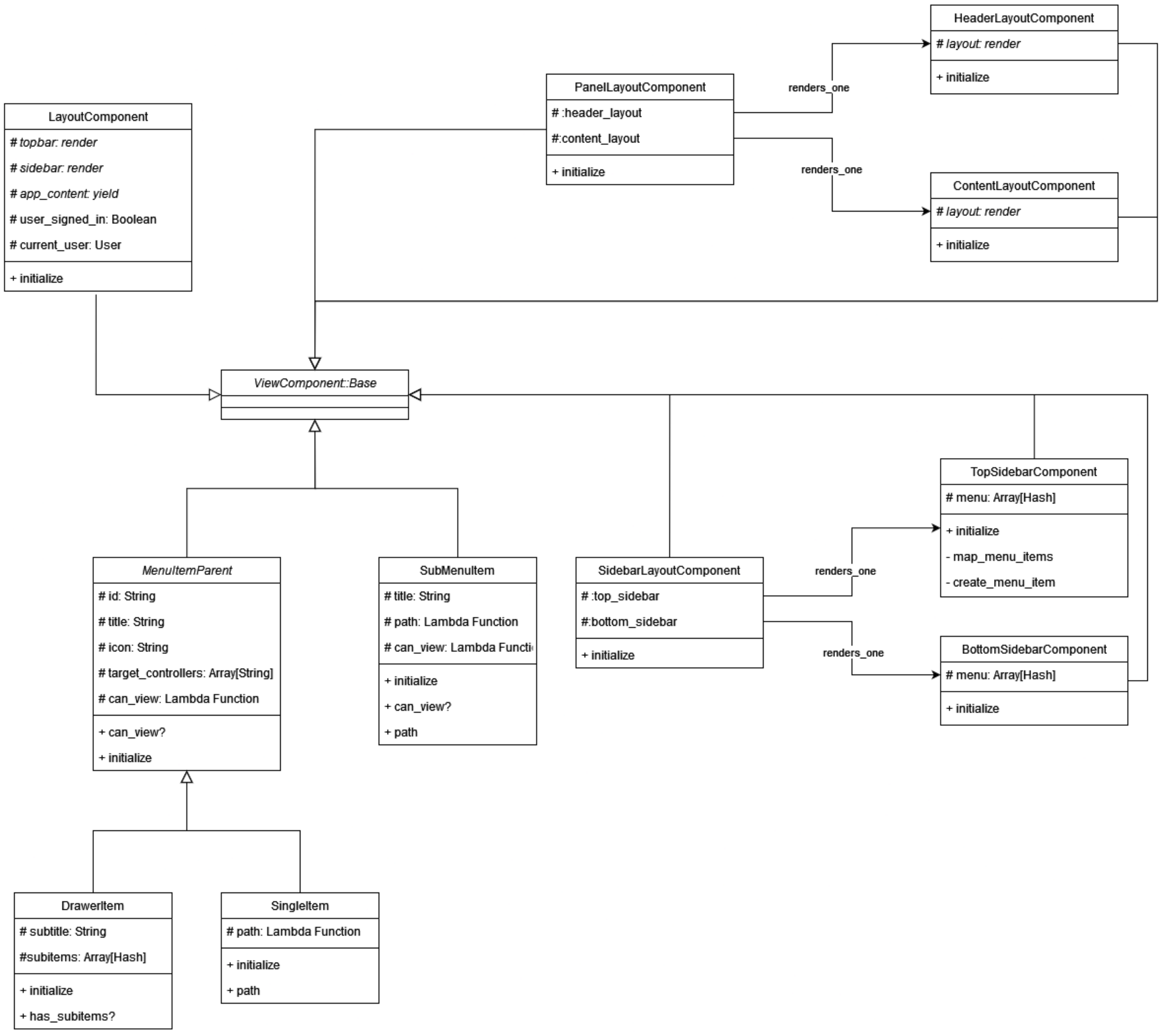
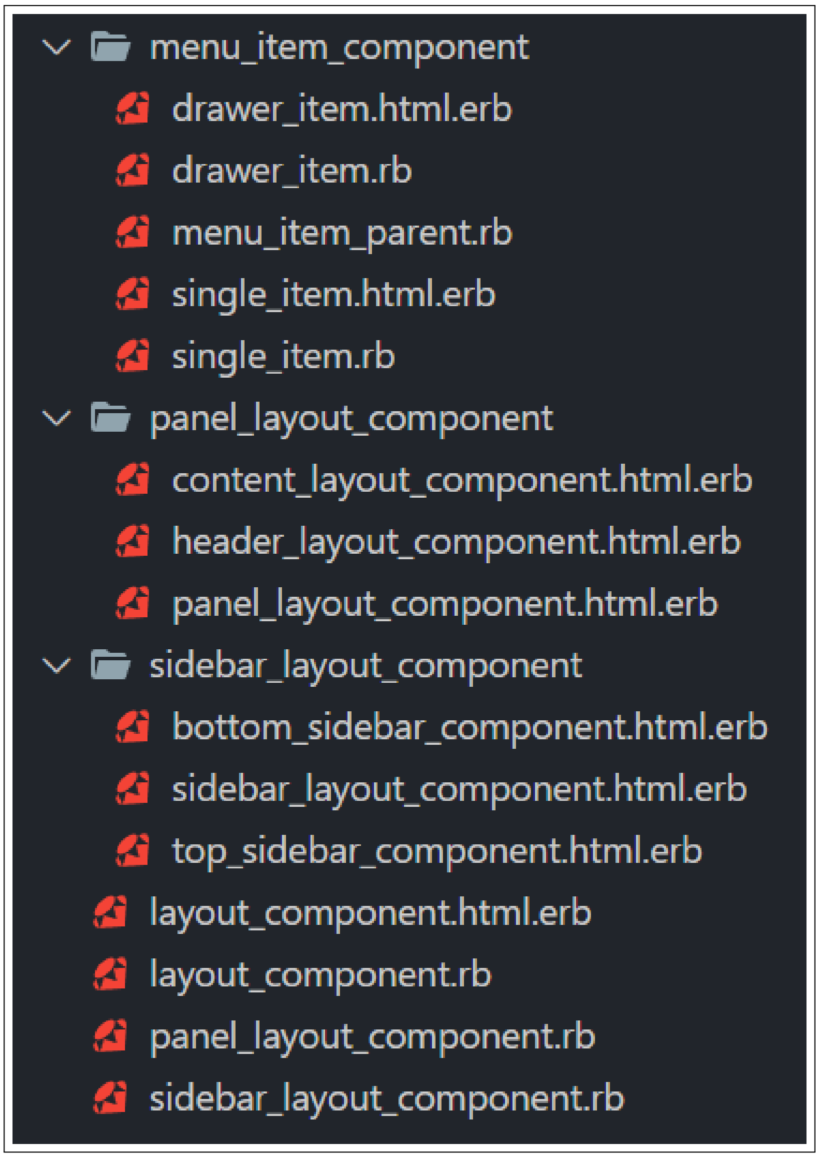


Figure 6.58: LoopOS UI: Class Diagram.



```
menu_item_component
    drawer_item.html.erb
    drawer_item.rb
    menu_item_parent.rb
    single_item.html.erb
    single_item.rb
panel_layout_component
    content_layout_component.html.erb
    header_layout_component.html.erb
    panel_layout_component.html.erb
sidebar_layout_component
    bottom_sidebar_component.html.erb
    sidebar_layout_component.html.erb
    top_sidebar_component.html.erb
layout_component.html.erb
layout_component.rb
panel_layout_component.rb
sidebar_layout_component.rb
```

Figure 6.59: Final View Components created.

In contrast with the LoopOS Onboarding, since this Engine was not fully developed, it did not have a final phase for overall testing. However, as explained in Chapter 5, Section 5.2, all this development that came before was split into tasks with its appropriate lifecycle, including multiple stages of testing by me and colleagues in 1QA and Team Review phase. Chapter 7 - Testing will overview the additional testing conducted of this application.

While some functional requirements have been successfully met, others are in progress or require additional attention. The groundwork laid in the current state of LoopOS UI provides a solid foundation for future development and implementation of the remaining features. Chapter 7 conducts the testing for LoopOS UI and analyzes the completion of the requirements.

The subsequent section will delve into a risk analysis of the risks identified in Chapter 4, Section 6.3.

6.3 Post-development Risks Analysis

This final section of the Development chapter aims to conduct a brief analysis of the risks identified through the progress of this thesis to evaluate the occurrence of the risks and the effectiveness of the mitigation plans.

[R-1] - Incomplete or unclear requirements leading to incorrect implementation or unsatisfied customer needs

Mitigation Strategy: Perform detailed requirements gathering and analysis, and ensure clear communication with stakeholders throughout the project.

This risk did not manifest itself during development. While it could have greatly impacted the development, the mitigation plan to perform a detail requirement gathering and to verify it with the company significantly minimized the probability of it occurring.

[R-2] - Infeasible requirements that cannot be implemented within the available resources or technology constraints

Mitigation Strategy: Communicate if a requirement seems unlikely to be completed on time, to be reassigned or be given help.

Similar to the earlier risk, this issue did not surface during the development phase. The successful execution of the mitigation plan involved effectively communicating to the company the necessity for additional support or the potential reallocation of assigned requirements (in cases where it would affect the successful development of this thesis).

[R-3] - Poor coding practices that make the code difficult to maintain or modify.

Mitigation Strategy: Adopt good coding practices and conduct thorough unit testing to ensure code maintainability and ease of modification.

After undergoing coding reviews conducted by company colleagues and adhering to coding practices, whether assessed manually or automatically (utilizing Rails gems such as *rubocop*), the successful implementation of the mitigation plan ensured that the risk was effectively avoided.

[R-4] - Insufficient product control leading to deviations from the original project scope.

Mitigation Strategy: Establish a well-defined development process that includes robust *project control measures.*

Alongside coding reviews, the product underwent a comprehensive testing approach. Initially, a colleague from the company conducted the first Quality Assurance (1QA) test of any given task, followed by an internal team review meeting at the end of the sprint. Finally, the Product Owner conducted a thorough review. This development process effectively mitigated the risk of the product deviating from the defined project scope.

[R-5] - Development System: Lack of experience

Mitigation Strategy: Training on required tools, and leverage experienced team members to mentor.

Initially, the lack of experience with Ruby on Rails was mitigated with the tutorials indicated by the company and additional material I found on my own. Additionally, the development of minor feature enhancements, bug fixes, and testing across various applications aided in a better understanding of the web app framework. This allowed a swift development and testing of LoopOS Onboarding.

