



UNIVERSIDADE D  
COIMBRA

Pedro Miguel Pinheiro Fernandes

**DEVELOPMENT OF A MOBILE APPLICATION AND BACKEND  
FOR THE SIMULATED USE OF CRYPTOCURRENCIES**

Dissertation in the context of the Master in Informatics Engineering, Specialization in  
Software Engineering, advised by Professor Paulo José Osório Rupino da Cunha, PhD and  
Professor Manuel Paulo de Albuquerque Melo, PhD and presented to  
Faculty of Sciences and Technology / Department of Informatics Engineering.

September 2021

Faculty of Sciences and Technology  
Department of Informatics Engineering

# Development of a mobile application and backend for the simulated use of cryptocurrencies

Pedro Miguel Pinheiro Fernandes

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering advised by Prof. Paulo José Osório Rupino da Cunha, PhD and Prof. Manuel Paulo de Albuquerque Melo, PhD and presented to the  
Faculty of Sciences and Technology / Department of Informatics Engineering

September 2021



UNIVERSIDADE D  
COIMBRA

This page is intentionally left blank.

---

## Acknowledgements

Firstly, I want to thank my family for all the support throughout the execution of the dissertation and for keeping me going forward even when there were setbacks.

To all my friends that lived with me in Coimbra these last years, which we all kept motivating each other to complete this last step of our academic path.

Finally, I would like to express my gratitude to my advisors, Professor Paulo Rupino and Professor Paulo Melo, who were always available to help in every step of this dissertation. And, also, a big thank you to the professors Helder Sebastião, Pedro Godinho and Tiago Sequeira, involved in the project *Descripto*, that gave great advice and recommendations throughout the project.



This page is intentionally left blank.

---

## Abstract

The main objective of these results is to help developing a simulated system of cryptocurrency trading and digital payments, that's compatible with Android and iOS. This system will be used by a group of selected participants for a pseudo-experiment with the main goal of evaluating the preferences of participants related to different cryptocurrencies and their characteristics (volatility, anonymity, trading velocity and settlement time).

In order to develop the system several tasks were conducted. First, a state of the art study was done. It covered the decision about the technology to use for the creation of the frontend side (graphical user interface) where it was chosen the Progressive Web Application solution. An analysis of Graphical interfaces of existing trading and wallet/payment mobile applications was also performed, as well as, evaluated which backend architecture pattern would be more suitable. Then, security considerations for web applications were also described. Secondly, a software requirements document was written, that consists of functional and non-functional requirements, use cases, and mockups. The next step was the definition of technologies to use during the development of the application. Then, the software architecture for the system, that's going to be developed, was designed. After that and before starting the implementation, the risk analysis and the development methodology definition were done. Finally, the system was implemented and tested.

## Keywords

Cryptocurrency, Payment, Trading, Progressive Web Application, Wallet, Cryptocurrency Transactions

This page is intentionally left blank.

---

## Resumo

Estes resultados têm como principal objetivo auxiliar no desenvolvimento de um sistema simulado de troca de criptomoedas e pagamentos digitais, compatível com *Android* e *iOS*. Este sistema vai ser usado por um grupo de participantes selecionados para uma pseudo-experiência com o principal propósito de avaliar as preferências dos participantes relativamente a diferentes criptomoedas e as suas características (volatilidade, anonimato, velocidade de troca e tempo de liquidação).

Para desenvolver o sistema foram feitas uma série de tarefas. Numa fase inicial foi feito o estudo do estado da arte que incidiu particularmente sobre a decisão para a tecnologia a usar para a criação da parte de *frontend* (interface gráfico do utilizador) onde foi escolhida a solução de *Progressive Web Application*. Também é feita uma análise de interfaces gráficas de aplicações já existentes no mercado, tanto de *trading* como de carteira/pagamentos. Por fim, foi analisada qual é a arquitetura mais adequada para a implementação da parte do *backend* e descritos quais os aspectos de segurança a considerar numa aplicação *web*. De seguida, foi elaborado o documento de requisitos que contém requisitos funcionais e não funcionais, casos de uso e *mockups*. O próximo passo foi definir as tecnologias a utilizar durante o desenvolvimento da aplicação. Depois, foi desenhada a arquitetura do sistema a desenvolver. Depois disso e antes de começar a implementação, foi feita uma análise de riscos e definida a metodologia de desenvolvimento. Finalmente, o sistema foi implementado e testado.

## Palavras-Chave

Criptomoedas, Pagamento, *Trading*, *Progressive Web Application*, Carteira, Transações de Criptomoedas

This page is intentionally left blank.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Motivation . . . . .	1
1.3	Objectives . . . . .	2
1.4	Document Structure . . . . .	2
1.5	Work Plan . . . . .	3
<b>2</b>	<b>State of the Art</b>	<b>10</b>
2.1	Analysis of Mobile Payment Systems . . . . .	10
2.1.1	MB WAY . . . . .	10
2.1.2	Paypal . . . . .	10
2.1.3	Trust Wallet . . . . .	10
2.1.4	Conclusion . . . . .	11
2.2	Analysis of Mobile Cryptocurrency Trading Systems . . . . .	12
2.2.1	FOREX.com . . . . .	12
2.2.2	Kraken . . . . .	12
2.2.3	Conclusion . . . . .	12
2.3	Frontend Solutions . . . . .	13
2.3.1	Native Applications . . . . .	13
2.3.2	Cross-Platform Applications . . . . .	14
2.3.3	Comparison of Mobile Development Approaches . . . . .	21
2.4	Backend Solutions . . . . .	24
2.4.1	Types of Databases . . . . .	24
2.4.2	Patterns of Software Architecture . . . . .	25
2.5	Security Considerations . . . . .	29
2.5.1	Authentication . . . . .	29
2.5.2	Access Control . . . . .	30
2.5.3	Session Management . . . . .	30
2.5.4	Data and Input Validation . . . . .	31
2.5.5	Buffer Overflows . . . . .	31
2.5.6	Logging . . . . .	31
2.5.7	Error Handling . . . . .	32
<b>3</b>	<b>Software Requirements Specification</b>	<b>34</b>
<b>4</b>	<b>Selection of Technologies</b>	<b>38</b>
4.1	Progressive Web Application Frameworks . . . . .	38
4.1.1	ReactJS . . . . .	38
4.1.2	AngularJS . . . . .	39
4.1.3	VueJS . . . . .	39
4.1.4	Framework Decision . . . . .	39

---

4.2	Backend Technologies	40
4.2.1	Flask	40
4.2.2	Spring Boot	40
4.2.3	ExpressJS	41
4.2.4	Technology Decision	41
4.3	SQL Databases	41
4.3.1	MySQL	42
4.3.2	PostgreSQL	42
4.3.3	Database Decision	42
4.4	NoSQL Databases	43
4.4.1	MongoDB	43
4.5	Containerization Technology	43
4.5.1	Docker	43
4.6	Message Queue Software	44
4.6.1	Apache Kafka	44
4.7	SMS (Short Message System) Communication API	45
4.7.1	Vonage	45
4.7.2	Plivo	45
4.7.3	Sinch	46
4.7.4	Twilio	46
4.7.5	SMS API Decision	46
<b>5</b>	<b>Software Architecture</b>	<b>48</b>
5.1	C4 Model	48
5.1.1	Level 1	48
5.1.2	Level 2	49
5.1.3	Level 3	50
5.2	Entity Relationship Diagram (ERD)	55
5.2.1	User Service	55
5.2.2	Cryptocurrency Service	56
5.2.3	Market Service	57
5.2.4	Transactions Service	60
5.2.5	Wallet Service	62
<b>6</b>	<b>Development</b>	<b>65</b>
6.1	Development Environment Configuration	65
6.2	Risk Analysis	67
6.3	Development Methodology - Scrum	68
6.4	Implemented Functional Requirements	69
6.5	Client-side Implementation (React.js)	70
6.6	Server-side Implementation (Node.js)	71
6.7	System Implementation and Description	72
6.8	Security mechanisms implemented	86
<b>7</b>	<b>Testing</b>	<b>89</b>
7.1	Functional Testing	89
7.1.1	Unit Testing	90
7.1.2	End-to-End Testing	91
7.2	Load Testing	93
7.3	Usability Testing	94
7.3.1	Test Methodology	94
7.3.2	Test Results	95

<b>8 Conclusion and Future Work</b>	<b>102</b>
-------------------------------------	------------



This page is intentionally left blank.

# Acronyms

- API** Application Programming Interface. 13
- App** Application. 10, 11, 13, 16–24, 26, 29, 38
- Apps** Applications. 11
- IDE** Integrated Development Environment. 11
- iOS** iPhone Operating System. 10–12, 18
- Mac** Macintosh (Apple Computer). 11
- MVC** Model-View-Controller. 29
- OS** Operating System. 10–13, 15, 17
- PWA** Progressive Web Applications. 16–18, 21
- SDK** Software Development Kit. 11
- SIBS** *Sociedade Interbancária de Serviços*. 22
- TLS** Transport Layer Security. 17
- UI** User Interface. 11, 17
- URL** Uniform Resource Locator. 17, 19, 30
- UX** User Experience. 11, 18

This page is intentionally left blank.

# List of Figures

1.1	Expected work plan for the 1st semester . . . . .	5
1.2	Real work plan for the 1st semester . . . . .	6
1.3	Expected work plan for the 2nd semester . . . . .	7
1.4	Real work plan for the 2nd semester . . . . .	8
2.1	Hybrid Application Structure [70] . . . . .	15
2.2	Interpreted Application (using react native framework) Structure [19] . . . . .	16
2.3	Cross Compiled Application Structure . . . . .	17
2.4	Web Application Structure . . . . .	18
2.5	Service Worker life cycle on the first installation [42] . . . . .	20
2.6	Client/Server Pattern high level structure [89] . . . . .	26
2.7	Model-View-Controller Pattern high level structure . . . . .	27
2.8	Microservices Pattern high level structure [49] . . . . .	28
3.1	Payment screen . . . . .	36
5.1	System Context Diagram (Level 1) . . . . .	49
5.2	Container Diagram (Level 2) . . . . .	50
5.3	Component Diagram of the user side API application (Level 3) . . . . .	52
5.4	Component Diagram of the admin side API application (Level 3) . . . . .	54
5.5	Entity Relationship Diagram for User Service . . . . .	55
5.6	Entity Relationship Diagram for Cryptocurrency Service . . . . .	56
5.7	Entity Relationship Diagram for Market Service . . . . .	57
5.8	Entity Relationship Diagram for Transactions Service . . . . .	60
5.9	Entity Relationship Diagram for Wallet Service . . . . .	62
6.1	Authentication service container configuration . . . . .	65
6.2	MySQL users database container configuration . . . . .	65
6.3	Zookeeper and Kafka containers configurations . . . . .	66
6.4	Configuration file for the <i>NGINX reverse proxy</i> . . . . .	67
6.5	Risk analysis table . . . . .	68
6.6	Scrum process [10] . . . . .	68
6.7	Trello board for the project . . . . .	69
6.8	Implemented Functional User Requirements . . . . .	70
6.9	Implemented Functional Administrator Requirements . . . . .	70
6.10	React project structure (general) . . . . .	71
6.11	React project structure ("src" folder) . . . . .	71
6.12	React project structure (components folder) . . . . .	71
6.13	Back-side project structure (general) . . . . .	72
6.14	Back-side project structure ( <i>wallet</i> service directory) . . . . .	72
6.15	React project structure ( <i>wallet</i> service "src" directory) . . . . .	72
6.16	Register screen . . . . .	73

---

6.17	Password hashing implementation . . . . .	74
6.18	Login screen . . . . .	75
6.19	User authorization code . . . . .	75
6.20	View wallet screen . . . . .	76
6.21	View all activity screen . . . . .	76
6.22	Change wallet filters screen . . . . .	76
6.23	Payment menu screen . . . . .	77
6.24	Payment confirmation screen . . . . .	78
6.25	Payment pin insertion screen . . . . .	78
6.26	Send cryptocurrency menu screen . . . . .	79
6.27	Send cryptocurrency confirmation screen . . . . .	79
6.28	Send cryptocurrency pin insertion screen . . . . .	79
6.29	Request cryptocurrency menu screen . . . . .	80
6.30	Request cryptocurrency qr code generated screen . . . . .	80
6.31	Create sell order screen . . . . .	81
6.32	Create buy order screen . . . . .	81
6.33	Create sell order confirmation screen . . . . .	81
6.34	Create sell order pin insertion screen . . . . .	81
6.35	User's open orders screen . . . . .	82
6.36	User's closed orders screen . . . . .	82
6.37	Settings menu screen . . . . .	83
6.38	Edit personal information screen . . . . .	83
6.39	Edit security information (PIN code) screen . . . . .	83
6.40	Help menu screen . . . . .	84
6.41	Help login's section screen . . . . .	84
6.42	Admin panel users screen . . . . .	84
6.43	Admin panel cryptocurrencies screen . . . . .	85
6.44	Admin panel wallets screen . . . . .	85
6.45	Admin panel orderbooks screen . . . . .	86
6.46	Admin panel logs screen . . . . .	86
7.1	Test case for login with invalid credentials . . . . .	90
7.2	Test case for login with valid credentials . . . . .	91
7.3	Results for the login function unit testing . . . . .	91
7.4	Example of a end-to-end testing file for login . . . . .	92
7.5	Example of a end-to-end testing result for login . . . . .	93
7.6	Load testing results for the make a cryptocurrency payment API endpoint . . . . .	94
7.7	Questionnaire first question . . . . .	96
7.8	Questionnaire second question . . . . .	96
7.9	Questionnaire third question . . . . .	96
7.10	Questionnaire fourth question . . . . .	97
7.11	Questionnaire fifth question . . . . .	97
7.12	Questionnaire sixth question . . . . .	97
7.13	Usability testing tasks results (clicks) . . . . .	98
7.14	Usability testing tasks results (difficulty level) . . . . .	98
7.15	Pos-Questionnaire first question . . . . .	99
7.16	Pos-Questionnaire second question . . . . .	99
7.17	Pos-Questionnaire third question . . . . .	99
7.18	Pos-Questionnaire fourth question . . . . .	100
1	Main dashboard . . . . .	113
2	Activity menu . . . . .	113

3	Send money to someone . . . . .	114
4	Set amount of money to send . . . . .	114
5	Security PIN . . . . .	114
6	Payment with NFC . . . . .	115
7	Payment with QR Code . . . . .	115
8	Payment with Number or Barcode . . . . .	116
9	Ask for money from someone . . . . .	117
10	Split the bill with someone . . . . .	118
11	Information menu . . . . .	119
12	Application settings . . . . .	120
13	Personal information . . . . .	121
14	Consent for usage of personal information . . . . .	121
15	Main dashboard . . . . .	122
16	Activity menu . . . . .	122
17	Send money . . . . .	123
18	Request money . . . . .	123
19	Receive money instructions . . . . .	124
20	Receive money . . . . .	124
21	Set amount of money to receive . . . . .	125
22	Receive money with amount set . . . . .	125
23	Settings . . . . .	126
24	Main dashboard . . . . .	127
25	Main dashboard with hidden values . . . . .	127
26	Filter to select dashboard's cryptocurrencies . . . . .	128
27	Dashboard with last notifications . . . . .	128
28	Send cryptocurrency (select recipient) . . . . .	129
29	Confirm send transaction . . . . .	129
30	Receive cryptocurrency (through QR Code) . . . . .	130
31	Receive cryptocurrency with amount set . . . . .	130
32	Buy set amount of cryptocurrency (through third-party provider) . . . . .	131
33	Cryptocurrency dashboard . . . . .	132
34	Cryptocurrency market information . . . . .	132
35	Swap crypto tokens . . . . .	133
36	Confirm swap operation . . . . .	133
37	Exchange crypto tokens . . . . .	133
38	Confirm exchange operation . . . . .	133
39	Settings . . . . .	134
40	Account funds menu . . . . .	135
41	History of trade positions (Open and Closed) . . . . .	136
42	History of trade orders (Open and Closed) . . . . .	136
43	Watchlists menu . . . . .	137
44	More actions for Watchlists Menu . . . . .	137
45	Markets menu . . . . .	138
46	Bitcoin (€) market dashboard . . . . .	138
47	Menu to set alerts of price changes . . . . .	139
48	Notification for alerts set of price changes . . . . .	139
49	Menu to make a trade in the Market (sell) . . . . .	140
50	Menu to make an order in the Market (sell) . . . . .	140
51	Markets menu . . . . .	141
52	Markets menu with one selected . . . . .	141
53	Filter of markets menu . . . . .	141

---

54	Menu with every order, trade and position done . . . . .	142
55	Menu to make a cryptocurrency trade in the Market (buy) . . . . .	142
56	Pair Bitcoin/€ market dashboard . . . . .	143
57	Menu to select the currency pair . . . . .	143
58	Results for the login route unit testing . . . . .	145
59	Results for the logout route unit testing . . . . .	145
60	Results for the password recovery route unit testing . . . . .	145
61	Results for the register route unit testing . . . . .	146
62	Results for the add cryptocurrency route unit testing . . . . .	147
63	Results for the edit cryptocurrency route unit testing . . . . .	148
64	Results for the get all cryptocurrencies route unit testing . . . . .	148
65	Results for the get error logs route unit testing . . . . .	149
66	Results for the get fatal logs route unit testing . . . . .	149
67	Results for the get info logs route unit testing . . . . .	149
68	Results for the get warn logs route unit testing . . . . .	149
69	Results for the add limit order route unit testing . . . . .	150
70	Results for the add market order route unit testing . . . . .	150
71	Results for the cancel order route unit testing . . . . .	150
72	Results for the get buy orders history route unit testing . . . . .	150
73	Results for the get sell orders history route unit testing . . . . .	151
74	Results for the get last buy orders history route unit testing . . . . .	151
75	Results for the get last sell orders history route unit testing . . . . .	151
76	Results for the get user buy orders route unit testing . . . . .	151
77	Results for the get user sell orders route unit testing . . . . .	151
78	Results for the get complete orderbook route unit testing . . . . .	151
79	Results for the get user available cryptocurrencies route unit testing . . . . .	151
80	Results for the get number of actions per day route unit testing . . . . .	152
81	Results for the set number of actions per day route unit testing . . . . .	152
82	Results for the edit wallet quantity route unit testing . . . . .	152
83	Results for the make payment route unit testing . . . . .	152
84	Results for the send cryptocurrency route unit testing . . . . .	153
85	Results for the get fiat balance route unit testing . . . . .	153
86	Results for the get wallet filters route unit testing . . . . .	153
87	Results for the get wallet information route unit testing . . . . .	153
88	End-to-end testing results for the register action . . . . .	155
89	End-to-end testing results for the login action . . . . .	155
90	End-to-end testing results for the forgot password action . . . . .	155
91	End-to-end testing results for the contact us while logged off action . . . . .	156
92	End-to-end testing results for the send cryptocurrency action . . . . .	156
93	End-to-end testing results for the request cryptocurrency action . . . . .	156
94	End-to-end testing results for the add market sell order action . . . . .	156
95	End-to-end testing results for the add limit sell order action . . . . .	156
96	End-to-end testing results for the add market buy order action . . . . .	157
97	End-to-end testing results for the add limit buy order action . . . . .	157
98	End-to-end testing results for the edit personal information action . . . . .	157
99	End-to-end testing results for the edit PIN code action . . . . .	157
100	End-to-end testing results for the invite user action . . . . .	157
101	End-to-end testing results for the add cryptocurrency action . . . . .	158
102	Usability testing page 1 . . . . .	159
103	Usability testing page 2 . . . . .	160
104	Usability testing page 3 . . . . .	161

105	Usability testing page 4 . . . . .	162
106	Usability testing page 5 . . . . .	163
107	Usability testing page 6 . . . . .	164
108	Usability testing page 7 . . . . .	165
109	Usability testing page 8 . . . . .	166
110	Usability testing page 9 . . . . .	167
111	Usability testing page 10 . . . . .	168
112	Usability testing page 11 . . . . .	169
113	Usability testing page 12 . . . . .	170
114	Usability testing page 13 . . . . .	171
115	Usability testing page 14 . . . . .	172
116	Usability testing page 15 . . . . .	173



This page is intentionally left blank.

# List of Tables

2.1	Comparison Table of features for the different mobile development approaches	22
3.1	Functional user requirement for payments . . . . .	34
3.2	Pay for goods use case description . . . . .	35
3.3	Non-Functional Usability Requirement . . . . .	36

This page is intentionally left blank.

# Chapter 1

## Introduction

This document reports all the work conducted for the curricular unit of Dissertation/Internship, advised by Prof. Paulo José Osório Rupino da Cunha and Prof. Manuel Paulo de Albuquerque Melo. The dissertation is part of the Master's degree in Informatics Engineering (MEI) with specialization in Software Engineering, in the Department of Informatics Engineering (DEI) of the Faculty of Sciences and Technologies of the University of Coimbra (FCTUC).

This chapter is divided in five sections that present the context and motivation of the project, list all the objectives for this dissertation, describe the document structure and explain the work plan for its execution.

### 1.1 Context

Nowadays, the emergence of an economy that is not supported by a physical currency is becoming a subject of investigation in several sectors, such as universities, banks and governments. In October of 2008, the first cryptocurrency called Bitcoin appeared, through an article published by someone under the pseudonym Satoshi Nakamoto [56]. Since then, cryptocurrencies have been growing and gaining extreme importance.

This work is part of a bigger project called *DesCripto*. DesCripto's main objective is to purpose desired architectures that can be used, as reference, for central banks or private institutions that look to introduce new cryptocurrencies. The proposals are based on the analysis of the key success factors of cryptocurrencies and the main characteristics of digital coins.

### 1.2 Motivation

*DesCripto* is split into multiple tasks. This document will cover one of them, which seeks to determine the impact of different characteristics in cryptocurrencies, by doing a pseudo-experiment. This experiment will be repeated with breaks in between them in order to help evaluate its consistency. The participants are university students from *Faculdade de Economia da Universidade de Coimbra* and, also, university partners, such as, workers from school cafeteria and canteen.

In order to achieve what was just described, a simulated system that allows cryptocurrency

trading in a small market and digital payments will be created. The developed system will be used by the selected participants for the pseudo-experiment. In each experiment, they will receive a precise amount of coins and numeraire (in euros) that, in an initial phase, can be traded in the market created for the experiment. This period allows to evaluate the preferences of participants related to different cryptocurrencies and their characteristics that can be: more or less volatility, anonymity, trading velocity and settlement time. Based on the prices prevailing in the final days of this period, each cryptocurrency will have a fixed price expressed in units of the cash-equivalent (euros). Then, for the final period of each experiment, participants may no longer trade and can exchange their cryptocurrencies for goods in the school cafeteria or canteen. Even though the results of the experiment are directly related with the first period, which corresponds to the trading phase, the last one is also indispensable, because it allows the participants to understand that they can use the traded cryptocurrencies in their wallet to make purchases of real things.

### 1.3 Objectives

The system to be developed consists of a mobile web application that is compatible with Android and iOS, which is identified by *Crypta Project*. It also specifies the *backend* of the system where cryptocurrency exchanges and transactions are controlled.

In order to begin the development of this system, there are several aspects that need to be addressed, such as, study technologies for both the frontend and backend side of the application, graphical interfaces of existing trading and wallet/payment applications, security considerations for a web application, elaboration of a software requirements document, definition of technologies that will be used for the development and an initial software architecture of the system. For the frontend an approach of development and framework needs to be chosen, for the backend an architecture pattern and a frameworks need do be selected. In the elaboration of the software requirements document was considered the input of the stakeholders (professors) involved, as well as, the graphical interfaces of existing applications analysed. There is also a decision making process that needs to be done in order to define the technologies that are going to be used in the implementation, which will also helps in the next step, that corresponds to the design of the software architecture. Before starting the development, there are two things that must be done, the risk analysis and the development methodology definition. Finally, with everything prepared, the implementation will be done, as well as, its validation through a testing process.

### 1.4 Document Structure

The remaining of this document is structured as follows:

- State of the Art, where in the section 2.3 different approaches are compared for mobile applications development, in the sections 2.1 and 2.2 graphical interfaces of wallet/payment and trading mobile applications are analysed, then in the section 2.4 different software architecture patterns are compared and the last section 2.5, security considerations for web applications are presented.
- Software Requirements Specification, that describes how the software requirements are collected, and section 3 displays an example extracted from the *Software Requirements Specification Document*.

- Technologies Definition, this chapter contains every technology considered to develop the system and the decisions made for each one, with the following sections: section 4.1 - Progressive web application frameworks, section 4.2 - Backend technologies, section 4.3 - Relational databases, section 4.4 - Non-relation databases, section 4.5 - Containerization technology, section 4.6 - Message queue software and section 4.7 - SMS (Short Message System) communication API.
- Software Architecture, which describes the architecture design for the system by using the C4 Model in the section 5.1 and the database design for each service by using entity relationship diagrams (ERD) in the section 5.2.
- Development, which contains the risk analysis and development methodology of the project, the implemented functional requirements, the configuration for the development environment, the client-side and server-side implementation, the system implementation/description and the security mechanisms implemented.
- Testing, this chapter describes all the performed tests, such as, functional testing (unit and end-to-end testing), load testing and usability testing.

## 1.5 Work Plan

For each semester, the project tasks and their respective scheduling are represented through Gantt Charts. Regarding the 1st Semester, two charts are displayed: in the Figure 1.1 the expected work plan for the 1st semester is represented, and the figure 1.2 shows the real work plan done during the 1st semester. Lastly, there is also a chart with the planned work for the 2nd semester in the figure 1.3. Each task has an associated timestamp with a duration in days and a start and an end date which are chronologically represented with a blue bar and its dependencies.

In the first semester, it is possible to observe that some deviations occurred with some of the tasks. In the state of the art study, there was a meaningless delay in the task 1.3 of just one day, there was also a significant delay in the task 1.4 that took two more days to complete and started twelve days late, this can be explained by the delayed started in the collection of the software requirements. The task 3 took twenty nine more days than expected, this can be simply explained by the fact that the task 3.3 was done in late December rather than early December. The tasks 4 and 5 also suffered some delays, because their start dates coincided with other projects from courses of the 1st Semester.

In the third figure 1.3 below, it is displayed a Gantt chart with the expected work plan for the 2nd semester. This will help to better understand in which order every task needs to be done and, approximately, how much time they will take. This chronological order is extremely important for the 2nd semester, because the configuration of environments (developing, testing and exploration) is a must before starting developing the system and writing any type of code.

Lastly, for the second semester, the first conclusion we can take from both Gantt charts (expected - fig. 1.3 and real fig. 1.4) is that one task was removed and another was added. The removed one ("Writing of paper about developed software") was due to the lack of time to completion and, after discussing with thesis advisors, it was not essential for right now. The added one was due to the necessity of having to learn most of the technologies used. Another main difference in both charts is that the expected ends in the 30th of June, whilst the real one ends in the 07th of September, this is because the project deadline was postponed to the special season (in September), mostly because of

development delays. Regarding the deviations between the expected and real chart, there are a couple aspects that can be addressed, such as: the addition of task 2 which delayed the beginning of the development by 22 days and, also, the development in itself took longer than expected cause of its difficulty, especially the market trading service. Finally, the last observable difference is the fact that the testing task was expected to be done after the development, but was actually done along with the development, this is due to the fact that the development methodology was only defined in the second semester, and in this methodology, the development is divided in sprints and in each sprint some features are implemented and tested.

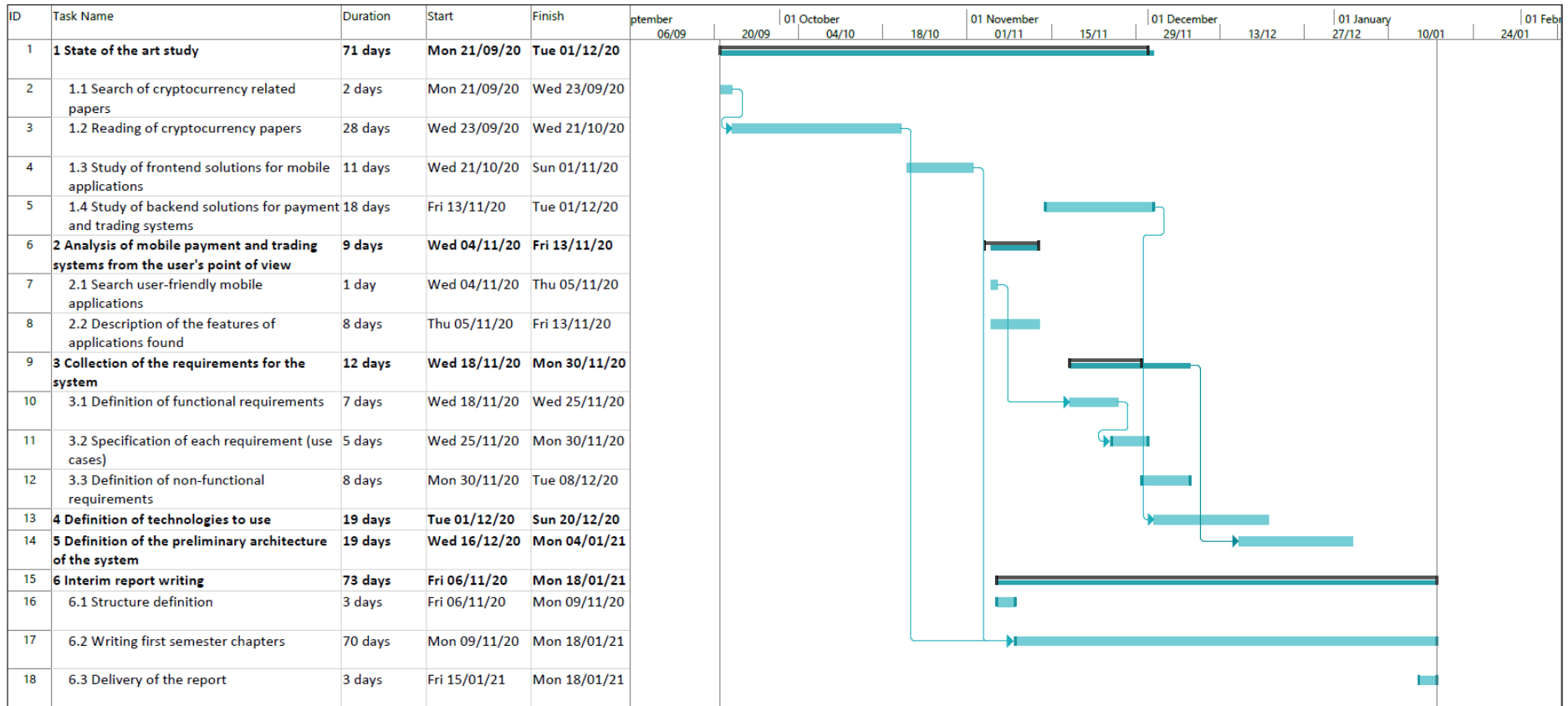


Figure 1.1: Expected work plan for the 1st semester

5



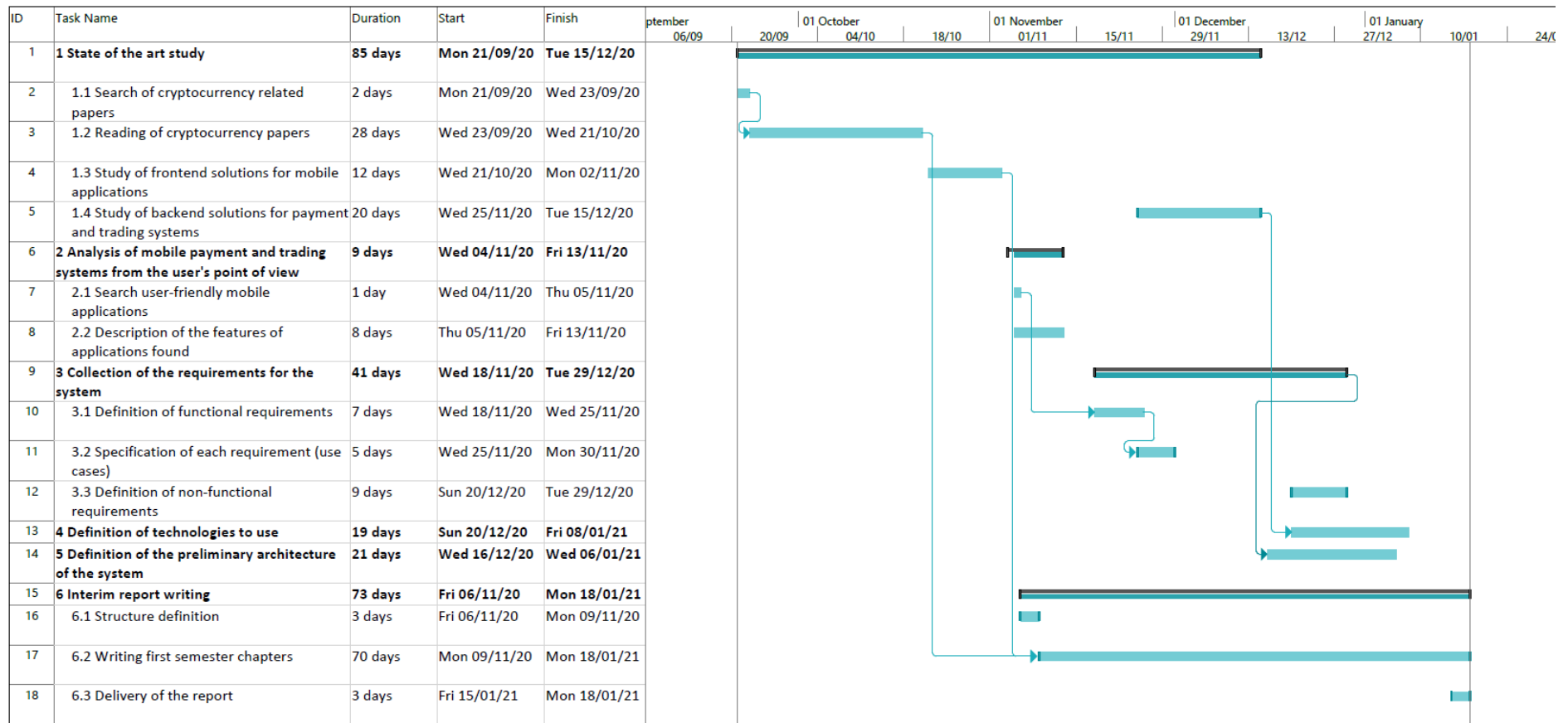


Figure 1.2: Real work plan for the 1st semester

ID	Task Name	Duration	Start	Finish	Qtr 1, 2021			Qtr 2, 2021		
					Jan	Feb	Mar	Apr	May	Jun
1	1 Configuration of development, testing and exploration environments	14 days	Tue 26/01/21	Tue 09/02/21		■				
2	2 Development of the system	81 days	Tue 09/02/21	Sat 01/05/21		■	■	■		
3	3 System and usability testing	30 days	Sat 01/05/21	Mon 31/05/21				■	■	
4	4 Writing of paper about developed software	19 days	Tue 01/06/21	Sun 20/06/21					■	■
5	5 Writing of final thesis	140 days	Tue 09/02/21	Tue 29/06/21		■	■	■	■	■
6	6 Delivery of thesis	1 day	Tue 29/06/21	Wed 30/06/21						■

Figure 1.3: Expected work plan for the 2nd semester

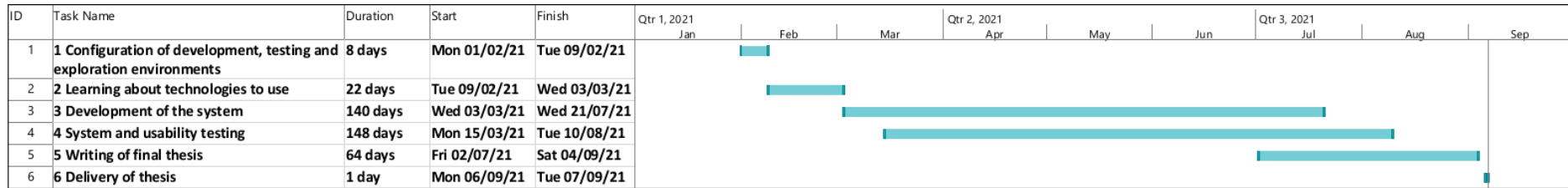


Figure 1.4: Real work plan for the 2nd semester

This page is intentionally left blank.

# Chapter 2

## State of the Art

### 2.1 Analysis of Mobile Payment Systems

This chapter helps better understand how mobile payment systems work. This will assist in the definition of functional requirements and the design of mockups for the application, more specifically, in the wallet/transactions side of the app. For this, screenshots were taken from the different systems and each one was described to understand its importance. In this particular case, the mobile applications studied were MB Way, PayPal and Trust Wallet, being that the only criteria to choose them was its market popularity.

#### 2.1.1 MB WAY

MB WAY<sup>1</sup> is an App that was created in 2014 by SIBS with no associated costs, that allows to make purchases (online and in physical stores), generate credit cards, send and ask money and split the bill. All the screenshot's descriptions are in the subsection MB Way of the Appendix A.

#### 2.1.2 Paypal

Paypal<sup>2</sup> is one of the biggest companies of online payments. It's a financial service that allows the user to pay for purchases by associating a bank account, a debit card or a credit card. He can also send and receive money to/from someone else, as long as he has the email or the phone number. The description of each screenshot is in the subsection Paypal of the Appendix A.

#### 2.1.3 Trust Wallet

Trust Wallet<sup>3</sup> is a decentralized wallet where the user holds the keys to his own cryptocurrency wallet, meaning that only he has control over his funds. It also doesn't keep any personal information and its main goal is to make crypto more accessible. It allows the user to send and receive cryptocurrency, buy and trade crypto and keep track of the

---

<sup>1</sup><https://www.mbway.pt/perguntas>

<sup>2</sup><https://www.paypal.com/pt/webapps/mpp/personal>

<sup>3</sup><https://community.trustwallet.com/t/what-is-trust-wallet/189>

account movements. The description of the screenshots taken is in the subsection Trust Wallet of the Appendix A.

#### **2.1.4 Conclusion**

After analyzing each payment system, there are several aspects that stand out and are fundamental in a mobile application that deals with currency transactions, such as: a main dashboard where a user can check the currencies balance and recent activity, then a payment menu that allows a user to scan a QR code in order to execute a payment, also it should be able to send some currency to someone by using an identifier (e.g. using a phone number or email) and a request money menu where it's possible to generate a QR code which will then help to execute a payment. Finally it needs to have a PIN code in order to verify each operation and a settings menu.

## 2.2 Analysis of Mobile Cryptocurrency Trading Systems

It's also important to comprehend how systems for trading cryptocurrency work, which will help define the functional requirements and mockups for the market trading side of the application. Therefore this chapter describes the functionalities from two mobile applications that allow cryptocurrency trading, that are FOREX.com and Kraken. The criteria to choose these applications was the market popularity and the similarities with the application to be developed.

### 2.2.1 FOREX.com

The forex.com<sup>4</sup> mobile App is a trading platform that allows the user to make trades, analyse markets (through market changes charts), access news and analyses, set up alerts and make deposits and withdrawals of funds. Screenshot's descriptions are in the subsection FOREX.com of the Appendix A.

### 2.2.2 Kraken

Kraken<sup>5</sup> released a mobile App in October 2019. There an user can: view trading and staking portfolio balances, trade with market, limit, stop loss and take profit order types, margin trade and view market price charts. The descriptions of each screenshot is in the subsection Kraken of the Appendix A.

### 2.2.3 Conclusion

Analyzing every single mobile cryptocurrency trading system made it possible to understand which features need be implemented when developing an application like this. The following features are essential for a trading system: A portfolio menu where users can check their open/closed orders in the market and a trade menu to make two possible types of orders, limit and market, in order to buy or sell some kind of cryptocurrency.

---

<sup>4</sup><https://www.forex.com/en-uk/support/faqs/mobile-trading>

<sup>5</sup><https://support.kraken.com/hc/en-us/articles/360049788312-Kraken-Pro-mobile-app-FAQ>

## 2.3 Frontend Solutions

Smartphone applications are part of a market that's constantly growing, in particular for mobile operating systems, such as, Android and iOS. According to Statista [76], in the 3rd quarter (July, August and September) of 2020 there were 2.87 million applications available in the Google Play App Store and 1.96 million in Apple App Store. Both Android and iOS have their own native development structure, meaning that each OS has its own architecture. This native behaviour allows accessing to the platform and hardware features of each mobile device. But, when creating mobile solutions, the main challenge is to provide a solution that works across multiple platforms, being that, going with a native methodology, the company has to create and maintain separate solutions for each OS [66].

With that being said, the focus point would be to create a single application that would work across multiple platforms, and here is where cross platform mobile development comes into play. The ultimate goal of this kind of development is to achieve native performance [98] and create a single application that can be used across different OS, which reduces maintenance and release overheads for different platforms since there's only one code base [66]. A mobile application can be made cross platform with different approaches, some of them focus on the application construction phase to do this, while others focus on the application execution phase [66]. The most common approaches for cross platform that will be covered are: Hybrid, Interpreted, Cross Compiled and Web.

### 2.3.1 Native Applications

Nowadays, the native development approach is one of the most common when it comes to making mobile applications, mainly because there are no concerns about browser behavior and compatibility, and it also takes advantage of the native features of each mobile operating system to deliver the best user experience. But one of the main concerns about the native approach is the fact that, to make a new mobile application, each mobile OS has a different code base, which means that there's a need to have multiple proficient teams capable of developing for each mobile OS or, at least, the main ones such as Android and iOS.

#### Native Approach for Android

When it comes to the native approach for Android, the most used programming languages are Java, Kotlin and C++ [97]. Also, Google provides an Android development tool called Android SDK<sup>6</sup>, which is bundled with Android Studio, the official IDE for Android and, as soon as the native Android application is developed, it can be submitted to the Google Play App Store (official App store for Android applications).

#### Native Approach for iOS

Developing a native application for iOS can be more challenging, mostly because, unlike Android, they can only be developed in a device with Mac OS. It uses Apple tools such as, the official Apple Development Kit called iOS SDK<sup>7</sup> along with the official IDE for

---

<sup>6</sup><https://developer.android.com/studio>

<sup>7</sup><https://developer.apple.com/ios>



iOS developing named XCode<sup>8</sup>, as well as TestFlight<sup>9</sup> which is a tool that helps developers test beta versions of their applications before releasing them in the App Store (official App store for iOS applications).

### Main Advantages and Disadvantages

This way of developing mobile applications has some advantages that really make it popular among companies in this sector, such as [47]:

1. **Performance:** Native mobile applications are fast, reliable and responsive since they are developed specifically for each mobile OS and make use of device's built-in features.
2. **Ability to work offline:** Native mobile applications work even if there's no Internet connection, this is one of the main topics when comparing to a mobile web application that needs internet access, although there's some that can work offline, called Progressive Web Apps. They can make use of a service worker which pre-caches custom offline pages.
3. **User Interface and User Experience (UI/UX):** Users are more likely to enjoy the application and there will be a reduced learning curve because of each mobile OS's specific UI/UX guidelines and standards.
4. **Support from OS App Store:** Google Play and App Store provide complete support to their users, also it's a way of having all the applications for each OS centralized in one single place, making it easier to find for the end users.

Although native development has a lot of positive things, it can also have some negative ones, like [75]:

1. **Multiple codebases:** Each OS will have its own codebase, for instance, iOS applications will not run on any other mobile OS.
2. **Development expenses:** Different codebases for each mobile OS also means that there's a need to have multiple development teams, and each has to be in charge of developing for a specific OS, which can also add cost to the development process.
3. **Frequent Updates:** When a new update is launched, whether it was to fix a number of bugs or simply to add new features, there's always a chance that the users don't update their application, maybe because they didn't notice the update or they don't have enough storage space, which may cause users to eventually delete the application due to unfixed bugs or outdated features.

