UNIVERSIDADE D
COIMBRA

Maria Beatriz Delgado Gomes Santos Vieira

MOBILE MONEY AGENT MANAGEMENT USING BLOCKCHAIN

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Prof. Luís Macedo and Eng. Marisa Martins, and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2022

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Maria Beatriz Delgado Gomes Santos Vieira

Mobile Money Agent Management using Blockchain

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Prof. Luís Macedo and Eng. Marisa Martins, and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2022

Acknowledgements

With the internship concluded, it would be impossible not to thank several people. Who directly or indirectly contributed and helped me in this phase of my professional and personal life.

First of all, I would like to thank WIT Software for the opportunity they gave me to do my internship and the way they welcomed me. To my technical tutor, João Sousa, for all the support provided, constant monitoring, and concern with the work being developed. To my supervisor, Marisa Martins, for always being available to answer any questions I might have. To my business analyst, Jana Lage, for all the advice and support during this internship. To all of them, I am grateful for the transmission of knowledge that was very important for my growth as a professional.

A word of thanks to the teachers of the Department of Informatics Engineering. For the knowledge and values transmitted during the classes. A special thanks to Professor Luís Macedo for his guidance in this report development and the support demonstrated.

A huge thank you to my family, first to my father, who was the person who believed in me the most and was the proudest. Second, to my mother, for being my biggest support and safe haven. And to my two older brothers, who were always there to give me the strength to overcome all obstacles.

And finally, thank my friends, classmates, and childhood friends. For believing in me more than I believed in myself and their constant concern throughout this time.

Abstract

In a world more technological than ever, it is crucial to remember that technology is not only about making our lives easier. People born and residing in developing countries do not have the same ease of access as most people have to the most basic technologies. The leading reasons for this are the lack of signal coverage and lack of literacy. Mobile money services intend to narrow this technological gap, and the purpose is to facilitate financial transactions for those under these conditions. The present document and internship aim to study the integration of blockchain technology in a mobile money service. The motivation is not to implement the mobile money service itself but a new agent rating feature based on blockchain.

Competitor research was carried out to understand how mobile money services are designed. From this study, it was possible to conclude that there are no solutions based on blockchain, which makes this internship unique. Due to privacy issues, the architectural solution is not entirely decentralized. The designed architecture is hybrid by considering a centralized database that stores users' personal and sensitive data. Plus, the information related to the rating feature is stored in a smart contract on the blockchain.

There are some hurdles in this blockchain-based solution. The most critical impediment is that the user must have some experience and knowledge in blockchain to be able to do the digital wallet setup before using the application. For future work, Gas Station Network (GSN) is the solution for this obstacle, removing all perceptions that there are blockchain mechanisms in the application.

Keywords

Mobile Money, Agents, Clients, Mobile Application, Blockchain, Smart Contracts

Resumo

Num mundo cada vez mais tecnológico, é importante relembrar que a tecnologia não deve ser apenas usada para facilitar as nossas vidas. As pessoas nascidas e residentes em países em desenvolvimento não têm a mesma facilidade de acesso às tecnologias mais básicas. Duas razões principais para isso são a falta de cobertura de sinal e o analfabetismo. Os serviços de dinheiro móvel pretendem reduzir esta lacuna tecnológica e o objetivo é facilitar as transações financeiras para estas pessoas. O presente documento e estágio visam estudar a integração da tecnologia blockchain num serviço de dinheiro móvel. A motivação não é implementar o próprio serviço de dinheiro móvel, mas uma nova funcionalidade de classificação de agentes baseada em blockchain.

Para entender melhor como é que os serviços de dinheiro móvel são desenhados, foi realizada uma pesquisa de concorrentes. A partir deste estudo, foi possível concluir que não existem soluções baseadas em blockchain, o que torna este estágio único. Devido a questões de privacidade, a solução arquitetónica não é totalmente descentralizada. A arquitetura projetada é híbrida, pelo que uma base de dados centralizada armazena os dados pessoais e sensíveis dos utilizadores. Além disso, as informações relacionadas à funcionalidade de classificação de agentes, são armazenadas num contrato inteligente na blockchain.

Existem alguns obstáculos nesta solução baseada em blockchain. O impedimento mais crítico é que o utilizador deve ter alguma experiência e conhecimento prévio em blockchain. Isto para poder fazer a configuração da carteira digital antes de utilizar a aplicação. Para trabalho futuro, a Gas Station Network (GSN) é a solução para este obstáculo, removendo todas as percepções de que existem mecanismos de blockchain na aplicação.

Palavras-Chave

Dinheiro Móvel, Agentes, Clientes, Aplicação Móvel, Blockchain, Contrato Inteligente

Contents

Acronyms

AML Anti Money Laundering.

CRUD Create, Read, Update and Delete.

DApp Decentralized Application.

ES Epic Stories.

GDPR General Data Protection Regulation.

GPS Global Positioning System.

GSN Gas Station Network.

JVM Java Virtual Machine.

KPI Key Performance Indicator.

KYC Know Your Customer.

MRZ Machine Readable Zone.

OCR Optical Character Recognition.

UI User Interface.

US User Stories.

UX User Experience.

List of Figures

List of Tables

Chapter 1

Introduction

This document was developed and written for the Curricular Internship for the Master's Degree in Computer Engineering - Software Engineering - at the University of Coimbra for the academic year 2021/2022. Also, this internship proposal was from the WIT Software company. Professor Luís Macedo from the Department of Informatics Engineering, and engineer Marisa Martins from WIT Software, were responsible for guiding this internship.

This first chapter describes the context and motivation of this internship, as well as the objectives to be achieved. In addition, it presents how the document is structured.

1.1 Context

"Mobile money services are being deployed rapidly across emerging markets as a key tool to further the goal of financial inclusion" [1]. WIT Software is a company already experienced and has worked with mobile money systems. Is aware that mobile money services are very successful in developing countries due to the lack of infrastructure to support financial transactions. These services allow to send and receive money, make bill payments, and withdraw money, and the people who make it possible are the agents, who can more easily obtain fiat money.

In the context of this internship, WIT Software presented a project involving the development of a mobile application prototype. This mobile application consists of a Decentralized Application (DApp), which is an app that runs in the blockchain network. The focus is not to implement the mobile money services because the company already has developed services to perform those operations. The key feature of the proposed project is a system to assign a rating to the agents after any service provided to their clients, based on blockchain technology. The purpose of WIT Software with this internship is to understand if a solution based on blockchain would make sense. That's if the risks and costs associated with this type of technology are worth it.

The project itself is already challenging due to the complexity of blockchain tech-

nology, and adding it to a mobile application makes it even more defiant. Blockchain allows for public and unambiguous registration of transactions and enables clients and agents to register only using their mobile devices. The transparency that the blockchain offers increases the trust in the agent.

1.2 Goals

As stated above, the intention is to found this mobile app on the mobile money service theme. However, focusing on the rating given by the clients to the agents. In this application, agents can check their performance according to the ratings, and clients can search for agents corresponding to their needs, by rating, or by distance. The objective of this feature in a mobile money service is to include the clients in the process and allow them to give their opinion about a service they are using. At the same time, agents can re-evaluate their performance and improve their service if clients haven't been given satisfactory ratings.

The main goal is to have a well-developed prototype that can easily demonstrate the explored concepts, and that is under the proposed project. This objective only can be achieved with a well-structured engineering process.

1.3 Document Structure

This document is divided into eight chapters. Each one describes the work carried out under each component. These are:

- The current chapter - Chapter 1 - introduces the project, its context, and the company that proposed it.
- The second chapter - Chapter 2 - is dedicated to the State of The Art. This section describes what mobile money and agents are, competitors' research, and which of the competitors' features can be applied to the project.
- The third chapter - Chapter 3 - describes the proposed solution and which frontend and backend technologies were studied and chosen for the project development.
- The fourth chapter - Chapter 4 - describes the methodology used in this project and the work plan for the 1st and 2nd semesters.
- The fifth chapter - Chapter 5 - intends to describe the project requirements, represented in different ways, such as user stories, use cases, and quality attributes. Furthermore, this chapter describes the project risks and the architecture based on the C4 Model. Finally, a navigation diagram is provided.
- The sixth chapter - Chapter 6 - includes the development component. This chapter describes the project organization and structure, the developed re-

quirements, and security and privacy topics. Lastly, it presents the future work.

- The seventh chapter - Chapter 7 - is dedicated to the testing made to validate the final product.
- The last chapter - Chapter 8 - intends to draw the conclusions of the internship.

Chapter 2

State of The Art

A good research about the theme and the project the intern is proposing to develop is indispensable. The output should give the adequate knowledge to succeed and a critical headstart for what is yet to come.

The purpose of this section is to describe what mobile money is, its usage, and its importance in developing countries. Also, how mobile money is managed and by whom, i.e., Agents. In the following sections, these topics will be approached first, and then an overview of existing similar solutions will be made.

2.1 Mobile Money

So, what is money after all? Fiat money, coins, and money bills? Yes, it is but not only.

"Money is any clearly identifiable object of value that is generally accepted as payment for goods, services and repayment of debts within a market or which functions in a manner similar to the legal tender of a country." [2]

Based on that definition, there's a service of money that allows everyone to send, receive and store money just by using a mobile phone. That is Mobile Money. It is an excellent alternative to fiat money and banks because it facilitates financial transactions, and the users can make transactions anywhere. As WIT Software company works with these services in Africa, the definition of mobile money is lightly different from the one stated above. "While mobile money can include access to e-money, surprisingly most mobile money services are still largely cash-based with service providers acting as intermediary cash agents. This partially explains why evolutions in mobile money are expected to contribute to financial inclusion" [2]. Instead of relying on e-money, the clients need an intermediary to use the mobile money service, like deposit and withdraw cash.

Mobile Money services have greater importance in developing countries than in developed countries. Accordingly to the 2017 statistics report, 45.6 percent of

mobile money services are located in Sub-Saharan Africa [3]. Therefore, in developing countries, this type of service has become very popular because few banks or infrastructure support financial transactions. Another reason is the lack of signal coverage and the lack of literacy to deal with complex money services [4]. As a consequence, people in these places have shown more interest in this service, as it grants them access to money more easily and quickly. Some examples of the most well-known mobile money services that are currently active in Africa are:

- M-Pesa, Safaricom and Vodacom (Vodafone Group Unit)
- Airtel Money, Airtel Payments Bank
- Orange Money, Orange S.A.

To have a chance of succeeding in building these services in these countries, the companies need facilitators that can convert cash, which means someone that can receive and withdraw physical money. Those are called agents.

2.1.1 Agents

In a simple statement, one can define an agent as someone that makes financial transitions easier for customers. Mobile Money services need agents mostly to make operations of cash-in and cash-out, cash-in as an action to credit the user account by the agent, and cash-out as an action to withdraw physical money from the user account [5]. Depending on how the operators implement the Mobile Money service, agents may or may not be the ones that register new customers. Nevertheless, they are the ones that have direct contact with customers. Therefore, they will have a role in helping and teaching how these services operate and how to carry out any transaction. Agents are not exclusive to a mobile money service. They can work for many - figure 2.1 - is an example of how an agent shop is.

Figure 2.1: Mobile Money Agent Shop

And how are these agents incentivized? They receive a commission for every transaction made. In practical terms, the more clients they have and extra transactions performed, the more these agents get at the end of the month. Agents also receive an incentive when performing external services like registering new customers. Furthermore, an authority (central bank or other financial authority) normally regulates these agents.

Why is there a need to manage the agents? Agents acquire new clients. What about recruiting agents? To whom belongs this job? Not everyone can become an agent. They must have some minimum requirements, and the Aggregator plays the role of deciding and evaluating who can become an agent of their network. After this step, agents are trained and onboarded by the respective mobile money service.

Figure 2.2 helps to understand how an agent network works. This agent network is of M-Pesa operated by Safaricom in Kenya, which belongs to the Vodafone Group. M-Pesa means "m-money" in Swahili, started in 2007 in Kenya, and has evolved its business to more than nine countries [6].

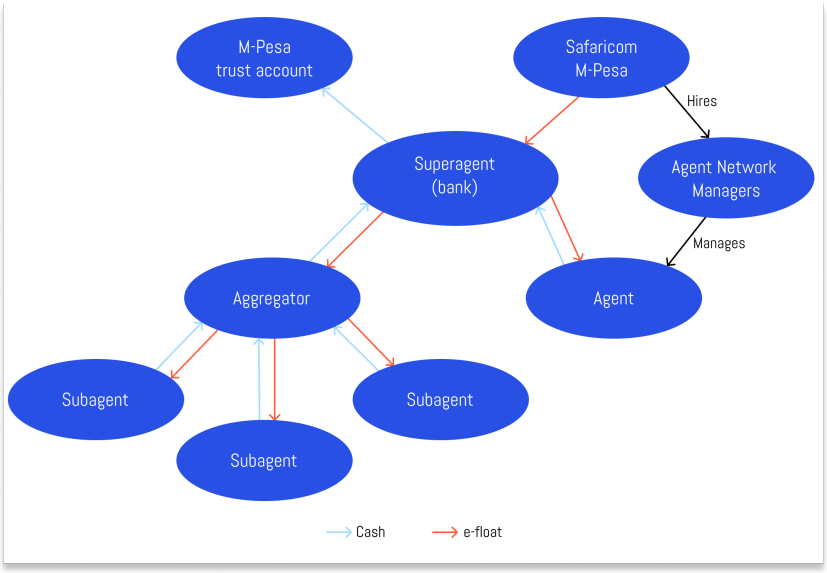

Figure 2.2: Current M-Pesa Agent Network Structure in Kenya (updated) [7]

Starting from the bottom, as stated above, Aggregators select, train, and manage subagents. Aggregators base their assessment on the agents' business location, the size of the customer base, and whether the area is under-supplied by M-Pesa's outlets. Subagents can become agents after three months spent with the Aggregators.

Agents must be able to provide two types of services to their customers: cash-in and cash-out. This implies that they must have enough float and liquidity to provide their customers with these needs, first one is the e-money (electronic money) or physical cash that the agent can access immediately and the second one refers to the sum of the agent's e-money and float balance. To ensure that agents have

quick access to these two needs, Superagents, which are usually banks, were introduced. They deposit funds into an M-Pesa trust account and receive e-float, which they sell to agents and aggregators for a 1% commission.

There are also agent network managers, which are services that Safaricom hires to monitor agents to check their performance and assure that everything is working the way it's supposed to be working. One of M-Pesa's services to manage its agents is Waynbo from Papersoft.

2.2 Competition

This section describes which companies and products have similar solutions for the initial internship problem: "Mobile Money Agent Management using Blockchain". The intern did not find any solution using blockchain, so this research was focused only on how companies manage their agents.

2.2.1 Direct Competitors

Waynbo, Papersoft

The first competitor identified is Waynbo, created by Papersoft. This company was founded in 1999 in Portugal and later expanded to the UK and Mozambique. The purpose was to manage and simplify business processes [8]. It has grown in recent years in new product development for financial inclusion in Africa [9].

Figure 2.3: Waynbo, Papersoft

Waynbo aims to manage a network of agents, offering tools for tracking agents and their performance. Waynbo is a "Built for Africa" solution - meaning it has special care with limitations in the region, such as providing an Offline Mode for zones where the network connectivity is unreliable [10]. This feature can be achieved because the software stores the data in packs, and when the agent recovers the internet signal, all the data is uploaded automatically [11].

First, if the applicant fulfills the needed requirements to become an agent, Waynbo offers an interface for creating agents with different permissions, capturing all the personal data. This is done by reading the document data and scanning MRZ (Machine Readable Zone) or QR codes, making this procedure effortless and easier. Throughout the registration process and to get the new agent onboard, this service provides Bots on digital channels [12].

This service has data reports associated with each agent, monitoring their performance and evaluating their results to keep track of them. Additionally, Waynbo has control tools like GPS location and an e-form for supervisors to assess the agents' skills like a "mystery shopping" program [12]. It is a technique used to evaluate a product or service, posing as a customer.

In this type of service, transparency is a key factor in building a successful Agent Network. The fact that agents can check their performance and that managers can track the agents' transactions supports this transparency concept.

FieldPro, Optimetriks [13]

Figure 2.4: FieldPro, Optimetriks

The second one is FieldPro, developed by Optimetriks. Founded in 2015 with the beginning of activities in East Africa, it is a relatively small company in terms of workers. However, its presence has been evolving and expanding to over 30 countries.

FieldPro is a monitoring tool that allows geo-location tracking, among many other features.

The first thing to do, to ensure the good behavior of the system and so that there are no failures, is to populate a database of all the agents with some information, such as:

- Shop name and an outside/inside picture
- Type of shop
- GPS location
- Shop owner and shop assistant contact details: email address, phone, name
- Unique identifier - agent code or some kind of number that identifies the business to be used as a primary key
- GPS capture (Geo-mapping)

When a new agent joins the network, the operator must add him with all these details, and an attribute is assigned to it that is unique so that the field user cannot create an agent that already exists with that attribute.

Furthermore, agents can have different permissions and restrictions because not all the agents on the same network operate in the same region and not all have the same role. So, FieldPro allows defining a structure to filter what each agent can access, like the activities in his area. Each agent has a unique supervisor that visits him at least once a month. In these visits, the supervisor will check parameters like float and cash availability, branding presence of items such as agent code and street sign, capture some customers feedback, and Compliance with Know Your Customer (KYC) and Anti Money Laundering (AML) standards. This last one ensures that agents ask for the customers' ID when transacting and filling logbooks.

Supervisors have a list of all the agents they have responsibility for. This list allows for measuring an important Key Performance Indicator (KPI) - coverage. Coverage demonstrates the percentage of agents that the supervisor has visited.

The supervisor's productivity can be measured based on the KPIs registered on the app of FieldPro.

MoMoAgent, MTN

The 3rd competitor is MoMoAgent, developed by MTN. This company was created in 1994 and established in South Africa [14]. This mobile operator focuses on mobile and digital changes. They believe that everyone deserves access to better digital devices and better technologies. Essentially they want everyone to get the same chances [15].

Figure 2.5: MoMoAgent, MTN

MoMoAgent is a mobile application that allows agents to manage their financial transactions and allows supervisors to check their agents' performance.

It's necessary to meet some requirements before using MoMoAgent, to become an agent. These requirements include owning a business, having valid documents of identification like Passport, Voter ID Card, National IDs, National Health Insurance Card, SSNIT Biometric Card, and Driver's License [16] [17]. Another requisite is meeting a capital threshold that depends from place to place. So, if these requirements check, the potential agent has to go to an MTN outlet to fill in the application form. Finally, the staff from MTN will analyze and decide if the potential agent can be a MoMo Agent.

As mentioned above, before using the MoMoAgent mobile application, it's necessary to be recognized as an agent by MTN. In this application, the agents can make financial transactions - send and receive money - for their customers, whether they have a bank account or not [18]. MoMoAgent is a simple mobile application to use so that any mobile phone can use this service. In addition, it allows fast transactions because MTN designed this application to run on any network.

2.2.2 Features

With the analysis made of the competitors mentioned before, some features can be extracted.

The features are:

- **Agent registering manually**: entering all necessary information, such as personal and business information, is done manually in the app. Unique identifier for each agent so that there is no replications of agents with that attribute.
- **Agent registering with OCR (Optical Character Recognition) Scanning**: the process of registering the agent speeds up because the data from documents is read automatically and also allows scanning MRZ codes (long

number at the bottom of the passport) or QR codes. The documents that can be used for identification are an ID card, passport, voter's card, military card and driver's license [19]. The information retrieved depends on which document was used, at least the first and last name and the number of the card are retrieved.

- **Different permissions can be defined**: this is very helpful to maintain a certain hierarchy and the restrictions well defined.
- **Remote onboarding for Agents**: bots are used to onboard the new Agents which is easier, faster and more efficient. This eliminates errors from manual validation.
- **Performance Monitoring**: dashboards that allow tracking key metrics of the agents, checking the most important KPIs. Also, gives insights and useful information to certain decisions. An example could be if an agent has not been productive as expected and their supervisor can take action.
- **Control Monitoring - GPS Location**: this makes it possible to know where is the location of any agents' activity.
- **Control Monitoring - Quality Control**: another control tool is the quality control of the agents with the feature of a special e-form that is available for supervisors to evaluate agents' skills.
- **Offline Mode**: this allows the application to keep on monitoring all the data of the agents even on Offline Mode.
- **Direct Assistance to Agents**: each agent has a direct channel on WhatsApp to facilitate helpdesk communications and assistance.
- **Simple Interface**: easy to use and designed to be intuitive.
- **Documentation section**: reserved to each agent, this section is helpful to organize the information into only one place, allows uploading any kind of documents so that the user can find it easily.

Table 2.1 presents the products mentioned before and which features one has.

Features	Products		
	Waynbo	FieldPro	MoMoAgent
1. Agent registering manually	✗	✓	✓
2. Agent registering with OCR Scanning	✓	✗	✗
3. Different permissions can be defined	✓	✓	?
4. Remote onboarding for Agents	✓	✗	✗
5. Performance Monitoring	✓	✓	✗
6. Control Monitoring - GPS Location	✓	✗	✓
7. Control Monitoring - Quality Control	✓	✗	✗
8. Offline Mode	✓	✗	✓
9. Direct Assistance to Agents	✓	✗	✗
10. Simple Interface	✓	✓	✓
11. Documentation section	✗	✓	✗

Table 2.1: Features of Products

2.2.3 Feature Classification

Table 2.2 is a cross-table with the classifications given to each competitor's feature.

The classification was based on the MoSCoW scale [20].

Features	Prototype
1. Agent registering manually	Should Have
2. Agent registering with OCR Scanning	Could Have
3. Different permissions can be defined	Won't Have
4. Remote onboarding for Agents	Could Have
5. Performance Monitoring	Must Have
6. Control Monitoring - GPS Location	Should Have
7. Control Monitoring - Quality Control	Could Have
8. Offline Mode	Must Have
9. Direct Assistance to Agents	Could Have
10. Simple Interface	Must Have
11. Documentation section	Should Have

Table 2.2: Features classification based on the MoSCoW scale

Two different features were stated for registering new agents: Agent registering manually, which is the default mode, and Agent registering with OCR Scanning, granting a simpler option for registration. In this particular case, the focus is not the registration process. If that were the case, the OCR Scanning feature would be a "must-have" as it is not the case, this feature is a "could have".

The third feature, Different permissions can be defined, will probably not be necessary, and the reason for this will be more detailed in the proposed solution Section 3.1. However, the main idea is if the agents' information is public in the various blocks of blockchain, everyone will be able to see every transaction and movement of each agent, which means that there is no need to set different access permissions.

Remote onboarding for agents using bots is a feature that can be classified as a "could have". Again not the focus, but it would be nice to have because it allows removing errors from manual validation.

Having a performance monitoring dashboard is good to have in the application because it allows the agent to get an overview of how good or bad their work as an agent has been. Therefore, this feature is considered a "must have".

There are two features associated with control monitoring, GPS location and quality control. The first will be classified as a "should have" and the second as a "could have" because it has no considerable impact if left out. The GPS location allows to keep track or at least know where the agent is performing his duties.

The eighth feature - Offline Mode - is indisputably a "must have" in any application developed for use in developing countries. This is due to the lack of signal coverage that still exists in some areas.

Having a direct channel to facilitate helpdesk communications and assistance is good but it involves using another application which is a disadvantage. This feature will be classified as a “could have”.

The "Simple Interface" feature can be a little subjective but it essentially means building an application that is intuitive, easy to use so that anyone, even non-literate, can easily learn and use it. Therefore, this feature is also a "must have".

Finally, a documentation section for each agent can be very useful to gather information in a single place, this one is classified as a “should have”.

2.2.4 Conclusion

As stated at the beginning of this chapter, a good state of the art should provide the necessary tools and a solid basis for any project or work. In addition to now having a more realistic idea of what mobile money is and who the agents are, it was also possible to draw some conclusions about what is being done by other companies regarding the management of mobile money agents.

An important fact is that no solution using blockchain was found, which in itself makes the solution proposed by this internship unique.

Chapter 3

Proposed Solution

This chapter aims to describe the proposed solution for this internship. The first section explains how blockchain will be used, and then the backend and frontend technologies are described.

3.1 Blockchain

In its literal interpretation, a blockchain is a chain of several blocks attached (Figure 3.1). Each block contains the data, the block fingerprint represented by the hash, and the previous block hash. Preventing data tampering if someone tries to change the data within a block because, when that happens, the block's hash changes meaning the next block will not have the correct previous hash, making the entire chain invalid.