For LoopOS UI, the lack of expertise in asset pipelines and frontend development hindered the development process due to causing delays. Furthermore, since the creation of Rails Engines that handle frontend development was new to the company, limited assistance was available, which also delayed the setup of this Engine.

However, having successfully employed the mitigation strategy to the best of my abilities, the consequences of this risk manifesting itself were reduced.

[R-6] - Communication breakdowns leading to misunderstandings and missed deadlines

Mitigation Strategy: Foster a culture of open and effective communication among team members, establish clear expectations and timelines for deliverables and use communication software or conduct in-person meetings.

Throughout the development process, this mitigation strategy was effectively implemented. Daily meetings with the assigned team and weekly meetings with the company maintained alignment and ensured everyone was on the same page. Communication software such as Click Up, and Microsoft Teams kept all team members informed and up-to-date on project progress, facilitating seamless collaboration.

By implementing these strategies, the risk was successfully mitigated.

[R-7] - Tight schedule leading to potential quality issues or delays due to balancing the internship and university work

Mitigation Strategy: Prioritize tasks based on their level of importance and urgency *and promptly communicate when these deadlines seem infeasible.*

The first semester of the thesis allowed for a more flexible schedule due to the reduced workload at the company. This provided an opportunity to focus more on university work and ensure that courses were completed successfully. Additionally, early collaboration with the company helped to align expectations and identify potential scheduling conflicts.

During the second semester, the full-time internship at the company required a dedicated focus on the thesis project. However, the absence of university courses minimized the workload and allowed for a more concentrated effort on the project. This allowed for efficient task prioritization and proactive communication with the company to address potential delays.

Hence, this thesis maintained a balance between university requirements and project deadlines, ensuring the successful completion of both.

[R-8] - Company's need to allocate resources to other parts of LoopOS may lead to delays

Mitigation Strategy: Aid in the development of other project areas so the development may processed.

As outlined in the Work Plan, there were times when the focus shifted towards enhancing and testing minor aspects of the Network Manager and other LoopOS applications. This was driven by the dynamic nature of the company environment, where adherence to the product lifecycle and the need to complete releases or test tasks from team members took priority.

However, extending contributions beyond the thesis project's immediate scope was advantageous. It ensured the smooth progression of the overall LoopOS development and yielded several benefits, including improved stability in the environment where my applications resided. Furthermore, testing other tasks provided an opportunity to evaluate my work. In return, my contributions were subjected to rigorous testing, and I gained improvement suggestions from team members.

Therefore, the mitigation strategy helped minimize the risk of delays associated with resource allocation shifts while also further reinforcing the project's overall quality

[R-9] - Changes in the way of accomplishing objectives may lead to overall changes in the thesis (Objectives, State of the Art, Requirements, etc.).

Mitigation Strategy: Maintain thorough documentation of the initial thesis focus and any subsequent modifications. This documentation serves as a reference point, ensuring clarity on its evolution and the reasons behind each adjustment.

To address this risk, thorough documentation of the initial thesis focus and any subsequent modifications was maintained. This documentation served as a valuable reference point, ensuring clarity on the evolution of the project and the rationale behind each adjustment.

During the first and second semesters of this thesis, there was a change in the approach to completing the proposed objectives. While all objectives were addressed, some initial planned functional requirements to be implemented in the Network Manager became a stand-alone application implementing a new User Experience (UI) underwent a significant expansion with the creation of LoopOS UI, an Engine to be employed in LoopOS applications to standardize and streamline the front-end development.

Thus, comprehensive documentation was maintained to track the project's progress, including actors, user stories, and requirements in the Appendix, and the Work Plan section depicts the change between the initial timeline and the actual implementation.

Despite needing an overall modification of the thesis, including a new background, state-of-the-art, and functional and non-functional requirements, the thesis remains compliant with the objectives. It effectively portrays the evolution of the project planning and execution throughout the thesis.

Chapter 7

Testing

The testing pyramid, also known as the test automation pyramid, essentially describes the test types the development and Quality Assurance (QA) teams should incorporate in an automated test suite. Moreover, it defines the order and frequency of such assessments. The purpose is to provide rapid feedback to ensure that code changes do not impact existing functionality. Figure 7.1 illustrates these layers.

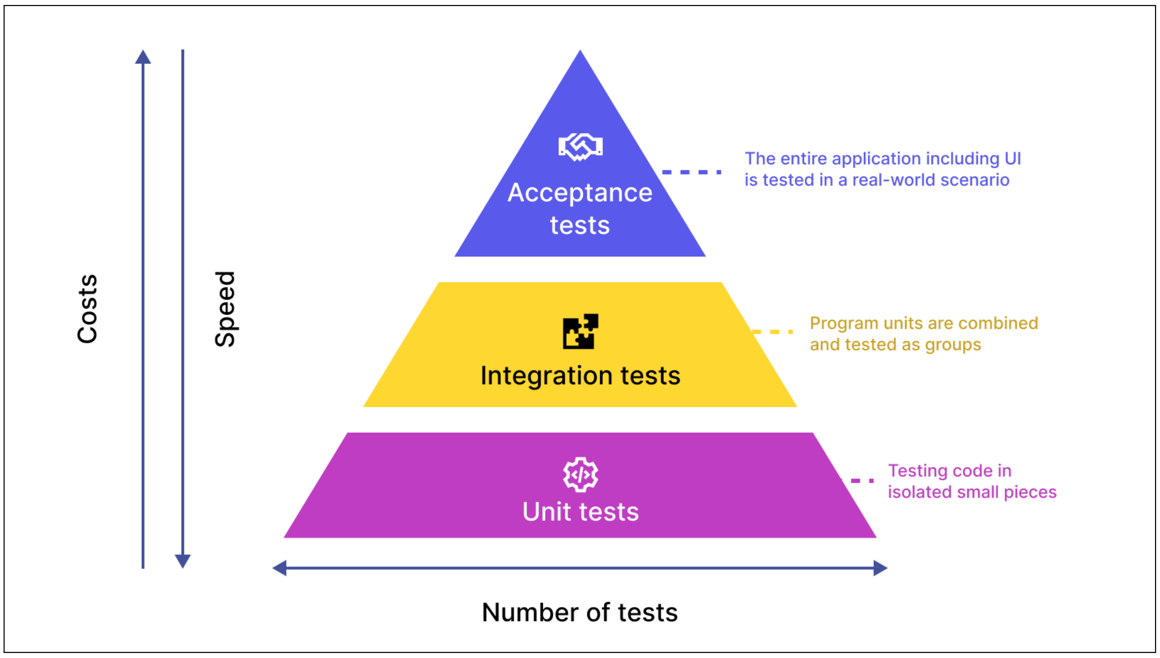

Costs
Speed
Acceptance tests
The entire application including UI is tested in a real-world scenario
Integration tests
Program units are combined and tested as groups
Unit tests
Testing code in isolated small pieces
Number of tests

Figure 7.1: Testing Pyramid [44].

The LoopOS Onboarding and LoopOS UI testing encompasses these layers: Acceptance Testing, Integration Testing, and Unit Testing.

Firstly, Acceptance Test [44][38] were the ones conducted at the end of each application to test the overall functionality of the product, such as in LoopOS Onboarding, which will be presented in Section 7.1, along with a final analysis of the functional requirements identified. Although LoopOS UI did not conduct a full-on Acceptance Test, Section 7.4 will also present an analysis of the functional requirements identified.

The product lifecycle involved multiple phases of testing per task: QA testing, Team Review testing, and Project Owner (PO) testing. These types of testing fall into the Integration Test [44][38] of the previously mentioned pyramid, where components are integrated and tested into the production environment. This testing was conducted along with the development of the LoopOS Onboarding and LoopOS UI.

Finally, Unit Tests [5][38][44] represent testing conducted on specific code units. For LoopOS Onboarding, I employed the Rails Gem Rspec [59] and Postman [51] to test the endpoints. For LoopOS UI, I leveraged the built-in testing methods within the View Components framework. This specialized testing phase will be detailed in Sections 7.2 and 7.5 for the LoopOS Onboarding and LoopOS UI, respectively.

Furthermore, I conducted an additional testing phase to test and analyze the Non-Functional Requirements (NFR) identified, although with limited depth due to the prioritization of the LoopOS UI development. This testing phase will be detailed in Sections 7.3 and 7.6 for the LoopOS Onboarding and LoopOS UI, respectively.

This comprehensive testing strategy ensured the delivery of high-quality, reliable, and functional LoopOS Onboarding and LoopOS UI applications.

7.1 LoopOS Onboarding: Acceptance Tests

As mentioned in Chapter 5, all application development was split into tasks following the company's product lifecycle, including multiple stages of testing. However, once the application was completed, there was a final phase of testing and improvements. Three main issues/missing features were encountered, which I fixed:

- **Missing translations:** Cross-language support was implemented by incorporating the Rails gem i18n [40], a library employed in other LoopOS applications. My prior experience with this gem, gained from previous tasks in the first semester of this thesis, facilitated the development process. I created two translation files, one for Portuguese and the other for English, containing the text for buttons, titles, subheadings, and other elements. Invoking these text elements in their respective locations automatically activated the appropriate language based on the user's preference.

- **User notification:** A functional requirement was added at this stage, where the user should not need to stay on the Waiting Page to be notified of the completion of the boilerplate script but instead be able to receive email notifications. Similar to the previous task, I was familiar with this process as I had previously been tasked with generating a closing shipment PDF and sending it by email. So, to notify the user of the completed onboarding process, I created an email template within the Network Manager. I added an *onboarding_email_template_id* field in the Partner Group settings. I also developed an endpoint within the Network Manager that sends an email based on the requesting application, fetching the associated Partner Group and its *onboarding_email_template_id* value. This approach allowed the Onboarding to trigger the endpoint without sending any data except for its authentication token.

- **Correct User flow:** This was my final fix to the Onboarding application. At the start of this section, I outlined the development of user states in the User model: “active”, “inactive”, “invited”, and “onboarding”. I also clarified that only the “onboarding” state would grant users access to the Onboarding application. However, this restriction only applies to entering the application, and in turn, it doesn’t prevent users from navigating back and forth between the Partner Registration Page and Boilerplate Choice Page. This could happen intentionally or unintentionally if a Partner, for instance, accidentally closes their browser. To address this, I decided to use the *extra_data*, a JSON parameter that belongs to both the user on the Network Manager and the Onboarding Application and employed an existing endpoint in the Network Manager to update the user to synchronize both models. Then, after the Partner Registration Page, I modified the user *extra_data* to contain “onboarding: created_partner” and triggered the endpoint to update the user. At the end of the Boilerplate Choice Page, I changed it to “onboarding: triggered_boilerplate,” which overrode the previous state and triggered the endpoint to update the user again. I then used this new feature to redirect the user to the appropriate page and ensure the application flow remained unidirectional.

