UNIVERSIDADE D
COIMBRA

Oleksandr Yakovlyev

Ferramentas modulares de e-commerce direcionadas para economia circular

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software, orientada pelo Professor Doutor Mário Alberto Zenha-Rela e pela Mestre Marta Mercier e apresentada e ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2023

Faculty of Sciences and Technology

Department of Informatics Engineering

Modular e-commerce tools targeting circular economy

Oleksandr Yakovlyev

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering advised by Professor Mário Alberto Zenha-Rela and by the Master Marta Mercier and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

September 2023

1 2 9 0

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Aknowledgements

I would like to begin by extending my heartfelt gratitude to my family. To my parents, whose unwavering support has been my backbone through the years of my academic journey, thank you for your endless love and patience. A special mention to my grandmother, who has always been a constant source of wisdom and comfort, and to my brother, whose cheerfulness has been an invaluable source of encouragement.

I am profoundly thankful for the guidance, support, and mentorship of my advisors, Marta Mercier and João Rodrigues. Your constant motivation and expertise have been invaluable in the completion of this dissertation.

To my advisor, Professor Mário Alberto Zenha-Rela, I extend my deepest appreciation. Your availability and extensive knowledge have been monumental in the progress and quality of this work. Thank you for always being just an email away and for offering insights that enriched my understanding and approach to this field of study.

I am also indebted to my friend Bruno Sousa for his role in the review process and for his motivation and support throughout the development of this project. Your critical perspective and constructive comments were vital in the final revisions of this dissertation.

Finally, I would like to express my gratitude to everyone who has directly or indirectly contributed to this work.

This page is intentionally left blank.

Resumo

A economia circular e o desenvolvimento sustentável têm ganho crescente interesse público e académico na última década.

Com o crescimento do comércio eletrónico e crescentes preocupações com a sustentabilidade, há atualmente ênfase forte no modelo de economia circular que destaca a redução de resíduos, prolongando a vida útil dos produtos e promovendo o consumo sustentável.

Paralelamente a estas preocupações ecológicas, o desenvolvimento e maturação das plataformas e lojas de comércio eletrónico têm aumentado a um ritmo acelerado.

Apesar desta tendência crescente, muitas empresas têm dificuldade em integrar os princípios da economia circular na sua presença digital.

Este projeto aborda esta lacuna, introduzindo um conjunto de aplicações web modulares projetadas para permitir com que as empresas adotem sem problemas as práticas da economia circular nas suas plataformas de comércio eletrónico, integrando-se com a infraestrutura digital existente.

Através de ferramentas que promovem o recondicionamento e a renovação, as empresas não só podem mitigar o impacto ambiental do consumerismo excessivo, mas também descobrir novas oportunidades de negócio.

Este documento descreve cada fase do desenvolvimento de um piloto para este projeto, que consiste em vários módulos, coletivamente conhecidos como LoopOS, e o piloto OLX 2nd Life.

várias práticas de engenharia de software são exploradas, desde o planeamento e análise até ao desenvolvimento e testes, para dar uma visão abrangente do ciclo de vida do projeto.

Palavras-chave

Comércio eletrónico, Economia circular, Software interoperável, Desenvolvimento web, Ruby on Rails

Abstract

Circular economy and sustainable development have gained increased public and academic interest over the last decade.

With the growth of e-commerce and increasing sustainability concerns, there is a stronger emphasis currently on the circular economy model that emphasizes reducing waste, extending the lifespan of products, and promoting sustainable consumption.

Alongside these ecological concerns, the development and maturing of e-commerce platforms and stores have been increasing at a rapid pace.

Despite this rising trend, many businesses struggle to integrate circular economy principles into their digital presence.

This project addresses this gap by introducing a suite of modular web-based applications designed to enable businesses to seamlessly adopt circular economy practices on their e-commerce platforms, integrating with existing digital infrastructure.

Through tools that promote refurbishing and reconditioning, businesses can not only mitigate the environmental impact of excessive consumerism but also discover new economic opportunities.

This document describes every phase of the development of a pilot for this project which consists of various modules, collectively known as LoopOS, and the pilot OLX 2nd Life.

It delves into various software engineering practices, from planning and analysis to development and testing, to give a comprehensive view of the project's lifecycle.

Keywords

E-commerce, Circular economy, Interoperable software, Web development, Ruby on Rails

This page is intentionally left blank.

Contents

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

The modern world's emphasis on sustainability and efficient resource utilization has led to the exploration of innovative solutions, such as the circular economy. The following chapter introduces the collaborative project between The Loop Co. and OLX in this cause.

The project here presented consists of designing and developing a modular solution for the circular economy market. Its purpose is to enable the integration of circular economy practices into existing business applications such as e-commerce platforms.

It will be developed by the Software Engineering Masters student Oleksandr Yakovlyev within the scope of the course Dissertation/Internship in Software Engineering in the Faculty of Sciences and Technology / Department of Informatics, under the guidance of Professor Mário Alberto Zenha-Rela, from the Department of Informatics, and Master Marta Mercier, from The Loop Company.

1.1 The Loop Company

Since its conception, and as the name suggests, the *The Loop Co.* [1] had a great focus on sustainability and circular economy. The company began in 2016, with the *Book in Loop* project, a platform for the re-utilization of school manuals. Many more ecologically conscious projects have been made possible since then, thanks to The Loop. For example *BabyLoop*[2], *Bebidas+Circulares*[3], *Do Velho se Faz Novo*[4], and *ZEROO Cups!*[5].

For the last few years, The Loop has been expanding. Recently they opened a second office in the city of Coimbra and currently employ more than 100 talented people, working remotely from all over the country.

[1] `https://www.theloop.pt/`
[2] `https://babyloop.pt/`
[3] `https://www.bebidascirculares.pt/`
[4] `https://dovelhosefaznovo.pt/`
[5] `https://do-zero.pt/zero-cups/`

1.2 Context and Motivation

This project has two main clients: The Loop Co. itself and the online marketplace company OLX[6].

As mentioned before, The Loop has had many projects related to the circular economy, however, custom software solutions have been developed for each one. Now, having the experience and insight into the requirements for these types of systems, The Loop decided that it is possible, and useful, to generalize and create a modular and reusable solution, named LoopOS.

LoopOS is a product that has been conceptualized by The Loop for some time. It aims to streamline the process of recovering and reusing second-hand/returned items. It is designed to help retailers tap into the circular economy world, without the initial costly investment necessary to explore this market.

The Loop Co. wants to create LoopOS for future internal projects and as a potential software solution to be sold to third parties, like the aforementioned retailers.

On the other hand, this will also serve as a software project for OLX, which wants to create a new buying experience for their marketplace - the *OLX 2nd Life*[7]. This marketplace currently allows buyers and sellers to exchange secondhand goods by letting people create public listings with their contact information.

OLX wants to add an alternative way of performing this exchange. They want to give the buyer the possibility to shop in the marketplace with a typical online shopping experience, as depicted in the figure 1.1, and to facilitate the seller in disposing of their item. This alternative flow, aside from being convenient, also guarantees the product quality to the end buyer and speeds up the process for the seller, as they spend less time waiting to sell their item.

Circular Economy is about adding value to products. LoopOS has been designed to bridge the gap between supply and demand by including second-hand items. The objective is not just to streamline OLX's operations but also to enhance the buying and selling experience for users, making second-hand buying as intuitive as purchasing new items.

To achieve this, OLX needs a way to source used products, pass them through Quality Assurance, and finally re-inject them into an existing online store.

These three steps fit into the flow that LoopOS wants to provide. The modularity and extensibility of the project are a major concern. However, for the scope of this dissertation, the focus will be primarily on the pilot project - OLX 2nd Life, which serves as an example of LoopOS's application and potential.

1.3 Objectives and Overview

LoopOS aspires to be a comprehensive solution, an ecosystem, that manages the complexities of a circular economy from sourcing items to reintroducing them into the market.

One of the problems LoopOS aims to solve is the great inefficiency and manual effort in the process of recovering items from customers' homes, which has been created by the

[6] `https://www.olx.pt/`

[7] `https://help.olx.pt/olxpthelp/s/article/olx-2nd-life-ganha-dinheiro-e-ajuda-o-planeta-V10`

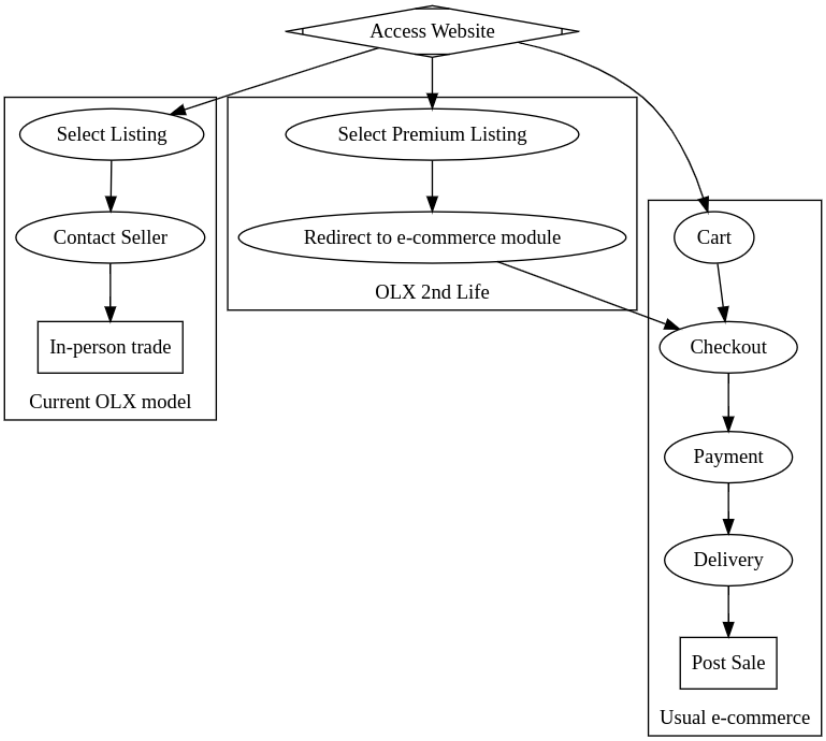

Figure 1.1: Comparison of the OLX, OLX 2nd Life, and regular e-commerce user experience flow.

rise of e-commerce and its ease of returns. Current retail channels are designed for a unidirectional flow of goods and have not adapted well to the contrary flow of returns.

As the pilot project, OLX 2nd Life will need to implement solutions to some of these problems. The solutions can be divided into two categories: the circular economy problems, and the e-commerce ones.

The usual e-commerce application deals with problems such as:

- Back-office Product and Inventory Management
- Order processing
- Payment processing
- Warehouse logistics
- Customer support

These issues have established solutions, which will be elaborated upon later.

However, the circular economy introduces additional steps to this flow, mainly product acquisition, retrieval, quality assurance, reconditioning, and the subsequent reintroduction of these items into an e-commerce platform. These are the challenges that need to be solved.

- **Product acquisition:** The process of gathering second-hand or returned items from customers. It is also necessary to filter out undesirable products and determine which items are worth refurbishing and selling. The goal is to create a streamlined

experience, and in the case of OLX 2nd Life, there will be no acquisition through returns.

- **Product retrieval:** The corresponding logistics problem of actually getting the physical product from the seller to the warehouse/refurbishing facility.
- **Quality assurance:** The act of inspecting returned items to ensure they meet certain standards of quality before they are resold. The goal is to have a system that enables an operator to quickly and accurately determine which items are in good enough condition to be resold, and a software system that fits well into this workflow.
- **Reconditioning:** This refers to the process of repairing or restoring returned items to make them suitable for resale. Different categories of products have different quality assurance and reconditioning processes. They also require diverse skill sets and vary in cost.
- **Re-injecting into an e-commerce platform:** This refers to the process of adding acquired items back into a digital store. In the case of OLX, they will have a dedicated online store for these items and an advertisement on the existing marketplace that links to it.

In summary, the potential of LoopOS lies in its capacity to merge the functionalities of e-commerce with the requirements of the circular economy. By focusing on these steps, from product acquisition to seamless re-introduction into e-commerce platforms, LoopOS aims not only to revitalize the way returns and second-hand sales are perceived but also to redefine the e-commerce landscape. Through the OLX 2nd Life pilot, this project will validate the LoopOS concept and demonstrate its usefulness.

This page is intentionally left blank.

Chapter 2

Planning

In this chapter, the author outlines the key tasks, provides a detailed breakdown of the development timeline, discusses the adapted methodology, and analyzes associated risks.

2.1 Tasks

This section describes the tasks done during this project, divided into three phases: Setup, Development, and Maintenance.

The initial planning was first done using Gantt diagrams, but as the project progressed, it became clear that this was not the best approach. Given the project's Agile development methodology (described in section 2.2), these traditional Waterfall elements proved to be not the best fit, and the aforementioned Gantt diagrams were often mismatched with the reality of the project and the client's needs.

The planning and organization of the project's tasks are discussed in more detail in section 6.1.1 of the Development chapter. In this section, the focus is on the project's timeline and the general overview of the tasks.

2.1.1 Setup Phase

Scheduled between the 14th of February and the 22nd of April (2022), the initial phase is predominantly centered on setting the project foundation and planning the development process and the dissertation itself. The tasks in this phase encompass:

Planning

This task corresponds to the estimation of the work to be done, the correspondent time costs the choice of the development methodology. It is composed of establishing a roadmap for the project with the client and defining a tentative timeline.

State of Practice

The purpose of studying existing solutions is to make better and more informed decisions about the project in question. This task includes researching topics such as existing e-

commerce systems, configurable back-office solutions, modular system architecture and design, and circular e-commerce.

Requirements elicitation

The analysis and consolidation of the system requirements. These impact the Architecture of the system and the tools and technologies to be used. The result of this task should be user stories, quality attributes, and business constraints.

Architecture Design

After the requirements elicitation task, comes the design of the system itself, through the help of different diagrams. The purpose is to design and explain, on a higher level, how the system will meet the defined requirements. These are also useful as documentation, to help explain the system to others and the client.

Security considerations

Referring to the Security Document prepared by the business team and OLX, to understand and assimilate the planning, GDPR, and other relevant requirements that impact development.

Setting up boilerplate tasks and Establishing CRUD operations

Projects are initiated leveraging the chosen frameworks and technologies, complemented by the setting up of development environments (docker containers) and Gitlab CI/CD pipelines, and the terraform scripts for the infrastructure. The CRUD operations are the basic Create, Read, Update, and Delete operations that are the foundation of any application, and are the first step in the development of the Submission Module.

2.1.2 Development Phase

From 23rd April to 22nd September (2022), the Development phase corresponds to the core period of active development.

Intermediate Report and Presentation

Given that this project is also part of a master's thesis on Software Engineering, there is an intermediate evaluation to make sure the author is on the right path. The intermediate report is followed by a presentation and discussion with the assigned juries. This requires adequate preparation so any questions and comments can be answered and clarified during the defense.

Submission Module

Here, the focus is on the back office, the Submission Module, API for the E-commerce Module, and integration with essential services like payment systems and shipping. The initial demo for OLX, scheduled for 4th July, provides a tentative deadline for the primary components.

Submission Module Frontend

Design and implementation of the frontend for the Submission Module.

E-commerce Module

This task involves the creation of the back office, the online shopping components, OLX API integrations, integration with the Submission Module, and other service integrations.

Testing

Even though a formal methodology such as Test Driven Development (TDD) or *Behavior Driven Development* (BDD) is not being followed, testing is still a crucial part of the development process, manual, unit, and integration tests are still necessary, when applicable. The goal of this task is to make sure the system is stable and reliable.

Documentation

This task focuses on creating API documentation, complemented by artifact documentation including various diagrams.

2.1.3 Maintenance Phase

Running from 22nd September to May of the following year, the Maintenance phase ensures the platform's longevity and adaptability. Post the product's launch, the maintenance phase ensures its sustained functionality and efficiency. The main functionality will be close to done, so there will be more focus on testing and improvements according to the client feedback. The tasks can be categorized into the following topics:

Development and Improvement

While the core features should be operational by this phase, there might be additional or refined features based on real-world tests and feedback. Key tasks involve refining the e-commerce module and enhancing the catalog and store page functionalities.

Code Maintenance

Addressing any shortcuts or quick fixes implemented during development to meet deadlines, to ensure long-term stability and performance.

Deplopyment Monitoring and Maintenance

This task involves monitoring the system's performance and stability and addressing any issues that arise.

Final Report and Presentation

Similarly to the one in the first semester, there is one final deliverable that describes how the whole process was realized. This task will result in the preparation of a final report, followed by a presentation and discussion.

2.2 Methodology

This section describes the methodology used for this software development project. The rationale behind the methodology is explored first, followed by a summary of its foundational principles and values. The concrete applications for the project-specific needs are discussed in section 6.1.1.

In The Loop Company, smaller teams are typical. Considering the context of the project, and given the relatively short time frame for the initial release of OLX 2nd Life, an Agile approach was deemed most fitting. This decision works especially well with the existing culture, collaboration strategies, and source control practices in the company.

The source control practices are known as git workflows or git branching strategies [2], and the specific Git workflow will be discussed later in subsection 6.2.2.

Agile software development is an umbrella term for a group of practices based on the 12 Principles defined in the Agile Manifesto. It was created in February of 2001, when trying to find common ground between existing methodologies of the time, such as SCRUM, Extreme Programming, Crystal Clear, and Feature-Driven Development (FDD), among others.

The 12 Agile principles, as seen in the Agile Alliance Website [3], can be summarized as follows:

1. Prioritize customer satisfaction through continuous delivery.
2. Embrace changing requirements, leveraging them for competitive advantage.
3. Frequently deliver working software.
4. Facilitate daily collaboration between business stakeholders and developers.
5. Cultivate motivated teams and provide the necessary support.
6. Prioritize face-to-face communication.
7. Evaluate progress primarily through working software.
8. Advocate for sustainable development pacing.
9. Commit to technical excellence and robust design.
10. Emphasize simplicity in design and execution.
11. Promote self-organizing teams for best outputs.
12. Encourage regular team reflections and continuous improvement.

And the four Agile Values:

1. Individual and interactions **over** processes and tools.
2. Working software **over** comprehensive documentation.
3. Customer collaboration **over** contract negotiation.
4. Responding to change **over** following a plan.

It is important to note that the items on the right are also important, nevertheless, the items on the left are valued more.

The main focus is on the people, a working product, and collaboration between team members [4]. Also, a great emphasis is put on self-organization and personal responsibility. Empirical data consistently shows[5, 6] that workers who feel appreciated and responsible for the project decisions and direction are more engaged and put more effort into the produced work.

The following list highlights some activities and Agile values applied in this project:

- **User Stories:** Requirements summarized into simple sentences that convey the expected system behavior and its beneficiaries.
- **Daily Meetings:** Regular meetings to discuss updates, plans, and challenges, inspired by SCRUM.
- **Incremental and Iterative Development:** Enabled by tools like *git* and *Gitlab*, emphasizing iterative progress and incorporation of feedback. Detailed in section 6.2.2. The cyclic nature of this process is depicted in figure 2.1.
- **Milestone Retrospective:** Periodic reviews of significant project milestones.

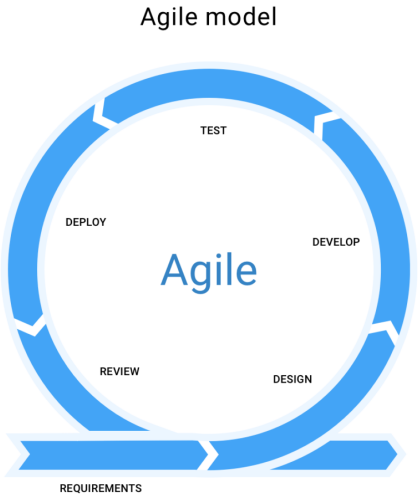

Figure 2.1: Agile development cycle, also known as a Sprint.

In conclusion, the process used was similar to SCRUM, but without some formalities, such as a dedicated Scrum Master, or rigid, explicit, and planned sprints.

This page is intentionally left blank.

Chapter 3

State of Practice

In this chapter, an analysis of the current state of practice is carried out.

As the main objective of this dissertation revolves around a new software solution for *e-commerce* within the framework of a *circular economy*, both of these concepts are explored and defined.

The *E-commerce* section consists of a brief introduction and impact of e-commerce, followed by the analysis and comparison of different software architectures used to build these systems, ending with a comparison of existing solutions.

Similarly, the *Circular Economy* section also begins with an overview. The role of software systems in enabling this type of economy is then discussed.

Finally, the existing solutions that solve similar problems are reviewed.

3.1 E-commerce

E-commerce is a familiar term to most people. It is a business model that allows companies to sell goods and services over the Internet. It was a logical consequence of the rapid evolution of information technology in the 1990s. While e-commerce has evolved into a multifaceted domain, its essence remains the digital dissemination of goods and services, encompassing the order placement and the eventual product delivery to consumers.

To highlight the magnitude and influence of this domain, the following statistics [7] are presented:

- Online retail sales amounted to 4.9 Trillion U.S dollars worldwide. It is expected to grow by over 50% until 2026.
- Most internet users (58.4%) buy online every week. The most prevalent spending categories are consumer electronics and fashion, amounting to 988.4 billion and 904.5 billion U.S. dollars.
- Revenue from physical stores is almost identical to digital sales.

Given these figures, there exists a compelling economic incentive to increase sales from online channels.

Primarily driven by young adults (ages 16-24), the appeal of e-commerce lies in the convenience, security, and comparative ease it offers to consumers [8].

With this much potential income, the e-commerce domain is highly competitive, and online platforms compete to provide the most streamlined, efficient shopping experience. Consumers appreciate and favor e-commerce characteristic features such as the ease of price comparison, access to reviews, and the convenience of purchasing from the comfort of their homes.

3.1.1 Types of e-commerce

There are several types of possible transactions in e-commerce:

- Direct to Consumer (D2C)
- Business to Consumer (B2C)
- Business to Business (B2B)
- Business to Government (B2G)
- Consumer to Consumer (C2C)
- Consumer to Business (C2B)

This thesis primarily centers on B2B, B2C, and the integrative concept of B2B2C [9]. While B2B involves businesses selling to other businesses and B2C focuses on business-to-consumer sales, B2B2C merges both by enabling businesses to access new consumer markets through partnerships. Prominent e-commerce platforms, such as Amazon and Alibaba, exemplify the B2B2C model. Of these, B2C sees the highest transaction volume, while B2B, with fewer but larger transactions, garners significant profitability [10].

Customer Journey

In the following section, the concept of customer journey will be explored to better understand the requirements for common e-commerce software.