### 2.3.2 Cross-Platform Applications

#### Hybrid Approach

This approach consists of a mix between native and web approaches. The application is essentially written using web technologies (HTML, CSS and JavaScript) and embed HTML5

---

<sup>8</sup><https://developer.apple.com/xcode>

<sup>9</sup><https://developer.apple.com/testflight>

applications inside a native container (UIWebView in iOS and WebView in Android)[98]. This approach uses the device's browser engine which renders and displays the HTML, whilst the device-specific hardware is accessed through native APIs (Application Programming Interface). Unlike web applications, these applications need to be downloaded and installed from an official store[66]. Some examples of frameworks to make hybrid applications are Cordova, PhoneGap and Ionic. The following Figure 2.1 represents an hybrid application diagram:

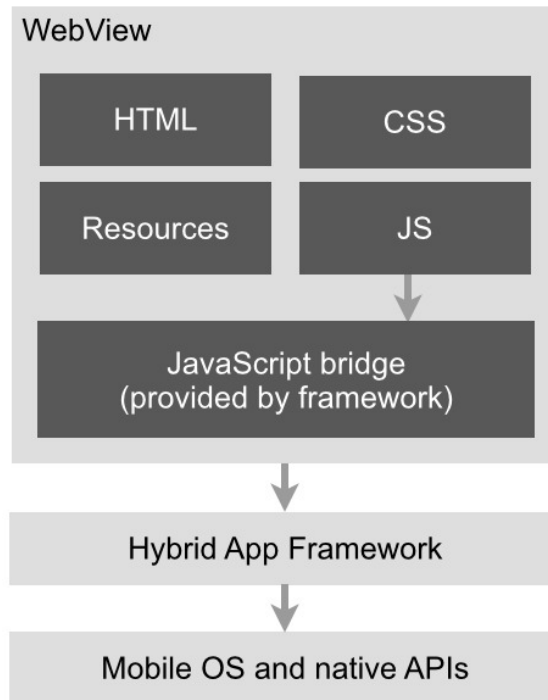


Figure 2.1: Hybrid Application Structure [70]

### Main Advantages and Disadvantages

The main advantages of Hybrid Applications are[33][66]:

1. It's distributed through the platform application store.
2. Because it uses native features, the user interface can be reused across different platforms.
3. It can access the device features.

On the other hand, it has some disadvantages like[33][66]:

1. Worse performance than a native application because it executes in the browser engine.
2. The user interface lacks the look and feel from a native application.
3. Since operating systems operate differently, design issues might be difficult to solve, because the solution can be different from OS to OS.

## Interpreted Approach

In this approach, the application code is deployed to the mobile phone and will be interpreted there. The source code is interpreted at runtime, by an interpreter, across distinct operating systems. Then, the interpreted App will access the native while the native features will be available through an abstraction layer (varies with the framework used) that interacts with the native APIs[66]. The users of this kind of App interact with platform-specific native user interface components. The application logic is implemented in a platform-independent way, using languages like XML, Java, JavaScript, etc[98]. Some of the most common frameworks that use this approach are React Native and NativeScript. An application made with the React Native framework as the follow structure (Figure 2.2):

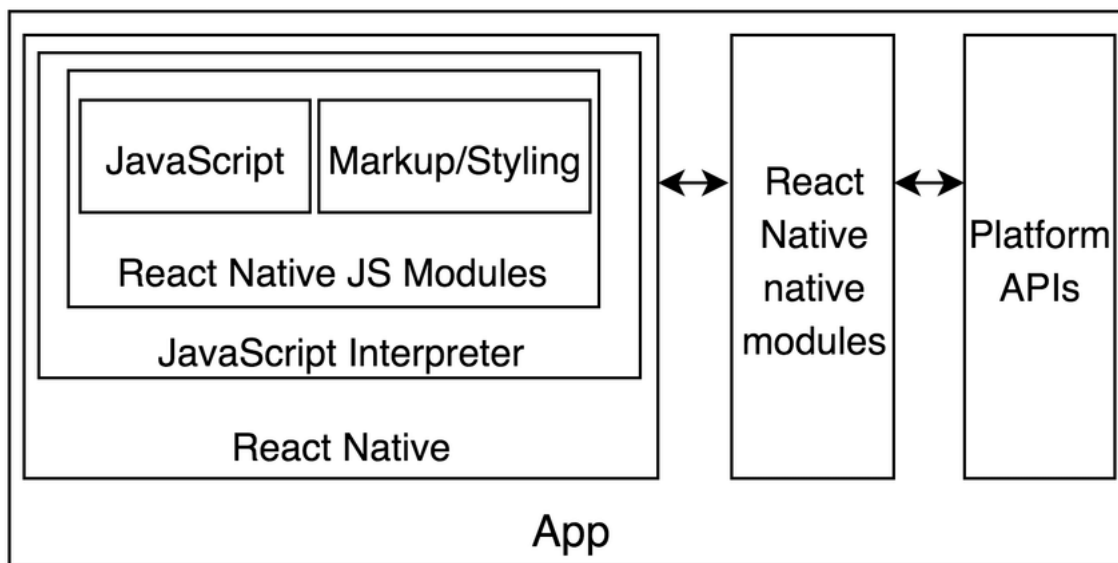


Figure 2.2: Interpreted Application (using react native framework) Structure [19]

## Main Advantages and Disadvantages

The main advantages when using an interpreted approach are[33][66]:

1. An interpreted application has the look and feel similar to a native application.
2. There's only one source code for all the target platforms.
3. It's distributed through the platform application store.

On the other side, the major disadvantages are[33][66]:

1. The development heavily depends on the set of features provided by the chosen framework.
2. The performance of the application might be a bit lower, because the code is interpreted in runtime and needs to be done every time.
3. The user interface reusability also depends on the framework selected for development.

## Cross Compiled Approach

In this approach there's a cross compiler that converts the source code from a common language into native binaries for each of the support platforms, producing real native applications[20]. This approach depends heavily on the efficiency and reliability of the cross compiler[66]. The most common framework for this approach is Xamarin. The diagram in the Figure 2.3 represents the structure of a cross compiled application.

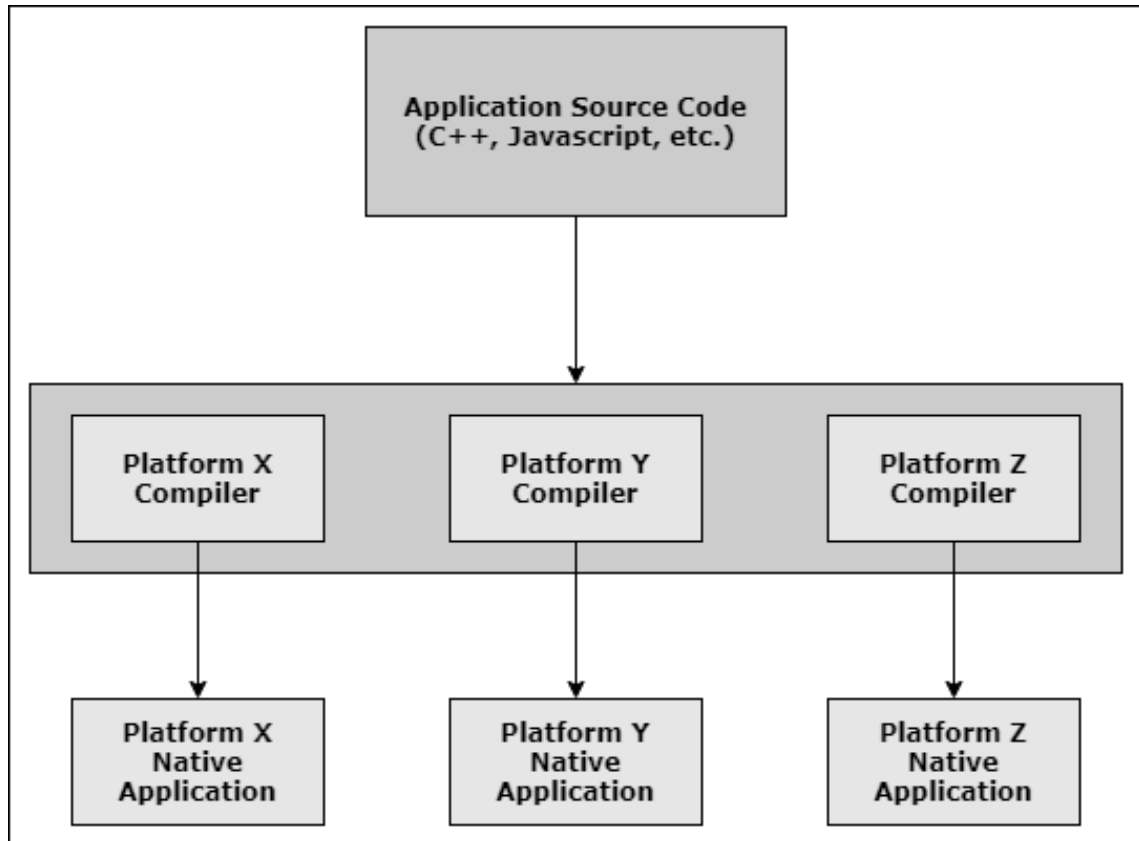


Figure 2.3: Cross Compiled Application Structure

## Main Advantages and Disadvantages

Using this approach has the following advantages[33][66]:

1. Has the same features as a native application does.
2. Can make use of the native user interface components and device hardware/software.
3. There's only one application source code that gets cross compiled into an application for each platform.
4. Its performance is very close to the native approach.

However it has a couple disadvantages, such as[64][33]:

1. The user interface and platform specific features (camera, location, notifications, etc.) can't be reused, because they are specific for each OS and the way they are utilized change from platform to platform.

2. Applications that are developed using this approach are typically larger than native ones.

### Web Approach

The Web Approach consists of developing web applications that are designed to run in the mobile web browser. These are mostly developed with HTML, CSS and JavaScript, also they are browser based which grants platform independency and the data is store in a server[66]. There are also some features that are available with HTML5, such as, getting data from mobile sensors (accelerometer, gyroscope, etc.) and have access to user and device information (e.g. contact list)[25]. An Web Application is structured as follows (Figure 2.4):

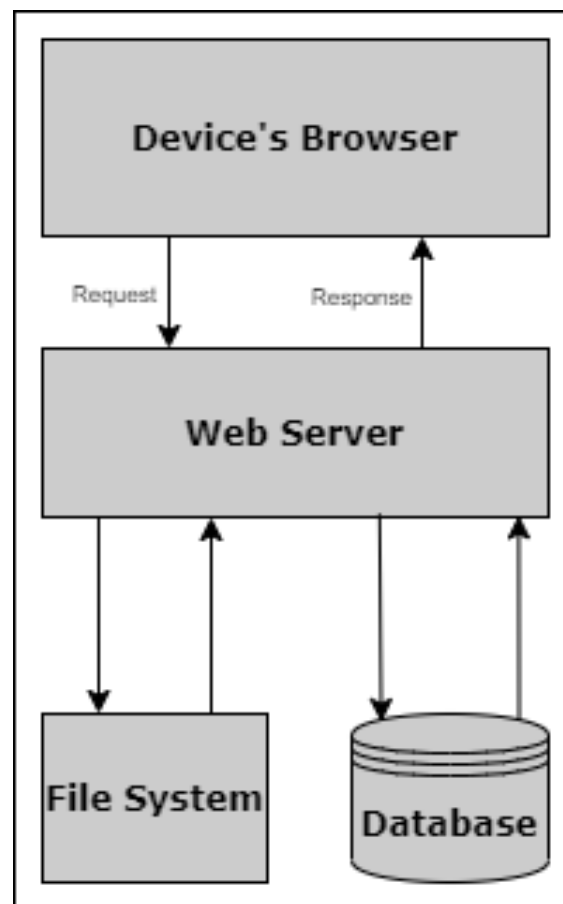


Figure 2.4: Web Application Structure

### Progressive Web Applications

The term *Progressive Web Apps (PWA)* was introduced in 2015 by Alex Russell in a blog post [72] that stated initial design ideas for this new type of Web App with the main goal of filling the gap between web applications and native/cross-platform applications [20]. These characteristics initially identified by him are still considered relevant for *PWA* nowadays [69]:

- **Responsive:** To fit any form factor, i.e. users are able to use the application on

any screen size and everything is available at any viewport size. Even though this concept is not particular of PWAs, it is essential for web applications that want to provide a good user experience in mobile devices[20].

- **Connectivity independent:** This is one of the main differences from regular Web App. PWAs are progressively-enhanced with Service Workers (technology detailed further ahead) to let them work offline, meaning that, during a service worker install event, custom offline pages can be pre-cached for later use.
- **App-like-interactions:** A good PWA can act, look and feel like a native application, by adopting a Shell + Content application model to replicate their behaviour (navigation and interaction)[20].
- **Fresh:** Service Workers help fetching new content whenever is needed, either when the user opens the app or during a background synchronization task. This ensures that the App is always up-to-date[20].
- **Safe:** Served via TLS, that's a Service Worker requirement, for optimal communication security[20].
- **Discoverable:** PWAs are identifiable as "applications" thanks to W3C Manifests and Service Worker registration scope allowing search engines to find them.
- **Re-engageable:** PWA can access the re-engagement UI of the OS, such as, push notifications. This concept is important to draw users into using the app again.
- **Installable:** It's possible to save the application to the home screen through browser prompts, allowing users to install applications without the inconvenience of an App store.
- **Linkable:** PWA don't require installation and are easy to share via URL.

In order to achieve some of the characteristics previously mentioned, there are technologies that need to be implemented, such as:

- **Service Workers:** This technology consists of a JavaScript script that executes background operations since it can't interact with the Document Object Model[42] (DOM, that's a programming interface for HTML documents[46]). It also works as an application-level network proxy, that allows developers to control caching of data and assets for offline availability, background synchronization and registering for push notifications[42]. The following figure represents the life cycle of a service work on the first installation.

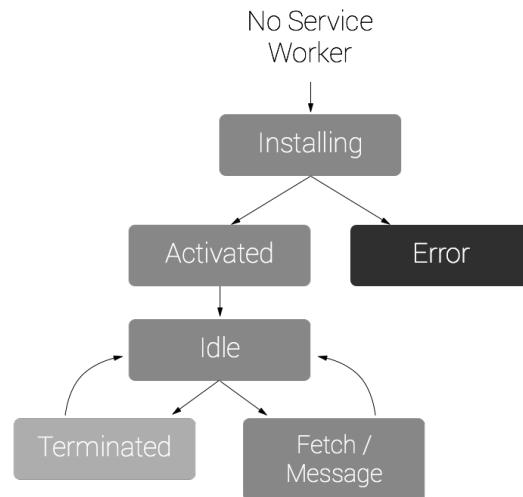


Figure 2.5: Service Worker life cycle on the first installation [42]

- **Application Shell (AppShell):** This component is the first user interface to render, that looks to avoid the feeling of a slow App. It consists of static assets and interface components (e.g. navigation bars, toolbars and other elements developed with HTML, CSS and JavaScript), which means that they don't depend of anything external to render. The AppShell when well developed and optimized provides an optimal user experience[20].
- **Web App Manifest:** The manifest provides information in a JSON file and can be used to configure PWAs, that then can be installed in a computer or mobile device[84]. This file contains the following info: name, author, icon(s), version, description, list of necessary resources, splash screen images, background colors, display types and more that are less relevant and not commonly used[20][84].

Building this kind of progressively-enhanced web applications has some main **advantages** [40] over native applications, such as:

1. **Cheaper:** PWA are cheaper than other applications, because they are developed just once, when compared to creating applications separately for Android and iOS.
2. **Small size and fast download:** PWA don't take nearly as much space as native applications, making them also faster to download.
3. **Offline mode:** PWA can be cached by the web browser, meaning that they can also be used when offline.
4. **Require no installation or manual updates:** With PWA, there's no need to visit the Play Store or App Store. Users can just use the application by typing the URL in the web browser, also when there's a new update, users don't need to manually install it.

But they also have some **disadvantages** [40] that can have some negative impact when choosing the right solution to build applications, like:

1. **No access to App stores:** Meaning that PWA don't have the Play Store or App Store exposure that native applications do.

2. **Less functionalities:** PWA have fewer features when compared to native applications. As a result, UX can be lost, which may also cause a drop in the number of potential users.
3. **Issues with older devices:** Since PWA have been around for just a few years, some older mobile devices with outdated web browsers might not be able to support them.

### 2.3.3 Comparison of Mobile Development Approaches

In order to comprehend which mobile development approach is the most suitable for this project, a comparative analysis of native, hybrid (Ionic framework), cross compiled (Xamarin framework), interpreted (NativeScript framework) and web approach is presented. The analysis was adapted from several scientific papers ([59], [29], [20] and [98]), mainly the following set of features along with the metric values used:

1. **Access to device native features[59][29]:** Represents the level of accessibility to the device native features that an approach offers. **Possible values:** Very high, high, medium, low, very low.
2. **App Size[29]:** Space needed to install an application. **Possible values:** Very high, high, medium, low, very low.
3. **Battery use[29]:** One important factor to take into consideration is battery consumption of each approach. **Possible values:** Very high, high, medium, low, very low.
4. **Distribution[20][29]:** An application can be distributed through App Stores or directly in a web browser. **Possible values:** App Store or URL (Web Browser).
5. **Initial boot time[29]:** Since this project will deal with payment, it's important to consider how much time an application takes to boot. **Possible values:** Very high, high, medium, low, very low.
6. **Installable[20]:** Whether it's possible to install an application on a mobile device or not. **Possible values:** Yes or No.
7. **Offline Usage[20][29]:** If an application can be used offline or only with internet connection. **Possible values:** Yes or No.
8. **Speed and Cost of Development[29]:** This feature depends on the code reusability, because sometimes there's a need to develop more than one codebases. **Possible values:** Very high, high, medium, low, very low.
9. **User Experience (look and feel)[29]:** This is directly related with the level of satisfaction an user feels when using an application. **Possible values:** Very high, high, medium, low, very low.
10. **User Interfaces[98][29]:** Evaluates if an application interface uses native or web components. **Possible values:** Native or Web.
11. **User-perceived performance[98][59][29]** Represents the performance perceived by an end user when using an application, the main factors that affect this feature are the loading time and execution speed). **Possible values:** Very high, high, medium, low, very low.



Features	Mobile Development Approaches					
	Native		Hybrid (Ionic)	Interpreted (NativeScript)	Cross Compiled (Xamarin)	PWA
	Android	iOS				
Access to Device Native Features	Very High	Very High	Low	Medium	Medium	Very Low
App Size	Very Low	Medium	Medium	Very High	Medium (Android) Very High (iOS)	Very Low
Battery Use	High	Very Low	Low	Medium	High	Low
Distribution	App Stores	App Stores	App Stores	App Stores	App Stores	URL (Web browser)
Initial Boot Time	Very Low	Low	High	High	Medium	Very Low
Installable	Yes	Yes	Yes	Yes	Yes	Yes <sup>a</sup>
Offline Usage	Yes	Yes	Yes	Yes	Yes	Yes
Speed and Cost of Development	Very High	Very High	Low	Medium	High	Very Low
User Experience	Very High	Very High	Medium	High	High	Low
User Interfaces	Native	Native	Web	Native	Native	Web
User-Perceived Performance	Very High	Very High	Medium	Medium	Medium	Low

Table 2.1: Comparison Table of features for the different mobile development approaches

(a) With Progressive Web Applications there's the possibility to implement the feature "Add to home screen", this allows the user to "install" a Web App and use it as a native installed application. To enable this, the website must have a valid Web App Manifest and load its assets through a Service Worker[6].

**Mobile Development Approach Decision**

After analysing the comparison table (Table 2.1), the right mobile development approach, considering the context of this project, is the PWA (Progressive Web Application). Because, even though, it's not the one with the best user-perceived performance and user experience, and doesn't have as much access as the others to native features, it has the following advantages that are much better for this application, in particular: very low size, battery usage, initial boot time, speed and cost of development. This is particularly important, because one of the main objectives of this App is to make and receive payments, this means that, if it does not consume much battery, someone who is making transactions frequently does not need to be repeatedly charging the phone, also, whether it is to pay or receive, if the boot time is fast, the process becomes much faster, which is really good in a time depending environment (e.g. school cafeteria). On the other side, very low speed and cost of development is also very important, considering the small time window to make the application (around five months) and the number of developers (in this case, only one).

## 2.4 Backend Solutions

Backend is an extremely important aspect to consider when developing an application as it refers to the server side of an App. This consists of the side that the user doesn't see. It's responsible for storing data, accessing third-party services, handle the application logic, etc. It communicates with the frontend by sending and receiving information that is then displayed in a web browser (for a web application). In order to build the backend side, there's two main aspects that need to be taken into consideration: the type of database (Relational or Non-relational) and the pattern for the software architecture.

### 2.4.1 Types of Databases

#### Relational Databases (SQL)

This type of database is based on the Relational Model, that represents the database as a collection of relations. This database stores and provides access to data that is related with one another. This relation is represented by a table of values, where every row consists of a collection of related data values [87][67].

#### Main Advantages and Disadvantages

The main advantages of Relational Databases are:

- **Simplicity:** This type of database uses tables which makes it the most simple strategy, because it does not require any complex structure or querying processes. It's perfectly possible to handle this database with simple queries [68].
- **Data Accuracy:** With this database there can be multiple tables related to each other by using a primary and foreign key, which makes the data to be non-repetitive [68].
- **Data Integrity:** One of the main features of this model is data integrity. Strong data typing and validity certify that data checks with acceptable ranges and required data is present. This also helps ensuring data accuracy and consistency [79].
- **Flexibility:** This model is also scalable and extensible, meaning that the system will be flexible to increasing amounts of data [68][79].
- **High Security:** It's possible to improve security, because the data is split into tables, and tables can be tagged as confidential to control access [68].

This model also has some main disadvantages, such as:

- **Speed:** It can be slow when compared to a NoSQL model [79].

#### Non-Relational Databases (NoSQL)

On the other side, Non-Relational Databases are different from traditional relational databases because they don't store data in tables. This model has a variety of database types, such

as, document (the data is stored in a file similar to JSON (JavaScript Object Notation)), key-value (the data is stored as a key-value pair), wide-column (data is stored in tables, rows and dynamic columns, and the difference to a relational database is that each row doesn't need to have the same columns) and graph (data is stored in nodes and edges, it's usually used with highly connected databases, like social networks applications) [94]. They provide flexibility and scalability with large amounts of data.

### Main Advantages and Disadvantages

The major advantages of a non-relational database are [12]:

- **Easy Management:** Because there's no structure required, they are easier to manage.
- Easily scalable.
- Doesn't require database administrators.
- Faster and more flexible when compared to a relational database.

It also has some disadvantages, such as:

- Doesn't have a standard query language [12].
- There's no standard interface to manage the database [12].
- It's difficult to maintain [12].
- **Consistency:** NoSQL databases don't support ACID (Atomicity, Consistency, Isolation, Durability) natively, which could also compromise consistency [50].

### Database Decision

Considering the context of this project and the advantages/disadvantages of both types of databases, it makes sense to choose a relational database for everything except the logs because, even though they are not as fast as a non-relational database, they have three highly important advantages that are, data accuracy, data integrity and high security, also NoSQL databases don't support ACID (Atomicity, Consistency, Isolation, Durability) natively. Considering the context of this project, we are dealing with an application that represents a wallet with monetary value, meaning that security and integrity are indispensable. In order to save the logs it makes sense to use non-relational database because it grants easy scalability along with faster operations, which is the most adequate for logging since every action executed in the backend will be logged, making it important to have good scalability.

## 2.4.2 Patterns of Software Architecture

### Client-Server Pattern

In this pattern, the system is segregated into two main components: service user (clients) and service providers (servers), that are connected via network or internet connection.

The server side continually listens for requests from the clients, when one is received, it's processed, and then a response is sent back to the client. A server must be multi-threaded in order to serve multiple clients at the same time, it can also be classified as stateful or stateless [26]. A stateful server store a session state, meaning that it can keep client data from one request to the next. On the other hand, stateless servers don't keep any client data [30]. Applications that follow this pattern contain three functional units: Presentation Logic (interface that a client interacts with), Business Logic (where requests and responses are processed) and Data (where information is stored). This pattern can also be 2-tier and 3-tier, in other words, in a 2-tier application the business logic can be combined with the presentation logic in the client side or with the Data layer. In the 3-tier one, the business logic is separated from the presentation logic and the data layer, representing a middle tier [13].

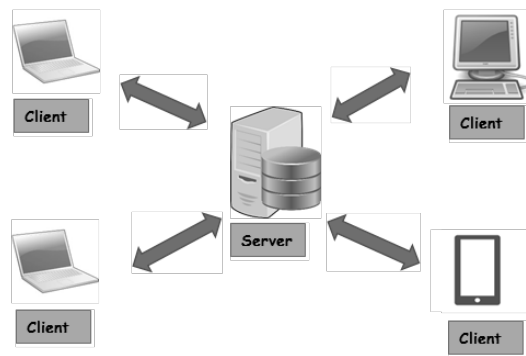


Figure 2.6: Client/Server Pattern high level structure [89]

### Main Advantages and Disadvantages

For this software architecture pattern the main advantages are [27]:

- **Centralization:** This pattern is integrated with centralized control and all the information is centralized in one place, making it easier for the administrator to control the system.
- **Scalability:** Systems that follow this pattern are easily scalable, because clients and servers capacity can be increased separately.
- **Maintenance:** It's easy to make changes (updates/upgrades/fixes) to the system, without the clients noticing.

The major disadvantages of this model are [3]:

- **Traffic Congestion:** The server may get overloaded if a big number of clients make requests at the same time.
- **Robustness:** Since it's centralized, it has a single point of failure, which means that, if a server fails, the client requests will not succeed, and the App stops working completely.
- **Cost:** Building and maintaining a system based on this pattern can be very costly.

- **Security:** The communication between clients and servers can lead to some security problems. It's particularly vulnerable to Denial of Service (DOS) attacks, because of the traffic congestion previously mentioned.

## Model-View-Controller (MVC) Pattern

The Model-View-Controller is a software architecture pattern composed by three main logical components: the Model, the View and the Controller. The Model component is responsible for storing and managing data, and often corresponds to a database. The View component often represents a graphical user interface, it contains all the functionalities that directly interact with the user. Then the Controller component is where all the logic of the application is and bridges the gap between the View and the Model, the controller receives an input from the user (request), processes it, communicates with the model to grab the data and then responds back to the View that displays the data [54][92].

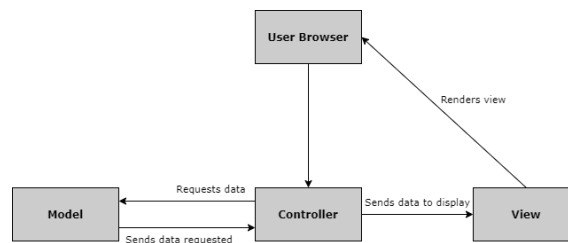


Figure 2.7: Model-View-Controller Pattern high level structure

## Main Advantages and Disadvantages

The model-view-controller pattern has the following advantages [92]

- **Simultaneous development:** Developers can work simultaneously on the model, view and controller, because they are separated.
- **High Cohesion:** With MVC, related actions can be grouped on a controller together. Views can also be grouped.
- **Loosely Coupled:** In this pattern, there is low coupling among models, views and controllers.
- **Modifiability:** Because components are separated, it's easier to make modifications to the system.

This pattern also has some disadvantages, such as:

- **Learning Curve:** MVC can use multiple different technologies, meaning that learning every single one to develop an application can take a long time [92].
- **Frequent Model Changes:** If the model is frequently changed, it can result in excessive updates of the views as well, consequently, changing controllers too [80].

## Microservices Pattern

Microservices software architecture pattern is a form of service-oriented architecture that consists of a collection of loosely coupled services. Each microservice can be created independently and even in different programming languages from one another. In this pattern there's an API gateway that functions as the entry point for users. Instead of calling the services directly, the users call an API gateway that then forwards that call to the right service on the backend side [52].

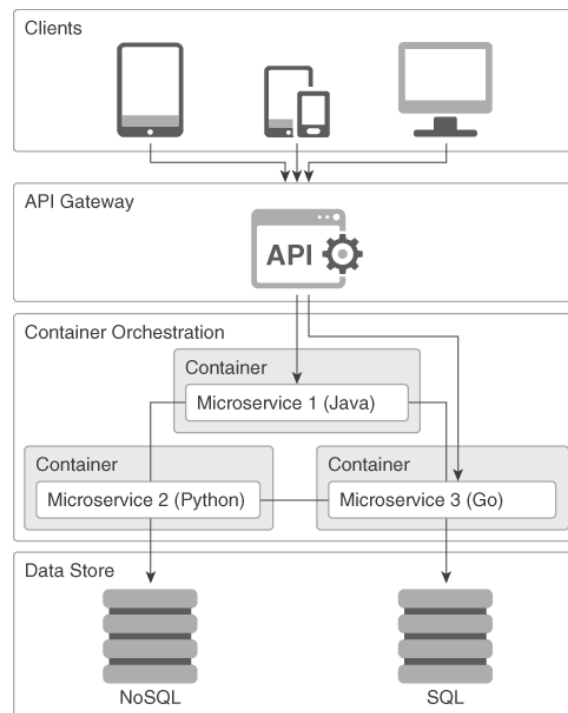


Figure 2.8: Microservices Pattern high level structure [49]

### Main Advantages and Disadvantages

This pattern has a lot of advantages, like:

- **Agility:** Because microservices are loosely coupled, application bugs are easier to fix, as it can be done for each individual service without affecting others and having to completely shutdown the entire system. It's also possible to roll back an update if something goes wrong [52].
- **Small codebase:** As each service has its own code, it's easier to learn it and add new features [52].
- **Fault isolation:** If one microservice has a problem, it won't affect the rest of the application, as long as upstream microservices are designed to handle faults correctly [52].
- **Scalability:** Services can be scaled separately, meaning that it's possible to only scale the necessary microservices without having to scale the whole application [52].

- **Reusability:** Because services are separated, it's possible to reuse them [58].

Microservices architecture also present some disadvantages, such as:

- **Integration:** The developer needs to make sure that the services are loosely coupled as much as possible, otherwise, a change in one service can have an affect in another service [51].
- **Complexity:** Even though, the point of this pattern is to separate the application into simpler microservices, if the system is considerably big it can become complex as well [52].
- **Data integrity:** Each microservice is responsible for its own data persistence, so data consistency can be a challenge across the whole system [52].

### Software Architecture Pattern Decision

As previously said, in the context of this project, security is definitely one of the main concerns of this application, meaning that the first pattern (Client-Server) can be immediately excluded since it's one of the disadvantages. Furthermore, the two patterns left are somewhat similar in terms of advantages, though the MVC has a disadvantage that can be problematic, that is the long learning curve, which, considering the time period left to develop the App (roughly 5 months) might not be the right one. The remaining one is the Microservices model which is really adequate for the context of this project, because it has a small codebase (easier to learn) and fault isolation (if a microservice stops working, it won't affect the rest of the application), even though this pattern has complexity as one of the disadvantages, it doesn't apply in this case, because the application to be developed is not of big dimension.

## 2.5 Security Considerations

When developing a web application, there are several security aspects that need to be considered and carefully handled. For the context of this project, security is the number one priority, because this application will lead with wallets with real money. In order to better understand how to protect a system, a checklist with the most common security challenges is presented along with possible techniques to lower the probability of malicious abuse. The following checklist was based on the white paper written by Gail Bayse [18].

### 2.5.1 Authentication

#### Challenge

The authentication represents the process of identifying if a user is actually who he claims to be and it's usually done with an username/email and a password.

#### Possible Techniques

- The system should lock the login function after a defined number of log in attempts. This login lock can be set to last a set number of hours to discourage the attacker



and the system administrator should be notified about this.

- All account activity regarding authentication should be logged (log in, log out, failed attempts, password changes, log in locks).
- Strong password rules must be applied, containing numbers, upper and lower case letters, special characters, and have minimum length of seven characters. The password should also not be found in any dictionary.
- When a user forgets a password, it must be changed rather than recovered.
- When changing a password, a user must also provide the older one first.
- Set an expiration time for all passwords, to enable frequent changes.
- Use two-factor authentication.
- Authentication details must be transported in a secure way, and must be encrypted using Secure Socket Layer (SSL) that can provide confidentiality, integrity and authentication (CIA Triad) for the transported data.
- Any passwords stored should be hashed (one-way hash).

### 2.5.2 Access Control

#### Challenge

After a user is authenticated, other challenge emerges, that consists of controlling what a user is allowed to do and what data can be seen and modified.

#### Possible Techniques

- There must be defined roles in the system, that control access to resources.
- Make sure that users cannot type custom URLs to access unauthorized pages.

### 2.5.3 Session Management

#### Challenge

Not protecting account credentials or session identifiers (ID) are known points of vulnerability for web applications.

#### Possible Techniques

- Cookies cannot contain confidential information, because they are transmitted in clear text. The user should always be aware and allow the application to use cookies.
- Session IDs shall be unique for each user, not contain personal information and can only be generated when the user authentication was successfully done.

- Inactive sessions should expire after set amount of time, deleting the session ID, and active ones should also have a set expire time, generating a new ID immediately after, to reduce the time window for possible malicious actions.
- Sessions IDs must be protected with Secure Socket Layer (SSL).
- When users log out, the session ID must be deleted.

#### **2.5.4 Data and Input Validation**

##### **Challenge**

Assuring that every data input is what it should be, rather than malicious code, such as, cross-site scripting or command injection.

##### **Possible Technique**

- The single best solution to avoid these attacks is validating every input against good criteria, that will depend on the data that is being required.

#### **2.5.5 Buffer Overflows**

##### **Challenge**

This happens when an attacker sends large amounts of data, through a data field, that exceed the capacity of the application. Attacks like this can make an application leave its normal state.

##### **Possible Techniques**

- Every HTTP request with input from users must be analysed to identify large amounts of data, and if the data is not suitable, that activity should be logged and the data refused.
- Every input field must have a limit length and a specific data type.

#### **2.5.6 Logging**

##### **Challenge**

Logs are an indispensable tool for developers that help towards monitoring, troubleshooting and debugging. They record information about an application's execution. They are also useful to rebuild events that led to a system failure.

##### **Possible techniques**

- Any log should always have date/time and description.

- Authentication and authorization actions must be logged (log in, log out, attempted log in, password changes) and include: date/time, success/fail, resources affected and the user involved.
- All administrator activity shall be logged.
- Log every time some data is deleted, to allow possible reconstruction.
- Data modifications should be logged as well.
- Most importantly, all the log files must be encrypted, because they contain critical information.

### **2.5.7 Error Handling**

#### **Challenge**

In any system errors will always occur, either caused by the user or the system itself. And, even though there's a testing process that looks to identify all the possible errors, there will always be errors that are unanticipated. There must be a process to handle these kind of errors as well, that need to include what it was, when it happened, and where it occurred. This allows the system administrator to have appropriate information to then act on its resolution.

#### **Possible Techniques**

- All the errors must be logged to an event log, including time/date, error code and, if possible, the code line.
- When an error occurs during the execution of the system, the end user shall be able to see an error page with some information.
- The application has to be thoroughly tested to identify possible errors.
- If an error occurs causing the application to completely or partially fail, it is important to block any unauthorized operations.

This page is intentionally left blank.

## Chapter 3

# Software Requirements Specification

In order to fully understand what needs to be in this system and how it should be, there needs to be a specification of software requirements. This also helps figuring out what the stakeholders (in this case, teachers involved in this project) want, and come to full agreement with them about functionalities and goals of this system.

To achieve clear and well defined software requirements, a document named "Software Requirements Specification for Crypta Project" was written. This document defines 9 general functional requirements that represent functionalities that a user can't see but are implemented in the backend side, 15 user functional requirements that correspond to functionalities directly related with the user with which he will interact in the frontend side, 5 administrator functional requirements that constitute the functionalities related to an admin that he will be able to access with a custom URL and manage the back-office, 13 non-functional requirements (split in: 2 Availability, 2 Maintainability, 2 Performance, 1 Portability, 1 Reliability, 3 Security and 2 Usability), 14 use cases and mockups for every user functionality. Below there's an example of a software requirement defined in the SRS document.

### Example of Software Requirement

To better understand how this document was written, there's an example of a user functional requirement, the corresponding use case and mockup, as well as, one non-functional usability requirement that directly affects the user functionality exemplified. The functionality chosen for the example corresponds to the payment process, where a user scans the QR code provided by a merchant. Each requirement is constituted of an identifier (ID), description and a MoSCoW priority [93].

#### User Functional Requirement

Requirement ID	Description	MoSCoW
C10	A user shall be able to pay for goods in the pay/send menu, by scanning a QR code.	Must Have

Table 3.1: Functional user requirement for payments

## Use Case

Use Case	Pay for Goods
Primary Actor	User
High Level Description	This use case describes how users pay for goods through the web application.
Preconditions	The user must be logged in, have sufficient funds in his wallet and have camera permissions enabled for the application.
Main Flow	<ol style="list-style-type: none"> <li>1) The user clicks the "Pay/Send" button in the bottom toolbar.</li> <li>2) The user is redirected to the payment page.</li> <li>3) The user scans the QR code with the phone's camera.</li> <li>4) The user confirms the transaction.</li> <li>5) The user inserts his PIN.</li> <li>6) The system processes the transaction.</li> </ol>
Exceptions	<ol style="list-style-type: none"> <li>3a) The user has the camera permissions disabled for the application.</li> <li>5a) The user enters an incorrect PIN.</li> <li>6a) The user has insufficient funds to complete the transaction.</li> </ol>

Table 3.2: Pay for goods use case description

Mockup



Figure 3.1: Payment screen

Non-Functional Usability Requirement

Requirement ID	Description	MoSCoW
U01	The system shall have a uniform look and feel between all screens.	Must Have

Table 3.3: Non-Functional Usability Requirement

This page is intentionally left blank.



# Chapter 4

## Selection of Technologies

In this chapter some technological decisions are made, which means that, for each technology considered, advantages and disadvantages are considered in order to make the most educated choice. This is made for both frontend and backend side. For the frontend side there's only one decision that needs to be made, that is the progressive web application framework. Moreover, for the backend side there are multiple decisions that need to be made, such as, backend technologies, SQL databases, NoSQL databases (MongoDB and Redis), containerization technology, message queue software and SMS (Short Message System) Communication API.

### 4.1 Progressive Web Application Frameworks

In order to develop the frontend side, it is necessary to select the appropriate progressive web application framework. The main aspect of the project that needs to be considered is the tight schedule for the development. This means that the right option has to be easy to learn, have a lot of community support and have a rich and well defined documentation. For this process three frameworks are considered: **ReactJS**, **AngularJS** and **VueJS**.

#### 4.1.1 ReactJS

##### Main Advantages of ReactJS

- Scalable and Flexible applications [82].
- This framework uses Virtual-DOM technology that ensures fast rendering of the code [62].
- React code for web applications can also be deployed for native Apps [62].
- Easy to learn [73].
- Very popular in the market - very frequent contributions to open source libraries made by huge community of developers [73].
- Clear and well defined documentation [43].

### **Main Disadvantages of ReactJS**

- Besides JavaScript, it also requires knowledge in JSX [82].
- With increased flexibility, also comes a higher chance of issues [82].

### **4.1.2 AngularJS**

#### **Main Advantages of AngularJS**

- Easy implementation due to well defined methodologies [82].
- Supports IntelliSense (intelligent code completion) and Typescript [62].
- Huge and active community of developers [73].
- Has a complex structure for large scale projects [73].

#### **Main Disadvantages of AngularJS**

- Requires knowledge of Typescript [82].
- Has a steep learning curve [73].

### **4.1.3 VueJS**

#### **Main Advantages of VueJS**

- Has clear and detailed documentation [73].
- VueJS allows fast software delivery, making it perfect for smaller solutions [4].
- Better performance due to Virtual-DOM like ReactJS [73].
- Easy to learn [62].

#### **Main Disadvantages of VueJS**

- Its high flexibility can create a lot of issues [82].
- This framework is owned by a single person and not a company, meaning that it doesn't have much support [82].
- Has a small community of developers, which can make it hard to find open source libraries and community support [4].

### **4.1.4 Framework Decision**

Considering the relevant aspects of the decision making previously mentioned, the right option is ReactJS, because it's easy to learn, has a huge community and has a clear and well defined documentation. On the other hand, AngularJS and VueJS were not chosen because AngularJS has a steep learning curve and VueJS has a small community with very small community support and contributions, making them not the best options.

## 4.2 Backend Technologies

For the backend technologies the options addressed are: **Flask**, **Spring Boot** and **ExpressJS**. Here the main concerns are also tight schedule for development and integration with third party services, mainly databases. So, the best option has also to be easy to learn, have community support and be easy to integrate with databases.

### 4.2.1 Flask

#### Main Advantages of Flask

- Easy to learn and use, mainly because it is a Python language that's considered one of the easiest to learn, also this frameworks provides the necessary tools to easily and quickly build web applications [37][24].
- This framework has a very detailed documentation with clear examples [37].
- High flexibility for web applications development [24].
- Has integrated support for unit testing [65].
- This framework is also very minimalistic and lightweight and can be used with very low effort [37].

#### Main Disadvantages of Flask

- Not asynchronous-friendly, which means web application done with this framework can only be handle one request at a time [37].
- For big projects, Flask can be very time consuming [37].

### 4.2.2 Spring Boot

#### Main Advantages of Spring Boot

- This framework is very easy to use with Java [74].
- Spring Boot reduces the time of development and increases productivity [96].
- It makes testing easier by providing default setups for unit and integration tests [1].
- Avoids writing a lot of boilerplate code (reusable code) [1].
- Avoids XML configurations [74].
- It also has a lot of libraries, plugins and tools that make development easier and faster [74].

#### Main Disadvantages of Spring Boot

- Spring Boot might not be the best for large applications [85].
- When deploying applications, Spring Boot might include dependencies that are not used, causing a huge deployment file size [85].

### 4.2.3 ExpressJS

#### Main Advantages of ExpressJS

- This framework allows using the same language (JavaScript) in the frontend and backend, making the development process much faster and easier, since one person can take care of both the front and back sides of the application [36].
- It's ideal for applications that handle a lot of requests (more scalable), as it uses asynchronous communication [36].
- Fast learning process [36].
- Has a vast open-source community [35].
- Easy to integrate with third-party services and middlewares [36].
- It's also simple to connect with databases, such as, MySQL, Redis and MongoDB [35].

#### Main Disadvantages of ExpressJS

- As an asynchronous based model, it relies on callback functions. This means that keeping a lot of queued tasks with its own callback, might impact the quality of the code, making it harder to understand and maintain [63].
- Asynchronous communication can also be a drawback since it can make development more difficult and time consuming.

### 4.2.4 Technology Decision

Given the previous aspects to decide which one is the best option, it's possible to consider ExpressJS the more reliable option, since it has a fast learning process, a vast open-source community, it's easy to integrate with third-party services and middlewares, and it's simple to connect with databases, which is one of the main concerns. For the other options, Flask was not considered because it's not asynchronous friendly meaning that it can only take one request at a time (low scalability) and might not be indicated for a payment/trading system. As for Spring Boot, seems as good as ExpressJS, but the latter one is preferred because is JavaScript based like the frontend framework chosen. This avoids having to learn two different languages.

## 4.3 SQL Databases

Relational Databases (SQL) in this system are in charge of storing user's informations, user's wallets and trading information, so the main aspects to consider are its availability, security, performance and simplicity. On the other side, scalability is not a real problem here, because this project is an experiment with a limited number of participants, meaning that it won't have that many transactions or trades, to the point where scale becomes an issue. For the SQL databases, **MySQL** and **PostgreSQL** are the two considered for comparison, because they are the most popular ones.

### 4.3.1 MySQL

#### Main Advantages of MySQL

- It's very easy to use (install and setup) [39].
- MySQL is very portable and can run on different platforms [55].
- Ensures 24x7 uptime and offer a lot of other solutions with high availability [55].
- It's very popular and widely used [78].
- Has great performance because it takes advantage of caching [86].
- MySQL has increased security as a result of the database administrator being able to grant permissions to specific database tables according to roles [86].