Thus, I have successfully fulfilled all of the functional requirements I proposed for the LoopOS Onboarding app, in addition to those later added to the requirements, such as email notifications and translations. Tables 7.1 and 7.2 revisit these requirements.

Table 7.1: LoopOS Onboarding: Partner Group Functional Requirements.

ID (Priority)	**Requirement**	**Completion**
FR1 (Should)	The Partner Group should be able to customize the Onboarding landing page with their brand's logo, colors, and styles.	✓
FR2 (Must)	The Partner Group should be able to define which settings they want their Partners to fill in during the onboarding process.	✓
FR3 (Must)	The Partner Group should be able to provide their Partners with their services created in LoopOS.	✓

Table 7.2: LoopOS Onboarding: Partner Functional Requirements.

ID (Priority)	**Requirement**	**Completion**
FR4 (Could)	The Onboarding app should be accessible in multiple languages, including English and Portuguese.	✓
FR5 (Must)	Partners should be able to register on the Onboarding platform by providing necessary information such as their brand's name, logo, and user-specific details.	✓
FR6 (Must)	Partners should be able to select from a list of tailored boilerplates representing specific business logic, generating from App Instances to LoopOS Flows.	✓
FR7 (Should)	Partners should be informed about the progress of app creation through a dedicated wait page.	✓
FR8 (Could)	Partners should receive an email notification upon successfully completing the onboarding process.	✓
FR9 (Should)	Partners should be able to access a list of newly created apps after completing the onboarding process.	✓

7.2 LoopOS Onboarding: Unit Tests

The LoopOS Onboarding endpoints were subjected to two testing phases throughout their development process: initial security assessment and subsequent content validation. Table 7.3 provides a comprehensive overview of these endpoints.

Table 7.3: LoopOS Onboarding: Endpoints.

Endpoint	Method	Description
users/:user_id	GET	Retrieve information about the user with the specified user_id.
users/:user_id	POST	Update information for the user with the specified user_id.
applications/:application_id/settings	GET	Retrieve settings for the application with the specified application_id.
partner_management/partner_groups/ :partner_group_id	GET	Retrieve information about the partner group with the specified partner_group_id.
partner_management/partners	POST	Create a new partner based on the submitted data.
loop_os_scripts/boilerplates	GET	Retrieve a list of boilerplate scripts belonging to the application that made the request.
loop_os_scripts/trigger_boilerplate/:id	POST	Trigger the execution of the boilerplate script with the specified id.
loop_os_scripts/log/:id	GET	Retrieve log information for the boilerplate script with the specified log id.
onboarding/email	POST	Send an email warning on environment setup completion.

The first testing strategy was employed using Postman [51] to ensure the proper authentication of the LoopOS Onboarding endpoints. Each endpoint was thoroughly tested under three distinct scenarios:

- **No Authorization:** Requests were sent without authentication tokens in the headers. These requests were expected to fail.
- **Invalid Tokens:** Requests were sent with invalid authentication tokens. These requests were expected to result in unauthorized access errors.
- **Valid Tokens:** Requests were sent with valid authentication tokens obtained from the LoopOS authorization server (the Network Manager). These requests were expected to be successful.

The testing process yielded positive outcomes, with all endpoints passing all three scenarios. Requests without authorization or with invalid authorization tokens triggered appropriate authorization errors, and valid tokens granted access to protected resources and enabled successful operations.

RSpec [59] was employed to further validate the endpoint's functionality, a behavior-driven development (BDD) testing framework specifically designed for Ruby applications. This approach emphasizes tests being written in a way that describes the desired behavior of the code rather than the implementation details.

The following will use the Partner Creation endpoint to exemplify these tests, since it was the most complex to test in this manner. Since authentication was already tested, authentication-related aspects are excluded here. Instead, the test focuses on two distinct scenarios:

- **Valid Partner Creation:** This scenario verifies that a successful Partner creation results in a created response and the creation of a new Partner with the provided parameters and the corresponding response.
- **Invalid Partner Creation:** This scenario ensures that attempting to create a partner with invalid parameters generates an appropriate error response.

Listing 7.1 presents a portion of the RSpec code used to test the Partner creation endpoint's behavior.

```
RSpec.describe Api::V1::PartnerManagement::
   PartnersController, type: :controller do
  describe 'POST #create' do

    context 'with valid params' do
      let(:valid_params) do
        {
          name: 'Partner Name',
          slug: 'partner-slug',
          logo: 'path/to/logo.png',
          icon: 'path/to/icon.png',
          settings: { key: 'value' }
        }
      end

      it 'creates a new partner and associates it with
         the user' do
        post :create, params: valid_params
        expect(response).to have_http_status(:created)
        partner = Partner.last
        expect(partner).to eq('Partner Name')
        expect(partner.slug).to eq('partner-slug')
        #other expectations

      end
    end

    context 'with invalid params' do
      let(:invalid_params) { { name: 'Invalid Partner' }
         }

      it 'returns unprocessable entity status and error
         messages' do
```

```
            post :create, params: invalid_params
            expect(response).to have_http_status(:
               unprocessable_entity)
            expect(response.body).to include('errors')
          end
        end
      end
    end
```

Listing 7.1: LoopOS Onboarding - Create Partner Endpoint Testing.

The development of these tests involved incorporating the relevant endpoints into the swagger file. This file adheres to a standardized format for describing RESTful APIs, providing detailed information about endpoints, operations, parameters, responses, and authentication mechanisms. Maintaining this comprehensive documentation facilitates seamless utilization of the API in the future.

Table 7.4 summarizes the results of these two testing phases.

Table 7.4: LoopOS Onboarding: Endpoint Authentication Testing.

Endpoint	**Method**	**Authentication**	**Unit Test**
users/:user_id	GET	✓	✓
users/:user_id	POST	✓	✓
applications/:application_id/settings	GET	✓	✓
partner_management/partner_groups/ :partner_group_id	GET	✓	✓
partner_management/partners	POST	✓	✓
loop_os_scripts/boilerplates	GET	✓	✓
loop_os_scripts/trigger_boilerplate/:id	POST	✓	✓
loop_os_scripts/log/:id	GET	✓	✓
onboarding/email	POST	✓	✓

In conclusion, the LoopOS Onboarding endpoints have undergone thorough testing to ensure their security and functionality. The authentication of these endpoints has been verified using Postman, while their behavior has been validated through RSpec tests. Additionally, the endpoints have been documented using Swagger to facilitate their use.

7.3 LoopOS Onboarding: Non-Functional Requirement Tests

Availability

The availability of LoopOS Onboarding is a top priority, and the framework employed by deploying and managing the applications ensures this requirement. Firstly, LoopOS Onboarding is deployed and managed using Kubernetes, which employs the concept of replicas, which creates multiple instances of the application to handle traffic and maintain service even if one instance fails. This redundancy ensures that the application remains operational even if a pod or server experiences downtime, minimizing downtime and user disruptions.

Furthermore, LoopOS Onboarding is hosted on DigitalOcean, a cloud computing platform, that guarantees an uptime of 99.95% [26]. This uptime commitment, coupled with Kubernetes's replication capabilities, translates to a near-guaranteed application availability, ensuring users can access LoopOS Onboarding without interruptions.

Performance

LoopOS Onboarding relies heavily on API endpoints to facilitate user interactions and data retrieval. Optimizing these calls is crucial to enhance overall performance and user experience. Here are some effective strategies to achieve this:

- **Minimize API Requests:** Concise and efficient API calls are essential for responsive onboarding experiences. Combining multiple data requests into a single API call whenever possible minimizes the overhead of establishing and managing multiple connections. For instance, the login process was enhanced by incorporating user state information directly into the initial authentication response. This eliminates the need for separate requests to fetch user state.
- **Pagination:** When dealing with extensive datasets, pagination is a valuable tool for efficient data retrieval. Instead of loading the entire dataset at once, pagination allows for fetching data in manageable chunks, preventing unnecessary data transfer and improving page load times. For example, fetching the available boilerplates for a specific application only retrieves the relevant subset.
- **Caching for Frequently Accessed Data:** Server-side caching plays a pivotal role in enhancing performance by storing frequently requested data. Caching is implemented across various endpoints, such as application settings, accessed multiple times and therefore benefiting from this strategy.

By adopting these optimization strategies, LoopOS Onboarding can achieve faster response times, reduced data transfer, and a more streamlined user experience, making it a more efficient and user-friendly onboarding tool.

I employed the Developer Tools integrated into the Google Chrome browser to measure the loading times for the different web pages. Between each test, the browser's cache was cleared to ensure consistency. Twenty measurements were taken for each page. Figure 7.2 presents the recorded values in a Measure/Time (seconds) plot. A Critical Area was added to this plot highlighting the region where values within it signify a failure to meet the minimum load time of 2.5 seconds. This value was determined in Chapter 4 - Requirements & Risk, based on the average page load time in 2023 is 2.5 seconds on desktop [67] [49] [60].