By examining the various stages of the customer journey, including initial awareness and interest, conversion, purchase, and follow-up customer support, the specific needs of this type of software can be identified. These interactions can be categorized into five phases:

- **Awareness:** how the customer becomes aware of the product or service offered. Channels for this are social media, word of mouth, or traditional advertising.
- **Consideration:** the evaluation step. The user does research reading reviews and comparing it to similar products, to decide whether or not it meets their needs.
- **Conversion:** the act of deciding on the purchase. A marker of conversion can be adding the product to the cart and performing the checkout.
- **Service:** the post-sales customer support and relations that include returns, exchanges, and other aftercare aspects.

- **Loyalty:** the customer becomes loyal to the brand businesses and is likely to continue to use their service/make purchases again.

Although the journey depends and varies on the nature product/service and business/brand, understanding this flow remains crucial. Analyzing and refining these stages enhances the conversion rate and increases the number of customers completing the journey.

Now from a functional point of view, to enable the customer journey, an e-commerce platform needs to have the following features:

- **Catalog Management:** The platform should allow the merchant to create, modify, and organize the products. This includes the product's properties, such as price, description, and images but also how they relate to each other.
- **Order Management and Payment Processing** The platform should give the user an interface to capture the user information, such as address and payment information, to be able to process and keep track of the order. Payment processing and the handling of the user's data are critical to the functionality and trust of the whole system and as such a primary security concern. This is why many e-commerce platforms integrate with trusted third-party payment processors (such as PayPal, Apple Pay, and EasyPay) and delegate the payment and possibly invoicing burden to those.
- **Logistics and Inventory Management** The platform should have a way to manage the stock if the product being sold is limited. It should also provide information on the delivery. It is also often the case that the system needs to integrate with external warehouse management systems to automate order processing.
- **Customer Relationship Management Tools** Mechanisms to engage with customers, address queries, gather feedback, and troubleshoot issues.
- **Analytics and reporting tools** Lastly, the platform should also provide tools to gather and analyze metrics, such as sales, traffic, demand, and general customer behavior.

The following section will explore the different software architectures used to build systems that enable these features.

3.1.2 Typical Architectures

This section gives an overview of different software architectures for e-commerce applications.

The architecture of a software project is dictated by several factors such as the size and complexity of the project, the resources available, and the long-term goals of the business in question.

Each architecture has its pros and cons, like how scalable it is and how easy it is to develop. That's why it is crucial to know the trade-offs before picking an architecture.

These trade-offs can depend on several factors [11], such as the following:

- **Security:** Making sure the software system is safe from unauthorized access and that sensitive data is stored securely.

- **Cost:** The financial resources required to develop, maintain, and operate a software system, including the cost of hardware and other software.

- **Ease of development:** The degree to which a software system can be easily developed, tested, and maintained by the development team.

- **Ease of maintenance:** How simple it is to update or modify the software it has been deployed.

- **Performance and Reliability:** The efficiency and speed with which a software system performs its intended tasks, and the ability of the system to handle high levels of usage or data without experiencing delays or errors.

- **Deployment:** The process of delivering and installing a software system or application in a production environment, where it can be used by end users.

The following sections discuss software architecture in increasing levels of complexity. This is done in two dimensions: firstly from a more physical perspective, each with an increasing number of tiers and layers, and secondly from a more conceptual perspective, contrasting classical approaches like Monolithic with modern ones such as Microservices.

It is important to note that the terms *tier* and *layer*, which, though often used interchangeably, have different meanings [12].

Specifically, a *tier* operates independently of the other components and usually runs on separate infrastructure, whereas a *layer* shares the same machine with other parts of the application.

From Simplicity to Complexity

This section outlines architectures in ascending order of complexity, ranging from Two-Tier to N-Tier systems.

Two-Tier

This subsection shortly examines the two-tier architectural model, which is the simplest form of software architecture in the context of web-based applications. The architecture fundamentally consists of two principal components: the *client* and the *server*.

The client component is responsible for managing user interaction. It usually has a Graphical User Interface (GUI) and mainly works by generating and sending user requests to the server.

The server, commonly a Relational Database Management System (RDMS), assumes the role of the data repository. It is responsible for the storage, retrieval, and manipulation of data.

An example diagram is shown in figure 3.1.

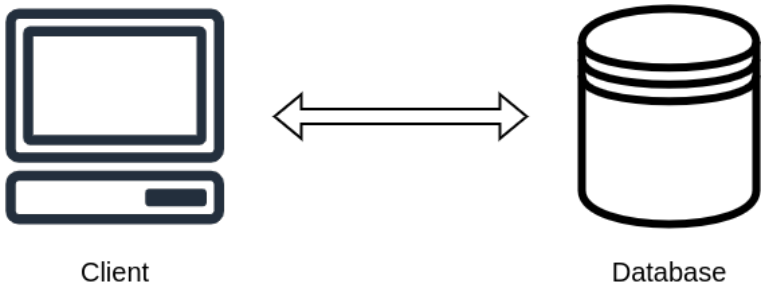

Figure 3.1: Example of a two-tier architecture.

Despite the relative simplicity and ease of implementation, this architecture has some inherent limitations.

- **Security Vulnerabilities:** The business logic is encapsulated within the client side, creating a significant security risk. Given that client-side applications are susceptible to tampering [13], functionalities such as validation, authentication, and authorization must be executed on the server. In this two-tier architecture, this would fall upon the database, which is suboptimal.
- **Scalability Concerns:** The architecture's scalability is predominantly constrained by its server component. Although resources like CPU or memory can be added to a single machine (known as *Vertical Scaling*), such enhancements have limits.

Due to these limitations, the two-tier architecture is limited in applicability for applications such as e-commerce platforms and is reserved for simpler projects requiring rudimentary data access capabilities.

Three-Tier

The Three-Tier architecture introduces a middle component known as the *Application Server*, effectively adding a layer of abstraction between the client and the data storage tier. An example is shown in figure 3.2.

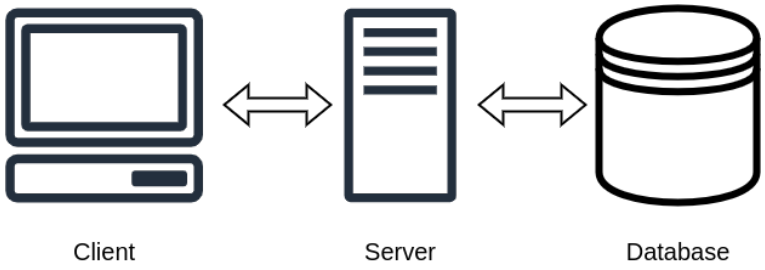

Figure 3.2: Example of a three-tier architecture.

The components of this architecture are as follows:

Client Side: Similar to the Two-Tier model, the client side is responsible for user interaction. Its purpose is to provide a user interface to generate requests to the application server.

Application Server: The application server serves multiple functions, which can be divided into three categories:

- **Business Logic:** Responsible for running validations, and performing authentication and authorization procedures, and other business-specific logic.
- **Data Access Logic:** Responsible for interacting with the data tier, executing CRUD operations (Create, Read, Update, Delete).
- **Presentation Logic:** Generation of the client-facing views.

Data Tier: Usually a storage system, the data tier is similar in function to that of the server in the two-tier architecture, however with less responsibility.

The application server itself can also have different architectures, like Event-based or Thread-based, which heavily impact performance, especially for applications with high load volumes in terms of requests per second.

Ruby on Rails and Next.js are two popular web frameworks that exemplify the differences between these two approaches. Given that both of these frameworks are used in this project (as seen in section 4.2.2), a brief comparison is presented below.

Ruby on Rails adopts a more traditional server-side rendering approach, as it is made to be operated in a multi-threaded environment. Each incoming HTTP request is handled by a new thread or a thread from a pool, which then performs all the operations—from database queries to HTML rendering—before sending the response back to the client.

On the other hand, Next.js primarily focuses on a server-side rendering approach too, but it is optimized for a single-threaded, event-loop architecture commonly found in Node.js applications. In this architecture, non-blocking I/O operations allow the system to handle a large number of simultaneous connections with minimal threads, making it highly efficient for I/O-bound applications.

Security and Scalability: The separation of concerns across multiple tiers allows for more robust security measures and the opportunity for horizontal scaling, particularly at the application server level.

In summary, the Three-Tier architecture provides a balanced trade-off between complexity, security, and scalability, making it a widely adopted model for complex web-based applications.

N-Tier architecture

The N-tier architecture can be considered an extension or generalization of the three-tier architecture, and in many real-world scenarios, applications go beyond three tiers. As the complexities grow—ranging from increased data, more business logic, or higher user traffic—it becomes necessary to break down the application into more specialized tiers to maintain performance, scalability, and manageability.

Figure 3.3[1] shows a more complex example of an architecture that includes not just the traditional client, application server, and database tiers, but also adds additional layers

[1]Source: `https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier`

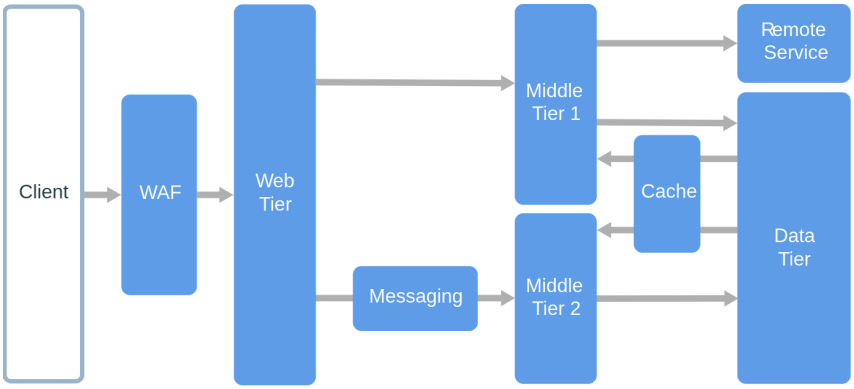

Figure 3.3: Example of an N-tier architecture.

such as a Web Application Firewall (WAF), a Message Queue Service, and a Cache layer that serves multiple middle tiers.

Each additional tier serves a specific purpose:

- **Web Application Firewall (WAF):** Protects the application from various attacks by filtering and monitoring incoming traffic between a web application and the Internet.
- **Message Queue Service:** Manages asynchronous tasks and enables application components to communicate more effectively.
- **Cache Servers:** Increases application performance by storing frequently accessed or expensive-to-generate data, so that future requests can be served faster. Redis is a popular choice for this tier, for example.
- **Background Processing:** Handles long-running tasks that are not time-sensitive, such as sending emails, generating reports, or performing data analysis. Sidekiq is an example, also used in this project.

In summary, as software projects evolve and requirements become more complex, the architecture often needs to move from a simplistic three-tier approach to a more robust and flexible N-Tier model.

Classical vs. Modern Architectures

In this section, the focus is on the conceptual differences between traditional architectures like the Monolith and more modern ones like Microservices and MACH.

Monolithic Monolith architecture involves building an application as a single, unified unit. All code for routing, business logic, and database operations are managed as one system.

These architectures are easier to develop and test, but more difficult to scale.

Microservices A microservice-based architecture is a set of interconnected services with distinct responsibilities. These communicate with each other via APIs such as REST or

Name	Type	Customizability	Cost
Shopify	SaaS	Limited due to SaaS model	Subscription
Solidus	Open Source	High, source code accessible	Development and Hosting
BigCommerce	SaaS	Moderate with API integrations	Subscription
Magento	Open Source	Very high, extensive plugins	Development and Hosting
Commercetools	Headless	High, decoupled front and back end, AI tools	Upon request

Table 3.1: Comparison of some existing e-commerce solutions

SOAP. Each microservice has its own architecture and business logic which greatly reduces coupling. The main advantages consist of making the process of modifying or replacing a part of the system less painful and easier to scale. On the other hand, these systems are also more difficult to test, require more resources to scale, are more complex to develop, and introduce operational overhead (service discovery, load balancing, etc.).

MACH architecture MACH stands for Microservices, API-first, Cloud-native, and Headless. This approach to software architecture has been gaining traction in cloud-managed solutions [14], especially in the world of e-commerce, due to its scalability, flexibility, and adaptability.

In e-commerce, this can be especially beneficial as different aspects like product catalog, user management, and payment gateway can be developed and scaled independently.

This type of architecture requires a steeper learning curve, but it is more flexible and scalable than the other two. Also, it enables the use of different technologies for each service, which can be beneficial for a larger development team.

3.1.3 Existing e-commerce solutions

There are several e-commerce solutions that businesses can utilize. In this section, they were divided into three types: SaaS (Software as a Service), Open Source, and Headless frameworks. SaaS platforms, such as Shopify, provide a subscription model. The e-commerce system is managed, maintained, and updated by the provider, making it cost-effective and efficient for rapid market entry. Open Source solutions, like Solidus, allow businesses to tailor the software according to their needs by accessing the source code but come with added responsibilities in terms of updates and security. Headless solutions act more like a service, providing the back end, and leaving it up to the buyer to build the front end.

These different types of solutions can differ in terms of customizability, cost, and ease of use. The table 3.1 shows a comparison of some of the most popular e-commerce solutions.

3.2 Circular Economy

In this section, the concept of a circular economy is explored, and the role of software systems in enabling this type of economy is discussed. Finally, some existing businesses operating in this domain are reviewed.

3.2.1 Background

The term circular economy has gained momentum both in academia and among the general population, as ecological consciousness and concern for sustainability and the environment grow. A notable example is the recent law approved by the European Union, which mandates that all new smartphones, tablets, e-readers, and portable speakers—among other small electronic devices—sold in the EU should use the USB-C type charging port by the year 2024. For laptops, the deadline is extended to 2026 [15].

Companies are also increasingly aware of the opportunities promised by the Circular Economy and have started to realize its value potential for themselves and their stakeholders [16].

The Loop Company is rooted in this concept. The term has a rather broad definition; one study even analyzed 114 definitions to refine it [1]. Generally, the main aim of the circular economy is economic prosperity, closely followed by environmental quality.

One method used to rigorously define concepts like this is the *coding framework*, a technique that converts verbal or visual data to numeric form for purposes of data analysis. An example is shown in Figure 3.4, illustrating the various strategies to which circular economy can refer.

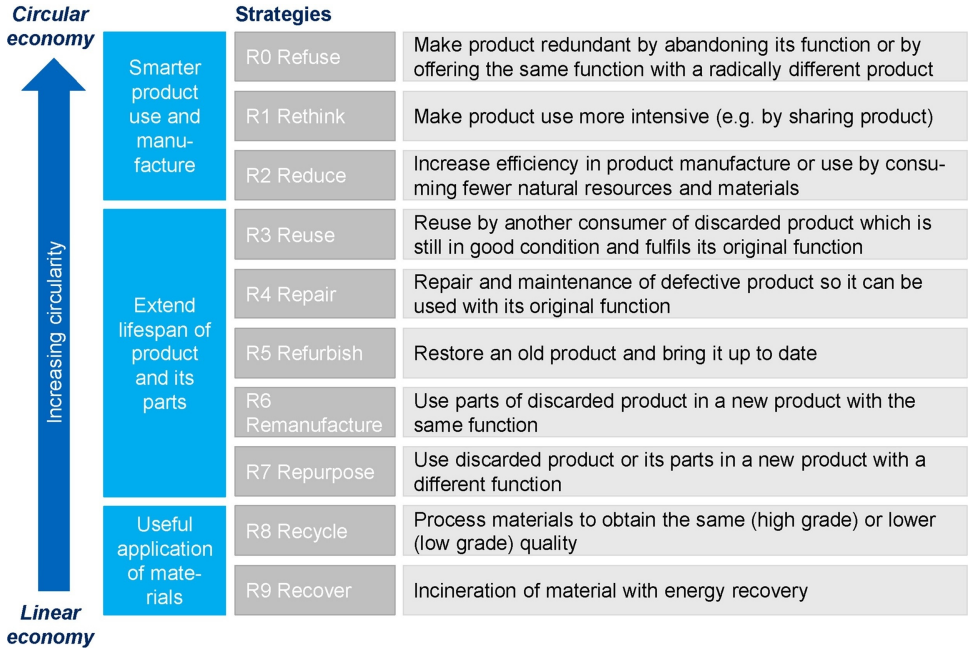

Figure 3.4: The R9-framework. Image from Potting et al. (2017, p.5)[1]

.

These strategies are also called loops [17], and the tighter the loop (the lower the R-value), the fewer external inputs are needed to close it—resulting in a more impactful circular

strategy.

The aim of this project, and that of The Loop Co. as a software company, is not to solve broader consumerism problems, which are represented in the R0 - R2 scale of the R9 framework. Such issues can only be tackled through nationwide policies and by the manufacturers themselves. The Loop Co.'s objective is to provide the tools that enable existing e-commerce platforms and companies to incorporate and close the R5 - Refurbish, R4 - Repair, and R3 - Reuse loops into their existing operations, with minimal changes to their business models.

In this context, a succinct and complete description of the circular economy [18] would be:

"The definition of circular economy is a model for product consumption to extend the life cycle of a product, through reusing, repairing, refurbishing and recycling existing materials".

3.3 Similar solutions

This section presents two existing circular economy applications, BabyLoop and CEX, and analyzes their features and business models.

BabyLoop

As mentioned before, BabyLoop operates in a niche of second-hand baby care products, specializing in their refurbishment and resale.

Their services cover a variety of Puericulture care items, including strollers, car seats, and more.

The business model is structured as an online marketplace. Sellers submit their second-hand items for initial evaluation, after which the product undergoes a quality control process. The item is then integrated into BabyLoop's online storefront, where it is made available for resale. The company generates revenue through transaction fees and, likely, a markup on the refurbished items.

The main challenges for BabyLoop involve maintaining product quality and safety standards, crucial for their targeted consumer base.

The platform uses a mix of e-commerce software solutions for inventory management, customer relationship management, and data analytics tools to understand customer behavior and market trends. All of the software was built in-house, using open-source tools, aside from external services like payment processing and logistics.

CEX

Another example is CEX[2], a company that specializes in the buying and selling of second-hand electronics and entertainment products.

The business model is similar to BabyLoop, albeit the company has a greater physical presence, and the website acts primarily as a catalog.

[2] `https://pt.webuy.com/`

Summary

One of the key distinctions that set circular economy products apart from typical second-hand items is the element of trust. In this economic model, each product must be evaluated and refurbished, aside from inverting the flow of goods, increasing the cost of operations, and decreasing the profit margin.

Another aspect of the circular economy is that no two items are identical, even if they are of the same make and model. Each item can be in a different state of usage, varying levels of wear and tear, and may necessitate different refurbishment requirements. This is especially pronounced in certain categories over others, like the Puericulture mentioned before.

Due to the aforementioned reason, a generalized approach and existing systems, often are not sufficient, because this necessitates a high degree of customization in both business operations and software solutions.

For instance, the process of reviewing, cataloging, and pricing items tends to be highly manual. Experts must evaluate the condition of each item, determine its market value based on its present state, and ensure its quality before it is listed for sale. From a software standpoint, there are currently limited systems that can automate these tasks.

In short, the circular economy business model introduces challenges that require specialized solutions. The highly individualized nature of each product demands unique approaches to product sourcing, assessment, and pricing.

Chapter 4

System Requirements

This chapter serves as an overview of the system requirements and constraints. The section begins with User Stories, in order to explain who and how they will benefit from this system. The purpose is to identify both the Architecturally significant requirements and the functional requirements.

4.1 User Stories

User stories are a powerful tool for describing requirements and demonstrating user needs and the user's points of view. It is a common, core technique of Agile development, used to identify and structure requirements.

User stories are collaborative design tools. All stakeholders are expected to participate in the definition and sorting of user stories. These are not requirements, but rather a way to facilitate the communication between the development team and the stakeholders.

The benefits of user stories are: that they are short, specific, and goal-oriented. They are simple and accessible sentences structured with the following structure:

"As a <type of user>, I want to <goal> so that <reason>".

This process answers the important questions [19]:

- Who — The role of the user or system making the action.
- What — The action executed.
- Why — The added value the user gets from the action.

Another benefit of user stories is that given their short and simple nature, they are easy to create and understand from the user's perspective. By focusing on this perspective, it avoids two common pitfalls:

1. Describing the system from a technical point of view. This can lead to a misunderstanding of the user's needs.
2. Describing a solution instead of a problem. This leads the development team to focus on the wrong problem.

When it comes to prioritization, it is again done collaboratively and periodically. This is expanded upon in the Development Process section 6.1.

The next section describes the user stories for both the Submission and E-commerce modules.

Submission Module

The Submission Module is to be used by two types of users, the *Loop Manager* and the *Seller*. The Loop Manager can have three roles inside the Submission module. These are:

- *manager*: the primary user of the Back Office. Business and logistic background. Handles the submission process, from creating and sending the purchase offer to confirming the item reception, and deciding on quality assurance.
- *financial manager*: user from the financial department. Can only view the existing submissions. Only this role can export financial reports describing the offers awaiting payment and upload proofs of payment.
- *admin* has total control of the system. This role is mainly reserved for the tech team to handle configurations, and to access incomplete features.

The Seller, the person who will apply to sell an item, interacts with the system through a specific *proposal* page. After the *manager* fills out and sends the submission draft, the seller receives an email with a unique link to fill out the form for the purchase offer.

1. Submission Back Office - Authorization and Authentication
   (a) **As a** admin or manager **I want to** login to the back office **so that** can execute my tasks.
   (b) **As a** admin or manager **I want to** create new accounts and manage permissions **so that** so I can add other staff to the platform and enable them to manage submissions.
2. Submission Frontend - Configuration
   (a) **As a** admin **I want to** customize the logo **so that** the app style is coherent.
   (b) **As a** admin **I want to** customize the text styles **so that** the app style is coherent.
   (c) **As a** admin **I want to** define text and label translations for different languages **so that** the app is accessible.
3. Submission Back Office - Configuration
   (a) **As a** manager **I want to** customize predefined questions for new submissions **so that** I can have more flexible form templates.
   (b) **As a** manager **I want to** configure the questions based on the submission type and/or category **so that** I can create context-appropriate submission forms.
   (c) **As a** manager **I want to** choose the item collection method depending on the weight or category of the item **so that** the seller experience is practical.
4. Submission Back Office - Submission/Proposal creation

(a) **As a** manager **I want to** add advert specific questions to the submission **so that** the proposal is targeted and relevant to the product.

(b) **As a** manager **I want to** choose the submission type **so that** I can configure when and how the seller will get paid.