#### Main Disadvantages of MySQL

- MySQL is not recommended for very large databases [55].
- It's difficult to debug and maintain [86].
- Hard to scale [78].

### 4.3.2 PostgreSQL

#### Main Advantages of PostgreSQL

- Has a lot of community support [95].
- It's open source [60].
- Compatible with a lot of different programming languages and platforms [61].
- Highly compliant with SQL standard [60].

#### Main Disadvantages of PostgreSQL

- It's not as popular as other SQL databases [95].
- PostgreSQL performance is not as good when compared to MySQL [95].
- Database replication is more complex [95].
- Installation and configuration is not easy for beginners [61].

### 4.3.3 Database Decision

After reviewing both options and the main aspects that influence the decision, MySQL is the better option, because it meets every requirement set with the following advantages: easy to use, ensures 24x7 uptime, has great performance and increased security.

## 4.4 NoSQL Databases

In this system there's only need to have one non-relational database that's in charge of storing the application logs. The main points that need to be addressed here are high scalability because there will be logs for everything with a lot of redundancy, high access speed for quick checking and simplicity to use. Here it's only considered one NoSQL database (MongoDB) because it meets all the mentioned concerns and is the one that the author has experience with.

### 4.4.1 MongoDB

#### **Main Advantages of MongoDB**

- It's very scalable, since it has no schema [9].
- High speed of access, because it's a document oriented database where access is done by indexing [9].
- MongoDB is highly scalable. It uses shards for horizontal scalability, making it easier to increase storage capacity [83].
- Very clear and well defined documentation [83].
- It's simple to understand and learn, also, installation, setup and implementation are easy and don't take much time [83].
- This database is free to use [11].

#### **Main Disadvantages of MongoDB**

- Doesn't support JOIN operations like a relational database [9].
- Uses high memory for data storage [9].
- Each document has a limit size of 16MB [11].

## 4.5 Containerization Technology

Since this system will be developed based on a microservices architecture, it's a good practice to use a containerization technology. This can increase security and simplifies the isolation and segregation of microservices. For this technology there's not many options in the market, being that the most popular and widely used solution is Docker, hence why it's the only option considered, also there's no relevant disadvantages that would make this a bad option.

### 4.5.1 Docker

#### **Main Advantages of Docker**

- Docker allows for rapid deployment, since application are packed into containers that require minimal runtime [7].

- Increased security, because applications run on containers that are completely segregated and isolated from each other. This gives the developer complete control over traffic flow [7].
- It's very flexible, giving developers the advantage of making their own configuration and then deploy [32].
- Docker also offers automation, this means that tasks can be scheduled for each individual container that will occur at set times, without human intervention. This saves a lot of time and effort for the developers [32].
- It's a very stable environment, because, even though it has very frequent updates, the environment remains stable and there's no need to roll-back to past versions [32].

### **Main Disadvantages of Docker**

- Docker doesn't have cross-platform compatibility, meaning that, if an application was designed to run in a Docker container on Windows, it won't run on Linux or vice-versa [31].
- Using Docker also means increasing complexity a bit, because there's an additional layer that needs to be learned and developed [31].
- Docker itself has poor monitoring of running containers, and only provides very basic information [2].

## **4.6 Message Queue Software**

For intra-process communication between microservices, a message queue software is used in order to achieve asynchronous communication. The main aspects to consider here are: high performance to ensure fast communication, high scalability so multiple messages can be handled at once and fault tolerance mechanism meaning that the system doesn't stop working if any issues happen. The only software addressed is Apache Kafka since it's very popular, mature and meets every requirement set, and has no relevant disadvantages, which means that it's a good option to choose for the message queue of this system.

### **4.6.1 Apache Kafka**

#### **Main Advantages of Apache Kafka**

- Kafka is very durable, meaning that data is persistent and can't be replicated [15].
- It's able to handle high velocity and high volume of data, as well as, handling these messages with very low latency, because it decouples the message, letting the consumer consume that message at any time [16].
- Has a fault tolerance mechanism [16].
- High scalability, which means that it can handle large amount of message at once [16].
- Kafka processes data and messages in real time [16].

## Main Disadvantages of Apache Kafka

- It lacks a full set of management and monitoring tools [15].
- Has issues with message modification, in other words, performance decreases when making modifications [15].
- Can behave uncoordinated and slow if the number of queues increases a lot [16].
- There are some message paradigms missing in Kafka, such as, point-to-point queues and request/reply [8].

## 4.7 SMS (Short Message System) Communication API

This API is responsible for sending messages to users of the system with verification codes to increase security. The main aspects to consider, in order to choose the right service, are the price and the compatible programming languages (has to be compatible with the backend framework language). The APIs considered are: **Vonage**, **Plivo**, **Sinch** and **Twilio**.

### 4.7.1 Vonage<sup>1</sup>

#### Price of Vonage (per message)

In this service one message costs 0.0426€ to send, making it 42.6€ for 1000 messages.

#### Programming Languages Compatibility

The supported programming languages for the SMS API in this service are: JavaScript, Java, .NET, PHP, Python and Ruby.

### 4.7.2 Plivo<sup>2</sup>

#### Price of Plivo (per message)

Sending one message using Plivo costs 0.040€, which equals to 40€ for 1000 messages.

#### Programming Languages Compatibility

The compatible programming languages with the provided SMS API by Plivo are: PHP, Python, Java, JavaScript, Ruby, .NET and Go.

---

<sup>1</sup><https://www.vonage.com/communications-apis/sms/>

<sup>2</sup><https://www.vonage.com/communications-apis/sms/>



### 4.7.3 Sinch<sup>3</sup>

#### Price of Sinch (per message)

Each message costs 0.018€ in Sinch, making it 18€ for 1000 messages.

#### Programming Languages Compatibility

Sinch's SMS API is available for the following programming languages: Java, PHP and Python.

### 4.7.4 Twilio<sup>4</sup>

#### Price of Twilio (per message)

With Twilio, for SMS communication each message costs 0.037€, meaning that sending 1000 messages costs 37€.

#### Programming Languages Compatibility

The SMS API from Twilio can be used with the following programming languages: PHP, .NET, Java, JavaScript, Ruby and Python.

### 4.7.5 SMS API Decision

Since the programming language chosen for this project is JavaScript (ExpressJS), the Sinch API can immediately be discarded. The remaining options all have JavaScript compatibility, meaning that the deciding factor is the lowest price per message. So, in this case, the best option is Twilio.

---

<sup>3</sup><https://www.sinch.com/products/apis/messaging/sms/>

<sup>4</sup><https://www.twilio.com/sms>

This page is intentionally left blank.

## Chapter 5

# Software Architecture

The Software Architecture consists of a scheme that describes several aspects and decisions that add value to a software. So, when designing a software architecture that are some things that need to be considered, such as, all the functional and non-functional requirements defined, how the system is organized, how the different components in the system communicate with each other, how external dependencies may affect the software, what risks need to be considered, and a lot more.

### 5.1 C4 Model

The C4 Model provides a way for software development teams to efficiently and effectively communicate their software architecture at different levels of detail. This is an "abstraction-first" approach to create software architecture diagrams. The division of the system in small sets of abstractions and diagram types makes this model easy to learn and use [81]. For this project, the first three levels will be addressed.

#### 5.1.1 Level 1

This level represents the "System Context Diagram". This diagram gives a very high level where detail isn't important, because this only shows a zoomed out view of the entire system. This level focus mainly on the actors of the system and the software systems as a whole, rather than technologies and other low-level details. It's a great approach to show to people that have no expertise in the field [81]. For this project, the system context diagram (Figure 5.1) is as follows:

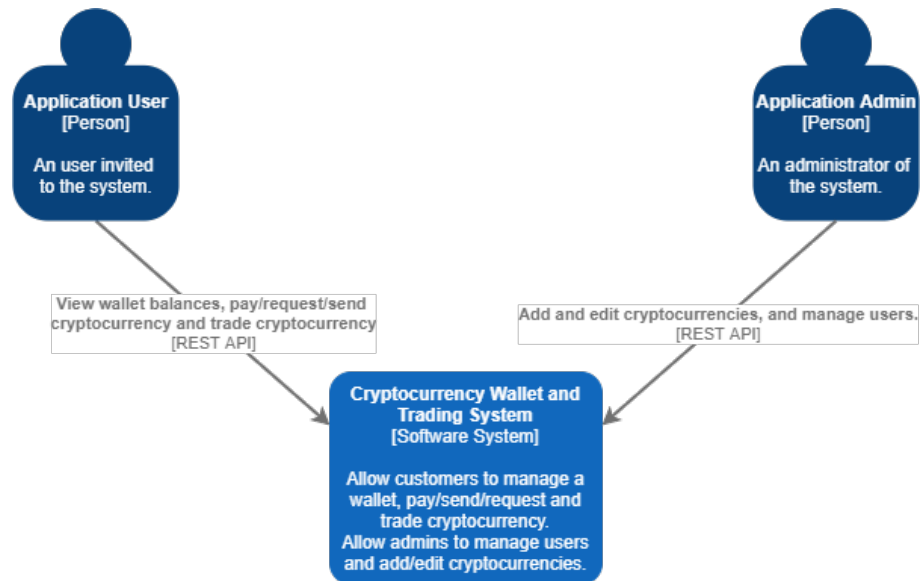


Figure 5.1: System Context Diagram (Level 1)

#### Description of the system components:

- **Application User:** End-user of the system, who is invited to use the application.
- **Application Admin:** Person in charge of changing the system in the backend side.
- **Cryptocurrency Wallet and Trading System:** Represents the whole system, where a user can manage a wallet, pay/send/request and trade cryptocurrencies.

#### 5.1.2 Level 2

This level represents a "Container Diagram". Here there's a bit more detail, specifically about the system. It shows which technologies are used and how responsibilities are distributed. This intended audience shifts more towards technical people inside and outside the development team, mainly software architects and developers [81]. In this project, the container diagram (Figure 5.2) identified is the following:

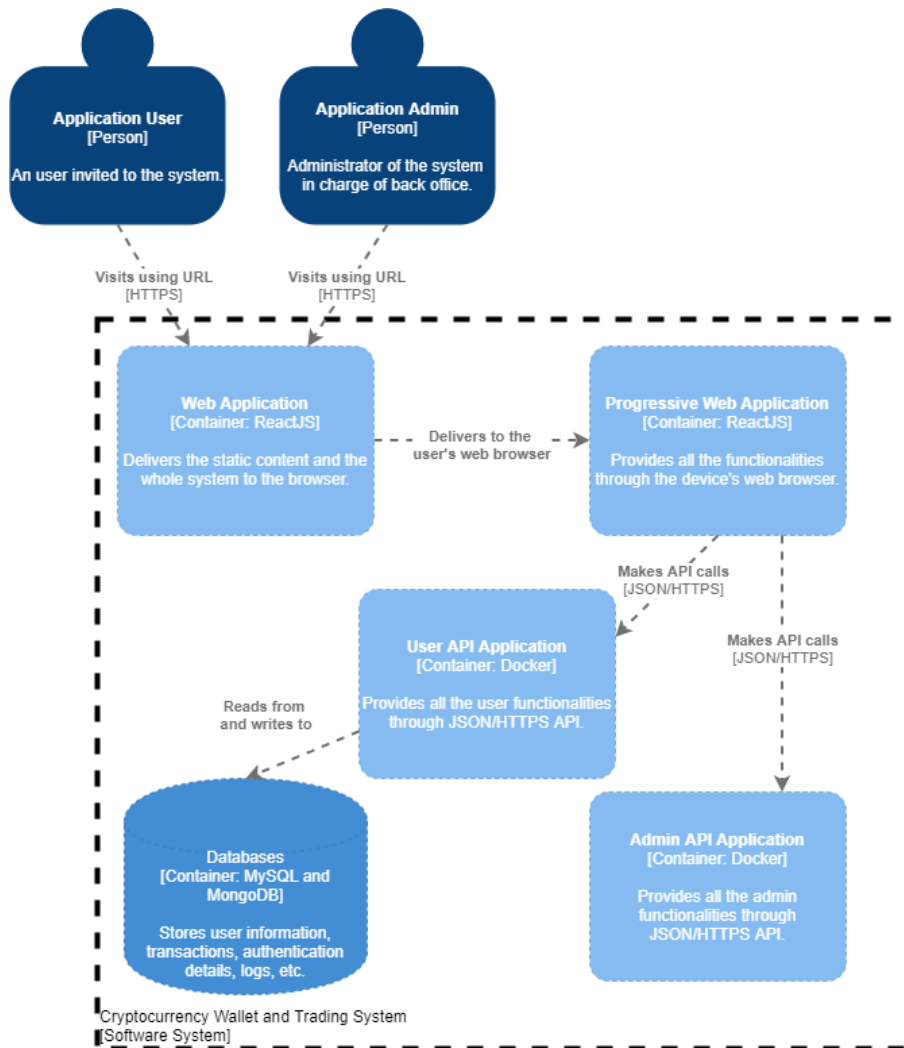


Figure 5.2: Container Diagram (Level 2)

#### Description of the system components:

- **Web Application:** This container is responsible for deliver the static content and the frontend side to the browser.
- **Progressive Web Application:** Provides all the functionalities through a progressive web application to the user's device web browser.
- **API Application:** Represents the main container of the system that has all the services with the business logic of the system.
- **Databases:** Consists of all the databases that will communicate with the microservices in the API Application container.

#### 5.1.3 Level 3

After defining every system container in the previous level, each one will be decomposed in this level ("Component Diagram"). This diagram shows every single component of a container and describes them, with its role, responsibilities and technologies/implementation

details. This diagram is designed particularly for inside software architects and developers [81]. The designed diagrams for this project are shown next, the first corresponds to the user side API application architecture (Figure 5.3) and the other one to the admin side API application architecture (Figure 5.4).

## User Side API Application

### Description of the system components:

- **User Service:** Handles the user's registration, log in and password recovery. It will also communicate with the email service to send confirmation emails and password recovery links, with the security service to encrypt data and verify confidential information and with the logging service to log all user activity (sign in and sign out actions).
- **Wallet Service:** This service is in charge of providing an user with cryptocurrencies balances in his wallet, account activity and filter cryptocurrencies displayed in his wallet screen. It will also read and write in a SQL Database in order to update and get wallet data.
- **Transaction Service:** Here is where all transactions will be made, such as, payments and sending and receiving cryptocurrency. There's communication with the security service to verify the user's PIN, with an SQL Database to update the wallet's balance when a transaction is executed and the logging service to record every transaction made.
- **Trading Service:** Every trade order will be executed in this component, will also log every order made by using the logging service and will also use an SQL Database to store/update orders.
- **Email Service:** Sends out emails to users.
- **Security Service:** Handles security functionalities, such as, data encryption and confidential information verification.
- **Logging Service:** Service that logs every action that's executed in the system to keep chronological track of things.
- **SQL Database:** There are four SQL Databases in this system: one that stores information of users, other that stores wallet information of each user, another that keeps transactions information and a last one that keeps everything trading related (orders and market prices).
- **NoSQL Database:** There's also a NoSQL Database that stores logs for user activity, transactions made and trades/orders done.

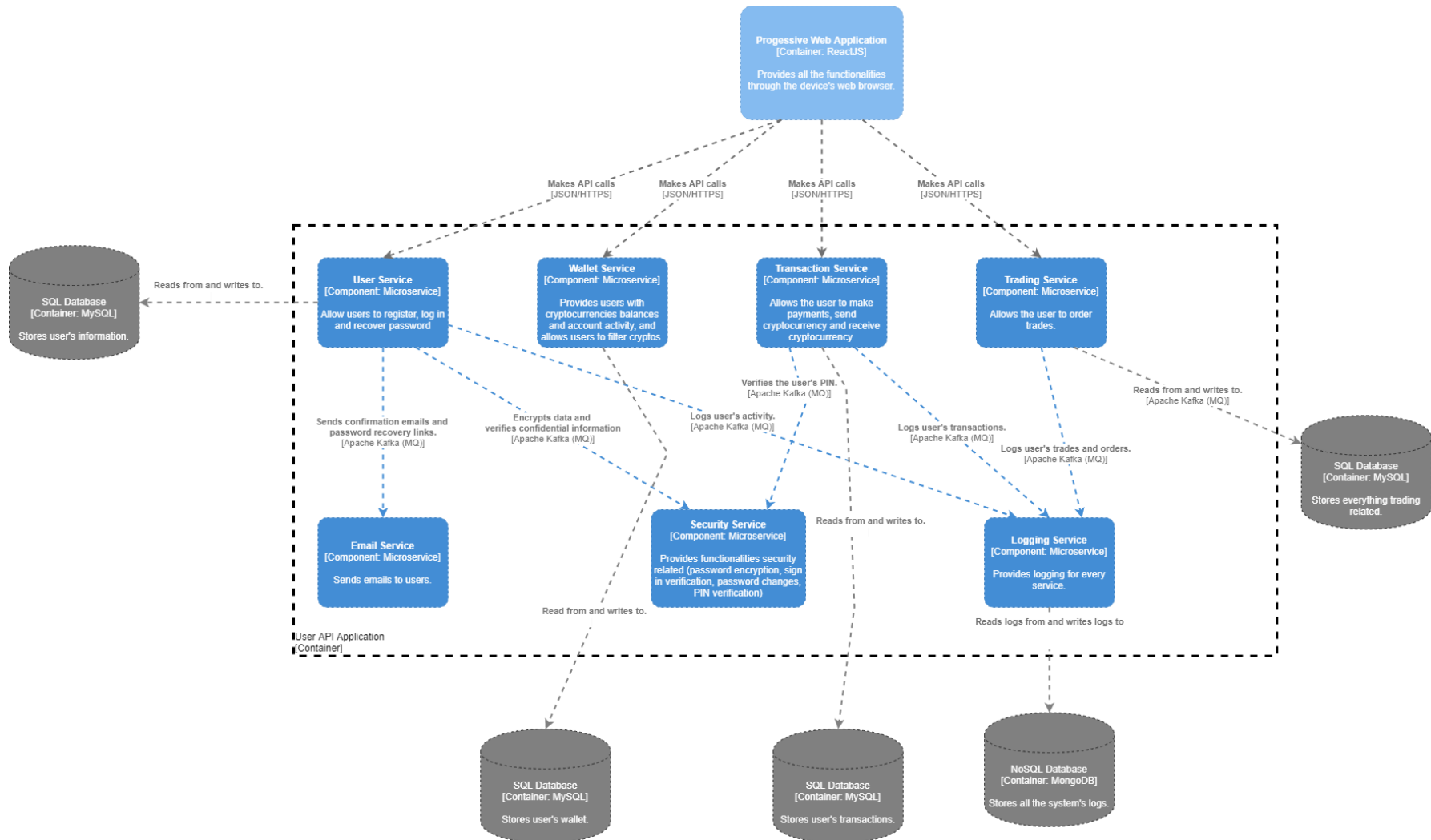


Figure 5.3: Component Diagram of the user side API application (Level 3)

## **Administrator Side API Application**

### **Description of the system components:**

- **User Management Service:** This service is in charge of inviting and editing users. There's internal communication with the security service to check roles privileges and with the logging service to save records of operations. It also reads and writes from a SQL database to check credentials.
- **Cryptocurrency Service:** Here is where cryptocurrencies will be added/edited. Each cryptocurrency has the following features that need be to defined: settlement time, transaction cost and transaction privacy). This service communicates with the security service to check role privileges, with the logging service to log operations (add and edit) and the SQL database where the crypto added/edited will be created/updated.
- **Security Service:** Handles security functionalities, such as, confidential information verification and system role checking.
- **Logging Service:** Service that logs every action that's executed in the system to keep chronological track of things.



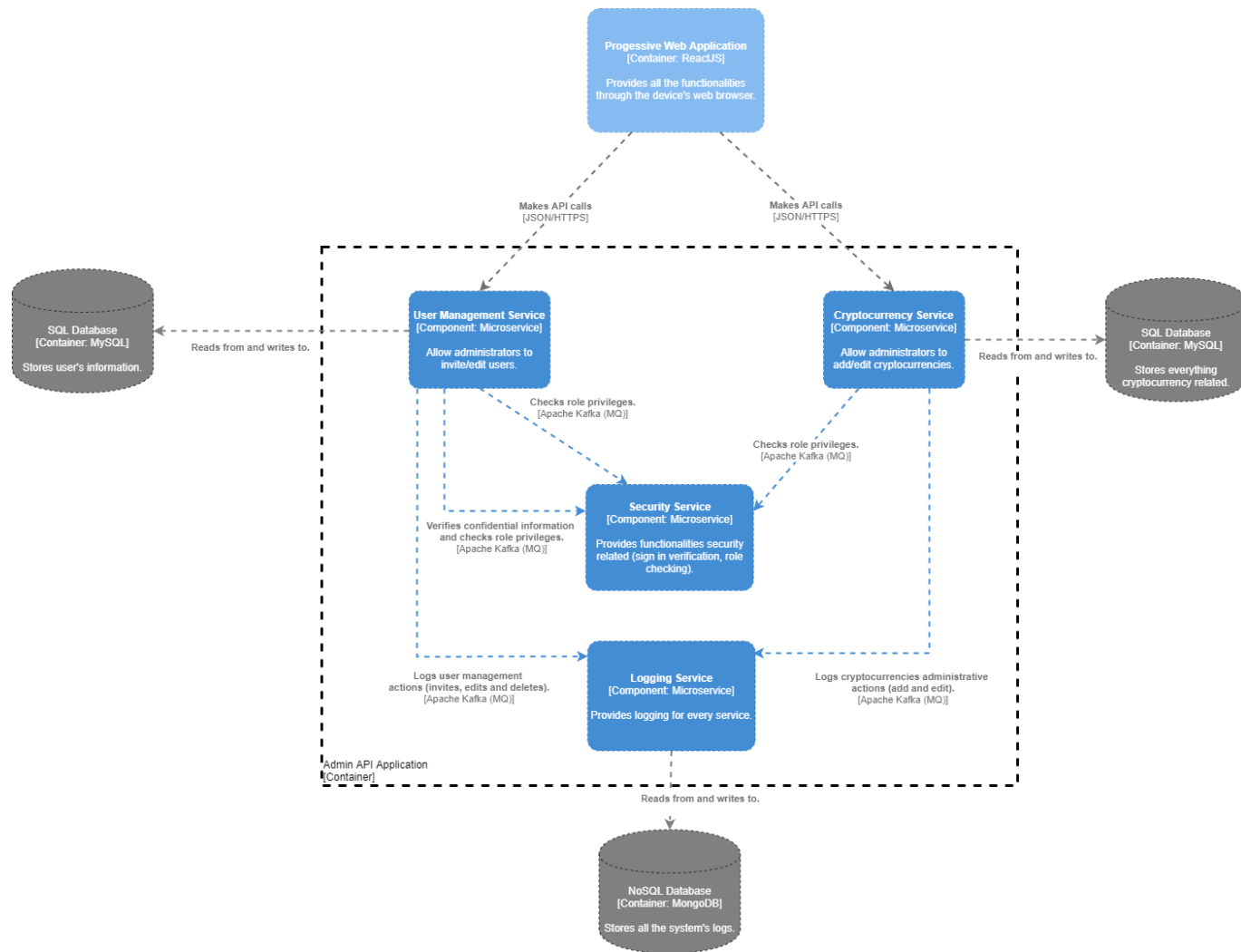


Figure 5.4: Component Diagram of the admin side API application (Level 3)

## 5.2 Entity Relationship Diagram (ERD)

An Entity Relationship Diagram, also known as ERD, ER Diagram or ER Model, is a type of structural diagram used in designing a database. It shows the relationship between entities stored in a database [91]. Since a microservices architecture is used, each service has its own *MySQL* database. The following subsections have each service's ERD, as well as the description of every entity. Some of the entities are defined in more than one database, because microservices need to be as independent as possible from each other, therefore there will be some necessary data redundancy between databases.

### 5.2.1 User Service

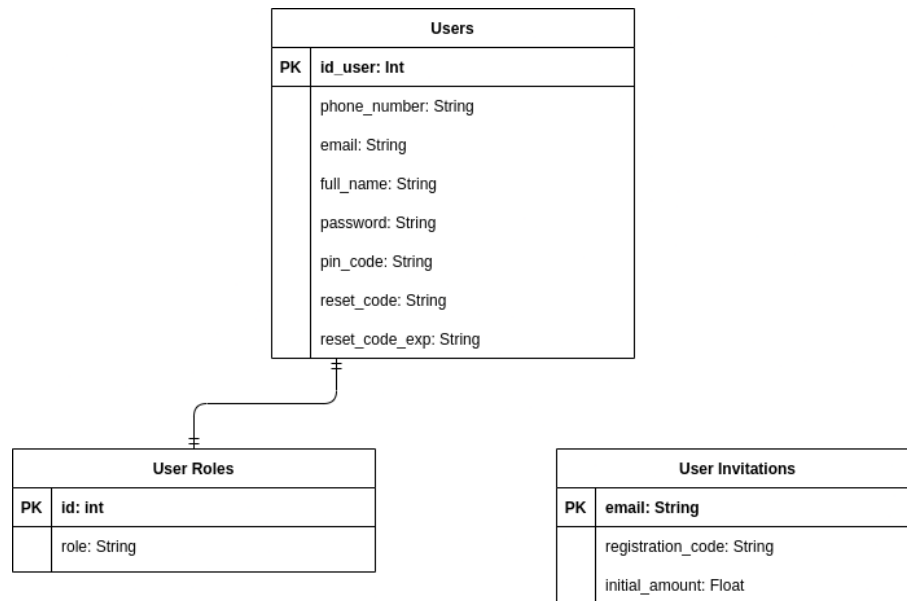


Figure 5.5: Entity Relationship Diagram for User Service

Description of each entity in the figure 5.5:

- **Users** - Entity that keeps information about a user.
  - `id_user`: Integer with unique ID to identify each user.
  - `phone_number`: 9 character string with user's phone number (portuguese format).
  - `email`: String with user's email.
  - `full_name`: String with user's full name.
  - `password`: String with user's hashed password concatenated with the hash salt used to encrypt.
  - `pin_code`: String with user's hashed pin code along with its hash salt.
  - `reset_code`: String with a code used to confirm the password recovery process.
  - `reset_code_exp`: String with a unix timestamp that corresponds to the exact date when the reset code expires.
- **User Roles** - Entity that keeps track of each user's role (user or admin).

- id: Integer with user’s ID.
- role: String with user’s role.
- **User Invitations** - Entity that saves user invitations until they register in the system.
  - email: String with the email invited to the application.
  - registration\_code: String with registration code sent through email that has to be used in the registration form.
  - initial\_amount: Initial balance of fiat currency (in Euros) that an user will have after registering to the system.

### 5.2.2 Cryptocurrency Service

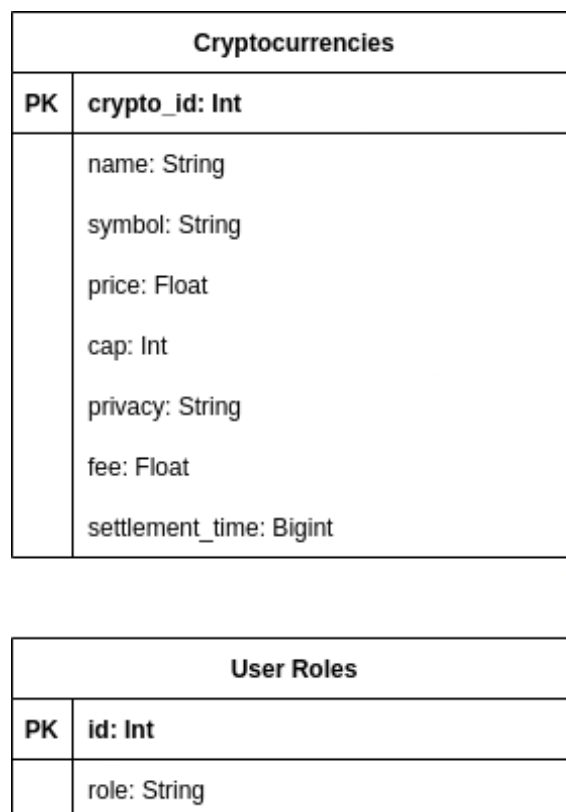


Figure 5.6: Entity Relationship Diagram for Cryptocurrency Service

Description of each entity in the figure 5.6:

- **Cryptocurrencies** - Entity that keeps information about a cryptocurrency.
  - crypto\_id: Integer with unique ID to identify each cryptocurrency.
  - name: String with the cryptocurrency’s name.
  - symbol: String with the cryptocurrency’s symbol, usually a three character word.
  - price: String with the cryptocurrency’s price in Euros.

- cap: Integer with the cryptocurrency's maximum quantity in the application.
  - privacy: String with the cryptocurrency's privacy that can be:
    - \* private: Users can not see who made the transaction/trade.
    - \* semi-private: Users can only see the user ID of who made the transaction/trade, but can not map that ID to that person.
    - \* public: Users can see the name or phone number of who made the transaction/trade.
  - fee: Float with a percentage value that corresponds to the transaction/trade fee applied when they are executed.
  - settlement\_time: Integer with the number of hours of the settlement time, which is the time that an amount of cryptocurrency transacted/traded takes to be available in the wallet.
- **User Roles** - Entity that keeps track of each user's role (user or admin).
    - id: Integer with user's ID.
    - role: String with user's role.

### 5.2.3 Market Service

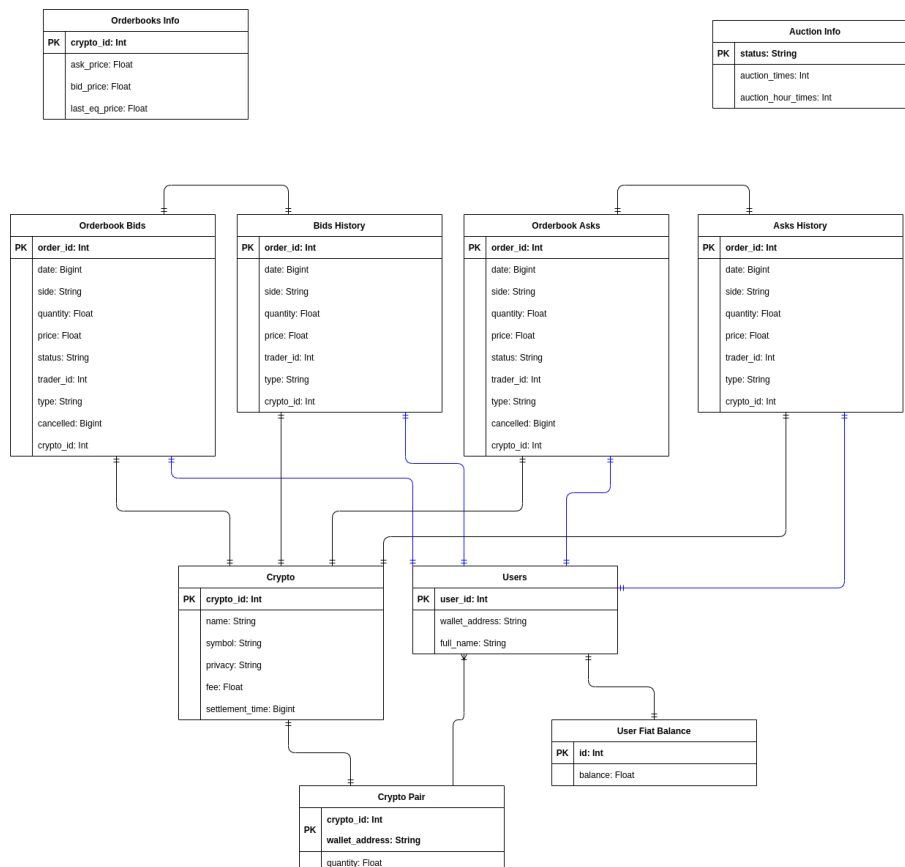


Figure 5.7: Entity Relationship Diagram for Market Service

Description of each entity in the figure 5.7:

- **Asks and Bids History** - Entities that keep information about buy or sells orders created by users and that were already closed.

- order\_id: Integer with the ID of an order.
  - date: Integer with a unix timestamp from when the order was created.
  - side: String with the side of the order (*BUY* or *SELL*).
  - quantity: Float with the amount of cryptocurrency traded of that order.
  - price: Float with the price of trade execution for that order.
  - trader\_id: Integer with the ID of the user that created the order.
  - type: String with type of order: market when a price is not specified and limit when a price is set.
  - crypto\_id: Integer with the ID of the cryptocurrency related with that order.
- **Auction Info** - Entity that keeps information about the state of the market.
    - status: String with status of the market, can be *open* when it is accepting new orders and *closed* when is resolving the market and can not accept new orders.
    - auction\_items: Integer with number of market resolutions per day (maximum 24).
    - auction\_hour\_times: Integer with number of market resolutions per hour (maximum 60).
- **Crypto** - Entity that keeps information about a cryptocurrency.
    - crypto\_id: Integer with unique ID to identify each cryptocurrency.
    - name: String with the cryptocurrency's name.
    - symbol: String with the cryptocurrency's symbol, usually a three character word.
    - privacy: String with the cryptocurrency's privacy that can be:
      - \* private: Users can not see who made the transaction/trade.
      - \* semi-private: Users can only see the user ID of who made the transaction/trade, but can not map that ID to that person.
      - \* public: Users can see the name or phone number of who made the transaction/trade.
    - fee: Float with a percentage value that corresponds to the transaction/trade fee applied when they are executed.
    - settlement\_time: Integer with the number of hours of the settlement time, which is the time that an amount of cryptocurrency transacted/traded takes to be available in the wallet.
- **Crypto Pair** - Entity that tracks how much each user has in wallet of each cryptocurrency.
    - crypto\_id: Integer with the cryptocurrency's ID.
    - wallet\_address: String with the user's wallet address.
    - quantity: Float with the amount of some cryptocurrency in wallet.
- **Orderbook Asks and Bids** - Entities that keep information about buy or sells orders created by users and that are currently open.
    - order\_id: Integer with the ID of an order.
    - date: Integer with a unix timestamp from when the order was created.

- side: String with the side of the order (*BUY* or *SELL*).
  - quantity: Float with the amount of cryptocurrency in the created order.
  - price: Float with the price in the order created (*NULL* if market order).
  - status: String with status of the order, can be *none* when it has not changed since created and *partially* when it was partially trade some amount of that order.
  - trader\_id: Integer with the ID of the user that created the order.
  - type: String with type of order: market when a price is not specified and limit when a price is set.
  - cancelled: Integer with a unix timestamp that corresponds to the date when that order should be cancelled.
  - crypto\_id: Integer with the ID of the cryptocurrency related with that order.
- **Users** - Entity that keeps necessary information about an user.
    - user\_id: Integer with the user's ID.
    - wallet\_address: String with the user's wallet address.
    - full\_name: String with the user's full name.
  - **User Fiat Balance** - Entity that keeps track of how much each user has of fiat currency (Euros).
    - id: Integer with the user's ID.
    - balance: Float with the amount of fiat currency that a user has.

## 5.2.4 Transactions Service

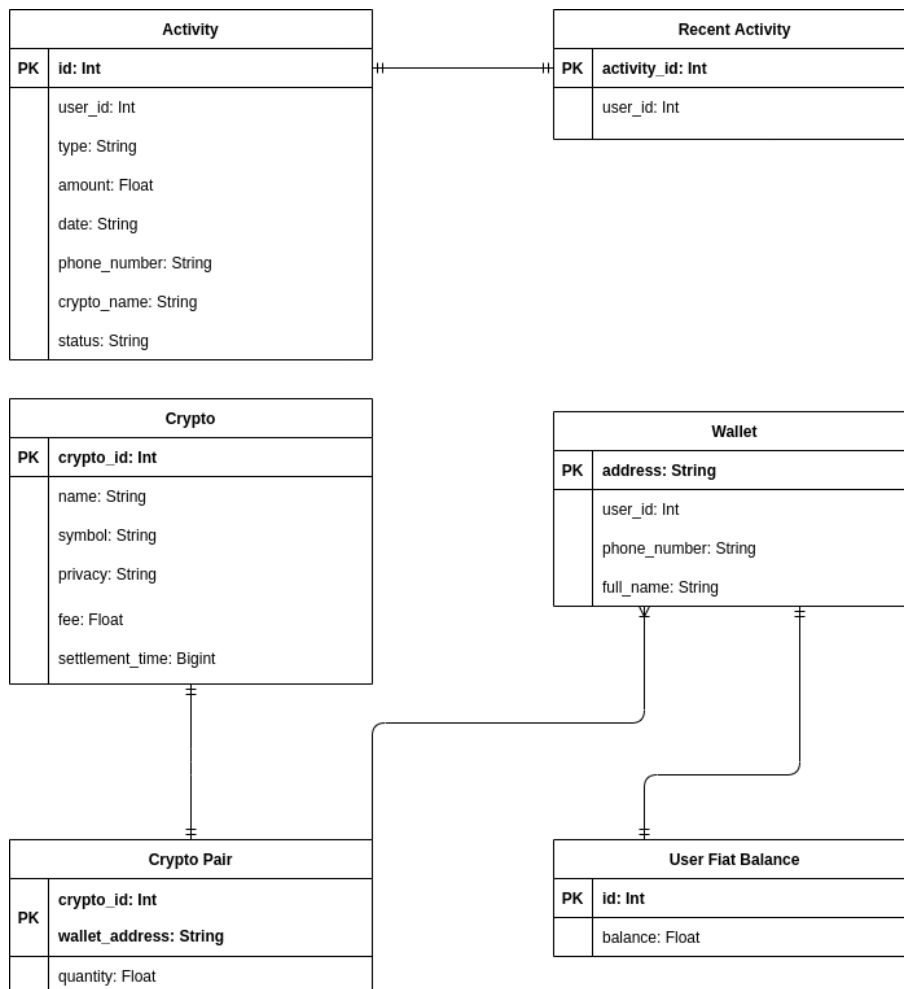


Figure 5.8: Entity Relationship Diagram for Transactions Service

Description of each entity in the figure 5.8:

- **Activity** - Entity that keeps all details about an user's transaction executed.
  - `id`: Integer with activity's ID.
  - `user_id`: Integer with the user's ID.
  - `type`: String with type of activity, it can be:
    - \* `received`: represents a transaction where that user received some amount of cryptocurrency.
    - \* `sent`: represents a transaction where a user sent some quantity of cryptocurrency.
  - `amount`: Float with the amount received/sent in the transaction represented by that activity.
  - `date`: String with a unix timestamp from when the transaction was executed.
  - `phone_number`: String with the phone number of the other user involved in the transaction.
  - `crypto_name`: String with the cryptocurrency's name.

- status: String with the status of the transaction (pending or completed).
- **Crypto** - Entity that keeps information about a cryptocurrency.
  - crypto\_id: Integer with unique ID to identify each cryptocurrency.
  - name: String with the cryptocurrency's name.
  - symbol: String with the cryptocurrency's symbol, usually a three character word.
  - privacy: String with the cryptocurrency's privacy that can be:
    - \* private: Users can not see who made the transaction/trade.
    - \* semi-private: Users can only see the user ID of who made the transaction/trade, but can not map that ID to that person.
    - \* public: Users can see the name or phone number of who made the transaction/trade.
  - fee: Float with a percentage value that corresponds to the transaction/trade fee applied when they are executed.
  - settlement\_time: Integer with the number of hours of the settlement time, which is the time that an amount of cryptocurrency transacted/traded takes to be available in the wallet.
- **Crypto Pair** - Entity that tracks how much each user has in wallet of each cryptocurrency.
  - crypto\_id: Integer with the cryptocurrency's ID.
  - wallet\_address: String with the user's wallet address.
  - quantity: Float with the amount of some cryptocurrency in wallet.
- **Recent Activity** - Entity that only keeps an activity ID and a user ID. It helps keeping only the most recent activities.
  - activity\_id: Integer with activity's ID.
  - user\_id: Integer with the user's ID.
- **Wallet** - Entity that keeps details about a user's wallet.
  - address: String with the user's wallet address.
  - user\_id: Integer with the user's ID.
  - phone\_number: String with the user's phone number.
  - full\_name: String with the user's full name.
- **User Fiat Balance** - Entity that keeps track of how much each user has of fiat currency (Euros).
  - id: Integer with the user's ID.
  - balance: Float with the amount of fiat currency that a user has.



## 5.2.5 Wallet Service

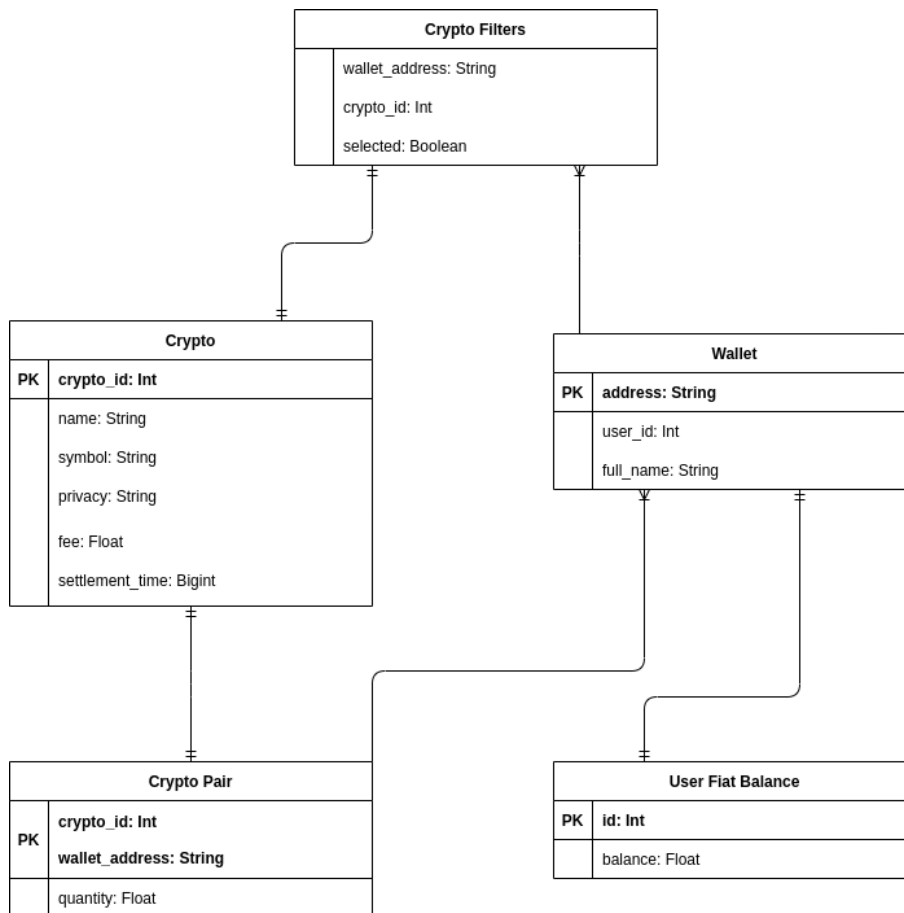


Figure 5.9: Entity Relationship Diagram for Wallet Service

Description of each entity in the figure 5.9:

- **Crypto** - Entity that keeps information about a cryptocurrency.
  - `crypto_id`: Integer with unique ID to identify each cryptocurrency.
  - `name`: String with the cryptocurrency's name.
  - `symbol`: String with the cryptocurrency's symbol, usually a three character word.
  - `privacy`: String with the cryptocurrency's privacy that can be:
    - \* `private`: Users can not see who made the transaction/trade.
    - \* `semi-private`: Users can only see the user ID of who made the transaction/trade, but can not map that ID to that person.
    - \* `public`: Users can see the name or phone number of who made the transaction/trade.
  - `fee`: Float with a percentage value that corresponds to the transaction/trade fee applied when they are executed.
  - `settlement_time`: Integer with the number of hours of the settlement time, which is the time that an amount of cryptocurrency transacted/traded takes to be available in the wallet.