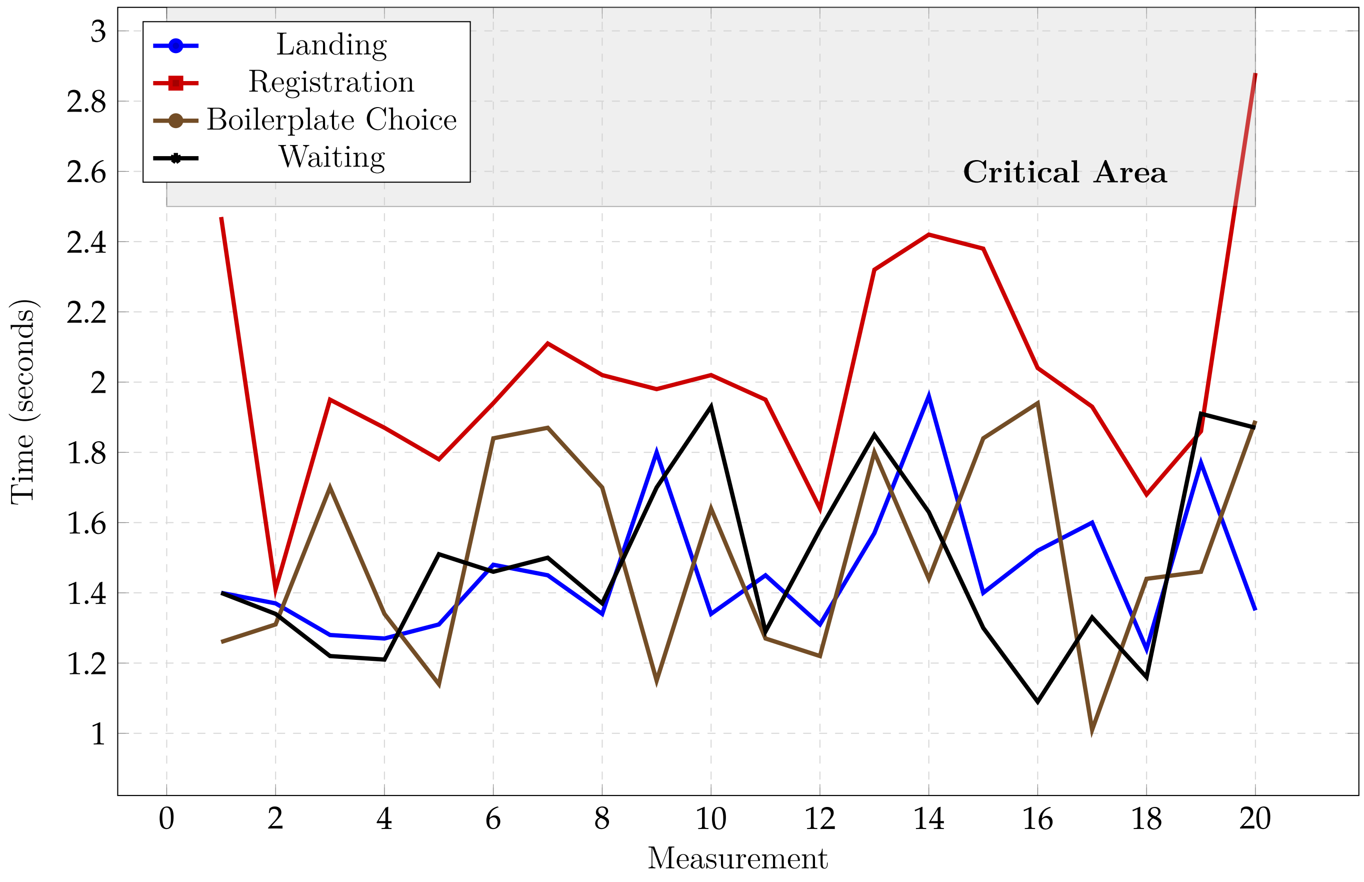


Figure 7.2: LoopOS Onboarding: Performance Testing - Measurement/Time Plot.

Finally, the averages and standard deviations of the loading time of each page were calculated. The results are presented in Table 7.5, where seconds are abbreviated with “s”.

Table 7.5: LoopOS Onboarding: Performance Testing - Average and Standard Deviation Metrics.

Page	**Avg**	**Std**	**Result**
Landing Page	1.46 s	0.189 s	✓
Partner Registration Page	2.03 s	0.323 s	✓
Boilerplate Choice Page	1.51 s	0.287 s	✓
Waiting Page	1.48 s	0.253 s	✓

While all pages achieved an acceptable average loading time of less than 2.5 seconds, the Partner Registration Page requires further optimization to ensure optimal performance under real-world conditions.

The current testing methodology used a simplified Partner Group settings schema, limiting the number and complexity of settings rendered for partner creation. However, in realistic scenarios, as discussed in Chapter 6, the Partner Registration Page will encounter more extensive and diverse settings, potentially impacting loading performance.

Furthermore, these tests were conducted in a controlled environment without concurrency. While the Kubernetes architecture mitigates this issue to some extent through its Load Balancer, with traffic redirection to available pods, a more thorough evaluation requires testing under varying concurrency levels.

In conclusion, comprehensive testing should be implemented to assess the Partner Registration Page’s performance under diverse concurrency scenarios and using different Partner Group settings schemas. This will provide a better understanding of how this page handles real-world traffic patterns and ensures its loading performance remains efficient while rendering various settings for partners to fill.

7.4 LoopOS UI: Acceptance Tests

Moving on to the testing of LoopOS UI, it s important to mention that I have not fulfilled all the functional requirements I proposed for the LoopOS UI Engine. Table 7.6 revisit these requirements.

Table 7.6: LoopOS UI: App Developer Functional Requirements.

ID (Priority)	Requirement	Completion
FR10 (Must)	Enable App Developers to seamlessly integrate the LoopOS UI Engine into their applications.	✓
FR11 (Must)	Support a progressive implementation approach for rendering components.	✓
FR12 (Must)	Provide App Developers with the ability to dynamically implement sidebars using LoopOS UI.	✓
FR13 (Should)	Provide App Developers with the ability to dynamically implement the header of show pages using LoopOS UI.	✗
FR14 (Could)	Provide App Developers with the ability to dynamically implement the content of show pages using LoopOS UI.	✗
FR15 (Could)	Provide App Developers with the ability to dynamically implement the header of index pages using LoopOS UI.	✗
FR16 (Could)	Provide App Developers with the ability to dynamically implement the content of index pages using LoopOS UI.	✗

Analyzing the Functional Requirements that were completed:

- **FR10:** Integration with applications has been successfully addressed, enabling app developers to seamlessly integrate the LoopOS UI Engine into their applications. Further integration testing in other LoopOS applications and adjustments to the asset pipeline are still needed, but the foundation for integration with LoopOS has been established.

- **FR11:** The current development method in LoopOS UI allows developers in LoopOS Core and LoopOS Hubs to implement a dynamic Sidebar and their Show and Index pages. Although implementing these pages currently involves sending their full content, it provides a functional starting point, for progressive development on these apps.

- **FR12:** Dynamic Sidebar Implementation has been completed as best as possible. The development of mockups for the bottom sidebar will continue to drive advancements in this component.

However, unfortunately, due to time constraints, **FR13** to **FR16** could not be completed. Nevertheless, comprehensive planning and documentation, including possible implementations, have been undertaken. The Sidebar's implementation is an illustrative example for developers to follow, showcasing the path to developing and implementing Show and Index pages through the Panel Layout View Component.

7.5 LoopOS UI: Unit Test

Unit testing for the LoopOS UI was conducted by focusing on the developed View Components. View Components have a dedicated testing framework that simplifies the testing process. Similar to RSpec, its a behavior-driven development (BDD) testing framework, which emphasizes tests being written in a way that describes the desired behavior of the code rather than the implementation details.

This framework also provides a comprehensive guide covering various aspects such as unit testing components, testing slots, using previews as test cases, testing components with behaviors, and offering examples and code snippets to illustrate different testing scenarios and best practices for View Component testing in Rails applications.

Listing 7.2 demonstrates a code snippet that exemplifies testing the creation of the *SingleItem* View Component and ensuring that the provided hash is correctly incorporated into the template.

```ruby
test "renders single item component" do
  item_hash = {
    id: 1,
    title: "Example Item",
    icon: "example-icon",
    target_controllers: ["example_controller"],
    can_view: -> { true },
    path: -> { "/example_path" }
  }

  render_inline(LooposUi::MenuItemComponent::SingleItem.
     new(item_hash: item_hash))

  assert_selector(".tooltip.tooltip--right p", text: "
     Example Item")
  # more asserts
end
```

Listing 7.2: LoopOS UI - *SingleItem* View Component Testing.

The View Components that underwent the most extensive testing were the *Item-Component* and the *DrawerComponent*. Table 7.7 provides a comprehensive overview of the test cases for both components and their respective outcomes.

Table 7.7: LoopOS UI: View Components Test Cases.

Test Case	**Description**	**Result**
Rendering Basic Structure	Verify that the component renders with the correct HTML structure elements (title, subtitle, icon, tooltip, etc).	✓
Link Generation	Check that the link is generated with the correct href attribute based on the provided path.	✓
Active State	Test whether the component applies the "active" class (core-button-borderless–active) when the current controller matches one of the target controllers.	✓
Inactive State	Verify that the "active" class is not applied when the current controller does not match any of the target controllers.	✓
Visibility	Based on can_view, ensure that the component is visible only when the can_view lambda returns true.	✓
Mouseover Action	Ensure the mouseover action triggers the specified JavaScript method (enginebar#hide or engine#show depending if its Drawer of Single).	✓

It should be noted that in the MouseOver Action Test Case, it is not possible to directly test it using View Components Testing. At most, it is possible to assert that the HTML has the correct data controller linked, as shown with the code in Listing 7.3.

However, with the use of RSpec, it is possible to test the controller itself. Thus, combining the two tests ensures this Test Case is a success.

```
    assert_selector(".sidebar__link[data-action='
      mouseover->enginebar#hide'][data-controller='
      enginebar']")
    end
```

Listing 7.3: LoopOS UI - *SingleItem* and *DrawerItem* View Component Testing.

In conclusion, unit testing was conducted on the LoopOS UI, particularly on View Components. These test cases examined essential aspects like rendering, link generation, active/inactive states, visibility based on conditions, and actionsthus validated the functionality and behavior of the developed LoopOS UI components.

7.6 LoopOS UI: Non-Functional Requirement Tests

Performance

I leveraged the Developer Tools integrated into the Google Chrome browser to evaluate the loading times for different web pages. To maintain consistency, I cleared the browser's cache between each test. I conducted these tests on two Network Manager environments: one with LoopOS UI and another in an older version without LoopOS UI.

Figure 7.2 presents the recorded values in a Measure/Time (seconds) plot. A Critical Area was added to this plot highlighting the region where values within it signify a failure to meet the minimum load time of 2.5 seconds. This value was determined in Chapter 4 - Requirements & Risk, based on the average page load time in 2023 is 2.5 seconds on desktop [67] [49] [60]. Additionally, the legend uses the abbreviations "w/" for with LoopOS UI and "w/o" for without LoopOS UI.