(c) **As a** manager **I want to** choose existing or create new product properties such as *categories, brands and models* when reviewing a submission **so that** can quickly review and edit the submission.

(d) **As a** manager **I want to** leave the submission in a draft state **so that** I can finish the process later and not lose the data I already changed.

(e) **As a** manager **I want to** know the fields that need to be completed **so that** I know what information is missing before sending the proposal.

5. Submission Back Office - Submission management

(a) **As a** manager **I want to** receive a notification that a new OLX advertisement can be of interest **so that** I can contact the seller and make an offer.

(b) **As a** manager **I want to** send the submission proposal to the Seller **so that** I can start the negotiation process.

(c) **As a** manager **I want to** re-send a proposal email to the seller **so that** the seller has another chance to accept it in the case he missed it.

(d) **As a** manager **I want to** reject or accept the received item **so that** we can guarantee the quality of the products that get sent to the store (Quality Assurance).

(e) **As a** manager **I want to** view the history of a submission live cycle, including when and who changed its state **so that** I can know what happened and who to ask in the case of an abnormality.

(f) **As a** manager **I want to** search and filter submissions **so that** it is easier to search for specific submissions and create detailed reports.

(g) **As a** manager **I want to** specify the rejection reason **so that** a potential seller can know why his submission was rejected and possibly try again.

(h) **As a** manager **I want to** mark the submission as inactive **so that** the seller does not have access to the proposal link anymore.

(i) **As a** manager **I want to** re-activate a submission **so that** the seller has another chance to accept it.

(j) **As a** manager **I want to** block a submission without notification **so that** to reject a submission without notifying the seller.

(k) **As a** manager **I want to** to able to edit a submission in the Submission Module and have the changes be reflected in the E-commerce module if the item has not yet been paid **so that** to not have to jump between modules.

6. Submission Back Office - Category management

(a) **As a** manager **I want to** create, edit and delete categories **so that** the item is properly categorized and cataloged.

(b) **As a** manager **I want to** deactivate categories and brands temporarly **so that** I can prepare the catalog for new options.

(c) **As a** manager **I want to** create category specific questions **so that** it is easier to create similar proposals.

(d) **As a** manager **I want to** configure the pickup options available for each category **so that** we can offer a more practical experience for the seller for heavier items.

7. Submission Back Office - Question Management

(a) **As a** manager **I want to** create and modify different question types, such as yes/no questions, select questions, and image submission questions **so that** I can build the submission form with any question type, tailored to the specific product.

(b) **As a** manager **I want to** marks questions as mandatory, dependent on the current submission state **so that** the answers to the questions are validated for presence when needed.

8. Submission Back Office - Exports and Payments

(a) **As a** manager **I want to** export the purchase offers to XLSX format **so that** I can use the information in other reporting tools.

(b) **As a** financial manager **I want to** export financial details for the offers **so that** can fill in the TAX reports.

(c) **As a** financial manager **I want to** to be able to generate SIBS exports in XML for multiple submissions **so that** I can pay the sellers in bulk.

(d) **As a** financial manager **I want to** pay the Sellers using Revolut, individually or in bulk, **so that** I can have an alternative and practical way to pay the sellers.

(e) **As a** financial manager **I want to** see the status of the payments **so that** I know if the Seller has already been paid.

(f) **As a** financial manager **I want to** filter the submission by payment state and date **so that** I can easily find the submissions that need to be paid.

9. Proposal Form

(a) **As a** seller **I want to** see a credible email asking me to buy my product **so that** I can sell the product faster.

(b) **As a** seller **I want to** access the purchase offer page through an email link **so that** I can fill out the details for a smooth selling experience.

(c) **As a** seller **I want to** view the images and information of my advertisement, with the proposed price **so that** I immediately recognize what the item in question and if the proposed price is of interest.

(d) **As a** seller **I want to** have some pre-filled fields on the form **so that** I do not have to retype information that I already inserted in the original OLX ad.

(e) **As a** seller **I want to** have the form remember my answers if I close the webpage or lose connection **so that** I do not get frustrated and have to retype everything.

(f) **As a** seller **I want to** receive instructions on how to send the item **so that** to be able to package and deliver the item.

(g) **As a** seller **I want to** know in what state the submission is in and what action I need to take, if any **so that** I know what to expect and when I will get paid.

(h) **As a** seller **I want to** have a page where I can follow the status of my accepted submission **so that** I know what is happening and how much I will need to wait for payment.

(i) **As a** seller **I want to** to receive a confirmation email when I accept the offer **so that** I know what are the next steps I need to take, in case I don't come back to the submission page.

(j) **As a** seller **I want to** to receive a confirmation email when I decline the offer **so that** I know that I wont be contacted again.

E-Commerce Module

The E-Commerce Module has two actors, the **e-commerce manager** and the **buyer**. In this platform, the admin role and the manager have the same level of permissions, and at this point, there is no need to differentiate between them. For the sake of brevity, **e-commerce manager** will be abbreviated to just **manager** in the following user stories.

1. Store Back Office

   (a) **As a** manager **I want to** see the items available on the store **so that** I know the OLX advertisement was created and the product is ready for sale.

   (b) **As a** manager **I want to** see the list of orders and the respective checkout step **so that** I am able to know what is being bought.

   (c) **As a** manager **I want to** update the item details **so that** I can ensure that all information is accurate and up-to-date, using a rich text editor.

   (d) **As a** manager **I want to** remove items from the store **so that** I can take down listings that are no longer available but have not been sold.

   (e) **As a** manager **I want to** choose if the item is presented as a discriminated item or not in the storepage **so that** equivalent items are sold with stock management, while others are sold as unique items.

   (f) **As a** manager **I want to** have access to buyer contact information for order processing **so that** I can contact the buyer if any issues arise during order processing.

   (g) **As a** manager **I want to** be able to create returns and reintroduce the item into the store **so that** I can handle the return process.

   (h) **As a** manager **I want to** invite people to the back office **so that** I can delegate tasks.

2. Buyer Experience

   (a) **As a** buyer **I want to** be able to click on a link, from the OLX 2nd Life advertisement, that leads me to a checkout page **so that** I can buy the item I want.

   (b) **As a** buyer **I want to** be able to complete the checkout process without any kind of registration **so that** it is straightforward to complete the purchase.

   (c) **As a** buyer **I want to** know that my order was completed with success/failure **so that** I will know if I need any further action.

   (d) **As a** buyer **I want to** filter items by categories, price ranges, and brand **so that** I can tailor my browsing experience to find items that meet my specific needs.

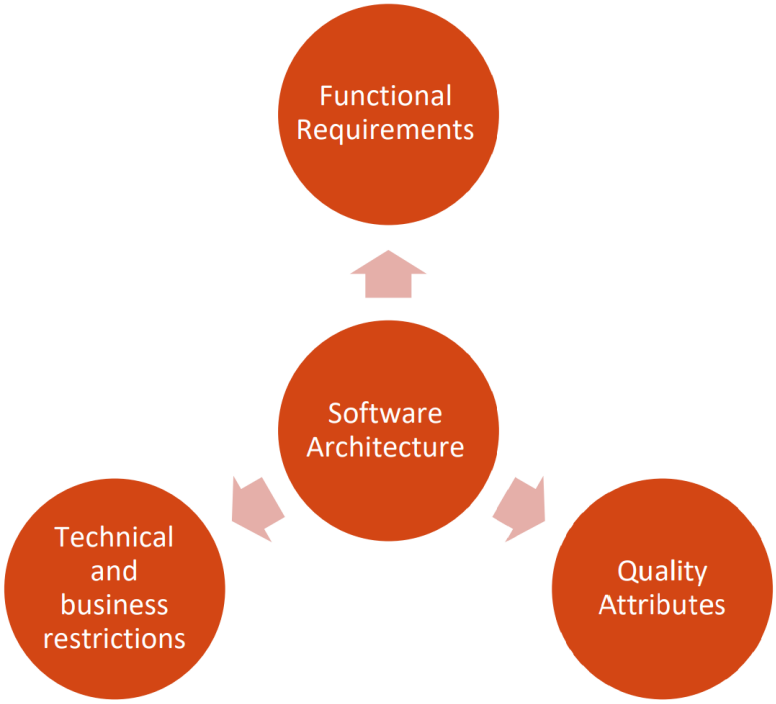

Figure 4.1: The three aspects of ASRs/Architectural Drivers

4.2 Architecturally Significant Requirements

Software architecture is a fundamental part of any large-scale software-intensive system. It significantly impacts the quality and cost of the project. The architecture is extracted from the requirements, but not all of them affect the system architecture equally. The ones that do, are called Architecturally Significant Requirements (ASRs), also known as Architectural Drivers.

Significant is the key term here. Usually, [20] it is measured *"by the high cost of change"*. This cost can be money, time, or resources like infrastructure, etc.

The incomplete or inaccurate definition of ASRs correlates with a higher chance that the produced software contains errors [20].

ASRs can be grouped into three different categories, as shown in figure 4.1, depending on the type and source of requirement.

The Functional requirements were already discussed in the User Stories section (4.1). In this section, the Quality Attributes and the Business Constraints.

4.2.1 Quality Attributes

Quality Attributes are nonfunctional requirements. These are mensurable and testable system properties used to evaluate if the system meets the needs of its stakeholders. They will be described with the use of Scenarios.

Interoperability and Modifiability

Interoperability is the software's ability to work and communicate with other products or systems. LoopOS aims to be modular and has several independently deployable parts. Communication between those two is relatively easy because all of them are developed internally, yet parts of the system are not: for example, payment processing, invoicing, and ad creation (integration with the OLX) are external systems.

Modifiability measures the cost of change. For LoopOS, this is of utmost importance, because even though the pilot project will only work with the OLX API, it must be flexible enough to enable other implementations without many modifications. Also, aside from being modifiable to be used for other projects, it must be customizable to fit the theme and look of the current project.

Scenario: In the future, the client wants the system to output the items into another e-commerce platform not yet known.

Source of Stimulus	The Loop, Client
Stimulus	Desire to output items to an alternative e-commerce platform
Environment	Under normal operations
Artifact	Submission Module
Response	The item is successfully created in the alternative e-commerce system, and the status is updated in the Submission Module
Response Measure	Submission state the Submission Module BO is updated to "In External Store," or an error message is generated specifying the point of failure

Table 4.1: Quality Attribute - System Compatibility

Given this scenario, the architecture should aim for flexibility by, for example, implementing the business logic in a client-agnostic manner. This could be achieved for example by defining an interface for the e-commerce platform and then implementing it for each platform. This would enable interaction with multiple e-commerce platforms, essentially future-proofing the system. To validate this architectural choice, a dummy REST client could be used for testing, ensuring that the implementation works independently of the specific client. The same approach could be used for payment processing, invoicing, and email services.

Another interoperability scenario, albeit less likely, is presented below:

Scenario: The client wants to use a relational database storage.

Source of Stimulus	Client
Stimulus	Requirement to use relational database storage
Environment	During system design and development
Artifact	Database System
Response	Use ActiveRecord queries and Rails ORM to interact with the database
Response Measure	No direct SQL queries are used, and the system allows for database operations via Rails ORM

Table 4.2: Quality Attribute - Database Interoperability

Being a pilot project, the system can be used (in a separate deployment) as a template for other projects or demos in the future. Given this, the system should be easily customizable to fit the theme and look of the current project.

Scenario: The Submission Proposal Page needs to change the application style and feel for another project for a demonstration of the concept.

Source of Stimulus	Project Owner
Stimulus	Need to rebrand or modify visual elements of the LoopOS Submission Module.
Environment	System Configuration Interface
Artifact	LoopOS Submission Proposal Page
Response	The system architecture allows easy modification of visual elements like color schemes, icons, and other brand-related settings.
Response Measure	An admin user can successfully update all visual elements within one hour without requiring code changes.

Table 4.3: Quality Attribute - Modifiability (Branding Adaptability)

Usability

Usability is the measurement of how well the user can complete a task to reach a certain objective. Usability was a big concern during the design phase. There are several metrics for measurement of usability, for example, the success rate of completing new tasks (Learnability), time taken (Efficiency), error rate, and user satisfaction [21].

An interesting idea in Web Design is the "Three-Click Rule":

"It's widely agreed, even by people who are not idiots, that web users are driven by a desire for fast gratification. If they can't find what they're looking for within three clicks, they might move on to somebody else's site. Hence the so-called "Three-Click Rule," which, as you might expect, states that users should ideally be able to reach their intended destination within three mouse clicks." - excerpt form "Taking Your Talent to the Web A Guide for the Transitioning Designer" [22]

Next are two scenarios, one for the Submission Module Backoffice and one for the Submission Module Proposal Page.

Scenario: Loop Manager trainee uses the Submission Module without much prior experience.

Source of Stimulus	End-user, Loop Manager trainee
Stimulus	New submissions need to be created; Additional information needs to be added to the submission can advance to the next stage.
Environment	Under normal operations
Artifact	System - Submission Module Back office
Response	For new submissions: - Learnability: The system shows which fields are necessary to send the proposal to the seller. - Error impact minimization: does not let the user send an incomplete submission, notifies what is needed to advance to the next step. - Comfortable with working at his own pace: allows the user to save the submission, even if incomplete, in a Draft state. For advancing to the next state: - Efficiency: Bulk actions e.g.: sending submission en masse.
Response Measure	Creating and sending a single submission should be doable in less than three minutes after the first interaction. Should be able to create a Draft submission in less than three-page navigation clicks (excluding clicking on the form inputs). System highlights the missing fields clearly. The user should be able to discover how to create the submissions and how to advance them to the next state via exploration. (Gain of user knowledge)

Table 4.4: Quality Attribute - Usability for Loop Manager

Scenario: Seller receives a link for the offer proposal by email. Inside, there is a link that will redirect to the form page.

Source of Stimulus	New OLX Seller
Stimulus	LoopOS wants to buy the product from the OLX ad, and the Seller wants to sell the item swiftly.
Environment	Under normal operations
Artifact	System - Submission Module Proposal Page
Response	The system provides a step-by-step visual guide for completing the form and clarifying expected actions.
Response Measure	The user completes and submits the form within 3 minutes of first-time use, without requiring external guidance. Form submission success rate is at least 95%.

Table 4.5: Quality Attribute - Usability for Seller

Performance and Scalability

Due to the unknown number of users that are to be expected once the Submission Form is released, the system must be able to handle the submission process promptly and if

necessary scale up to handle the increased load.

Source of Stimulus	End-user
Stimulus	Makes a submission in the Proposal Page.
Environment	Under normal operations
Artifact	Submission Module
Response	The API processes and lists the submission in the Back-office.
Response Measure	The submission is processed within 2 seconds 99% of the time.

Table 4.6: Quality Attribute - Performance

Source of Stimulus	System Monitor
Stimulus	An increase in submission traffic.
Environment	High-load scenario
Artifact	Submission Module
Response	The system dynamically allocates additional resources to handle the increased load.
Response Measure	The system scales up to handle 2X the normal load within 5 minutes without performance degradation.

Table 4.7: Quality Attribute - Scalability

4.2.2 Technical and Business Constraints

This section explores technical constraints presented by The Loop. These are different from the functional requirements and Quality Attributes in the sense that there is no control - the decision has already been made, and the Architecture and Design will have to abide. Constraints will be identified with TC for Technical Constraints and BC for Business Constraints.

ID	TC01
Title	Ruby on Rails
Source	The Loop
Object	Back-end Development
Environment	Software Development
Objective	Leverage known technology and developer expertise
Description	The Loop uses the RoR framework for most of its projects and as such most of its developers are already familiar. Additionally, the company already has boilerplate projects that integrate with Ruby-based services, of which some are paid, so it is also a monetary incentive to stick with a known and tested technological stack.

Table 4.8: Technical Constraint TC01

ID	TC02
Title	NEXT.JS
Source	The Loop
Object	Front-end Development
Environment	Software Development
Objective	Capitalize on modern front-end development practices
Description	The Loop has often used NEXT.JS in its front-end development projects. As many of its developers are skilled in using this framework, and there are existing templates and components, it becomes economically sensible to continue using this framework for new projects.

Table 4.9: Technical Constraint TC02

ID	TC03
Title	Solidus E-commerce framework and EasyPay
Source	The Loop
Object	E-commerce module, Back-End Development
Environment	Software Development
Objective	Reuse existing expertise and code base. Save on development time.
Description	Solidus is a Rails E-Commerce platform, and EasyPay is a payment service. The Loop has worked with both before and with EasyPay, already has a subscription. Being a paid service, moreover already tested, it is decided this will be used for the E-commerce module.

Table 4.10: Technical Constraint TC03

ID	BC02
Title	User Data Censorship
Source	Legal and Privacy Guidelines
Object	Proposal Page
Environment	Runtime, production
Objective	To comply with privacy regulations and protect user confidentiality
Description	User data must always be censored in the pages following the submission of the proposal form. This is to ensure that sensitive information is not inadvertently exposed, maintaining user privacy and meeting legal requirements.

Table 4.11: Business Constraint BC02

Chapter 5

System Architecture

This chapter presents an overview of the system architecture, including the design decisions that shaped the system. Each section corresponds to a different artifact generated during the development process. The level of detail varies across sections and artifacts, reflecting their different purposes and target audiences.

5.1 C4 Architecture

The C4 model [1] was the primary tool for creating the system architecture.

This graphical notation technique employs a standard box and arrow notation to describe the system at different scopes, each with increasing levels of detail.

The C4 model introduces several conceptual abstractions: a **person** represents a human user of a software system (also known as an *actor, persona*). A **software system** is a high-level abstraction that delivers value to its users. A **container** represents an application or data store and is a context in which code executes or data is stored. Containers are separately deployable units. A **component** is a grouping of related functionality encapsulated behind a well-defined interface, however, it is not separately deployable.

The C4 model recommends four levels of detail, each providing a different perspective. Following is a description of these levels:

- **Level 1, Context:** This is the highest level of scope, focusing on the actors and external systems that interact with the main system. It is intended for both technical and non-technical audiences.
- **Level 2, Containers:** This level breaks down the system into smaller, interconnected containers. Each container represents an application or a data store and is usually a separate deployable or runnable unit. The architecture's overall shape is formed by these containers. This level is intended for technical audiences, such as software architects, developers, and operations personnel.
- **Level 3, Components:** This level focuses on the concrete interface implementations, technology choices, and responsibilities within the system. The primary intended audience is software architects and developers.

[1] `https://c4model.com/`

- **Level 4, Code:** This is the lowest level of scope to illustrate how a component is implemented. This level is often not recommended [23] as it can be restrictive during the development phase and can often be generated automatically from the code. The primary intended audience is developers.

For this project, given its scope and the nature of the chosen development process, only the Context and Container diagrams were used.

It is important to note that in C4 models, the term *container* does not necessarily represent or has a direct correlation to the OS-level virtualization containers [24], such as Docker [25], which were also used in this project during development and deployment, as described in section 6.2.2.

5.1.1 C4 Context Diagram

The C4 Context diagram (figure 5.1) is presented in this subsection. It comprises three actors and several software systems, some of which are external to the OLX project.

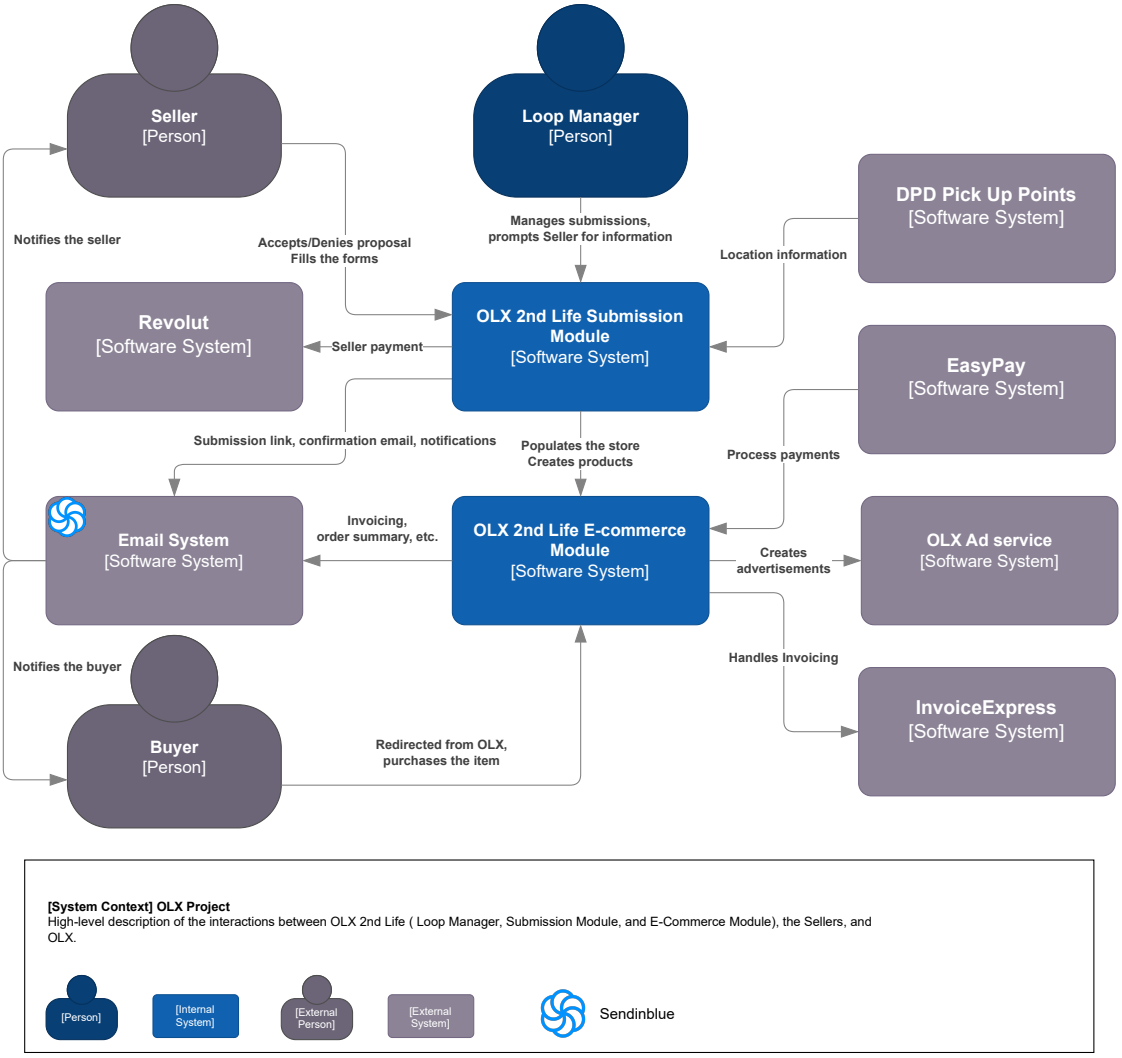

Figure 5.1: C4 Context Diagram

The actors are described below:

- **Seller**: The Seller is the individual selling the item. They interact with the Submission Module by creating and then completing the submission.