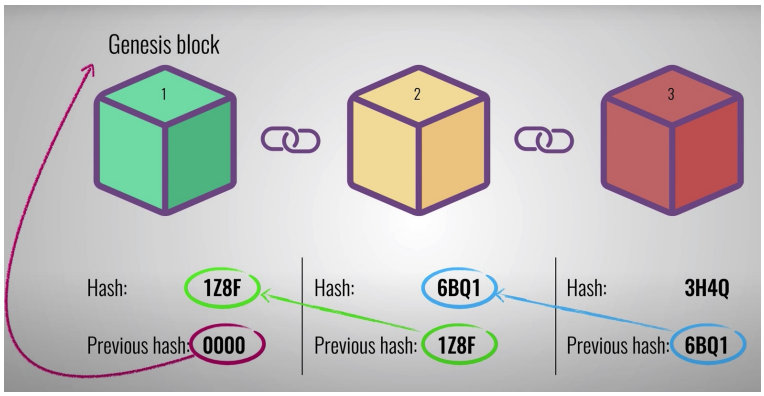

Figure 3.1: Blockchain Explained [21]

The solution is to have a DApp, a decentralized application, which is an application that exists and runs on a blockchain. DApps are open-source, decentralized, and free from control or interference by an authority [22]. The advantages of developing a DApp are that there is no downtime even if one node fails, another node will instantly take over, and censorship-resistant because it's unlikely that

powerful governments or individuals will take over the network [22]. This DApp will support the UI for the backend supported by smart contracts that will write data to the blockchain (Figure 3.2). Smart contracts are like real-life contracts, but digital, that execute predetermined conditions and provide automation, transparency, and immutability [23]. Automation does not need an intermediary for transactions, transparency, and immutability because it avoids fraud. There is a risk with using smart contracts. When deployed to the blockchain, they can never be changed. If there is a bug, it will remain there.

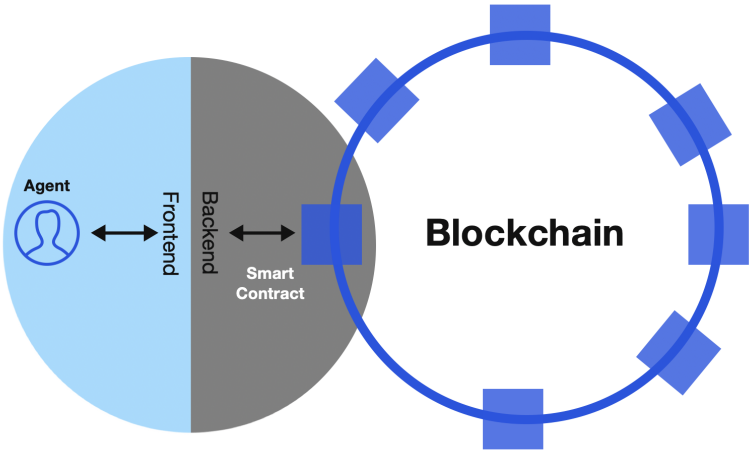

Figure 3.2: Proposed Solution

Clients and agents can register on the blockchain, and all transactions will be associated with each user, ensuring the data is immutable. One of the features of this DApp is the agents' rating. Clients use the agents' services and rate them at the end. In the beginning, new agents may have higher rates which will decrease as they gain reputation. That way, customers can go to an agent with a better reputation or an agent that is new and can charge better taxes. This feature is similar to what happens at the end of a Uber trip. Customers can assign a rating to the driver. In this case, customers can give the agent a rating at the end of the transaction, and agents will improve their rating as they complete transactions.

Blockchain allows transparency of the users' transactions, ensuring the data has not been tampered with, increasing the agents' trust since everything is publicly registered.

3.2 Technologies

In this subsection, all the necessary comparisons will be made to reach a conclusion about the backend and frontend technologies to be used in the project development.

3.2.1 Backend

This subsection makes all the necessary comparisons to conclude the backend and frontend technologies used in the project development.

Choosing the backend for this project means selecting a blockchain platform. Currently, there are several platforms, each with different characteristics and a differentiating factor between them. Among many, there were four different ones compared. For the comparison, the intern considered five criteria:

1. Available Information
    - Documentation structure and quality
    - Available tutorials
2. Ease for a new programmer to get started with this platform
3. Mobile Application Scalability
4. Mobile Application Performance
    - How fast can the application answer requests?
5. Transaction Fee

All four blockchains compared, represented by table 3.1, can support decentralized applications (DApp)s and smart contracts. Performance is the time it takes to add the block to the blockchain, directly related to how many transactions are possible per second.

Criteria	**Blockchain platforms**			
	Ethereum	**Cardano**	**Polkadot**	**Solana**
1. Available Information	✓	✗	✗	✗
2. Easy to learn	✓	✓	✗	✗
3. Scalability	30 TPS	250 TPS	1000 TPS	65 000 TPS
4. Performance	✗	✓	✓	✓
5. Transaction Fee	Prohibitive	Affordable	Low	Low

Table 3.1: Blockchain Platform Comparisons

The first one - Ethereum - launched in 2015 - is best known for supporting DApps and smart contracts. Ethereum has a large amount of documentation and available tutorials that help beginners due to its popularity. Nevertheless, its scalability and performance are not the best and are very low compared to many others.

Ethereum's transaction fee is considered cost-prohibitive because it is so high that most people can not afford it [24].

Secondly, the Cardano platform - launched in 2017 - offers greater scalability than Ethereum and is not difficult to learn. However, its smart contracts were only available in September 2021. Cardano's transaction fee is considered affordable [25].

In third and fourth place, Polkadot and Solana. Both launched in 2020 and therefore have much less documentation and a smaller community that can help with questions and errors. The transaction fee of both these two platforms is considered very low [26][27].

Despite the scalability issue and the prohibitive transaction cost, the Ethereum network keeps about half of all DApps running on the market. On top of that, more than 600 000 active users interact with these DApps regularly [28].

Ethereum would be a good choice as a backend technology. The obstacle is the high transaction fee. However, other Ethereum-based solutions can help increase scalability and therefore reduce cost. These are called Ethereum Layer 2 scaling solutions, and the three most popular will be compared. These are Polygon, Arbitrum, and Optimism - Table 3.2.

The primary focus of any Layer 2 scaling solution is to move most transactions out of the main chain. As a result, transaction speed increases and also reduces gas fees required for transactions [29].

Criteria	**Layer 2 Scaling solutions**		
	Polygon	**Arbitrum**	**Optimism**
1. Available Information	✓	✗	✗
2. Scalability	10 000 TPS	40 000 TPS	2000 TPS
3. Off-chain Protocol	Sidechain	Rollup	Rollup

Table 3.2: Layer 2 Scaling Solution

Polygon, released in 2017 [30], compared to Arbitrum released in May 2021 [31], and Optimism released in July 2021 [32], has more documentation available because it had more time to establish and be used by more people. In the scalability criteria, any one of them [33][34][35] would be good enough.

Sidechains like Polygon and rollups like Optimism and Arbitrum are independent blockchains. They have their own set of blocks and Smart Contract environments. The main difference between these two types of off-chain protocols lies in the bridge contract that allows assets to be moved from the main chain (Ethereum) to another blockchain network. Precisely, it is the trust assumption that protects the funds held by the bridge contract [36].

For a sidechain, the bridge contract does not verify the integrity of the other network. Instead depends on a set of parties to attest to its validity. The sidechain assumes that at least a threshold of parties should have a financial incentive to remain honest and protect the user funds. On the other hand, in rollups, one set of parties is responsible for providing the state of the other network to the bridge contract. The bridge validates and verifies that the other network is not compromised [36]. Rollups can indeed retain Ethereum security but at a cost. Rollups consume more Ethereum resources, increasing transaction costs in a rollup network in comparison to a sidechain.

Polygon uses Proof of Stake validation instead of Ethereum's Proof of Work validation, which allows for increasing transaction speed. Also, it is a sidechain which means that, compared to the rollups Arbitrum and Optimism, Polygon is cheaper. Since May 2021, the number of daily active users of Ethereum has decreased, and Polygon users have increased, figure 3.3. Polygon features all the tools used by developers to create optimized Ethereum instances. Also, this platform provides an improvement in flexibility for developers and Ethereum's security.

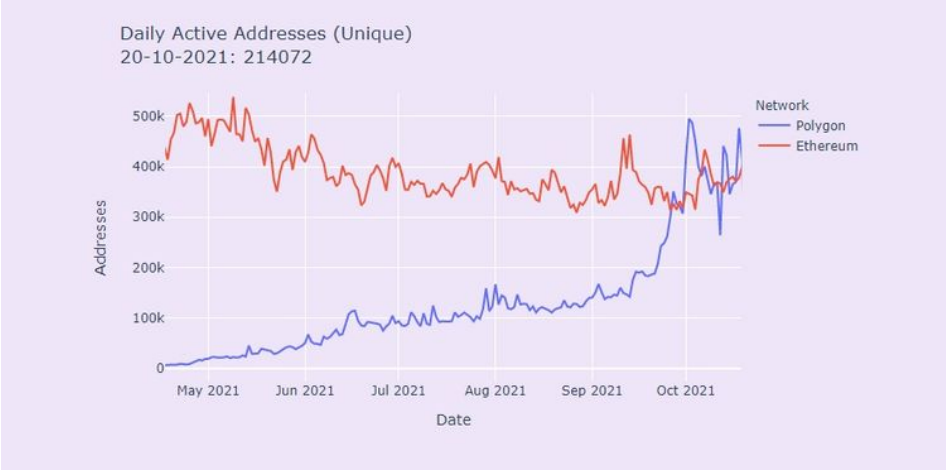

Figure 3.3: Polygon vs Ethereum [37]

The difference between Arbitrum and Optimism is how they resolve a dispute in Layer 2. Both are optimistic rollups, assuming all new additions to the blockchain are valid unless a network participant challenges within a week. Optimism re-executes the transaction disputed in Layer 1 (Ethereum) and checks which part is correct. Arbitrum, on the other hand, continually subdivides the transaction until the disputed information is so small that it can be sent and resolved quickly by Layer 1, thus reducing network congestion relative to Optimism [38].

Between the two rollups, the choice would be Arbitrum because it allows more transactions per second and reduces network congestion. However, transaction cost rollups can be more expensive than sidechains.

As Polygon is already a well-established and cheaper off-chain protocol and also one of the technologies used by WIT Software, therefore it was chosen for this project.

3.2.2 Frontend

This internship aims to develop a mobile app. This section compares four languages to each other to choose the best one, balancing the advantages and disadvantages. Among the possible options, the languages used by WIT Software and the most popular ones will be compared. They are Java, Kotlin, React Native, and Flutter (table 3.3).

Criteria	Frontend Technologies			
	Java (Android)	**Kotlin**	**React Native**	**Flutter**
1. Release	Oracle	2016, JetBrains	2015, Facebook	2017, Google
2. Available Documentation	✓	✓	✓	✗
3. Easy to learn	✓	✓	✓	✓
4. Support	Android	Cross-Platform	Cross-Platform	Cross-Platform
5. Ideal for	Intermediate	Beginners	Intermediate	Intermediate

Table 3.3: Frontend Platform Comparisons

Firstly, comparing the first two - Java and Kotlin - Java was the first and official language for Android development but has now been replaced by Kotlin. Java has many tutorials and documentation available, as well as an online community for support in case of problems. However, for a beginner, Java contains complex topics like constructors, null pointer exceptions, and concurrency. On the other hand, the official stable version of Kotlin, released in 2016 by JetBrains, is much simpler compared to Java for Android App Development [39]. It supports interoperability with Java, runs on the JVM (Java Virtual Machine), and removes Java features such as null pointer exceptions. It is also more concise than Java due to its type interference and reduces the amount of boilerplate code. In short, Kotlin is a Java alternative with newer, more modern features, an attractive layout, and more readable code. On the contrary, Java is faster and more mature [40][39].

Like Kotlin, the third and fourth platforms can build apps for Android and IOS. React Native was launched in 2015 by Facebook and Flutter in 2017 by Google. React Native and Flutter are the most popular technologies for cross-platform software development as they allow only one code base for iOS and Android devices. So apps can be launched faster and on a smaller budget [41]. Both have the "fast refresh" feature, which means code updates automatically as it is written, which means fast coding [42]. Flutter uses the Dart Programming Language, and React Native uses JavaScript. React Native is more established and has a larger developer community. In terms of programming languages, Dart isn't as widely used as JavaScript [42]. However, only if the developer has experience in web or mobile development and has already used popular React solutions, can they easily work with React Native.

Despite the advantages of these two cross-platforms fast coding, and cost savings of developing just a single code for iOS and Android, they are harder to learn. Due to the lack of experience in the chosen backend technology, the choice of the frontend technology has to pay off. This means choosing a platform that is easier to learn and allows the development of a good application. The choice is between Java and Kotlin. Even though the Kotlin community is still young and the learning resources are more limited than Java [39], it is used for Android Applications Development on WIT Software. For these reasons, the chosen language was Kotlin.

Chapter 4

Methodology and Planning

It is important to emphasize that, in the 1st semester, there was no adoption of a specific methodology, as the internship was not full-time, and the activities performed did not require constant monitoring.

This chapter describes the methodology adopted for the 2nd semester of this project and the planning carried out for the first and second semesters. Besides, the analysis of what happened and was planned in the two semesters is done.

4.1 Methodology

Scrum is the method used, and it is an Agile one. Agile methodologies aim to deliver a good product, incremental and frequent delivery of functionalities, enable constant customer feedback, and make it easier to correct mistakes [43].

Scrum is iterative and works with Sprints, which means splitting the work into iterations or cycles. (typically two or three weeks). In these sprints, the development team develops the tasks of the Sprint Backlog. The Product Backlog contains the needed tasks to be developed for the project. The product owner manages and prioritizes the artifact. At the beginning of each sprint, the team decides which tasks and how many will be in the Sprint Backlog.

4.1.1 Roles

There are three roles in the Scrum Methodology:

- **Product Owner:** representative of stakeholders and customers who use the software, has a good perception of the product to be developed. Its principal responsibility is to prioritize and verify the requirements contained in the Product Backlog.
- **Scrum Master:** is the person who leads the team, ensuring that the team has no problems and securing the project's success.

- **Development Team:** the ones responsible for developing and testing the product's backlog functionalities.

In this project, Marisa Martins, the advisor to the intern at WIT Software, portrays the Product Owner role, João Sousa, the technical tutor of the intern at WIT Software, the Scrum Master role, and the development team is composed only of the intern.

4.1.2 Ceremonies

Figure 4.1 represents the Scrum Lifecycle. The Scrum Lifecycle has ceremonies such as:

- Daily Scrum
- Sprint
- Sprint Planning
- Sprint Review

As previously mentioned, sprints are iterations where the features described in the Sprint backlog should be done. Sprint planning is the meeting where the team decides which tasks will be in the Sprint Backlog. The sprint review is a meeting to review whether the team completed all the issues during the sprint and if there are any that need to be carried over to the next sprint. Finally, the Daily Scrum is a 15-minute event for every role of Scrum to improve communication, check progress, identify issues and promote quick decision-making [44]. Also, if there are blocking issues that no one on the development team can fix, the Scrum Master should find someone to help or even the Product Owner if the issue is related to the requirement being defined incorrectly.

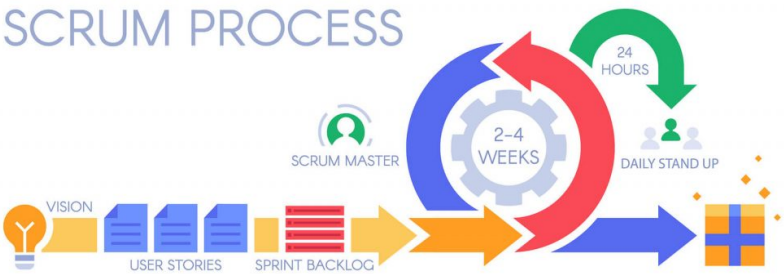

Figure 4.1: Scrum Lifecycle [43]

4.2 Planning

For each semester, a plan is presented and structured to determine which tasks need to be done and how long each one will take.

4.2.1 1st Semester

The objectives of the first semester were to understand the proposed problem, research competitors and their solutions, and start preparing the project development for the second semester.

The Gantt Chart represented by figure 4.2 is the planning drawn for the first semester. All Gantt Charts were developed using the teamgantt tool [45].

When the intermediate report delivery date changed due to the Covid-19 pandemic, this plan suffered some changes.

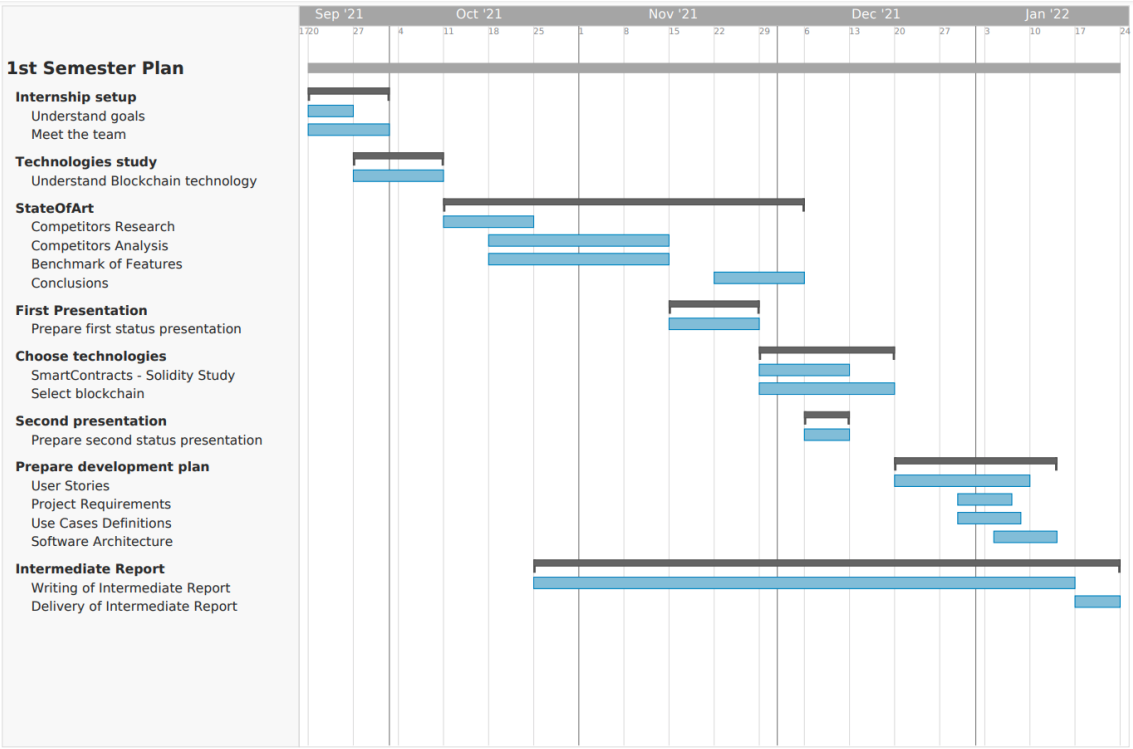

Figure 4.2: 1st Semester Planning

To compare the planned timeline with what actually happened, another Gantt Chart was developed, figure A.1 in appendix A.

In general, the deviations that occurred were always delays. Firstly, the holdbacks of the state of art module are justified, as there were few competitors, and it was hard to find them and extract their features as there was not much information available.

Secondly, the delays related to the "Choose Technologies" module are justified by the lack of experience in these technologies.

Finally, there were delays in the development plan module due to needing a team meeting to discuss these matters.

4.2.2 2nd Semester

The 2nd semester is the period in which the project is developed. There is a need for rigorous planning to achieve success. The semester was divided into two-week periods (sprints), and the Gantt Chart represents the 2nd-semester planning, figure 4.3.

At the end of each sprint, the team gathered to do a sprint review and retrospective. In this meeting, the team evaluated what went well, what did not go so well, and what we could do to improve. This analysis was beneficial for the next sprints to improve the team performance.

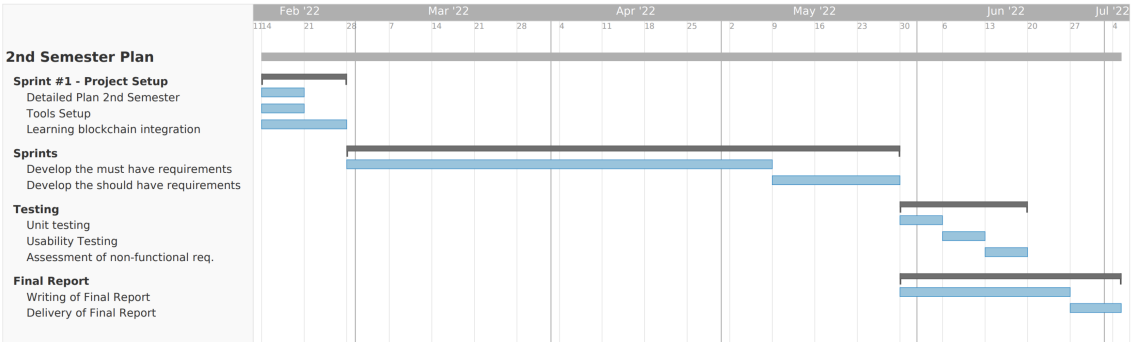

Figure 4.3: 2nd Semester Planning

As cited above, the 2nd semester was divided into sprints - 10 sprints. At the beginning of each sprint, the team used planning poker to estimate the product backlog tasks of that sprint. The team reached a consensus on the story points attributed. These points could be:

- **1:** 2 to 4 hours
- **2:** 4 to 8 hours
- **3:** 1 to 2 days
- **5:** 3 to 4 days
- **8:** 5 to 6 days

After the sprint review and before the retrospective, the team evaluated the burn-down chart. This chart's purpose is to show the amount of work the team completed in the sprint and the total work remaining. Figure 4.4 represents a burn-down chart of sprint #4.

The red line shows how much work remains in the sprint. The grey line shows the ideal progress rate. The illustrated sprint went well, and all tasks were concluded on time.

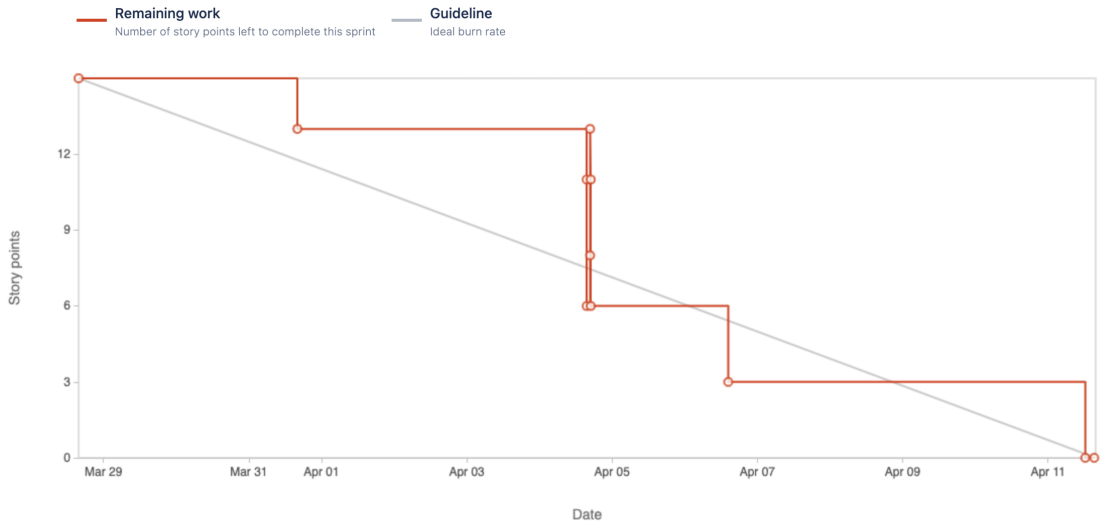

Figure 4.4: Burndown Chart - Sprint #4

In addition, figures 4.5 and 4.6 represent the velocity report of sprints #3 to #9. This report gives the average amount of work a scrum team completes during a sprint, measured in story points.