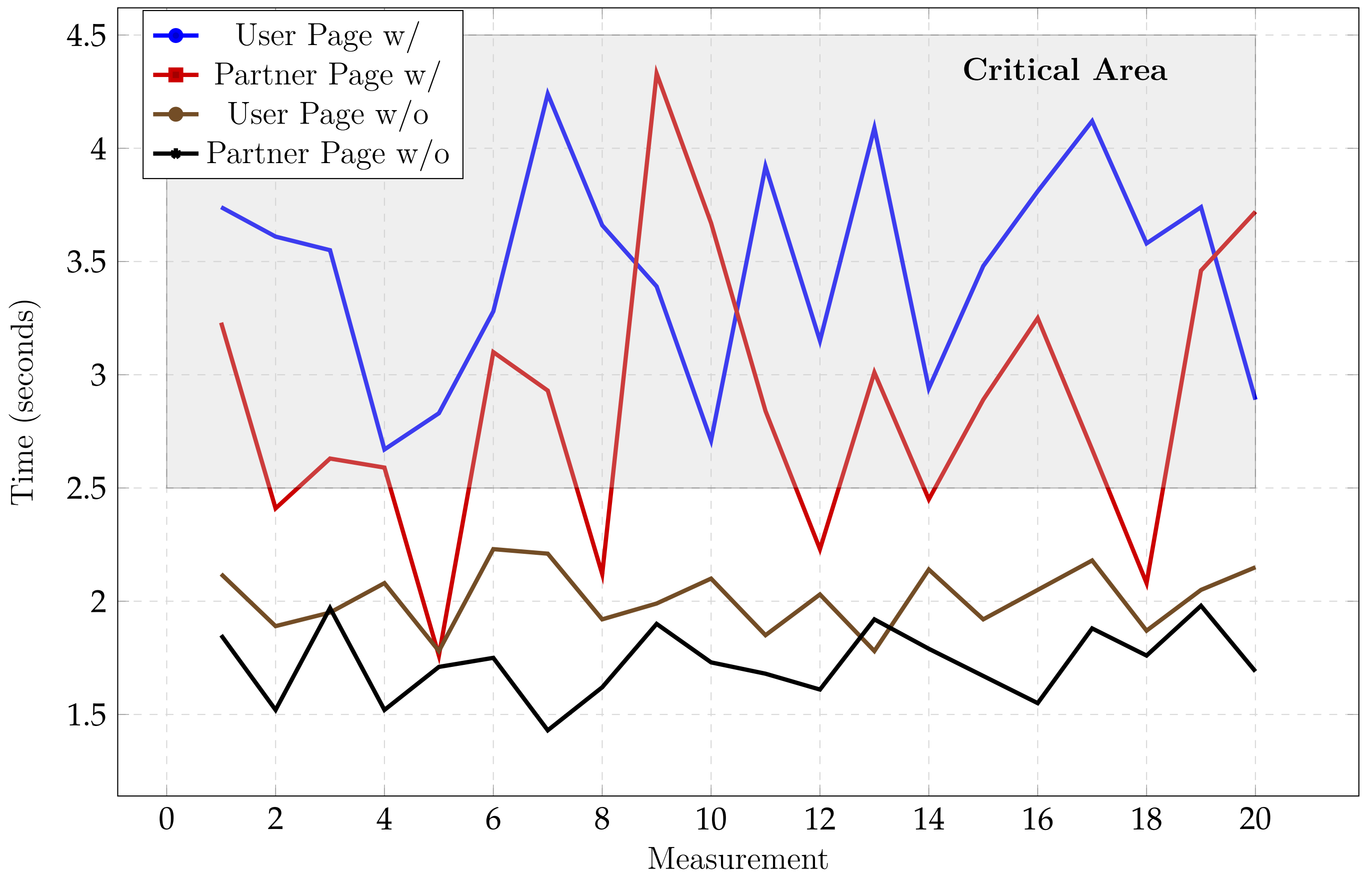


Figure 7.3: LoopOS UI: Performance Testing - Measurement/Time Plot.

Finally, the averages and standard deviations of the loading time of each page were calculated. The results are presented in Table 7.8, where seconds are abbreviated with “s”.

Table 7.8: LoopOS UI: Performance Testing - Average and Standard Deviation Metrics.

Page	Avg	Std	Result
User Page with LoopOS UI	3.47 s	0.465 s	✗
Partner Page with LoopOS UI	2.89 s	0.616 s	✗
User Page without LoopOS UI	2.01 s	0.136 s	✓
Partner Page without LoopOS UI	1.73 s	0.154 s	✓

It should be noted that LoopOS UI is not yet fully complete, so this test aimed to assess whether its use significantly delays the rendering of the Network Manager pages.

While these test results were not promising, a more definitive assessment will require subjecting LoopOS UI to scenarios involving the creation of increasingly complex components. This is because, in theory, LoopOS UI will render components of higher complexity faster due to its modular implementation.

Furthermore, despite its incomplete state, using LoopOS UI is expected to enhance performance due to using Turbo [68] throughout LoopOS Applications. Turbo’s primary function is to prevent page reloads when a user initiates an action, such as navigating to a different page. Instead, Turbo dynamically updates the page content without requiring a full refresh, significantly improving user experience.

LoopOS UI’s architecture is based on the modular reuse of components, which aligns perfectly with Turbo’s capabilities. This standardization and reusability will allow Turbo to be employed to its fullest potential, further streamlining the rendering of front-end elements and enhancing overall performance across LoopOS Applications.

Security

LoopOS UI employs Role-Based Access Control (RBAC) to safeguard user access to different pages. When a user lacks the necessary permissions, the sidebar will not display links/buttons corresponding to these pages. As LoopOS UI is developed, the Show and Index pages will incorporate RBAC checks directly within their respective builders. This means that even if a user cannot access a page through the sidebar, they will be unable to navigate to it directly using the URL.

Any additional security measures fall within the applications where the LoopOS UI is inserted.

The sidebar was thus tested to ensure that items and subitems were appropriately hidden when they were not allowed to be viewed. Table 7.9 summarizes the tests performed and their results.

Table 7.9: LoopOS UI: Security Testing.

Class	**Test Case Description**	**Expected Outcome**	**Result**
SingleItem	Create *SingleItem* with *can_view* set to true	Appear in Sidebar	✓
SingleItem	Create *SingleItem* with *can_view* set to false	Does not appear in Sidebar	✓
DrawerItem	Create *DrawerItem* with *can_view* set to true and subitems	Appear in Sidebar	✓
DrawerItem	Create *DrawerItem* with *can_view* set to false and subitems	Does not appear in Sidebar	✓
DrawerItem	Create *DrawerItem* with *can_view* set to true and without subitems	Appear in Sidebar	✓
DrawerItem	Create *DrawerItem* with *can_view* set to false and without subitems	Does not appear in Sidebar	✓
DrawerItem	Create *DrawerItem* with *can_view* set to true and all subitems set to true	All appear in Sidebar	✓
DrawerItem	Create *DrawerItem* with *can_view* set to true and some subitems set to true	Only allowed appear in Sidebar	✓
DrawerItem	Create *DrawerItem* with *can_view* set to true and all subitems set to false	Only *DrawerItem* appears in Sidebar	✓

All test cases were executed successfully. However, the last test case, "Create *DrawerItem* with *can_view* set to true and all subitems set to false" raised a question: should a *DrawerItem* appear if all subitems are not allowed to be viewed? Additionally, should a *DrawerItem* have no subitems, should it appear?

To address these concerns, I added logic to prevent the rendering of *DrawerItems* that lack subitems or where all subitems are not visible to the user. This logic should be clearly explained in future documentation for the Sidebar View Component to ensure developers understand the behavior and do not mistakenly interpret a missing *DrawerItem* as an error.

Chapter 8

Conclusion

This thesis has successfully developed and enhanced key components within the LoopOS technological ecosystem, namely the LoopOS Onboarding and LoopOS UI. The project involved a comprehensive examination of development processes, risk mitigation strategies, and testing approaches.

Effective risk mitigation strategies were employed throughout the development phases, including documentation of changes, open communication, and task prioritization. These strategies ensured alignment with defined project objectives. A robust testing approach, encompassing unit tests and non-functional requirement assessments, validated the reliability and performance of the developed applications. The culmination of these efforts resulted in the successful completion of the LoopOS Onboarding and laid the groundwork for future LoopOS UI applications, significantly contributing to LoopOS.

The LoopOS Onboarding application has reformed the onboarding process for Partners and their environments, replacing the manual and time-consuming approach previously handled through the Network Manager. This streamlined solution empowers Partners to effortlessly create an account and generate their entire LoopOS environment with a single click, tailored to the specific needs of their associated Partner Groups. This flexibility addresses Partners' diverse needs and unique business models, enabling seamless integration into LoopOS.

LoopOS UI has effectively established a framework for creating modular and maintainable user interfaces within the LoopOS ecosystem, ensuring a consistent and cohesive user experience across all LoopOS applications. This is achieved by adopting a View Components architecture, which successfully encapsulates UI elements into self-contained components and layouts that promote reusability and enhance the codebase's overall organization. Thus, establishing a single source of truth is necessary throughout all LoopOS applications where LoopOS UI is and will be employed.

Furthermore, the planning and development of LoopOS UI lays a foundation for future work:

- **Extended LoopOS UI Functionality:** Future endeavors should prioritize fulfilling remaining functional requirements for LoopOS UI, particularly the dynamic implementation of headers and content for Show and Index pages. This would enhance the flexibility and customization capabilities of LoopOS UI across various applications.
- **Full Integration Testing:** While LoopOS UI has undergone integration testing involving the Network Manager, comprehensive testing with other LoopOS Applications is essential to ensure seamless interoperability and integration.
- **Security Refinement:** While security measures have been implemented, ongoing work on LoopOS UI must involve continuous security refinement. This includes employing Pundit, as mentioned in Chapter 4 - Requirements & Risks, to establish policies and standardize role-based access control across all LoopOS applications.
- **Automation Testing:** The implementation of automated testing tools and frameworks, such as Rspec [59] and View Component Testing [70], should remain a core aspect of LoopOS UI development.
- **User Feedback Integration:** As proposed in Chapter 3 - State of the Art, the Network Manager could incorporate user feedback mechanisms that would provide valuable insights for iterative improvements, ensuring user satisfaction and usability. Future development of LoopOS UI could aid in achieving this objective by adding it to other LoopOS applications.