- **Loop Manager**: The Loop Managers are responsible for the acceptance, validation, and approval of submissions. They interact with the Submission Module by reviewing and approving submissions, adding necessary information, and managing the submissions' lifecycle. They also interact with the E-commerce module, managing the listings and orders, and handling any incidents that may occur.

- **Buyer**: The Buyers are the individuals purchasing the item from the E-commerce module's store. They interact with the module by browsing the catalog and placing orders.

The two software systems developed for OLX 2nd Life are:

- **Submission Module**: This is the main software system. It stores and manages all the submissions, generates the catalog, and interacts with external systems for payment (for the seller), and email notifications. It also provides the seller access to the Proposal Form.

- **E-commerce module**: This is the module that manages product listings, orders, stock, invoicing, payment processing, and returns.

The external systems are:

- **Brevo (formerly known as SendInBlue)**[2]: Brevo is the email notification system used by OLX 2nd Life. It is responsible for sending emails to both the Sellers and the Buyers. For the former, it is used to send submission approval, rejection, and additional information emails. For the Buyers, it is used to send order confirmation emails.

- **DPD Pick Up Points**: DPD Pick Up Points is an API used to get the currently available pickup points for equipment delivery (from the Seller to the warehouse).

- **Revolut**[3]: Revolut is a financial platform to pay the sellers after the item has been received and approved.

- **Easypay**[4]: Easypay is a payment gateway. It is responsible for processing payment transactions in the E-commerce module.

- **InvoiceExpress**[5]: InvoiceExpress is the invoicing system. It generates the invoices for the Buyers in the E-commerce module.

- **OLX Ad service**[6]: The OLX API that is used to create listings in the OLX Marketplace.

[2] `https://www.brevo.com/`
[3] `https://www.revolut.com`
[4] `https://www.easypay.pt`
[5] `https://www.invoiceexpress.com`
[6] `https://developer.olx.pt/pt`

5.1.2 C4 Container Diagram

The C4 Container diagram is presented in this subsection. The diagram illustrates the two primary containers of the Submission Module and their respective technologies, as well as the E-commerce Module.

As seen in figure 5.2, the Submission Module is divided into two containers: the Back office web application, also referred to as the *backend* application, developed using Ruby on Rails[7], and the Proposal Form web application (*frontend* application) built with NEXT.JS[8].

The module relies on a PostgreSQL[9] database hosted on Digital Ocean[10] for data storage.

The E-commerce Module is composed of a single container developed in Ruby on Rails. At a component level, it is subdivided by responsibility into three sections: the back office, the store, and the API endpoint. This separation is achieved by using namespaces in the code. This module also uses its own PostgreSQL database for data management.

The Submission module *backend* application communicates with the external services using their respective REST APIs. It also communicates with the E-commerce module using its API endpoint, which is a REST API as well. This communication is unidirectional, to create and populate the catalog of items. Once the item leaves the Submission module, it is no longer managed by it, but updates to the item are still possible through the E-commerce module, or updates from the Submission module if the item has not yet left the store.

The E-commerce module also communicates with external services using REST APIs, to create advertisements in the OLX Marketplace, process payments, and generate invoices.

Both databases are Relational Databases and are hosted on Digital Ocean (in the production environment, as described in section 6.3.1). These databases are separate because they have different purposes, and even though they may contain related data, one is relative to the submission process and the seller, and the other is relative to the e-commerce process, products, and the buyer. The only replicated data are the item's properties and the associated taxonomy trees, although with some differences in the data structure.

Both backends of the Submission and E-commerce Modules use the MVC (Model-View-Controller) architecture, where the model is the data layer, the view is the presentation layer, and the controller is the business logic layer.

The Proposal Form, also referred to as the Form Frontend, is a Single Page Application (SPA) that uses the NEXT.JS[11] framework. It consists of various components that handle different aspects of the dynamic form, submission status updates, and the overall seller user experience.

5.1.3 C4 Component Diagram

This subsection provides a simplified version of the Component view of the Submission Module (figure 5.3). The E-commerce Module follows a similar structure.

There are four types of components detailed in the next sections: the **Views**, the **Con-**

[7] `https://rubyonrails.org/`
[8] `https://nextjs.org/`
[9] `https://www.postgresql.org/`
[10] `https://www.digitalocean.com/`
[11] `https://nextjs.org/`

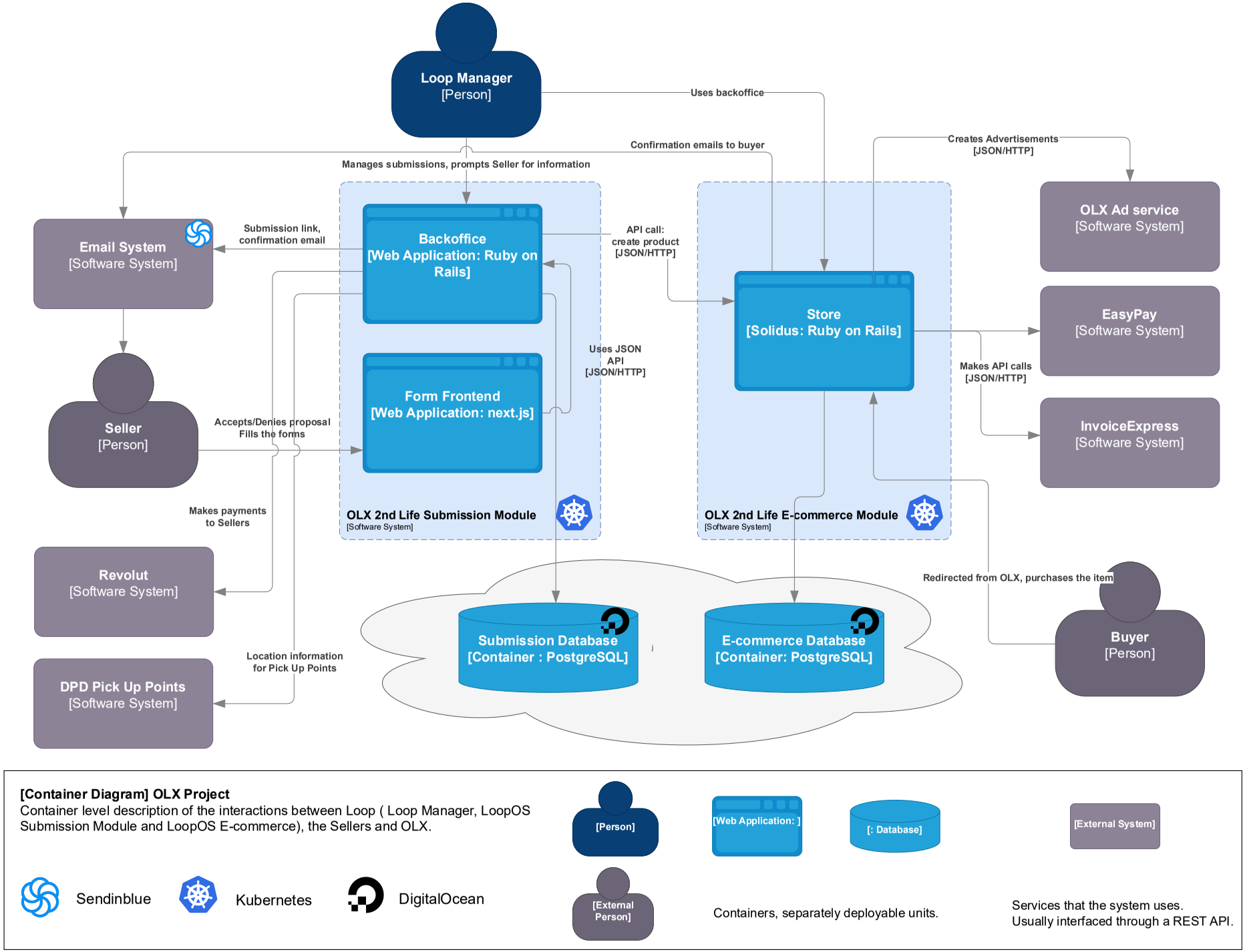

Figure 5.2: C4 Container Diagram

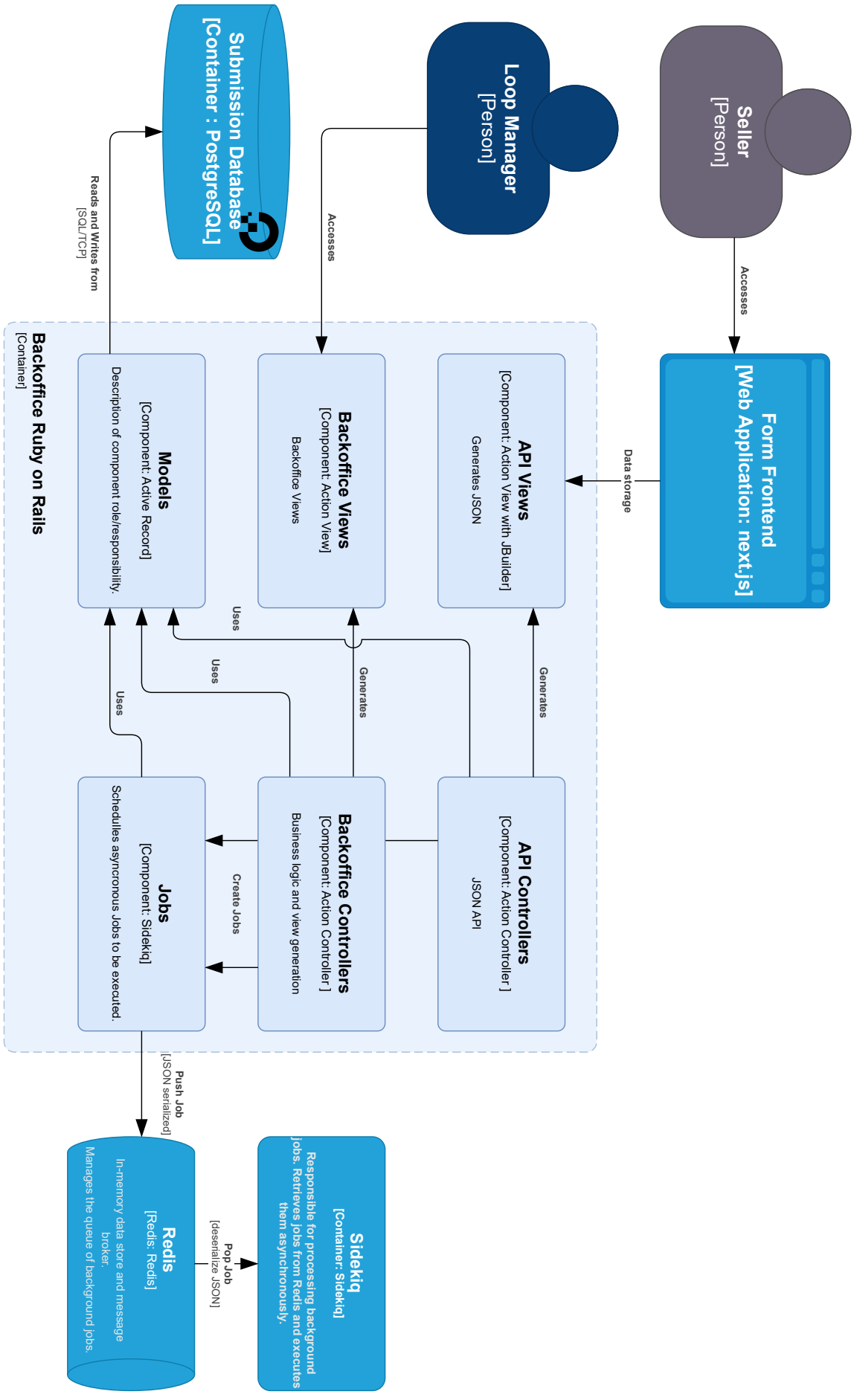

Figure 5.3: C4 Component Diagram

trollers, the **Models**, and the **Jobs**.

Controllers, Views

Both the Views and the Controllers are grouped by namespace, these being the API and the Admin/Backoffice namespaces.

Aside from these, the Redis, Sidekiq, and PostgreSQL containers are also included in the diagram.

Controllers handle the business logic of the application, while Views are responsible for the presentation layer.

In the Submission Module, the Backoffice components are responsible for managing the submissions, including reviewing, approving, and controlling the lifecycle of the submissions. The API components are responsible for presenting and receiving data, in a JSON format, from the Form Frontend application.

Most views use Hotwire[12], an open-source tool from the creators of Ruby on Rails. It allows the transmission and replacement of HTML directly in the HTML DOM without the need for a page refresh or custom Javascript, giving the Backoffice interface a Single Page application feel

Similarly to the Submission Module, in the E-Commerce Module, the Controllers and Views are also organized into namespaces to group related functionality and promote a separation of concerns.

The namespaces are as follows: the Admin, the Store, and the API. The Admin namespace has Controllers and Views responsible for managing the listings and orders, as well as handling any incidents that may occur. The Store is responsible for managing the buyer-facing store: product listings, orders, stock, invoicing, payment processing, and returns. Lastly, the API namespace is used for receiving Create-Read-Update-Delete (CRUD) requests on the products and taxonomies.

Models and Jobs

The Models are responsible for the data layer, and the jobs are responsible for the background tasks.

Rails provides the Active Record, an ORM (Object-Relational Mapping) model. Active Record encapsulates the functionality related to database access, providing a well-defined interface for these operations. Each model in the system, which is a class that inherits from Active Record, represents the various entities in the system, such as submissions or orders, and provides methods for creating, reading, updating, and deleting records in the corresponding database tables. Aside from these operations, this is also the place where model-specific methods are defined.

Active Jobs is a framework provided by Rails for declaring jobs that can be run in the background.

These jobs are used for tasks that are done outside of the immediate request-response cycle, encapsulating specific pieces of functionality and providing a well-defined interface

[12] `https://hotwired.dev/`

for invoking these tasks.

In the context of the Submission Module and the E-commerce Module, Active Jobs are used for a variety of tasks. For example, in the Submission Module, Active Jobs are used to send email notifications to the seller, generate exports, and process the payments. Similarly, in the E-commerce Module, Active Jobs are used to generate invoices and send confirmation emails to the buyer.

The Jobs themselves, while scheduled in the Submission/E-commerce modules, are executed by Sidekiq, a background processing framework for Ruby.

The job is first created in the appropriate Controller/Model or Service Objects. This is typically done by calling `perform_async` on a Sidekiq worker class and passing any necessary arguments. It is also possible to schedule a job to run at a specific time in the future, to retry a job if it fails, or to run a job periodically.

The job object is then serialized and pushed into one of many possible queues. The serialized job includes the name of the worker class, the arguments, and various metadata such as when the job was enqueued.

This queue is stored in the Redis container, an in-memory data structure store. Redis also acts as a cache, improving the performance of the system by storing the results of database queries.

Finally, Sidekiq polls the jobs pushed to Redis. When it finds a job, it removes the job from the queue, deserializes it back into a Ruby object, and executes it.

If the job raises an exception, Sidekiq places it in a separate *retry* queue in Redis. It will later attempt to reprocess the job at increasing intervals. After a certain number of failures, the job will be moved to the *dead* queue.

In this process, Redis acts as a durable and crash-resistant data store. Even if the main application crashes or is restarted, the jobs in Redis will persist.

5.2 State Machine Diagram

The submission process is a critical component of the OLX workflow, where both the Seller and the Loop Manager provide information and documents. Submissions model the business flow of the item that enters the OLX platform.

To ensure the efficient and effective management of these submissions, it is essential to have a clear understanding of the submission process, including the different stages and steps involved.

It needs to relate to the Seller's information, to the Product information, to all the form questions (including price, valuations, and refurbish costs), and also the shipping documents and transactional emails that need to be sent.

Aside from the data, the submission also models the flow itself, meaning that it has to implement the business logic, such as validation and rules to advance to the next step.

There was a need to model this process in software, and naturally, a state machine diagram is appropriate.

State machine diagrams are useful in computer architecture because they provide a visual representation of the desired behavior of a system or program. The different states of the

system or component are represented as nodes or circles, and the transitions between the states are represented as directed arcs or arrows. From the point of view of the user, the transitions represent the actions or events that cause the application state to change. It allows the modeling of the behavior of a system or program as a set of states, transitions between those states, and actions that occur when the system or program moves from one state to another.

One of the reasons why a state machine diagram was used, was due to the inherent benefit of facilitating communication between technical and non-technical stakeholders at design time. Furthermore, it also allowed the modeling and visualization of the submission process concisely.

The diagram changed from the initial design, to accommodate new requirements that arose during the development process.

The first iterations of the state machine were drawn by hand and/or drawn in [13], and the final one was generated from the code itself, using a Rails gem [14] that wraps Graphviz[15]. The diagram can be seen in figure 5.4.

5.3 Sequence Diagram

This section depicts the interaction between the Submission Module, the e-commerce module, the buyer, the seller, and OLX through a series of sequence diagrams. It models the same flow as the state machine diagram in section 5.2 but with an emphasis on the sequence of interactions aspect of the submission.

These diagrams are useful to model the behavior of a system by showing the order of messages exchanged between different actors and components.

In this case, for the submission flow, there are two human actors, the LoopOS Manager and the Seller. The boxed components represent the modules or external services, in this case, the Proposal Frontend and the Submission Module.

The Email System is omitted from the diagram as it exhibits a unidirectional nature and is not inherently relevant as a component within the architecture as it rather functions as a service that informs the involved actors about any updates or changes.

The process can begin in two ways:

- A LoopOS manager can create a submission in a *draft* state, that then will be sent to the Seller as a Proposal. This was the original flow, requested by OLX 2nd Life.
- A Seller finds out about the OLX 2nd Life project from the OLX website and submits an item for evaluation himself, resulting in a proposal in the *new* state. This is the new flow, added to the system after the initial design due to changes in the requirements.

In the first scenario, the LoopOS manager will need to perform an *Internal Review* step where they may choose to *enqueue* the submission for later, set it to *pending* because of lacking information, *reject* it right away or to accept it and move it to the *draft* state.

[13] `https://app.diagrams.net/`
[14] `https://github.com/state-machines/state_machines-graphviz`
[15] `https://graphviz.org/gallery/`

Figure 5.4: State machine diagram depicting the possible Submission states and flow.

From there, all the submissions are managed similarly.

The sequence diagram 5.5 models the initial steps of the second starting point mentioned above, beginning with the user (Seller) submitting a potential item for evaluation.

Once the form is filled, it is sent back to the Seller as a Proposal, through an email containing the link that gives the user access to the follow-up form.

This link leads to the Proposal page, where the user needs to first accept or reject the offer, and if accepted, provide all the additional information that the form requests.

After the form is submitted, the Seller receives a confirmation email, and the previous Proposal page becomes an informational page where the Seller can check the state at which his submission is. The page reflects the current state of the submission.

Finally, the Loop Manager can review the updated submission and send it to the next state, at which point the shipping guide is emitted and the Seller is notified and instructed on how to ship the item.

Each update on the submission model runs the validations required for the target state. The validations consist of a set of rules that must be met for the submission to be able to advance to the next state, and if unmet, the submission should remain in its current state, throw an error, and display the error to the user.

Figure 5.6 illustrates the second part of the submission flow, up to the emission of the shipping guide. All the Seller-provided input is done through the proposal form, which adapts to the different stages of the submission process. The Loop Manager can always update or correct the information.

In this state, the seller can Accept or Reject the offer, and if accepted, provide all the additional information, such as pick-up address and payment information. At any point, given that the proposal has not yet expired, the seller can accept an already rejected submission and continue the process. Now the Loop Manager can review the updated submission and send it to the next state, at which point it is possible to emit the shipping guide, and the Seller is notified and instructed, both via e-mail and an updated proposal page, on how to ship the item.

The email service is responsible for sending all the communication emails and can be configured to send different templates at specific transitions.

The process continues similarly, with the Loop Manager reviewing and completing the submission at each step if necessary, while the Seller is notified and instructed on how to proceed.

5.4 Entity-relationship Diagrams

This section presents the relational database diagram for the Submission and E-commerce Modules, using Entity-relationship diagrams.

Figure 5.7 shows some examples of the arrows used in the diagrams in this chapter. The circles represent the constraints - an empty circle means that the association is not mandatory, while the black circle means it is. The arrows represent the cardinality between the entities - the arrowhead points to the entity that has the *foreign key*. The many-to-many relationships are represented by two arrows, one in each direction, omitting the junction table.

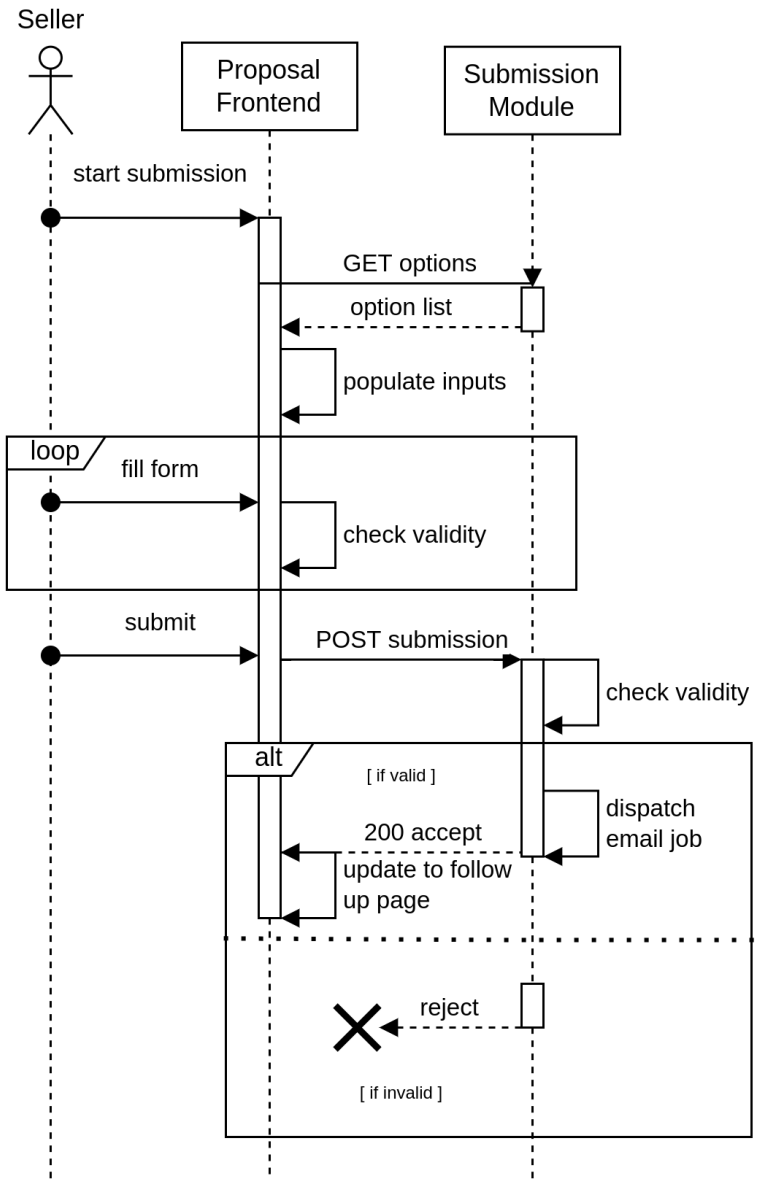

Figure 5.5: Sequence Diagram - Initial Proposal Flow.