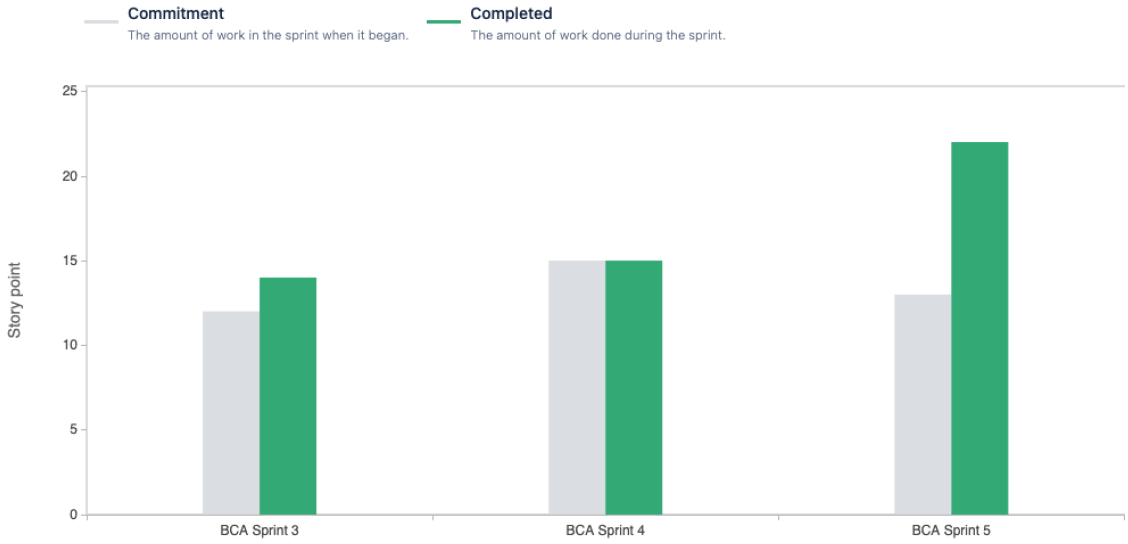

Figure 4.5: Velocity Report Charts - Sprint #3 to #5

The vertical axis displays the estimated story points. The grey line (commitment) is the total story points estimated for the sprint issues. The green line (completed) is the actual story points completed at the end of the sprint.

In general, the intern completed the story points committed. In some cases, there were even more points completed than expected. Only two sprints (#6 and #9) could not be completed.

Figure 4.6: Velocity Report Charts - Sprint #6 to #9

To compare the 2nd semester planned timeline with what actually happened, another Gantt Chart was developed, figure 4.7.

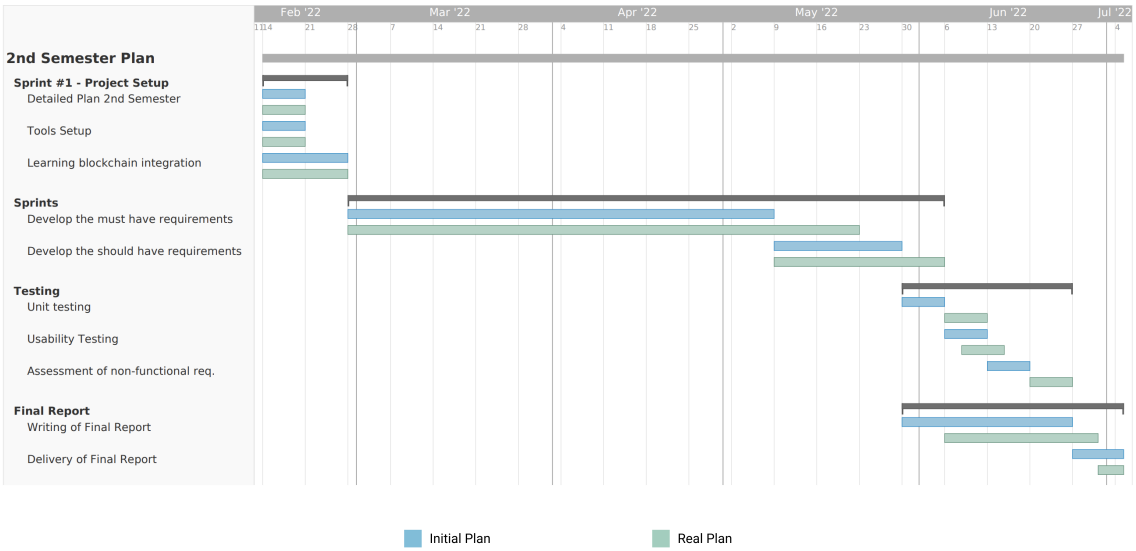

Figure 4.7: Comparison between planned and actual timeline

As it happened in the first semester, in the second, the deviations that occurred were always delays. The must-have requirements took one more sprint than they should, which caused a chain reaction. It delayed the should-have requirements and the final testings.

During the various sprints, it was possible to understand the number of story points the intern could complete due to a better understanding of the project complexity. It was possible to perceive the pace of work of the intern, and the graph analysis contributed to this progress.

The delay caused by the must-have requirements was due to some blockers not being dependent on the intern. Additionally, the intern's inexperience in the technologies also contributed to this.

Chapter 5

System Description

In this chapter, the first objective is to define the requirements, describe a software feature from the user point of view - user stories - and then extract functional requirements from these stories.

Next is to define the use cases based on user stories and extracted requirements and finally describe four non-functional requirements that were considered essential.

The following is to specify a list of risks for this project and ways to mitigate them. As the project develops, the intention is to update the risks list.

Finally, the software architecture and navigation diagram are defined.

As commonly in software engineering projects, there was a need to update the user stories and, consequently, initial requirements and use cases. User stories and added requirements will be represented with a (*) in front of them.

5.1 User Stories

The purpose of this section is to capture a simplified description of a requirement [46]. This Agile software development tool allows describing the type of user, what they want and why.

The user story structure is:

As a **[type of user]**, I want **[goal]** so that **[reason]**.

User Stories (US) were grouped into Epic Stories (ES), which means epic stories are larger user stories. To summarize, user stories are grouped by module.

5.1.1 Agents

1. **Authentication**
   ES-1: As an agent, I want to ensure that the application is only used by authenticated agents so that all the functionality can only be accessed by authenticated users.

   - US-1: Login
     - As an agent, I want to login into the platform, so that I can use all the functionalities.
   - US-2: Logout
     - As an agent, I want to log out of the platform, so that nobody that's not me can access my account.
   - US-3: Registration
     - As an agent, I want to be able to register, so that I can start my activity as a mobile money agent.
       * Profile Picture
       * First and Last Name
       * Email
       * Phone Number
       * Location
   - US-4: Cryptographic wallet setup *
     - As an agent, I want to be able to associate and configure my Metamask wallet, so that I can use blockchain features like registration and assign a rating to an agent.

2. **Manage personal data**
   ES-2: As an agent, I want to be able to manage my personal data so that I can change it whenever I want.

   - US-5: Change Password
     - As an agent, I want to be able to change my password, so that I can use the platform securely.
   - US-6: Change personal data
     - As an agent, I want to change my personal data, so that it is always correct.
       * Profile Picture
       * First and Last Name
       * Email
       * Phone Number
       * Location
   - US-7: Delete account *
     - As an agent, I want to be able to delete my account, so that I can use erase all my data.

- US-8: Switch roles *
  - As an agent, I want to be able to switch to my client account, so that I can use clients' features.

3. **Main Dashboard**
   ES-3: As an agent, I want to have all the functionality in one place, so that I can access it quickly.

   - US-9: Rating
     - As an agent, I want to be able to see how my agent rating going so that I can keep up with the good work or improve.
   - US-10: Performance Dashboard
     - As an agent, I want to be able to see my rating performance by week, so that I can monitor my activity.
   - US-11: Check the balance
     - As an agent, I want to be able to see my balance, so that I can manage it.
   - US-12: View recent transactions *
     - As an agent, I want to be able to see my most recent transactions, so that I can keep track of them.
       * When
       * How Much
       * Whom

4. **Offline Mode**
   ES-4: As an agent, I want to be able to perform offline operations, so that I can stay up to date.

   - US-13: Read Operations
     - As an agent, I want to be able to view my personal and performance information even in offline mode, so that I can stay up to date.

5.1.2 Clients

1. **Authentication**
   ES-5: As a client, I want to ensure that the application is only used by authenticated clients so that all the functionality can only be accessed by authenticated users.

   - US-14: Login
     - As a client, I want to login into the platform, so that I can use all the functionalities.

- US-15: Logout
  - As a client, I want to log out of the platform, so that nobody that's not me can access my account.
- US-16: Registration
  - As a client, I want to be able to register, so that I can start my activity as a mobile money client.
    - Profile Picture
    - First and Last Name
    - Email
    - Phone Number
- US-17: Cryptographic wallet setup *
  - As a client, I want to be able to associate and configure my Metamask wallet, so that I can use blockchain features like registration and assign a rating to an agent.

2. **Manage personal data**
   ES-6: As a client, I want to be able to manage my personal data so that I can change it whenever I want.

   - US-18: Change Password
     - As a client, I want to be able to change my password, so that I can use the platform securely.
   - US-19: Change personal data
     - As a client, I want to change my personal data, so that it is always correct.
       - Profile Picture
       - First and Last Name
       - Email
       - Phone Number
   - US-20: Delete account *
     - As a client, I want to be able to delete my account, so that I can use erase all my data.
   - US-21: Register as an agent *
     - As a client, I want to be able to register as an agent, so that I can start my business.
   - US-22: Switch roles *
     - As a client, I want to be able to switch to my agent account, so that I can use agents' features.

3. **Main Dashboard**
   ES-7: As a client, I want to have all the functionality in one place, so that I can access it quickly.

- US-23: View rating history
  - As a client, I want to be able to see my rating history so that I can keep track of it.
- US-24: Check the balance *
  - As a client, I want to be able to see my balance, so that I can manage it.
- US-25: View recent transactions *
  - As a client, I want to be able to see my most recent transactions, so that I can keep track of them.
    - When
    - How Much
    - Whom

ES-8: As a client, I want to be able to search agents so that I can give them a rating.

- US-26: Search agent by location
  - As a client, I want to be able to search agents by location so that I can go to one near me.
- US-27: Search agent by rating
  - As a client, I want to be able to search agents by rating so that I can choose the best one.
- US-28: Search agent by name *
  - As a client, I want to be able to search agents by name so that I can go to one I know.
- US-29: Assign a rating to a specific agent
  - As a client, I want to be able to give a rating to the agent, so that I can give him a review.

4. **Offline Mode**
   ES-9: As a client, I want to be able to perform offline operations, so that I can stay up to date.

   - US-30: Read Operations
     - As a client, I want to be able to view my personal and transaction history even in offline mode, so that I can stay up to date.

5.2 Functional Requirements

Based on the previous section, the functional requirements were extracted. Subsequently, they were evaluated based on the MoSCoW scale [20].

The first table, 5.1, represents common requirements for agents and clients. The second is the agents' requirements - 5.2 - and the third table represents the clients' requirements, 5.3.

Module	Features	Prototype
Authentication and Manage personal data	1. Login	**Must Have**
	2. Logout	**Must Have**
	3. Registration	**Should Have**
	4. Cryptographic wallet setup *	**Must Have**
	5. Change Password	**Should Have**
	6. Change Personal Data	**Should Have**
	7. Delete account *	**Must Have**
	8. Switch roles *	**Must Have**
	9. Register as an agent (Client only) *	**Must Have**
	10. Offline Mode	**Must Have**

Table 5.1: Common requirements

Module	Features	Prototype
Main Dashboard	11. View Rating	**Must Have**
	12. View performance dashboard	**Should Have**
	13. Check Balance	**Must Have**
	14. View recent transactions *	**Should Have**

Table 5.2: Agents' requirements

Module	Features	Prototype
Main Dashboard	15. View Rating History	**Must Have**
	16. Check Balance *	**Must Have**
	17. View recent transactions *	**Should Have**
	18. Search agent by location	**Must Have**
	19. Search agent by rating	**Must Have**
	20. Search agent by name *	**Must Have**
	21. Assign a rating to a specific agent	**Must Have**

Table 5.3: Clients' requirements

5.3 Use Cases

So, based on the user stories and requirements extracted in the two previous sections, three use cases were designed. There are two different users, one is the agent and the other is the client.

5.3.1 Authentication

In the following figure, 5.1, it is represented that for users to be able to access the features they must register, log in and associate their Metamask Wallet.

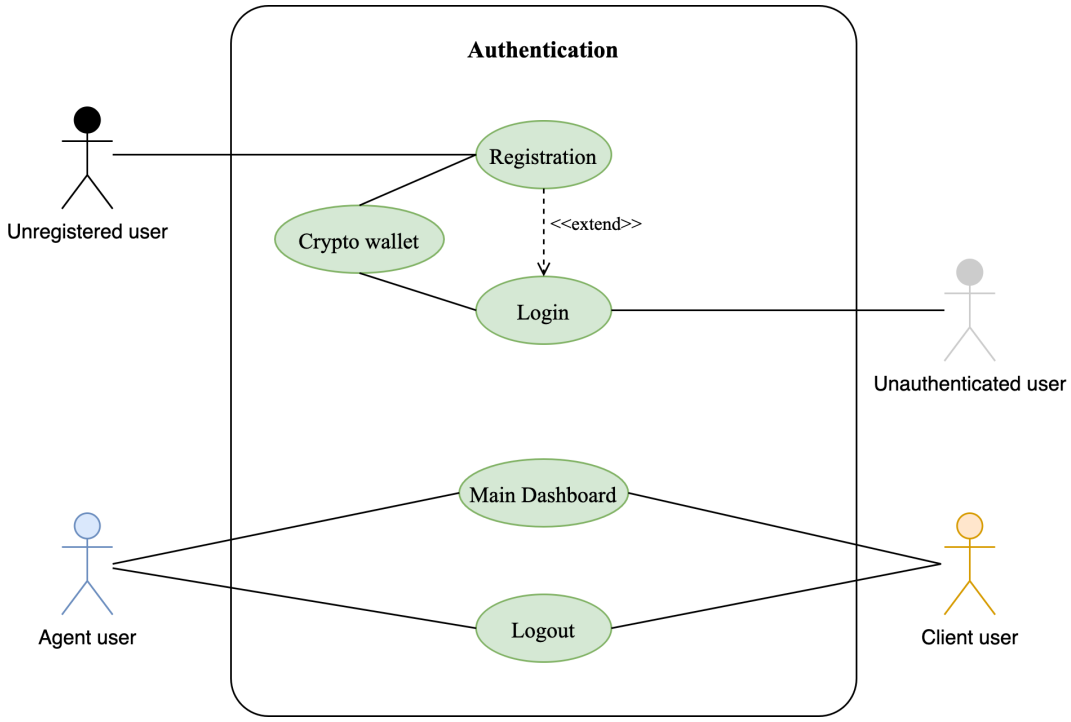

Figure 5.1: Use Case - Authentication

5.3.2 Agent user

This use case, represented by the figure 5.2, illustrates the functionalities that the agent user can have access to.

5.3.3 Client user

The last use case, represented by the figure 5.3, illustrates the functionalities that the client user can have access to.

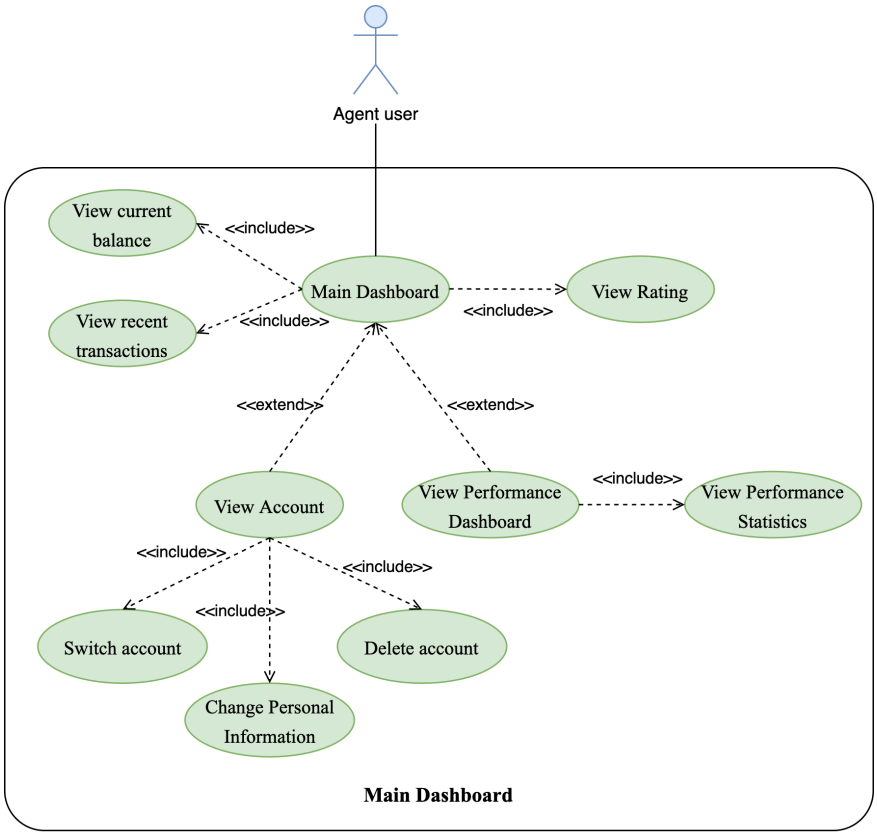

Figure 5.2: Use Case - Agent Dashboard

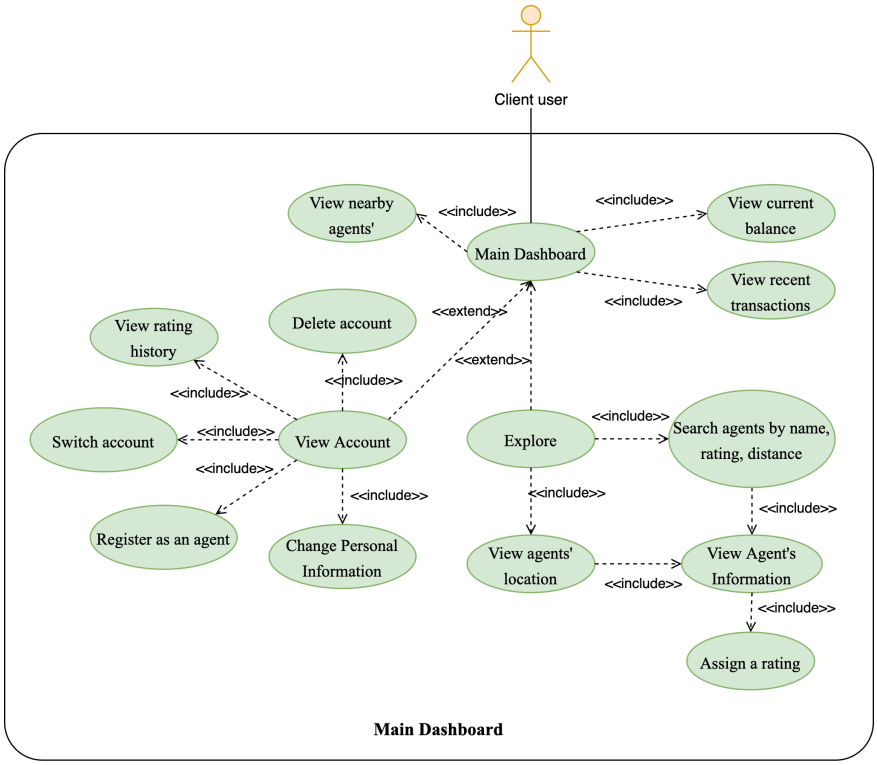

Figure 5.3: Use Case - Client Dashboard

5.4 Non-Functional Requirements

In this section, the application's non-functional requirements are defined, also known as quality attributes. These are indicators of how well the system satisfies stakeholder needs. To well-define a quality attribute, it is helpful to write a quality attribute scenario.

A quality attribute scenario is composed of six factors, as portraited by figure 5.4.

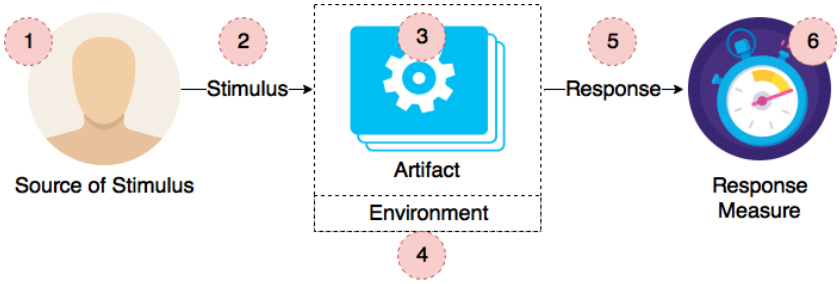

Figure 5.4: Quality Attribute Scenario [47]

1. **Source of Stimulus:** this is an entity that can create a stimulus
2. **Stimulus:** is the action that triggers a response in a system.
3. **Artifact:** refers to which artifact receives the stimulus. It can be the whole system or just a part of it.
4. **Environment:** refers to what kind of environment the stimulus occurs. It can be working under normal conditions, high latency, or any other state.
5. **Response:** this is the response of the stimulus.
6. **Response Measure:** is used to test whether the response was well implemented.

Four quality attributes were defined:

5.4.1 Security

Scenario: Someone trying to access disallowed data

1. Attacker/malicious user
2. Someone tries to access data they don't have permission to
3. Mobile app
4. Protected System
5. The application must ensure that only authorized users can access it and block access to those who are not.
6. After the fifth attempt, access is blocked.

5.4.2 Learnability

Scenario: A new user should easily learn how the app works

1. User
2. The new user registers and does not understand how the application works.
3. Mobile app
4. Normal Operation
5. The application should show tips to the user as they explore for the first time.
6. Given two scenarios, where no tips are provided to the user and another where tips are present, the average number of clicks for the same operations in each scenario can be compared.

5.4.3 Usability

Scenario: A user must use the app with ease

1. User
2. A user wants to use the mobile app to perform a task.
3. Mobile app
4. Normal Operation
5. The application must be built in such a way that the user can easily reach and perform any task.
6. The user must achieve this in a maximum of 3 clicks.

5.4.4 Availability

Scenario: The user performs a task without signal coverage

1. User
2. The user wants to perform a certain task but has no signal coverage.
3. Mobile app
4. Normal Operation
5. The requested task will be performed by accessing the stored data.
6. This should not take longer than 4 seconds.

5.5 Risks

Identifying the risks to a project during its development helps reduce future impacts that can be catastrophic to achieving success. After this identification, a mitigation plan is defined.

This section enumerates a list of risks that can occur in this project. Each risk is characterized by an ID, a description, impact, probability, consequences, and a mitigation plan. These risks are:

- **ID_1:** Inexperience with blockchain technology
- **ID_2:** Inexperience with the chosen smart contracts language - Solidity
- **ID_3:** Low experience in mobile app development
- **ID_4:** Deployment of smart contracts with bugs

ID	Impact	Probability	Consequences	Mitigation Plan
ID_1	High	High	Initial tasks will take longer.	Ask the advisor or other WIT Software members for help and learn from tutorials.
ID_2	High	High	Initial tasks will take longer.	Ask the advisor or other WIT Software members for help and learn from tutorials.
ID_3	Medium	Low	Initial tasks will take longer.	Before getting started, learn from tutorials and read the documentation.
ID_4	High	Medium	The smart contract cannot be fixed.	More than one person must review the smart contract before deployment.

Table 5.4: Risks

During the development, these risks will be evaluated to understand if they occurred and if the mitigation plan worked.

5.6 Software Architecture

The study and description of the requirements enabled the development of the software architecture, which followed the C4 Model. The choice was based on the previous contact and the ease and intuitiveness of building and understanding this tool.

As with many other software engineering projects, there was a need to update the initial architecture, which happens when we get a better understanding of the project as the development begins. All the C4 Model levels were revised, except for the last one. This one is not mandatory and requires too much detailed information.

5.6.1 Architecture

As already mentioned, the software architecture was developed based on the C4 Model. So, the architecture was divided into four levels.

1. Context Level
    - This level is an overview of the software, who the stakeholders are, and what other systems will be integrated.
2. Containers Level
    - The container level represents which systems, databases, or applications the software system will use.
3. Components Level
    - At the third level of the C4 Model, the container is decomposed into components, which are mostly abstractions of the codebase.
4. Code Level
    - The last level of the C4 Model requires a lot of detail to show how the code of a single component works.

The following subsections represent the levels of the C4 Model at which the project architecture is designed.