Building upon the existing capabilities of the Network Manager, I propose in Chapter 3 - State of the Art, the integration of comprehensive monitoring functionalities, aligning with similar tools used for application deployment and management. In the first semester of this thesis, I had defined the following functional requirements for monitoring App Instances:

- **Real-time monitoring of infrastructure metrics:** Monitor CPU and memory usage of all App Instances in real-time, providing a dynamic view of resource utilization.
- **Historical monitoring of infrastructure metrics:** View historical data for CPU and memory usage of all App Instances, allowing for retrospective analysis and trend identification.
- **Resource alarm notification:** Configure and receive timely notifications regarding resource-related issues, such as excessive CPU or memory consumption, to proactively address performance bottlenecks.

Given the ability to interact with the Kubernetes pods hosting these App Instances and the presence of passive monitoring tools like Prometheus [52] within the Kubernetes environment, it is feasible to retrieve relevant metrics and seamlessly integrate them into the Network Manager's interface. This integration will enhance the overall observability and management capabilities of LoopOS, empowering users to gain deeper insights into App Instance performance and resource consumption.

In collaboration with the Design Team, I aided in developing mockups showcasing the envisioned App Instance monitoring interface. Figure 8.1 illustrates the proposed App Instance dashboard, providing a clear and intuitive representation of real-time and historical resource utilization metrics. Additionally, Figure 8.2 depicts the anticipated design for resource alarm notifications, ensuring prompt and informative alerts regarding performance issues.

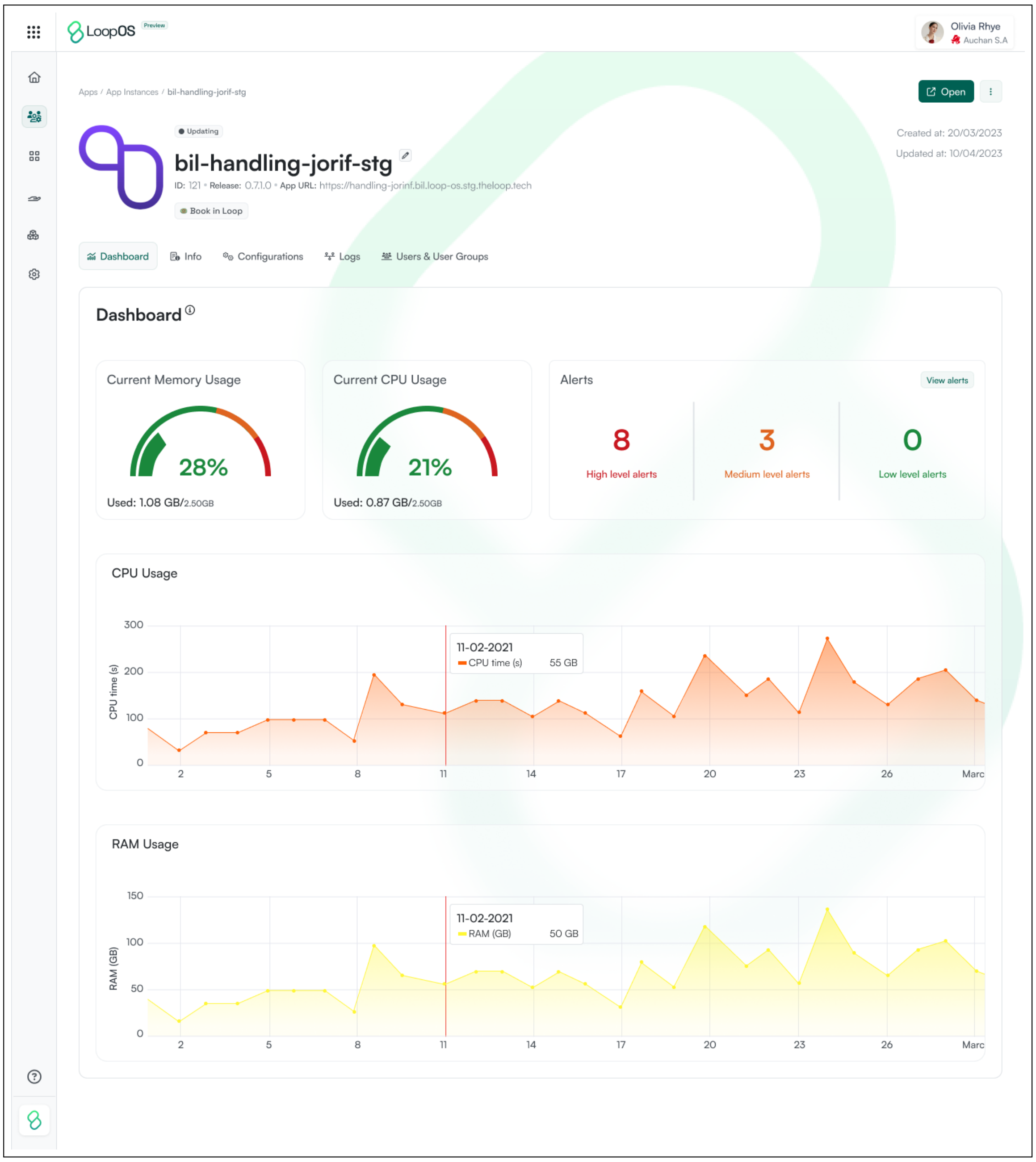
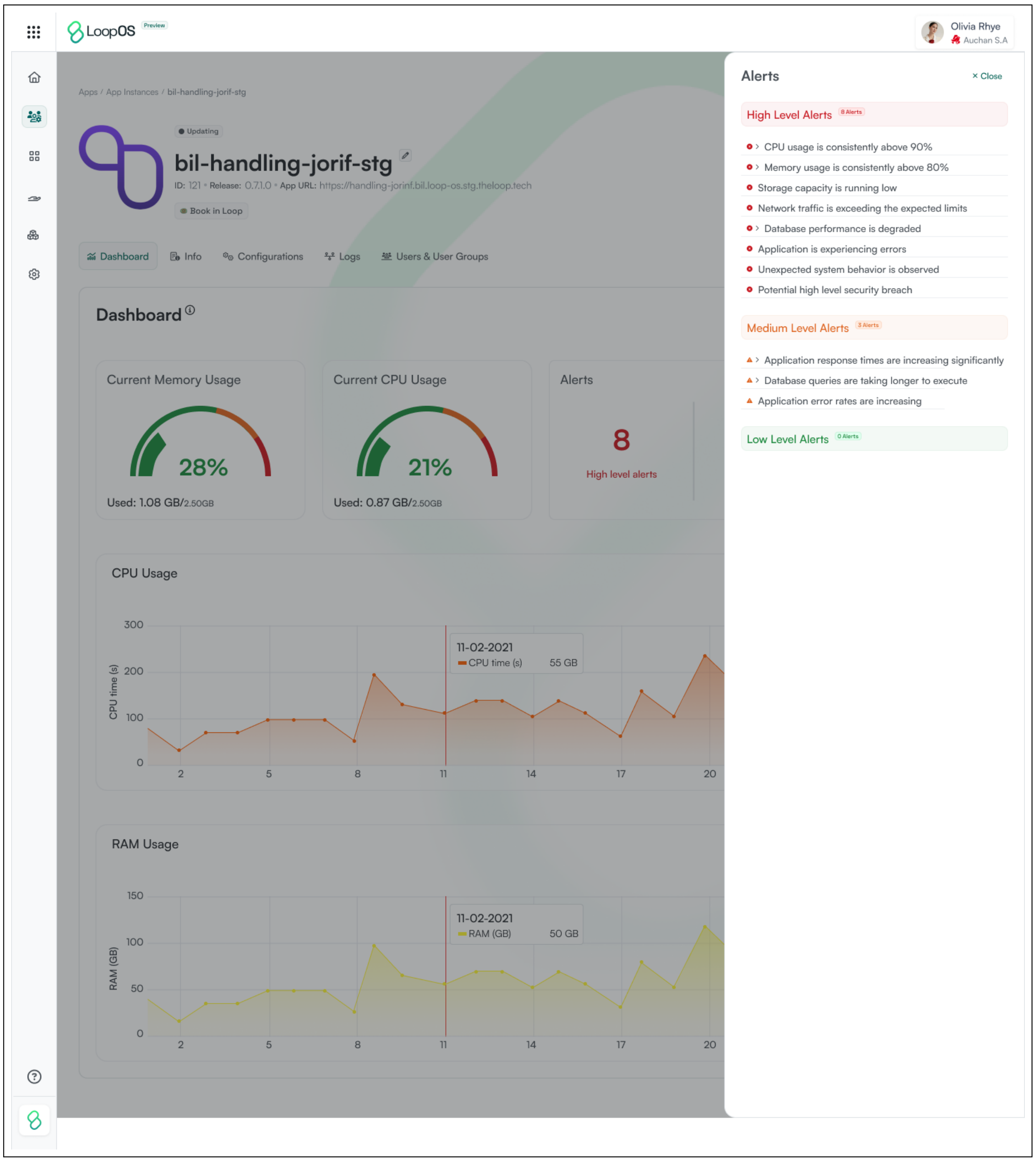


Figure 8.1: Network Manager Mockups - App Instance Dashboard.



- CPU usage is consistently above 90%
- Memory usage is consistently above 80%
- Storage capacity is running low
- Network traffic is exceeding the expected limits
- Database performance is degraded
- Application is experiencing errors
- Unexpected system behavior is observed
- Potential high level security breach

- Application response times are increasing significantly
- Database queries are taking longer to execute
- Application error rates are increasing

Figure 8.2: Network Manager Mockups - App Instance Alerts.

However, thorough evaluation and implementation studies are necessary to confirm its feasibility and suitability within the LoopOS ecosystem.

In conclusion, the thesis has successfully contributed to the LoopOS ecosystem's evolution into a more robust, feature-rich, and user-friendly platform. The development of LoopOS Onboarding and LoopOS UI have addressed the identified problems, including the impracticality of manual Partner onboarding, manual environment instantiation, and the user-unfriendliness of the Network Manager. LoopOS Onboarding has automated the environment creation process, while LoopOS UI has taken the first steps to enhance the overall user experience across all LoopOS applications.

On a final note, my internship journey has been an invaluable learning experience that has provided me with a comprehensive understanding of the product development lifecycle, from conceptualization and planning to hands-on implementation. The opportunity to work on real-world projects with clients has sharpened my ability to understand their unique needs and translate them into functional solutions.

I refined my proficiency in building robust and scalable web applications through my work with technologies like Ruby on Rails. I was also exposed to DevOps practices that have given me valuable insights into the strategies necessary to ensure these applications' efficient deployment and maintenance.

Working alongside skilled professionals has equipped me with practical insights into effective communication, agile methodologies, and the importance of a cohesive team in delivering successful projects.