- **Crypto Filters** - Entity that keeps track of the state of a cryptocurrency in a user's wallet filters.
  - `wallet_address`: String with the user's wallet address.
  - `crypto_id`: Integer with the cryptocurrency's ID.
  - `selected`: Boolean with the state of that cryptocurrency in the wallet (true or false).
- **Crypto Pair** - Entity that tracks how much each user has in wallet of each cryptocurrency.
  - `crypto_id`: Integer with the cryptocurrency's ID.
  - `wallet_address`: String with the user's wallet address.
  - `quantity`: Float with the amount of some cryptocurrency in wallet.
- **Wallet** - Entity that keeps details about a user's wallet.
  - `address`: String with the user's wallet address.
  - `user_id`: Integer with the user's ID.
  - `full_name`: String with the user's full name.
- **User Fiat Balance** - Entity that keeps track of how much each user has of fiat currency (Euros).
  - `id`: Integer with the user's ID.
  - `balance`: Float with the amount of fiat currency that a user has.

This page is intentionally left blank.

# Chapter 6

## Development

In this chapter numerous aspects about the development phase are described. Starting with the development environment configuration, risk analysis, development methodology and organization and the project structure. Then, some of the most important details taken into account during the implementation are explained, along with some of the decisions made during the development.

### 6.1 Development Environment Configuration

Before starting to develop the application, all the environment needed to be set up. As shown in the architecture, in the Chapter 5, each microservice and database needs to be inside a container. In order to containerize each one, Docker is used. To configure all the docker containers, a file named *docker-compose.yml* was created where each microservice and database was defined. The following two figures (6.1 and 6.2) show an example of how the containers were defined, one for the authentication service (*User Service* in the architecture) and other for the MySQL database for the users.

```
auth-srv:
  build:
    context: .
    dockerfile: ./auth/Dockerfile
  image: auth-srv
  env_file: ./env
  ports:
    - $AUTH_SRV_LOCAL_PORT:$AUTH_SRV_DOCKER_PORT
  volumes:
    - ./auth:/usr/src/app-auth
    - /usr/src/app-auth/node_modules
  depends_on:
    - mysqldb-users
  stdin_open: true
  tty: true
  command: "npm run start"
```

Figure 6.1: Authentication service container configuration

```
mysqldb-users:
  container_name: mysqldb-users
  image: mysql
  restart: always
  env_file: ./env
  environment:
    MYSQL_ROOT_PASSWORD: $MYSQL_ROOT_PASSWORD
    MYSQL_DATABASE: $MYSQL_USERS_DATABASE
  ports:
    - $MYSQL_USERS_LOCAL_PORT:$MYSQL_USERS_DOCKER_PORT
  volumes:
    - db-config-users:/etc/mysql
    - db-data-users:/var/lib/mysql
    - ./db/backup/files:/data_backup/data
    - ./scripts/users.sql:/docker-entrypoint-initdb.d/users.sql:ro
```

Figure 6.2: MySQL users database container configuration

Apart from the containers for the microservices and the MySQL databases, two other containers need to be set up, one for zookeeper which is a centralized service that provides synchronization services in distributed systems [17], and another one for kafka that is a distributed event streaming platform that allows to publish (write) and subscribe to (read) streams of events [14]. Figure 6.3 represents the configuration used to define the zookeeper and kafka containers.

```

zookeeper:
  image: confluentinc/cp-zookeeper:latest
  restart: always
  ports:
    - "32181:32181"
  environment:
    ZOOKEEPER_CLIENT_PORT: 32181
    ZOOKEEPER_TICK_TIME: 2000
  volumes:
    - /vol1/zk-data:/var/lib/zookeeper/data
    - /vol2/zk-txn-logs:/var/lib/zookeeper/log

kafka0:
  image: confluentinc/cp-kafka:latest
  restart: always
  ports:
    - "9090:9090"
  depends_on:
    - zookeeper
  environment:
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:32181
    KAFKA_ADVERTISED_LISTENERS: LISTENER_INTERNAL://kafka0:29090,LISTENER_EXTERNAL://localhost:9090
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_INTERNAL:PLAINTEXT,LISTENER_EXTERNAL:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_INTERNAL
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
  volumes:
    - /vol3/kafka-data:/var/lib/kafka/data

```

Figure 6.3: Zookeeper and Kafka containers configurations

After creating the file with all the necessary configurations, the containers can be initiated by running the command `docker-compose up`. The configurations for the MySQL containers, like the one in the figure 6.2, all have one line at the end which is responsible for running a MySQL script when the container starts up. This ensures that when the systems first fires up, all the tables and queries are created and executed in the different databases for each microservice.

After starting up all the containers it is important to make sure that everything initiated correctly. That can be done by running the command `docker ps` which lists all the running containers and also, by pinging each container through its name and port.

Finally, this environment needs to be set up three different times. First it was set up in a local machine for development, then it was requested a virtual machine (VM) from *DEI Helpdesk* which has a public IP address that allows to access it remotely. It worked as a staging VM to perform all the necessary tests. Lastly, it was also set up in another virtual machine that works as a final VM when the software is all implemented and tested, which is available in the public address <https://crypta.dei.uc.pt>. In order to expose the web application and redirect the traffic to the API endpoints through the public IP address in the VMs, *NGINX reverse proxy* was used with the configuration shown in the figure 6.4.

```
server {
    server_name crypta.dei.uc.pt;
    root /home/admin/Development/crypta-project-frontend-staging/build;
    index index.html;

    access_log /var/log/nginx/demo-app.access.log;
    error_log /var/log/nginx/demo-app.error.log;
    location / {
        try_files $uri /index.html =404;
    }

    location /api/auth {
        proxy_pass https://localhost:4000;
    }

    location /api/crypto {
        proxy_pass https://localhost:4001;
    }

    location /api/wallet {
        proxy_pass https://localhost:4002;
    }

    location /api/transactions {
        proxy_pass https://localhost:4003;
    }

    location /api/market {
        proxy_pass https://localhost:4004;
    }

    location /api/logs {
        proxy_pass https://localhost:4005;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/crypta.dei.uc.pt/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/crypta.dei.uc.pt/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}
server {
    if ($host = crypta.dei.uc.pt) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    server_name crypta.dei.uc.pt;
    return 404; # managed by Certbot
}
```

Figure 6.4: Configuration file for the *NGINX reverse proxy*

## 6.2 Risk Analysis

Risk Analysis is crucial in order to contain and mitigate risks that can occur during software development. During a risk analysis the following tasks are fundamental [71]: Identify risks and their triggers and consequences, classify likelihood and impact (1 - low, 3 - medium, 5 - high) of the risk that then generate a severity value, craft a mitigation plan to be used when a risk is triggered, monitor the project for risk triggers and execute the mitigation action when one is triggered. A risk can be described as a potential problem that can happen during software development, which can compromise the success of the project.

For this project the following risk analysis was performed (figure 6.5). It describes all the probable risks that can occur along with the severity, mitigation plan and consequences. To better understand and analyze risks, they are presented in a table like form, that was constantly updated during the project every time a risk is triggered or closed, also there was a periodic verification to check if there was new risks or if any should be removed.

ID	Date Raised	Risk Description	Likelihood	Impact	Severity	Mitigation Plan	Consequence	Status	Date Closed
1	4/10/2021	Project lifecycle (sprints) not clearly defined.	3	1	3	Analyse each sprint backlog carefully and in advance.	Delays in each sprint, which also may cause delay on the final delivery	Open	8/10/2021
2	4/10/2021	Lack of commitment	3	5	15	Plan thoroughly each sprint, mixing harder tasks with easier ones to balance workload. Make frequent pauses and keep a consistent schedule.	Sprint and final delivery delays.	Open	8/31/2021
3	4/10/2021	Poor documentation of the SRS (Software Requirements Specification Document)	3	3	9	Read the document carefully and make it consistent with the feedback received by the professors involved in the project.	Implementation of functional requirements that were not supposed to implement. Project Delays.	Closed	4/10/2021
4	4/10/2021	Adding new functional requirements that were not approved by the professors involved in the project.	1	3	3	Follow the initial SRS document, and if needed, speak to the professors involved to approve the addition of new requirements.	Project delays and may not meet the professors expectations for the application.	Open	8/10/2021
5	4/10/2021	Using a stack of technologies without experience with them.	3	3	9	Learn those technologies quickly and in advance, to avoid having to learn whilst implementing the application.	Sprint delays and possibility of not meeting the final delivery date.	Closed	5/7/2021

Figure 6.5: Risk analysis table

### 6.3 Development Methodology - Scrum

Scrum is based on iterative development where requirements and implementation evolve and refine along with the project development [88]. This project management framework is currently used in most of the agile projects in Worldwide [22]. As shown in figure 6.6, this process starts with the definition of the product backlog which is a list of the project requirements, then the focus shifts towards the creation of the sprint backlog. After that, the spring begins which usually includes the development and validation of the requirements set in the sprint backlog. Finally, at the end of the sprint, a meeting is held to make demonstrate the developed functionalities during the sprint to the product owner [22].

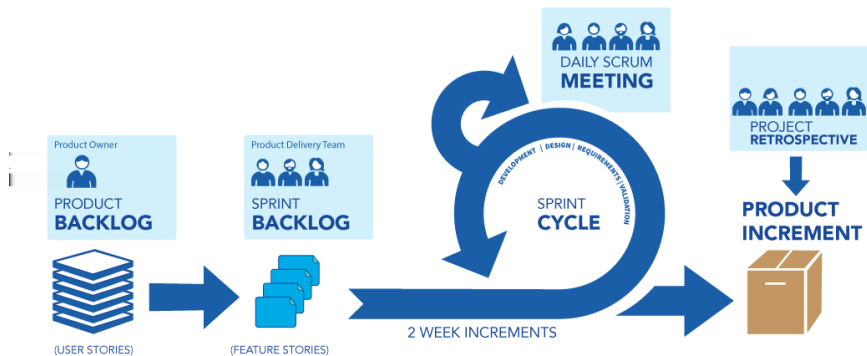


Figure 6.6: Scrum process [10]

### Project Organization

This section describes how the development of this project was organized. A *Trello*<sup>1</sup> board was created before starting the development stage. The main objective of this tool is to organize project of any size, by creating cards, which represent tasks, and distributing them through different sub-boards.

<sup>1</sup><https://trello.com/>

Figure 6.7 displays the trello board for this project at a stage close to the project deadline. The board is sub-divided in five columns:

- **Product Backlog:** List with all the functionalities to implement during the project.
- **Sprint Backlog:** List with the functionalities to implement during a single sprint, in this case it lasts two weeks.
- **In Progress:** Tasks that are currently being worked on.
- **Ready for test:** Tasks that were already developed and are waiting to be tested.
- **Done:** Tasks that were already implemented and tested.

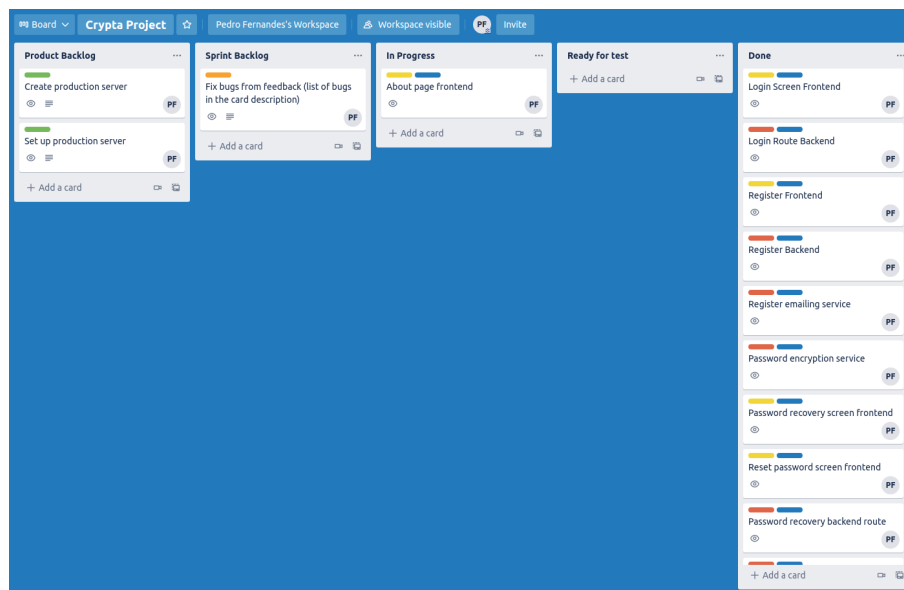


Figure 6.7: Trello board for the project

At the start of every sprint, tasks were picked depending on their MoSCoW priority, difficulty of a task (harder and longer tasks first) and dependency between them (some tasks implementation might depend on others).

## 6.4 Implemented Functional Requirements

In the document *Software Requirements Specification (SRS)* several functional requirements were defined, more specifically, 15 user functional requirements and 5 administrator functional requirements. However, some of them were only partially implemented mostly due to changes in those requirements during its implementation.

The following tables 6.8 and 6.9 contain the list of user and admin functional requirements, as well as their respective implementation status, being that it either is *implemented* (green), *partially* (yellow) and *not implemented* (red).



Implemented Functional User Requirements					
Requirement ID	Status	Requirement ID	Status	Requirement ID	Status
C01	Implemented	C06	Implemented	C11	Implemented
C02	Implemented	C07	Implemented	C12	Implemented
C03	Implemented	C08	Partially	C13	Implemented
C04	Implemented	C09	Implemented	C14	Partially
C05	Implemented	C10	Implemented	C15	Implemented

Figure 6.8: Implemented Functional User Requirements

Implemented Functional Administrator Requirements	
Requirement ID	Status
A01	Partially
A02	Implemented
A03	Implemented
A04	Implemented
A05	Implemented

Figure 6.9: Implemented Functional Administrator Requirements

As we can see in the figure 6.8, the requirement C08 ("A user shall be able to instantly navigate to the main menu, see his notifications and display the settings menu by using the buttons at the top toolbar") and C14 ("A user shall be able to view his orders, positions and history of trades in the Portfolio Menu") were only **partially** implemented. For the requirement C08, notifications were not implemented and for C14 positions were not considered and everything is considered an order (either open or closed).

## 6.5 Client-side Implementation (React.js)

For the client-side implementation, the mobile web application interface was developed in React.js alongside CSS (*Cascading Style Sheets*) for styling. Furthermore, Webpack<sup>2</sup> was used as a module bundler and Babel<sup>3</sup> as a Javascript transpiler. Both these tools help building the React.js application.

When it comes to the structure of the code, it is important to keep it as uniform as possible. To avoid confusions and to facilitate future code re-factoring, we used framework standards to organize the project directories [38]. The following figures 6.10, 6.11 and 6.12 show how the project is structured.

<sup>2</sup><https://webpack.js.org/>

<sup>3</sup><https://babeljs.io/>

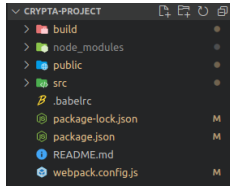


Figure 6.10: React project structure (general)

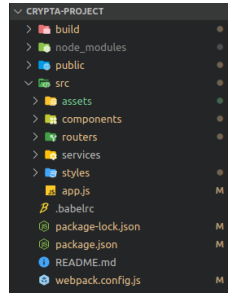


Figure 6.11: React project structure ("src" folder)

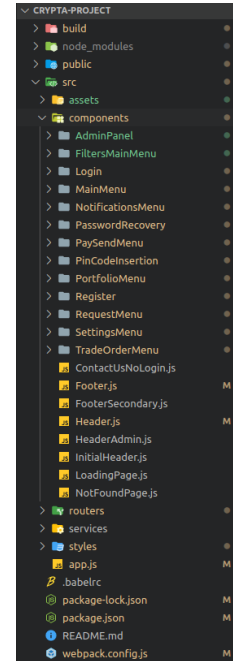


Figure 6.12: React project structure (components folder)

As we can see by the figures above, the project directory consists of: a *build* folder where the files for production are created, a *node\_modules* folder where all the libraries used during development are installed, a *public* folder that has the HTML files and images to compile during deployment, *src* folder with all the project source code, *routers* with the code for the routing between URLs and components, *services* folder with a single file that help communicating with the API endpoints, *styles* folder with all the styling CSS code, an *app.js* file from where the react application starts up from, *.babelrc* file with the configuration for *Babel*, *package.json* with project properties and dependencies and *webpack.config.js* with the Webpack configuration for module bundling.

## 6.6 Server-side Implementation (Node.js)

For the server-side implementation, Node.js was used with the package Express, MongoDB for the service that saves the logs and MySQL in the remaining services.

Since the backend is developed on top of a microservices architecture, each service has its own project directory and is created inside distinct docker containers to ensure independence between each other. Each container also has an independent MySQL database. The following figures 6.13, 6.14 and 6.15 will help understand how each service implementation was structured.

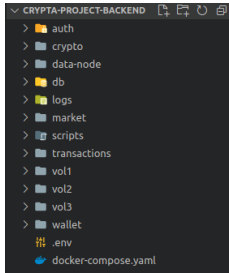


Figure 6.13:  
Back-side project  
structure (gen-  
eral)

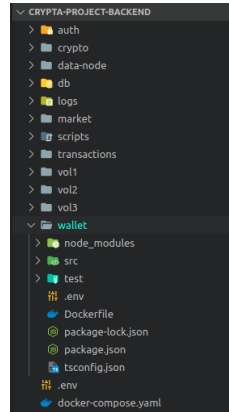


Figure 6.14:  
Back-side project  
structure (*wallet*  
service directory)

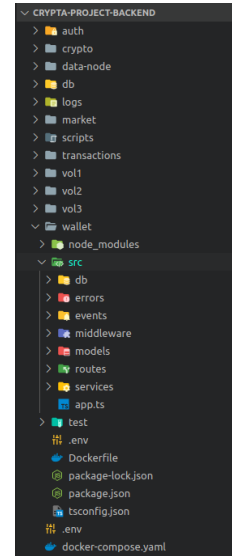


Figure 6.15: Re-  
act project struc-  
ture (*wallet* ser-  
vice "src" direc-  
tory)

The figures above shown how the backend is structured (figure 6.13) and, since all the services were structured the same way, it is possible to check in the figures 6.14 and 6.15 how a service folder and a service source code (*src*) folder are structured. As we can see, a service directory has the following folders and files: *node\_modules* folder which has all the libraries used during the implementation installed, *src* folder with all the source code, *test* folder with all the test cases for unit testing, *.env* file where the environment variables are declared, *Dockerfile* file which has the configuration for the docker container, *package.json* with the project properties and dependencies and *tsconfig.json* with the configuration for typescript. Furthermore, inside the *src* folder the structure is: *db* folder with database configuration, *errors* folder with the errors that can be thrown in a request, *events* folder with the code for the event synchronization (publish events and subscribe event streams), *middleware* folder where the functions that execute before a request is processed in its route, *models* folder with the functions responsible of querying to the database, *routes* where the different API endpoints are defined along with the logic that processes each request, *services* with functions that are called inside the routes logic but are not exposed through the API and *app.js* file that starts up the service.

## 6.7 System Implementation and Description

Now, all the implemented functionalities will be explained using screenshots and describing them. This will help better understand some of the implementation decisions made in the front-side development and how the application can be used by its users.

### Registration

Figure 6.16 illustrates the registration screen. In order to register in the application, users need to be invited by an administrator using an email. An invitation email is sent

containing a randomly generated 6 digit invitation code and a button which redirects to the registration page. In that page, a user needs to fill a form with: 6 digit invitation code, valid phone number from Portugal (9 digits), email with a valid format, full name with only letters and a minimum of 5 characters, a strong password (length between 8 and 20, at least 1 lowercase and 1 uppercase letter, 1 number and 1 symbol), repeat the same password and a PIN code that is used to confirm operations in the application. After filling everything, the user needs to click the "Create Account" button, which first validates all the fields. If there's an invalid field, its box is highlighted in red and an error message is displayed. If everything is valid, a request is sent to the server which will process it and respond with a success or error message. In case of error, the message with the error description received from the server is displayed, otherwise the user is redirected to the login page (figure 6.18).

In regards to the backend, when a request arrives to register a user, all fields are first validated. Then it checks if phone number or email inserted are already being used, if that email has an invitation code associated in the database and finally if the invitation code inserted by the user corresponds to the one saved in the database. After all the validations, the password and PIN code are encrypted so they can be saved in the database, that encryption is done with the code in the figure 6.17 which first creates a "salt", that consists of the hashing of a string with the user's email concatenated with his full name (to guarantee uniqueness) with the hash function *SHA-256* (Secure Hash Algorithm of 256 bits), then that salt is used to hash the original password/PIN code with the hash function *Scrypt*, finally the hashed password/PIN code is returned with its salt concatenated. Lastly, all the information originally inserted by the user, along with the hashed password and PIN code, saved in the database and a success message is returned to the client-side in the response.

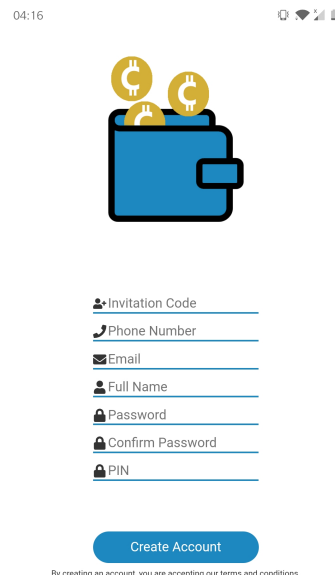


Figure 6.16: Register screen

```
1 import { scrypt } from 'crypto';
2 import { promisify } from 'util';
3 import { sha256 } from 'js-sha256';
4
5 const scryptAsync = promisify(scrypt);
6
7 export class Password {
8   static generateSalt(email: string, full_name: string) {
9     return sha256(email.concat(full_name));
10  }
11
12   static async toHash(email: string, full_name: string, password: string) {
13     const salt = this.generateSalt(email, full_name);
14     const buf = await scryptAsync(password, salt, 64) as Buffer;
15
16     return `${buf.toString('hex')}.${salt}`;
17   }
18
19   static async compare(storedPassword: string, suppliedPassword: string) {
20     const [hashedPassword, salt] = storedPassword.split('.');
21     const buf = (await scryptAsync(suppliedPassword, salt, 64)) as Buffer;
22
23     return buf.toString('hex') === hashedPassword;
24   }
25 }
```

Figure 6.17: Password hashing implementation

## Login

In figure 6.18 the screen for the login is displayed. In this page users can do three actions: log into the application, recover password (through the "Forgot password?" button) and contact the support team (system administrators). In order to log in, a user needs to type his phone number and password. Then click the "Login" button which validates the login fields (phone number and password). If all fields are valid, a request is sent to the server, otherwise the invalid field is highlighted in red and an error message is displayed. If the response from the server is successful, the user is redirected to the main page (wallet screen in figure 6.20), else the error message sent by the server is shown to the user.

In the backend, when a login request arrives, it first validates the parameters, then it checks, by the phone number, if the user exists in the system. If so, it gets the user role from the database (user or administrator) and creates a jwt token with the user's ID, email and role as payload. Finally, that token created along with the user ID is sent in the request session. By doing this, the session will be persisted in a cookie in the client-side that will be sent in every request to the server during 1 hour (expiration time for cookies in the configuration). This helps with the user authorization in the server-side when requests arrive, which can be done by validating the jwt token as shown in the figure 6.19.

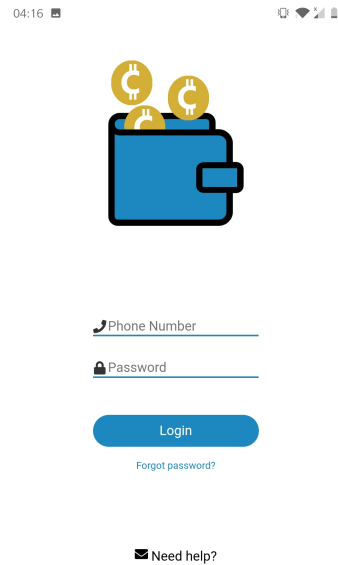


Figure 6.18: Login screen

```
18 export const currentUser = (req: Request, res: Response, next: NextFunction) => {
19   if (!req.session?.jwt) {
20     return next();
21   }
22
23   try {
24     const payload = jwt.verify(req.session.jwt, process.env.JWT_KEY!) as UserPayload;
25     req.currentUser = payload;
26   } catch (err) {}
27
28   next();
29 };
```

Figure 6.19: User authorization code

## View wallet and all activity

After a user logs in, he's redirected to the wallet screen shown in the figure 6.20. This screen is considered the main page of the application and consists of two toolbars, one at the top with the settings button and other at the bottom with the different menus available in the application. It also has a display panel with the user's current fiat balance (in Euros (€)) and a recent activity panel that has the user's last two transactions executed with compacted information. Finally there's a wallet panel with cryptocurrencies and its balance in that user's wallet, this panel also has a filters option to select which cryptocurrencies should be displayed.

The recent activity panel has a "View all activity" button which opens a page with all the activity as shown in figure 6.21. It includes every transaction made by the user, as well as, complete information about them.

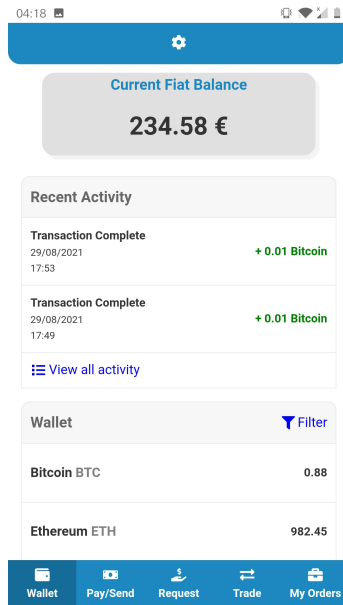


Figure 6.20: View wallet screen

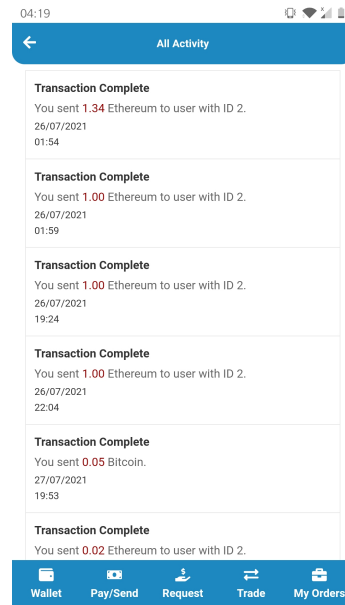


Figure 6.21: View all activity screen

## Change wallet filters

When the "Filter" button is clicked in the wallet panel, a new page opens (figure 6.22) which allows the user to select which cryptocurrency he wants to be displayed in the main wallet screen. He can also press the "Select all available in wallet" that automatically selects all the cryptocurrencies that have a balance bigger than 0 in the wallet. Finally, the user presses the "Save" button which makes a request to the server in order to save his filters, so the next time he opens the application, his wallet is consistent with the last time he used it.

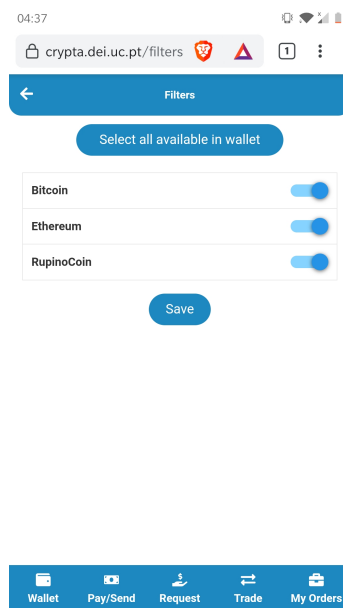


Figure 6.22: Change wallet filters screen

## Make payment with QR code

Figure 6.23 illustrates the screen where a user can make a payment by reading a QR code. This starts by navigating to this menu in the bottom toolbar and giving permissions for the application to use the phone camera. Then, the user needs to point the camera to the QR code generated by another user and confirm the transaction the modal screen that pops up (figure 6.24). By confirming the transaction, a new modal screen prompts the user to insert his PIN code (figure 6.25).

In the backend, when the server receives the payment request, it validates its parameter which corresponds to the string read from the QR code that needs to have the following structure:  $\langle \text{Cryptocurrency name} \rangle : \langle \text{Recipient wallet's address} \rangle ? \text{amount} = \langle \text{transaction quantity} \rangle$ . After the parameter's validation, it checks if the wallet address exists and if the user paying has enough balance the execute the transaction. Finally, the transaction's quantities are calculated taking in consideration the transaction fee and the transaction is completed. Though the user paying will have his balance instantly updated, the user receiving will only have his balance updated depending on settlement time. Until concluded, the transaction appears as pending.

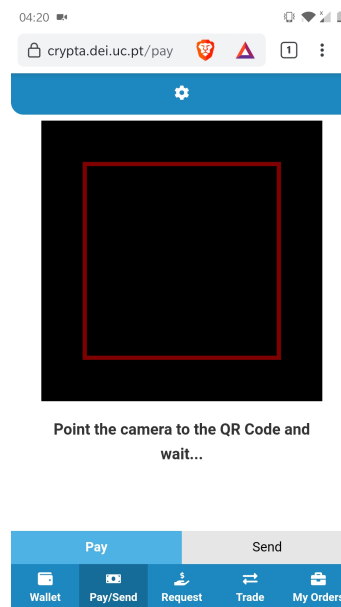


Figure 6.23: Payment menu screen



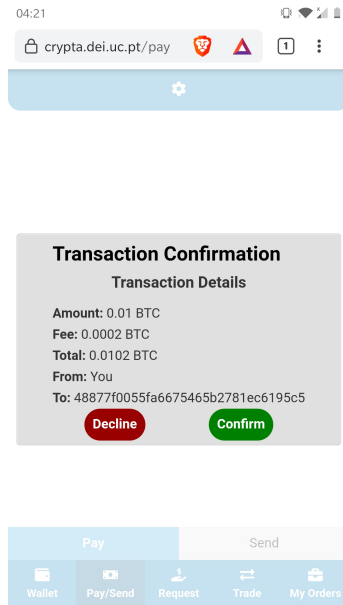


Figure 6.24: Payment confirmation screen

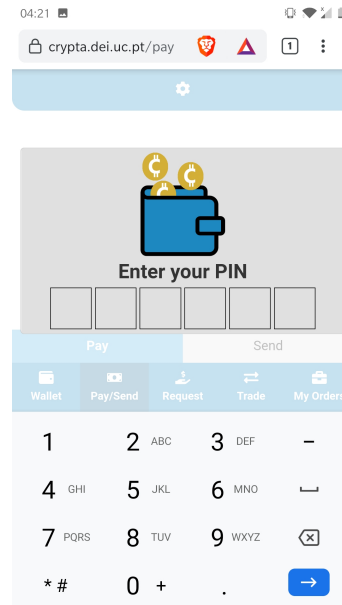


Figure 6.25: Payment pin insertion screen

## Send cryptocurrency

Figure 6.26 shows how to send cryptocurrency. The user needs to input the recipient's phone number, the amount, and select the cryptocurrency that he wishes to send. Then, the fields are validated, highlighting in red and displaying an error message in case of invalid fields, else it displays a modal screen to confirm the transaction (figure 6.27) and then another modal screen to insert his PIN code (figure 6.28).

In the backend, the logic to process the request is the same, with the only difference being the request parameters, in this case the request has a phone number which is then used to get the user's wallet address from the database.

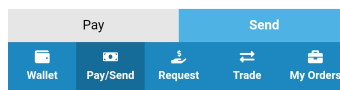
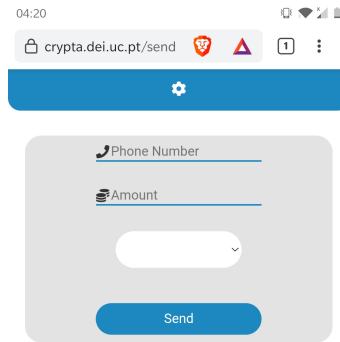


Figure 6.26: Send cryptocurrency menu screen

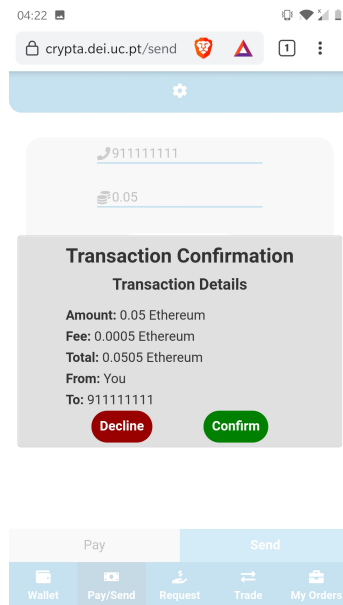


Figure 6.27: Send cryptocurrency confirmation screen

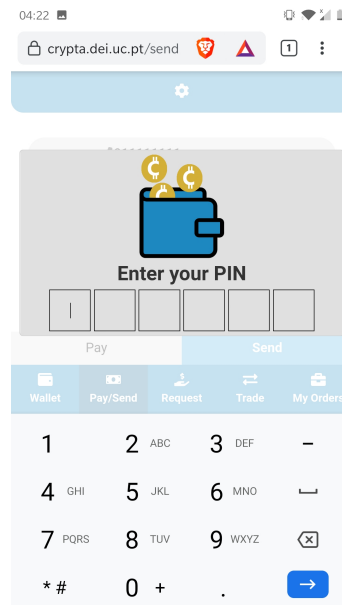


Figure 6.28: Send cryptocurrency pin insertion screen

## Request cryptocurrency

Figure 6.29 illustrates the request cryptocurrency screen. In this page, the user can type an amount, select a cryptocurrency and, by clicking the "Generate QR code", a modal screen is shown with a QR code (figure 6.30). This code can be scanned by another user in the payment screen. When read, it translates into a string with the following pattern: *<Cryptocurrency name>:<Recipient wallet's address>?amount=<transaction quantity>*.

The user can also save the code as an image and send it, which allows remote transactions.

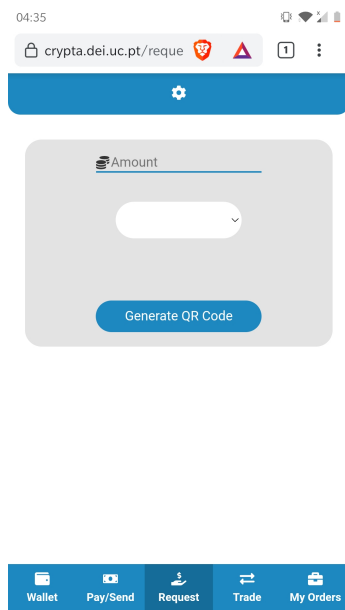


Figure 6.29: Request cryptocurrency menu screen

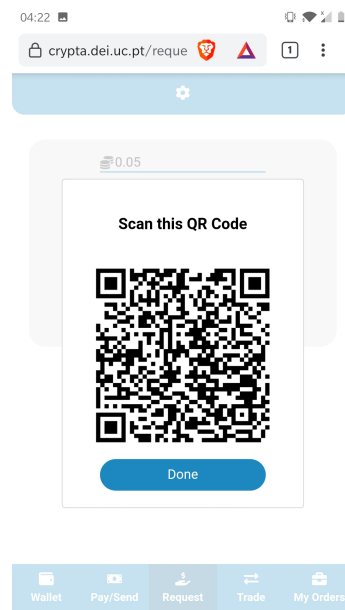


Figure 6.30: Request cryptocurrency qr code generated screen

## Create market and limit orders

In the trade menu, a user can create orders. Orders can be to sell or to buy some cryptocurrency (figures 6.31 and 6.32). After choosing if he wants to sell or buy, he needs to select if he wants to create a market order, which means that he does not need to type a price because the order will execute at the market price at the execution moment, or a limit order where the user has to type at what minimum/maximum price he wishes to execute the trade (minimum for sell orders and maximum for buy orders). Then, with those two options chosen, the user needs to fill the form with the information asked. In case of a market order, the user has to select the pretended cryptocurrency, type an amount to trade and, optionally, input a date for the order to expire. For a limit order, the only difference is a new field which takes the minimum/maximum price for the trade. Then, when the user clicks the button "Place SELL/BUY Order", the fields inserted are validated and an error message is shown if invalid, otherwise it will ask for confirmation through a modal screen (figure 6.33) and for the PIN code as well (figure 6.34). In case of error during this process (inserted PIN is wrong) it shows a failure screen, else it sends the request to the server and shows a successful screen. Finally, this screen also has a panel at the bottom that updates when the cryptocurrency selected in the form changes. It essentially shows the history of orders that were executed for that same crypto. The information shown depends on the privacy property of each one, if it is private, it does not show anything, if semi-private, it only shows the trader's ID, and if public, it shows the trader's full name.

In the backend, when the request to create an order arrives, it first validates its parameters and then it saves the order in the database. The database has two different tables, one that keeps an orderbook for the buy orders and another for the sell orders. It was implemented an auction trade market, which is a market that has a period where it receives orders (considered open) and another period where the market is resolved and the orders are

executed (considered closed, because it can't receive orders in that period). Each order in the orderbook includes the cryptocurrency name.

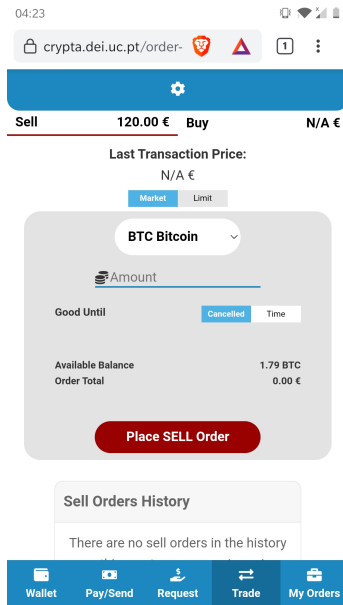


Figure 6.31: Create sell order screen

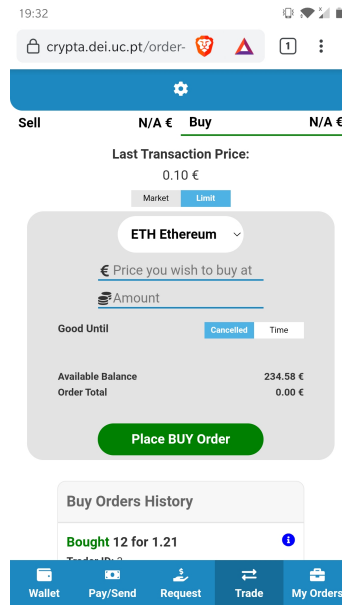


Figure 6.32: Create buy order screen

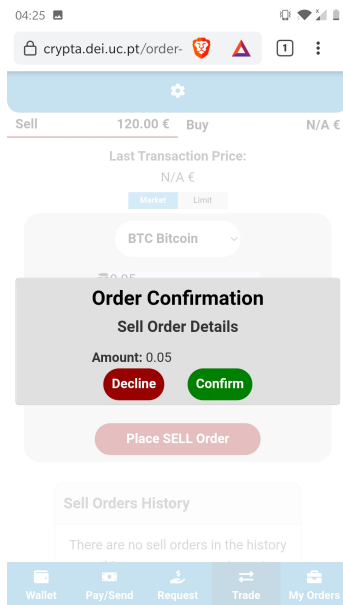


Figure 6.33: Create sell order confirmation screen

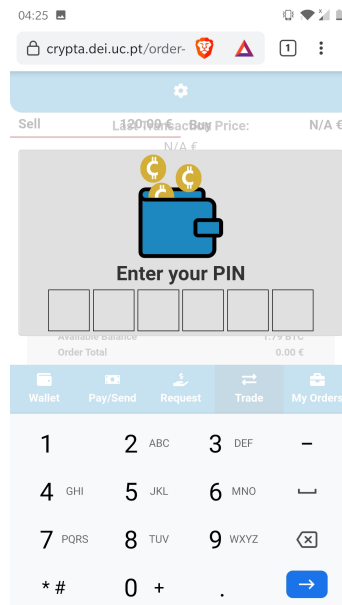


Figure 6.34: Create sell order pin insertion screen

## View open and closed orders

In the menu "My Orders", an user can view his open and closed orders (figures 6.35 and 6.36)). In the open orders menu all the orders that are still waiting to be executed are displayed with their properties, such as, order side (buy or sell), type (market or limit), amount, price, date until when the order is valid and the date of creation. On the other

hand, the closed orders screen shows all the orders that were already executed along with the side of the order (bought or sold) and execution price, amount and date.

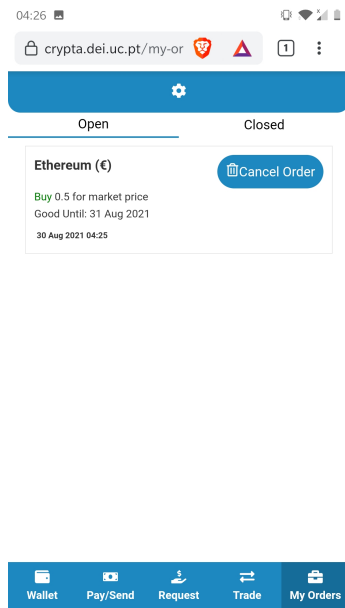


Figure 6.35: User's open orders screen

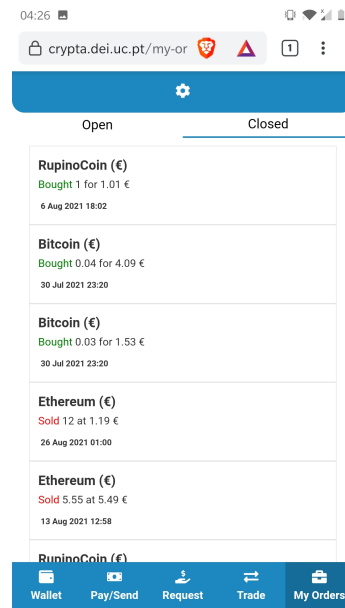


Figure 6.36: User's closed orders screen

## Settings menu

At the top toolbar, a user can press the button in the middle represented with a cogwheel to be redirected to the settings menu shown in figure 6.37. In this menu, the user can check his information (full name and email) associated to his account and access the following: "Admin Panel" (if he has administrator role, otherwise this option is hidden), "Personal Info" to edit his personal information (full name and password) (figure 6.38), "Security" to change his PIN code (figure 6.39), "Help" which has several topics with tutorials to help understand how the application functionalities work (figures 6.40 and 6.41), "About" with information about the system and "Logout" to log the user out of the app.