5.6.2 Context Level

At this level - figure 5.5, it is visible that there are two types of users, a client can be an agent, and an agent is always a client. In addition, the application interacts with five external software systems:

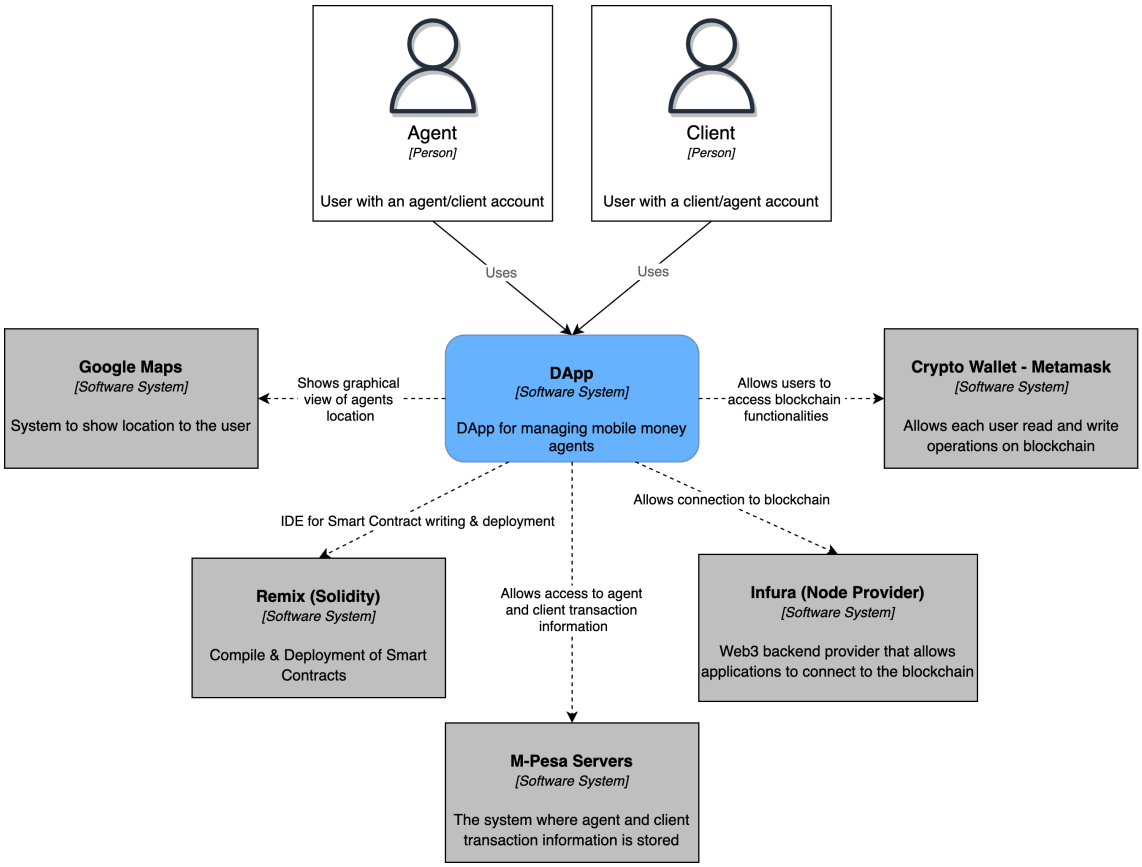

Figure 5.5: Context Level

1. **Google Maps** - is used to show the agents' location to the clients.
2. **Remix** - is an IDE for writing, compiling, and deploying smart contracts.
3. **M-Pesa Servers** - are used to obtain each user's transaction information.
4. **Infura** - is a web3 backend provider that allows applications to connect to the desired blockchain.
5. The last one, the **cryptographic wallet (Metamask)** - is used to perform operations on the blockchain by each user. To achieve this, they need to set up a wallet.

5.6.3 Containers Level

In the second level - figure 5.6 - the software system is decomposed and divided into five components:

1. **Mobile Application** - is developed in the Kotlin language.
2. **Smart contract** - which allows to read and write data in the blockchain and connects the mobile app to the blockchain by the node provider, Infura.
3. **Blockchain** - the chosen one was Polygon.

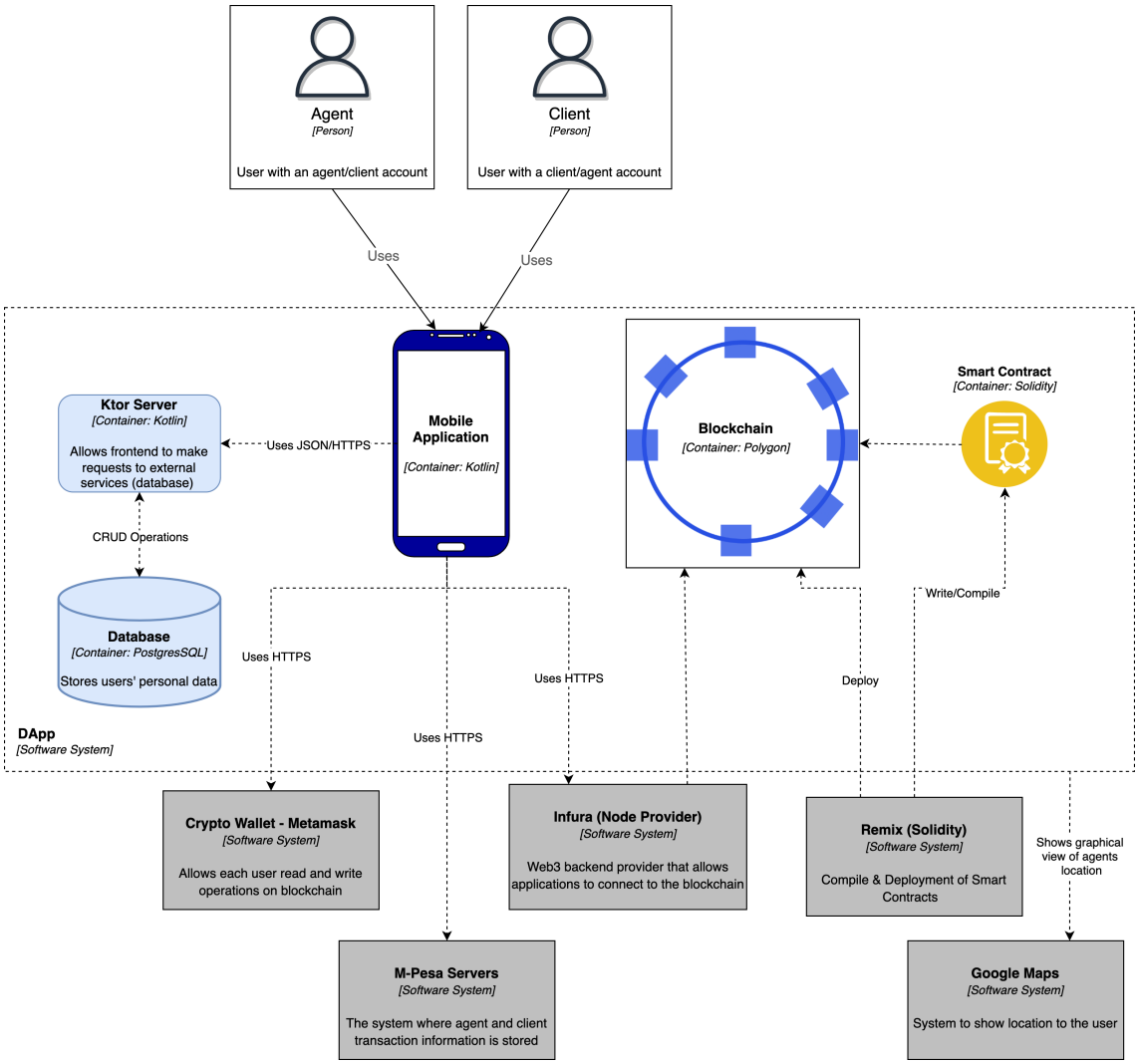

Figure 5.6: Container Level

4. **Ktor Server** - this server, written in kotlin, allows the application frontend to make requests to the database.

5. **Database** - the PostgresSQL database stores all the personal users' data.

5.6.4 Components Level

To visually facilitate the understanding of the third level, it has been divided into two: frontend and backend.

Frontend

The two diagrams below, figures 5.7 and 5.8, represent the application's visual module. In the normal development of Kotlin and Android, fragments are used for each screen of the mobile app. And for each fragment, there is a Kotlin controller that implements the features.

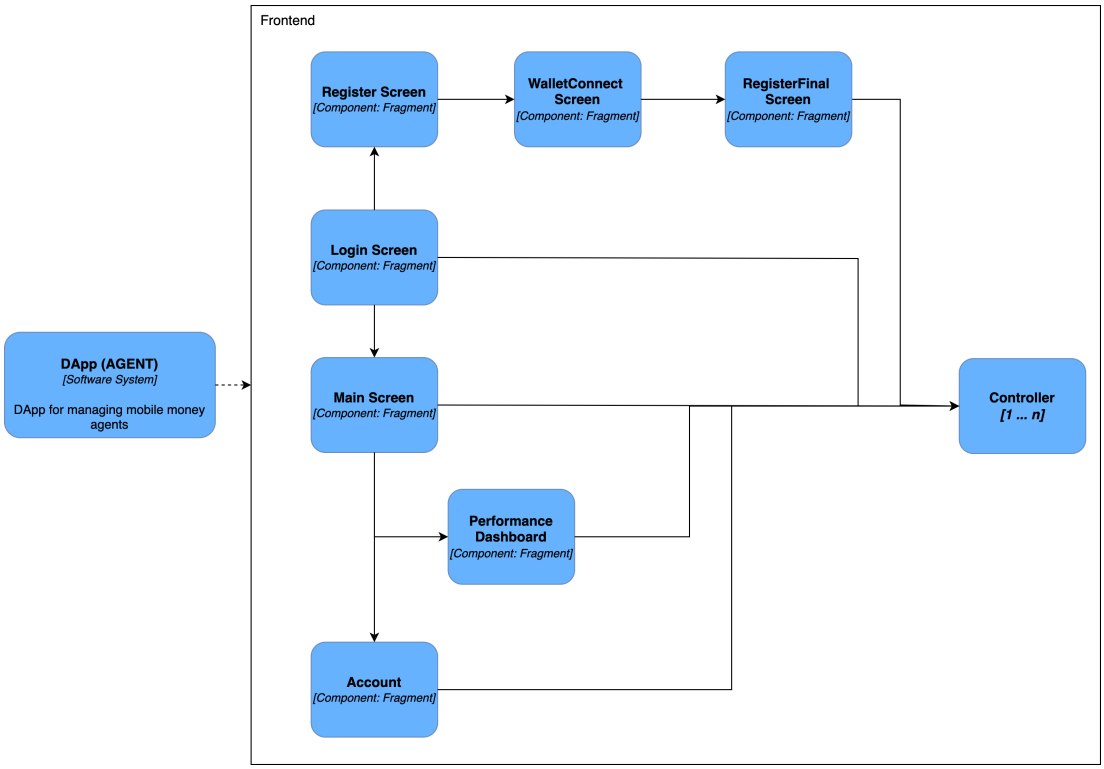

Figure 5.7: Component Level - Frontend (Agent)

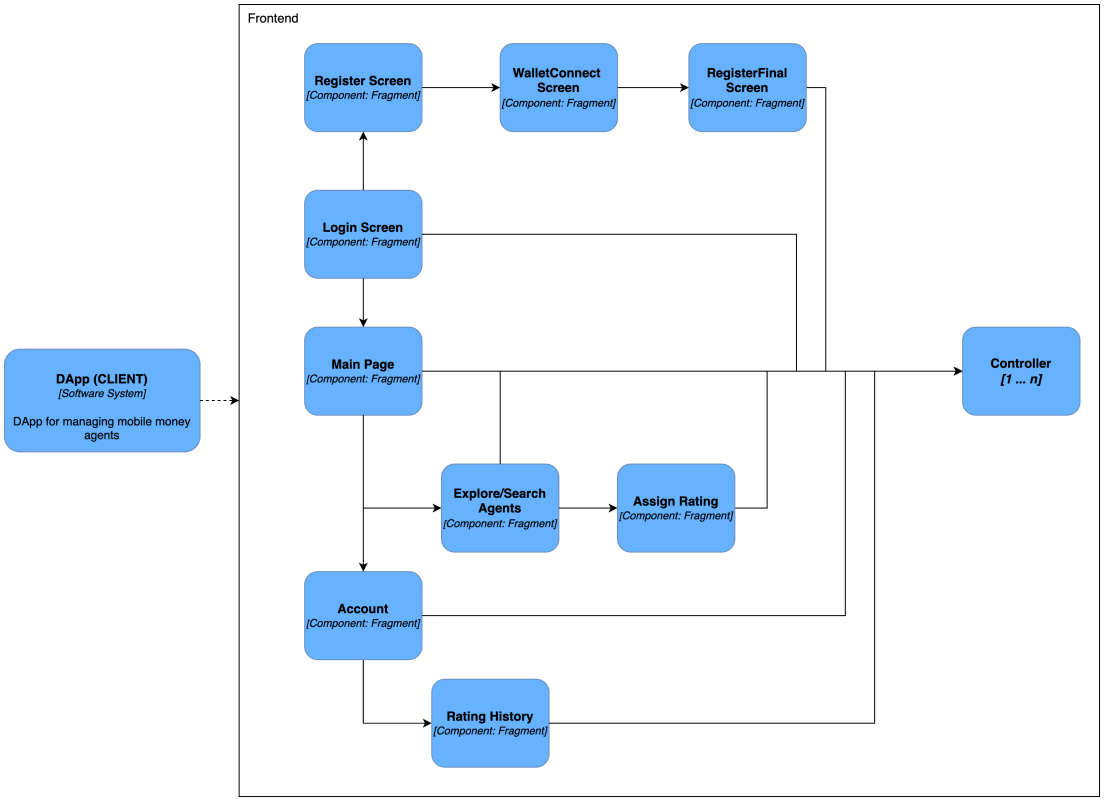

Figure 5.8: Component Level - Frontend (Client)

Backend

As with the frontend, the backend was divided into agent and client, figures B.1 and B.2 in appendix B. Furthermore, the interactions of controllers with the classes and the external software systems are represented.

5.6.5 Code Level

As this level is not mandatory and requires a lot of unnecessary details to understand the software, it was not developed.

5.7 Navigation Diagram

The mobile application navigation diagram is represented in figure 5.9, which shows the application flow from the user's perspective.

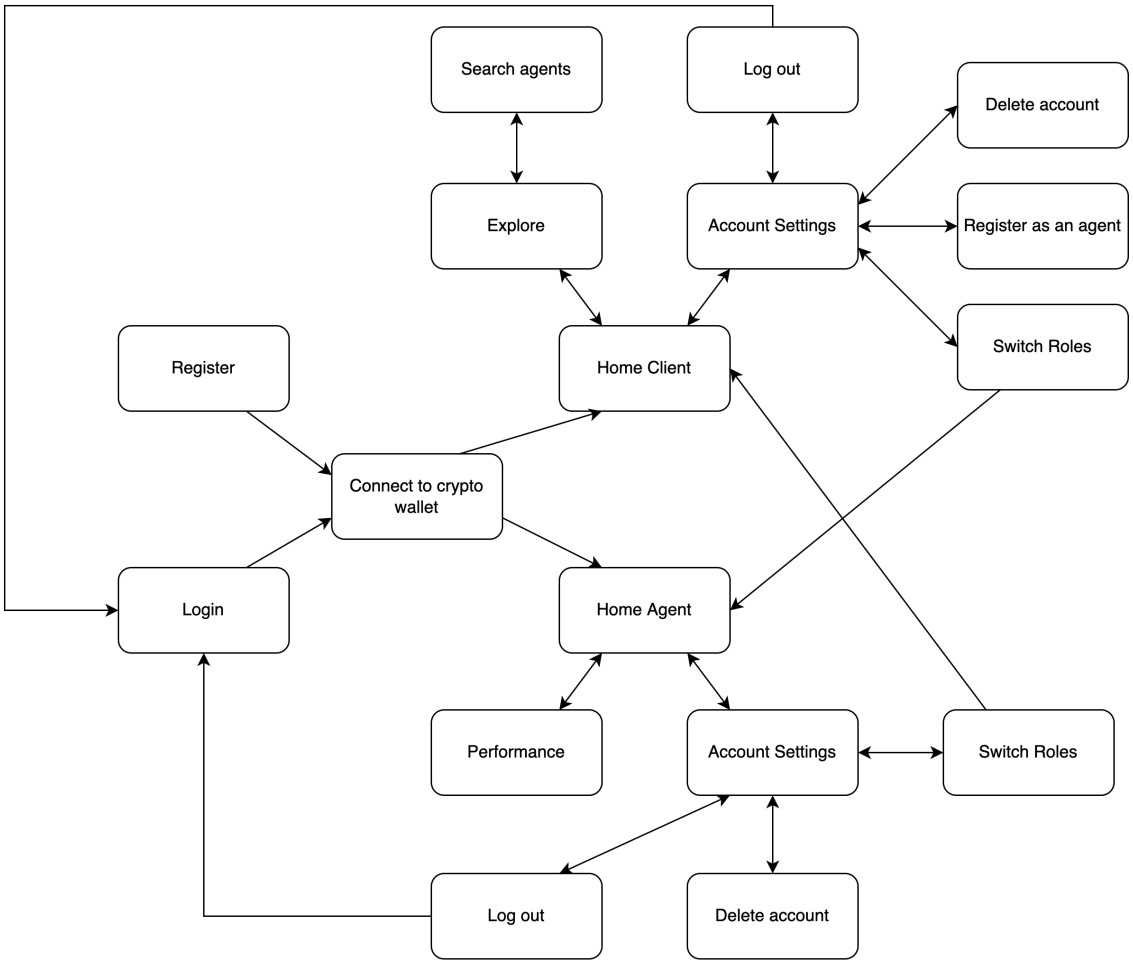

Figure 5.9: Navigation Diagram

In this navigation diagram, it's possible to start on the registration screen or login screen. If the user doesn't have an account, his path will begin on the registra-

tion screen. On the other hand, if the user already has an account, it can lead to two different scenarios. The first scenario is if the user is not logged yet into the device he is using. In this case, it will be redirected to registration to store user information locally. In the second scenario, the user is already logged into the device he is using, so he will be redirected to the login screen.

The "Home Client" is the screen where the user can see his balance, his latest transactions, and the agents near him. He can also explore and search agents by name, rating, and distance, and choose one to give a rating. In the account settings, it's possible to register as an agent or, in case he's already one to switch to the agent account. Deleting his account and logging out is also available.

The "Home Agent" is the screen where the user can see his balance, his latest transactions, and the average rating. He can also see and evaluate his performance in the latest months based on the rating obtained. In the account settings, it's possible to switch to the client account. Deleting his account and logging out is also available.

Chapter 6

Development

The objective of this chapter is to detail and explain all the steps of the development process. It starts by detailing how was the initial setup, the support tools used, and the work dynamic. Subsequently, clarify the project structure and visually show the developed functionalities, also a review to understand the features that were developed for both the users, clients, and agents. Then, topics such as security and privacy, as well the insurance to comply with General Data Protection Regulation (GDPR), are addressed. Finally, explain what would be the future work for this mobile application.

6.1 Process and project organization

To choose the process used for the 2nd semester, the team reached a decision, the project manager, the scrum master, the business analyst, and the intern. As with many other software projects, organizing is critical to have a chance to achieve success. For this purpose, a Jira Software project was set up at the beginning of the development. This tool allows to keep a record of every sprint, how it went, and evaluate if the effort was too low or high. In addition, Jira allows documenting important information that all team members can see. Figure 6.1 represents the Jira Board in one of the final sprints.

This board is divided into three columns for each sprint:

- **To do:** list of features to address
- **In progress:** list of features being addressed
- **Done:** list of features already done

During the sprint, if a task that is in the Done column needs a review, this task is dragged to the to-do column. At the end of the sprint, if there are unfinished tasks, these are moved to the next sprint.

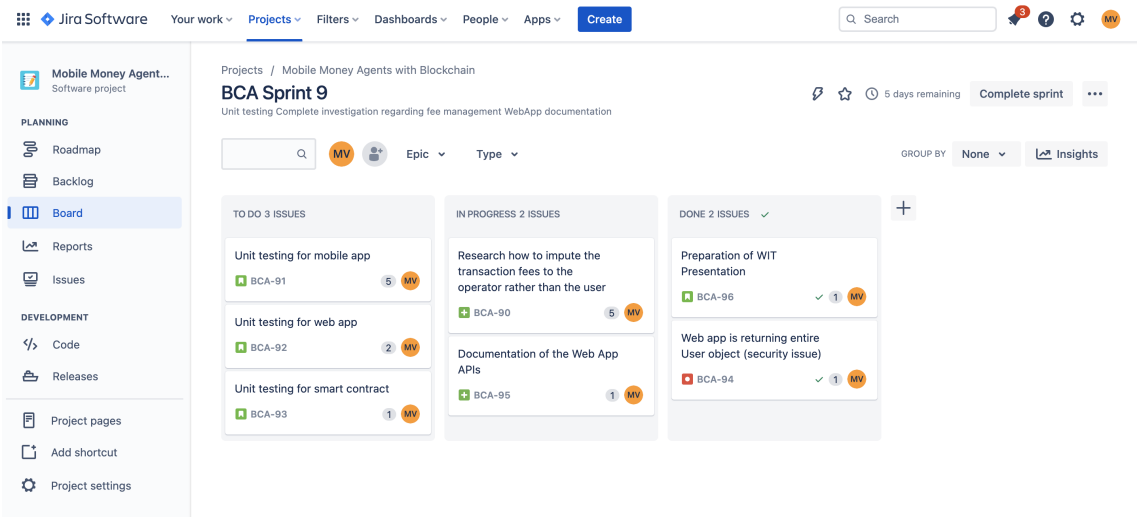

Figure 6.1: Jira Board

After the tools were set up and before the code development, the scrum master created a repository at GitLab for the mobile app. Later on, the scrum master created another repository for the server-side. Both repositories follow the git-flow represented by the figure 6.2.

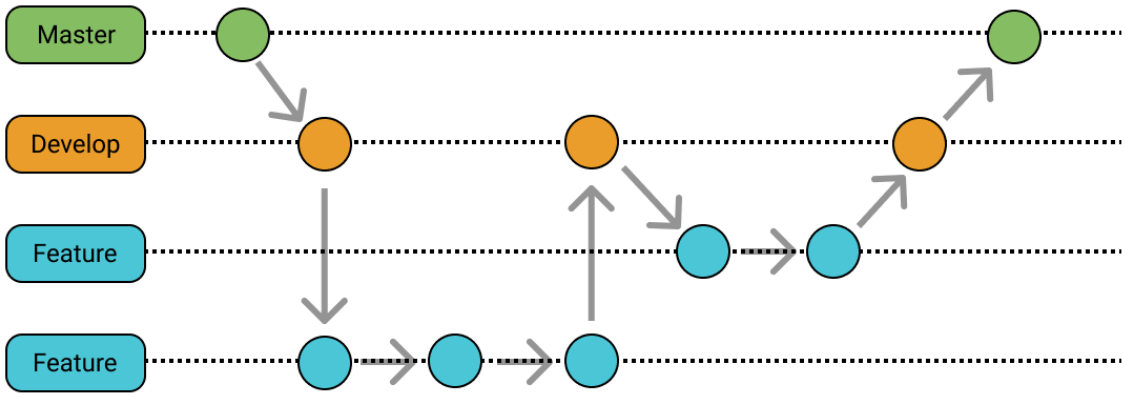

Figure 6.2: GitLab Flow

When the intern completed a task, the scrum master created a merge request to the "develop" branch. Then, he reviewed the code, and if everything was done accordingly, he accepted the merge request, and the code written was added to the "develop" branch. If not and the code needed any improvement, he would comment on the respective code lines so that it could be solved. Only when the intern has tested all the features of the "develop" branch can it be merged into the "master" branch.

6.2 Project Structure

If code needs revision or updates, it's important to maintain a good code structure. Not just for other developers but for us, who can get back to the project for updates or add more functionalities. Having and maintaining a clean and organized code is required to avoid unnecessary delays.

The following subsections present an overview and an explanation of the organization of the frontend and backend.

6.2.1 Frontend

This subsection represents the application frontend, which is the code developed in Android Studio in Kotlin. This project has two different User Interface's (UI's), one for agents and the other for clients, but there are mutual screens like the registration, login and wallet connect.

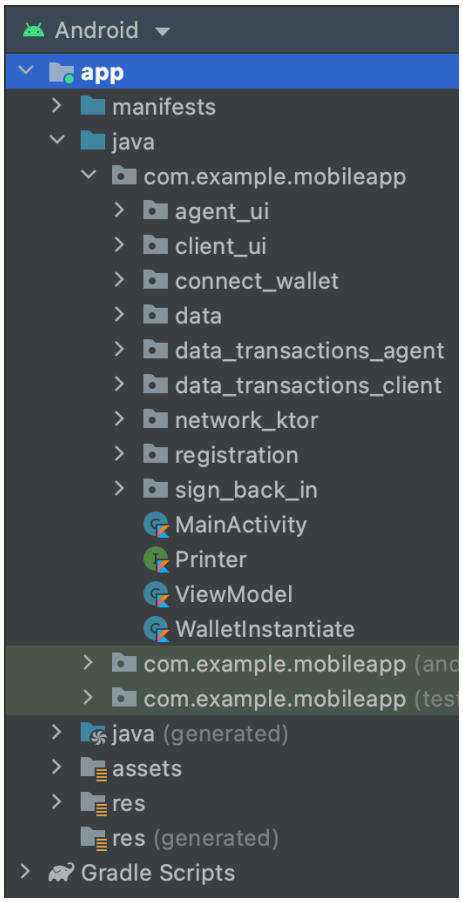

Figure 6.3: Frontend Structure

The two main folders of the frontend are the "java" folder and "res" folder. The first one has the classes to handle users' requests, and the second one represents the design that is visible to the users. "java" folder is divided into packages to have a clear division between each component.