This internship has not only deepened my technical knowledge but has also enriched my understanding of the collaborative and dynamic nature of the software development industry. It has also strengthened my soft skills, particularly my ability to work effectively as part of a team, meet deadlines, and solve problems creatively.

This transformative chapter in my professional growth has prepared me to confidently pursue a career in software development by actively contributing to the creation of innovative and impactful solutions, exemplified by projects like LoopOS.

References

[1] 15Five (2023). Don't just measure performance. manage it, with heart. https://www.15five.com/why-15five/. Accessed: 2023-11-10.

[2] Arora, R., Mutz, D., and Mohanraj, P. (2023). *Innovating for The Circular Economy: Driving Sustainable Transformation.* CRC Press. Accessed: 2023-20-03.

[3] Authors, T. K. (2023). kubernets. `https://kubernetes.io/`. Accessed: 2023-30-05.

[4] AvoHQ (2023). Build ruby on rails apps 10x faster. `https://avohq.io/`. Accessed: 2023-07-04.

[5] AWS (2022). What is unit testing? https://aws.amazon.com/what-is/unit-testing/. Accessed: 2024-01-02.

[6] Azure, M. (2023). Azure app service. https://azure.microsoft.com/en-us/services/app-service/. Accessed: 2023-11-10.

[7] BambooHR (2023). Hr, payroll, benefits. the complete hr software. https://www.bamboohr.com/. Accessed: 2023-11-10.

[8] Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice.* SEI series in software engineering. Addison-Wesley. Accessed: 2023-17-05.

[9] Batsov, B. and contributors, R. (2023). Rubocop - the ruby linter/formatter that serves and protects. `https://rubocop.org/`. Accessed: 2023-26-04.

[10] Beck, K. and et al. (2001). Manifesto for agile software development. `https://agilemanifesto.org/principles.html`. Accessed: 2023-12-04.

[11] Carr, M., Konda, S., Monarch, I., Ulrich, F., and Walker, C. (1993). Taxonomy-based risk identification. *Technical Report CMU/SEI-93-TR-6 ESC-TR-93-183.* Accessed: 2023-03-05.

[12] Clickup (2023). One app to replace them all. `https://clickup.com/`. Accessed: 2023-06-30.

[13] Cloud, G. (2023). Google app engine. https://cloud.google.com/appengine/. Accessed: 2023-11-10.

[14] co., T. L. (2016). Book in Loop: loja em segunda mão de material escolar. `https://bookinloop.pt/`. Accessed: 2023-06-03.

[15] co., T. L. (2019). The Loop Co.: b2b tech solutions. https://www.theloop.pt/. Accessed: 2023-06-03.

[16] co., T. L. (2021). LoopOS. https://bookinloop.pt/. Accessed: 2023-06-03.

[17] Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison-Wesley signature series. Addison-Wesley. Accessed: 2023-17-05.

[18] Commission, E. (2022a). A European Green Deal. https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal_en. Accessed: 2023-20-03.

[19] Commission, E. (2022b). Circular Economy Action Plan. https://environment.ec.europa.eu/strategy/circular-economy-action-plan_en. Accessed: 2023-20-03.

[20] Commission, E. (2022c). Sustainable Product Initiative. https://ec.europa.eu/info/law/better-regulation/have-your-say/initiatives/12567-Sustainable-products-initiative_en. Accessed: 2023-20-03.

[21] Commission, E. (2023). Circular Economy. https://environment.ec.europa.eu/topics/circular-economy_en. Accessed: 2023-20-03.

[22] CTT (2023). Ctt - correios de portugal. https://www.ctt.pt. Accessed: 2023-12-20.

[23] datatracker (2012a). Rfc 6749 - the oauth 2.0 authorization framework: 1.3.3. resource owner password credentials. https://datatracker.ietf.org/doc/html/rfc6749#section-1.3.3. Accessed: 2023-26-04.

[24] datatracker (2012b). Rfc 6749 - the oauth 2.0 authorization framework: 1.3.4. client credentials. https://datatracker.ietf.org/doc/html/rfc6749#section-1.3.4. Accessed: 2023-26-04.

[25] De Giovanni, P. and Folgiero, P. (2023). *Strategies for the Circular Economy: Circular Districts and Networks*. Routledge-Giappichelli Studies in Business and Management. Taylor & Francis. Accessed: 2023-23-03.

[26] DigitalOcean (2022). Digitalocean kubernetes service level agreement (sla). https://docs.digitalocean.com/products/kubernetes/details/sla/. Accessed: 2023-12-17.

[27] DigitalOcean (2023). Dream it. build it. grow it. https://www.digitalocean.com/. Accessed: 2023-12-17.

[28] Divante (2019). 10 companies that implemented the microservice architecture and paved the way for others. https://www.divante.com/blog/10-companies-that-implemented-the-microservice-architecture-and-paved-the-way-for-others. Accessed: 2023-10-04.

[29] Docker (2023). Dcoker: Develop faster. run anywhere. https://www.docker.com/. Accessed: 2023-10-06.

[30] Erl, T. (2008). *SOA Design Patterns*. The Prentice Hall Service Technology Series from Thomas Erl. Pearson Education. Accessed: 2023-10-06.

[31] Fernandez, T. (2022). 5 options for deploying microservices. https://semaphoreci.com/blog/deploy-microservices#option-5-deploy-microservices-as-serverless-functions. Accessed: 2023-19-04.

[32] Floren, S. (2021). Implementation and analysis of authentication and authorization methods in a microservice architecture : A comparison between microservice security design patterns for authentication and authorization flows. Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS). Accessed: 2023-03-04.

[33] Foundation, E. M. (2014). Towards the Circular Economy: Accelerating the Scale-up across Global Supply Chains. https://www3.weforum.org/docs/WEF_ENV_TowardsCircularEconomy_Report_2014.pdf. Accessed: 2023-20-03.

[34] GrafanaLabs (2023). Operational dashboards for your data here, there, or anywhere. https://grafana.com/. Accessed: 2023-22-06.

[35] Group, T. P. G. D. (2023). The world's most advanced open source relational database. https://www.postgresql.org/. Accessed: 2023-11-10.

[36] Gusto (2023a). App gusto demo. https://app.gusto-demo.com/. Accessed: 2023-11-10.

[37] Gusto (2023b). Hire, pay, and manage your team all in one place. https://www.gusto.com/. Accessed: 2023-11-10.

[38] headspin (2022). The testing pyramid: Simplified for one and all. https://www.headspin.io/blog/the-testing-pyramid-simplified-for-one-and-all. Accessed: 2024-01-02.

[39] Heroku (2023). What is heroku? https://www.heroku.com/. Accessed: 2023-11-10.

[40] i18n (2023). Internationalization (i18n) library for ruby. https://github.com/ruby-i18n/i18n. Accessed: 2023-12-25.

[41] Impact (2023). Partnership management platform accelerates growth. https://impact.com/partnership-management-platform/. Accessed: 2023-11-10.

[42] imperva (2023). What is minification. https://www.imperva.com/learn/performance/minification/. Accessed: 2023-12-29.

[43] Islam, Z. and Ferworn, A. (2020). A comparison between agile and traditional software development methodologies. *Global Journal of Computer Science and Technology*, page 7–42.

[44] Katalon (2020). What is unit testing? https://katalon.com/resources-center/blog/unit-testing. Accessed: 2024-01-02.

[45] Kirchherr, J., Reike, D., and Hekkert, M. (2017). Conceptualizing the circular economy: An analysis of 114 definitions. *Resources, Conservation and Recycling*, 127:221–232. Accessed: 2023-29-03.

[46] Labs, T. (2023). Automate infrastructure on any cloud with terraform. https://www.terraform.io/. Accessed: 2023-11-10.

[47] Lavanya, N. and Malarvizhi, T. (2008). Risk analysis and management: a vital key to effective project management. In *PMI® Global Congress 2008–Asia Pacific*, Newtown Square, PA. Project Management Institute. Accessed: 2023-03-05.

[48] Leapsome (2023). Build a high-performing and resilient organization. https://www.leapsome.com/. Accessed: 2023-11-10.

[49] mycodelesswebsite (2023). Website performance statistics. `https://mycodelesswebsite.com/website-load-time-statistics/`. Accessed: 2023-12-17.

[50] PartnerStack (2023). Connect. earn. grow. https://www.partnerstack.com/. Accessed: 2023-11-10.

[51] Postman (2023). Postman api platform. https://www.postman.com/. Accessed: 2024-01-02.

[52] Prometheus (2023). From metrics to insight. `https://prometheus.io/`. Accessed: 2023-22-06.

[53] Pundit (2023). Minimal authorization through oo. `https://github.com/varvet/pundit#readme`. Accessed: 2023-12-17.

[54] Rancher (2023a). Monitoring and alerting. `https://ranchermanager.docs.rancher.com/pages-for-subheaders/monitoring-and-alerting`. Accessed: 2023-12-17.

[55] Rancher (2023b). Why rancher? `https://www.rancher.com/why-rancher`. Accessed: 2023-19-04.

[56] React (2023a). Code splitting. https://legacy.reactjs.org/docs/code-splitting.html. Accessed: 2023-12-29.

[57] React (2023b). The library for web and native user interfaces. https://react.dev/. Accessed: 2023-12-29.

[58] Risener, K. (2022). A study of software development methodologies. Accessed: 2023-12-04.

[59] RSpec (2023). Behaviour driven development for ruby. https://rspec.info/. Accessed: 2023-12-25.

[60] Ryan, E. (2023). Website load time statistics: Why speed matters in 2024. `https://www.websitebuilderexpert.com/building-websites/website-load-time-statistics/`. Accessed: 2023-12-17.

[61] Salesforce (2023). Partner relationship management. https://www.salesforce.com/products/partner-relationship-management/. Accessed: 2023-11-10.

[62] Services, A. W. (2023). Aws elastic beanstalk. https://aws.amazon.com/elasticbeanstalk/. Accessed: 2023-11-10.

[63] Sommerville, I. (2015). *Software engineering*. Pearson Education Limited, 10th edition. Accessed: 2023-08-05.

[64] Stapleton, J. (1997). *DSDM, Dynamic Systems Development Method: The Method in Practice*. Addison-Wesley.

[65] Stimulus (2023). A modest javascript framework for the html you already have. https://stimulus.hotwired.dev/. Accessed: 2023-12-29.

[66] Tailwind (2023). Rapidly build modern websites without ever leaving your html. https://tailwindcss.com/. Accessed: 2023-12-29.