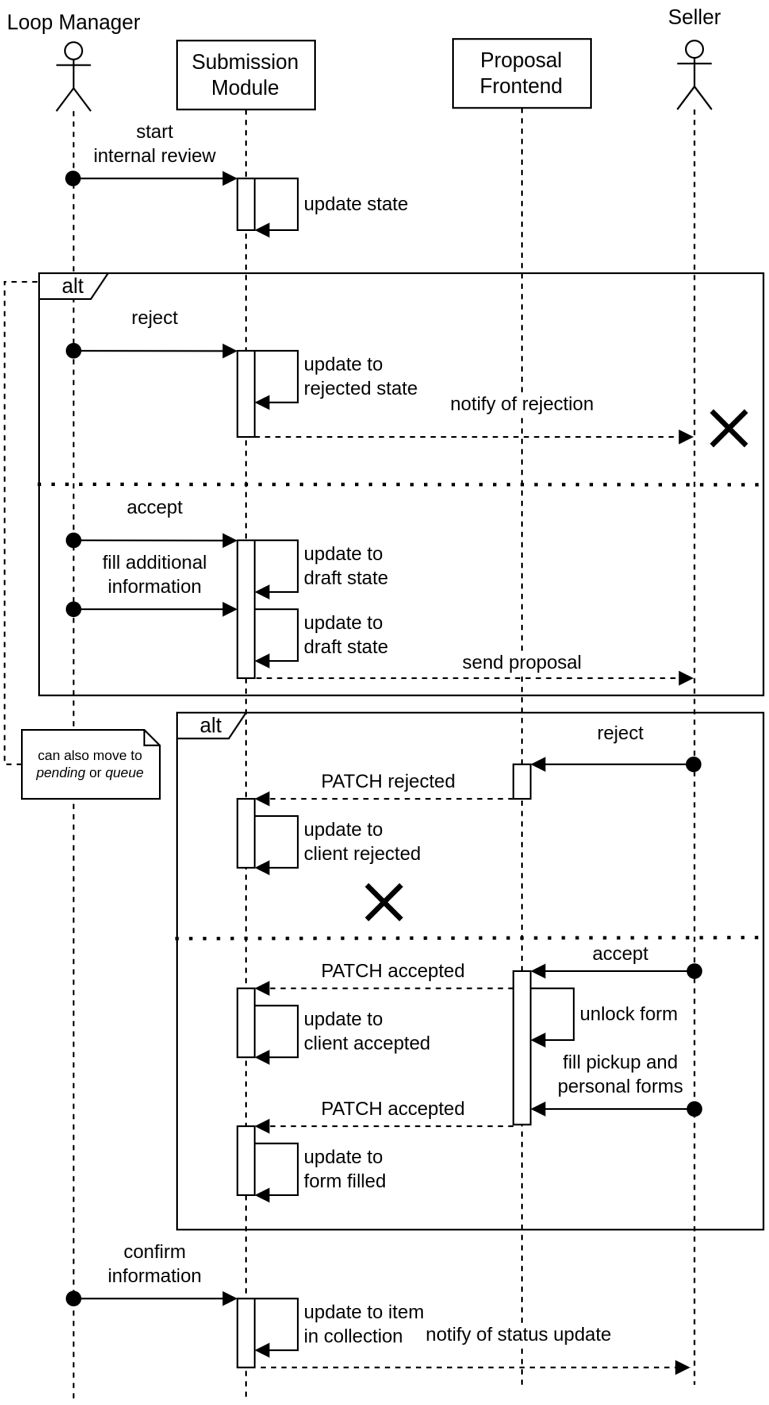

Figure 5.6: Sequence Diagram - Proposal Flow, Acceptance and Evaluation.

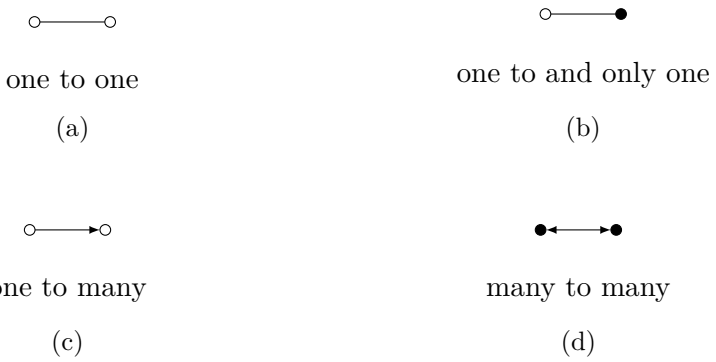

Figure 5.7: Examples of ER diagram arrows.

The following sections briefly describe the models and their attributes.

5.4.1 Submission Module

The ER diagram for the Submission Module is shown in figure 5.8 - simplified, without the column attributives, and figure 5.9 - with them.

The Submissions table is the central model of the Submission Module, containing the information about the items submitted by the Sellers. Each submission has a unique identifier (token, UUID) and includes information about the product's brand, model, and associated URLs (e-commerce, invoice, original ad, and premium ad).

This unique token is the one used to identify the submission and is used in the URL of the submission page. One advantage of using a UUID is its inherent randomness, making it extremely difficult, and practically infeasible, to guess. This way it is possible to send the link to the Seller, assuming a secure channel (email), without the need for any authentication.

Additionally, this model stores details about the submission's status, such as the current state, rejection reasons, and any relevant notes and observations that may be added during the review process.

It also stores key dates, such as pick-up and expiration, location, and number of boxes necessary to ship the item.

Another category of fields stored in this model are payment-related fields, such as payment type, payment method, and the associated payment reference.

Finally, it also contains the timestamps for creation, and updates are automatically managed by Rails ActiveRecord, as do all the models.

The Submission's model associations can be grouped as follows:

- **Product Information:** Unique identifiers (id and token), brand, model, and associated URLs (e-commerce, invoice, original ad, and premium ad).
- **Submission Status:** Collection state, submission state, rejection reason, rejection message, and review notes.
- **Observations:** Client observations and notes.

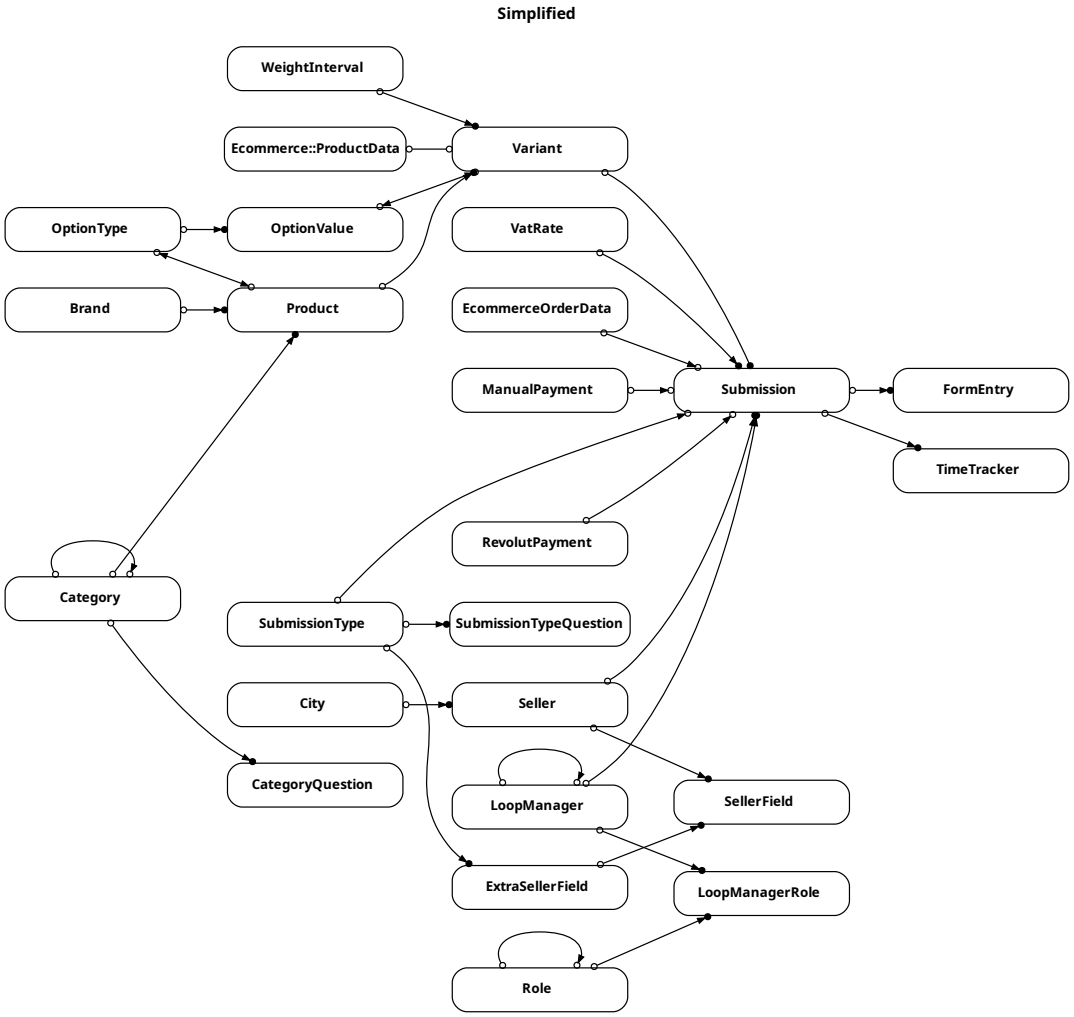

Figure 5.8: Simplified ER.

- **Logistics:** Number of boxes, pick-up date, and picking location.
- **Payment Details:** Payment type, seller payment method, and related IDs (payment, VAT of purchase, and VAT rate for this type of submission).

In ER of the Submission Module mirrors some of the common e-commerce platform models. A brief explanation of the relevant models is provided below:

- **Product:** This is the basic unit of sale and refers to the item that will be listed in the catalog. It serves as a template for creating various versions or "Variants" of that item. An example Product is a "Smartphone" or a "PlayStation 5".
- **Variant:** A Variant is a specific combination of Option Values associated with a 'Product.' For example, for a smartphone, a Variant could be a specific model with 128GB storage and a silver color. Due to the nature of the system, each variant is unique.
- **Option Types:** These define the kinds of customization or variations that a Product can have. For instance, color or storage space.
- **Option Values:** These are the specific instances of an Option Type. As per the Option Type example, the color Option type could have Option Values 'Red,' 'Blue,' and 'Green'.

The Variants model attributes can be grouped as follows:

- **Identification:** Unique identifier (id), and name.
- **Product Details:** Buy date, product value, and associated product and ecommerce_product_data IDs.
- **Pricing Information:** Loop cost, public sale price, public sale price competition (acquisition and sale), and original advertisement public sale price. These properties are used to calculate the profit margin and to generate financial reports.
- **Logistics:** Shipping cost and associated weight_interval.

The LoopManagers, LoopManagerRoles, and Roles models are used together to manage user authentication, authorization, and role assignment within the system:

- **LoopManagers:** Represents users with access to the system, storing usual attributes such as email and encrypted password. It also keeps track of invitation-related information, such as limits, tokens, and counts.
- **Roles:** Represents the various roles that can be assigned to loop managers, including a and parent role (parent_id) if any.
- **LoopManagerRoles:** Serves as a bridge between LoopManagers and Roles, associating a loop manager with one or more roles. This is an intermediate model that is not directly used in the application.

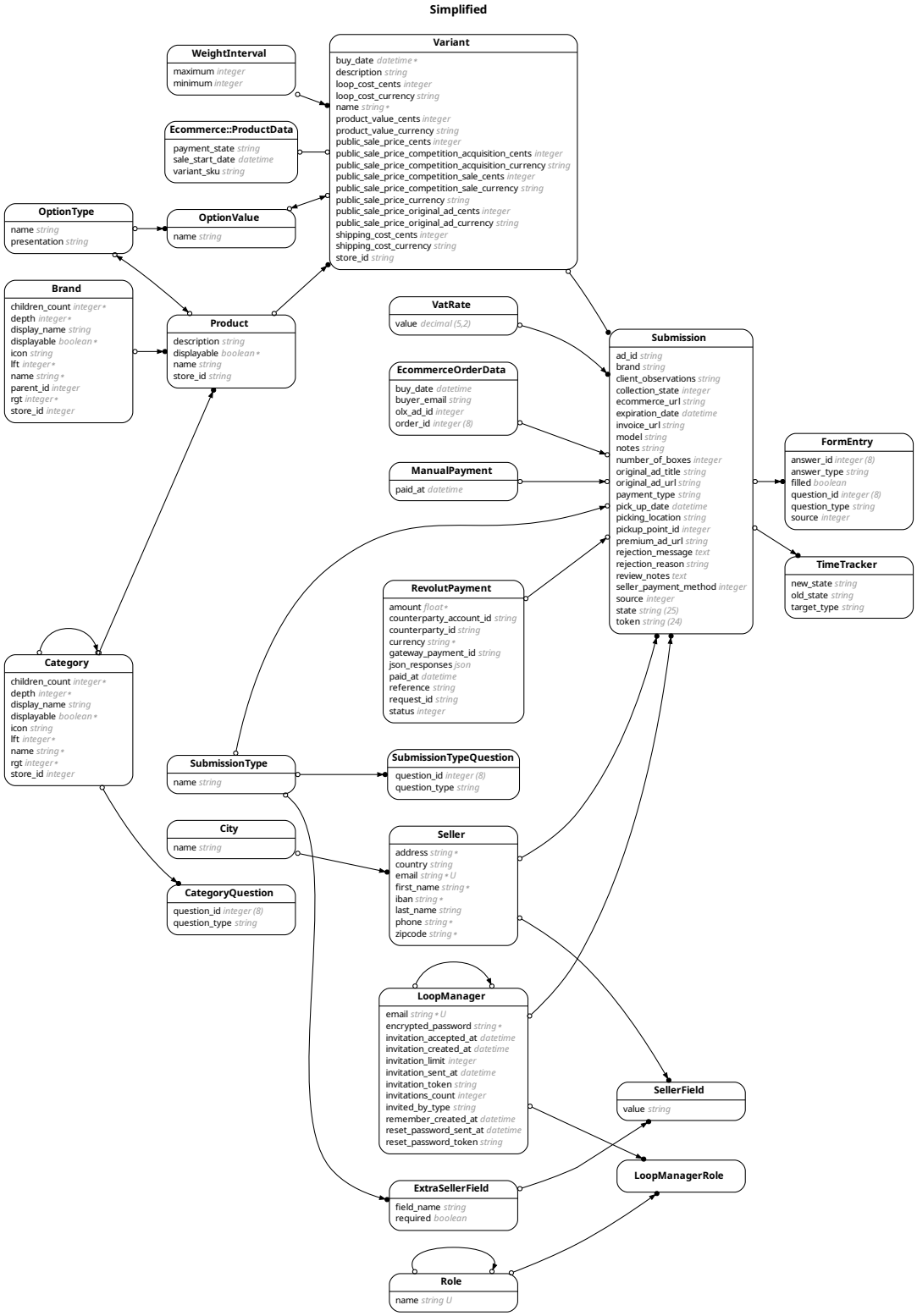

Figure 5.9: Simplified ER with Content

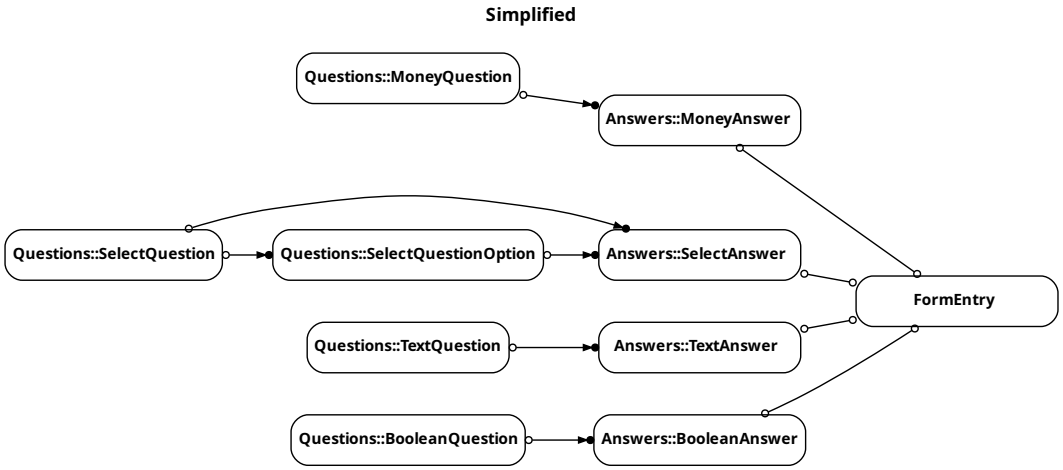

Figure 5.10: FormEntries ER

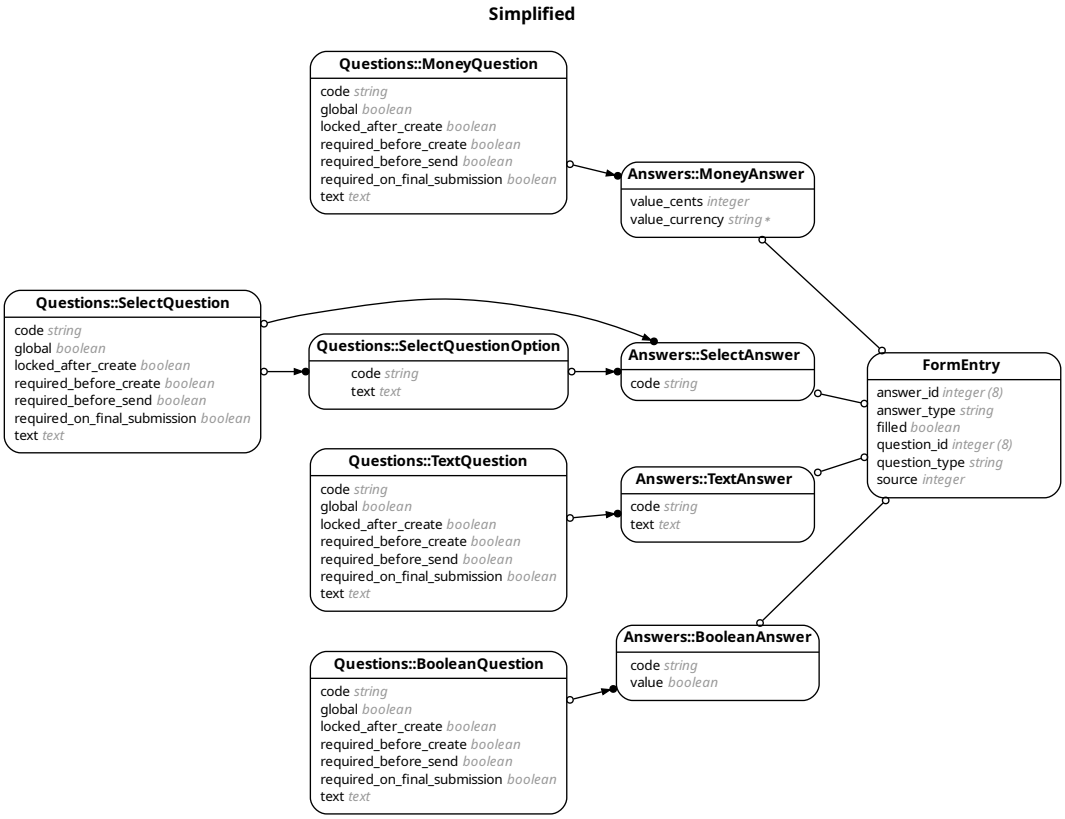

Figure 5.11: FormEntries ER with Content

These models work in conjunction to give controlled access to the system, enabling secure user authentication and role-based authorization within the application.

The dynamic nature of the forms is achieved through the FormEntries model, which stores the answers to each question associated with a specific product/category for a Submission.

The Answers and Questions themselves have different models and correspond to different tables in the database (as seen in figure 5.10 and 5.11), but through Rails' polymorphic tables, they can all be treated as instances of base classes Answer and Question.

Finally, the Category model is used to catalog the products and their associated questions. It is used to group similar products and to define the questions that will be asked for each product.

Managing category hierarchies in a relational database is a challenging problem due to the complexity associated with parent-child relationships and deep nesting.

The Categories are organized in a tree structure, which is hierarchical.

Traditional methods such as adjacency lists or path enumeration can lead to inefficient queries, especially when there is a need to retrieve an entire sub-tree or calculate the depth of nodes.

The nested set model was used as a solution to this problem. In this approach, each node in the hierarchy is assigned a pair of numbers: the left and the right value. These numbers are unique across the tree and are used to ascertain the hierarchical position of each node. Essentially, a parent node's left and right values encapsulate the left and right values of all its child nodes. This structural organization allows for efficient read queries. To find all descendants of a particular node, only one query is necessary for all nodes whose left and right values fall between the parent node's left and right values.

The implementation of the nested set model in the Submission Module is based on the *awesome nested set* gem [16].

5.4.2 E-commerce Module

The ER diagram for the E-commerce Module is shown in figure 5.12 - simplified, without the column attributives, and figure 5.13 - with them.

This module is based on the Solidus[17] E-commerce platform, which itself is based on Spree E-commerce. Out of the box, it provides a comprehensive set of functionalities for managing an online store, including product management, order processing, and payment processing. Yet many adaptations to the database schema were necessary.