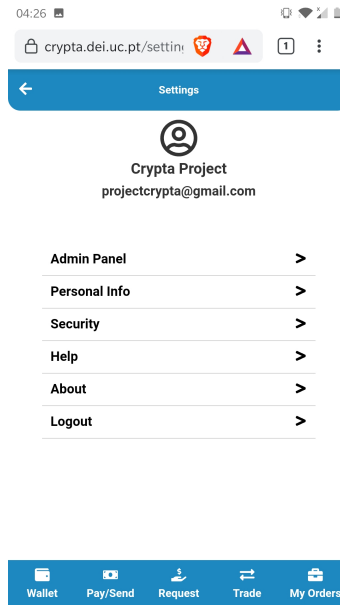


Figure 6.37: Settings menu screen

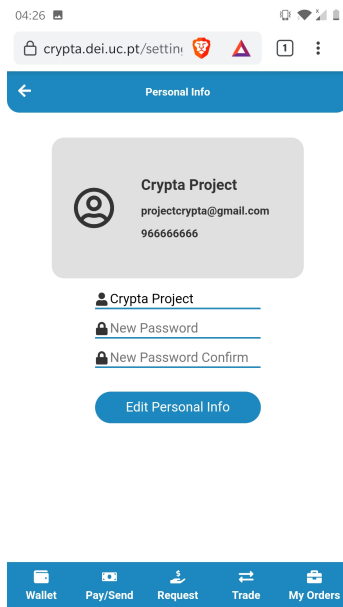


Figure 6.38: Edit personal information screen

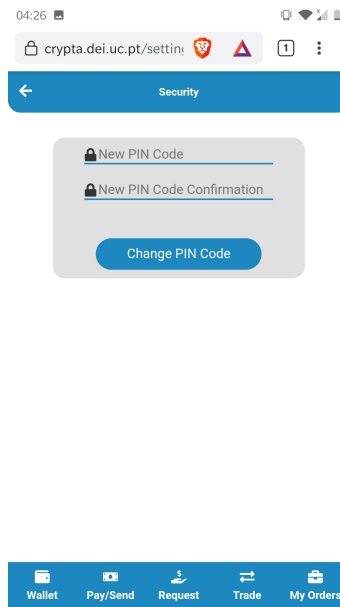


Figure 6.39: Edit security information (PIN code) screen

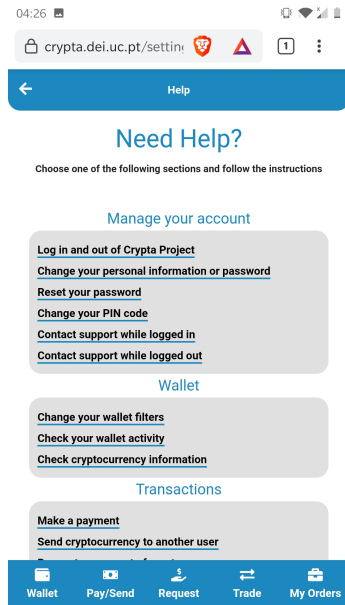


Figure 6.40: Help menu screen

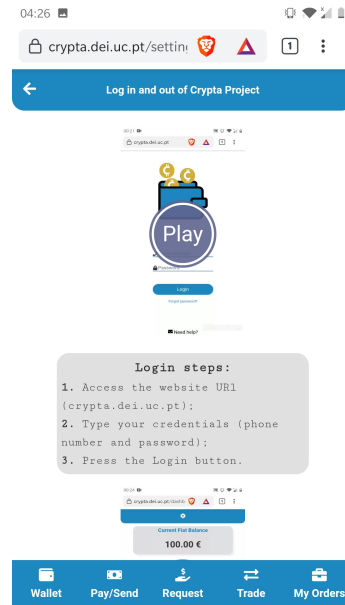


Figure 6.41: Help login's section screen

## Admin panel

In this application, there are two roles: user and admin. The only difference between them two is that admin has an extra option in the settings menu that allows him to access an administrator panel. The following subsections describe each menu in that panel.

### Users

Figure 6.42 displays users menu in the administrator panel. Here, an admin can view every user registered in the system along with their personal information. It is also possible to invite a new user. This can be done by typing a new email that is not already registered and an initial amount that corresponds to the fiat currency balance (in Euros) when the user first enters his account.

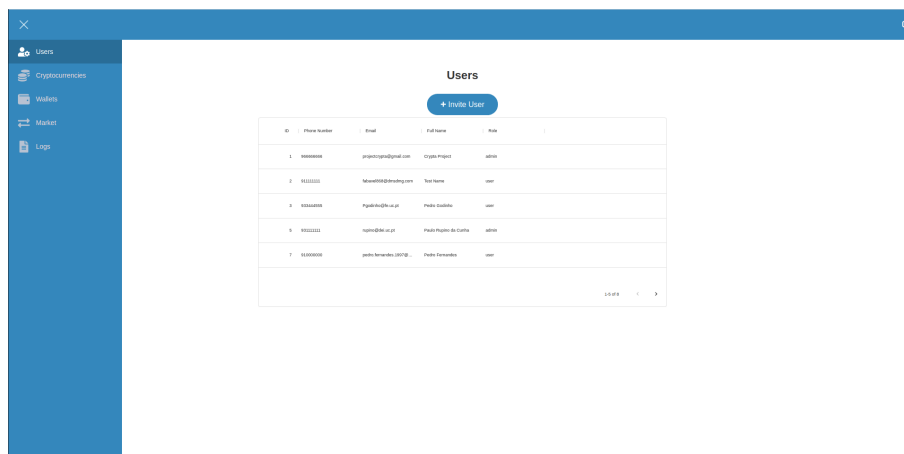


Figure 6.42: Admin panel users screen

## Cryptocurrencies

In the cryptocurrencies menu (figure 6.43), the admin can check all the cryptocurrencies present in the system, as well as their information. He can also add a new cryptocurrency by inserting the following information: name, symbol, price, cap, privacy, transaction fee and settlement time.

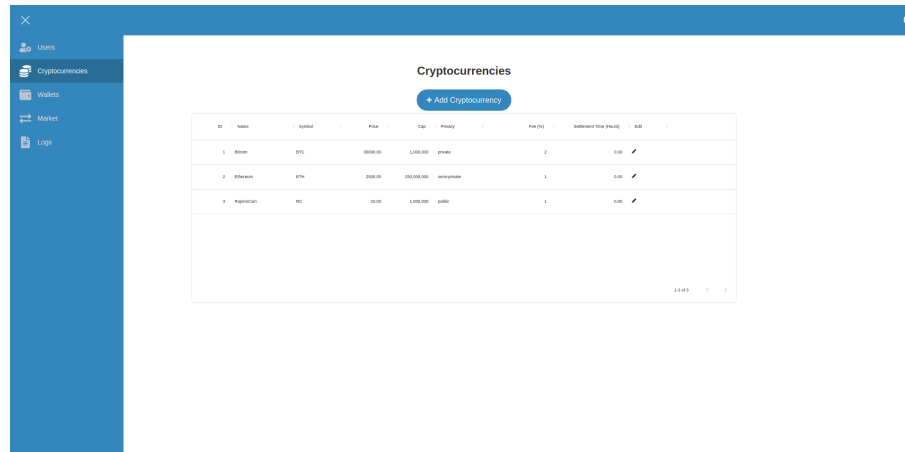


Figure 6.43: Admin panel cryptocurrencies screen

## Wallets

Figure 6.44 represents the wallets menu where administrators can check how much of each cryptocurrency each wallet has, and also edit that same amount.

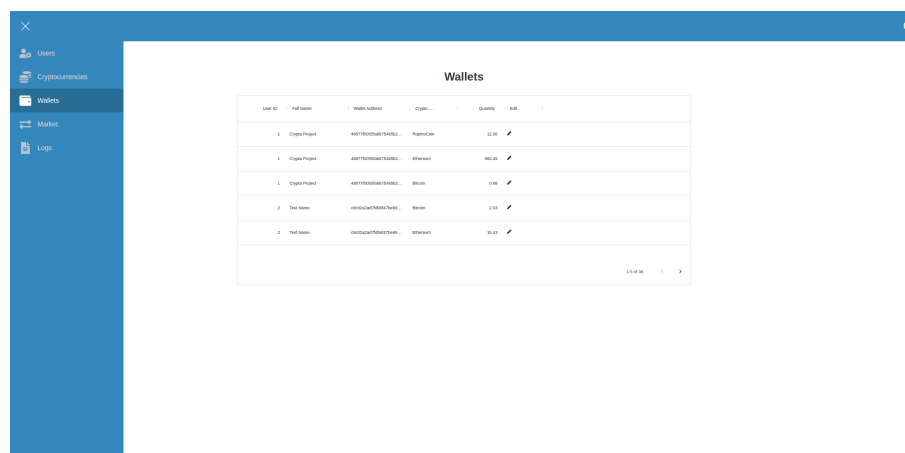


Figure 6.44: Admin panel wallets screen

## Orderbooks

In the market menu displayed in figure 6.45, it is possible to check all the orders currently waiting for market execution in the orderbooks, alongside their information. An admin can also set how many auctions should happen, by setting the number of auctions per day (maximum of 24, i.e. 1 per hour) and the number of auctions per hour (maximum of 60, i.e. 1 per minute), meaning that the maximum possible number of auctions in one day is



1440 (one auction every minute of the day). He also has the ability to instantly execute a market resolution (auction) by click the "Resolve Market Now" button.

Order ID	Order	Order Date	Side	Quantity	Price	Order	Type
0	BUY	152289977	BUY	2	102.22	Trade Market	LIMIT
1	BUY	152289982	BUY	95		Trade Market	MARKET
2	BUY	152289983	BUY	95.48		Trade Market	MARKET
3	BUY	152289988	BUY	2	102	Trade Market	LIMIT
4	BUY	152289992	BUY	0.03	102.42	Trade Market	LIMIT

Figure 6.45: Admin panel orderbooks screen

## Logs

Finally, an admin can check every log produced in the backend servers by pressing one of the buttons in the logs screen (figure 6.46) which allows him to check four levels of logs: fatal, error, warning and info. By pressing one of these buttons, a csv (comma-separated values) file is generated and downloaded to the admin computer with a list of logs of that specific level order by creation date.

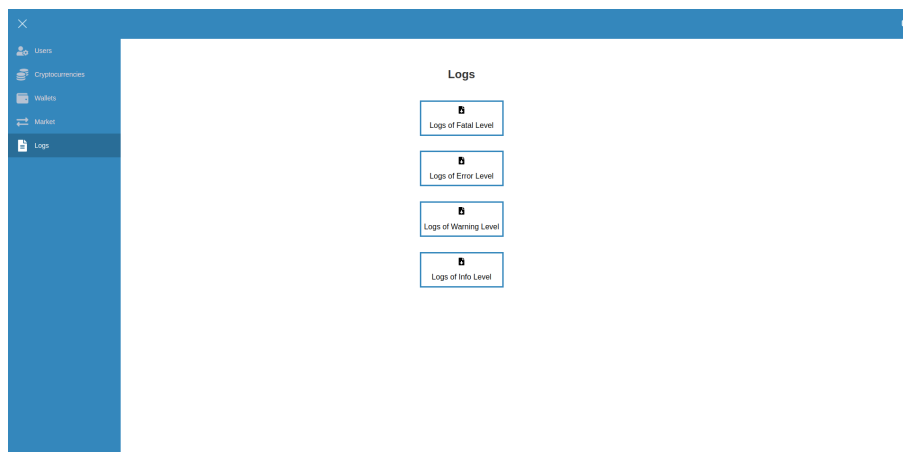


Figure 6.46: Admin panel logs screen

## 6.8 Security mechanisms implemented

This section lists the mechanisms used to meet the non-functional security requirements and make the web application more secure. The following are the strategies adopted to accomplish that:

- **Use of Transport Layer Security (TLS):** TLS is a security protocol that evolved

from an older one called Secure Sockets Layer (SSL) and is used to encrypt the communication between web applications and servers [28]. It is responsible for encryption (hides the data being transferred), authentication (makes sure that the parties exchanging data are who they claim to be) and integrity (verifies that the data was not forged or tampered) [28].

- **Access tokens with short expiration time:** For authentication purposes the jwt access tokens generated on login are stored in a cookie and have a expiration time of one hour. Even though it may look that it violates the non-functional security requirement S03 ("The system shall not store cookies on the customer's device containing sensitive information"), it does not, because the cookie only stores the jwt token, that, when decrypted, only has the user's ID, email and role, which is not considered sensitive information. Also, the use of cookies for access tokens over local storage has the advantage of not being vulnerable to cross-site scripting (XSS) attacks.
- **Encrypted password in database:** It is extremely important to keep the user's password secure. To that end, passwords are hashed before being stored in the database, this protect against attackers that manage to access the database, for example, through SQL injection.

This page is intentionally left blank.

# Chapter 7

## Testing

In 1984, *Barry W. Boehm* expressed the difference between software verification and software validation by defining the following questions: "Am I building the product right?" (verification) and "Am I building the right product?" (validation) [21]. According to the *IEEE Standard Glossary of Software Engineering Terminology* [44] the definitions of "verification" and "validation" are:

- **Verification:** *The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.*
- **Validation:** *The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.*

Regarding the first question raised by *Boehm* (*Am I building the product right?*), several tasks have been done to answer this question, such as, requirements gathering 3, where the *Software Requirements Specification* document was originated, and the software architecture design.

At this point, we need to check if the software is working like it is supposed to, in other words, we need to validate the system.

This chapter focuses on the testing techniques and its results. Firstly functional testing was performed which includes unit testing in the server side and end-to-end testing in the client side. Then, load tests were executed in order to check response time and capacity, depending on the number of requests. Finally, usability tests were conducted to get some feedback from the end-users, as well as, check the usability of the system.

### 7.1 Functional Testing

Functional testing is a type of software testing that determines if a each part of a system is working in accordance with the defined requirements [41]. In this case, we will validate each functional requirement defined in the *Software Requirements Specification* document, by doing the following techniques of functional testing: *unit testing* and *end to end testing*.

### 7.1.1 Unit Testing

Unit testing is a type of white-box testing (tests depend on both the implementation and the specification [48]) that is performed by the developers, since it is code based. It is supposed to be done by testing individual units separately, being that a unit is the smallest testable part of an application, that is usually not larger than a class [57].

In order to execute unit tests two *Node.js* libraries: *Chai* [23] which is an assertion library that provides functions and methods that help comparing the output of a certain test with its expected value, this is called an assertion. Assertions will pass when the expected value is not equal to the actual value and fail if vice versa. The other library used was *Mocha* [53] which provides functions and methods that are executed in a specific order and logs its result in the terminal. Mocha regularly uses the functions *describe* and *it* that provide structure to the tests by grouping them into test suites and test cases. A *test suite* is a group of tests related to a single functionality and a *test case* (also called a *unit test*) is a single description of a part of code that either passes or fails. Test suites use the keyword *describe* and test cases the keyword *it*. Mocha also provides tools that allow cleaning the state of the software to assure that test cases are being run independently of each other [45].

In order to perform unit testing, the following steps were conducted:

1. **Definition of each isolated functionality for testing with an input and output:** Since microservices architecture was used, where each service has several functional requirements implemented as single and isolated units, we can define our tests suites to meet each of the API endpoints which correspond to all the functional requirements, being that each test suite has multiple test cases with different inputs/outputs.
2. **Write the test suites and test cases:** The tests were conducted on the *Mocha* framework with *Chai* library that allows to make assertion on the inputs/outputs. The figures 7.1 and 7.2 are examples of two test cases for the login function, one where the credentials are invalid and the login fails and other where they are valid and the login succeeds. By those figures it is possible to check that two methods from the *Mocha* library are used to define the test suites: **describe** which gives structure to test suites by grouping test cases and **it** which is utilized to identify each individual test case. Then, from the *Chai* library, two methods are also used: **request** that makes request to each route on the server, which also allows to add parameters in the request header, and **expect** which allows to make assertions in order to check if values meet certain predefined conditions by using sub-methods, such as, "**equal**" and "**to have**".

```
describe('POST /api/auth/login', () => {
  it('it should not login user and return a bad request error (Invalid credentials, password entered is not correct)', () => {
    return chai.request(app).post('/api/auth/login')
      .send({
        phone_number: '900000000',
        password: 'Password.123456'
      })
      .then(res => {
        chai.expect(res.status).equal(400);
        chai.expect(res.body).to.have.deep.property('errors', [{ message: 'Invalid credentials' }]);
      });
  });
});
```

Figure 7.1: Test case for login with invalid credentials

```

describe('POST /api/auth/login', () => {
  it('it should login user and return a session', () => {
    return chai.request(app).post('/api/auth/login')
      .send({
        phone_number: '960000000',
        password: 'Password.12345'
      })
      .then(res => {
        console.log(res.body);
        chai.expect(res.status).equal(200);
        chai.expect(res.body).to.have.property('jwt');
        chai.expect(res.body).to.have.property('uid');
      });
  });
});

```

Figure 7.2: Test case for login with valid credentials

3. **Execution of each test suite independently and analysis of results:** After writing each test suite and test case, it is possible to run the unit tests by running the command "mocha -r ts-node/register" followed by the directory of the file with the test suites/cases. After running that command, the tests are performed and the results are displayed by listing all the tests cases executed with the time taken to perform and a cross that indicates that it passed or a cross if it failed, like shown in the following figure 7.3.

```

POST /api/auth/login
  ✓ it should not login user and return a bad request error (Invalid credentials, password entered is not correct) (43ms)
POST /api/auth/login
  ✓ it should login user and return a session (45ms)
2 passing (133ms)

```

Figure 7.3: Results for the login function unit testing

Since each microservice has several routes (API endpoints), the unit testing results are all written in the Appendix B at the end of this document.

### 7.1.2 End-to-End Testing

End-to-End testing is a technique that helps testing the application flow from the beginning until the end to make sure that it behaves as expected. The main purpose is to simulate real scenarios and validate the system and its components for integration and data integrity [90].

In order to perform this type of tests, the library *Cypress* was used, which performs end-to-end expressions. It also has the ability to run the application in the browser and provides real-time debugging. In every action during the execution of each test, a snapshot of the interface is saved, which lets the testers know what changed in between actions [34].

To use this library, a npm package was installed using the command `npm install cypress --save-dev`. Then, it is possible to open the cypress web interface by running the command `node_modules/.bin/cypress open`. In this interface all the end-to-end testing files created in the testing directory are listed and can all be executed at once. The main structure of a test file looks as follows in the figure 7.4.

```
describe('Login page', () => {
  it('it should not log in and print phone number invalid message', function () {
    cy.visit('https://192.168.1.77:8080/')
    cy.get('input[name=phone_number]').type('9666666661')
    // {enter} causes the form to submit
    cy.get('input[name=password]').type('Password.12345')
    cy.get('#login_form').submit()
    cy.get('#div_error').should('have.text', 'You must enter a valid phone number from Portugal')
  });

  it('it should log in, set session cookie and redirect to dashboard', function () {
    cy.visit('https://192.168.1.77:8080/')
    cy.get('input[name=phone_number]').type('9666666666')
    // {enter} causes the form to submit
    cy.get('input[name=password]').type('Password.12345')
    cy.get('#login_form').submit()

    // we should be redirected to /dashboard
    cy.url().should('include', '/dashboard')

    // our auth cookie should be present
    cy.getCookie('express:sess').should('exist')
  });
});
```

Figure 7.4: Example of a end-to-end testing file for login

The main methods used in the configuration of the test files for end-to-end testing are [77]:

- **visit:** allows to access a remote URL.
- **get:** used to get one or more HTML DOM (Document Object Model) elements by selector or name.
- **focused:** gets the DOM element that is currently focused.
- **type:** types into a DOM element that was previously selected with the commands `get` or `focused`.
- **select:** selects an option in a *select* DOM element.
- **submit:** utilized to submit a *form* element.
- **click:** clicks a DOM element.
- **should:** used to create an assertion, usually takes two arguments, a chainer (for example, *have.text* or *include*) that works as a condition and a value to assert against that chainer.
- **url:** gets the URL of the current page.
- **getCookie:** gets a browser cookie by its name.

Figure 7.5 above shows how a result is displayed when a test file is executed with cypress in the web interface. The complete results for this type of integration tests will be presented in tables at the end of this document in the Appendix C.

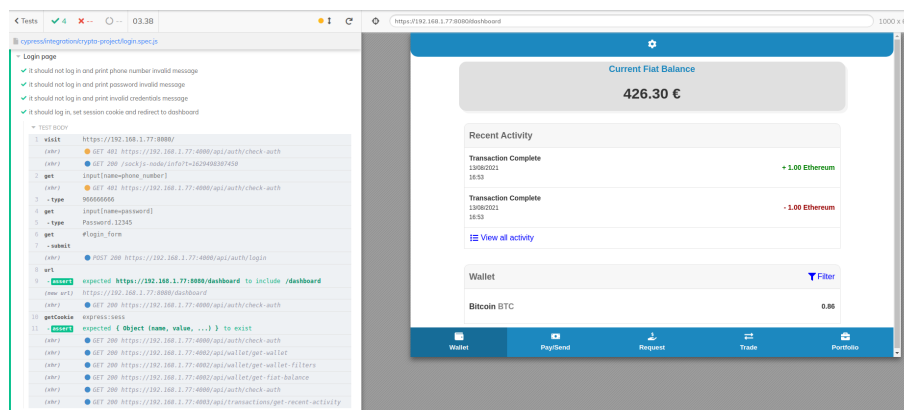


Figure 7.5: Example of a end-to-end testing result for login

## 7.2 Load Testing

Load testing is a technique of non-functional testing with the main goal of testing the performance of a software application under varying load conditions by simulating multiple users using the application concurrently. This technique allows the measuring of response times, throughput rates, resource utilization levels and identify the application's breaking point, assuming that that point is below the peak load condition.

In order to start the staging phase, the application (server and client side) were deployed to a VM (Virtual Machine) to ensure that everyone involved in the project would be able to test it as well. This VM has the following specification: 4 VCPUs, 4 GB of RAM and 50 GB of storage.

Although the application will not be released to the public during the duration of the project, it is crucial to understand how well the system performs under certain load conditions and how many number of users and server requests it can handle. This analysis will make it easier to verify if a upgrade to the VM is needed when the app goes into a final stage of deployment to the public. When conducting this analysis, the most important step is to make sure that it complies with the non functional performance requirement PF02 ("The system shall execute transactions in less than 3 seconds").

To perform this test, the library *ApacheBench* (*ab*) was used, which is a tool capable of testing HTTP servers [5]. In order to perform this test, several parameters had to be input, such as: number of total requests, number of concurrent requests (sent in the same moment), a cookie with the JWT token to authenticate the user sending the request, a JSON file with the body of the request, the content type of the data sent in the request and the URL of the API endpoint to test. In this case the command used to load test is as follows: *ab -n <number of total requests> -c <number of concurrent requests> -C <cookie with JWT token> -p <path for directory with request body> -T <content type of the request> <url of the API endpoint to test>*.

Since this system has several API endpoints, only the one which is expected to deal with the bigger traffic will be tested. In this case, the one chosen was the endpoint responsible for making a cryptocurrency payment (*/api/transactions/make-payment*). The following table 7.6 presents the test outputs from the tool used when dealing with a certain number of concurrent requests.



Load Testing for Make Payment API Endpoint				
Number of requests	Number of concurrent requests	Number of requests per second (average)	Average time per request (milliseconds)	Maximum time per request (milliseconds)
10	1	3.43	291.777	362
50	5	8.20	609.765	713
100	10	9.92	1007.603	1297
150	15	11.40	1315.711	1587
200	20	12.05	1659.671	2103
300	30	11.99	2501.104	3179
350	35	12.66	2764.823	4022
400	40	12.57	3182.702	4820

Figure 7.6: Load testing results for the make a cryptocurrency payment API endpoint

One of the first observations that can be taken from the table 7.6 is the fact that, from the line with 10 concurrent requests made forward the number of requests per second (average) stabilized at 12, which means that this endpoint can only process that number of requests per second, approximately. Other conclusion we can take from the average time per request (in milliseconds) is that it crosses the 3 seconds mark (3000 ms) when sending 40 concurrent requests, breaking the non functional performance request (PF02) set, which says that "the system shall execute transactions in less than 3 seconds". We can also check that the maximum time per request (in milliseconds) surpasses the 3 second mark by a large margin when more than 30 concurrent requests are sent. With this in mind and considering the context of this project, which states that the application will be used by a limited number of users from the *Faculdade de Economia da Universidade de Coimbra*, it is safe to say that is very unlikely that 30 requests will be made concurrently to the same endpoint. In case of need to expand the number of users, we can scale the system by upgrading the specifications of the virtual machine by adding more processing speed (CPU) and memory RAM.

## 7.3 Usability Testing

After performing the functional testing to make sure that the application has the minimum number of bugs possible, we can now perform the non-functional usability testing. This technique is used to get an external perspective of the application being developed since it is performed by real people that then give their feedback. Since the scope of this project is a web application that will eventually be used by a real audience, its usability is a extremely important factor to consider.

To conduct the usability tests, four participants were chosen with fairly different backgrounds. Their ages range from early 20s to early 50s. Their academic qualifications include high school, bachelor's degree and master's degree. We also looked to find testers with and without a technological background. And the most important aspect is that none of the testers had previous knowledge of this application.

### 7.3.1 Test Methodology

The tests were performed in a quiet environment to ensure maximum focus and supervised to guarantee the veracity of the results. In order to perform the test a *Google Forms*<sup>1</sup> was created and its template is in the Appendix D. The test (form) starts with an brief explanation of the scope of this project and what the form is about (what is usability testing). Then the tester needs to fill a questionnaire with his age, academic qualifica-

<sup>1</sup><https://www.google.com/forms/about/>

---

tions, academic field (if applicable), current job and 6 questions that help understand the knowledge and experience of the tester with mobile application, mobile wallets and trading markets. Then, he is asked to perform 17 tasks in the application and, for each one, fill with the number of clicks necessary to complete, the task's success (success or fail) and the difficulty level (from 1 to 5, being 1 very easy and 5 very hard).

The tasks asked to perform were the following:

1. Register into the application through the email received.
2. Login into the application.
3. Change wallet filters to see only the first cryptocurrency in the list.
4. Check all activity in the wallet.
5. Make payment with the QR code below.
6. Send 0.01 Bitcoin (BTC) to the phone number 966666666.
7. Generate a QR code to request 0.01 Bitcoin (BTC).
8. Create a market sell order for 0.01 Bitcoin (BTC) that is only good until tomorrow.
9. Create a limit buy order for 0.01 Bitcoin (BTC) at 14.42€ that is good until cancelled.
10. Check your open and then closed orders.
11. Cancel the market sell order for 0.01 Bitcoin (BTC).
12. Edit your personal information (full name).
13. Edit your security settings (PIN code).
14. Check the instructions to reset your password.
15. Visit the "About" section of the application.
16. Contact the application support team and send a message with subject "Greeting" and message "Hello team."
17. Logout of the application.

Finally, after performing every task asked, the tester needs to fill a pos-questionnaire which helps understand how he felt about using the mobile web application and to get some feedback.

### 7.3.2 Test Results

#### Questionnaires

The figures below represent the graphs generated from the answers to the questions asked to the testers before performing the tasks. They were asked in order to understand how much knowledge and experience each tester had of mobile application, trading markets and mobile wallets. The following graphs range from 1 to 5, being that, the higher the value the more experienced the tester is with the subject asked in the questionnaire.

Rank your experience with mobile web applications.  
4 respostas

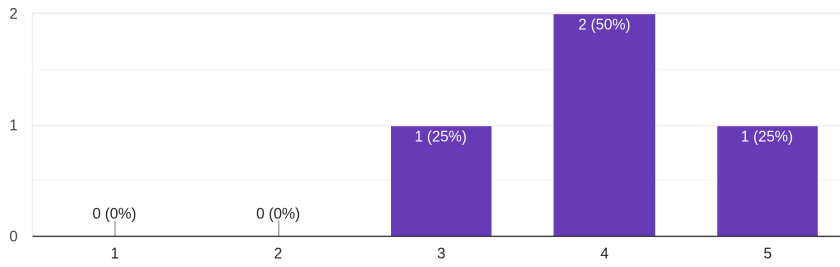


Figure 7.7: Questionnaire first question

Rank your knowledge of cryptocurrencies.  
4 respostas

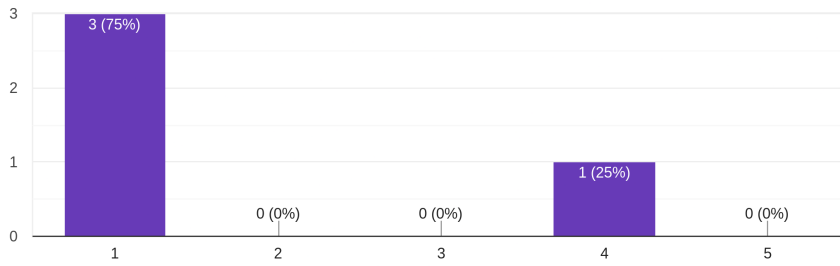


Figure 7.8: Questionnaire second question

Rank your knowledge of trading markets.  
4 respostas

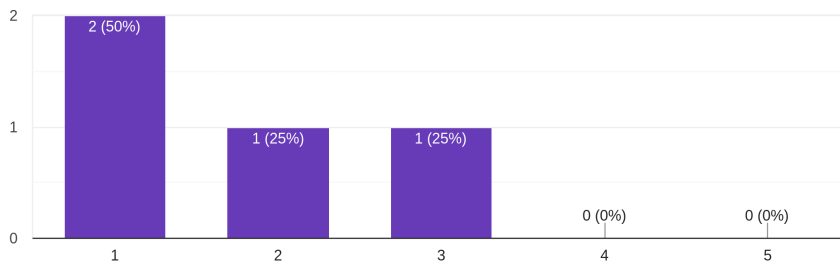


Figure 7.9: Questionnaire third question

Rank your knowledge of mobile wallets.

4 respostas

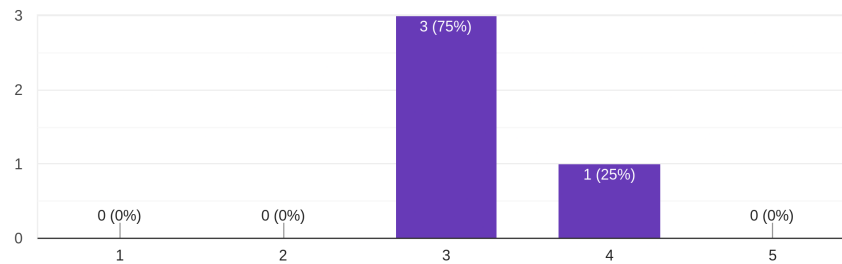


Figure 7.10: Questionnaire fourth question

Rank your experience with mobile market trading applications.

4 respostas

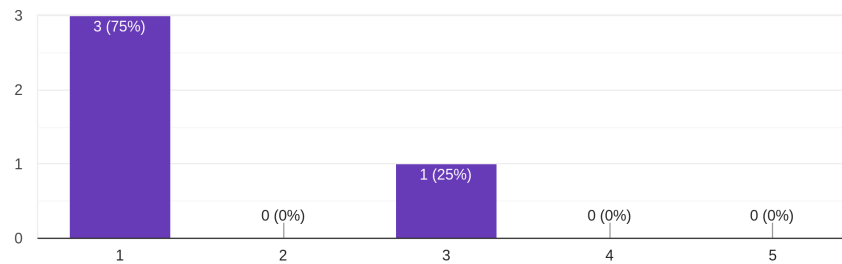


Figure 7.11: Questionnaire fifth question

Rank your experience with mobile wallet applications.

4 respostas

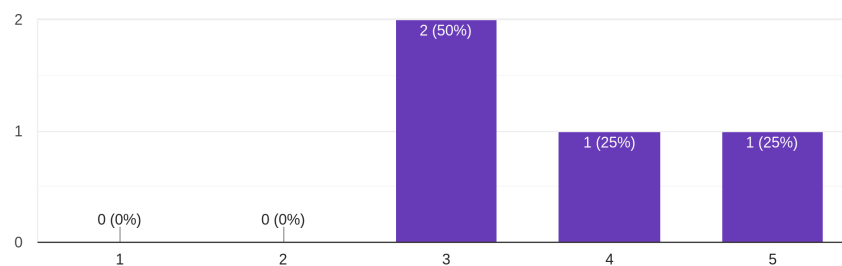


Figure 7.12: Questionnaire sixth question

Overall all the testers have some experience with mobile applications, though most of the testers had very few knowledge and experience with trading markets. Also, in general, the testers had some knowledge and experience with mobile wallets and, when asked about it, most of them said that they use *MB Way*.

## Tasks Performance

In order to better understand the results obtained from the usability tests two tables were made. One that shows the expected number of clicks per task, which considers that a user performed the task without any mistakes, and the actual number of clicks per task for each tester (figure 7.13). And another one that represents the difficulty level per task that each tester felt (figure 7.14).

Number of clicks for each task (by tester)																	
Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Expected	9	3	4	1	4	9	6	9	10	2	2	4	4	2	3	4	1
Tester 1	9	5	4	1	4	8	5	12	9	2	2	4	4	3	3	5	3
Tester 2	14	3	5	1	8	12	9	X	X	2	X	8	5	3	3	4	1
Tester 3	10	3	4	2	6	11	6	14	12	2	2	6	5	4	3	4	1
Tester 4	10	2	4	1	4	9	7	10	10	2	2	5	3	3	3	2	1

Figure 7.13: Usability testing tasks results (clicks)

Regarding the number of clicks it is considered that a task, that took more than two clicks than expected (margin for error), is abnormal and marked with red in the table.

In the table above it is possible to check that most testers required an excessive number of clicks to complete the task 8 (creating a market sell order), subsequently the task 9, despite being very similar to the previous one, did not take as much clicks. This can be explained by the fact that testers learned it by exploring around doing the task 8. Another explanation for this is also the fact that those testers did not have much knowledge and experience with trading markets like seen in the questionnaires (figures 7.9 and 7.11). There is also one tester that has a lot of outliers and even could not complete the tasks 8 and 9 (subsequently making the task 11 undoable), which can also be justified by the his knowledge and experience since this tester does not have much contact with newer technology and does not know and experienced mobile wallets and trading markets at all.

Difficulty level for each task (by tester)																	
Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Tester 1	1	1	1	1	1	1	1	4	3	1	1	1	1	1	1	1	1
Tester 2	3	1	1	1	3	2	2	5	5	1	1	3	1	1	1	1	1
Tester 3	1	1	1	1	1	1	1	5	3	1	1	2	1	1	1	1	1
Tester 4	1	1	1	1	1	1	1	2	2	1	1	1	1	1	1	1	1

Figure 7.14: Usability testing tasks results (difficulty level)

For the difficulty level of each task, the same methodology was used, where every outlier is marked with red and correspond to the task's difficulty level above than neutral (hard or very hard).

By the table, we can take the same conclusions, where the task 8 was considered the most difficult and the tester 2 was the one that found the tasks harder.

## Pos-Questionnaire

The following figures display the graphs generated from the answers to the questions asked to the testers after performing the tasks. In this case, testers have to select an option (from *strongly disagree* to *strongly agree*) for each question.

The application was easy to understand.  
4 respostas

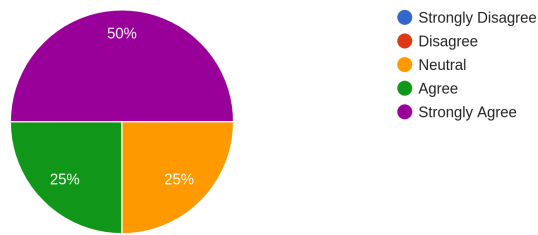


Figure 7.15: Pos-Questionnaire first question

The application was easy to use (complete all the tasks).  
4 respostas

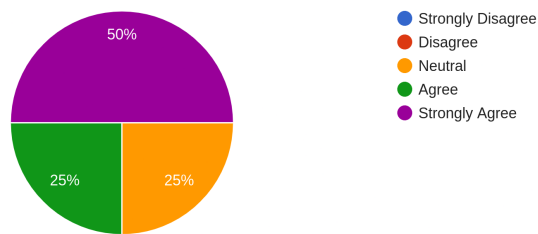


Figure 7.16: Pos-Questionnaire second question

If you would open the application tomorrow, you could still find and use all the functionalities.  
4 respostas

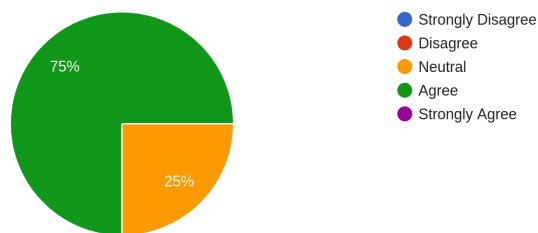


Figure 7.17: Pos-Questionnaire third question

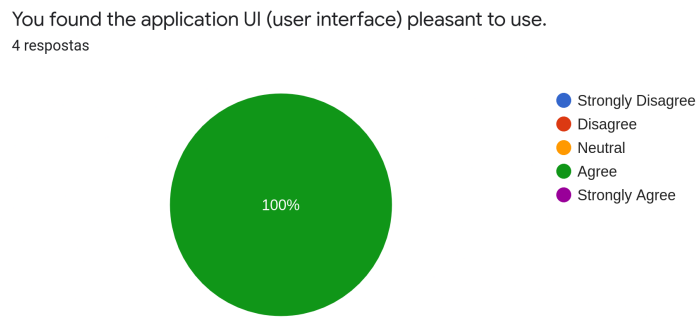


Figure 7.18: Pos-Questionnaire forth question

This questionnaire looks to understand how the tester felt about using the application and to get some feedback. Overall, the testers found the application easy to understand and use (figures 7.15 and 7.16, with exception of the tester 2 which had trouble doing the trading market related tasks. Regarding the third question three of the users fairly confident that they could still find and use all the functionalities if they would to open the application tomorrow, with the second tester being again neutral because of his lack of experience with mobile applications. Finally, all the users agreed that the application UI (user interface) was pleasant to use, but was not a strong agreement which means that there is still room to improve the design.

### Recommendations to improve the application

Lastly, it was asked the testers to provide some recommendations that they think could improve the application, obtaining the following feedback:

- Tester 1
  - Improve the user interface.
- Tester 2
  - Make the tasks related to the orders easier to understand.
- Tester 3
  - Make design more pleasant;
  - Make the market menu easier to use and understand.
- Tester 4
  - It would be nice if the presentation of the menu market was more intuitive to use.

This page is intentionally left blank.



## Chapter 8

# Conclusion and Future Work

The main purpose of this work was to develop a system capable of making cryptocurrencies transactions and trading in a simulated market. To achieve this, a state of the art study was conducted where the main focal points were: the approaches to adopt in the frontend development, the proper software architecture pattern for the backend of the application and security considerations for web applications. After that, the functional requirements were collected through the analysis of the graphical interfaces of existing applications and feedback given from the stakeholders (professors) and a "Software Requirements Specification" document was redacted with those requirements (nine general, fifteen user and five administrator functional requirements), along with thirteen non-functional requirements, fourteen use cases and mockups for every user functionality defined. Then the technologies that will be used for the development of the application were defined. This definition was done by comparison of different technologies or, in some cases, simply defined, when there was only one technology considered. The technological decisions made were: Progressive web application framework - ReactJS, backend technology - ExpressJS, relational database - MySQL, non-relational database - MongoDB, containerization technology - Docker, message queue system - Apache Kafka and SMS communication API - Twilio. Then, a software architecture was designed resorting to the C4 model. The first three levels of this model were considered and three diagrams for this system were generated: System Context, Container and Component. After that and before starting the implementation, the risk analysis and development methodology definition were done. Finally the application was implemented and validated by using functional testing (unit and end-to-end testing) where all tests passed successfully, load testing, that made it possible to check that the system can take up to 12 requests per second and that when 40 concurrent requests are sent the average time per request is more than 3 seconds, and usability testing, which made it clear that the application needs user-interface improvements and the market trading menu needs to be easier to use and understand.

In terms of future work and even though, the application was completely implemented, there is still some room for improvement. Regarding the front-end and taking into consideration the feedback from the usability tests, the design of the web application could be upgraded or even redone by a group of qualified designers to make it as pleasant as possible for the participants of the experiment (final deployment). Regarding the back-end, there is also several improvable aspects, such as, the integration with the SMS communication API, which was not implemented and replaced with email communication and the creation of several server endpoints which allows the system administrators to get more particular information about the application, specifically about the trading market and the transactions performed. Finally, one more general aspect that should be addressed in the future,

that is the testing of the system as a whole, by a controlled number of participants, with fake money. This would help better understand how the system behaves in a real scenario and if it is ready to be deployed and start the actual experimentation.

# References

- [1] 106 Spring Boot interview questions, n.d. [Online]. Available from: <https://www.javapedia.net/Spring-Boot#qanda1248> [Accessed 2021-01-16].
- [2] 3 Pros and 3 Cons of Working with Docker Containers - Sweetcode.io, n.d. [Online]. Available from: <https://sweetcode.io/3-pros-3-cons-working-docker-containers/> [Accessed 2021-01-16].
- [3] 5 Advantages and Disadvantages of Client Server Network | Drawbacks & Benefits of Client Server Network, n.d. [Online]. Available from: <https://www.hitechwhizz.com/2020/11/5-advantages-and-disadvantages-drawbacks-benefits-of-client-server-network.html> [Accessed 2021-01-08].
- [4] 7 Progressive Web App Development Frameworks to Know in 2021 - GeeksforGeeks, n.d. [Online]. Available from: <https://www.geeksforgeeks.org/7-progressive-web-app-development-frameworks-to-know-in-2021/> [Accessed 2021-01-16].
- [5] ab - Apache HTTP server benchmarking tool - Apache HTTP Server Version 2.4, n.d. [Online]. Available from: <https://httpd.apache.org/docs/2.4/programs/ab.html> [Accessed 2021-08-24].
- [6] Add to Home screen - Progressive web apps (PWAs) | MDN, n.d. [Online]. Available from: [https://developer.mozilla.org/en-US/docs/Web/Progressive\\_web\\_apps/Add\\_to\\_home\\_screen](https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Add_to_home_screen) [Accessed 2021-01-04].
- [7] Advantages and Disadvantages of Docker - Learn Docker - DataFlair, n.d. [Online]. Available from: <https://data-flair.training/blogs/advantages-and-disadvantages-of-docker/> [Accessed 2021-01-16].
- [8] Advantages and Disadvantages of Kafka - DataFlair, n.d. [Online]. Available from: <https://data-flair.training/blogs/advantages-and-disadvantages-of-kafka/> [Accessed 2021-01-16].
- [9] Advantages of MongoDB | Disadvantages of MongoDB - DataFlair, n.d. [Online]. Available from: <https://data-flair.training/blogs/advantages-of-mongodb/> [Accessed 2021-01-16].
- [10] Agile methodologies enhance Appian delivery: Part 1 - Bits In Glass, n.d. [Online]. Available from: <https://bitsinglass.com/agile-methodologies-enhance-appian-delivery-part-1/> [Accessed 2021-08-31].
- [11] All About MongoDB NoSQL Database: Advantages and Disadvantages, n.d. [Online]. Available from: <https://acodez.in/mongodb-nosql-database/> [Accessed 2021-01-16].

- 
- [12] Ameya, N., Anil, P. and Dikshay, P., 2013. Type of NOSQL databases and its comparison with relational databases. *International journal of applied information systems*, 5(January 2013), pp.16–19.
- [13] Anatomy of the Client/Server Model, n.d. [Online]. Available from: [https://docs.oracle.com/cd/E13203\\_01/tuxedo/tux80/atmi/intbas3.htm](https://docs.oracle.com/cd/E13203_01/tuxedo/tux80/atmi/intbas3.htm) [Accessed 2021-01-08].
- [14] Apache Kafka, n.d. [Online]. Available from: <https://kafka.apache.org/> [Accessed 2021-08-27].
- [15] Apache Kafka Advantage Disadvantage - Beyond Corner, n.d. [Online]. Available from: <https://beyondcorner.com/learn-apache-kafka-tutorial/apache-kafka-advantage-disadvantage/> [Accessed 2021-01-16].
- [16] Apache Kafka Advantages and Disadvantages - javatpoint, n.d. [Online]. Available from: <https://www.javatpoint.com/apache-kafka-advantages-and-disadvantages> [Accessed 2021-01-16].
- [17] Apache ZooKeeper, n.d. [Online]. Available from: <https://zookeeper.apache.org/> [Accessed 2021-08-27].
- [18] Bayse, G., 2021. *SANS Institute Information Security Reading Room A Security Checklist for Web Application Design*.
- [19] Biørn-Hansen, A. and Ghinea, G., 2018. Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development. *Proceedings of the 51st hawaii international conference on system sciences* [Online], (August). Available from: <https://doi.org/10.24251/hicss.2018.716>.
- [20] Biørn-Hansen, A., Majchrzak, T.A. and Grønli, T.M., 2018. *Progressive web apps for the unified development of mobile applications*, vol. 322. Springer International Publishing. Available from: [https://doi.org/10.1007/978-3-319-93527-0\\_4](https://doi.org/10.1007/978-3-319-93527-0_4).
- [21] Boehm, B.W., 1984. Verifying and Validating Software Requirements and Design Specifications. *Ieee software* [Online], 1(1), pp.75–88. Available from: <https://doi.org/10.1109/MS.1984.233702>.
- [22] Cervone, H.F., 2011. Understanding agile project management methods using Scrum. *Oclc systems and services* [Online], 27(1), pp.18–22. Available from: <https://doi.org/10.1108/106507511111106528>.
- [23] Chai, n.d. [Online]. Available from: <https://www.chaijs.com/> [Accessed 2021-08-18].
- [24] Choosing Python Web Frameworks: Django and Flask - Coding Dojo Blog, n.d. [Online]. Available from: <https://www.codingdojo.com/blog/choosing-python-web-frameworks> [Accessed 2021-01-16].
- [25] Ciman, M. and Gaggi, O., 2017. An empirical analysis of energy consumption of cross-platform frameworks for mobile development. *Pervasive and mobile computing* [Online], 39, pp.214–230. Available from: <https://doi.org/10.1016/j.pmcj.2016.10.004>.
- [26] Client-Server Architectures, n.d. [Online]. Available from: [http://www.cs.sjsu.edu/~sim\\$pearce/oom/ood/distArch/server](http://www.cs.sjsu.edu/~sim$pearce/oom/ood/distArch/server) [Accessed 2021-01-08].