/agent_ui: this package holds all the kotlin classes related to the agent UI.

/client_ui: this package holds all the kotlin classes related to the client UI.

/connect_wallet: this package holds all the kotlin classes related to the wallet connection before the registration. The purpose is to link the mobile app to the Metamask app and save the user wallet session.

/data: this package holds the data class that defines the User and its attributes.

/data_transactions_agent and _client: these packages have the same objective, which is to store in the local database, the data information about the agent and the client, respectively. The agent data stored in the database is its balance, performance, rating, and transactions. The client data stored in the database is its balance, rating history, and transactions.

/network_ktor: this package holds the network calls to the server (Ktor Server).

/registration: this package holds all the kotlin classes related to the registration UI for agents as clients.

/sign_back_in: this package holds all the kotlin classes related to the sign-back-in UI for agents as clients.

The "res" folder contains all the .xml files, like layout, drawable, font, and colors files.

6.2.2 Backend

This subsection represents the application backend. The first one to be addressed is the Ktor server which allows the calls to the user database, and the other is the smart contract.

Ktor Server (Web API)

In this API, the main files to analyze are the User.kt, UserRoutes.kt and Application.kt.

User.kt: contains the attributes to perform the CRUD operations.

Application.kt: is intended to start the server.

UserRoutes.kt: contains the connection to the database, the different operations, and the routes to make the calls to this server. This include GET, POST, and DELETE HTTP methods.

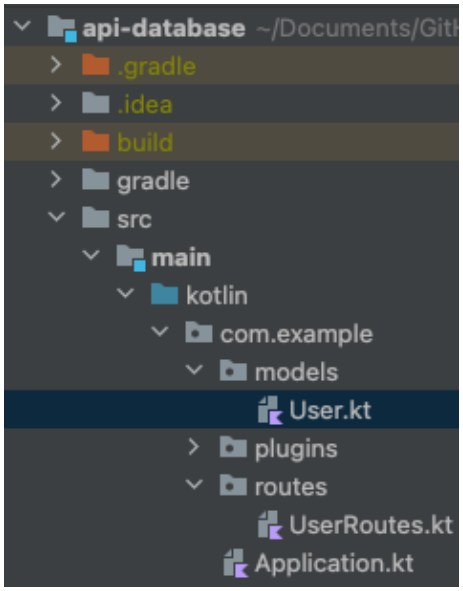

Figure 6.4: Ktor Server

Smart Contract

The smart contract is a single file, so there is no need to represent it visually. The smart contract contains the following methods:

- **addUser:** as the name suggests, this is to add a new user to the smart contract, registering only his public address.
- **assignRating:** this method is called by the mobile app when a client assigns a rating to an agent, storing the who, how much, and when for both of them.

- **getRatingAgent:** this method is to get the rating of the public address passed as an argument.
- **getRatingHistoryClient:** this method is for the client to see his rating history.
- **getRatingHistoryAgent:** this method is for the performance dashboard of each agent.

6.3 Developed requirements

This section presents three tables to show which requirements were satisfied during the project development. The first table is the module for authentication and management of personal data (table 6.1), then the module for agents (table 6.2), and, finally, the module for clients (table 6.3).

In addition, the user functionalities aggregated by the layouts of the mobile applications are described.

Module	Features	Priority	Status
Authentication and Manage personal data	1. Login	Must Have	Done
	2. Logout	Must Have	Done
	3. Registration	Should Have	Done
	4. Cryptographic wallet setup *	Must Have	Done
	5. Change Password	Should Have	Not done
	6. Change Personal Data	Should Have	Not done
	7. Delete account *	Must Have	Done
	8. Switch roles *	Must Have	Done
	9. Register as an agent (Client only) *	Must Have	Done
	10. Offline Mode	Must Have	Done

Table 6.1: Developed common requirements

Module	Features	Priority	Status
Main Dashboard	11. View Rating	Must Have	Done
	12. View performance dashboard	Should Have	Done
	13. Check Balance	Must Have	Done
	14. View recent transactions *	Should Have	Done

Table 6.2: Developed agents' requirements

Module	Features	Priority	Status
Main Dashboard	15. View Rating History	Must Have	Done
	16. Check Balance *	Must Have	Done
	17. View recent transactions *	Should Have	Done
	18. Search agent by location	Must Have	Done
	19. Search agent by rating	Must Have	Done
	20. Search agent by name *	Must Have	Done
	21. Assign a rating to a specific agent	Must Have	Done

Table 6.3: Developed clients' requirements

The first conclusion is that all the must-have requirements were satisfied. The second is that only two should-have were left undone, the "forget password" and the "change personal data" requirements. As there were other significant issues to address, it was decided as a team that it would not be a big problem if this app version didn't meet these two requirements.

6.3.1 Authentication

To be able to use the application and its features, users need to be registered. As referenced in the navigation diagram, when a user is not registered, their first registration is always as a client, and only then can they become an agent. If the user is already logged in to the device his using, the Sign Back In is the screen presented.

Registration

First, this subsection presents a registration flow to understand better the logic behind the registration, figure 6.5.

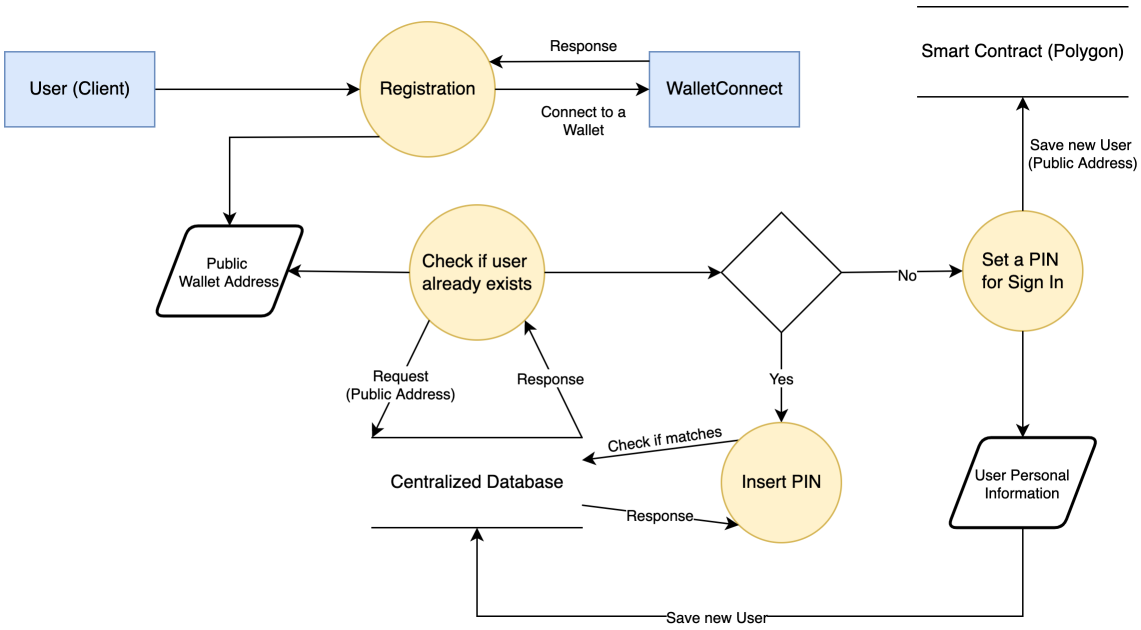

Figure 6.5: Registration Flow

In the use case of first registration or registration in a new device, the screens presented in figure 6.6 are shown. First, the user connects his wallet to use the blockchain resources (registration or assign rating). Then, the app redirects the user to the MetaMask app, where a confirmation popup appears, figure 6.7.

Next, as this is the user's first registration, he enters a new number to store, and, in this case, he typed a phone number already registered. The user chooses another phone number, which can be seen at the top of the second screen, figure 6.8. On this screen, the user finishes filling up his personal information. The MSISDN parameter is a unique number to identify a client and is assigned previously by the operator.

Finally, the user enters a numeric pin of his choice, and the application redirects him to the MetaMask app. Here, he accepts the transaction to register in the smart contract, where a function stores his public address, figure 6.9.

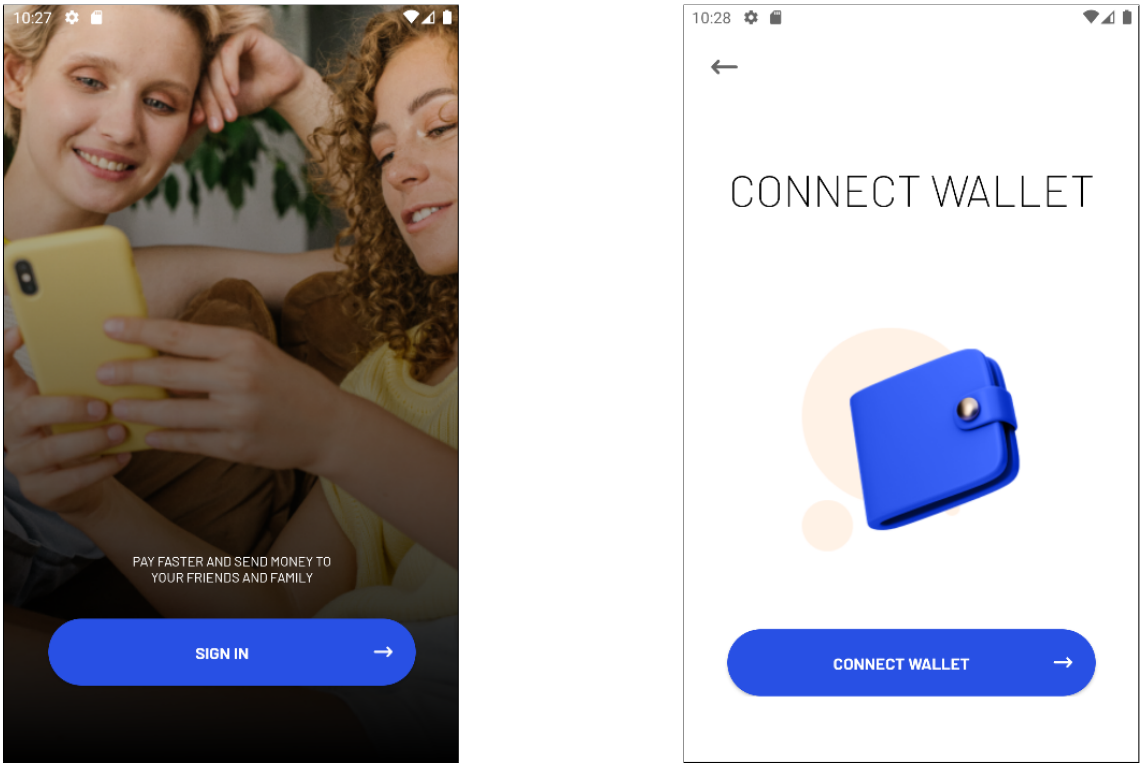

Figure 6.6: Initial screen and connect wallet

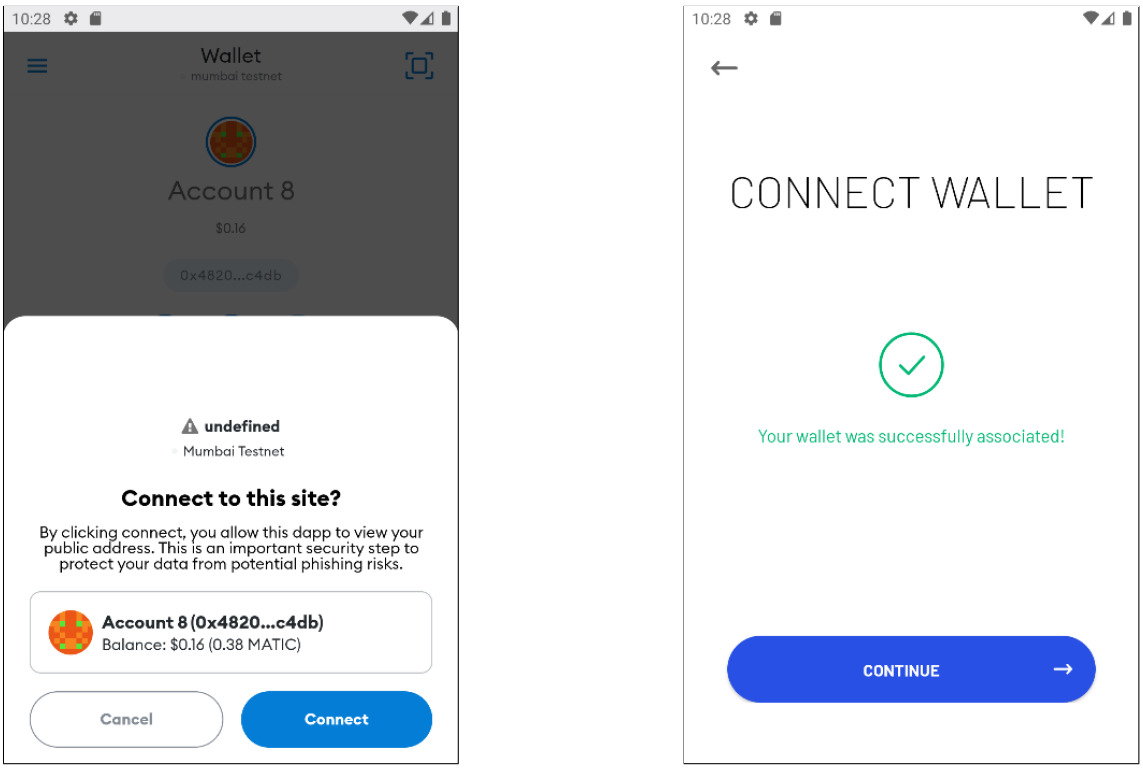

Figure 6.7: Connect wallet confirmation

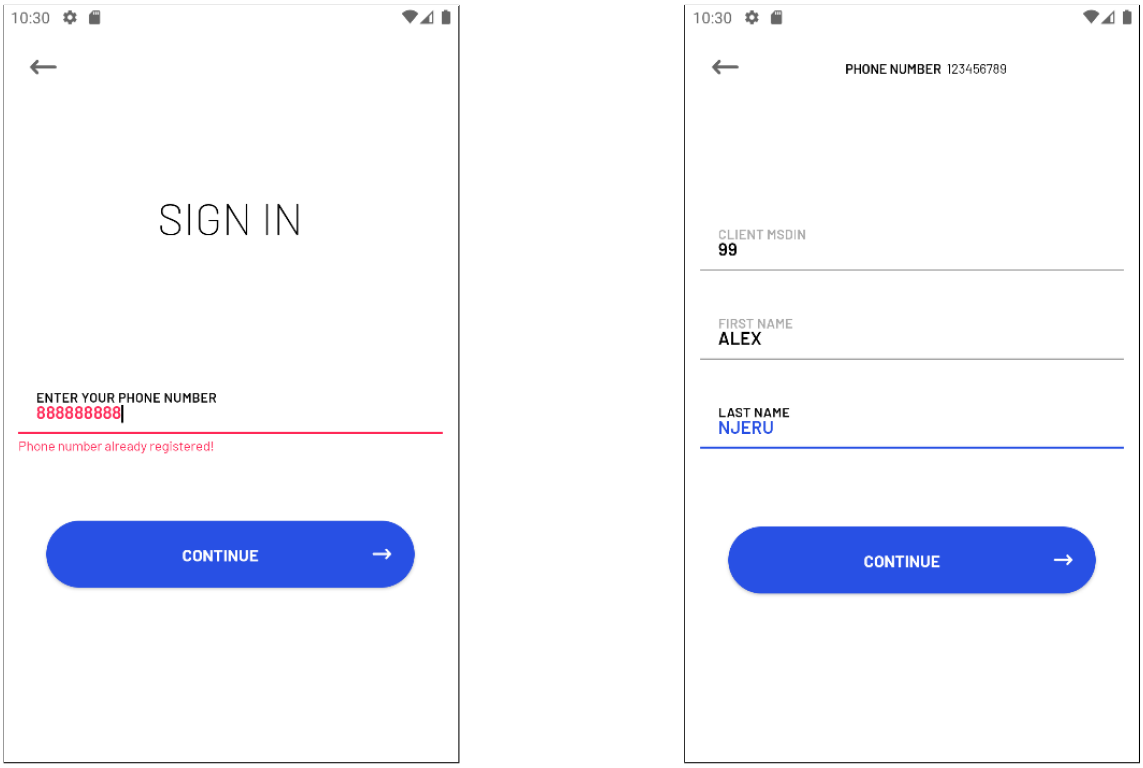

Figure 6.8: Sign up form

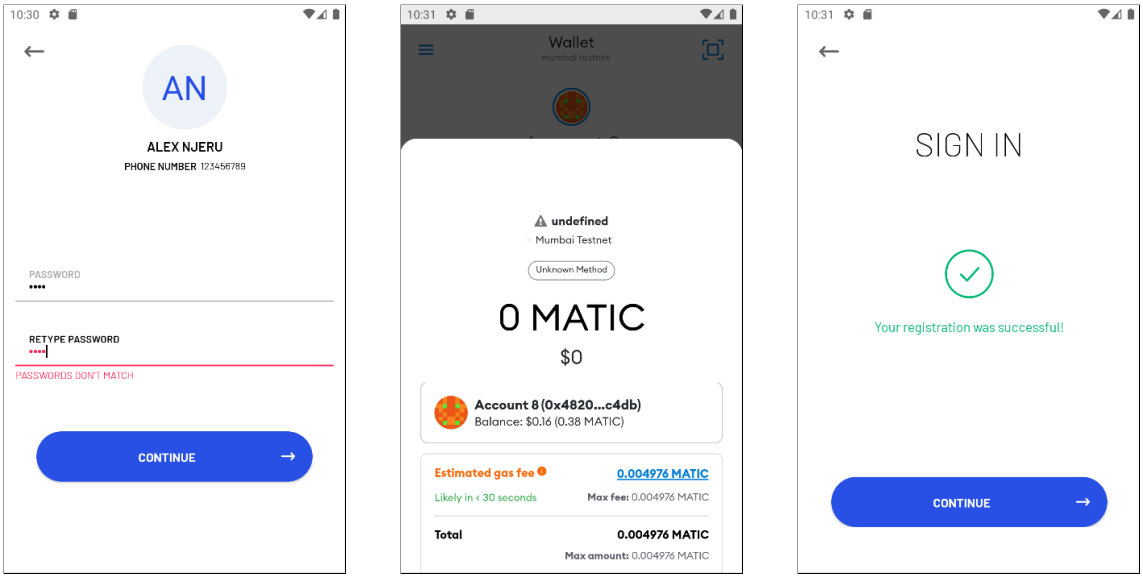

Figure 6.9: Smart contract registration

Sign back in

The sign-back-in screen is for the users already logged in to the device in use. The "change account" feature allows you to sign as a client or as an agent (if registered as one), figure 6.10.

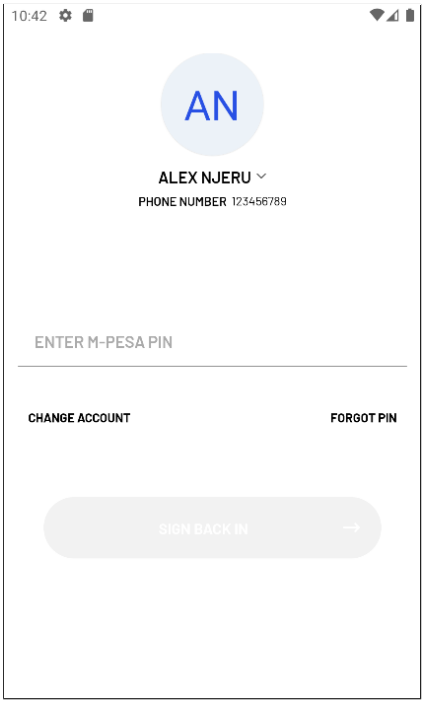

Figure 6.10: Sign back in

Account Settings

The client account settings screen allows the user to perform four different operations (figure 6.11):

1. Register as an agent / Switch Roles (if agent)
2. See his rating history
3. Logging out of the app
4. Delete his account

The first one is represented by figure 6.12, and the application prompts the user if he wants to set his current location as his business location. The shortcode number that represents the agent is assigned automatically in the backend. This number, like the client MSISDN, is set previously by the operator.

After that, where it was the "register as an agent" operation, it's now the "switch roles" operation. The user can switch from client to agent account and vice versa.

The agent account settings screen allows performing almost the same operations, except for the rating history option (figure 6.11).

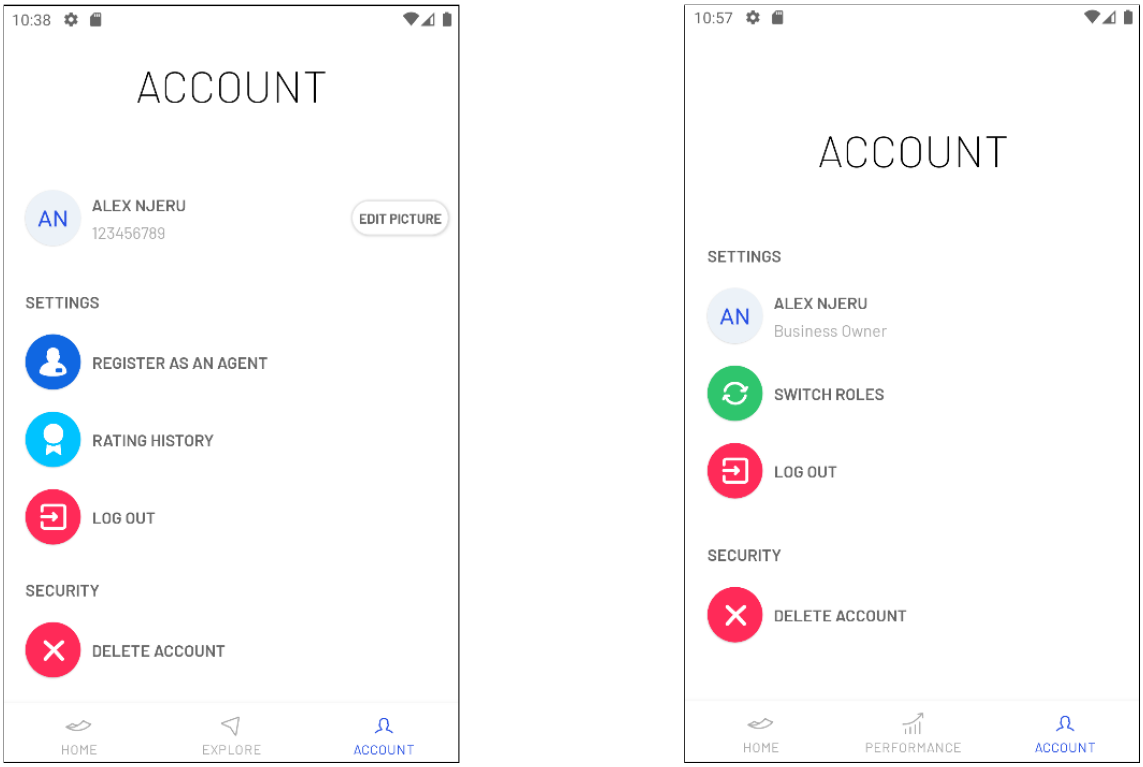

Figure 6.11: Account Settings (Client and Agent)

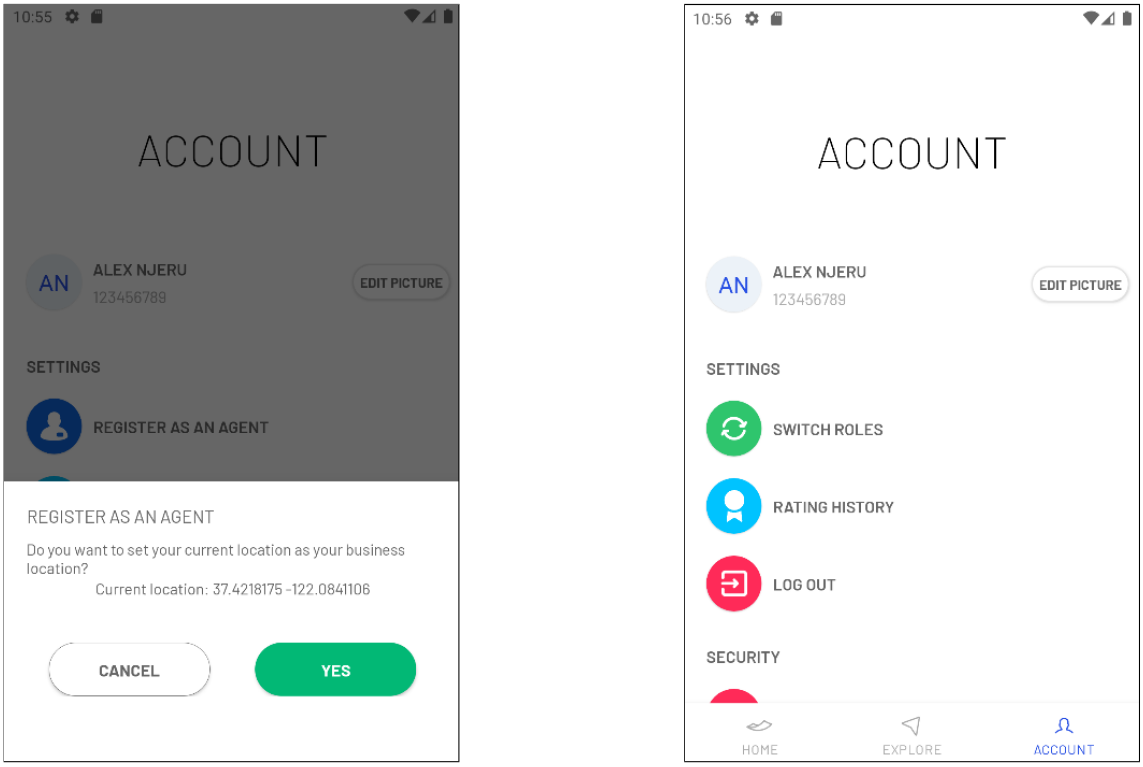

Figure 6.12: Registration as an agent

6.3.2 Agents' Module

This subsection represents the agents' UI and its functionalities.