Following is a simple description of the relevant models:

- **Spree::Taxon**: Represents a classification category for products, organized hierarchically.
- **Spree::Taxonomy**: Represents a tree structure for organizing taxons.
- **Spree::Classification**: Represents the many-to-many relationship between a product and a taxon. The classification of categories in the E-Commerce module is more

[16] `https://github.com/collectiveidea/awesome_nested_set`
[17] `https://solidus.io/`

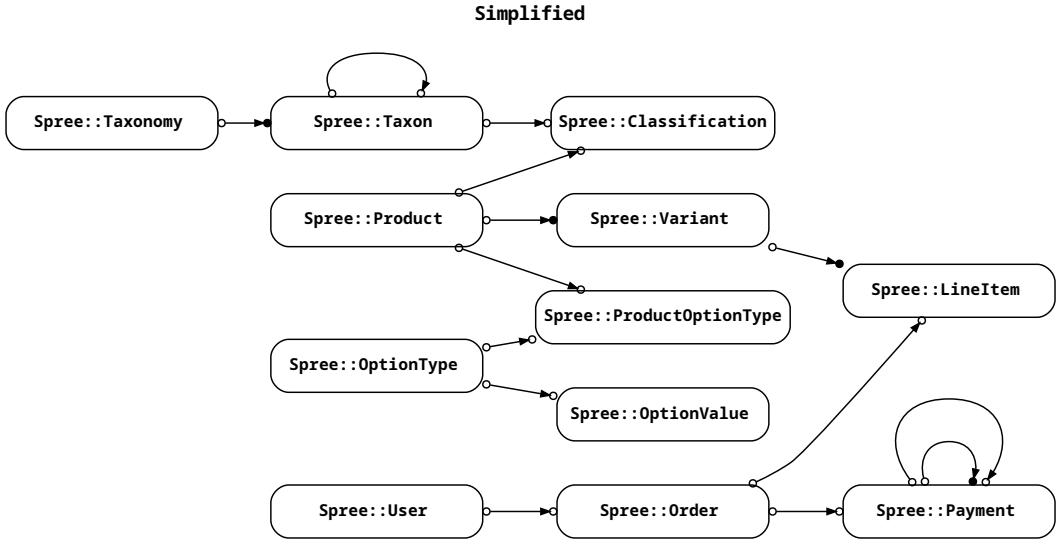

Figure 5.12: E-commerce Module Entity-Relationship Diagram - Simplified

powerful than the one in the Submission Module, as a product can be present in multiple categories or taxonomies.

- **Spree::Order**: Represents an order placed by a user. It includes information about the order total, state, associated user, shipping and billing addresses, and other relevant data.
- **Spree::Payment**: Represents a payment made by a user for an order. It includes information about the amount, associated order, payment method, and state.
- **Spree::LineItem**: Represents an item in an order. It includes the associated variant, order, quantity, and price.
- **Spree::User**: Represents a user of the e-commerce system. It includes information like email, password, login, shipping and billing addresses, and various authentication-related attributes.

The Product, Variant, Option Type, and Option Value models are similar to the ones described in the Submission Module.

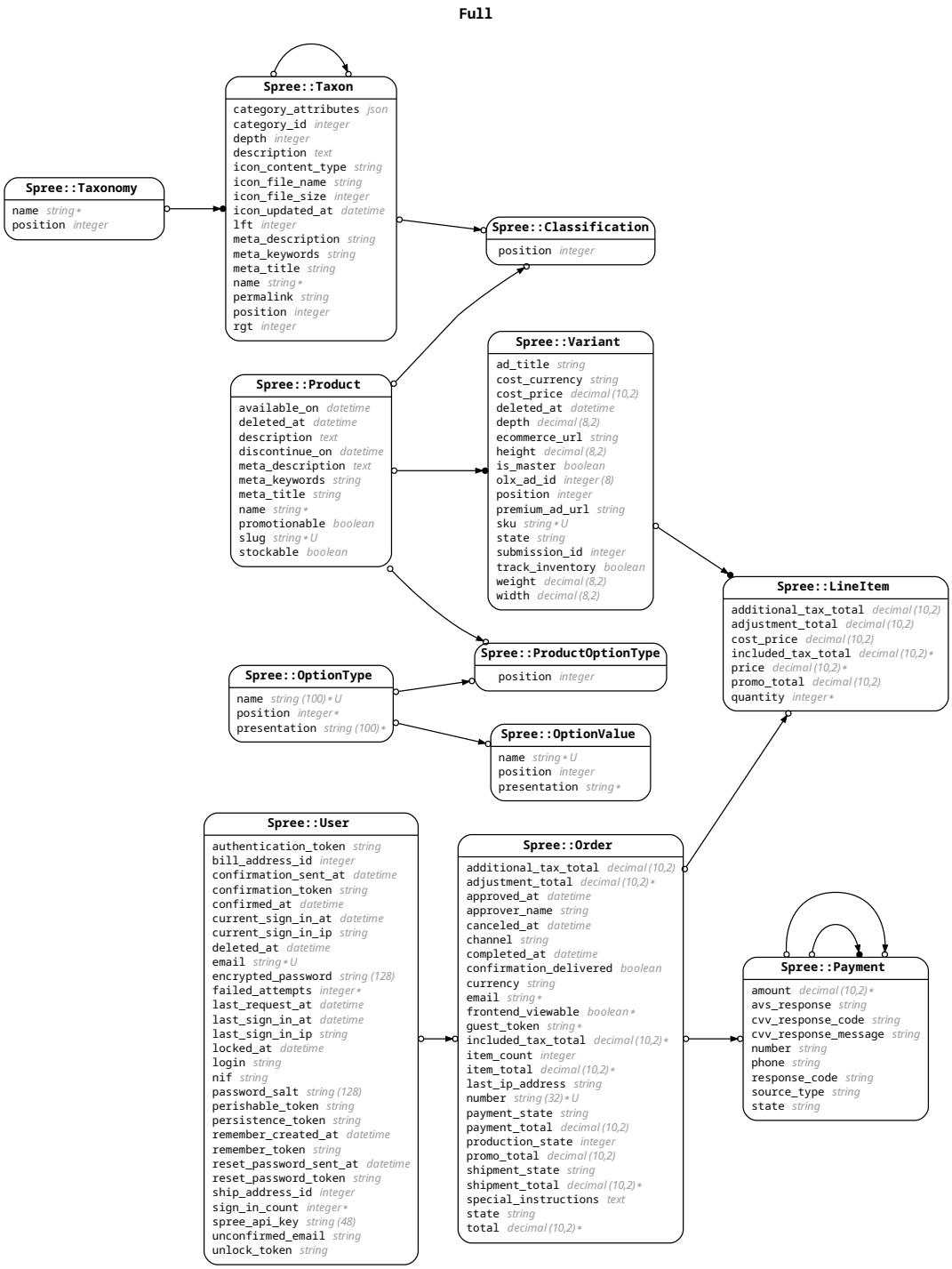

Figure 5.13: E-commerce Module Entity-Relationship Diagram - With content

Chapter 6

Development

This chapter describes the implementation phase of the project, discussing the development process, execution, project management, and deployment. It provides an overview of the software development work process and organizational standards followed, the tools utilized, and how the project requirements were met. Furthermore, it addresses some of the challenges encountered during the development phase and the solutions selected to overcome them, concluding with the current state of the project.

6.1 Development Process

This section describes the applied Agile methodology, team organization, task organization, and the process of code review and quality assurance.

6.1.1 Agile Methodology in Practice

As mentioned before, an Agile methodology was adopted and adapted to manage the development process for this project. This section provides a summary of the implementation of Agile principles and the necessary adjustments made to fit the project's requirements and the teams' needs. Specific adaptations are illustrated throughout the rest of the chapter.

The project was divided into smaller, manageable iterations similar to sprints in SCRUM. These sprints were planned weekly or bi-weekly, depending on the Project Owner's and business teams' priorities.

Each sprint contained a defined set of goals, that corresponded to one or more tasks, prioritized based on the project's requirements and currently available resources (namely team members). The roles, responsibilities, meeting schedules, and organization of said meetings are discussed in section 6.1.2.

Flexibility and responsiveness to change, key principles of Agile, were necessary during this project. Changes in requirements were incorporated into sprints, with task reprioritization as necessary. Often OLX or the business team would request changes after testing a feature, which would create new tasks and require the reprioritization of existing ones. Notable changes are discussed in subsection 6.1.1.

Continuous improvement was also a focus throughout the project. The weekly meeting also served as a retrospective moment to discuss successes, challenges, and potential im-

provements for the next sprint. This feedback was used to refine the development process and increase efficiency over time.

Aside from these meetings, the company's CTO, lead Project Manager, and Project Managers (in which the author was also later included) participated in weekly meetings to discuss the projects' progress, address any inter-project issues, discuss noteworthy design decisions, and manage the allocation of developers between projects.

In conclusion, the Agile methodology provided a flexible framework for managing the development process. By adapting its principles to the specific needs of the project and the team, the methodology enabled the team to consistently deliver value to the client and respond to changes in requirements.

Notable Changes and Adaptations

During the project, several notable changes in requirements occurred. These changes and the corresponding adaptations are summarized below.

Submission Main Flow Change

The original requirements specified that the OLX Manager would receive, either by email, or another means, the list of OLX advertisements that may be of interest to buy, recondition, and re-inject the listing into OLX as a new premium advertisement. The first steps of the flow would be as follows:

1. The OLX Manager would receive a list of advertisements.
2. The OLX Manager would appraise and filter the advertisements to buy.
3. The OLX Manager would, using the Submission BO, create a proposal to the Seller.
4. The Seller would receive an email with the link to the Proposal form, where they would accept or reject, and potentially fill in the required information.
5. The OLX Manager would confirm the submission, the item would be bought, and the flow would continue.

This flow was implemented, tested, and released to production. However, shortly after, due to changes in the management personnel responsible for the project on the OLX team, OLX decided to change the flow. It was decided that it would be best to add a button to their home page that would redirect to the proposal form. The first steps of the new flow became as follows:

1. The Seller would click on the button on the OLX home page.
2. The Seller would be redirected to the proposal form.
3. The Seller would fill in the required information and submit the form.
4. The OLX Manager would receive a new *submission proposal*, evaluate it, correct it if necessary, and advance it to the next step (or reject it).

This new proposal form was different from the one that was already implemented because at the time there was no predetermined information about the categorization and properties of the item. This created unforeseen problems and questions that needed to be addressed:

- Should the Catalog be open and/or enable the user to create new categories?
- Would the user know how to correctly categorize the item?
- How to handle spam and abuse, given that the proposal page is now public?
- How to handle the expected increase in submissions knowing that there are limited human resources to handle them?

To address these challenges, several adaptations were made. The catalog was initially opened only on leaf categories, and the user could choose the "Other" option, which would enable the creation of a new category/brand/product if needed. However, this led to the creation of many similarly named categories. To counter this, a "displayable" property was added to categories, brands, and products. When the user creates them, they are hidden from the publicly available options. The OLX Manager could later enable them.

Another feature that was added was the ability to merge categories, brands, and products. This way, if the user created a new duplicated category (name mismatch or typo, for example), the OLX Manager could merge it with an existing one, greatly simplifying the process of correcting the user's mistakes.

Usability Testing was also conducted with members who worked on the review process, to identify potential bottlenecks in terms of usability, and to try and improve the efficiency and speed of the process.

In the usability testing sessions, the Loop Managers were asked to perform their routine tasks, namely the submission review process from start to finish.

The tasks were as follows:

- Evaluate and Reject a new submission.
- Evaluate, Correct, Appraise, and make an offer to a new submission.
- Fill in the necessary data after the client accepts the offer, and emit the package slip.

While the Loop Managers were performing these tasks, they were asked to think aloud, to explain what they were doing and why, and to provide feedback on the process.

The main issues identified were:

- Some actions, such as rejecting a submission, were not accessible at the first stages of the review process. Even though the Submission Manager knew that the submission was not suitable, they had to go through some extra steps to reject it. The solution was to add a dynamic action bar, that is always on top of the submission page and contains the most common actions (available for the current state).
- The "Correct" action was comprised of filling in some data, and after watching the Loop Managers perform the task, it was clear that the data had grouping and that the form could be split into multiple steps. This was done, and the form was split into tabs for each type of data.
- The "Appraise" action required the Loop Managers to gather information, which required them to open multiple tabs and windows. This in conjunction with the fact that Submission Managers are often multitasking and reviewing multiple submissions

at the same time, led to the creation of a new feature. The websites of the most common sources of information were embedded in the Submission form, allowing the Loop Managers to quickly access them.

These changes improved the efficiency of the review process, and the usability of the application, but eventually, the decision was made to reduce the number of incoming submissions. The "Other" option was removed due to factors such as misuse from the Sellers; the lack of time for reviews; the effort necessary to review them; high reject rate for non-listed categories.

Aside from these changes, the feature was implemented in a way to also keeps the original flow working. Section 5.3 contains a sequence diagram that illustrates these flows.

Security reprioritization

In anticipation of the store's release, a security audit was scheduled by OLX. The audit consisted of penetration testing [26], utilizing a methodology known as black box testing [27], by their security team.

The testing was done on both authenticated and non-authenticated application endpoints of the Submission back office, proposals app, and the E-Commerce Backoffice.

To anticipate the audit, in addition to the standard tools that were already utilized during the regular development process (see section 6.1.4), the Open Web Application Security Project's Zed Attack Proxy (OWASP ZAP) tool [1] was used to scan the application for potential vulnerabilities.

The audit report identified several issues, many of which were determined to be false positives. However, in the interest of ensuring the secure launch of the store, it was decided to prioritize the resolution of these security issues instead of the scheduled sprints' tasks.

An adapted report containing the identified issues and the corresponding resolutions can be found in the appendix A.

Delivery Provider Change

In the initial stages of the project, the plan was to utilize CTT as the delivery provider. However, due to business delays and considerations, a decision was made to switch to DPD. As this change occurred early in the development process, the impact on the project was minimal.

To accommodate this change in a flexible and maintainable manner, while also preparing for potential modifications, it was decided to implement a more robust solution. This solution consists of using the Adapter design pattern, in conjunction with the Service Object pattern. It was developed in an isolated project, a ruby gem.

Ruby gems are packages of Ruby code, similar to modules or libraries in other languages.

The gem contains a service class (the PickUpPointsService) which acts like the API for the rest of the application.

The Service Object pattern is a design pattern commonly used in Ruby on Rails applications. It encapsulates a specific business logic operation or a distinct action in the application. This pattern is used to keep controllers and models lean by extracting complex logic or operations.

[1] `https://www.zaproxy.org/`

The Adapter pattern is a structural design pattern that allows objects with incompatible interfaces to work together. This pattern involves a single class, the adapter, which is responsible for communication between the main object (in this case, the PickUpPointsService) and the object that the main object cannot use directly (DPD Client).

The Adapter to be utilized by the Service is established in the initializer, a process that is essentially Inversion of Control (IoC). IoC is a programming technique where the flow of control is inverted compared to traditional procedural programming. The concrete application of this principle is called Dependency Injection, where the dependencies of a class are supplied to the class (injected) instead of the class creating them itself. The following code snippet illustrates this:

```ruby
# config/initializers/delivery_system.rb
# Setup
DeliverySystem.setup do |config|
    config.pickup_points_source = DpdPickupPointsAdapter
end

# Each adapter uses its own interface
class DpdPickupPointsAdapter < DeliverySystem::PickupPointsInterface
  def self.all
    Dpd::Client.instance.pickup_points.map do |pickup_point|
      DeliverySystem::PickupPointAdapter.new(
        id: pickup_point["number"]&.to_i,
        name: pickup_point["name"],
        county: pickup_point["postalCodeLocation"],
        #...

# Usage. For the rest of the application, it is transparent which adapter/source is being us
DeliverySystem.pickup_points.all # return all pickup points
```

In this case, the Adapter (the dependency) is injected into the Service, allowing the Service to remain agnostic of the specific implementation of the Adapter.

This design allows for greater flexibility and maintainability, as changes to the Adapter or Client implementations do not necessitate changes to the Service.

This change, although unplanned and more time-consuming, allowed for a seamless transition from the initially planned delivery provider and also provided a flexible architecture that could accommodate future changes and be reused in other projects.

6.1.2 Team Organization

The core development team was composed of one project manager (later replaced by the author), two backend developers, and one frontend developer and designer.

Initially, the author contributed to the project as a developer. Over time, his role expanded to encompass project management responsibilities, which entailed overseeing the development process in addition to active programming.

Inspired by the SCRUM methodology's *daily*, the team held daily meetings where each member reported on their current tasks, plans for the day, and any encountered blockers. This routine ensured that everyone on the team was updated on the current state of the project, helping resolve doubts. If necessary, there would be follow-up technical discussions with relevant team members.

The team's tech lead, a role assumed by the primary developer, facilitated technical planning meetings at the beginning of the project. During these sessions, the team discussed topics such as architectural planning, code organization, and library configurations.

In the later stages of development and maintenance, two business and operations team members also began attending the daily meetings. They provided insights into usability and informed the developers of business aspects related to the project.

Later, weekly meetings were also instated, whose purpose was to identify and address issues, facilitate communication and collaboration, plan and organize the next week's work, and set and track goals. Contrary to the daily meetings, these were mostly conducted in person and were longer in duration. These weekly meetings were akin to the *sprint planning* and *sprint review* meetings in the SCRUM methodology.

During these, the team made decisions on what tasks to include for the next week, taking into account the project's needs and any changes in the requirements.

6.1.3 Task Organization

The tasks were organized using a Kanban-style board for visualization, facilitated by the Software as a Service (*SaaS*) platform ClickUp.

The Kanban system, originating from Toyota's just-in-time (JIT) production system, is a method for managing knowledge work with an emphasis on just-in-time delivery while not overloading the team members.

In this system, tasks are represented as cards that progress through different stages of a workflow as they are completed.

This approach offers several benefits. Some of these can be seen in figure 6.1 [2]. Firstly, it provides a visual representation of work in progress, which aids in understanding the flow of work and identifying any bottlenecks. This is especially useful for Project Managers, in the context of software development, where tasks can be complex and have many dependencies. Secondly, it allows for flexibility in task management, as tasks can be added, removed, or prioritized based on the project's current needs. Lastly, it encourages collaboration and transparency among team members, as everyone has a clear view of the state of the project.

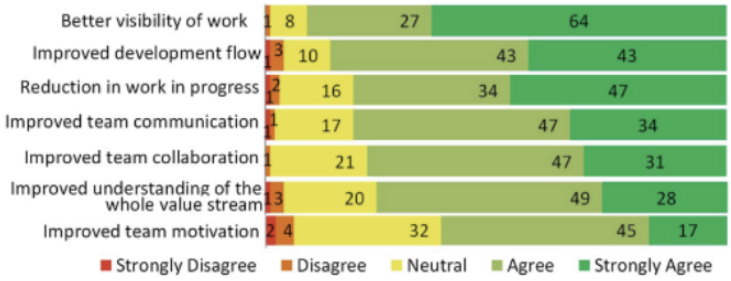

Figure 6.1: Kanban benefits

In addition to task management, ClickUp also facilitated:

[2] (sourced from: https://www.google.pt/books/edition/Agile_Processes_in_Software_Engineering/RtlCDwAAQBAJ?hl=en&gbpv=1&dq=kanban+benefits&pg=PA162&printsec=frontcover)

- Team collaboration, through card-specific discussion threads.
- Time tracking. Task-specific time tracking was not enforced, only project-based tracking, for two reasons. First, it was been shown that task-based time tracking can be detrimental to productivity and team morale [28], and second, the team was small enough that it was easy to keep track of the time spent on each task. Moreover, the time tracking was only useful for company management reasons, because some people were working on multiple projects at the same time.

ClickUp is designed for flexibility and customization, and each team adapted the boards to their needs. The OLX project has eight boards (called *spaces* in ClickUp), the first four are management-oriented:

- **Planning:** This board was used for initial development, encompassing research tasks, flow definition, benchmarking of similar platforms or functionalities, database schema design, and the creation of several architectural diagrams. Tasks would be created in the *Backlog* and would flow linearly, reaching the *Closed* column or *Artifact* if the task produced a diagram or another artifact.

$$Backlog \rightarrow To\ Do \rightarrow In\ Progress \rightarrow Review \rightarrow Closed/Artifact.$$

- **Management:** This board mirrors the columns of the Planning board, except *Artifacts*, and is mainly used to track time for management tasks, CI/CD tasks (such as updating the environmental variables for deployments), and minor development environment-related tasks.

$$Backlog \rightarrow To\ Do \rightarrow Pending \rightarrow In\ Progress \rightarrow Review \rightarrow Closed$$

- **QA:** Standing for Quality Assurance, this board is where the business team submits cards for bugs, improvements, and feature requests. This board works more akin to a ticketing system because the tasks can only be moved to a final *Moved to Development* column. Tasks can be also moved between boards, but usually, because the business team has a different and less technical understanding of the software, the cards often need to be modified or adapted by the project manager before being moved to the development boards.
- **Deliverables:** This board is exclusive to the Project Manager. It is used to schedule and visualize the timeline for larger features instead of smaller, specific tasks. It also has a Gantt-style timeline view. It was later replaced by a new, dedicated ClickUp feature, *Goals*, that served the same purpose but had a more adequate user interface, such as goal completion metrics. Although only the project manager had *write* access to the board, it was built with the whole team present during the weekly meetings.

The next four boards are developer-oriented and have the following purposes:

- **Design:** Used to track design tasks, mainly the making of components using Figma wireframes. The columns are slightly different, to take into account the client feedback:

$$Backlog \rightarrow Pending \rightarrow To\ Do \rightarrow In\ Progress \rightarrow Internal\ Approval \rightarrow Client\ Approval \rightarrow Rejected \rightarrow Approved$$

The last three, Frontend, SM Backend, and EC Backend, all follow the same flow. The columns are the same, but the tasks are separated by the module they belong to. The purpose of each column is summarized in the table 6.1.

Backlog → *Planning* → *Pending* → *Issue/Bug* → *To Do* → *In Progress* → *MR Review* → *PM Review* → *PO Review* → *Ready for Production* → *Closed*

- **Frontend:** Used by the frontend developers. These include tasks from the Submission Module (SM) Frontend and e-commerce module (EM) Frontend.
- **SM Backend:** Used by the backend developers, for tasks in the SM.
- **EC Backend:** Used by the backend developers, for tasks in the EC.