- [27] Client Server Model - CIO Wiki, n.d. [Online]. Available from: [https://cio-wiki.org/wiki/Client\\_Server\\_Model](https://cio-wiki.org/wiki/Client_Server_Model) [Accessed 2021-01-08].
- [28] CloudFlare, 2021. What is Transport Layer Security? | TLS protocol | Cloudflare UK [Online]. Available from: <https://www.cloudflare.com/en-gb/learning/ssl/transport-layer-security-tls/> [Accessed 2021-08-31].
- [29] Delia, L., Thomas, P., Corbalan, L., Sosa, J.F., Cuitiño, A., Cáseres, G. and Pesado, P., 2019. Development approaches for mobile applications: Comparative analysis of features. *Advances in intelligent systems and computing* [Online], 857(January), pp.470–484. Available from: [https://doi.org/10.1007/978-3-030-01177-2\\_34](https://doi.org/10.1007/978-3-030-01177-2_34).
- [30] Difference between stateful and stateless server | Practice | GeeksforGeeks, n.d. [Online]. Available from: <https://practice.geeksforgeeks.org/problems/difference-between-stateful-and-stateless-server> [Accessed 2021-01-08].
- [31] Discussing Docker. Pros and Cons., n.d. [Online]. Available from: <https://phauer.com/2015/discussing-docker-pros-and-cons/> [Accessed 2021-01-16].
- [32] Docker Containers: The Pros and Cons of Docker | The Iron.io Blog, n.d. [Online]. Available from: <https://blog.iron.io/docker-containers-the-pros-and-cons-of-docker/> [Accessed 2021-01-16].
- [33] El-Kassas, W.S., Abdullah, B.A., Yousef, A.H. and Wahba, A.M., 2017. Taxonomy of Cross-Platform Mobile Applications Development Approaches. *Ain shams engineering journal* [Online], 8(2), pp.163–190. Available from: <https://doi.org/10.1016/j.asej.2015.08.004>.
- [34] End to End Testing Framework | cypress.io, n.d. [Online]. Available from: <https://www.cypress.io/how-it-works> [Accessed 2021-08-20].
- [35] Express.js Mobile App Development: Pros and Cons for Developers | Binariks, n.d. [Online]. Available from: <https://binariks.com/blog/tools/express-js-mobile-app-development-pros-cons-developers/> [Accessed 2021-01-16].
- [36] Express.js Mobile App Development: pros and cons of Node.js framework, n.d. [Online]. Available from: <https://apiko.com/blog/express-mobile-app-development/> [Accessed 2021-01-16].
- [37] ExpressJS vs. Flask – Sweetcode.io, n.d. [Online]. Available from: <https://sweetcode.io/espressjs-flask-sweetcode-app-dev/> [Accessed 2021-01-16].
- [38] File Structure – React, n.d. [Online]. Available from: <https://reactjs.org/docs/faq-structure.html> [Accessed 2021-08-29].
- [39] Five Advantages & Disadvantages Of MySQL, n.d. [Online]. Available from: <https://www.datarealm.com/blog/five-advantages-disadvantages-of-mysql/> [Accessed 2021-01-16].
- [40] Fortunato, D. and Bernardino, J., 2018. Progressive web apps: An alternative to the native mobile Apps | Progressive Web Apps: uma alternativa às Apps móveis nativas. *Iberian conference on information systems and technologies, cisti*, 2018-June, pp.1–6.
- [41] Functional Testing: A Complete Guide with Types and Example, n.d. [Online]. Available from: <https://www.softwaretestinghelp.com/guide-to-functional-testing/> [Accessed 2021-08-17].

- 
- [42] Gaunt, M., n.d. Introdução aos service workers | Web Fundamentals | Google Developers [Online]. Available from: <https://developers.google.com/web/fundamentals/primers/service-workers> [Accessed 2021-01-08].
- [43] Getting Started – React, n.d. [Online]. Available from: <https://reactjs.org/docs/getting-started.html> [Accessed 2021-01-16].
- [44] Institute of Electrical and Electronics Engineers, 1990. IEEE Standard Glossary of Software Engineering Terminology. *Office* [Online], 121990(1), p.1. Available from: [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=159342](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=159342).
- [45] Introduction to Testing with Mocha and Chai | Codecademy, n.d. [Online]. Available from: <https://www.codecademy.com/articles/bapi-testing-intro> [Accessed 2021-08-18].
- [46] Introduction to the DOM - Web APIs | MDN, n.d. [Online]. Available from: [https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model/Introduction](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction) [Accessed 2021-01-08].
- [47] Jobe, W., 2013. Native Apps Vs. Mobile Web Apps. *International journal of interactive mobile technologies (ijim)* [Online], 7(4), p.27. Available from: <https://doi.org/10.3991/ijim.v7i4.3226>.
- [48] Kapfhammer, G.M., 2004. Software testing. *Computer science handbook, second edition* [Online], pp.105–1–105–43. Available from: <https://doi.org/10.1201/b11362-16>.
- [49] Learn About the Microservices Architecture, n.d. [Online]. Available from: <https://docs.oracle.com/en/solutions/learn-architect-microservice/index.html#GUID-1A9ECC2B-F7E6-430F-8EDA-911712467953> [Accessed 2021-01-06].
- [50] Leavitt, N., 2010. Will NoSQL Databases Live Up to Their Promise? *Computer* [Online], 43(2), pp.12–14. Available from: <https://doi.org/10.1109/mc.2010.58>.
- [51] Microservice Architecture Examples and Diagram - DevTeam.Space, n.d. [Online]. Available from: <https://www.devteam.space/blog/microservice-architecture-examples-and-diagram/> [Accessed 2021-01-08].
- [52] Microservices architecture style - Azure Application Architecture Guide | Microsoft Docs, n.d. [Online]. Available from: <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices> [Accessed 2021-01-08].
- [53] Mocha - the fun, simple, flexible JavaScript test framework, n.d. [Online]. Available from: <https://mochajs.org/> [Accessed 2021-08-18].
- [54] MVC Architecture in 5 minutes: a tutorial for beginners, n.d. [Online]. Available from: <https://www.educative.io/blog/mvc-tutorial> [Accessed 2021-01-08].
- [55] MySQL Advantages and Disadvantages - techstrikers.com, n.d. [Online]. Available from: <https://www.techstrikers.com/MySQL/advantages-and-disadvantages-of-mysql.php> [Accessed 2021-01-16].
- [56] Nakamoto, S., n.d. *Bitcoin: A Peer-to-Peer Electronic Cash System* [Online]. Available from: [www.bitcoin.org](http://www.bitcoin.org).

- [57] Nidhra, S., 2012. Black Box and White Box Testing Techniques - A Literature Review. *International journal of embedded systems and applications* [Online], 2(2), pp.29–50. Available from: <https://doi.org/10.5121/ijesa.2012.2204>.
- [58] O que são microsserviços? | AWS, n.d. [Online]. Available from: <https://aws.amazon.com/pt/microservices/> [Accessed 2021-01-08].
- [59] Patel, P., n.d. PWA vs Hybrid App vs Native: Choosing the Right Mobile App | by Priyesh Patel | Bits and Pieces [Online]. Available from: <https://blog.bitsrc.io/4-ways-to-build-your-mobile-app-make-the-right-choice-efe079c7c817> [Accessed 2021-01-03].
- [60] PostgreSQL: features of the open source database - IONOS, n.d. [Online]. Available from: <https://www.ionos.com/digitalguide/server/know-how/postgresql/> [Accessed 2021-01-16].
- [61] PostgreSQL vs. MySQL: Know The Major Differences, n.d. [Online]. Available from: <https://www.simplilearn.com/tutorials/sql-tutorial/postgresql-vs-mysql> [Accessed 2021-01-16].
- [62] Progressive Web Apps: How to Choose the Best Framework? | Danavero, n.d. [Online]. Available from: <https://danavero.com/blog/progressive-web-apps-how-choose-best-framework> [Accessed 2021-01-16].
- [63] Pros and Cons of Node.js Web App Development | AltexSoft, n.d. [Online]. Available from: <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/> [Accessed 2021-01-16].
- [64] Pros and Cons of Xamarin vs Native Mobile Development | AltexSoft, n.d. [Online]. Available from: <https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/> [Accessed 2020-12-31].
- [65] Python Flask: pros and cons - DEV Community, n.d. [Online]. Available from: <https://dev.to/detimo/python-flask-pros-and-cons-1m1o> [Accessed 2021-01-16].
- [66] Rahul Raj, C.P. and Tolety, S.B., 2012. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. *2012 annual ieee india conference, indicon 2012* [Online], pp.625–629. Available from: <https://doi.org/10.1109/INDCON.2012.6420693>.
- [67] Relational Data Model in DBMS: Concepts, Constraints, Example, n.d. [Online]. Available from: <https://www.guru99.com/relational-data-model-dbms.html> [Accessed 2021-01-04].
- [68] Relational Database Advantages | 8 Advantages of Relational Database, n.d. [Online]. Available from: <https://www.educba.com/relational-database-advantages/> [Accessed 2021-01-04].
- [69] Richard, S. and LePage, P., 2020. What makes a good Progressive Web App? [Online]. Available from: <https://web.dev/pwa-checklist/> [Accessed 2020-10-29].
- [70] Ripkens, B., 2014. Ionic: An AngularJS based framework on the rise. *2014-11-28* [Online]. Available from: <https://blog.codecentric.de/en/2014/11/ionic-angularjs-framework-on-the-rise/>.

- 
- [71] Risk Management in Software Development and Software Engineering Projects, n.d. [Online]. Available from: <https://www.castsoftware.com/research-labs/risk-management-in-software-development-and-software-engineering-projects> [Accessed 2021-06-01].
- [72] Russel, A., 2015. Progressive Web Apps: Escaping Tabs Without Losing Our Soul – Infrequently Noted. Available from: <https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/> [Accessed 2020-10-29].
- [73] Selected Best PWA Frameworks in 2020 - SimiCart, n.d. [Online]. Available from: <https://www.simicart.com/blog/pwa-frameworks/> [Accessed 2021-01-16].
- [74] Spring Boot Tutorial - JournalDev, n.d. [Online]. Available from: <https://www.journaldev.com/7969/spring-boot-tutorial> [Accessed 2021-01-16].
- [75] Stangarone, J., 2010. Native mobile apps : The wrong choice for business ? [Online]. Available from: <https://www.mrc-productivity.com/blog/2016/08/native-mobile-apps-the-wrong-choice-for-business/> [Accessed 2020-10-29].
- [76] Statista, 2020. Biggest app stores in the world 2020 | Statista [Online]. Available from: <https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/> [Accessed 2020-12-28].
- [77] Table of Contents | Cypress Documentation, n.d. [Online]. Available from: <https://docs.cypress.io/api/table-of-contents> [Accessed 2021-08-22].
- [78] The advantages and disadvantages of MySQL - MySQL - Makble, n.d. [Online]. Available from: <http://makble.com/the-advantages-and-disadvantages-of-mysql> [Accessed 2021-01-16].
- [79] The Advantages of a Relational Database Management System, n.d. [Online]. Available from: <https://www.techwalla.com/articles/the-advantages-of-a-relational-database-management-system> [Accessed 2021-01-04].
- [80] The Benefits and Drawbacks of using The MVC Pattern – Kanian’s Blog, n.d. [Online]. Available from: <https://kanian77.wordpress.com/2008/09/03/the-benefits-and-drawbacks-of-using-the-mvc-pattern/> [Accessed 2021-01-08].
- [81] The C4 model for visualising software architecture, n.d. [Online]. Available from: <https://c4model.com/> [Accessed 2021-01-06].
- [82] The Most Popular Progressive Web Apps Frameworks in 2020, n.d. [Online]. Available from: <https://www.mindinventory.com/blog/best-progressive-web-apps-frameworks/> [Accessed 2021-01-16].
- [83] The Pros and Cons of MongoDB - Virtual-DBA Remote DBA Services & Support - Certified Database Experts, n.d. [Online]. Available from: <https://www.virtual-dba.com/pros-and-cons-of-mongodb/> [Accessed 2021-01-16].
- [84] Web App Manifest (Manifesto da Aplicação da Web) | MDN, n.d. [Online]. Available from: <https://developer.mozilla.org/pt-PT/docs/Web/Manifest> [Accessed 2021-01-08].



- [85] What are the advantages and disadvantages of Spring Boot? - Java Interview Questions & Answers, n.d. [Online]. Available from: <https://www.java2novice.com/java-interview-questions/spring-boot-pros-and-cons/> [Accessed 2021-01-16].
- [86] What are the advantages and disadvantages of using MySQL stored procedures?, n.d. [Online]. Available from: <https://www.tutorialspoint.com/What-are-the-advantages-and-disadvantages-of-using-MySQL-stored-procedures> [Accessed 2021-01-16].
- [87] What Is a Relational Database | Oracle, n.d. [Online]. Available from: <https://www.oracle.com/database/what-is-a-relational-database/> [Accessed 2021-01-04].
- [88] What is AGILE? - What is SCRUM? - Agile FAQ's | Cprime, n.d. [Online]. Available from: <https://www.cprime.com/resources/what-is-agile-what-is-scrum/> [Accessed 2021-06-02].
- [89] What is Client-Server? Definition and FAQs | OmniSci, n.d. [Online]. Available from: <https://www.omnisci.com/technical-glossary/client-server> [Accessed 2021-01-05].
- [90] What Is END-TO-END Testing: E2E Testing Framework with Examples, n.d. [Online]. Available from: <https://www.softwaretestinghelp.com/what-is-end-to-end-testing/> [Accessed 2021-08-20].
- [91] What is Entity Relationship Diagram (ERD)?, n.d. [Online]. Available from: <https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/> [Accessed 2021-08-26].
- [92] What is Model-View-Controller(MVC)? Its components, advantages and disadvantages. | Web Designing, Web and Mobile Apps Development Company in Islamabad and Gujrat, Pakistan, n.d. [Online]. Available from: <https://www.webicosoft.com/blog/what-is-model-view-controllermvc-its-components-advantages-and-disadvantages/> [Accessed 2021-01-08].
- [93] What is MoSCoW Prioritization? | Overview of the MoSCoW Method, n.d. [Online]. Available from: <https://www.productplan.com/glossary/moscow-prioritization/> [Accessed 2021-09-06].
- [94] What is non relational database for beginners? - DEV Community, n.d. [Online]. Available from: <https://dev.to/duomly/what-is-non-relational-database-for-beginners-4pg6> [Accessed 2021-01-05].
- [95] What is PostgreSQL? | Features | Advantages and Disadvantages, n.d. [Online]. Available from: <https://www.educba.com/what-is-postgresql/> [Accessed 2021-01-16].
- [96] What is Spring Boot? | Features and Advantages of Spring Boot, n.d. [Online]. Available from: <https://www.educba.com/what-is-spring-boot/> [Accessed 2021-01-16].
- [97] Winer, D., 2020. Android Developers Blog: 11 Weeks of Android: Languages [Online]. Available from: <https://android-developers.googleblog.com/2020/07/11-weeks-of-android-languages.html> [Accessed 2020-11-10].
- [98] Xanthopoulos, S. and Xinogalos, S., 2013. A comparative analysis of cross-platform development approaches for mobile applications. *Acm international conference proceeding series* [Online], (June 2014), pp.213–220. Available from: <https://doi.org/10.1145/2490257.2490292>.

# Appendices

This page is intentionally left blank.

---

# Appendix A

## MB Way

### Main Dashboard and Activity Menu

The application is launched into a main screen with a dashboard and a toolbar in the bottom, as shown in the Figure 1. The dashboard has the associated cards at the top, as well as several buttons right below. These buttons allow the user to do numerous actions, some of them are: send money to someone, use the app to pay for purchases, ask money from someone and split the bill with another person. In the toolbar there's also an option that allows the user to see his activity. That information is displayed as shown in the Figure 2, with the date, the type of action, the amount and the other person involved in the transaction.

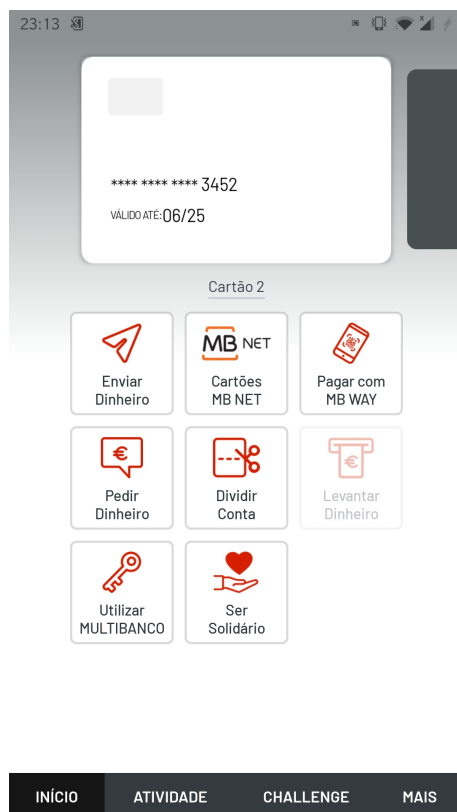


Figure 1: Main dashboard

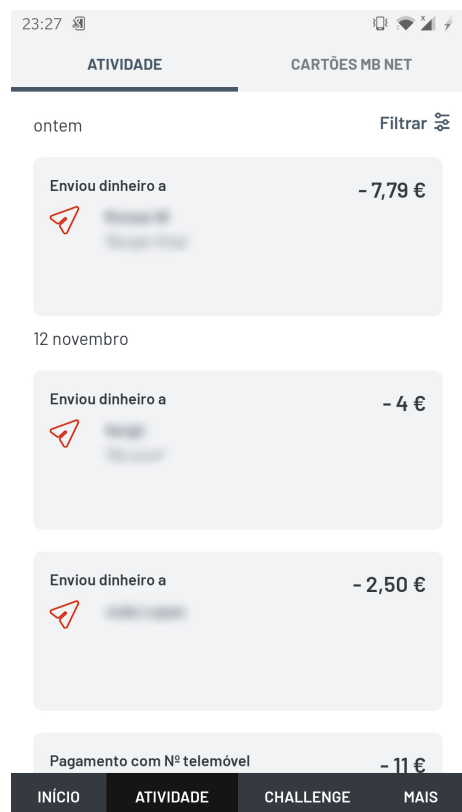


Figure 2: Activity menu

### Send Money

If the user pretends to send money to someone, he can do it by selecting one person from his phone contacts or by manually typing a phone number. After choosing the recipient, the amount has to be set and a description can also be added (optional). Finally, in order to confirm the transaction, the security PIN (Personal Identification Number) has to be inserted.

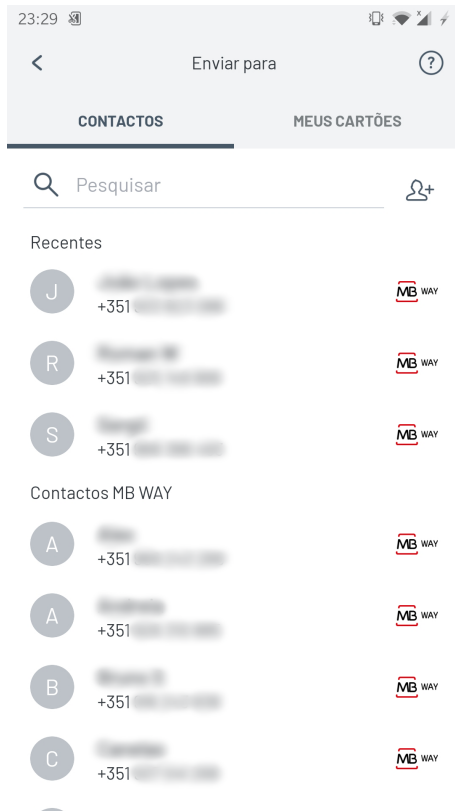


Figure 3: Send money to someone

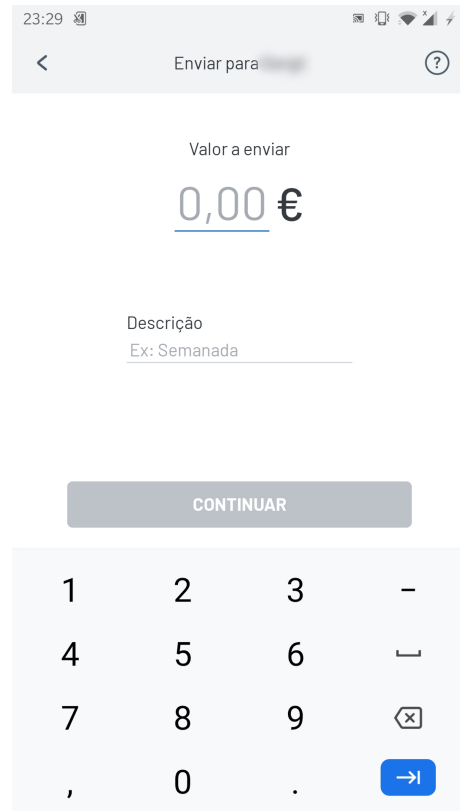


Figure 4: Set amount of money to send



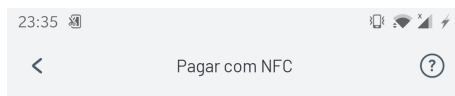
Figure 5: Security PIN

---

## Payment Methods

With MB WAY there's also the possibility to pay for purchases. This can be achieved through three distinct methods, which are:

- Payment with NFC (Figure 6): This method allows the user to pay with NFC (Near Field Communication), meaning that, as long as the user as the NFC sensor active, he can pay by holding his phone close to the vendor's NFC reader.
- Payment with QR Code (Figure 7): Purchases can also be paid with QR Code, which can be done just by giving the application access to the camera, and then reading the QR Code provided by the seller.
- Payment with Number of Barcode (Figure 8): Lastly there's also the option to pay with phone number or barcode, just by providing either one.



  
**NFC desativado**  
Para pagar com NFC é necessário ativar o sensor NFC do seu telemóvel.

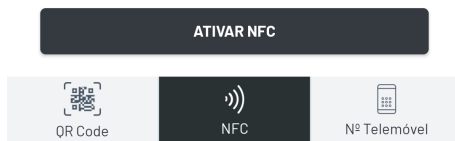
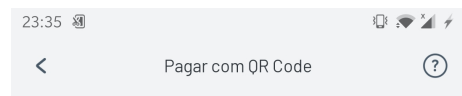


Figure 6: Payment with NFC



  
**Permita o acesso à câmara para ler o QR Code**  
A leitura do QR Code permite pagar com MB WAY.



Figure 7: Payment with QR Code

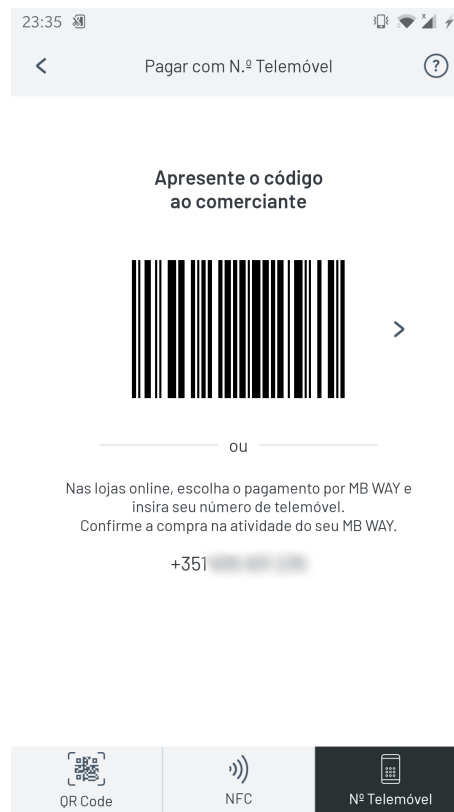


Figure 8: Payment with Number or Barcode

### Ask for Money

An user can also ask for money from someone else (Figure 9), either in his phone contacts or by typing a phone number, after that an amount has to be set, in order to confirm the action. This request will be valid for 7 days. There's also the option to remind the other person about this operation, by going to the activity menu and clicking *Relembrar*, which will send a reminder notification.

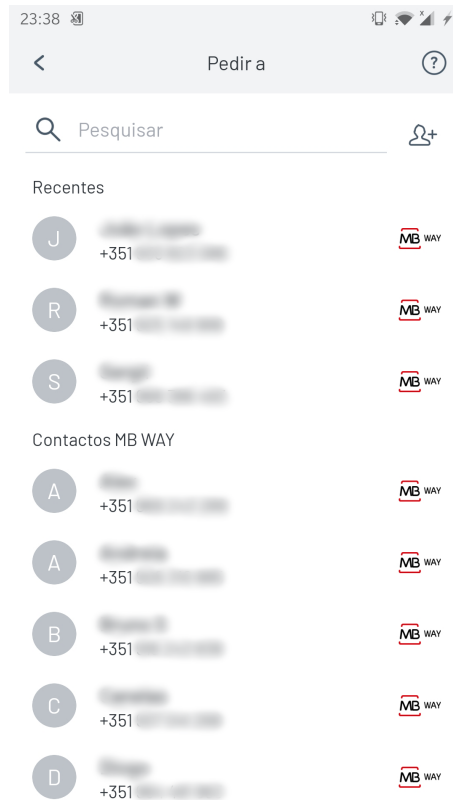


Figure 9: Ask for money from someone

### Split the Bill

Sometimes it's necessary to split the bill with other people (Figure 10, this can be done with MB WAY just by selecting who to split with. After that the total amount needs to be set and each person will receive a notification to pay some amount. This request expires in 7 days.



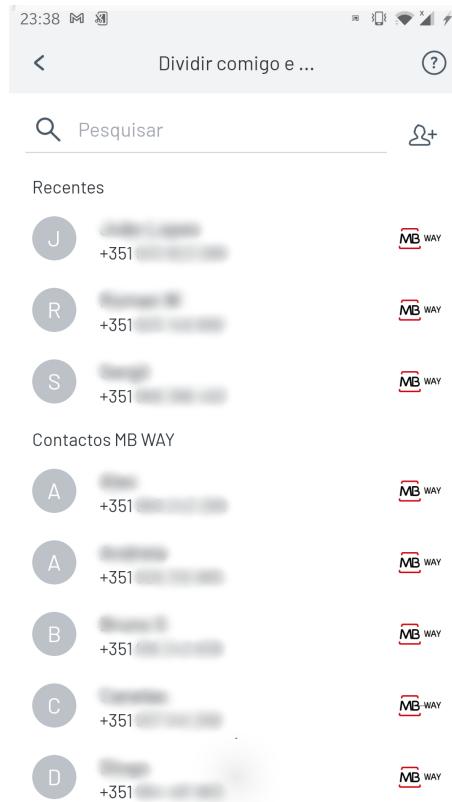


Figure 10: Split the bill with someone

## Information Menu

In the toolbar there's a "More" section as well. Here (Figure 11, the user can either see general information about the application or access the settings menu. Regarding the general information, he can see things, such as: news and discounts, merchants that have MB WAY services, find an ATM (Automated Teller Machine), associated banks, documents, rate the application, privacy policy, terms and conditions, and FAQ (Frequently Asked Questions).



Figure 11: Information menu

## Settings

Then, in the settings menu (Figure 12, the user can change some of the App settings, such as, security settings, where he can set a limit amount to purchases and credit cards (per day), change security PIN, choose type of authentication (for example, use biometric authentication), set MB WAY lock code that prompts every time someone opens the App and payment options, where some options can be set, like paying without PIN in purchases up to 20€ and paying with the phone locked. There's also bank cards settings where they are managed and a "my devices" option, where the user can see every device that logged using his phone number.

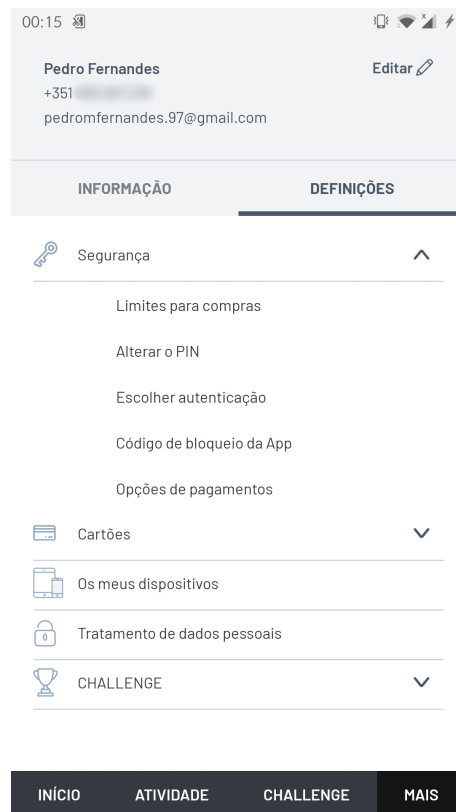


Figure 12: Application settings

## Personal Information

Lastly, MBWAY also allows to change personal info (Figure 13) and to change permission regarding the usage of personal information (Figure 14, mostly to receive e-mails, notifications and SMS with new merchants, offers and promotions).

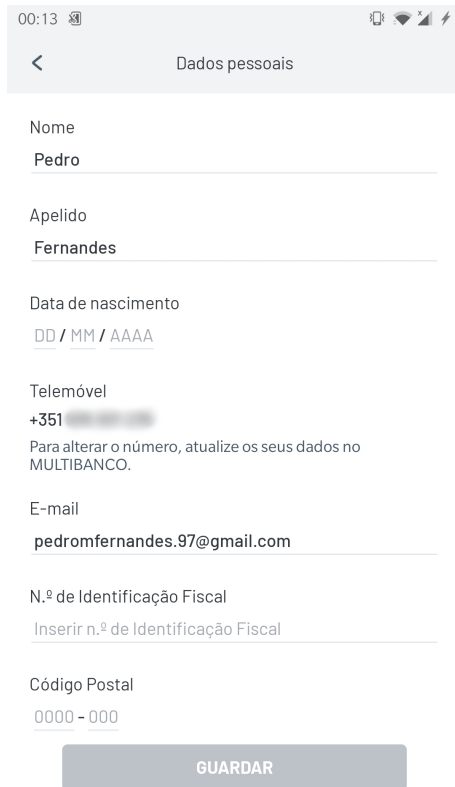


Figure 13: Personal information

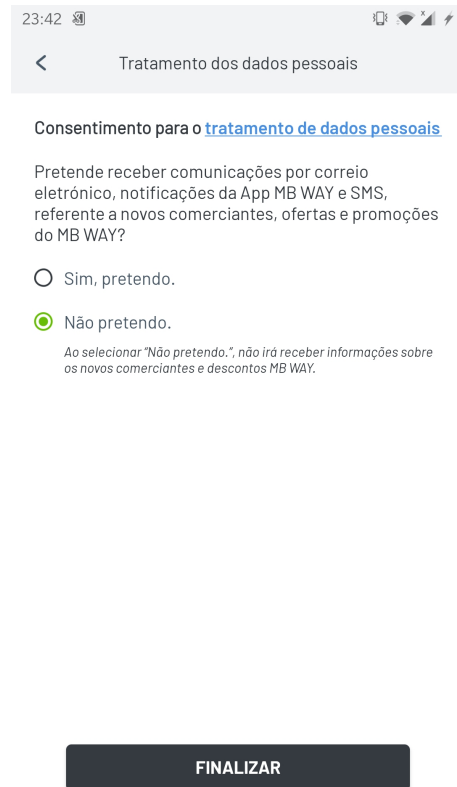


Figure 14: Consent for usage of personal information

## Paypal

### Main Dashboard and Activity Menu

Paypal also launches into a main screen with a dashboard and two toolbars, one at the top and another one at the bottom, as displayed in the Figure 15. The dashboard presents three main things, the account balance, recent activity and a button to view all the account activity (Figure 16). Then, the toolbar at the top has a notifications button (displayed as a bell), a profile button at the very center and a settings button in the far right. Finally, the bottom toolbar has the follow operations: pay with QR Code, send money, request money and a more option with some features as well, like reloading prepaid phones, getting paid with QR codes, sending bank deposits or cash and supporting a charity.

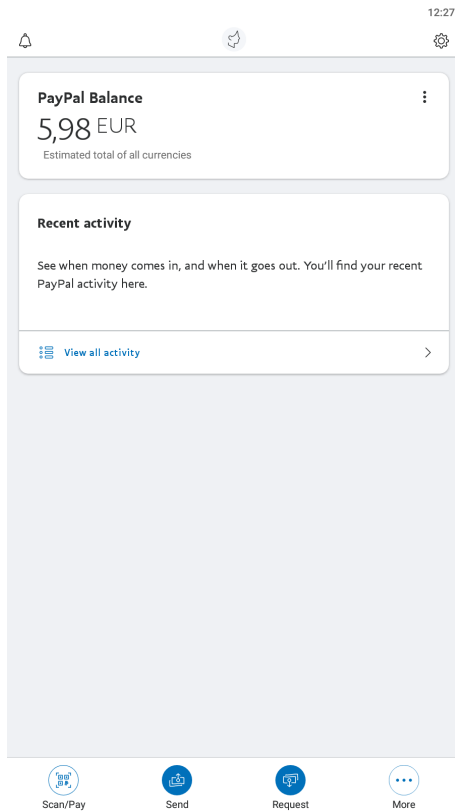


Figure 15: Main dashboard

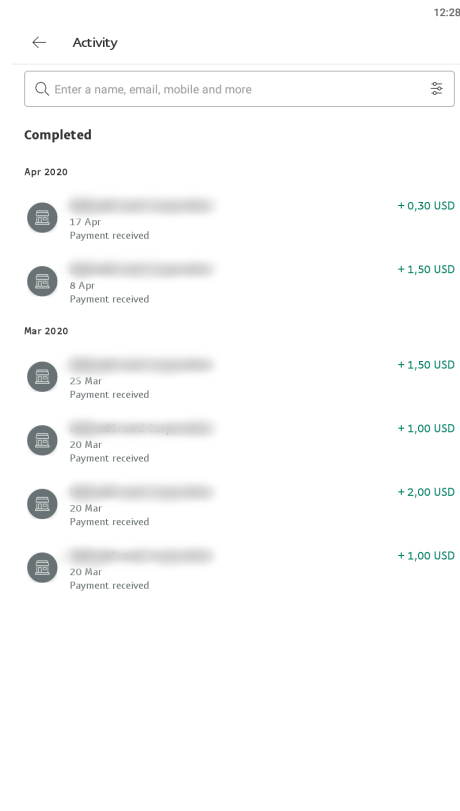


Figure 16: Activity menu

### Send and Request Money

Regarding one of the main goal of this application, that is sending (Figure 17) and requesting (Figure 18) money, it behaves as follows: the user either types a contact name, an username or an e-mail, or scans a QR code, both to send or request money. Furthermore, when requesting money, he can also share his link to get paid and split the bill with other people.

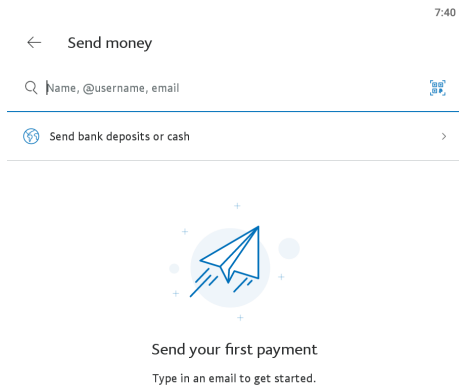


Figure 17: Send money

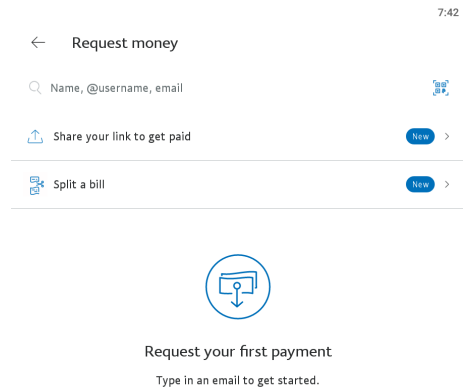


Figure 18: Request money

## Receive Money

An user can share his QR code (Figure 20) so people can scan it and send him money. There's also an option to set an amount of money to receive (Figure 21 and Figure 22).

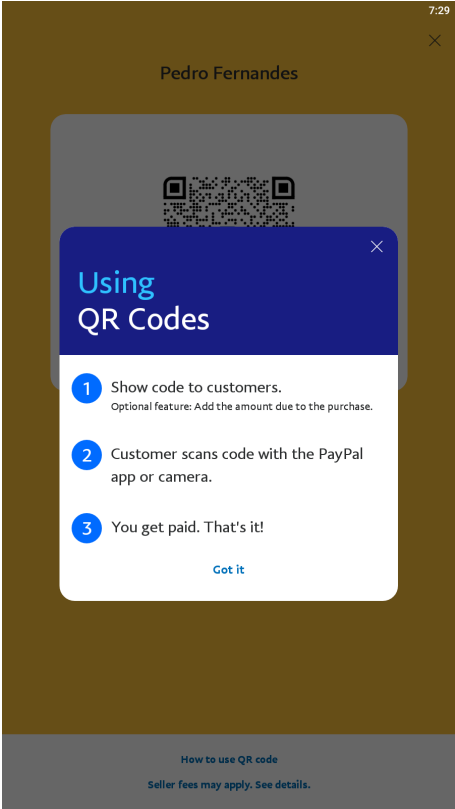


Figure 19: Receive money instructions

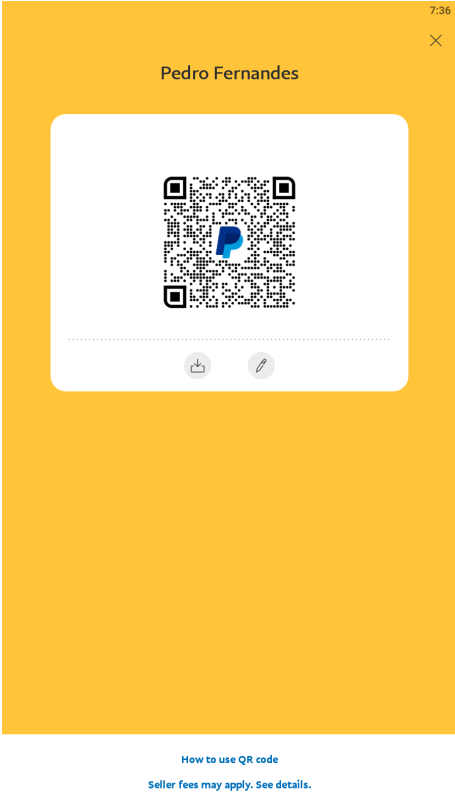


Figure 20: Receive money

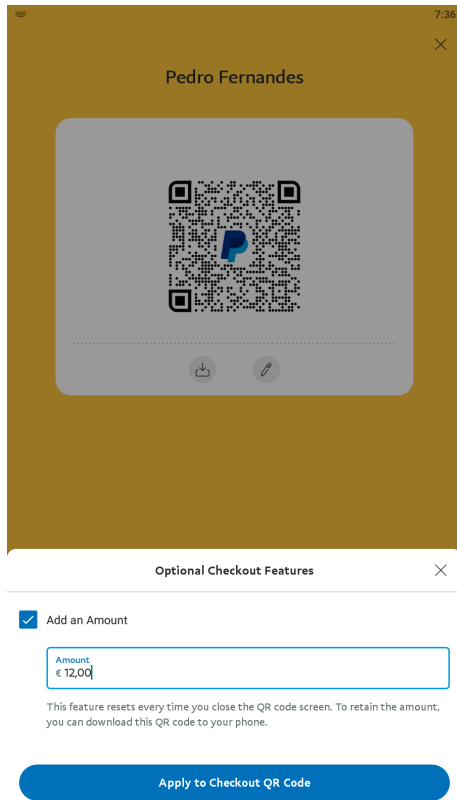


Figure 21: Set amount of money to receive

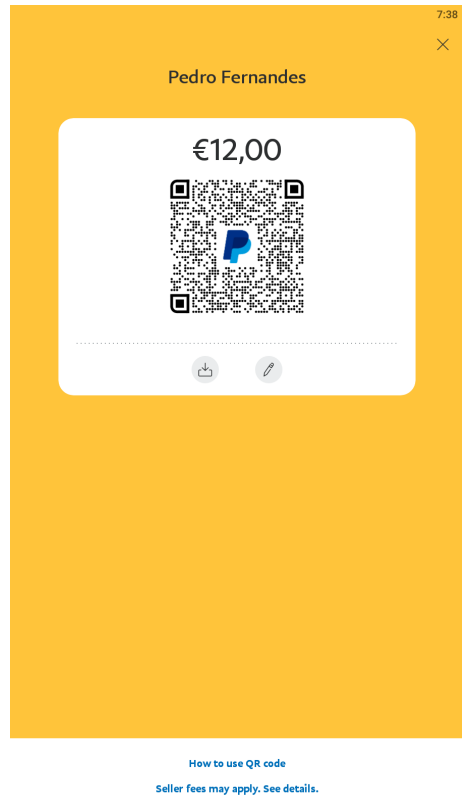


Figure 22: Receive money with amount set

## Settings

Finally, in the settings menu (Figure 23) the user can do operations, such as, linking banks and cards, managing permissions given to other websites (in the "Log in and Security" option) and setting up push notifications.



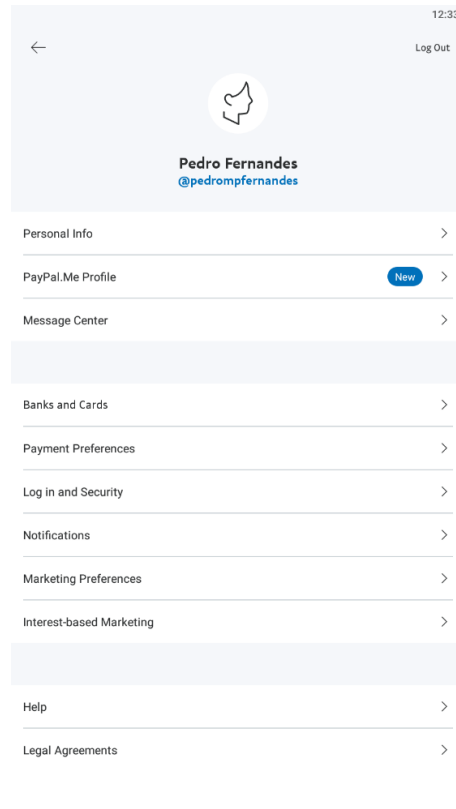


Figure 23: Settings

## Trust Wallet

### Main Dashboard and Notifications Dashboard

The Trust Wallet opens up into a main screen (Figure 24) that has a dashboard and two toolbars (top and bottom). In the dashboard all values can be hidden (Figure 25), also there's the wallet name, the amount of money there, several cryptocurrencies that can be selected with a filter (Figure 26) and three buttons: Send, Receive and Buy. Then the toolbar at the top has a notifications button (Figure 27), a filter button and three sections: Tokens, Finance and Collectibles. Finally, in the bottom toolbar, an user can change to a DApp (Decentralized Application), to a DEX (Decentralized Exchange) menu and to the settings.