Home

The home agent screen has three different components, figure 6.13:

1. Agent balance
2. Agent rating
3. Agent latest transactions

The first is a demonstrative value, and the objective is to represent the balance of an M-Pesa agent.

The second is the agent rating which is the medium of his latest 300 assigned ratings. The application obtains this value through the smart contract.

The last component represents the agent's latest transactions. These transactions are for demonstrative purposes only, not corresponding to reality.

The application fetches a fixed agent balance and transactions from the M-Pesa servers.

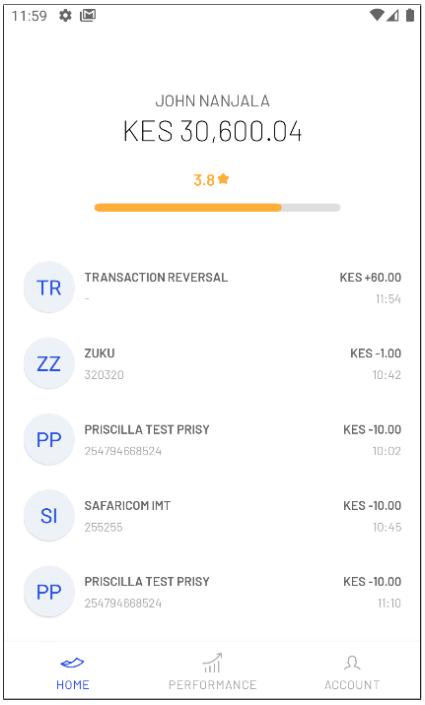

Figure 6.13: Agent Home

Performance Dashboard

The performance feature divides the screen into two components, figure 6.14:

1. Performance dashboard
2. Performance statistics

The former represents the evolution of the rating through the month. For each day, the average rating is obtained. The latter represents the statistics compared with the previous month. In the second screen of figure 6.14, it's possible to conclude that in the current month, there were fewer transactions and ratings than the month before (-33.3%). Also, the average rating decreased by 33.3%. The "total amount transacted" is a random and demonstrative value.

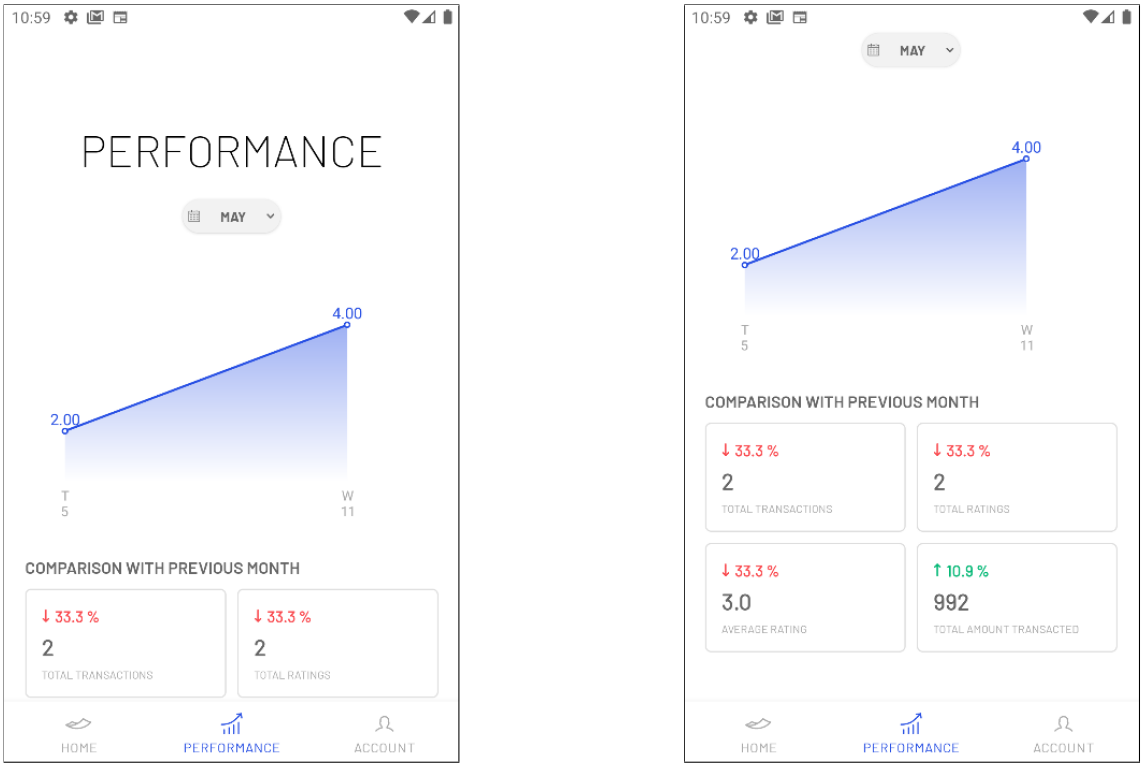

Figure 6.14: Performance Dashboard

6.3.3 Clients' Module

This subsection represents the clients' UI and its functionalities.

Home

The home client screen has three different components, figure 6.15:

1. Client balance
2. Client latest transaction
3. Nearby agents

Such as the agent balance, the client balance is a demonstrative value, and the objective is to represent the balance of an M-Pesa client.

The second represents the client's latest transaction. This transaction is for demonstrative purposes only, not corresponding to reality.

The last component expresses the nearby agents ordered by distance. When the user clicks on the name of any agent, the map highlights his location. Google Maps is the tool behind this feature.

Like the agent UI, the application fetches a fixed client balance and transactions from the M-Pesa servers.

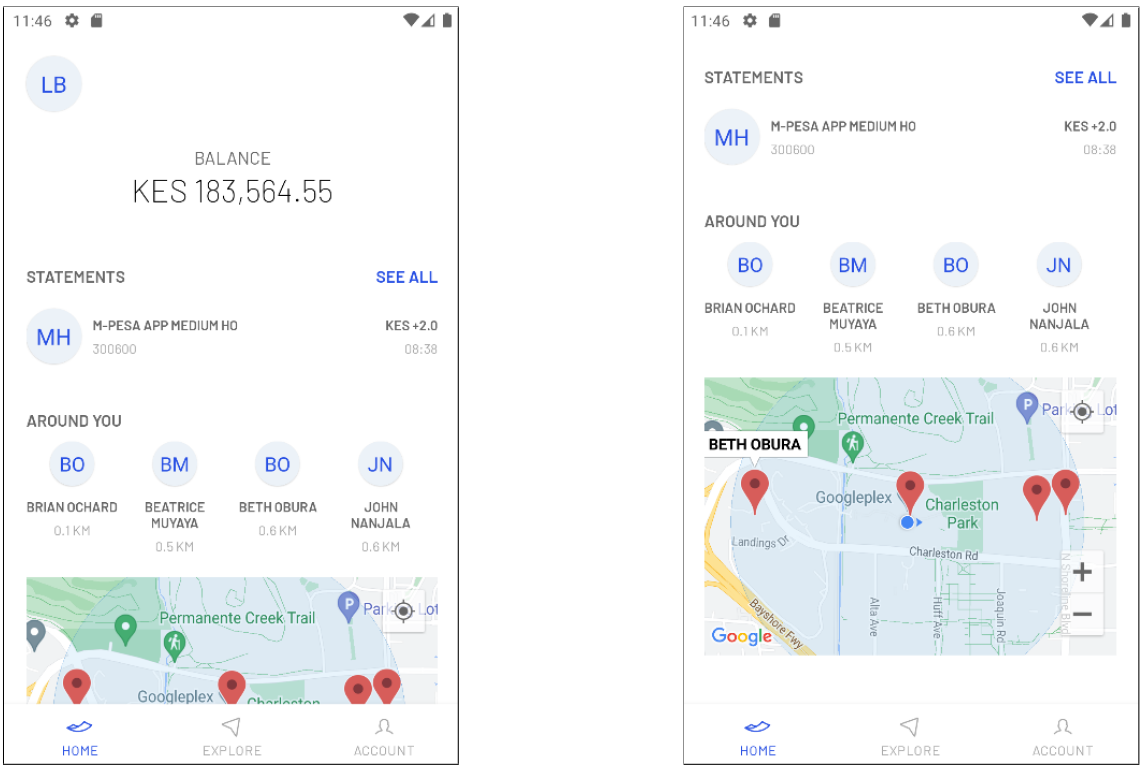

Figure 6.15: Client Home

Explore

The explore screen has four different components:

1. Explore agents directly on the map
2. Search and order agents

3. See more information when clicking an agent

4. Assign a rating to an agent

The user can drag around the map to find the agents' location and click on each pin to access more information about the agent, figure 6.16.

The second component allows the user to search the agent by name or number and order agents alphabetically, by rating, or by distance - figure 6.16.

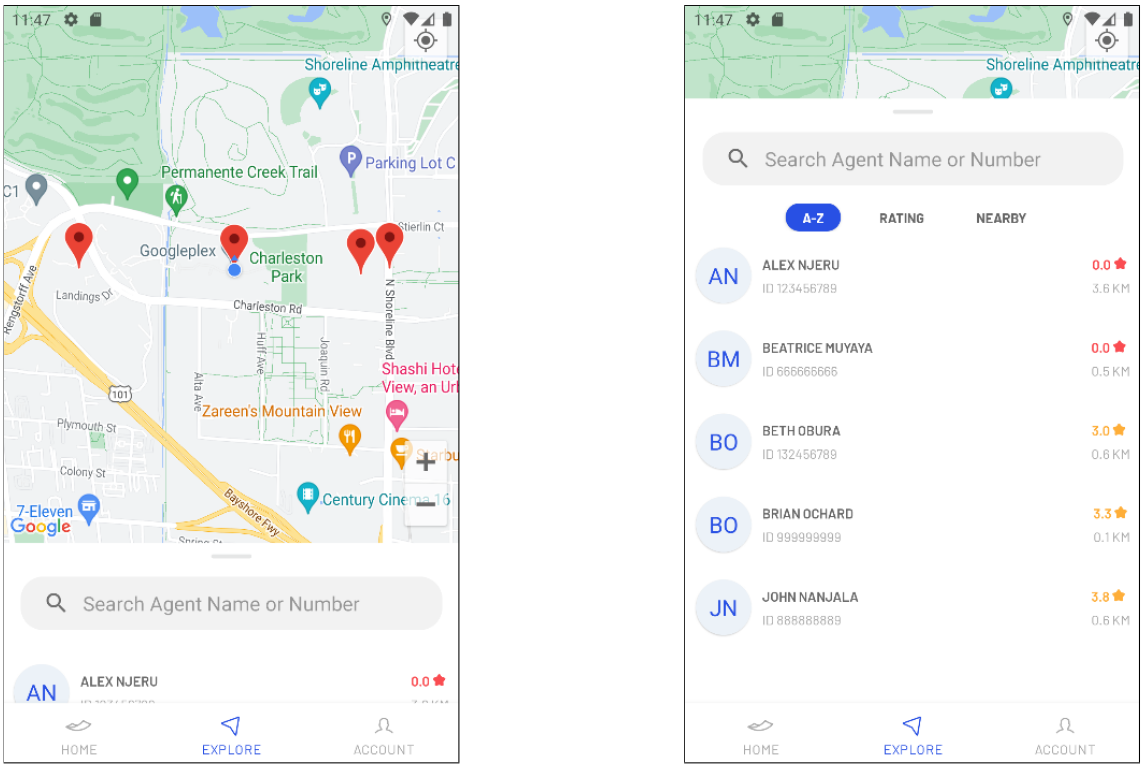

Figure 6.16: Explore Client

When the user clicks on an agent, the application shows more detailed information about that agent, figure 6.17. The operation "withdraw at agent" is not done because there was no need to add this complexity, so when the user clicks it, the application jumps to the assign rating feature, figure 6.17.

Like the registration feature, the smart contract has the "assign rating" feature. The client confirms and pays the transaction to write the rating assigned to the chosen agent, figure 6.18

Rating History

The client has a feature that allows him to check his rating history ordered by the most recent rating assigned. The user can see each rating and, in each one, to which agent, how much, and when - figure 6.19.

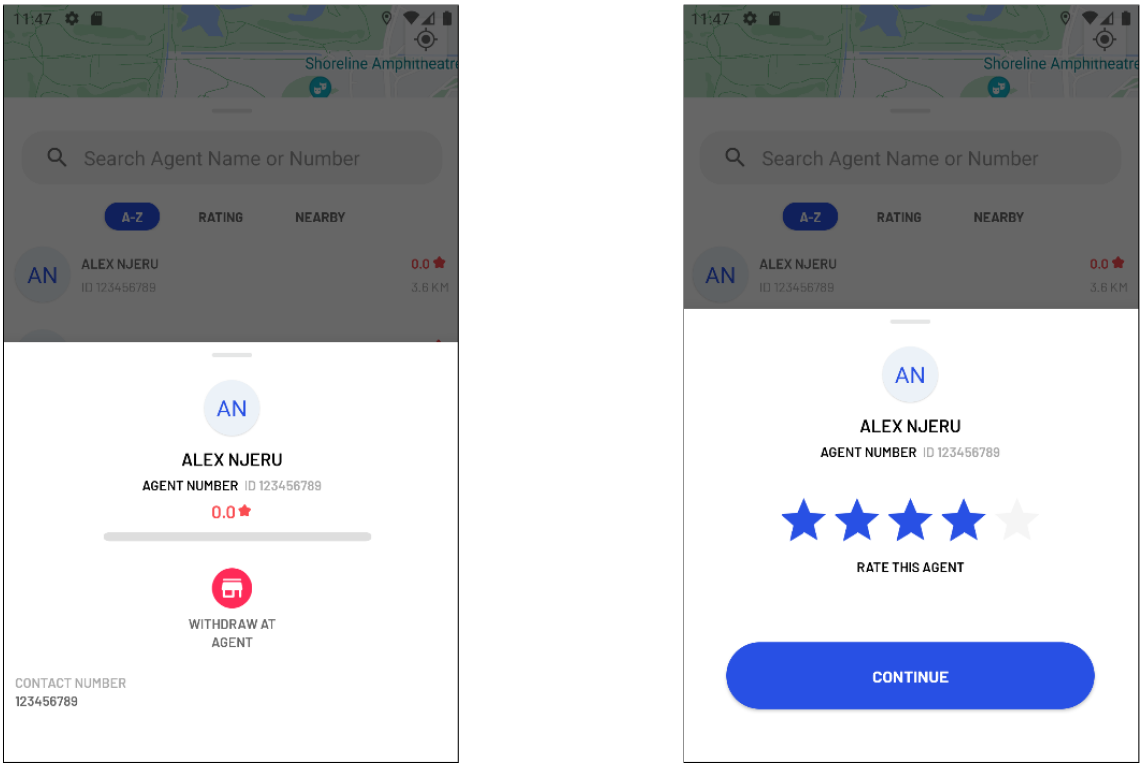

Figure 6.17: Operation - Assign rating to an agent

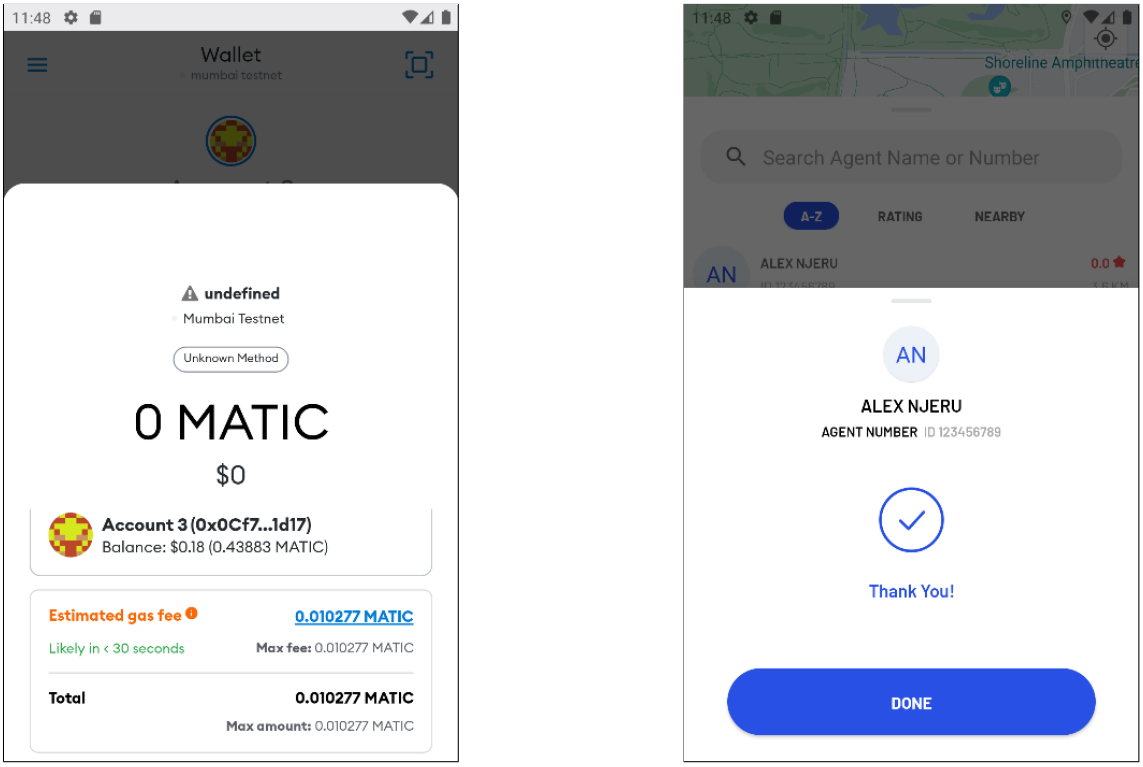

Figure 6.18: Confirmation - Assign rating to an agent

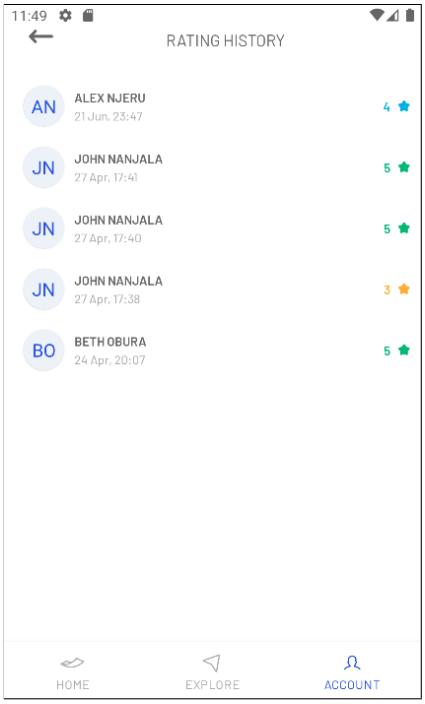

Figure 6.19: Rating History

6.4 Risks

During the project development, four risks were identified, represented in section 5.5. At this final stage of development, it is possible to conduct a general analysis of all risks and mitigation strategies.

- **ID_1:** Inexperience with blockchain technology
- **ID_2:** Inexperience with the chosen smart contracts language - Solidity
- **ID_3:** Low experience in mobile app development
- **ID_4:** Deployment of smart contracts with bugs

ID	Impact	Probability	Occurred
ID_1	High	High	Yes
ID_2	High	High	Yes
ID_3	Medium	Low	No
ID_4	High	Medium	Yes

Table 6.4: Risks

The first two risks occurred and had the impact expected due to the intern inexperience in these technologies, which caused an initial delay in the project's start. Over time, this impact was minimized since the intern gained experience and confidence.

The third risk did not occur because the mitigation plan was successful from the start, the intern understood the Kotlin language and how the Android Studio flow worked with ease.

The last risk has occurred repeatedly during the project development due to mistakes in the smart contract functions or new features added. Subsequently, there have been delays in development, and before any public release, the smart contract must be thoroughly reviewed and analyzed.

6.5 Security and Privacy

One of the concerns about the blockchain is that the data is immutable, which means there is no way to delete data once it enters the blockchain. To avoid dealing with this issue, the architecture adopted by the intern was hybrid. The data stored in the blockchain are the public wallet address and the ratings as an agent and a client.

6.5.1 GDPR

The app must respect the General Data Protection Regulation (GDPR). "All the rules, restrictions, and requirements placed in the GDPR share the aim of protecting data subjects (or users) and upholding their rights."[48] For those who doesn't respect these laws, "the GDPR will levy harsh fines against those who violate its privacy and security standards, with penalties reaching into the tens of millions of euros." [49]

The developed mobile app collects users' data, so there are some user rights [48] under GDPR that need to be respected.

1. The Right to Rectification
2. The Right to Erasure
3. The Right to Information

The first user right is not respected by the application yet. Those were the two should-have requirements that were left undone, change personal data and password. However, the app is not public, so that will be fixed until then.

The application respects the Right to Erasure (Right to be forgotten). The user can delete his account and all his data with it. Although blockchain data cannot be deleted, there is no relation between the public address and the users' personal information.

The last crucial right to mention is the Right to Information. The app needs to access users' locations, so in the first user login, the app shows a dialog to ask for user consent - figure 6.20.

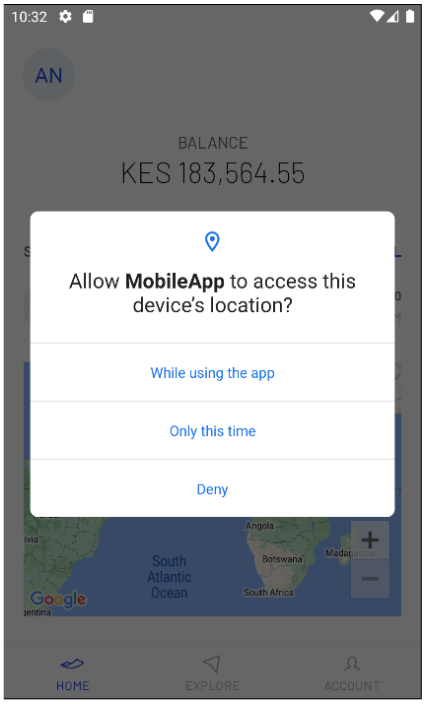

Figure 6.20: Location Permission

6.6 Future work

As for future work, there is an important aspect to be addressed regarding the technical setup of the digital wallet required to use the application. This section is divided into four topics:

1. Problem
2. Hypothesis
3. Comparative analysis between blockchain and a traditional database (Appendix C)
4. Conclusion

6.6.1 Problem

When building an application of any kind, an application with a backend supported by blockchain or not, we want a sustainable business model so that we can gain users. Without any tools or code, if a client uses a feature that needs a transaction, he needs to have ETH or MATIC (Polygon Network) to pay for gas fees. This forces users to have prior knowledge of crypto wallets and understand the blockchain concept, which can be an obstacle to users' adherence.

6.6.2 Hypothesis

Gas Station Network's [50] primary goal is to solve this problem.