Column	**Task status**
Backlog	Yet to be worked on.
Planning	Need more planning to be workable.
Pending	Reached a blocker, cannot continue.
Issue/Bug	Related to issues, or went back due to detected bugs.
To Do	Ready to be picked up by a developer.
In Progress	Currently being worked on.
MR Review	Have a corresponding Merge Request, awaiting approval.
PM Review	Ready for merging to the Staging branch, awaiting Project Manager's approval.
PO Review	Ready for testing by the Project Owner.
Ready for Production	Accepted by the Project Owner, ready for merging to the main branch.
Closed	Completed.

Table 6.1: Description of Task Stages

Following the Agile method, some tasks may bypass some columns, when these are not applicable. For example, utility tasks such as creating factory objects to help populate the database do not need the project owner's approval.

Furthermore, each task is assigned a priority level, ranked from *Low*, *Normal*, *High*, and *Urgent*, a due date, and optional tags, defined by the Project Manager.

6.1.4 Code Review and Quality Assurance

Code review is a process in which one or more developers review the source code of a software project to identify problems, suggest improvements, and ensure that the code adheres to best practices and standards. It has been shown to improve code quality and reduce the number of bugs in software projects [29], even with modern, lightweight, and asynchronous code reviews.

In this project, three primary branches were utilized: development, staging, and main. Feature branches, dedicated to the development of new features, originated from (branch of) and are targeted back to (merged to) the development branch.

Each merge request includes a description in a change-log-like format, outlining what was added, altered, or removed. This description also provides a slot to justify the changes

and an explanation of the chosen approach, aiding the reviewer in understanding the functionality before formally initiating the review.

The feature branches also have the ClickUp task ID in the branch name, the Merge Request body, or a commit message, to establish a clear link and aid the navigation between the code and its corresponding task.

Reviewed and approved changes are merged into the development branch. Given the relatively small size of the team, a single approval was sufficient for a merge request to be accepted.

Aside from Merge Reviews, code quality was also maintained with automatic code-checking tools:

- **RuboCop** [3]**:** is a code linter that checks code against predefined rules to ensure consistency and best practices in coding style. The rules are defined in a configuration file. The rules used in this project were based on the Ruby Style Guide by Shopify [4] and adapted to reach a consensus within the team.
- **Brakeman** [5]**:** is a security scanner that analyzes Ruby on Rails applications for known security vulnerabilities.
- **Bundle-Audit** [6]**:** is a tool for identifying and reporting vulnerabilities in Ruby gems (library packages) that are used in a Rails application.
- **Bullet** [7]**:** is a gem (library package) for Rails that helps detect and prevent N+1 queries, which can be a common performance issue in Rails applications.
- **Rails-perftest** [8]**:** is a tool that does benchmarking and provides profiling metrics. It generates performance reports and measures memory usage.
- **ESLint** [9]**:** is a static code analyzer for javascript and typescript, used for identifying, and reporting on patterns found in ECMAScript/JavaScript code, to make code more consistent and avoid bugs.

6.2 Development Execution

This section discusses the tools used for planning, managing, and executing the project, including the implementation of Continuous Integration and Deployment (CI/CD) and testing methodologies.

6.2.1 Planning and Management Tools

These tools were used while creating the diagrams and flowcharts.

[3] `https://github.com/rubocop-hq/rubocop`
[4] `https://github.com/Shopify/ruby-style-guide/tree/main`
[5] `https://github.com/brakeman/brakeman`
[6] `https://github.com/rubysec/bundler-audit`
[7] `https://github.com/flyerhzm/bullet`
[8] `https://github.com/notebook/rails-perftest`
[9] `https://eslint.org/`

- **Draw.io** [10]**:** Now also known as Diagrams.net - a free and open source, browser-based graph design tool. Used to create diagrams like the C4 architecture (see section 5.1), and others.
- **ONDA** [11]**:** Online database architect. Used to draft the initial database schema for the Submission module.

These are the tools and software used for communication, messaging, task and time management.

- **Microsoft's Teams** [12]**:** : The primary way of communication and information sharing within TheLoop Co. Has features such as video conferences, messaging, file sharing, and scheduling (reunions and daily meetings).
- **ClickUp** [13] : Project management suite of several tools. This tool enabled the team to do task management and tracking and issue/bug handling. The task list can have different views, but mainly the board view was used (similar to Kanban, as seen in figure 6.2). Discussions and clarifications of the tasks were also done here by sharing comments, images, and screen recordings. This process helps with keeping the discussion about specific features centralized and organized. It also has integration with Gitlab, discussed more in detail in section 6.2.2.

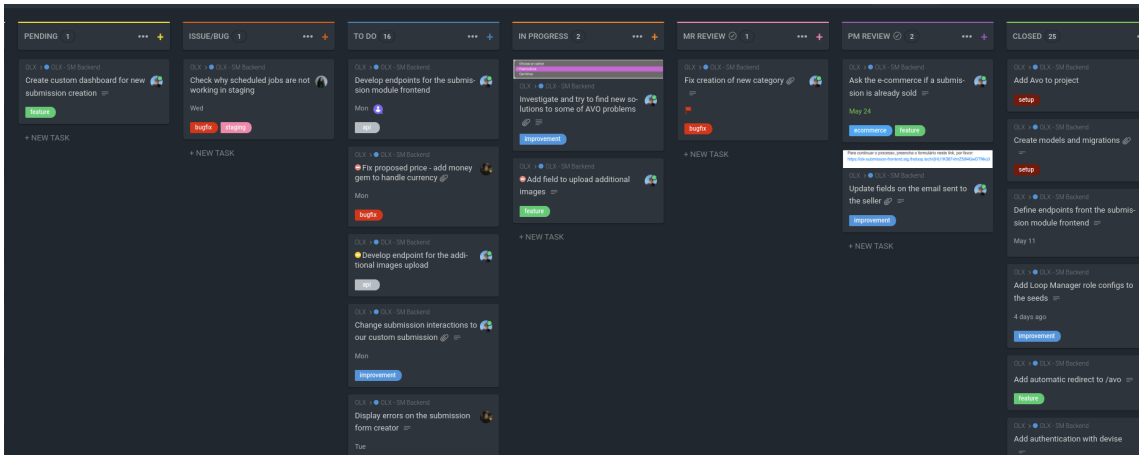

Figure 6.2: Example of a Kanban style task view during the development process in ClickUp

6.2.2 Development Tools

In this section are discussed the tools used during the actual development process, the writing of the code, source control, testing, and deployment.

Text Editor

According to a 2019 survey [30] an average developer spends approximately 32% of their working time writing new code or improving existing code, and 19% on code maintenance.

[10] `https://app.diagrams.net/`
[11] `https://onda.dei.uc.pt/v4/`
[12] `https://www.microsoft.com/en-us/microsoft-teams/group-chat-software`
[13] `https://clickup.com/`

This means that roughly half of the time is spent either writing or reading code. Having a good tool to facilitate this is very important.

In the Loop Co., the code editor used by the majority is VSCode (Visual Studio Code) [14], and sometimes, when accessing remote servers or doing small changes, the vim[15] terminal-based text editor. Although VS Code is not a fully-fledged Integrated Development Environment (IDE), it is lightweight, highly customizable, and can integrate with various workflow tools through the use of *extensions*. This, and the fact that it is the most popular code editor at the time of writing [31], makes it a reasonable choice.

Git and GitLab

For source control, Git was used. The Git workflow used was GitLab Flow [32]. A Git workflow is a recipe or recommendation on how to use Git, which includes which branches to use, the naming schemes, and the associated permissions.

The GitLab flow consists of the following types of branches:

- Development and main branches: The `main` branch must be always in a deployable state. Given that the project does not need to support several concurrent release versions, one stable branch is enough. The `development` branch is the base branch from which the developers create feature branches.
- Environment branches: specific to each environment. A `staging` branch for internal testing and client demos and a `development` branch for the most up-to-date version of the code.
- Feature branches: all features related to an assigned task, *branch of* the `development` branch. This also includes non-urgent bug fixes, improvements, and styling.
- Hotfix branches: for breaking bugs that were only detected after being released to production. They may be merged into the necessary branch directly (or cherry-picked), but they must also be merged into the development branch.

This flow is a good fit for this project, considering the team size and the feature-driven, agile development cycle focused on continuous delivery and multiple environments.

As for the remote repository for Git, GitLab was used. It offers features such as collaboration through Merge Requests, where developers may comment and perform asynchronous code reviews, and the CI/CD pipeline, which integrates with Rancher for continuous delivery and deployment (see section 6.3.1).

Docker

The various modules, while designed to interact with each other, must be developed as separate, decoupled, and deployable units. Each application (component) has its own set of system dependencies, configurations, and services that need to be available for it to function properly. To manage these requirements and facilitate the development process, Docker containers were utilized.

[14] `https://code.visualstudio.com/`
[15] `https://www.vim.org/`

These containers can be thought of as lightweight virtual machines [33].

Each container encapsulates all the dependencies and configurations needed by each module, creating a standalone unit that can be deployed. This allows the program to run consistently across any machine with Docker installed, regardless of the underlying operating system or its configuration.

The containers used were:

- Ruby on Rails containers: one for the submission module and another for the e-commerce module.
- PostgreSQL - relational database. One for each application.
- Sidekiq - asynchronous job scheduling. For sending emails and performing periodic tasks.
- Redis - an in-memory cache and message broker (connects Rails applications and Sidekiq).
- NextJS - frontend framework for the Submission Proposal page module.

These multi-container environments are defined and managed by a tool called Docker Compose [34] for the local development environment. For the staging and production environments, the process is different, and it is expanded in section 6.3.1.

6.2.3 Continuous Integration and Deployment

CI/CD, standing for Continuous Integration and Continuous Delivery, is a set of practices and tools that automate the process of building, testing, and deploying software. The CI/CD solution used was GitLab CI/CD.

The Continuous Integration part of the pipeline is responsible for the `build` and `test` stages. The common setup is to run tests (and or linters) when a new Merge Request is created. If the tests fail, the Merge Request is blocked until new changes are pushed, causing the pipeline to run again.

The Continuous Delivery part of the pipeline is responsible for the deployment stage.

In practical terms, it consists of configuring a pipeline (in a file called `.gitlab-ci.yml`) that defines different steps to be executed.

For this project, an existing configuration was used to bootstrap the project, and changes were made when the need arose. The final pipeline was divided into the following stages: `setup`, `test`, `build`, `validate`, and `deploy`.

During the `setup` stage, the environment was prepared for the subsequent stages. This consists of running a setup script that pulls the terraform file from the repository and sets up the necessary environment variables. Most of these variables are sensitive and are stored in GitLab's CI/CD settings, which are only accessible to project maintainers and project managers.

Terraform [16] is a tool managing infrastructure as code. It is used to define the infrastructure in a file, that is also contained in the repository. Aside from the variables and configurations such as the number of instances, the instance type, the memory limits, CPU limits, and hostname are defined in several terraform files.

[16] `https://developer.hashicorp.com/terraform/intro`

The `test` stage was designed to run tests on the application. It specifically ran request tests on the REST API endpoints, to ensure that all endpoints were functioning as expected. The tests are discussed in more detail in section 6.2.4.

The `build` stage involved building the application in a Docker container and pushing it to the GitLab Container Registry.

The `validate` state only runs the `terraform validate` command, checking whether the terraform files are valid.

Finally, the `deploy` stage involved applying the infrastructure changes to the Kubernetes cluster using Terraform. This was done for the staging branch automatically and for the master branch manually. The details of the deployment strategy are discussed in section 6.3.1. It is to be noted that even though the stages are defined in the pipeline, depending on the branch, some stages may be skipped or be different. This is done with the dependencies keyword for each stage. For example, the `deploy` state, when reached by starting the QA pipeline, creates and deploys a temporary QA instance.

This CI/CD pipeline played a crucial role in maintaining the quality of the software, facilitating collaboration among developers, and speeding up the software development process. The automation of these stages facilitates issue detection that is not directly related to the application code. This also ensures that the application is always in a deployable state, reduces the time and effort required to release new features, and by caching the build artifacts enables quick rollbacks in case of issues.

6.2.4 Testing

Testing is a crucial aspect of software development that ensures the application behaves as expected, improves the quality of code, and prevents regressions when changes are made.

In Ruby on Rails applications, RSpec [17] is a popular testing framework that is used for behavior-driven development (BDD). It provides a clean and clear syntax, using Domain Specific Language (DSL) for the test cases.

Two types of tests were used in this project: unit tests and API tests.

Unit tests were primarily used to test the most critical parts of the application, specifically the submission user models, and financial calculations. These tests were designed to validate the functionality of individual components in isolation, ensuring that each part performs as expected. For the submission, the functionalities being tested were the validations and integrity constraints of the model for the different states of the submission. For the user model, the tests were designed to validate the users' password security requirements and user roles and permissions.

An example of a unit test for the submission model is shown in figure 6.3, and the corresponding output in figure 6.4.

API testing is a type of software testing that involves testing application programming interfaces (APIs) directly and as part of integration testing to determine if they meet expectations for functionality.

During the initial phase of the project, when many changes were being made both to the API and to the submission model, the testing was done manually. As the project progressed and reached maturity, manual testing would be not only error-prone but also

[17] `https://rspec.info/`

```ruby
RSpec.describe(Submission, type: :model) do
    context "when advancing in the submission lifecycle" do
        let(:submission) { build(:submission) }

        it "fails to save if required before_create questions are not answered" do
            question = build(:question, required_before_create: true)
            submission.form_entries << build(:form_entry, question: question)
            expect(submission.save).to(be(false))
        end

        it "saves if before_create questions answered" do
            question = build(:question, required_before_create: true)
            answer = build(:text_answer, text_question: question)
            submission.form_entries << build(:form_entry, question: question, answer: answer)
            expect(submission.save).to(be(true))
        end

        #...
```

Figure 6.3: Unit test for the submission model

```
➜  /project git:(development) ✗ rspec spec/models/submission_spec.rb -fd

Submission
  when created
    is in draft
    global questions are added
    category specific questions are added
    submission type specific questions are added
  when advancing in the submission lifecycle
    fails to send if missing associations
    # ...
    when on qa accepted state
      checks if variants product value is set
      checks if variants PVP value is set
      checks if variants Loop Cost value is set
  when calculating financial values
    has correct direct margin without vat
    has correct loop sales commissions
    has correct olx final liquid margins
    has correct loop theoretic liquid margins

Finished in 19.82 seconds (files took 1.5 seconds to load)
31 examples, 0 failures, 0 pending
```

Figure 6.4: Unit test output for the submission model

```
#...
category: {
  type: :object,
  description: "A category can contain 0 or more sub-categories, the available"\
    " states and product models available for current category.",
  properties: {
    presentation: { type: :string },
    value: { type: :integer },
    subcategories: {
      type: :array,
      description: "Each sub-category is a select option.",
      items: { "$ref": "#/components/schemas/leaf_category" },
    },
  },
  required: [:presentation, :value, :subcategories],
},
#...
```

Figure 6.5: OpenAPI schema definition for the category object

extremely time-consuming.

One of the problems is rebuilding/recreating the data to test the behavior in different states. Constructing the submission model, having several states and state-specific validations, would be time-consuming if done manually. To solve this problem, the FactoryBot[18] and Faker[19] gems were used.

FactoryBot is a library that facilitates the creation of data. Its main advantage, besides the easy-to-use DSL, is that it integrates with RSpec, and with the correct configuration, can pull data from Faker, a library that generates such as emails, phone numbers, product descriptions, etc. It achieves this by using the factory pattern, where factories are defined for each model. Factories can have hierarchical structures and support traits and callbacks, making it easy to build complex data structures.

This facilitated the creation of test data quickly, in all the states and with all the configurations (e.g.: valid, invalid, with questions, without answers, etc.).

For the API testing, RSwag [20] was used.

RSwag allows to write and run RSpec tests that mock requests, which were then used to generate OpenAPI [21] specification. RSwag also provides a Swagger UI [22] interface that allows visualization and interaction with the API endpoints.

Then, a TDD (Test Driven Development) approach was used to write the tests for each endpoint. This first step was to design the API schema using the OpenAPI specification. An example of a schema definition is shown in figure 6.5.

Afterward, the tests were written, and the endpoints were implemented.

One of the benefits of using RSwag is that the tests generate the OpenAPI specification file. Each test belongs to a URL path (or application endpoint). The RSpec gem was expanded to also add the responses of successful requests as examples in the specification.

[18] https://github.com/thoughtbot/factory_bot
[19] https://github.com/faker-ruby/faker
[20] https://github.com/rswag/rswag
[21] https://www.openapis.org/
[22] https://swagger.io/tools/swagger-ui/

```ruby
RSpec.describe("api/submissions", type: :request, document_response: true) do
#...
path "/api/submissions/options" do
  get "parameters for new submissions" do
    description "For the flow where the seller makes the first proposal."\
      "Returns the available options and presentations for the select fields."
    produces "application/json"

    response 200, "Adheres to options schema" do
      schema "$ref": "#/components/schemas/new_submission_options"

      run_test!
    end

    response 200, "only sends displayable sub-categories" do
      schema "$ref": "#/components/schemas/new_submission_options"

      before do
        categories = Category.categories.not_leaves
        categories.update(displayable: true)
        10.times { create(:category, displayable: true, parent: categories.sample) }
        10.times { create(:category, displayable: false, parent: categories.sample) }
      end

      run_test! do
        data = JSON.parse(response.body)
        # get all the subcategory ids for all categories
        subcategory_ids = data["product"]["categories"].map do |c|
          # Select only non-negative categories. We use -1 to send the "Other" option
          c["subcategories"].filter_map { |sc| sc["value"] >= 0 ? sc["value"] : nil }
        end.flatten

        subcategories = Category.categories.where(id: subcategory_ids)
        expect(subcategories.count).to(be > 0)
        expect(subcategories.pluck(:displayable).all?).to(be(true))
      end
    end
# ...
```

Figure 6.6: RSpec tests for the submission API

An example of the submission API tests are shown in figure 6.6, and the corresponding results in figure 6.7.

This approach provided comprehensive and accurate documentation of the API, with an interactive page to test the API without axillary tools like Postman or Insomnia. Yet the bigger benefit was that it allowed automatic API endpoint testing, ensuring that the API was always in a deployable state.

6.3 Deployment and Maintenance

This section discusses the strategy used for deploying the software and how the software is maintained and updated after deployment.

```
➜  /project git:(development) ✗ rspec spec/requests/api/submissions_spec.rb -fd

api/submissions
/api/delivery_system/pickup_points
get
produces non-empty array
returns a 200 response
/api/submissions/options
get
Adheres to options schema
returns a 200 response
only sends displayable sub-categories
returns a 200 response
only sends displayable brands
returns a 200 response
models belong to chosen brand and category
returns a 200 response
# ...

Finished in 19.93 seconds (files took 1.6 seconds to load)
21 examples, 0 failures
```

Figure 6.7: RSpec tests output for the submission API

6.3.1 Deployment Strategy

The deployment strategy is used to ensure a continuous and smooth transition from the development environment to the production environment. As mentioned before, the project is composed of three repositories. Each repository contains a docker-compose file, and each project has its dependencies containerized, as described in section 6.2.2.

When the project is deployed, however, this is no longer the case. The containers are orchestrated using Kubernetes, which is an open-source platform for deployment, scaling, and operating application containers [23].

It groups containers that make up an application into logical units for easy management. All three projects are deployed to the same cluster, which is managed by Rancher.

Rancher [35] is a tool used to manage deployments of these containerized environments. The purpose of this tool is better explained from this excerpt from *Container Orchestration With Cost-Efficient Autoscaling in Cloud Computing Environments* [36]:.

> "Container orchestration platforms, such as Kubernetes (Hightower et al., 2017), Docker Swarm (Naik, 2016), and Apache Mesos (Hindman et al., 2011), are responsible for the efficient orchestration of such applications in shared compute clusters. These platforms manage the lifecycle of containers as well as the usage of cluster resources and hence, one of their main goals is to (near) optimally place containerized applications on the available nodes."

Rancher is also an open-source software platform that provides a unified way to work with Kubernetes. The Loop Co. hosts its own Rancher server. The clusters are organized by namespace (environment) and project.

Project Managers and some developers have access to the Rancher server, where they can

[23] `https://kubernetes.io/`

change the number of replicas, ingress rules, and other configurations. Access to Rancher is usually done through the web interface but for simpler tasks, like command line access or checking the logs of a specific pod, the *kubectl*[24] or *k9s*[25] CLI tools were also used.

The deployment itself is initiated through GitLab's CI/CD pipeline, as described in section 6.2.2. The pipeline is triggered when changes are merged into the staging or main branches.

Upon being merged to the staging branch, the CI/CD pipeline automatically builds the application in a Docker container, validates the infrastructure changes using Terraform, and deploys the changes to the staging cluster.

After the changes in the staging branch are tested and approved, they are merged into the main branch. The CI/CD pipeline then repeats the same process to deploy the changes to the production environment. However, this final deployment step requires confirmation from the Project Manager to ensure that only intended changes are deployed to production.

In this way, Kubernetes provided several features that contribute to the quality attributes of the software, summarized in Table 6.2.

6.3.2 Maintenance and Updates

Maintenance and updates were handled as an ongoing process. The Agile development methodology was followed, which allowed for regular updates and improvements to be made. As mentioned, features and bug fixes were developed in feature branches, reviewed, and tested before being merged into the development branch.

For error tracking and performance monitoring, Sentry[26] was used. Sentry is an open-source tool and was also self-hosted by The Loop Co.

This allowed developers to monitor the software in real-time, and catch errors in the production and staging environments, which is done by integrating Sentry with the application code, which is done by adding code in specific parts of the application, like exception blocks or error handlers. When an error occurs, the Sentry package captures the error and sends it to the Sentry server, where it is stored and displayed in a project-specific dashboard. The errors can be filtered by the environment, the type of error, and the frequency of the error. The developer can then inspect the error and its stack trace (if present) by accessing the dashboard. Email notifications for critical errors can also be configured.

What is sent to Sentry can be customized, and it is possible to send additional information, such as trace ID, environment, host system information, and any other relevant data. The trace ID is a unique identifier that is generated when a request is made, and it is passed along to all the services that are involved in the request. This identifier is passed in a specific request header and is used to correlate errors and logs across services. This improves traceability and overall system observability because it allows for quick identification of the service and code that caused the error, which streamlines the debugging process for errors that happen on live deployments.

Preventative steps were also taken to filter out sensitive information such as the user's email, from the logged errors.

This proactive approach to error detection enabled quick response times to issues, improving the overall reliability.