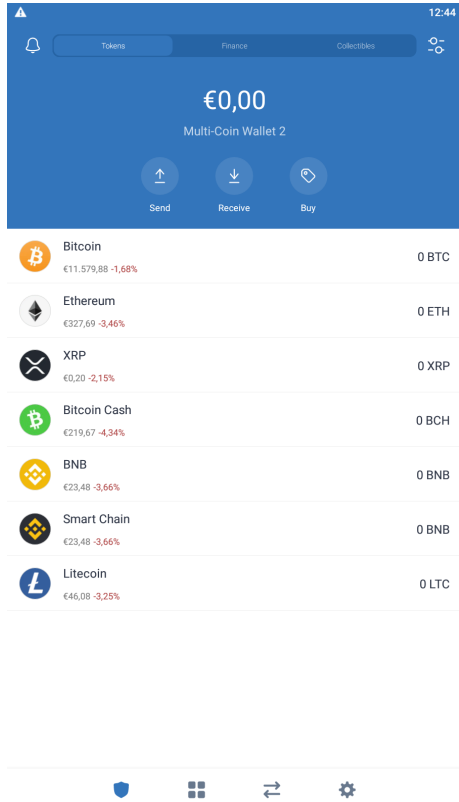


Figure 24: Main dashboard

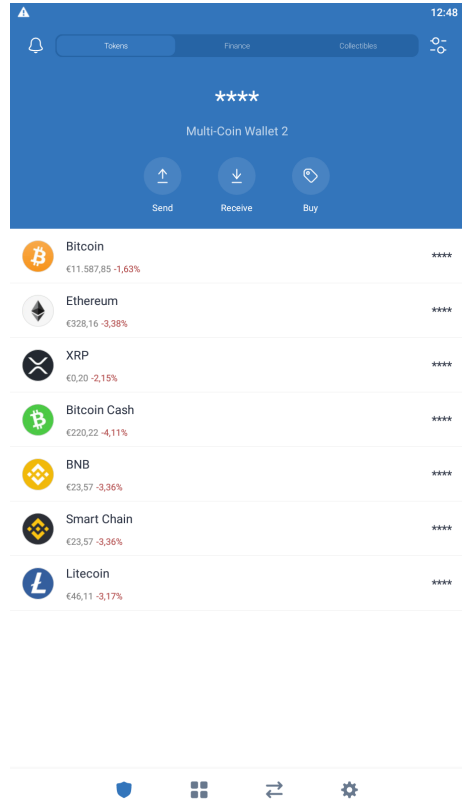


Figure 25: Main dashboard with hidden values

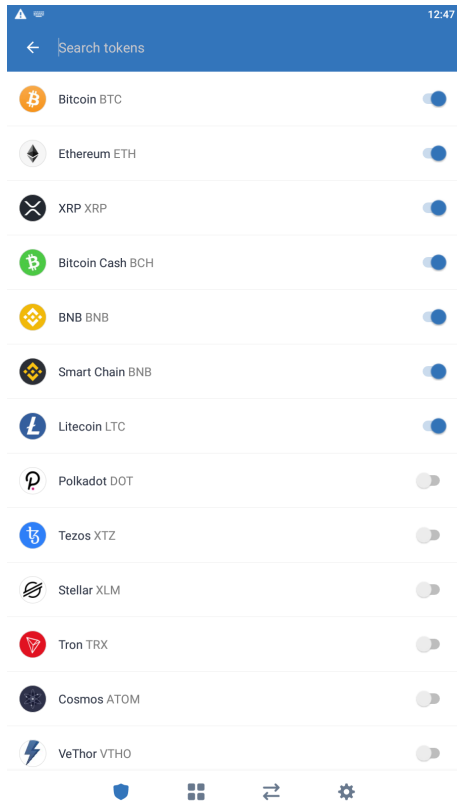


Figure 26: Filter to select dashboard's cryptocurrencies

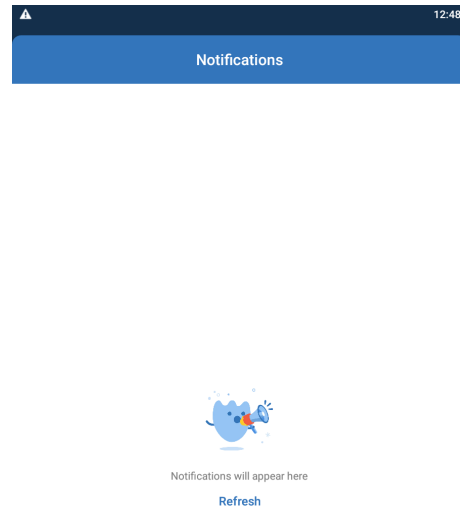


Figure 27: Dashboard with last notifications

## Send Cryptocurrency

As displayed in the Figure 28, sending cryptocurrency to someone can be done, after selecting which one to send, by typing the recipient address or scanning his QR code, and by inserting an amount. Lastly, the user can review the information inserted and the network fee, and confirm the transaction (Figure 29).

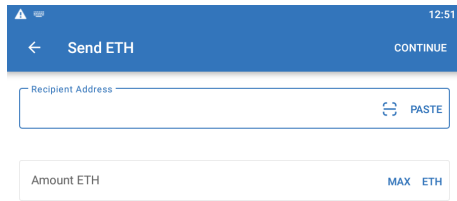


Figure 28: Send cryptocurrency (select recipient)

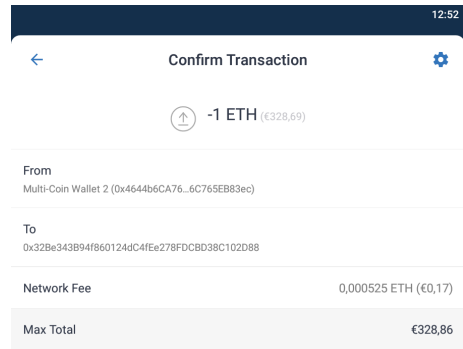


Figure 29: Confirm send transaction

## Receive Cryptocurrency

It's also possible for an user to share his QR code (Figure 30) so people can scan it and send him cryptocurrency. There's also an option to set an amount of money to receive (Figure 31).

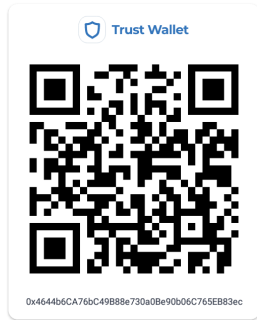


Figure 30: Receive cryptocurrency (through QR Code)



Figure 31: Receive cryptocurrency with amount set

## Buy Cryptocurrency

Buying cryptocurrency is possible too (Figure 32). It can be done by selecting which one to buy and setting an amount, after that it will redirect to a third-party provider to finish the transaction.



€50

~0.146616430013087 ETH


 Mercuryo Third Party Provider	Available	
1	2	3
4	5	6
7	8	9
.	0	✕
NEXT		

Figure 32: Buy set amount of cryptocurrency (through third-party provider)

## Cryptocurrency Dashboard

Each cryptocurrency has its own dashboard as shown in the Figure 33, where the user can send, receive and copy (copies address to receive directly). He can also check that cryptocurrency market information (Figure 34).

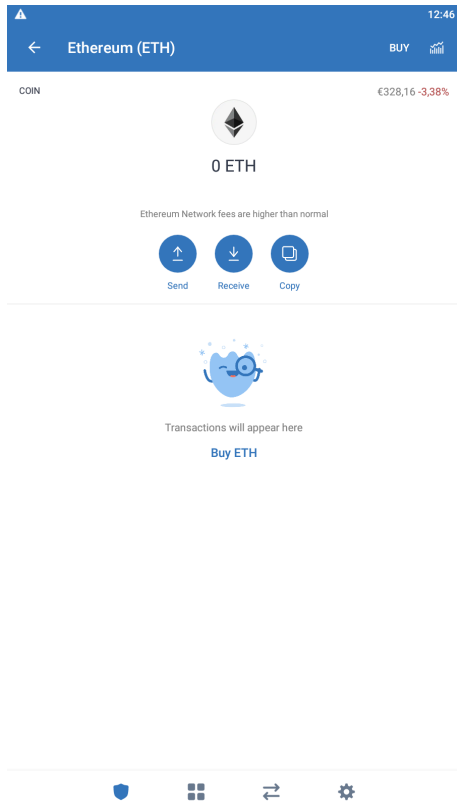


Figure 33: Cryptocurrency dashboard

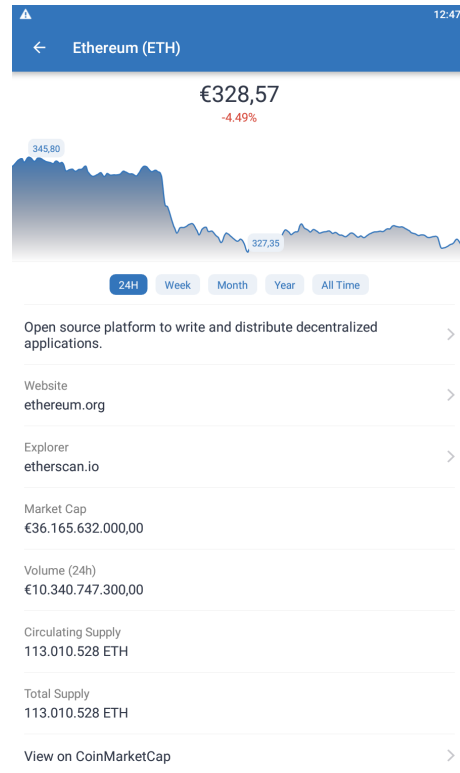


Figure 34: Cryptocurrency market information

## Swap and Exchange Crypto Tokens

In the DEX (Decentralized Exchange) users can swap and exchange tokens. Swapping tokens (Figure 35) is a more simplified way of buying and selling tokens, though it requires a network fee that is given to the miners/validators. After inserting the cryptocurrencies and the amount, a screen is shown with the network fee and to confirm the transaction (Figure 29). Another option is to exchange tokens (Figure 37), this is a more elaborate way of trading where an user can see an order book and can set "Buy" and "Sell" orders. To do that, a trading pair needs to be selected (upper left) which will update the order book, in the right side, for that token (red are sell orders and green are buy orders). Then, by default, it's set to the lowest sell/buy order, but the price per token and the amount can be changed. It's also possible to tap any sell/buy order in the right to automatically indicate the price. Finally a confirm transaction screen appears with the network fee (Figure 38).

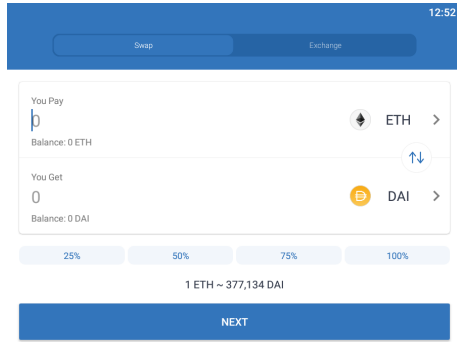


Figure 35: Swap crypto tokens

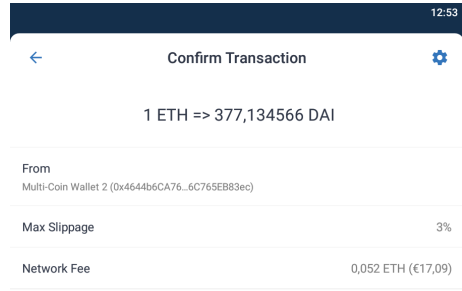


Figure 36: Confirm swap operation

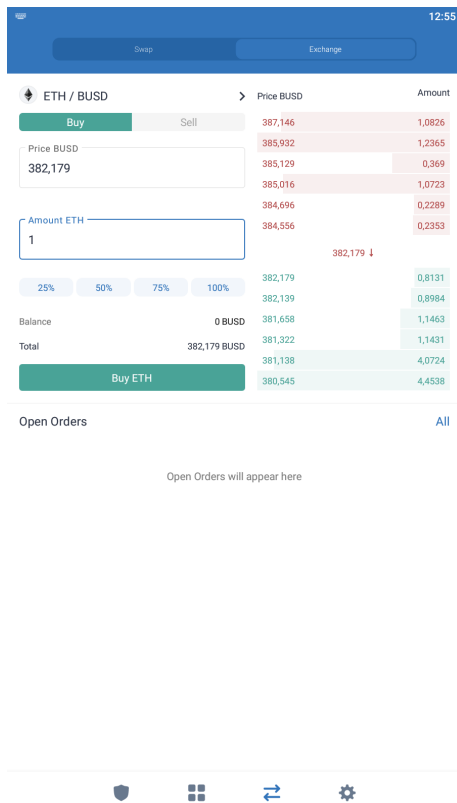


Figure 37: Exchange crypto tokens

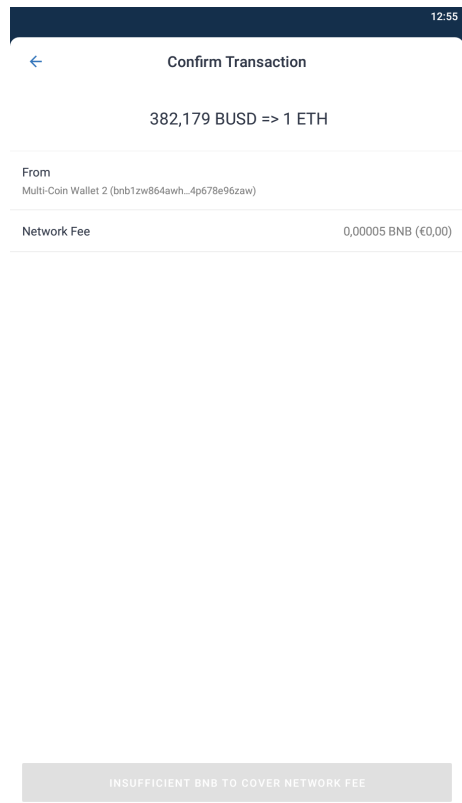


Figure 38: Confirm exchange operation



## Settings

Lastly, in the settings menu (Figure 39) the user can do actions, such as, switching between wallets, enabling dark mode, adding a six digit passcode to protect operations, managing push notifications and changing currency in the preferences. It also has links to the App social media.

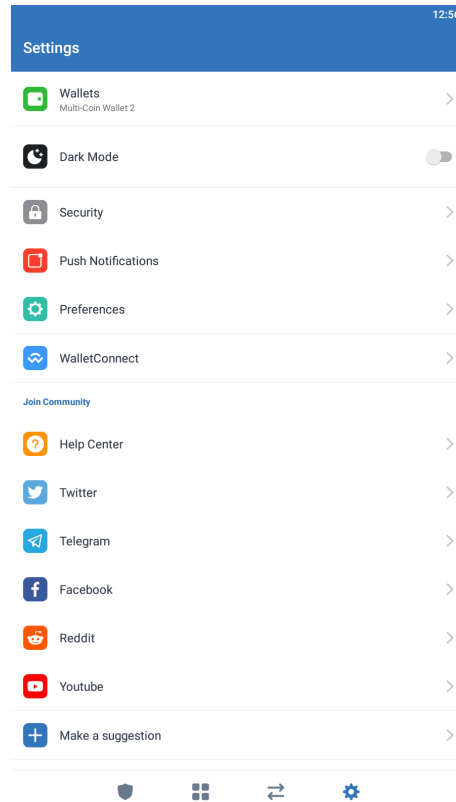


Figure 39: Settings

## FOREX.com

### Account Funds Menu

Besides the net equity value (consists of the total cash plus the profit/loss value) that's always displaying in the toolbar at the top, there's also an account funds menu (Figure 40) that has values such as, profit and loss (positive value means profit and negative value means loss), amount available to trade, total cash, total margin and margin level. It's also possible to withdraw and add funds.

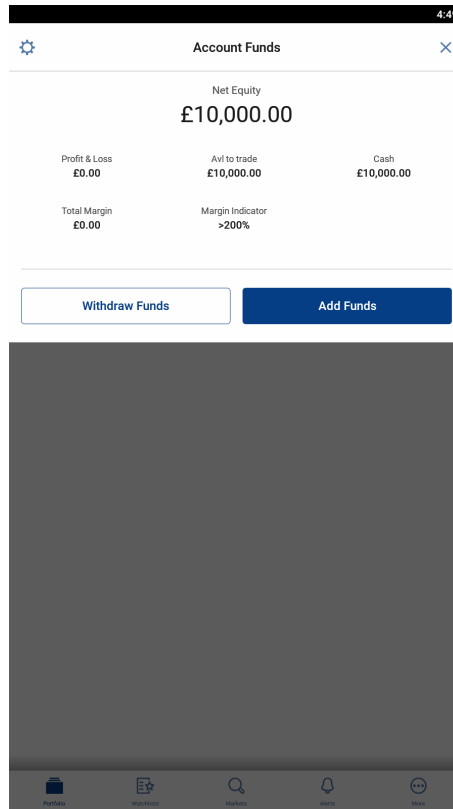


Figure 40: Account funds menu

## Portfolio Menu

In the portfolio menu it's possible to watch every trade position (Figure 41) and order (Figure 42) that's either *Open* or *Closed*.

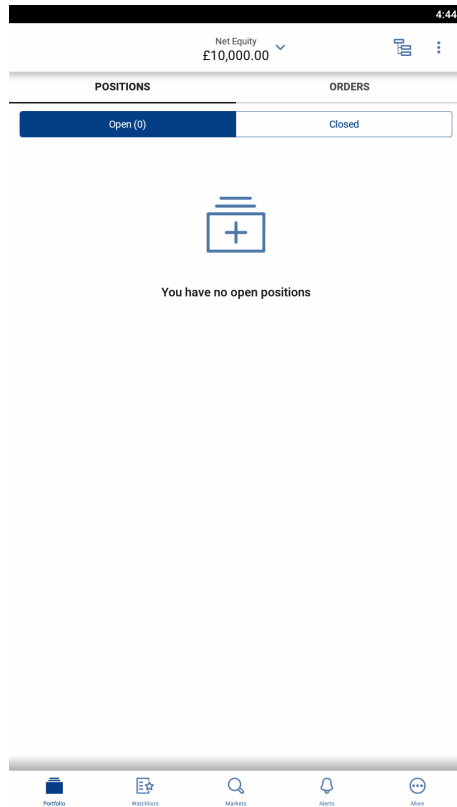


Figure 41: History of trade positions (Open and Closed)

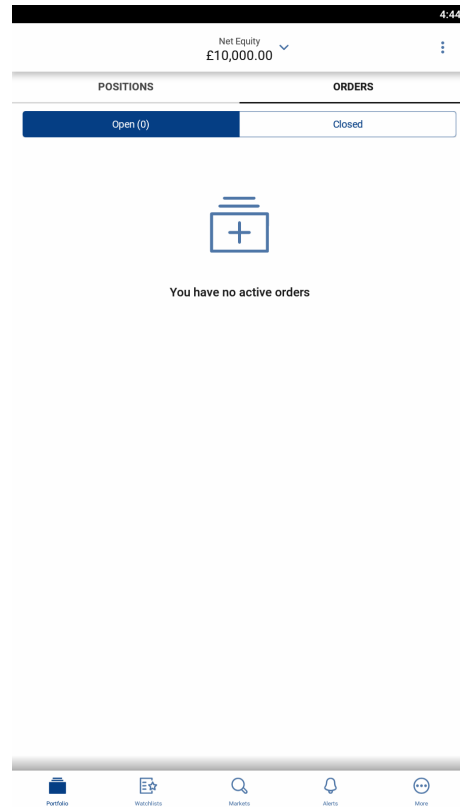


Figure 42: History of trade orders (Open and Closed)

## Watchlists Menu

The watchlists menu (Figure 43) has every watchlist created. Each one has a list of markets selected by the user, that can be searched by category or by name. As shown in the Figure 44, there's a menu with more actions, such as, add markets, edit watchlist, create a new watchlist and manage watchlists.

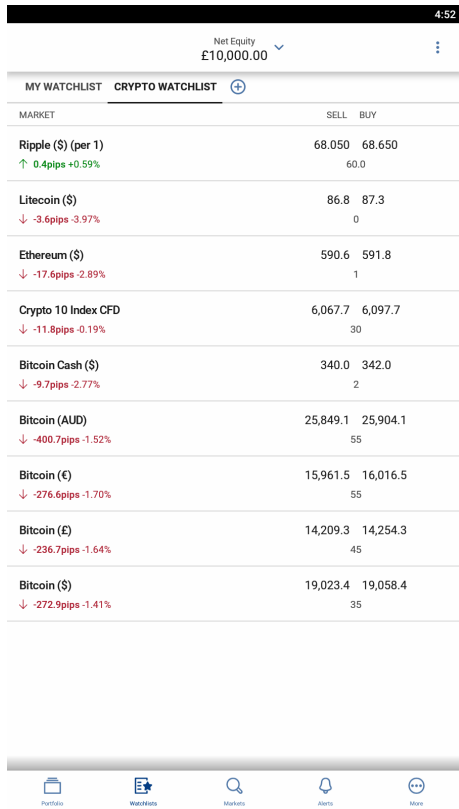


Figure 43: Watchlists menu

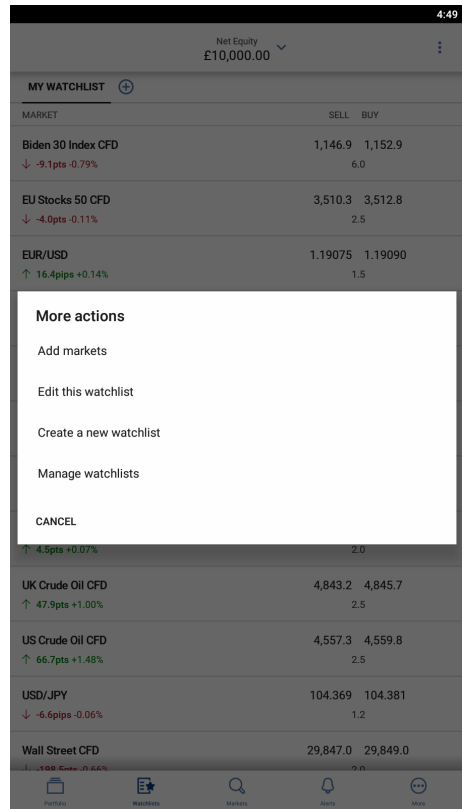


Figure 44: More actions for Watchlists Menu

## Markets Menu and Market Dashboard

As mentioned before, the users can search markets by name or by category (Figure 45). They can also press one of the markets to open its dashboard which displays a chart with the market price changes in a time frame like the one in the Figure 56, that can be set in the upper left corner and that's updating in real time. It also has two buttons at the top with values, that allow the user to buy or sell from that market.

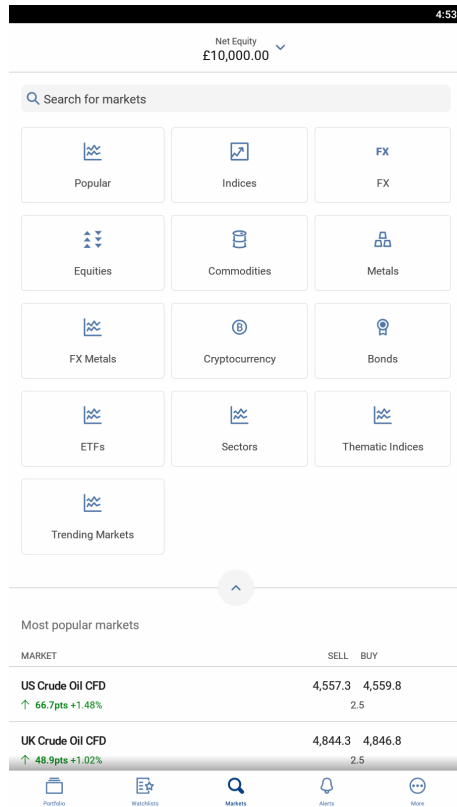


Figure 45: Markets menu



Figure 46: Bitcoin (€) market dashboard

## Alerts Menu

In the alerts menu, the user can set alerts that go off when a market hits a certain price (e.g. for Bitcoin(€) in the Figure 47). When it hits that price set a notification is pushed to the user as shown in the Figure 48.



Figure 47: Menu to set alerts of price changes

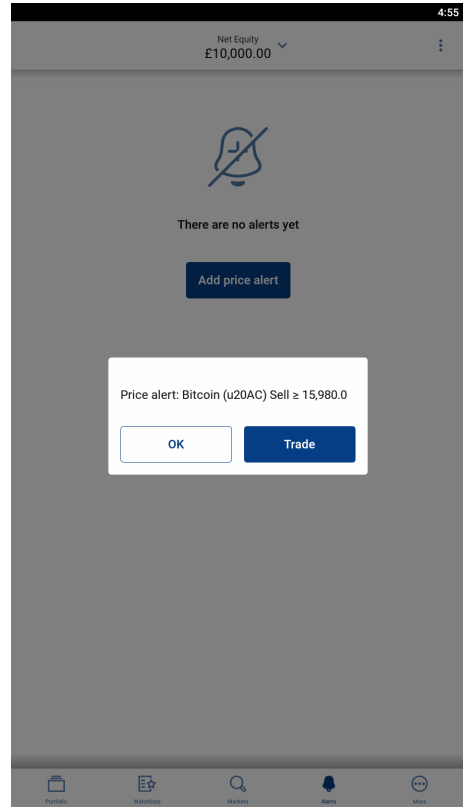


Figure 48: Notification for alerts set of price changes

## Market Trade Menu

With this application the user can also do two operations, trade and order. When trading (Figure 49) represents a trade buy, but it's the same for sell, an amount has to be inserted (minimum 1), then there's three options that can also be set: the take profit (closes a position when it reaches a pre-defined price), the stop loss (which can be regular meaning that closes a position if it drop to a pre-defined price to limit losses or trailing which will adjust automatically based on the market price) and the hedging (strategy that opens additional positions in order to protect adverse movements in the market). When ordering (Figure 50 where sell and buy are displayed equally, only with different colors), the process is the same as to trading, but in this case, the wished price to sell/order has to be set, as well as the option *Good until* to *Cancelled* (will be opened until the user chooses otherwise) or *Time* (will be opened until a certain date set).

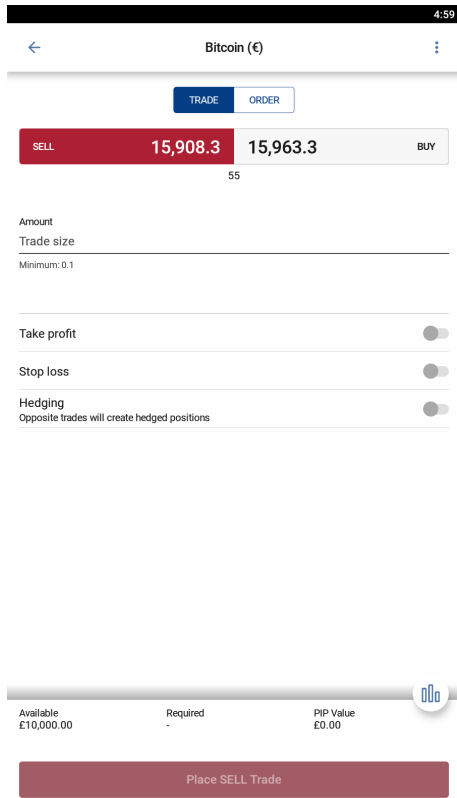


Figure 49: Menu to make a trade in the Market (sell)

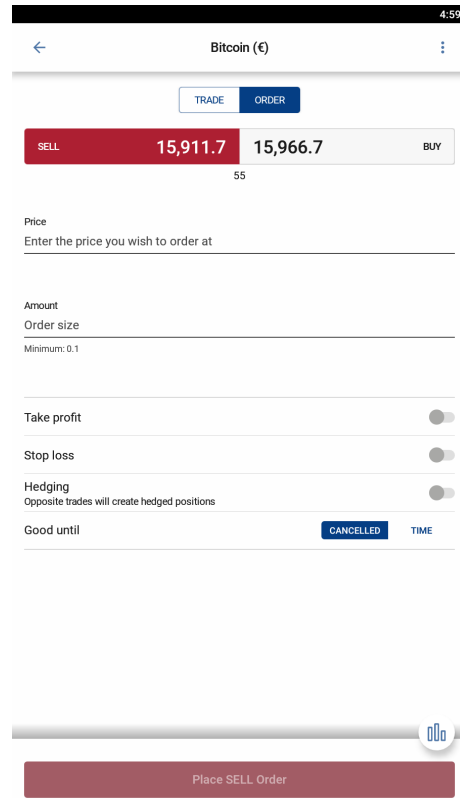


Figure 50: Menu to make an order in the Market (sell)

## Kraken

### Markets Menu

As shown in the Figure 51, all markets selected by the user are displayed in the *Markets Menu*. They can also be clicked and a drop-down list of currency pairs will appear (Figure 52). In the top right corner there's a button that allows to filter this list as seen in the Figure 53. There can be set options, such as, time period (for the market performance displayed), way of sorting (through volume or alphabetical) and a quote currency filter (displays only fiat, only crypto or both).

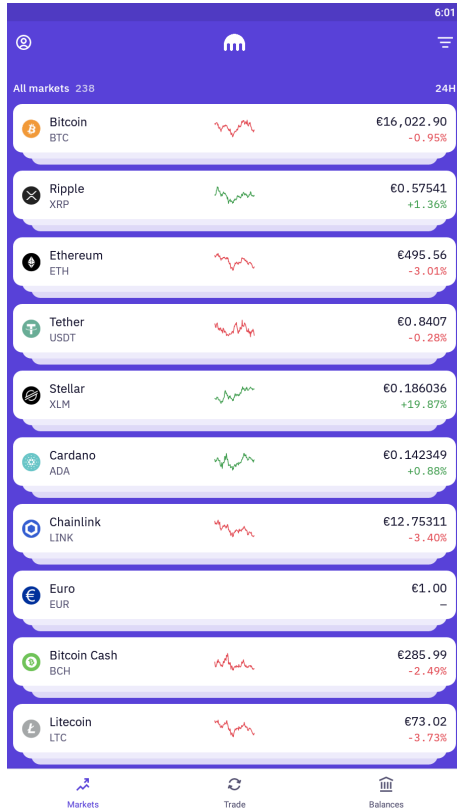


Figure 51: Markets menu

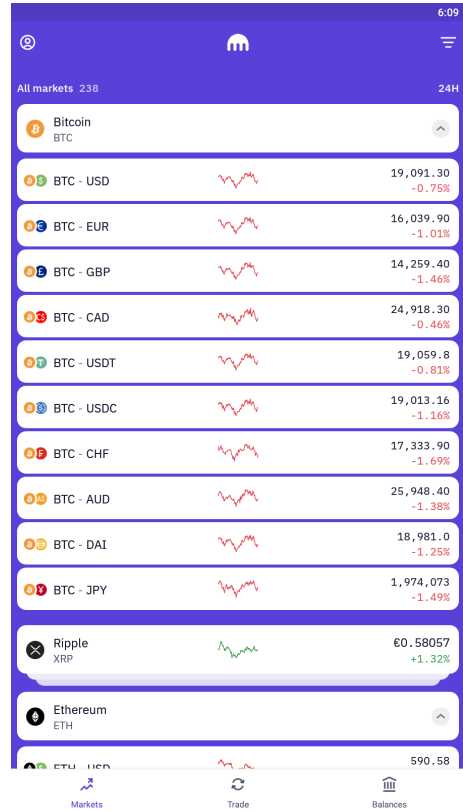


Figure 52: Markets menu with one selected

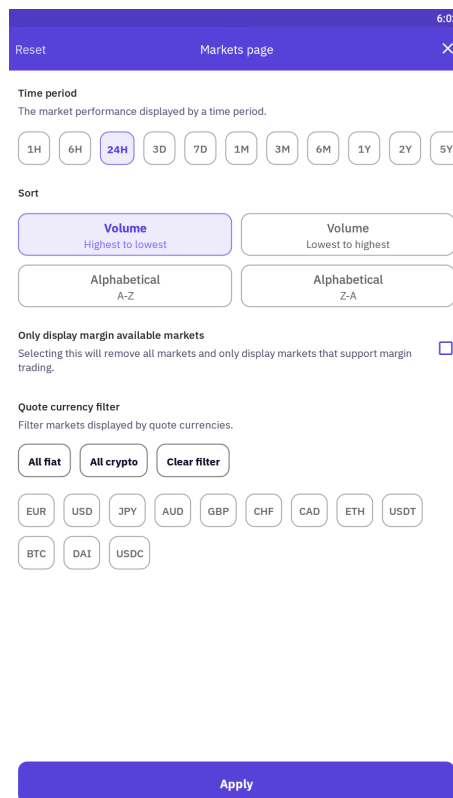


Figure 53: Filter of markets menu



## Cryptocurrency Trading Menu and Dashboard

The *Trade Menu* consists of a board (Figure 54) that shows every order, trade and position that's either open or closed. It also has a *New order* button at the bottom right corner that prompts a new form like the one in the Figure 55). This form allows the user to buy/sell cryptocurrency, by inserting the amount and the limit price. Other options can also be set, such as, leverage (none, 2x, 3x, 4x and 5x), limit (market, stop loss and take profit), start date (now or custom), expiration date (until cancelled or custom), post limit order (prevents order taker fees and ensures the maker fee if it's executed), fee currency (preferred currency to apply fees) and conditional close (none, limit, stop loss, take profit).

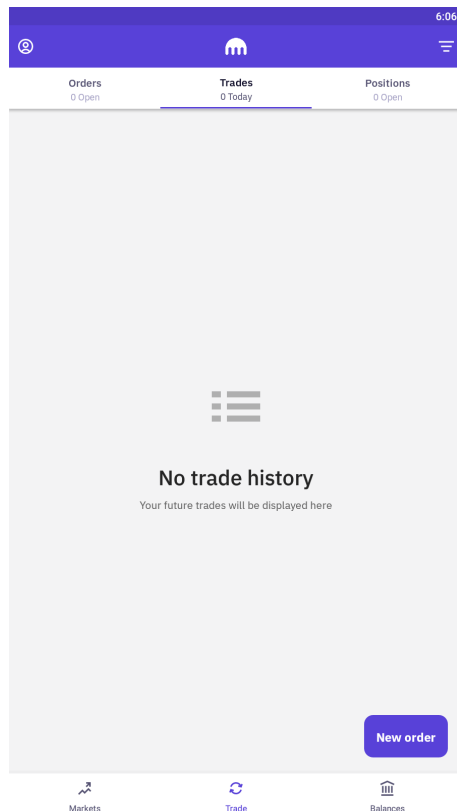


Figure 54: Menu with every order, trade and position done

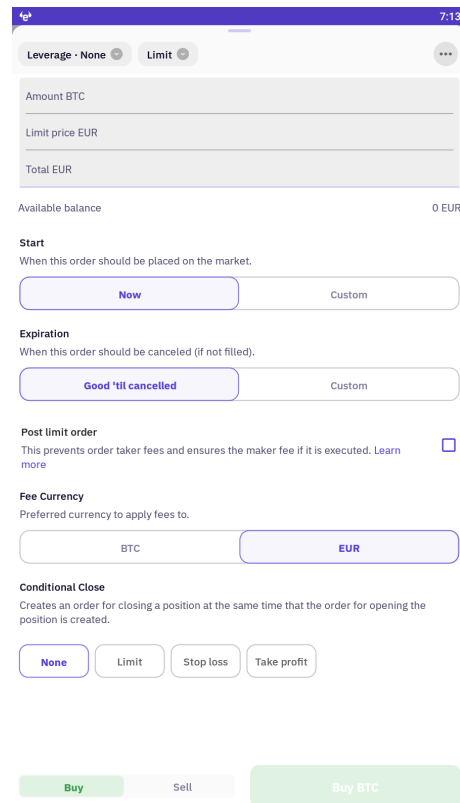


Figure 55: Menu to make a cryptocurrency trade in the Market (buy)

The user can also watch a currency pair dashboard (Figure 56) by clicking one in the markets menu. In this dashboard there's the price changes in the last 24 hours and 1 week (highest and lowest value reached), market changes chart (1 minute, 5 minutes, 15 minutes, 1 hour, 4 hours, 1 day and 1 week), order book, recent trades and the user history of orders and positions of that pair. Finally, it's also possible to switch the currency pair by clicking the one being displayed in the top left corner and switch for a new one like in the Figure 57).

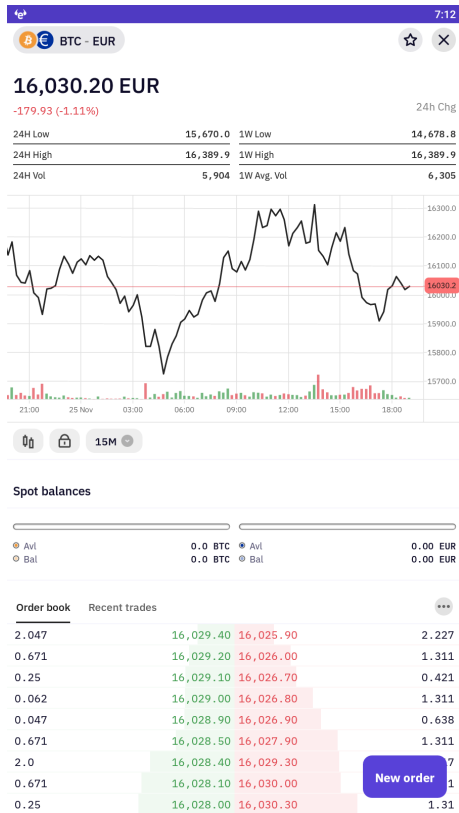


Figure 56: Pair Bitcoin/€ market dashboard

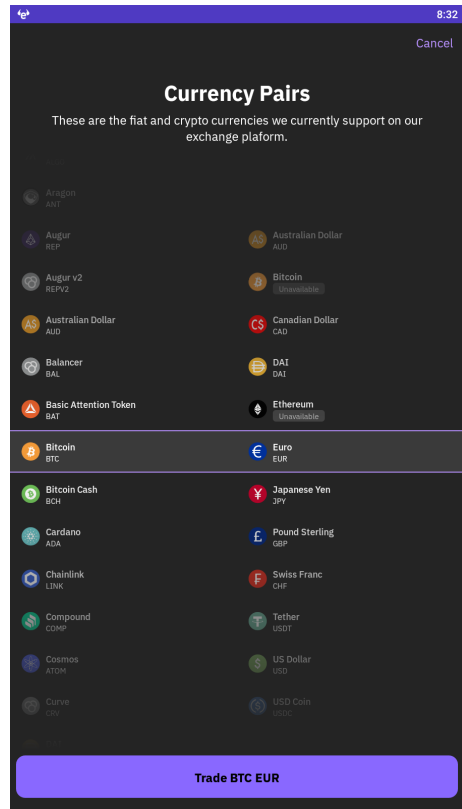


Figure 57: Menu to select the currency pair

This page is intentionally left blank.