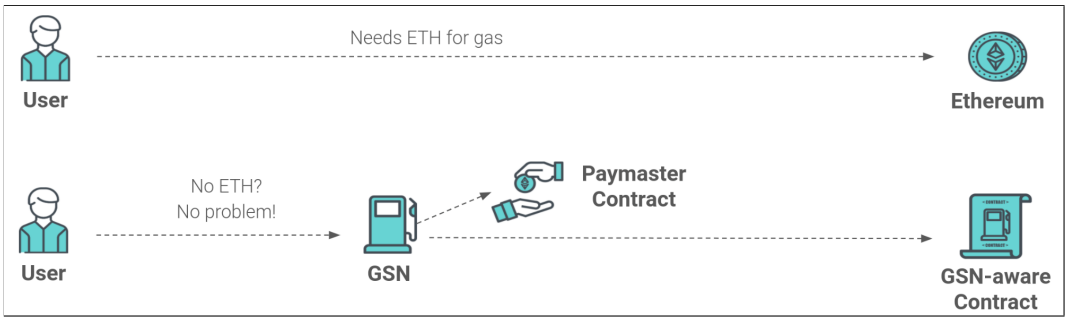

Figure 6.21: GSN [50]

"GSN is a network of servers — called relayers — that are waiting to execute transactions. Each relayer gets paid for every transaction put on the blockchain. Relayers are incentivized to behave correctly and not cheat." [51]

Figure 6.22 represents the GSN architecture.

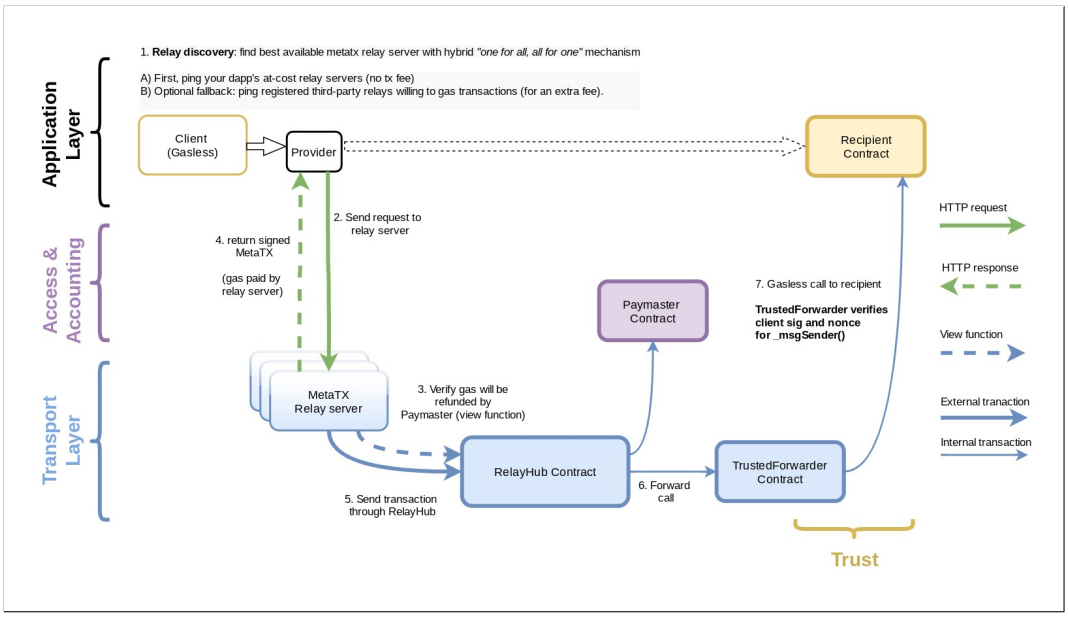

Figure 6.22: GSN Architecture [50]

- **Client:** interacts with a feature that requires a blockchain method, which triggers sending the meta transaction to the relay server.
- **Relay servers:**
  - **Meta-transaction:** relay server sends user's transaction and pays themselves for the gas cost. So, the user only signs the message containing information about a transaction they would like to execute and this is sent to the relay server.

  - First, the relay server checks if it will get refunded by the Paymaster contract, and only then does the relay server pays for gas.

- **Paymaster Contract:**
  - Control and gas refund logic
  - Gas tank of ETH in the RelayHub which has the power to refund relay server for gas fees
  - Business logic to decide whether to accept or reject a meta transaction
- **TrustedForwarder Contract:**
  - This contract verifies the signature and nonce of the original sender (user)
- **Recipient contract:**
  - _msgSender() – gets the original sender that signed the meta transaction request
- **RelayHub:**
  - RelayHub connects users running clients, relay servers, and paymasters so that participants don't need to know about or trust each other.

6.6.3 Comparative analysis (Appendix C)

6.6.4 Conclusion

Based on the comparative analysis made in Appendix C, conclusions can be drawn.

Despite the cost charged to the operators, GSN removes all the technical setup for the users. The operators can gain from the relay server when other DApps need it by defining the cost of each executed transaction.

Comparing the price values of the blockchain and an AWS Database, it is clear that, from a financial point of view, the option for blockchain may not offer the best conditions. The blockchain price calculated may be inflated because the smart contract functions can be improved and optimized so that the price of transacting with each method doesn't cost so much.

The question of value arises again because the blockchain offers features that a traditional database does not, which is harder to measure for this specific use case. Blockchain ensures better security, data integrity, immutability, and transparency. It is crucial to understand what we value more, the business model or the assurance of data integrity.

Chapter 7

Testing

This chapter describes the software tests performed on the mobile app. It is the evaluation of the application to verify that it does what it should do.

Software testing is indispensable because it prevents bugs, reduces development costs, and improves performance [52]. Although testing costs money, problems found early in testing can save companies millions.

There are many types of software testing techniques that can be performed on a product. The first section of this chapter presents the unit testing for three different components. These components include the smart contract, the mobile app, and the web API. Next, the second section describes the usability testing and its conclusions.

7.1 Unit Testing

Unit testing is a type of test that aims to validate a small component of an application. Regarding the smart contract and web API, all components were tested. On the other hand, only a few application parts were tested in the mobile app, prioritizing which were most valuable due to the time available.

Each test is represented by:

- **ID:** test ID
- **Description:** the purpose of the test
- **Expected outcome:** what the test should return
- **Result:** what the test actually returns - Pass (**P**) and Fail (**F**)

7.1.1 Web API

As mentioned in the previous chapters, this server allows the application frontend to make requests to the database. The tested methods are described in

the API Documentation represented by appendix E. Table 7.1 represents and describes the done unit tests. The database was previously populated with users with client accounts and with client and agent accounts.

Method	ID	Description	Expected outcome	Result (P/F)
checkCompositeKey	1	The user's public address and phone number match.	Return User object without pin	P
	2	The user's public address and phone number don't match.	Return empty object	P
checkPassword	3	Inserted pin matches.	"success"	P
	4	Inserted pin doesn't match.	"no match"	P
checkPhoneNumber	5	Phone number inserted taken.	"taken"	P
	6	Phone number inserted non taken.	"non taken"	P
createUser	7	Create new User object.	200 OK	P
deleteUser	8	Delete User object.	200 OK	P
getAgents	9	The list length must return different than zero.	list.size > 0	P
getUserByName	10	Get existing user by public address.	Return User object without pin	P
	11	Get non existing user by public address.	Return empty object	P
updateUser	12	Update user parameters.	200 OK	P

Table 7.1: Web API Testing

7.1.2 Smart Contract

The smart contract has four different functions related to rating operations. The tests were performed in the Remix IDE, which has a testing tool besides the writing, compiling, and deployment tools.

Table 7.2 represents and describes the done unit tests on the smart contract.

ID	Description	Expected outcome	Result (P/F)
13	Add user address and check if it is possible to assign rating.	Added successfully	P
14	Assign a rating to an agent. Add two ratings and see if the rating corresponds to the medium (getRating)	Assign rating successfully	P
15	Assign a rating to an agent. See if this was added to the agent rating history.	Added successfully	P
16	Assign a rating to an agent. See if this was added to the client rating history.	Added successfully	P

Table 7.2: Smart Contract Testing

7.1.3 Mobile App

The unit tests performed on the mobile app were restricted to three components. The first was to the local database of a client account, represented by table 7.3. The second was to the local database of an agent account, represented by table 7.4. Both of these tests were to perform CRUD (Create, Read, Update, Delete) operations. The final and third were to the unit components that interacted with the API methods, represented by table 7.5.

Local Client Database Methods	ID	Description	Expected outcome	Result (P/F)
addBalance ()	17	Add a balance and check if is stored.	get balance not empty	P
updateBalance ()	18	Update the balance added and check if updates.	get balance not empty	P
deleteBalance ()	19	Delete the balance and check if is deleted.	balance not found	P
addRatingHistory ()	20	Add a rating object and check if is stored.	get rating not empty	P
deleteAllRatingHistory ()	21	Delete all rating history and check if it is empty.	get rating list - empty	P
addTransaction ()	22	Add a transaction object and check if is stored.	get transaction not empty	P
deleteAllTransactions ()	23	Delete all transaction history and check if it is empty.	get transaction list - empty	P

Table 7.3: Mobile App Testing - Client Database

Local Agent Database Methods	ID	Description	Expected outcome	Result (P/F)
addBalance ()	24	Add a balance and check if is stored.	get balance not empty	P
updateBalance ()	25	Update the balance added and check if updates.	get balance not empty	P
deleteBalance ()	26	Delete the balance and check if is deleted.	balance not found	P
addPerformanceHistory ()	27	Add a performance object and check if is stored.	get performance not empty	P
deleteAllPerformanceHistory ()	28	Delete all performance history and check if it is empty.	get performance list - empty	P
addRating ()	29	Add a rating and check if is stored.	get rating not empty	P
updateRating ()	30	Update the rating added and check if updates.	get rating not empty	P
deleteRating ()	31	Delete the rating and check if is deleted.	rating not found	P
addTransaction ()	32	Add a transaction object and check if is stored.	get transaction not empty	P
deleteAllTransactions ()	33	Delete all transaction history and check if it is empty.	get transaction list - empty	P

Table 7.4: Mobile App Testing - Agent Database

API Service	ID	Description	Expected outcome	Result (P/F)
createUserTest ()	34	Add a user and check if is stored.	get user not empty	P
getUserTest ()	35	Get user by address.	get user not empty	P
checkPhoneNumberTest ()	36	Check if phone number is taken.	"taken"	P
checkCompositeKeyTest ()	37	Check if pubAddress and phone number match.	user object not empty	P
updateUserTest ()	38	Update the added user and check if short code is != " "	short code != " "	P
getAgentsTest ()	39	Get agents and check if list size is greater than zero.	agentsList.size > 0	P
deleteUserTest ()	40	Delete user added and check if is deleted.	get user empty	P

Table 7.5: Mobile App Testing - API Service

7.2 Usability Testing

Usability testing is a type of test that aims to validate how well a user can complete a task in an application. These tests allow assessing two non-functional requirements described in Section 5.4. The learnability and usability requirements.

The core elements of these tests are [53]:

- **Facilitator:** the intern assumes the role of administering tasks to the participant.
- **Tasks:** provided by the facilitator.
- **Participant:** performs the tasks and gives feedback to the facilitator.

Typically, participants should belong to the target user group or have experience in the application being tested. It is not possible to have target users because the target market is not within reach of the intern. The users chosen by the intern were five [54] software engineering students. The intern had to explain the concept precisely because none of them had experience in this topic.

7.2.1 Tests

First, the intern explained to each user that they needed to create a metamask wallet to perform smart contract functions (registration and rating). The intern was responsible for creating this for each user. Then, the intern asked users to perform the following tasks:

1. Register in the mobile application
2. Search an agent by name
3. Check his client account balance
4. Check agents around him
5. Find an agent on the map
6. Assign a rating to an agent
7. Check rating history
8. Register as an agent
9. Log out of the application
10. Log in to the mobile application as an agent
11. Check his agent account rating

12. Check his agent account transactions
13. Check his agent performance over the past month
14. Switch to the client account
15. Delete his account

The intern registered the clicks that took each user to perform the tasks.

7.2.2 Results

Table 7.6 represents the expected number of clicks for each task.

Tasks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Number of Clicks	7	3	1	1	2	6	2	3	3	3	1	1	3	2	3

Table 7.6: Expected number of clicks per-task

In the definition of the quality attribute (subsection 5.4.3), the maximum of clicks defined was three. However, as the table represents, there are two tasks where that cannot be possible to perform in three clicks max - tasks 1 and 6. The intern did not provide any tips to the users, only when asked.

Table 7.7 shows the number of clicks of each user when performing the required tasks.

		Tasks														
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Users	**1**	7	3	1	1	2	10	2	3	3	3	1	1	3	2	3
	2	9	3	1	1	2	6	3	3	3	5	1	1	3	2	3
	3	7	3	1	1	2	6	3	3	3	3	1	1	3	2	3
	4	7	3	1	1	2	8	2	5	3	3	1	1	3	2	3
	5	7	3	1	1	2	6	3	3	3	5	1	1	3	2	3

Table 7.7: Real number of clicks per-task

In general, the users corresponded to the expectations. In task 1, only one user deviated from the expected number of clicks because he had to repeat the connection to the wallet due the metamask app didn't open.

For the task of assigning a rating to an agent - task 6 - two users (users 1 and 4) did not understand where they could do that. Because when a user clicks on

an agent, the button has the text "Withdraw at agent". Withdraw is an operation performed by clients when they want to cash out money from their accounts. However, the rating is assigned directly without performing that operation.

Three users first went to the home screen when they were asked to perform task 7. And only then, when they couldn't find the rating history on the home screen, did they go to the account screen.

The register as an agent button is located on the account settings screen. When a user clicks it, the app shows a popup to register the client as an agent and asks if he wants to define his current location as his business location. User 4 thought he could determine another location later, so he clicked on the cancel button, but the application doesn't allow users to register without a business location. Therefore, this user had to repeat the steps.

Finally, in task 10, users 2 and 5 went to the correct component to change the account to log in. However, they didn't understand if this action got any response, so they went back to clicking the change account button.

7.3 Conclusions

Concerning the unit testing, the tests performed helped fix defects found during the analysis of the test results. This type of test was made before the usability testing, so there were no bugs in the application.

Regarding usability testing, the usability quality attribute is respected by the application. The variations were not far from what the intern defined and expected - three clicks per task. The app respects the learnability quality attribute. The intern did not apply the defined response, which was the app showing hints. Even so, problems encountered by users in some tasks are related to how the UI/UX is designed.

The intern performed the usability testing later than expected. Consequently, the issues found in this phase have not been fixed but will be in the next version of the application.

The features that need improvements are:

- Change the location of the rating history feature. This new location could be in the user's bottom navigation or the home screen.
- When the client wants to register as an agent, allow him to target another business location than the current one.
- On the login screen, define feedback that makes it possible to understand what type of account the user is logging in to the application.

Chapter 8

Conclusion

The project presented by WIT Software and consequently the subject of this internship was the merge of mobile money services and blockchain. The main objective was to understand how the blockchain could be integrated by developing a rating functionality for agents of mobile money services. The process to reach results and conclusions regarding this problem went through several stages and steps. First, the state of the art allowed the intern to understand better how mobile money services operate. Defining a methodology also helped to keep all progress organized. The definition of requirements, use cases, and architecture was fundamental in the steps before development. The development itself is one of the most important, as it allowed a better understanding of the project and deliberation about details that the intern did not consider before. And finally, testing, which mainly allowed the intern to find some usability issues.

The first step was the writing of the SoA, allowing an in-depth study of how mobile money services currently work. Although this study was not of much help regarding blockchain, it did help to understand how competitors design these services. Therefore, this study was a great help for the functional and usability component of the mobile application (frontend).

The next step was to understand how the intern could design the solution and the technologies to develop it. Initially, the proposed solution was based only on the blockchain, without having any centralized service, raising some security and privacy issues. These issues had to be evaluated in the second semester during the development. Regarding technologies, these were chosen in the first semester and remained until the end. The selected technologies were not a problem due to the rigorous study done by the intern.

The definition of the methodology adopted in the development and the planning helped maintain an organized project. The Scrum methodology was helpful when some changes had to be performed in the project architecture and, consequently, in the development. Because Scrum bases its methodology on iterations, it allowed the intern to make these changes without redoing the entire project.

The last step before development was the requirements study, use cases, and architecture. The description of the system was well explained and defined, so the

result of this phase provided an excellent basis for development.

The development is one of the most crucial phases. First, the work tools (Jira and Git) allowed constant support and proximity between the intern and the WIT Software team. Regarding the developed features, they were almost all finalized. The intern could not finish those because the team considered that there was more important work to do, the future work. The problem raised in the description of the future work is related to the fact that users have to do all the technical setup related to the blockchain before using the application. The intern addressed this issue at a very advanced stage of the project. Therefore there was no time for any implementation. This future work is very critical for the next phase of this project. It allows users to use a complex technology like blockchain without realizing they are using it. It is possible to have all the benefits that blockchain offers without prior knowledge of this technology.

The last stage of development, testing, allowed the intern to find errors in the main components on which the unit test was done. Before usability testing, the intern resolved these issues. The intern noticed UI/UX problems after the results of the usability tests. In the next phase of the project, these problems will be solved.

As a final consideration of the objective of the internship, is this solution plausible for the target users? It depends on the value given to a blockchain-based solution and the benefits offered, despite the associated costs. When analyzing the final product, it is noticeable that as it is, at the moment, it will not be very appealing to the users. The technical setup they go through is a negative point for using the application. However, when the solution described in the future work is implemented, the user will no longer be aware of the blockchain mechanisms involved in the application.

References

[1] Professor Rajiv Lal and Ishan Sachdev. Mobile money services - design and development for financial inclusion. In *Mobile Money Services - Design and Development for Financial Inclusion*, Harvard Business School, July, 2015.

[2] David Shrier, German Canale, and Alex Pentland. Mobile money. In *Mobile Money Payments: Technology Trends*, Massachusetts Institute of Technology, 2017.

[3] GSM. Mobile money percentage. In *State of the Mobile Money Industry in Sub-Saharan Africa*, GSM Association, 2017.

[4] Key trends in the African Mobile Economy. `https://pt.slideshare.net/ATBN/key-trends-in-the-african-mobile-economy`, 2016. [Online; accessed 18 November 2021].

[5] GSM. Mobile money definitions. In *Mobile Money Definitions*, GSM Association, 2017.

[6] M-Pesa. `https://www.vodafone.com/about-vodafone/what-we-do/consumer-products-and-services/m-pesa`. [Online; accessed 24 November 2021].

[7] T. Riley and A. Kulathunga. Agent network structure in kenya. In *Bringing E-money to the Poor: Successes and Failures*, 2017.

[8] Papersoft Initial Page. `https://papersoft-dms.com/`. [Online; accessed 24 November 2021].

[9] Papersoft. `https://papersoft-dms.com/about-us/`. [Online; accessed 24 November 2021].

[10] Feature Waynbo - Offline Mode. `https://waynbo.com/use-cases/agent-network-management-software/`. [Online; accessed 24 November 2021].

[11] Feature Waynbo - Offline Mode Explanation. `https://waynbo.com/customer-onboarding-kyc-compliance/`. [Online; accessed 24 November 2021].

[12] Features Waynbo. `https://waynbo.com/agent-management/`. [Online; accessed 24 November 2021].

[13] FieldPro. https://fieldproapp.com/managing-a-mobile-money-agent-network. [Online; accessed 24 November 2021].

[14] MTN Page. https://www.mtn.com/who-we-are/. [Online; accessed 24 November 2021].

[15] MTN Values. https://www.mtn.com/who-we-are/values/. [Online; accessed 24 November 2021].

[16] Documents of Identification for MoMoAgent. https://www.youtube.com/watch?v=YJj5_W0e6XQ. [Online; accessed 24 November 2021].

[17] Documents of Identification for MoMoAgent 2. https://www.youtube.com/watch?v=uXAqHsgpDOM. [Online; accessed 24 November 2021].

[18] MoMo Agents. https://posbusiness.com.ng/2021/08/28/momo-agent-everything-you-need-to-know/. [Online; accessed 24 November 2021].

[19] Documents of Identification for Waynbo. https://www.youtube.com/watch?v=YH8Ely8IAVc. [Online; accessed 24 November 2021].

[20] MoSCoW Prioritization. https://www.productplan.com/glossary/moscow-prioritization/. [Online; accessed 15 December 2021].

[21] Blockchain Explained. https://www.youtube.com/watch?v=SSo_EIwHSd4. [Online; accessed 16 December 2021].

[22] Wei Cai, Zehua Wang, Jason B. Ernst, Zhen Hong, Chen Feng, and Victor C.M. Leung. Decentralized applications. In *Decentralized Applications: The Blockchain-Empowered Software System*, Natural Sciences and Engineering Research Council of Canada, 2018.

[23] Bhabendu Kumar Mohanta, Soumyashree S Panda, and Debasish Jena. Smart contract. In *An Overview of Smart Contract and Use cases in Blockchain Technology*, IISC, Bengaluru, 2018.

[24] Ethereum Transaction Fee. https://decrypt.co/84866/ethereum-gas-fees-have-risen-2300-since-june. [Online; accessed 21 December 2021].

[25] Cardano Transaction Fee. https://cryptoslate.com/cardano-ada-average-transaction-fees-are-up-1500-in-a-year/. [Online; accessed 21 December 2021].

[26] Polkadot Transaction Fee. https://wiki.polkadot.network/docs/learn-transaction-fees. [Online; accessed 21 December 2021].

[27] Solana Transaction Fee. https://solberginvest.com/blog/how-much-are-solana-fees/. [Online; accessed 21 December 2021].

[28] Ethereum Statistics. https://www.gemini.com/cryptopedia/ethereum-blockchain-smart-contracts-dapps. [Online; accessed 21 December 2021].

[29] Best Ethereum Layer 2 Solutions. https://101blockchains.com/ethereum-layer-2-solutions/. [Online; accessed 27 December 2021].

[30] Polygon Launch. https://zebpay.com/blog/the-rise-of-polygon-matic/. [Online; accessed 28 December 2021].

[31] Arbitrum Launch. https://www.benzinga.com/money/what-is-arbitrum/. [Online; accessed 28 December 2021].

[32] Optimism Launch. https://beincrypto.com/optimism-deems-original-mainnet-launch-timeline-too-ambitious/. [Online; accessed 28 December 2021].

[33] Polygon Scability. https://www.makeuseof.com/what-is-polygon-matic/. [Online; accessed 28 December 2021].

[34] Arbitrum Scability. https://forkast.news/why-arbitrum-rollups-dominating-ethereum-scaling/. [Online; accessed 28 December 2021].

[35] Optimism Scability. https://crypto.writer.io/p/a-quick-dive-into-optimistic-ethereum. [Online; accessed 28 December 2021].

[36] Sidechains vs Rollups. https://blog.infura.io/offchain-protocols-sidechains-and-rollups/. [Online; accessed 28 December 2021].

[37] Polygon vs Ethereum. https://www.makeuseof.com/what-is-polygon-matic/. [Online; accessed 28 December 2021].

[38] Arbitrum vs Optimism. https://www.benzinga.com/money/what-is-arbitrum/. [Online; accessed 28 December 2021].

[39] Top Programming Languages for Android App Development. https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/. [Online; accessed 27 December 2021].

[40] The 10 Best Programming Languages to Learn for Android App Development. https://blog.back4app.com/best-programming-language-to-learn-for-android-apps/. [Online; accessed 27 December 2021].

[41] React Native vs Flutter. https://nix-united.com/blog/flutter-vs-react-native/. [Online; accessed 27 December 2021].

[42] React Native vs Flutter in 2021. https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2021. [Online; accessed 27 December 2021].

[43] Agile Principles. https://www.digite.com/agile/agile-methodology/#agile-principles. [Online; accessed 3 January 2022].

[44] Daily Scrum. https://www.scrum.org/resources/what-is-a-daily-scrum. [Online; accessed 3 January 2022].

[45] Team Gantt. https://www.teamgantt.com/. [Online; accessed 13 January 2022].

[46] User Stories. https://searchsoftwarequality.techtarget.com/definition/user-story. [Online; accessed 12 January 2022].

[47] Quality Attribute Scenario. https://tinyurl.com/39zc8632. [Online; accessed 5 January 2022].

[48] GDPR Regulation. https://www.privacypolicies.com/blog/gdpr-eight-user-rights/. [Online; accessed 23 June 2022].

[49] GDPR. https://gdpr-info.eu/. [Online; accessed 23 June 2022].

[50] Gas Station Network. https://docs.opengsn.org/. [Online; accessed 23 June 2022].

[51] Gas Station Network Definition. https://blog.openzeppelin.com/gsn-the-ultimate-ethereum-onboarding-solution/. [Online; accessed 23 June 2022].

[52] Software Testing. https://www.ibm.com/topics/software-testing. [Online; accessed 26 June 2022].

[53] Usability Testing 101. https://www.nngroup.com/articles/usability-testing-101/. [Online; accessed 26 June 2022].

[54] Test Users. https://www.nngroup.com/articles/how-many-test-users//. [Online; accessed 26 June 2022].