[24] `https://kubernetes.io/docs/reference/kubectl/`
[25] `https://k9scli.io/`
[26] `https://sentry.io/welcome/`

Kubernetes Feature	Contributed Quality Attributes
Automated rollouts and rollbacks: Progressively rolls out changes to an application or its configuration, ensuring that not all instances are updated at the same time. While one container is initializing, the old one is still active. When the new one responds to the health check, the old one is terminated. If something goes wrong, it is possible to roll back the changes.	Reliability, Availability
Load balancing: Distributes network traffic to ensure that the deployment is stable. For the public containers, two replicas were used.	Performance, Efficiency, Reliability
Storage services: Allows automatic mounting of a storage system of choice. For this project, a storage container running PostgreSQL was used for the staging environment. For production, the Digital Ocean PostgreSQL was used.	Modifiability, Portability
Secret configuration and management: Allows storage and management of sensitive information, such as passwords, OAuth tokens, and API keys.	Security
Self-healing: Detects and replaces instances when they fail, and kills instances that does not respond to the health check. Does not make them public until they are ready.	Reliability, Availability
Horizontal scaling: Allows scaling of the application up and down, with a UI, or automatically based on rules (such as CPU usage, for example).	Performance Efficiency, Scalability

Table 6.2: Kubernetes Features and their Contributions to Quality Attributes

6.4 Current State and Future Work

This section discusses the current state of the product and any planned future enhancements or features.

As of the time of writing, the project has processed over 3200 submissions. The distribution of the submissions by (leaf) category is shown in Figure 6.8.

The submissions are distributed across various states as shown in the table 6.3.

The total amount for items bought was around €4,000, three-fourths being manual payments (through the initial SIBS integration), the rest being paid through Revolut. Around 65 submissions have been accepted and paid, with an average payment per item of €70.00.

With this, the pilot phase of OLX 2nd Life has concluded, and The Loop Co. has moved

State	Number of Submissions
Canceled	128
Client Rejected	243
Loop Rejected	2222
Draft	1
In Collection	31
Form Filled	1
QA Accepted	6
QA Rejected	6
Client Accepted	37
Sent	325
In-Store	54
Pending	183

Table 6.3: Submissions by State

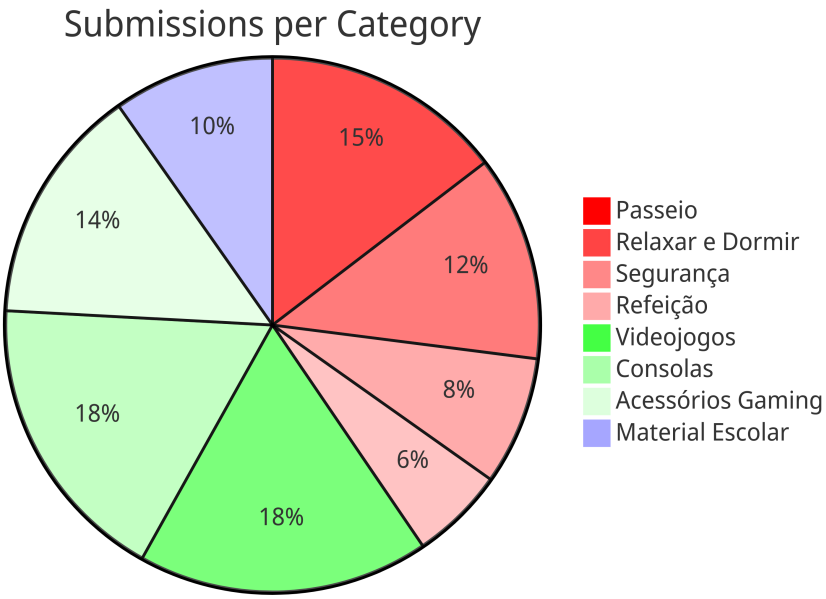

Figure 6.8: Submissions by State

forward with the development of the LoopOS project. This system aims to be an all-encompassing platform that not only facilitates the buying and selling of second-hand goods but also serves as an integral part of Loop Co.'s larger ecosystem.

The immediate next steps focus on gathering what was learned with this project and integrating the knowledge gained into the LoopOS system. Significant advancements have already been made in this direction. The Store module has been adapted and integrated, as has the SUbmission Frontend. Additionally, Revolut Payments, developed separately as a gem, have been used in this new system.

The Submission Module, one of the cornerstone elements of LoopOS, has served as a prototype and reference for developing a more robust and scalable submission system.

Chapter 7

Conclusion

The journey of this thesis has been invaluable, presenting many opportunities for learning, growth, and hands-on application of software engineering methodologies and principles. The process provided a unique opportunity for the author and the team to understand the landscape of Circular E-commerce, setting the foundation for building a base of knowledge for future developments in this area.

The engagement with a real-world client was an enlightening experience, requiring a balancing act between the client's needs and the project's scope while operating under a set of technical and business constraints.

During the implementation phase, the project began as a Minimum Viable Product, fulfilling the highest-priority requirements. This period posed considerable challenges, especially around mastering new and different technologies and working as a part of a development team. However, the obstacles encountered only served to enrich the learning experience.

On the testing and validation front, functional tests revealed essential insights into the system's robustness and highlighted several areas for improvement. The platform has demonstrated satisfactory performance and most importantly, reached the client's expectations and made its way to production.

Moreover, the experience allowed for a significant enhancement in professional skills, including working with modern technologies, effective teamwork, and full-stack development.

In summary, this thesis project has been a microcosm of the challenges and triumphs that come with large-scale software engineering projects. It has been a learning odyssey, with beneficial outcomes in professional pursuits in the years to come.

References

[1] Julian Kirchherr, Denise Reike, and Marko Hekkert. Conceptualizing the circular economy: An analysis of 114 definitions. *Resources, Conservation and Recycling*, 127:221–232, 12 2017.

[2] What are the best git branching strategies. `https://www.flagship.io/git-branching-strategies/`. Accessed: 2022-03-19.

[3] The 12 principles behind the agile manifesto. `https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/`. Accessed: 2022-06-28.

[4] Agile 101. `https://www.agilealliance.org/agile101/`. Accessed: 2022-06-28.

[5] Morgan R. Clevenger. Leveraging social responsibility for worker engagement : From recruitment to productivity, satisfaction, longevity, and happiness. *The Routledge Companion to Happiness at Work*, pages 273–282, 10 2020.

[6] Daniel W. Greening and Daniel B. Turban. Corporate social performance as a competitive advantage in attracting a quality workforce. *Business & Society*, 39:254–280, 2000.

[7] The 12 principles behind the agile manifesto. `https://ecommerceguide.com/ecommerce-statistics/`. Accessed: 2022-06-28.

[8] E-commerce statistics for individuals - statistics explained. `https://ec.europa.eu/eurostat/statistics-explained/index.php?title=E-commerce_statistics_for_individuals`. Accessed: 2023-07-28.

[9] B2b2c ecommerce definition and comparison. `https://www.bigcommerce.com/articles/b2b-ecommerce/b2b2c-ecommerce/`. Accessed: 2023-07-28.

[10] Leandro Ricardo Sabino, Greiciele Macedo Morais, Valdeci Ferreira, Dos Santos, Carlos, and Alberto Gonçalves. E-commerce: A short history follow-up on possible trends. *Article in International Journal of Business Administration*, 8, 2017.

[11] 2-tier vs. 3-tier application architecture? `https://nitrosphere.com/uncategorized/2-tier-vs-3-tier-application-architecture-could-the-winner-be-2-tier-2/`. Accessed: 2023-07-15.

[12] Ibm | what is three-tier architecture? `https://www.ibm.com/topics/three-tier-architecture`. Accessed: 2023-07-15.

[13] Owast | m9: Reverse engineering. `https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering`. Accessed: 2023-07-16.

[14] What is mach architecture for ecommerce? `https://www.mongodb.com/blog/post/what-mach-architecture-ecommerce/`. Accessed: 2022-07-20.

[15] Smartphones must have common charging port by 2024, e.u. says. https://www.washingtonpost.com/technology/2022/06/08/eu-common-phone-charger-law-usbc-apple/. Accessed: 2022-04-16.

[16] Martin Geissdoerfer, Paulo Savaget, Nancy M.P. Bocken, and Erik Jan Hultink. The circular economy - a new sustainability paradigm? *Journal of Cleaner Production*, 143:757–768, 2 2017.

[17] What the r?! the 9r framework. https://www.malbaproject.com/post/what-the-r-the-9r-framework-and-what-you-should-know-about-it. Accessed: 2022-05-01.

[18] Circular economy definition, importance and benefits. https://www.europarl.europa.eu/news/en/headlines/economy/20151201STO05603/circular-economy-definition-importace-and-benefits. Accessed: 2022-04-05.

[19] User stories: As a [ux designer] i want to [embrace agile] so that [i can make my projects user-centered]. https://www.interaction-design.org/literature/article/user-stories-as-a-ux-designer-i-want-to-embrace-agile-so-that-i-can-make-my-projects-user-centered. Accessed: 2022-05-01.

[20] Lianping Chen, Muhammad Ali Babar, and Bashar Nuseibeh. Characterizing architecturally significant requirements. *IEEE Software*, 30:38–45, 2013.

[21] Usability 101: Introduction to usability. https://www.nngroup.com/articles/usability-101-introduction-to-usability/. Accessed: 2022-06-28.

[22] Jeffrey Zeldman. Taking your talent to the web a guide for the transitioning designer.

[23] C4 - code diagram. https://c4model.com/#CodeDiagram. Accessed: 2023-07-20.

[24] Containers - os level virtualization. https://en.wikipedia.org/wiki/OS-level_virtualization. Accessed: 2022-03-14.

[25] Docker. https://www.docker.com/. Accessed: 2022-03-19.

[26] What is penetration testing. https://www.imperva.com/learn/application-security/penetration-testing/#:~:text=A%20penetration%20test%2C%20also%20known,web%20application%20firewall%20(WAF). Accessed: 2022-07-20.

[27] What is penetration testing. https://www.imperva.com/learn/application-security/black-box-testing/. Accessed: 2022-07-20.

[28] Vlad Mosessohn and Vlad-Gabriel Luca. Quantifying the effects of monitoring software on employee productivity and satisfaction rareș-valentin drĂghici 13 radu-alexandru ioniȚĂ 15.

[29] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 21:2146–2189, 10 2016.

[30] How much time do developers spend actually writing code? https://thenewstack.io/how-much-time-do-developers-spend-actually-writing-code/. Accessed: 2022-03-19.

[31] Integrated development environment. `https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-integrated-development-environment`. Accessed: 2023-07-17.

[32] Gitlab flow. `https://docs.gitlab.com/ee/topics/gitlab_flow.html`. Accessed: 2022-03-14.

[33] What is a container? `https://www.docker.com/resources/what-container/`. Accessed: 2022-03-19.

[34] Overview of docker compose. `https://docs.docker.com/compose/`. Accessed: 2022-03-19.

[35] Rancher overview. `https://rancher.com/docs/rancher/v2.6/en/overview/`. Accessed: 2022-03-19.

[36] Maria Rodriguez and Rajkumar Buyya. Container orchestration with cost-efficient autoscaling in cloud computing environments. *https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-7998-2701-6.ch010*, pages 190–213, 1 1.

Appendices

Appendix A

Security Report

A.1 Submission Module Backoffice and E-commerce Resolution Report

Context

This report is a summary of the vulnerabilities found in the OLX Second Life Submission Module, specifically the Backoffice, and the solutions implemented to fix them.

Schema Bypass

Severity: HIGH

Description: Authorization bypass refers to a vulnerability in a computer system or application where a user can bypass the authorization process and gain access to resources or privileges that they should not be able to access. This can occur due to weaknesses in the system design, coding errors, or the exploitation of security flaws. It was discovered that the Back-Office Submission Module was vulnerable to authorization bypassing by enabling less privileged roles to create and delete higher privileged users. Using a user with the role "financial manager" it was possible to create and delete a user with a higher privileged role "loop manager". It is an authorization schema bypass because the role "loop manager" is the parent role of the "financial manager".

Implemented Solution: The authorization schema was changed to prevent the creation and deletion of users with higher privileges by less privileged users. Specifically, the "Financial Manager" role cannot create or delete users, or change their roles, and has no access to this page, not through the UI or the API.

Enumeration at Forgot Password Page

Severity: HIGH

Description: User enumeration is the process of discovering valid users within an application, by searching for clues in the application that confirm their existence. User enumeration can be performed passively and/or actively. It was discovered that the Back-Office Submission Module did not enforce security mechanisms to prevent user enumeration. The

Forgot your password page located at the above endpoint, allowed user enumeration by disclosing the validity of usernames during attempts for recuperating the user account. The application responded with "1 error prohibited this loop manager from being saved: Email not found" for attempts to get a recuperation code for users that did not exist in the system.

Implemented Solution: The application was updated to prevent user enumeration by not disclosing the validity of usernames during attempts to recuperate the user account. Now it returns the same message for all attempts, regardless of the validity of the username. It was a default misconfiguration of the devise gem.

Outdated Components

Severity: HIGH

Description: A software component is part of a system or application that extends the functionality of the application. Since many software components run with the same privileges as the application itself, any vulnerabilities or flaws in the component can result in a threat to the web application. Applications using components with known vulnerabilities can be susceptible to attacks that target any part of the application stack. It was discovered that the Back-Office Submission Module used vulnerable/outdated components. The application contained vulnerable/outdated versions of Ruby (Ruby 3.0.2) and Bundler (Bundler 2.2.22). The specific outdated versions contain known high-risk vulnerabilities:

Ruby 3.0.2:

- `https://www.cvedetails.com/cve/CVE-2022-28739/`
- `https://www.cvedetails.com/cve/CVE-2022-28738/`

Bundler 2.2.22:

- `https://www.cvedetails.com/cve/CVE-2021-43809/`

Implemented Solution: Ruby and Bundler were updated to versions where the vulnerabilities were fixed, namely 3.0.5 and 2.2.33 respectively.

Weak Lockout Mechanism on Login Page

Severity: HIGH

Description: Enforcing a strong lockout mechanism is a security measure to prevent the takeover of user accounts within an application by malicious third parties. Enforcing rate limits contributes to a strong lockout mechanism, as it limits the number of requests during a time frame to a specific endpoint. The Back-Office Submission Module was found to implement a weak lockout mechanism by setting high rate limits on the login page at the above endpoint. After sending 128 requests consecutively, the server stops accepting requests, although the IP is locked for a short time which does not prevent a malicious attacker from performing a succession of brute force attacks.

Implemented Solution: The application was updated to implement a stronger lockout mechanism on the login page, with a back-off factor. The new rate limit is shown in the table A.1.

Requests	Time (seconds)
5	30
10	90
15	270
20	810
25	2430
30	7290
35	21870

Table A.1: Rate Limit Table

Origin Resource Sharing: Arbitrary Origin Trusted

Severity: HIGH

Description: Cross-origin resource sharing (CORS) is a browser mechanism that enables controlled access to resources located outside of a given domain. Many modern websites use CORS to allow access from subdomains and trusted third parties. Their implementation of CORS may contain mistakes or be overly lenient to ensure that everything works, and this can result in exploitable vulnerabilities. It was discovered that the Back-Office Submission Module allowed arbitrary origins to be trusted to the Admin panel. This vulnerability could be exploited to get sensitive information and redirect them to a malicious web server.

Implemented Solution: CORS configuration was updated to only allow trusted origins, namely the host itself and the host of the Submission Form. Additionally, the Content Security Policy has been configured to only allow those origins.

Persistent Session Cookie

Severity: MEDIUM

Description: A persistent cookie is a file stored on a user's computer that remembers information, settings, preferences, or sign-on credentials that the user has previously saved. The back-office submission module was found to enforce a persistent session cookie. The security analyst was able to test the vulnerability by following the below procedures: The same session cookie should not still work after logging out. When a user logs out of an application, the session should be terminated, and the session cookie should be invalidated.

Implemented Solution:

This was not considered a major security issue. You can only exploit this if you have direct access to the computer, and are actively storing session data.

This website details why it is not a problem if SSL is used: `https://bryanrite.com/ruby-on-rails-cookiestore-security-concerns-lifetime-pass/`

The best and easiest solution is simply to use SSL. Not just on your login forms and actions, but your entire site, or at least any pages where you have sessions turned on. With SSL on, the user will not be able to replay your cookies and the entire attack vector is shut down. Rails 3.1 has SSL enforced by default.

An alternative would be to add extra data (nonce) for the sessions and invalidate them at a specific time (would require storing session data in the database), but it is still tamperable.

Mass Submission of Form

Severity: MEDIUM

Description: The landing page had a form at the above location without protection against automated submission. Using this form an attacker could cause a degradation of service, overloading the server with a massive quantity of messages.

Implemented Solution: The form submission endpoint now has a rate limiting, of 5 posts per minute, per IP address.

Disclosure via user API page

Severity: MEDIUM

Description: Information disclosure, also known as information leakage, occurs when a website unintentionally reveals sensitive information to its users. Depending on the nature of the information disclosed, it can cause big consequences for the applications. It was discovered that the Back-Office Submission Module disclosed information about a user by showing personal information on the API page, which is accessible to the public.

Implemented Solution: Although the API page was only accessible to the original poster, because of the UUID token, which is inherently resistant to enumeration, the personal seller information returned by the API is now censored.

Vulnerable Components

Severity: MEDIUM

Description: A software component is part of a system or application that extends the functionality of the application. Since many software components run with the same privileges as the application itself, any vulnerabilities or flaws in the component can result in a threat to the web application. Applications using components with known vulnerabilities can be susceptible to attacks that target any part of the application stack. It was discovered that the back office e-commerce used vulnerable components. The application contained vulnerable versions of two javascript libraries: underscore.js (underscore.js 1.8.3) and Bootstrap (Bootstrap 4.0.0). The specific versions are known for containing the following vulnerabilities:

underscore.js 1.8.3:

- `https://nvd.nist.gov/vuln/detail/CVE-2021-23358`

Bootstrap 4.0.0:

- `https://nvd.nist.gov/vuln/detail/CVE-2019-8331`
- `https://nvd.nist.gov/vuln/detail/CVE-2018-14041`
- `https://nvd.nist.gov/vuln/detail/CVE-2018-14040`
- `https://nvd.nist.gov/vuln/detail/CVE-2018-14042`
- `https://nvd.nist.gov/vuln/detail/cve-2016-10735`

Implemented Solution:

Solidus update fixed the underscore js. To update the Bootstrap library version a new e-commerce app would have to be developed which is out of the current project scope. The development of a new e-commerce application would be subjected to a specific quote.

Session Cookie without Secure Flag

Severity: LOW

Description: Security headers and cookie attributes are directives that browsers must follow and that are passed along, through the HTTP header response. Including security headers and cookie attributes in server, responses is important as each one provides security against malicious attacks. The Back-Office Submission Module's response headers were found to set the session cookie without the secure flag.

Implemented Solution: The application was updated to set the secure flag on the session cookie in the production environment.

Missing Content Security Policy header

Severity: LOW

Description: The Content Security Policy (CSP) is an HTTP header through which site owners define a set of security rules that the browser must follow when rendering their site. The most common usage is to define a list of approved sources of content that the browser can load. This can be used to mitigate cross-site scripting (XSS) and clickjacking attacks effectively.

Implemented Solution:

With the correct URL and production environment enabled all the recommended policies are set (`https://www.w3.org/TR/CSP2/`) are present.

Missing clickjacking protection

Severity: LOW

Description: A frameable response occurs when one or multiple pages can be used on an iframe on any website. This allows the clickjacking attack to be used. Clickjacking is when an attacker uses a hidden iframe with multiple transparent or opaque layers above it to trick a user into clicking on a button or link on the iframe when they intend to click on the top-level page. Thus, the attacker is "hijacking" clicks meant for the top-level page and routing them to the iframe.

Implemented Solution:

Added 'frame-ancestors 'none'' to the CSP header.

Browser content sniffing allowed

Severity: LOW

Description: The application allows browsers to try to mime-sniff the content type of the responses. This means the browser may try to guess the content type by looking at the response content and render it in the way it was not intended. This behavior may lead to the execution of malicious code, for instance, to explore an XSS vulnerability.

Implemented Solution:

With the correct URL and production environment enabled, the Rails framework automatically adds the X−Content−Type−Options: nosniff header to necessary responses.

A.2 Submission Module Frontend Problem Resolution Report

Context

This report is a summary of the vulnerabilities found in the Submission Module Proposal App, and the solutions implemented to fix them.

Untrusted TLS certificate

Severity: MEDIUM

Description: The certificate sent by the server is not trusted.

Implemented Solution:

This happened because the penetration team used the internal URL. The URL was disabled, and the website was unreachable. `https://secondlife.olx.pt` is to be used instead.

The validity of the certificate can be validated with the following command:

```
openssl s_client -connect secondlife.olx.pt:443 \
-servername secondlife.olx.pt | openssl x509 -text
```

Missing Content Security Policy header

Severity: LOW

Description:

The Content Security Policy (CSP) is an HTTP header through which site owners define a set of security rules the browser must follow when rendering their site. The most common usage is to define a list of approved sources of content that the browser can load. This can be used to mitigate cross-site scripting (XSS) and clickjacking attacks effectively.

Implemented Solution:

No change, the front end uses meta tags to define the Content Security Policy.

Missing clickjacking protection

Severity: LOW

Description: A frameable response occurs when one or multiple pages can be used on an iframe on any website. This allows the clickjacking attack to be used.

Clickjacking is when an attacker uses a hidden iframe with multiple transparent or opaque layers above it to trick a user into clicking on a button or link on the iframe when they intend to click on the top-level page. Thus, the attacker is ”hijacking” clicks meant for the top-level page and routing them to the iframe.

Implemented Solution: Added header ‘X-Frame-Options: deny‘

Certificate without revocation information

Severity: LOW

Description:

A certificate without revocation information cannot be revoked by its owner in case its private key is compromised. Browsers consult the Certificate Revocation List (CRL) or the Online Certificate Status Protocol (OCSP) endpoints that should be present in the certificate, to validate it.

Implemented Solution: As described in the problem Untrusted TLS certificate A.2, the certificate is valid and also has OCSP - URI: `http://r3.o.lencr.org`.

The verification can be validated with the following command:

```
openssl ocsp -issuer chain.pem \
    -cert secondlife.pem \
    -text -url http://r3.o.lencr.org
```

Browser content sniffing allowed

Severity: MEDIUM

Description:

The application allows browsers to try to mime-sniff the content type of the responses. This means the browser may try to guess the content type by looking at the response content and render it in a way it was not intended to. This behavior may lead to the execution of malicious code, for instance, to explore an XSS vulnerability.

Implemented Solution:

The public URL has a ‘X-Content-Type-Options: nosniff‘ header.