[67] tooltester (2023). Website loading time statistics (2023). `https://www.tooltester.com/en/blog/website-loading-time-statistics/`. Accessed: 2023-12-17.

[68] Turbo (2023). The speed of a single-page web application without having to write any javascript. https://turbo.hotwired.dev/. Accessed: 2023-12-29.

[69] ViewComponent (2023a). A framework for creating reusable, testable and encapsulated view components, built to integrate seamlessly with ruby on rails. https://viewcomponent.org/. Accessed: 2023-12-25.

[70] ViewComponent (2023b). Testing. https://viewcomponent.org/guide/testing.html. Accessed: 2024-01-02.

[71] Webpack (2023a). A modest javascript framework for the html you already have. https://webpack.js.org/. Accessed: 2023-12-29.

[72] Webpack (2023b). Tree shaking. https://webpack.js.org/guides/tree-shaking/. Accessed: 2023-12-29.

[73] Zaporozhets, D. and Sijbrandij, S. (2023). Open source software to collaborate on code. `https://gitlab.com/gitlab-org`. Accessed: 2023-12-17.

[74] Zoho (2023). Zoho workplace partner program. https://www.zoho.com/mail/help/partnerportal/partner-portal.html. Accessed: 2023-11-10.

Appendices

Appendix A

Actors: Intermediate Defense

- **LoopOS Admin:** The super user of LoopOS who is responsible for configuring the Network Manager. Typically, this role is assigned to the engineering team, who manages the infrastructure and available services.
- **LoopOS Configurator:** A user of the Network Manager who helps the partners configure their platform. This role has access to lower-level functionalities and focuses more on operational tasks.
- **Partner Admin:** User(s) with administrative privileges who manage the partner account. All requirements for this role pertain only to the specific partner they are associated with. For example, the partner admin can only manage the UserGroups directly related to their partner account.
- **Partner User:** App workers and regular users (consumers) associated with the partner account.

Appendix B

User Stories: Intermediate Defense - v1

AppReleases

a) **As a** LoopOS Admin, **I want to** setup instructions for automatic release registration, **so that I** automate the release process.

AppInstances

a) **As a** LoopOS Admin, **I want to** view, associate, and dissociate a database server to an app instance, **so that I** manage the infrastructure on which the data is recorded.

DatabaseServer

a) **As a** LoopOS Admin, **I want to** create database servers, **so that I** define and manage the database infrastructure.

b) **As a** LoopOS Admin, **I want to** choose the main database server, **so that I** designate a primary server for the database infrastructure.

c) **As a** LoopOS Admin, **I want to** delete database servers without any associations, **so that I** can clean up unnecessary resources.

d) **As a** LoopOS Admin, **I want to** view the ID of a database server, **so that I** can manage the databases used by those instances.

e) **As a** LoopOS Admin, **I want to** view, associate, and dissociate app instances to a database server, **so that I** can manage their placement and distribution.

f) **As a** LoopOS Admin, **I want to** view, associate, and dissociate partners to a database server, **so that I** can manage the ownership or association of the server.

g) **As a** LoopOS Admin, **I want to** view and edit the configuration settings of a database server, **so that I** can configure the database-specific settings.

h) **As a** LoopOS Admin, **I want to** view and edit the name of a database server, **so that I** can easily identify it.

Partners

a) **As a** LoopOS Admin, **I want to** view, associate, and dissociate database servers that are associated with a partner, **so that I** can manage the ownership or association of the database server.

b) **As a** LoopOS Admin, **I want to** view, associate, and dissociate partner services offered by a partner, **so that I** can manage partners services.

Appendix C

User Stories: Intermediate Defense - v1

AppReleases

a) **As a** LoopOS Admin, **I want to** configure resource alerts, **so that I** can manage alert settings such the minimum value threshold of a pod memory use before an alert is sent.

AppInstances

a) **As a** LoopOS Admin, **I want to** be able to decommission app instances, **so that I** can remove unused or obsolete instances from the system.

b) **As a** LoopOS Admin, **I want to** be able to monitor the CPU and memory usage of all app instances in real-time, **so that I** can ensure optimal performance and resource allocation.

c) **As a** LoopOS Admin, **I want to** be able to view the history of CPU and memory usage for all app instances, **so that I** can track performance trends and identify potential issues.

d) **As a** LoopOS Admin, **I want to** be able to view and receive resource alarms for app instances, **so that I** can promptly address resource-related issues and ensure the availability and stability of the platform.

Partners

a) **As a** LoopOS Admin, **I want to** view, associate, and dissociate partner consumption that is relevant to a partner, **so that I** can log and track my consumptions of services and items.

b) **As a** LoopOS Admin, **I want to** view, associate, and dissociate partner documents that are relevant to a partner, **so that I** can manage partner's documents.

c) **As a** LoopOS Admin, **I want to** view, associate, and dissociate partner payments related to a partner, **so that I** can manage payments performed by partners.

PartnerConsumption

a) **As a** LoopOS Admin, **I want to** view the partner consumption data in LoopOS, **so that I** log and track the consumptions of services and items provided by a partner or service provider, monitor the usage and calculate corresponding prices.

PartnerPayment

a) **As a** LoopOS Admin, **I want to** view partner payment data in LoopOS, **so that I** can track financial transactions.

PartnerDocument

a) **As a** LoopOS Admin, **I want to** be able to view PartnerDocuments, **so that I** can access and review important partner-related documents.

b) **As a** LoopOS Admin, **I want to** be able to change the invoice status, **so that I** can update the status of invoices for proper tracking and management.

c) **As a** LoopOS Admin, **I want to** be able to upload invoices, **so that I** can add new invoices to the system for record-keeping and processing.

Self-Onboarding:

a) **As a** LoopOS Admin, **I want to** be able to create, update and delete a web page by using the self-onboarding service, **so that I** can provide a partner their online own stores on the LoopOS platform.

Templates:

a) **As a** LoopOS Admin, **I want to** be able to create and delete templates, **so that I** can automate the generation of various components for partners on the LoopOS platform.

b) **As a** LoopOS Admin, **I want to** be able to automatically generate app instances using templates, **so that I** can provide partners with pre-configured app instances.

c) **As a** LoopOS Admin, **I want to** be able to automatically generate partner services using templates, **so that I** can streamline the process of setting up services for partners.

d) **As a** LoopOS Admin, **I want to** be able to automatically generate app scopes using templates, **so that I** can quickly define the access permissions for partners.

e) **As a** LoopOS Admin, **I want to** be able to automatically generate self-onboarding using templates, **so that I** can quickly define create a web-page for partners.

f) **As a** LoopOS Admin, **I want to** be able to automatically generate user groups connected to app instances and app scopes using templates, **so that I** can efficiently manage user access for partners.

g) **As a** LoopOS Admin, **I want to** be able to automatically generate flows with their settings (categories, products, protocols, pricing rules, etc.) if a core exists using templates, **so that I** can automate the creation of predefined workflows for partners.

h) **As a** LoopOS Admin, **I want to** tag components created using templates to indicate their origin, **so that I** can track and identify the source of these components.

Appendix D

Functional Requirements: Intermediate Defense - v1

Table D.1: AppReleases Functional Requirements - v1.

ID	Requirement	Priority
FR-v1-001	Setup instructions for automatic release registration	Must

Table D.2: AppIntances Functional Requirements - v1.

ID	Requirement	Priority
FR-v1-002	View/Associate/Dissociate DatabaseServer (has one)	Must

Table D.3: DatabaseServer Functional Requirements - v1.

ID	Requirement	Priority
FR-v1-003	Create DatabaseServer	Must
FR-v1-004	Delete DatabaseServer without associations	Must
FR-v1-005	Choose a main DatabaseServer	Must
FR-v1-006	View id	Must
FR-v1-007	View/Associate/Dissociate AppInstances (belongs to)	Must
FR-v1-008	View/Associate/Dissociate Partners (belongs to)	Must
FR-v1-009	View/Edit config_settings	Must
FR-v1-010	View/Edit name	Must

Table D.4: Partners Functional Requirements - v1.

ID	Requirement	Priority
FR-v1-011	View/Associate/Dissociate PartnerServices (has many)	Must
FR-v1-012	View/Associate/Dissociate Databases (has many)	Must

Appendix E

Functional Requirements: Intermediate Defense - v2

Table E.1: AppReleases Functional Requirements - v2.

ID	**Requirement**	**Priority**
FR-v2-001	Configure resource alarms	Must

Table E.2: AppIntances Functional Requirements - v2.

ID	**Requirement**	**Priority**
FR-v2-002	Decommission of App Instances	Must
FR-v2-003	Monitor all app instances infrastructure metrics in real-time (CPU and memory usage)	Must
FR-v2-004	View history of all app instances infrastructure metric (CPU and memory usage)	Must
FR-v2-005	View/Receive resource alarms	Must

Table E.3: Partners Functional Requirements - v2.

ID	**Requirement**	**Priority**
FR-v2-006	View/Associate/Dissociate PartnerDocuments (has many)	Must
FR-v2-007	View/Associate/Dissociate PartnerConsumptions (has many)	Must
FR-v2-008	View/Associate/Dissociate PartnerPayments (has many)	Must

Table E.4: PartnerConsumptions Functional Requirements - v2.

ID	**Requirement**	**Priority**
FR-v2-009	View PartnerConsumption	Must

Table E.5: PartnerPayments Functional Requirements - v2.

ID	**Requirement**	**Priority**
FR-v2-010	View PartnerPayment	Must

Table E.6: PartnerDocuments Functional Requirements - v2.

ID	**Requirement**	**Priority**
FR-v2-011	View PartnerDocument	Must
FR-v2-012	Change invoice status	Must
FR-v2-013	Upload invoice	Must

Table E.7: Self-Onboarding Functional Requirements - v2.

ID	**Requirement**	**Priority**
R-014	Create Partner Webpage	Must

Table E.8: Templates Functional Requirements - v2.

ID	**Requirement**	**Priority**
FR-v2-015	Create/Delete template	Must
FR-v2-016	Automatically generate app instances using templates	Must
FR-v2-017	Automatically generate partner services using templates	Must
FR-v2-018	Automatically generate app scopes using templates	Must
FR-v2-019	Automatically generate user groups connected to app instances and app scopes using templates	Must
FR-v2-020	Automatically generate flows with their settings (categories, products, protocols, pricing rules, etc.) if a core exists using templates	Must
FR-v2-021	Tag components created using templates to indicate their origin	Must