Appendices

Appendix A

1st Semester Planning

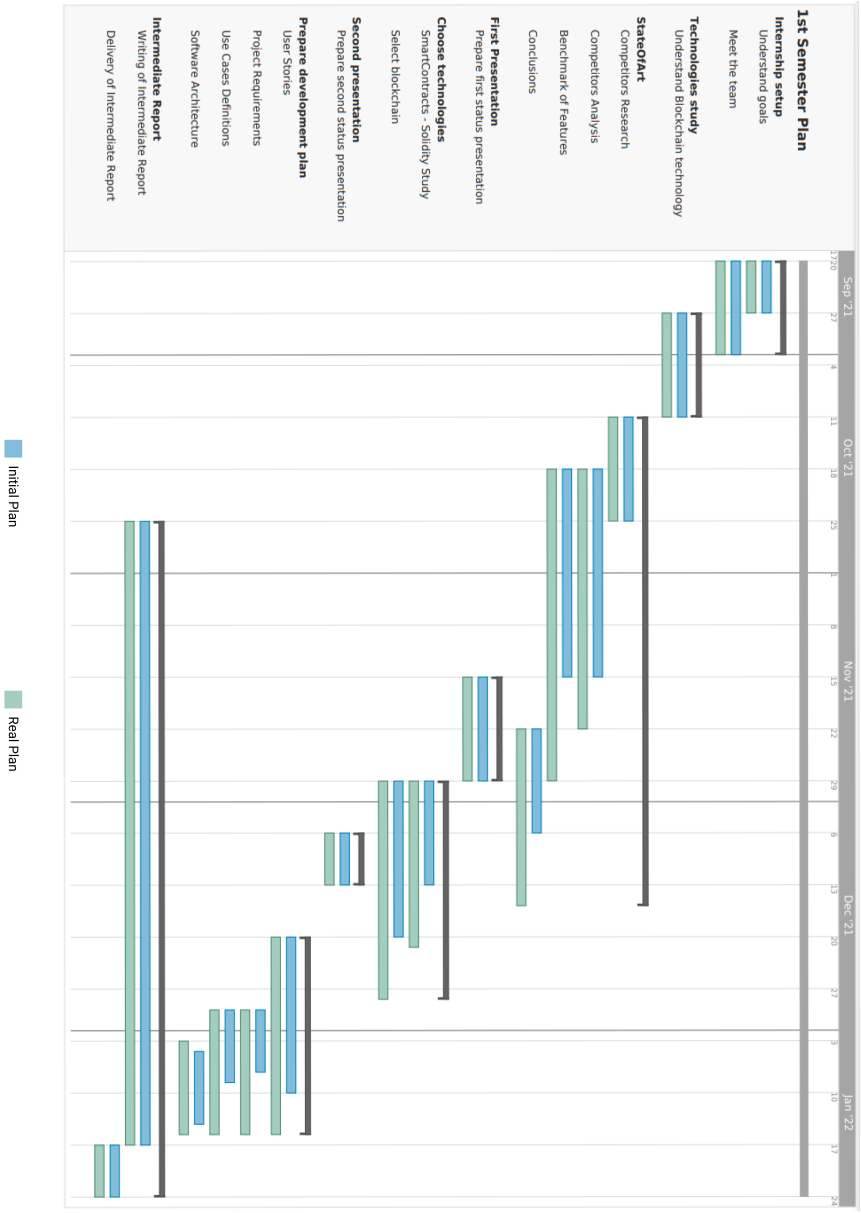

Figure A.1: 1st Semester - Comparison between planned and actual timeline

Appendix B

Software Architecture

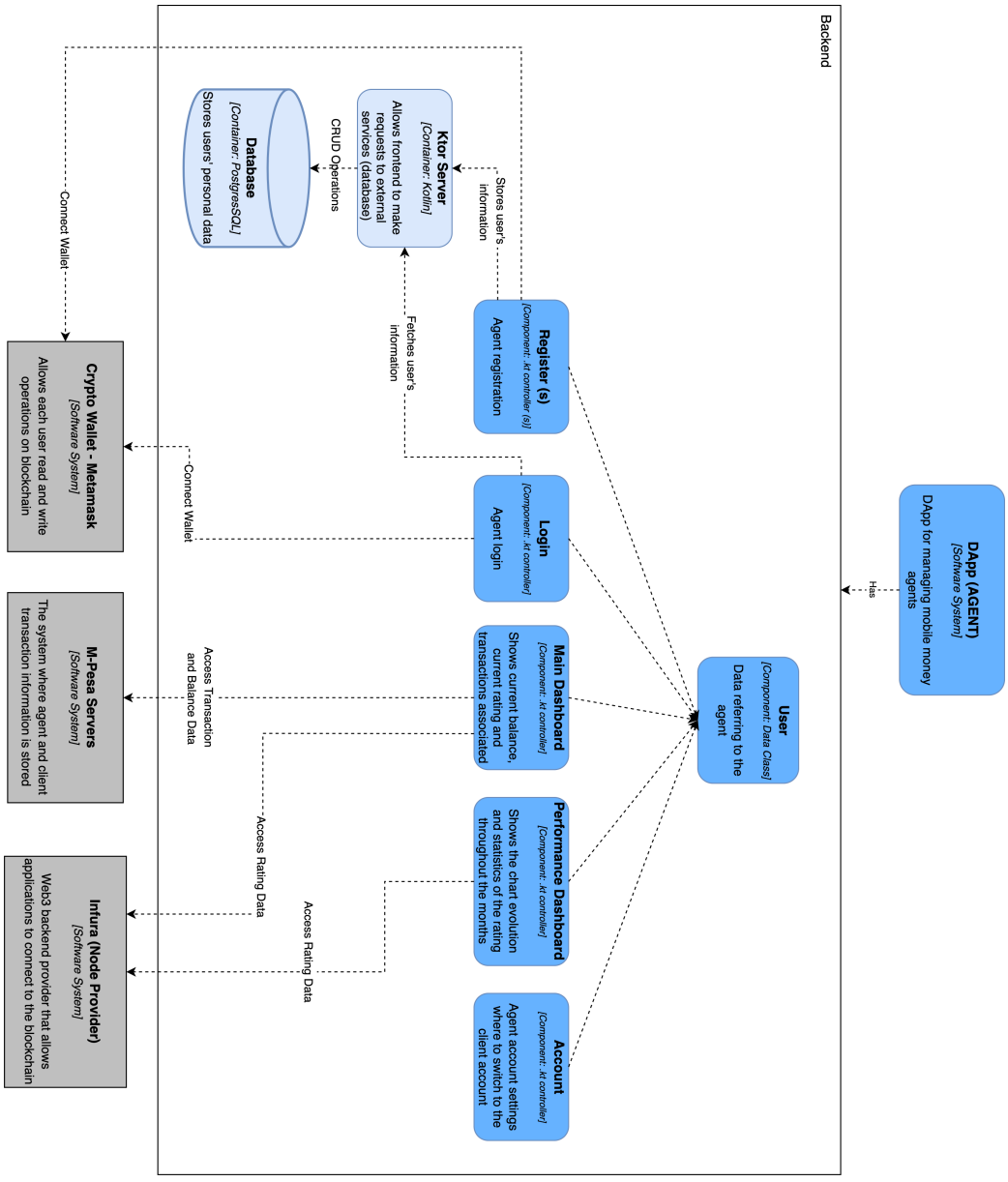

Figure B.1: Component Level - Backend (Agent)

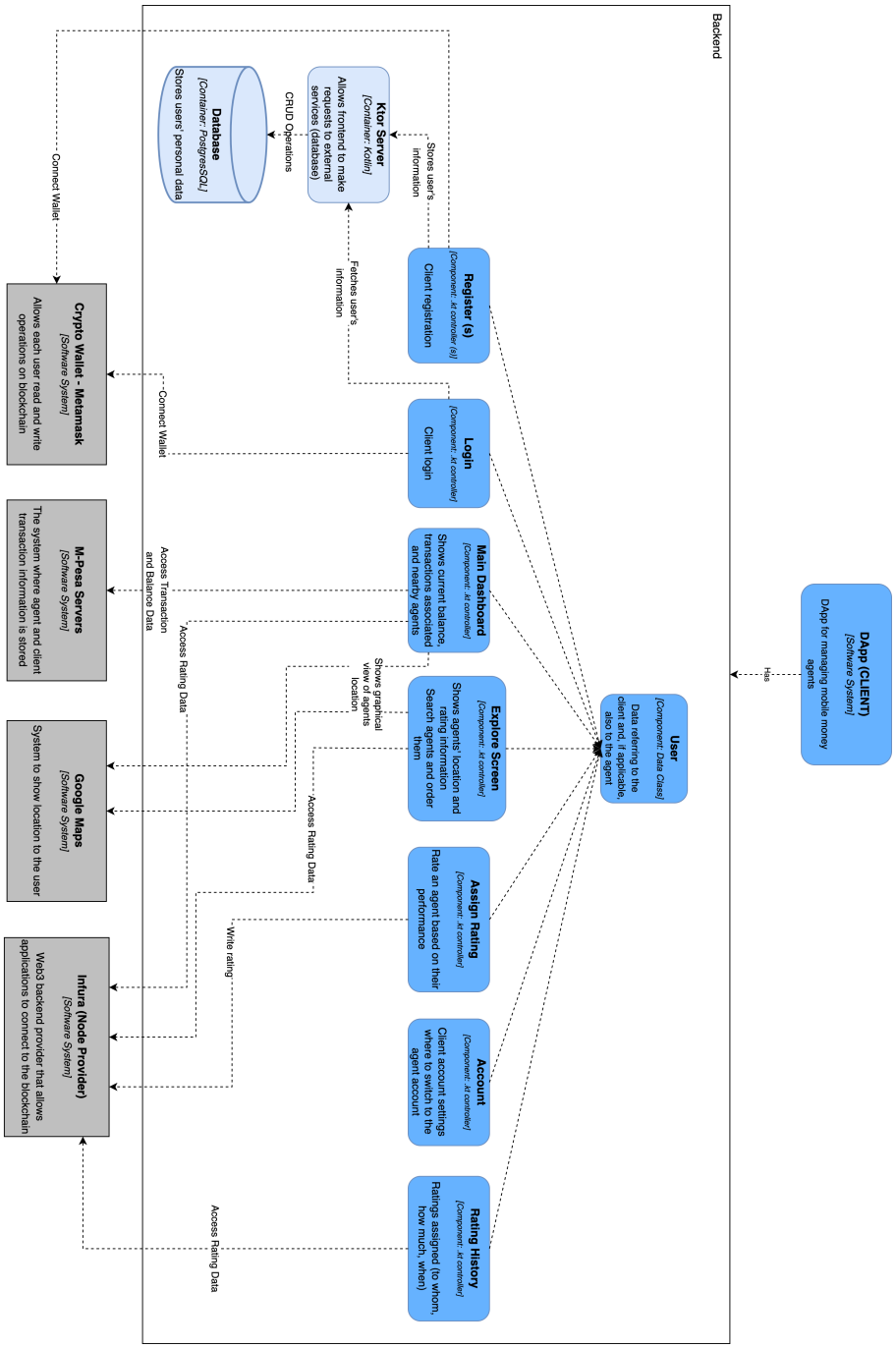

Figure B.2: Component Level - Backend (Client)

Appendix C

Comparative Analysis

Appendix C - Comparative analysis

This analysis aims to compare the cost to the operator using blockchain (using GSN) or an AWS Relational Database.

Blockchain

First, it's fundamental to get an idea of how much the smart contract functions cost [1] – table 1. The goal is to calculate prices based on the target users.

Per-User	Registration	Assign Rating
Transaction fee	0.00489288 MATIC	0.01027644 MATIC
Transaction fee (€)	0.0028 €	0.0058 €

Table 1: Cost of the smart contract functions

Then, there are two options: **run our relay server** or use **other relay servers available in the network**.

Option 1:

"Without significant transaction volume you won't necessarily get a super impressive interest rate, but you will get paid to support decentralization while taking little to no risk. Minimal resources are required to run a relay server so your costs will be low. For example, GCP does not charge you for running a single micro instance." [2]

Meaning that by using Google Cloud Platform and their free tier usage [3][4], it could be possible to use our relay server for our smart contract, we wouldn't pay anything, but in case of failure, we will look for other relay servers in the network.

Option 2:

If there is a need, it is necessary to pay an extra fee to the other relay servers, +60% per function [5]. However, we can also set this for other dapps to use our relay server and profit from it.

Both options can happen, option one as the first resort, but as already mentioned, option two in case of failure of option 1.

The target users of M-Pesa (Kenya) are currently 2.5 million [6], so this comparison is based on these numbers. Also, the average daily operations are 30,063 [6].

Table 2 represents how much the smart contract functions would cost for these numbers.

2.5M	Registration	Assign Rating	Assign Rating (Monthly)
Transaction fee (€)	0.0028 €	0.0058 €	--
Personal relay server	2 500 000 x 0.0028 = 7 000€	30 063 x 0.0058 = 174.4 €	174.4 x 30 = 5232€

Table 2: Cost of the smart contract functions with users

The registration only would happen once, both for agents and clients, and if all clients assigned a rating to the agent would have a medium daily cost of 174,4€. These values are the max it could cost (with our relay server) because the assumption is that all the currently active users would adhere to this. Also, the assumption is that at the end of each operation, the clients assign a rating always.

Google's free tier is limited by time, not by instance. It's possible to use the free e2-micro instance until the number of hours is equal to the current month [3]. Initially, this is a good solution because we are still attracting customers. Therefore, the availability of this solution is practically 100%, so in the initial phase, it will not be necessary to use other relay servers.

By increasing requests and users, other options could be seen to host the relay server, either with the operator's servers or using other services.

Relational Database - Amazon RDS:

The comparison with blockchain will be an Amazon Relational Database due to the ease of use and low prices. The database is relational because of the data that needs to be stored.

Amazon RDS offers a free tier (12 months) [7]: this include 750h per month of database usage and 20GB of memory.
After 12 months, or if the plan needs to be scaled up during the first year, estimates should be made. The result of this can give an idea of what instances are necessary and how many.

Storage:

- Registration: varchar (n) – pubAddress (42bytes)

- Assign Rating (x2) - a client assigns a rating to an agent (writing operation on client rating history and agent rating history)

    - varchar (n) – pubAddress (42bytes)
    - int – howMuch (4 bytes)
    - varchar (n) – date (34 bytes)

Registration (only once):

2 500 000 users x 42 bytes = 105 000 000 = 0.1GB

Assign Rating (daily):

30 063 daily operations x 2 x (42 + 4 + 34) = 4 810 080 bytes = 0.0045GB

Based on these values, there is no need for much storage because the daily storage calculated is 0.0045GB.

Reading operations:

24h per day → reading operations = 86 400sec

If each user, logs at least once:

Agent UI (2 requests):

- 1 request of rating
- 1 request of performance dashboard (rating)

Client UI (2 requests):

- 1 request of rating history
- 1 request (agents rating - explore screen)

Assuming that of 2 500 000 users:

- 2 252 131 are clients only (90.1%)
- 247 869 are clients and agents (9.9%) [8]

Based on these values, it's possible to estimate an average TPS – table 3.

	Clients (only) - TPS	**Agents and Clients - TPS**
Client	(2 252 131 users x 2 requests) / 86 400 sec = 52.13 TPS	(247 869 users x 2 requests) / 86 400sec = 5.74 TPS
Agent	--	(247 869 users x 2 requests) / 86 400sec = 5.74 TPS
Total	52.13 TPS	11.5 TPS
	63.6 TPS	

Table 3: Transactions Per Second (TPS) – reading operations

Writing Operations (Assign Rating):

30 063 operations per day → 12h (working hours) = 43 200sec

(30 063 x 2) / 43 200 = 1.39 TPS + support for bursts

Total	Reading Operations	Writing Operations
	63.6 TPS	1.39 TPS
	65 TPS	

Table 4: Transactions Per Second (TPS) – Total

From the results of the calculations, a price can be estimated in the AWS Pricing Calculator - Appendix D – represented by the table 5.

Per-month	**Blockchain**	**AWS RDS**
Cost	5232 €	237.88 USD = 226.48 €
Performance (Read and Write Operations)	2 second or less [9]	Less than 1 second [10]

Table 5: Price comparation

References

[1] Convert Matic to Euro. https://br.beincrypto.com/converter/matic-network-to-eu. [Online; accessed 22 June 2022].

[2] Running a Relay Server for Fun and Profit. https://docs.opengsn.org/relay-server/tutorial.html#relays-as-an-investment. [Online; accessed 22 June 2022].

[3] Google Cloud – Free Tier usage. https://cloud.google.com/free/docs/gcp-free-tier#free-tier-usage-limits. [Online; accessed 22 June 2022].

[4] Google Cloud – Compute Engine. https://cloud.google.com/compute. [Online; accessed 22 June 2022].

[5] GSN Relay Servers. https://relays.opengsn.org/#maticMainnet. [Online; accessed 22 June 2022].

[6] WIT Software Internal Report (May 2022). [accessed 22 June 2022].

[7] Amazon RDS. https://aws.amazon.com/free/?nc2=h_ql_pr_ft&all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all. [Online; accessed 22 June 2022].

[8] Active M-Pesa Agents. https://techweez.com/2021/10/13/m-pesa-agents-up-by-43-percent-as-the-mobile-money-platform-gains-3-4-million-new-users/. [Online; accessed 22 June 2022].

[9] Polygon transactions. https://permission.io/blog/polygon-faqs/. [Online; accessed 22 June 2022].

[10] AWS IOPS. https://aws.amazon.com/rds/postgresql/features/. [Online; accessed 22 June 2022].

Appendix D

AWS Pricing Calculator

Contact your AWS representative:
https://aws.amazon.com/contact-us/

Export date: **23/06/2022** Language: **English**

Estimate title: **My Estimate**

Estimate URL: **https://calculator.aws/#/estimate?id=87b50cb7a3b60cbb38caafbdcd37dcb145b35299**

Estimate summary

Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	**475.76 USD**	**5,709.12 USD** Includes upfront cost

Detailed Estimate

Name	Group	Region	Upfront cost	Monthly cost
Amazon RDS for PostgreSQL	No group applied	EU (Ireland)	0.00 USD	237.88 USD
Description: Estimation **Config summary**: Storage volume (General Purpose SSD (gp2)), Storage amount (20 GB per month), Nodes (2), Instance Type (db.t3.medium), Utilization (On-Demand only) (100 %Utilized/Month), Deployment Option (Multi-AZ), Pricing Model (OnDemand)				
Amazon RDS for PostgreSQL	No group applied	EU (Ireland)	0.00 USD	237.88 USD
Description: AWS Estimation **Config summary**: Storage volume (General Purpose SSD (gp2)), Storage amount (20 GB per month), Nodes (2), Instance Type (db.t3.medium), Utilization (On-Demand only) (100 %Utilized/Month), Deployment Option (Multi-AZ), Pricing Model (OnDemand)				

Acknowledgement

Appendix E

Web API Documentation

API Database

API and SDK Documentation

Version: 1.0.0

This is a server for accessing user data information.

Users

checkCompositeKey

Checks if wallet logged matches phone number

GET

/user/compositeKey/{pubAddress}/{phoneNumber}

Usage and SDK Samples

Curl | Java | Android | Obj-C | JavaScript | C# | PHP | Perl | Python

```
curl -X GET "https://localhost:8080/user/compositeKey/{pubAddress}/{phoneNumber}"
```

Parameters

Path parameters

Name	Description
pubAddress*	String *the pubAddress that needs to be fetched.* Required
phoneNumber*	Integer (int32) *the phoneNumber that needs to be compared.* Required

Responses

Status: 200 - OK

Schema

▾ **{ []**

Required: MSISDN,firstName,lastName,phoneNumber,pin,pubAddress

pubAddress:	string example: 0x5742fg32v3ja5D
phoneNumber:	integer (int32) example: 987654321
firstName:	string example: John
lastName:	string example: Nanjala
pin:	integer (int32) example: 1111
MSISDN:	string example: 99M992342
shortCodeAgentID:	string example: SH3141515
locationLat:	string

```
                    example: 37.4219506
    locationLong:   string
                    example: -122.078
    additionalInfoClient: string
    additionalInfoAgent: string
}
```

Status: 400 - Empty response

checkPassword

Checks login of user

This is used to check user password

GET

/user/checkPassword/{pubAddress}/{encryptedPass}

Usage and SDK Samples

Curl | Java | Android | Obj-C | JavaScript | C# | PHP | Perl | Python

```
curl -X GET "https://localhost:8080/user/checkPassword/{pubAddress}/{encryptedPass}"
```

Parameters

Path parameters

Name	Description
pubAddress*	String *the pubAddress that needs to be fetched.* Required
encryptedPass*	String *the encryptedPass that needs to be compared.* Required

Responses

Status: 200 - success

Status: 400 - no match

checkPhoneNumber

Checks if phone number is already taken

This is used to new users who enter a phone number

GET

/user/phoneNumber/{phoneNumber}

Usage and SDK Samples

Curl | Java | Android | Obj-C | JavaScript | C# | PHP | Perl | Python

```
curl -X GET "https://localhost:8080/user/phoneNumber/{phoneNumber}"
```

Parameters

Path parameters

Name	Description
phoneNumber*	Integer (int32) *the phoneNumber that needs to be fetched.* Required

Responses

Status: 200 - success

Status: 400 - no match

createUser

Create user

This is used in registration

POST

```
/user
```

Usage and SDK Samples

Curl | Java | Android | Obj-C | JavaScript | C# | PHP | Perl | Python

```
curl -X POST "https://localhost:8080/user"
```

Parameters

Body parameters

Name	Description
userItem	▾ **{ []** Required: MSISDN,firstName,lastName,phoneNumber,pin,pubAddress pubAddress: string example: 0x5742fg32v3ja5D phoneNumber: integer (int32) example: 987654321 firstName: string example: John lastName: string example: Nanjala pin: integer (int32) example: 1111 MSISDN: string example: 99M992342 shortCodeAgentID: string example: SH3141515 locationLat: string example: 37.4219506 locationLong: string example: -122.078 additionalInfoClient: string additionalInfoAgent: string }

Responses

Status: 200 - OK

deleteUser

Delete user

User wants to delete account

DELETE

/user/{pubAddress}

Usage and SDK Samples

Curl | Java | Android | Obj-C | JavaScript | C# | PHP | Perl | Python

```
curl -X DELETE "https://localhost:8080/user/{pubAddress}"
```

Parameters

Path parameters

Name	Description
pubAddress*	String *The pubAddress that needs to be deleted* Required

Responses

Status: 200 - OK

getAgents

Returns agents in database

Search for agents

GET

/user/agents

Usage and SDK Samples

Curl | Java | Android | Obj-C | JavaScript | C# | PHP | Perl | Python

```
curl -X GET "https://localhost:8080/user/agents"
```

Parameters

Responses

Status: 200 - OK

Schema

▾ **[[]**

undefined
]

Status: 400 - empty list

getUserByName

Get user by pubAddress

get the object user without pin parameter

GET

/user/{pubAddress}

Usage and SDK Samples

Curl | Java | Android | Obj-C | JavaScript | C# | PHP | Perl | Python

```
curl -X GET "https://localhost:8080/user/{pubAddress}"
```

Parameters

Path parameters

Name	Description
pubAddress*	String *the pubAddress that needs to be fetched.* Required

Responses

Status: 200 - OK

Schema

▾ **{ []**
Required: MSISDN,firstName,lastName,phoneNumber,pin,pubAddress

pubAddress:	string example: 0x5742fg32v3ja5D
phoneNumber:	integer (int32) example: 987654321
firstName:	string example: John
lastName:	string example: Nanjala
pin:	integer (int32) example: 1111
MSISDN:	string example: 99M992342
shortCodeAgentID:	string example: SH3141515
locationLat:	string example: 37.4219506
locationLong:	string example: -122.078
additionalInfoClient:	string
additionalInfoAgent:	string

}

Status: 400 - Empty response

updateUser

Update user

This is used when client register as an agent

POST

/user/update

Usage and SDK Samples

Curl | Java | Android | Obj-C | JavaScript | C# | PHP | Perl | Python

```
curl -X POST "https://localhost:8080/user/update"
```

Parameters

Body parameters

Name	Description
userItem	▼ { [] Required: MSISDN,firstName,lastName,phoneNumber,pin,pubAddress pubAddress: string example: 0x5742fg32v3ja5D phoneNumber: integer (int32) example: 987654321 firstName: string example: John lastName: string example: Nanjala pin: integer (int32) example: 1111 MSISDN: string example: 99M992342 shortCodeAgentID: string example: SH3141515 locationLat: string example: 37.4219506 locationLong: string example: -122.078 additionalInfoClient: string additionalInfoAgent: string }

Responses

Status: 200 - OK

Suggestions, contact, support and error reporting;

Information URL: https://helloreverb.com (https://helloreverb.com)
Contact Info: maria.vieira@wit-software.com (maria.vieira@wit-software.com)