## Appendix B

### Authentication Microservice Routes Unit Testing

Login Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Login_01	POST request: It should not login user and return a bad request error (invalid phone number).	Bad request error (code 400) with message: Phone Number must be valid.	Bad request error (code 400) with message: Phone Number must be valid.	Pass
UT_Login_02	POST request: It should not login user and return a bad request error (password not supplied).	Bad request error (code 400) with message: You must supply a password.	Bad request error (code 400) with message: You must supply a password.	Pass
UT_Login_03	POST request: It should not login user and return a bad request error (invalid credentials because phone number does not exist).	Bad request error (code 400) with message: Invalid credentials.	Bad request error (code 400) with message: Invalid credentials.	Pass
UT_Login_04	POST request: It should not login user and return a bad request error (invalid credentials because password is not correct).	Bad request error (code 400) with message: Invalid credentials.	Bad request error (code 400) with message: Invalid credentials.	Pass
UT_Login_05	POST request: it should login user and return a session.	A response (code 200) with a session (user ID and JWT token).	A response (code 200) with a session (user ID and JWT token).	Pass

Figure 58: Results for the login route unit testing

Logout Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Logout_01	GET request: It should not logout user and return not authorized error (cookie with session jwt token does not exist).	Not authorized error.	Not authorized error.	Pass
UT_Logout_02	GET request: It should logout user.	Response with code 200.	Response with code 200.	Pass

Figure 59: Results for the logout route unit testing

Password Recovery Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Password_Recovery_01	POST request: It should not initiate the password recovery process and return a request validation error (phone number and email missing).	Request validation error (code 400) with message: You must supply at least one field (phone number or email).	Request validation error (code 400) with message: You must supply at least one field (phone number or email).	Pass
UT_Password_Recovery_02	POST request: It should not initiate the password recovery process and return a bad request error (invalid phone number).	Bad request error (code 400) with message: Phone Number must be valid.	Bad request error (code 400) with message: Phone Number must be valid.	Pass
UT_Password_Recovery_03	POST request: It should not initiate the password recovery process and return a bad request error (invalid email).	Bad request error (code 400) with message: You must supply a valid email.	Bad request error (code 400) with message: You must supply a valid email.	Pass
UT_Password_Recovery_04	POST request: It should not initiate the password recovery process and return a bad request error (phone number not found).	Bad request error (code 400) with message: Phone number not found.	Bad request error (code 400) with message: Phone number not found.	Pass
UT_Password_Recovery_05	POST request: It should not initiate the password recovery process and return a bad request error (email not found).	Bad request error (code 400) with message: Email not found.	Bad request error (code 400) with message: Email not found.	Pass
UT_Password_Recovery_06	POST request: It should initiate the password recovery process and return a success message along with the user ID.	Response with code 200, the message "Email sent successfully" and the user ID.	Response with code 200, the message "Email sent successfully" and the user ID.	Pass

Figure 60: Results for the password recovery route unit testing

Register Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Register_01	POST request: It should not register user and return a bad request error (invite code must be numerical).	Bad request error (code 400) with message: Invitation code must be numerical.	Bad request error (code 400) with message: Invitation code must be numerical.	Pass
UT_Register_02	POST request: It should not register user and return a bad request error (invite code must have 6 numbers).	Bad request error (code 400) with message: Invitation field must have 6 numbers.	Bad request error (code 400) with message: Invitation field must have 6 numbers.	Pass
UT_Register_03	POST request: It should not initiate the password recovery process and return a bad request error (invalid phone number).	Bad request error (code 400) with message: You must enter a valid phone number from Portugal.	Bad request error (code 400) with message: You must enter a valid phone number from Portugal.	Pass
UT_Register_04	POST request: It should not initiate the password recovery process and return a bad request error (invalid email).	Bad request error (code 400) with message: You must enter a valid email address.	Bad request error (code 400) with message: You must enter a valid email address.	Pass
UT_Register_05	POST request: It should not initiate the password recovery process and return a bad request error (invalid full name).	Bad request error (code 400) with message: You must enter a valid full name with only letters and at least 5 characters.	Bad request error (code 400) with message: You must enter a valid full name with only letters and at least 5 characters.	Pass
UT_Register_06	POST request: It should not initiate the password recovery process and return a bad request error (full name has not enough characters).	Bad request error (code 400) with message: You must enter a valid full name with only letters and at least 5 characters.	Bad request error (code 400) with message: You must enter a valid full name with only letters and at least 5 characters.	Pass
UT_Register_07	POST request: It should not initiate the password recovery process and return a bad request error (invalid password).	Bad request error (code 400) with message: Invalid password.	Bad request error (code 400) with message: Invalid password.	Pass
UT_Register_08	POST request: It should not initiate the password recovery process and return a bad request error (password characters limit exceeded).	Bad request error (code 400) with message: You must enter a valid password (length between 8 and 20, at least 1 lowercase and 1 uppercase letter, 1 number and 1 symbol).	Bad request error (code 400) with message: Invalid password.	Pass
UT_Register_09	POST request: It should not initiate the password recovery process and return a bad request error (pin code must be numerical).	Bad request error (code 400) with message: Pin field must be numerical.	Bad request error (code 400) with message: Pin field must be numerical.	Pass
UT_Register_10	POST request: It should not initiate the password recovery process and return a bad request error (pin code must have 6 digits).	Bad request error (code 400) with message: Pin field must be numerical.	Bad request error (code 400) with message: Pin field must be numerical.	Pass
UT_Register_11	POST request: It should not initiate the password recovery process and return a bad request error (phone number already in use).	Bad request error (code 400) with message: Email or phone number in use.	Bad request error (code 400) with message: Email or phone number in use.	Pass
UT_Register_12	POST request: It should not initiate the password recovery process and return a bad request error (email already in use).	Bad request error (code 400) with message: Email or phone number in use.	Bad request error (code 400) with message: Email or phone number in use.	Pass
UT_Register_13	POST request: It should not initiate the password recovery process and return a bad request error (no invite associated).	Bad request error (code 400) with message: This email has no invite associated.	Bad request error (code 400) with message: This email has no invite associated.	Pass
UT_Register_14	POST request: It should not initiate the password recovery process and return a bad request error (incorrect invitation code).	Bad request error (code 400) with message: Invalid invitation code inserted.	Bad request error (code 400) with message: Invalid invitation code inserted.	Pass
UT_Register_15	POST request: It should register user and return a success message with the user ID.	Response with code 201 and a success message with the user ID.	Response with code 201 and a success message with the user ID.	Pass

Figure 61: Results for the register route unit testing

## Cryptocurrency Microservice Routes Unit Testing

Add Cryptocurrency Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Add_Crypto_01	POST request: It should not add a crypto and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Add_Crypto_02	POST request: It should not add a crypto and return a not authorized error because the user does not have administrator privileges.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Add_Crypto_03	POST request: It should not add a crypto and return a bad request error (invalid name).	Bad request error (code 400) with message: You must enter a valid alphanumerical name.	Bad request error (code 400) with message: You must enter a valid alphanumerical name.	Pass
UT_Add_Crypto_04	POST request: It should not add a crypto and return a bad request error (number of characters wrong).	Bad request error (code 400) with message: You must enter a name with a minimum of 3 characters and a maximum of 15.	Bad request error (code 400) with message: You must enter a name with a minimum of 3 characters and a maximum of 15.	Pass
UT_Add_Crypto_05	POST request: It should not add a crypto and return a bad request error (invalid symbol).	Bad request error (code 400) with message: You must enter a valid alphanumerical symbol.	Bad request error (code 400) with message: You must enter a valid alphanumerical symbol.	Pass
UT_Add_Crypto_06	POST request: It should not add a crypto and return a bad request error (number of characters wrong).	Bad request error (code 400) with message: You must enter a symbol with a minimum of 2 characters and a maximum of 6.	Bad request error (code 400) with message: You must enter a symbol with a minimum of 2 characters and a maximum of 6.	Pass
UT_Add_Crypto_07	POST request: It should not add a crypto and return a bad request error (amount must be a decimal number).	Bad request error (code 400) with message: Amount must be valid decimal.	Bad request error (code 400) with message: Amount must be valid decimal.	Pass
UT_Add_Crypto_08	POST request: It should not add a crypto and return a bad request error (invalid volume value).	Bad request error (code 400) with message: You must enter a valid volume (positive integer).	Bad request error (code 400) with message: You must enter a valid volume (positive integer).	Pass
UT_Add_Crypto_09	POST request: It should not add a crypto and return a bad request error (invalid privacy value).	Bad request error (code 400) with message: Privacy must be private, semi-private or public.	Bad request error (code 400) with message: Privacy must be private, semi-private or public.	Pass
UT_Add_Crypto_10	POST request: It should not add a crypto and return a bad request error (invalid transaction cost value).	Bad request error (code 400) with message: Transaction cost must be a integer between 0 and 100 (percentage).	Bad request error (code 400) with message: Transaction cost must be a integer between 0 and 100 (percentage).	Pass
UT_Add_Crypto_11	POST request: It should not add a crypto and return a bad request error (invalid settlement time value).	Bad request error (code 400) with message: Settlement time must be a valid unix timestamp.	Bad request error (code 400) with message: Settlement time must be a valid unix timestamp.	Pass
UT_Add_Crypto_12	POST request: It should not add a crypto and return a bad request error (cryptocurrency already in use).	Bad request error (code 400) with message: Crypto name already in use.	Bad request error (code 400) with message: Crypto name already in use.	Pass
UT_Add_Crypto_13	POST request: It should add crypto and return a success message with the crypto ID.	Response with code 201 and a success message with the crypto ID.	Response with code 201 and a success message with the crypto ID.	Pass

Figure 62: Results for the add cryptocurrency route unit testing

Edit Cryptocurrency Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Edit_Crypto_01	POST request: It should not edit crypto and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Edit_Crypto_02	POST request: It should not edit crypto and return a not authorized error because the user does not have administrator privileges.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Edit_Crypto_03	POST request: It should not edit crypto and return a bad request error (invalid name).	Bad request error (code 400) with message: You must enter a valid alphanumerical name.	Bad request error (code 400) with message: You must enter a valid alphanumerical name.	Pass
UT_Edit_Crypto_04	POST request: It should not edit crypto and return a bad request error (number of characters wrong).	Bad request error (code 400) with message: You must enter a name with a minimum of 3 characters and a maximum of 15.	Bad request error (code 400) with message: You must enter a name with a minimum of 3 characters and a maximum of 15.	Pass
UT_Edit_Crypto_05	POST request: It should not edit crypto and return a bad request error (amount must be a decimal number).	Bad request error (code 400) with message: Amount must be valid decimal.	Bad request error (code 400) with message: Amount must be valid decimal.	Pass
UT_Edit_Crypto_06	POST request: It should not edit crypto and return a bad request error (invalid volume value).	Bad request error (code 400) with message: You must enter a valid volume (positive integer).	Bad request error (code 400) with message: You must enter a valid volume (positive integer).	Pass
UT_Edit_Crypto_07	POST request: It should not add a crypto and return a bad request error (invalid privacy value).	Bad request error (code 400) with message: Privacy must be private, semi-private or public.	Bad request error (code 400) with message: Privacy must be private, semi-private or public.	Pass
UT_Edit_Crypto_08	POST request: It should not edit crypto and return a bad request error (invalid transaction cost value).	Bad request error (code 400) with message: Transaction cost must be a integer between 0 and 100 (percentage).	Bad request error (code 400) with message: Transaction cost must be a integer between 0 and 100 (percentage).	Pass
UT_Edit_Crypto_09	POST request: It should not edit crypto and return a bad request error (invalid settlement time value).	Bad request error (code 400) with message: Settlement time must be a valid unix timestamp.	Bad request error (code 400) with message: Settlement time must be a valid unix timestamp.	Pass
UT_Edit_Crypto_10	POST request: It should not edit crypto and return a bad request error (cryptocurrency does not exist).	Bad request error (code 400) with message: Crypto does not exist.	Bad request error (code 400) with message: Crypto does not exist.	Pass
UT_Edit_Crypto_11	POST request: It should edit crypto and return a success message.	Response with code 201 and a success message.	Response with code 201 and a success message.	Pass

Figure 63: Results for the edit cryptocurrency route unit testing

Get All Cryptocurrencies Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_All_Cryptos_01	GET request: It should not get all cryptos and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_All_Cryptos_02	GET request: It should not get all cryptos and return a not authorized error because the user does not have administrator privileges.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_All_Cryptos_03	GET request: It should get all the cryptos.	Response with code 200 and an array with all the cryptos.	Response with code 200 and an array with all the cryptos.	Pass

Figure 64: Results for the get all cryptocurrencies route unit testing

## Logs Microservice Routes Unit Testing

Get Error Logs Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Error_Logs_01	GET request: It should not return the error logs and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Error_Logs_02	GET request: It should not return the error logs and return a not authorized error because the user does not have administrator privileges.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Error_Logs_03	GET request: It should return the error logs.	Response with code 200 and an array with the error logs.	Response with code 200 and an array with the error logs.	Pass

Figure 65: Results for the get error logs route unit testing

Get Fatal Logs Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Fatal_Logs_01	GET request: It should not return the fatal logs and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Fatal_Logs_02	GET request: It should not return the fatal logs and return a not authorized error because the user does not have administrator privileges.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Fatal_Logs_03	GET request: It should return the fatal logs.	Response with code 200 and an array with the fatal logs.	Response with code 200 and an array with the fatal logs.	Pass

Figure 66: Results for the get fatal logs route unit testing

Get Information Logs Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Info_Logs_01	GET request: It should not return the information logs and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Info_Logs_02	GET request: It should not return the information logs and return a not authorized error because the user does not have administrator privileges.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Info_Logs_03	GET request: It should return the information logs.	Response with code 200 and an array with the information logs.	Response with code 200 and an array with the information logs.	Pass

Figure 67: Results for the get info logs route unit testing

Get Warning Logs Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Warn_Logs_01	GET request: It should not return the warning logs and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Warn_Logs_02	GET request: It should not return the warning logs and return a not authorized error because the user does not have administrator privileges.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Warn_Logs_03	GET request: It should return the warning logs.	Response with code 200 and an array with the warning logs.	Response with code 200 and an array with the warning logs.	Pass

Figure 68: Results for the get warn logs route unit testing



## Market Microservice Routes Unit Testing

Add Limit Order Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Add_Limit_Order_01	POST request: It should not add limit order and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Add_Limit_Order_02	POST request: It should not add limit order and return a bad request error (invalid cryptocurrency).	Bad request error (code 400) with message: Crypto must be valid.	Bad request error (code 400) with message: Crypto must be valid.	Pass
UT_Add_Limit_Order_03	POST request: It should not add limit order and return a bad request error (invalid amount value).	Bad request error (code 400) with message: Amount must be valid decimal.	Bad request error (code 400) with message: Amount must be valid decimal.	Pass
UT_Add_Limit_Order_04	POST request: It should not add limit order and return a bad request error (invalid volume value).	Bad request error (code 400) with message: You must enter a valid volume (decimal with 2 decimal digits maximum).	Bad request error (code 400) with message: You must enter a valid volume (decimal with 2 decimal digits maximum).	Pass
UT_Add_Limit_Order_05	POST request: It should not add limit order and return a bad request error (order side must be valid).	Bad request error (code 400) with message: Side must be valid.	Bad request error (code 400) with message: Side must be valid.	Pass
UT_Add_Limit_Order_06	POST request: It should not add limit order and return a bad request error (cryptocurrency does not exist in the system).	Bad request error (code 400) with message: Crypto does not exist in the system.	Bad request error (code 400) with message: Crypto does not exist in the system.	Pass
UT_Add_Limit_Order_07	POST request: It should not add limit order and return a bad request error (order side can only be BUY or SELL).	Bad request error (code 400) with message: Order must be a buy or sell.	Bad request error (code 400) with message: Order must be a buy or sell.	Pass
UT_Add_Limit_Order_08	POST request: It should not add limit order and return a bad request error (invalid good until value).	Bad request error (code 400) with message: Good until value must be a valid unix timestamp.	Bad request error (code 400) with message: Good until value must be a valid unix timestamp.	Pass
UT_Add_Limit_Order_09	POST request: It should add limit order and respond with code 200.	Response with code 200.	Response with code 200.	Pass

Figure 69: Results for the add limit order route unit testing

Add Market Order Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Add_Market_Order_01	POST request: It should not add market order and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Add_Market_Order_02	POST request: It should not add market order and return a bad request error (invalid cryptocurrency).	Bad request error (code 400) with message: Crypto must be valid.	Bad request error (code 400) with message: Crypto must be valid.	Pass
UT_Add_Market_Order_03	POST request: It should not add market order and return a bad request error (invalid volume value).	Bad request error (code 400) with message: You must enter a valid volume (decimal with 2 decimal digits maximum).	Bad request error (code 400) with message: You must enter a valid volume (decimal with 2 decimal digits maximum).	Pass
UT_Add_Market_Order_04	POST request: It should not add market order and return a bad request error (order side must be valid).	Bad request error (code 400) with message: Side must be valid.	Bad request error (code 400) with message: Side must be valid.	Pass
UT_Add_Market_Order_05	POST request: It should not add market order and return a bad request error (cryptocurrency does not exist in the system).	Bad request error (code 400) with message: Crypto does not exist in the system.	Bad request error (code 400) with message: Crypto does not exist in the system.	Pass
UT_Add_Market_Order_06	POST request: It should not add market order and return a bad request error (order side can only be BUY or SELL).	Bad request error (code 400) with message: Order must be a buy or sell.	Bad request error (code 400) with message: Order must be a buy or sell.	Pass
UT_Add_Market_Order_07	POST request: It should not add market order and return a bad request error (invalid good until value).	Bad request error (code 400) with message: Good until value must be a valid unix timestamp.	Bad request error (code 400) with message: Good until value must be a valid unix timestamp.	Pass
UT_Add_Market_Order_08	POST request: It should add market order and respond with code 200.	Response with code 200.	Response with code 200.	Pass

Figure 70: Results for the add market order route unit testing

Cancel Order Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Cancel_Order_01	POST request: It should not cancel an order and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Cancel_Order_02	POST request: It should not cancel an order and return a bad request error (invalid order ID).	Bad request error (code 400) with message: You must enter a valid order ID number.	Bad request error (code 400) with message: You must enter a valid order ID number.	Pass
UT_Cancel_Order_03	POST request: It should not cancel an order and return a bad request error (order can only be SELL or BUY).	Bad request error (code 400) with message: Order must be a buy or sell.	Bad request error (code 400) with message: Order must be a buy or sell.	Pass
UT_Cancel_Order_04	POST request: It should cancel an order and respond with code 200.	Response with code 200.	Response with code 200.	Pass

Figure 71: Results for the cancel order route unit testing

Get Buy Orders History Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Buy_Orders_History_01	GET request: It should not return the history of buy orders and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Buy_Orders_History_02	GET request: It should return the history of buy orders.	Response with code 200 and an array with the history of buy orders.	Response with code 200 and an array with the history of buy orders.	Pass

Figure 72: Results for the get buy orders history route unit testing

Get Sell Orders History Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Sell_Orders_History_01	GET request: It should not return the history of sell orders and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Sell_Orders_History_02	GET request: It should return the history of sell orders.	Response with code 200 and an array with the history of sell orders.	Response with code 200 and an array with the history of sell orders.	Pass

Figure 73: Results for the get sell orders history route unit testing

Get Last Buy Orders History Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Last_Buy_Orders_History_01	GET request: It should not return the history of the last buy orders and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Last_Buy_Orders_History_02	GET request: It should return the history of the last buy orders.	Response with code 200 and an array with the history of the last buy orders.	Response with code 200 and an array with the history of the last buy orders.	Pass

Figure 74: Results for the get last buy orders history route unit testing

Get Last Sell Orders History Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Last_Sell_Orders_History_01	GET request: It should not return the history of the last sell orders and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Last_Sell_Orders_History_02	GET request: It should return the history of the last sell orders.	Response with code 200 and an array with the history of the last sell orders.	Response with code 200 and an array with the history of the last sell orders.	Pass

Figure 75: Results for the get last sell orders history route unit testing

Get User Buy Orders Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_User_Buy_Orders_01	GET request: It should not return the user buy orders and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_User_Buy_Orders_02	GET request: It should return the user buy orders.	Response with code 200 and an array with user buy orders.	Response with code 200 and an array with user buy orders.	Pass

Figure 76: Results for the get user buy orders route unit testing

Get User Sell Orders Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_User_Sell_Orders_01	GET request: It should not return the user sell orders and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_User_Sell_Orders_02	GET request: It should return the user sell orders.	Response with code 200 and an array with user sell orders.	Response with code 200 and an array with user sell orders.	Pass

Figure 77: Results for the get user sell orders route unit testing

Get Complete Orderbook Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Complete_Orderbook_01	GET request: It should not return the complete orderbook and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Complete_Orderbook_02	GET request: It should not return the complete orderbook and return a not authorized error because the user does not have administrator privileges.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Complete_Orderbook_03	GET request: It should return the complete orderbook.	Response with code 200 and an array with the complete orderbook.	Response with code 200 and an array with the complete orderbook.	Pass

Figure 78: Results for the get complete orderbook route unit testing

Get User Available Cryptocurrencies Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_User_Available_Cryptos_01	GET request: It should not return the user available cryptos and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_User_Available_Cryptos_02	GET request: It should return the user available cryptos.	Response with code 200 and an array with user available cryptos.	Response with code 200 and an array with user available cryptos.	Pass

Figure 79: Results for the get user available cryptocurrencies route unit testing

Get Number of Auctions Per Day Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Auctions_Per_Day_01	GET request: It should not return the number of auctions per day and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Auctions_Per_Day_02	GET request: It should return the number of auctions per day.	Response with code 200 and the number of auctions per day.	Response with code 200 and the number of auctions per day.	Pass

Figure 80: Results for the get number of actions per day route unit testing

Set Auctions Per Day Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Set_Auctions_Per_Day_01	POST request: It should not set number of auctions per day and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Set_Auctions_Per_Day_02	POST request: It should not set number of auctions per day and return a not authorized error because the user does not have administrator privileges.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Set_Auctions_Per_Day_03	POST request: It should not set number of auctions per day and return a bad request error (invalid value).	Bad request error (code 400) with message: You must enter a valid integer number from 1 to 24.	Bad request error (code 400) with message: You must enter a valid integer number from 1 to 24.	Pass
UT_Set_Auctions_Per_Day_04	POST request: It should set number of auctions per day and respond with code 200.	Response with code 200.	Response with code 200.	Pass

Figure 81: Results for the set number of actions per day route unit testing

### Transactions Microservice Routes Unit Testing

Edit Wallet Quantity Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Edit_Wallet_Quantity_01	POST request: It should not edit the wallet quantity and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Edit_Wallet_Quantity_02	POST request: It should not edit the wallet quantity and return a not authorized error because the user does not have administrator privileges.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Edit_Wallet_Quantity_03	POST request: It should not edit the wallet quantity and return a bad request error (invalid wallet address).	Bad request error (code 400) with message: You must enter a valid wallet address.	Bad request error (code 400) with message: You must enter a valid wallet address.	Pass
UT_Edit_Wallet_Quantity_04	POST request: It should not edit crypto and return a bad request error (invalid cryptocurrency name).	Bad request error (code 400) with message: You must enter a valid cryptocurrency name.	Bad request error (code 400) with message: You must enter a valid cryptocurrency name.	Pass
UT_Edit_Wallet_Quantity_05	POST request: It should not edit crypto and return a bad request error (invalid quantity value).	Bad request error (code 400) with message: You must enter a valid quantity (decimal value).	Bad request error (code 400) with message: You must enter a valid quantity (decimal value).	Pass
UT_Edit_Wallet_Quantity_06	POST request: It should not edit crypto and return a bad request error (inexistent wallet address).	Bad request error (code 400) with message: You must supply an existing wallet.	Bad request error (code 400) with message: You must supply an existing wallet.	Pass
UT_Edit_Wallet_Quantity_07	POST request: It should edit the wallet quantity and return a success message.	Response with code 200 and a success message.	Response with code 200 and a success message.	Pass

Figure 82: Results for the edit wallet quantity route unit testing

Make Payment Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Make_Payment_01	POST request: It should not process the payment and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Make_Payment_02	POST request: It should not process the payment and return a bad request error (invalid payment payload format).	Bad request error (code 400) with message: Payment payload must have a valid format.	Bad request error (code 400) with message: Payment payload must have a valid format.	Pass
UT_Make_Payment_03	POST request: It should not edit crypto and return a bad request error (inexistent wallet address).	Bad request error (code 400) with message: You must supply an existing wallet.	Bad request error (code 400) with message: You must supply an existing wallet.	Pass
UT_Make_Payment_04	POST request: It should not edit crypto and return a bad request error (insufficient funds).	Bad request error (code 400) with message: You have insufficient funds.	Bad request error (code 400) with message: You have insufficient funds.	Pass
UT_Make_Payment_05	POST request: It should process the payment and return a success message.	Response with code 200 and a success message.	Response with code 200 and a success message.	Pass

Figure 83: Results for the make payment route unit testing

Send Cryptocurrency Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Send_Crypto_01	POST request: It should not send cryptocurrency and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Send_Crypto_02	POST request: It should not send cryptocurrency and return a bad request error (invalid phone number).	Bad request error (code 400) with message: Phone Number must be valid.	Bad request error (code 400) with message: Phone Number must be valid.	Pass
UT_Send_Crypto_03	POST request: It should not send cryptocurrency and return a bad request error (invalid decimal value).	Bad request error (code 400) with message: Amount must be valid decimal.	Bad request error (code 400) with message: Amount must be valid decimal.	Pass
UT_Send_Crypto_04	POST request: It should not send cryptocurrency and return a bad request error (invalid transaction cost value).	Bad request error (code 400) with message: Transaction cost must be valid decimal.	Bad request error (code 400) with message: Transaction cost must be valid decimal.	Pass
UT_Send_Crypto_05	POST request: It should not send cryptocurrency and return a bad request error (inexistent wallet address).	Bad request error (code 400) with message: You must supply an existing wallet.	Bad request error (code 400) with message: You must supply an existing wallet.	Pass
UT_Send_Crypto_06	POST request: It should not send cryptocurrency and return a bad request error (insufficient funds).	Bad request error (code 400) with message: You have insufficient funds.	Bad request error (code 400) with message: You have insufficient funds.	Pass
UT_Send_Crypto_07	POST request: It should send cryptocurrency and return a success message.	Response with code 200 and a success message.	Response with code 200 and a success message.	Pass

Figure 84: Results for the send cryptocurrency route unit testing

## Wallet Microservice Routes Unit Testing

Get Fiat Balance Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Fiat_Balance_01	GET request: It should not return the user fiat currency balance and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Fiat_Balance_02	GET request: It should return the user fiat currency balance.	Response with code 200 and the fiat balance value.	Response with code 200 and the fiat balance value.	Pass

Figure 85: Results for the get fiat balance route unit testing

Get Wallet Filters Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Wallet_Filters_01	GET request: It should not return the wallet filters and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Wallet_Filters_02	GET request: It should return the wallet filters.	Response with code 200 and an array with the wallet filters.	Response with code 200 and an array with the wallet filters.	Pass

Figure 86: Results for the get wallet filters route unit testing

Get Wallet Information Route				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
UT_Get_Wallet_Info_01	GET request: it should not return the wallet information and return a not authorized error because there's no session jwt token in the request.	Not authorized error with code 401.	Not authorized error with code 401.	Pass
UT_Get_Wallet_Info_02	GET request: It should return the wallet information.	Response with code 200 and an array with the wallet information.	Response with code 200 and an array with the wallet information.	Pass

Figure 87: Results for the get wallet information route unit testing

This page is intentionally left blank.

# Appendix C

## End-To-End Testing Results

Register				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Register_01	It should not register the user and print an invalid invitation code message.	HTML div with an invalid invitation code message: "You must enter a valid invitation code (6 digits)".	HTML div with an invalid invitation code message: "You must enter a valid invitation code (6 digits)".	Pass
E2E_Register_02	It should not register the user and print an invalid phone number message.	HTML div with an invalid phone number message: "You must enter a valid phone number from Portugal".	HTML div with an invalid phone number message: "You must enter a valid phone number from Portugal".	Pass
E2E_Register_03	It should not register the user and print an invalid email message.	HTML div with an invalid email message: "You must enter a valid email address".	HTML div with an invalid email message: "You must enter a valid email address".	Pass
E2E_Register_04	It should not register the user and print an invalid full name message.	HTML div with an invalid full name message: "You must enter a valid full name with only letters and at least 5 characters".	HTML div with an invalid full name message: "You must enter a valid full name with only letters and at least 5 characters".	Pass
E2E_Register_05	It should not register the user and print an invalid password message.	HTML div with an invalid password message: "You must enter a valid password (length between 8 and 20, at least 1 lowercase and 1 uppercase letter, 1 number and 1 symbol)".	HTML div with an invalid password message: "You must enter a valid password (length between 8 and 20, at least 1 lowercase and 1 uppercase letter, 1 number and 1 symbol)".	Pass
E2E_Register_06	It should not register the user and print a mismatch password message.	HTML div with a mismatch password message: "Both password must match".	HTML div with a mismatch password message: "Both password must match".	Pass
E2E_Register_07	It should not register the user and print an invalid PIN code message.	HTML div with an invalid PIN code message: "You must enter a valid PIN (6 digits)".	HTML div with an invalid PIN code message: "You must enter a valid PIN (6 digits)".	Pass
E2E_Register_08	It should not register the user and print a phone number/email already in use message.	HTML div with a phone number/email already in use message: "Email or phone number in use".	HTML div with a phone number/email already in use message: "Email or phone number in use".	Pass
E2E_Register_09	It should not register the user and print an email has no invite associated message.	HTML div with an email has no invite associated message: "This email has no invite associated".	HTML div with an email has no invite associated message: "This email has no invite associated".	Pass
E2E_Register_10	It should not register the user and print an invalid invitation code inserted message.	HTML div with an invalid invitation code inserted message: "Invalid invitation code inserted".	HTML div with an invalid invitation code inserted message: "Invalid invitation code inserted".	Pass
E2E_Register_11	It should register the user and redirect to the home page directory.	Register the user in the system and redirect to the home page directory.	Register the user in the system and redirect to the home page directory.	Pass

Figure 88: End-to-end testing results for the register action

Login				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Login_01	It should not login the user and print an invalid phone number message.	HTML div with an invalid phone number message: "You must enter a valid phone number from Portugal".	HTML div with an invalid phone number message: "You must enter a valid phone number from Portugal".	Pass
E2E_Login_02	It should not login the user and print an invalid password message.	HTML div with an invalid phone number message: "You must enter a valid password (length between 8 and 20, at least 1 lowercase and 1 uppercase letter, 1 number and 1 symbol)".	HTML div with an invalid phone number message: "You must enter a valid password (length between 8 and 20, at least 1 lowercase and 1 uppercase letter, 1 number and 1 symbol)".	Pass
E2E_Login_03	It should not login the user and print an invalid credentials message.	HTML div with an invalid credentials message: "Invalid credentials".	HTML div with an invalid credentials message: "Invalid credentials".	Pass
E2E_Login_04	It should login the user, set a session cookie with a jwt token and redirect to the dashboard directory.	Login the user, set a session cookie with a jwt token and redirect to the dashboard directory.	Login the user, set a session cookie with a jwt token and redirect to the dashboard directory.	Pass

Figure 89: End-to-end testing results for the login action

Forgot Password				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Forgot_Password_01	It should not submit the forgot password form and print an invalid email or phone number message.	HTML div with an invalid email or phone number message: "You must enter a valid phone number from Portugal or email address.".	HTML div with an invalid email or phone number message: "You must enter a valid phone number from Portugal or email address.".	Pass
E2E_Forgot_Password_02	It should not submit the forgot password form and print an invalid phone number message.	HTML div with an invalid phone number message: "You must enter a valid phone number from Portugal or email address.".	HTML div with an invalid phone number message: "You must enter a valid phone number from Portugal or email address.".	Pass
E2E_Forgot_Password_03	It should not submit the forgot password form and print an invalid email message.	HTML div with an invalid email message: "You must enter a valid phone number from Portugal or email address.".	HTML div with an invalid email message: "You must enter a valid phone number from Portugal or email address.".	Pass
E2E_Forgot_Password_04	It should submit the forgot password form and redirect to the password recovery directory.	Submit the forgot password form and redirect to the password recovery directory.	Submit the forgot password form and redirect to the password recovery directory.	Pass

Figure 90: End-to-end testing results for the forgot password action

Contact Us While Logged Off				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Contact_Us_Logged_Off_01	It should not submit the contact us form and print an invalid email message.	HTML div with an invalid email message: "You must enter a valid email address".	HTML div with an invalid email: "You must enter a valid email address".	Pass
E2E_Contact_Us_Logged_Off_02	It should not submit the contact us form and print an invalid full name message.	HTML div with an invalid full name message: "You must enter a valid name with only letters and at least 5 characters".	HTML div with an invalid full name message: "You must enter a valid name with only letters and at least 5 characters".	Pass
E2E_Contact_Us_Logged_Off_03	It should not submit the contact us form and print an invalid subject message.	HTML div with an invalid subject message: "Subject must not be empty".	HTML div with an invalid subject message: "Subject must not be empty".	Pass
E2E_Contact_Us_Logged_Off_04	It should not submit the contact us form and print an invalid "contact message" message.	HTML div with an invalid "contact message" message: "Message must not be empty".	HTML div with an invalid "contact message" message: "Message must not be empty".	Pass
E2E_Contact_Us_Logged_Off_05	It should submit the contact us form and redirect to the home page directory.	Submit the contact us form and redirect to the home page directory.	Submit the contact us form and redirect to the home page directory.	Pass

Figure 91: End-to-end testing results for the contact us while logged off action

Send Cryptocurrency				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Send_Crypto_01	It should not submit the send cryptocurrency form and print an invalid amount message.	HTML div with an invalid amount message: "You must enter a valid amount".	HTML div with an invalid amount message: "You must enter a valid amount".	Pass
E2E_Send_Crypto_02	It should not submit the send cryptocurrency form and print an invalid phone number message.	HTML div with an invalid phone number message: "You must enter a valid phone number from Portugal".	HTML div with an invalid phone number message: "You must enter a valid phone number from Portugal".	Pass
E2E_Send_Crypto_03	It should not submit the send cryptocurrency form and print an invalid cryptocurrency message.	HTML div with an invalid cryptocurrency message: "You must choose one cryptocurrency".	HTML div with an invalid cryptocurrency message: "You must choose one cryptocurrency".	Pass
E2E_Send_Crypto_04	It should not submit the send cryptocurrency form and print an insufficient balance message.	HTML div with an insufficient balance message: "You have insufficient funds".	HTML div with an insufficient balance message: "You have insufficient funds".	Pass
E2E_Send_Crypto_05	It should not submit the send cryptocurrency form and print an inexistent wallet message.	HTML div with an inexistent wallet message: "You must supply an existing wallet".	HTML div with an inexistent wallet message: "You must supply an existing wallet".	Pass
E2E_Send_Crypto_06	It should submit the send cryptocurrency form and display a success message.	Submit the send cryptocurrency form and display a success message.	Submit the send cryptocurrency form and display a success message.	Pass

Figure 92: End-to-end testing results for the send cryptocurrency action

Request Cryptocurrency				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Request_Crypto_01	It should not submit the request cryptocurrency form and print an invalid amount message.	HTML div with an invalid amount message: "You must enter a valid amount".	HTML div with an invalid amount message: "You must enter a valid amount".	Pass
E2E_Request_Crypto_02	It should not submit the request cryptocurrency form and print an invalid cryptocurrency message.	HTML div with an invalid cryptocurrency message: "You must choose one cryptocurrency".	HTML div with an invalid cryptocurrency message: "You must choose one cryptocurrency".	Pass
E2E_Request_Crypto_03	It should submit the request cryptocurrency form and display a QR code.	Submit the request cryptocurrency form and display a QR code.	Submit the request cryptocurrency form and display a QR code.	Pass

Figure 93: End-to-end testing results for the request cryptocurrency action

Add Market Sell Order				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Add_Market_Sell_Order_01	It should not add a market sell order and print an invalid amount message.	HTML div with an invalid amount message: "You must enter a valid amount".	HTML div with an invalid amount message: "You must enter a valid amount".	Pass
E2E_Add_Market_Sell_Order_02	It should not add a market sell order and print an invalid cryptocurrency selected message.	HTML div with an invalid cryptocurrency selected: "You must choose one cryptocurrency".	HTML div with an invalid cryptocurrency selected: "You must choose one cryptocurrency".	Pass
E2E_Add_Market_Sell_Order_03	It should not add a market sell order and print an insufficient balance message.	HTML div with an insufficient balance message: "You have insufficient balance".	HTML div with an insufficient balance message: "You have insufficient balance".	Pass
E2E_Add_Market_Sell_Order_04	It should not add a market sell order and print a wrong PIN message.	HTML div with a wrong PIN message: "Pin inserted is wrong. Try again!".	HTML div with a wrong PIN message: "Pin inserted is wrong. Try again!".	Pass
E2E_Add_Market_Sell_Order_05	It should add a market sell order and display a success message.	Add a market sell order and display a success message.	Add a market sell order and display a success message.	Pass

Figure 94: End-to-end testing results for the add market sell order action

Add Limit Sell Order				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Add_Limit_Sell_Order_01	It should not add a limit sell order and print an invalid amount message.	HTML div with an invalid amount message: "You must enter a valid amount".	HTML div with an invalid amount message: "You must enter a valid amount".	Pass
E2E_Add_Limit_Sell_Order_02	It should not add a limit sell order and print an invalid price message.	HTML div with an invalid price message: "You must enter a valid price".	HTML div with an invalid price message: "You must enter a valid price".	Pass
E2E_Add_Limit_Sell_Order_03	It should not add a limit sell order and print an invalid cryptocurrency selected message.	HTML div with an invalid cryptocurrency selected: "You must choose one cryptocurrency".	HTML div with an invalid cryptocurrency selected: "You must choose one cryptocurrency".	Pass
E2E_Add_Limit_Sell_Order_04	It should not add a limit sell order and print an insufficient balance message.	HTML div with an insufficient balance message: "You have insufficient balance".	HTML div with an insufficient balance message: "You have insufficient balance".	Pass
E2E_Add_Limit_Sell_Order_05	It should not add a limit sell order and print a wrong PIN message.	HTML div with a wrong PIN message: "Pin inserted is wrong. Try again!".	HTML div with a wrong PIN message: "Pin inserted is wrong. Try again!".	Pass
E2E_Add_Limit_Sell_Order_06	It should add a limit sell order and display a success message.	Add a limit sell order and display a success message.	Add a limit sell order and display a success message.	Pass

Figure 95: End-to-end testing results for the add limit sell order action



Add Market Buy Order				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Add_Market_Buy_Order_01	It should not add a market buy order and print an invalid amount message.	HTML div with an invalid amount message: "You must enter a valid amount".	HTML div with an invalid amount message: "You must enter a valid amount".	Pass
E2E_Add_Market_Buy_Order_02	It should not add a market buy order and print an invalid cryptocurrency selected message.	HTML div with an invalid cryptocurrency selected: "You must choose one cryptocurrency".	HTML div with an invalid cryptocurrency selected: "You must choose one cryptocurrency".	Pass
E2E_Add_Market_Buy_Order_03	It should not add a market buy order and print a wrong PIN message.	HTML div with a wrong PIN message: "Pin inserted is wrong. Try again!".	HTML div with a wrong PIN message: "Pin inserted is wrong. Try again!".	Pass
E2E_Add_Market_Buy_Order_04	It should add a market buy order and display a success message.	Add a market buy order and display a success message.	Add a market buy order and display a success message.	Pass

Figure 96: End-to-end testing results for the add market buy order action

Add Limit Buy Order				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Add_Limit_Buy_Order_01	It should not add a limit buy order and print an invalid amount message.	HTML div with an invalid amount message: "You must enter a valid amount".	HTML div with an invalid amount message: "You must enter a valid amount".	Pass
E2E_Add_Limit_Buy_Order_02	It should not add a limit buy order and print an invalid price message.	HTML div with an invalid price message: "You must enter a valid price".	HTML div with an invalid price message: "You must enter a valid price".	Pass
E2E_Add_Limit_Buy_Order_03	It should not add a limit buy order and print an invalid cryptocurrency selected message.	HTML div with an invalid cryptocurrency selected: "You must choose one cryptocurrency".	HTML div with an invalid cryptocurrency selected: "You must choose one cryptocurrency".	Pass
E2E_Add_Limit_Buy_Order_04	It should not add a limit buy order and print an insufficient balance message.	HTML div with an insufficient balance message: "You have insufficient balance".	HTML div with an insufficient balance message: "You have insufficient balance".	Pass
E2E_Add_Limit_Buy_Order_05	It should not add a limit buy order and print a wrong PIN message.	HTML div with a wrong PIN message: "Pin inserted is wrong. Try again!".	HTML div with a wrong PIN message: "Pin inserted is wrong. Try again!".	Pass
E2E_Add_Limit_Buy_Order_06	It should add a limit buy order and display a success message.	Add a limit buy order and display a success message.	Add a limit buy order and display a success message.	Pass

Figure 97: End-to-end testing results for the add limit buy order action

Edit Personal Information				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Edit_Personal_Info_01	It should not edit the personal information and print an invalid full name message.	HTML div with an invalid full name message: "You must enter a valid full name with only letters and at least 5 characters".	HTML div with an invalid full name message: "You must enter a valid full name with only letters and at least 5 characters".	Pass
E2E_Edit_Personal_Info_02	It should not edit the personal information and print a mismatch password message.	HTML div with a mismatch password message: "Both password must match".	HTML div with a mismatch password message: "Both password must match".	Pass
E2E_Edit_Personal_Info_03	It should not edit the personal information and print an invalid password message.	HTML div with an invalid password message: "You must enter a valid password (length between 8 and 20, at least 1 lowercase and 1 uppercase letter, 1 number and 1 symbol)".	HTML div with an invalid password message: "You must enter a valid password (length between 8 and 20, at least 1 lowercase and 1 uppercase letter, 1 number and 1 symbol)".	Pass
E2E_Edit_Personal_Info_04	It should edit the personal information and redirect to the settings page directory.	Edit the personal information and redirect to the settings page directory.	Edit the personal information and redirect to the settings page directory.	Pass

Figure 98: End-to-end testing results for the edit personal information action

Edit PIN Code				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Edit_PIN_Code_01	It should not edit the PIN code and print a mismatch password message.	HTML div with a mismatch password message: "Both PIN codes must match".	HTML div with a mismatch password message: "Both PIN codes must match".	Pass
E2E_Edit_PIN_Code_02	It should not edit the PIN code and print an invalid PIN code message.	HTML div with an invalid PIN code message: "Both password must match".	HTML div with a mismatch password message: "Both password must match".	Pass
E2E_Edit_PIN_Code_03	It should edit the PIN code and redirect to the settings page directory.	Edit the PIN code and redirect to the settings page directory.	Edit the PIN code and redirect to the settings page directory.	Pass

Figure 99: End-to-end testing results for the edit PIN code action

Invite User				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Invite_User_01	It should not invite the user and print an invalid phone number message.	HTML div with an invalid phone number message: "You must enter a valid email".	HTML div with an invalid phone number message: "You must enter a valid email".	Pass
E2E_Invite_User_02	It should not invite the user and print an invalid password message.	HTML div with an invalid password message: "You must enter a valid initial amount (decimal number)".	HTML div with an invalid password message: "You must enter a valid initial amount (decimal number)".	Pass
E2E_Invite_User_03	It should invite the user and redirect to the administrator panel page directory.	Invite the user and redirect to the administrator panel page directory.	Invite the user and redirect to the administrator panel page directory.	Pass

Figure 100: End-to-end testing results for the invite user action



Add Cryptocurrency				
Test ID	Description	Result Expected	Result Obtained	Pass/Fail
E2E_Add_Crypto_01	It should not add the cryptocurrency and print an invalid name message.	HTML div with an invalid name message: "You must enter a valid name".	HTML div with an invalid name message: "You must enter a valid name".	Pass
E2E_Add_Crypto_02	It should not add the cryptocurrency and print an invalid symbol message.	HTML div with an invalid symbol message: "You must enter a valid symbol for the cryptocurrency".	HTML div with an invalid symbol message: "You must enter a valid symbol for the cryptocurrency".	Pass
E2E_Add_Crypto_03	It should not add the cryptocurrency and print an invalid price message.	HTML div with an invalid price message: "You must enter a valid price".	HTML div with an invalid price message: "You must enter a valid price".	Pass
E2E_Add_Crypto_04	It should not add the cryptocurrency and print an invalid volume message.	HTML div with an invalid volume message: "You must enter a valid volume".	HTML div with an invalid volume message: "You must enter a valid volume".	Pass
E2E_Add_Crypto_05	It should not add the cryptocurrency and print an invalid privacy message.	HTML div with an invalid privacy message: "You must enter a valid privacy (private, semi-private, public)".	HTML div with an invalid privacy message: "You must enter a valid privacy (private, semi-private, public)".	Pass
E2E_Add_Crypto_06	It should not add the cryptocurrency and print an invalid transaction cost message.	HTML div with an invalid transaction cost message: "You must enter a valid transaction cost (percentage: between 0 and 100)".	HTML div with an invalid transaction cost message: "You must enter a valid transaction cost (percentage: between 0 and 100)".	Pass
E2E_Add_Crypto_07	It should not add the cryptocurrency and print an invalid settlement time message.	HTML div with an invalid settlement time message: "You must enter a valid settlement time".	HTML div with an invalid settlement time message: "You must enter a valid settlement time".	Pass
E2E_Add_Crypto_08	It should add the cryptocurrency and redirect to the administrator panel page directory.	Add the cryptocurrency and redirect to the administrator panel page directory.	Add the cryptocurrency and redirect to the administrator panel page directory.	Pass

Figure 101: End-to-end testing results for the add cryptocurrency action

---

# Appendix D

## Usability Testing Form

8/31/2021

Usability Test for Crypta Project - Pre Questionnaire

### Usability Test for Crypta Project - Pre Questionnaire

I am currently working on my thesis which consists in developing a mobile web application capable of simulating a cryptocurrencies market and wallet. But now, I need your help to perform usability tests in order to check how well different people can use this application. This questionnaire will only be used for academic purposes.

*\*Obrigatório*

1. Age \*

---

2. Academic Qualifications \*

---

3. Academic Field

---

4. Current Job \*

---

5. Rank your experience with mobile web applications. \*

*Marcas apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

---

[https://docs.google.com/forms/d/1Lpvj9x7ah783H-Hic9S\\_k-sEU66fD\\_wbJGLTSMJqbfq/edit](https://docs.google.com/forms/d/1Lpvj9x7ah783H-Hic9S_k-sEU66fD_wbJGLTSMJqbfq/edit)

1/15

Figure 102: Usability testing page 1

8/31/2021

Usability Test for Crypta Project - Pre Questionnaire

6. Rank your knowledge of cryptocurrencies. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

7. Rank your knowledge of trading markets. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

8. Rank your knowledge of mobile wallets. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

9. Rank your experience with mobile market trading applications. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure 103: Usability testing page 2

10. Rank your experience with mobile wallet applications. \*

*Marcar apenas uma oval.*

1    2    3    4    5

---

---

Usability  
Test for  
Crypta  
Project -  
Performing  
Tasks

Now that you filled the pre-questionnaire, I need you to perform the following list of tasks and fill the form with the results obtained. In order to start the tasks, you need to access the mobile web application through the link <https://crypta.dei.uc.pt>. Then you need to try to perform each task and track how many clicks you did to complete it. Finally you need to fill the line, correspondent to the task that you did, with the number of clicks, if you succeeded or failed and rank the difficulty level from 1 to 5 (being 1 - very easy and 5 - very hard). The tasks must be performed in sequential order and you can not count value inputs as clicks.

Task 1 - Register into the application through the email received.

11. Number of clicks. \*

---

12. Task success. \*

*Marcar apenas uma oval.*

Success  
 Fail

13. Difficulty level. \*

*Marcar apenas uma oval.*

1    2    3    4    5

---

---

Task 2 - Login into the application.

Figure 104: Usability testing page 3

8/31/2021

Usability Test for Crypta Project - Pre Questionnaire

14. Number of clicks. \*

---

15. Task success. \*

*Marcar apenas uma oval.*

Success

Fail

16. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 3 - Change wallet filters to see only the first cryptocurrency in the list.

17. Number of clicks. \*

---

18. Task success. \*

*Marcar apenas uma oval.*

Success

Fail

Figure 105: Usability testing page 4

19. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 4 - Check all activity in the wallet.

20. Number of clicks. \*

---

21. Task success. \*

*Marcar apenas uma oval.*

Success  
 Fail

22. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 5 - Make payment with the QR code below.

Figure 106: Usability testing page 5



23. Number of clicks. \*

---

24. Task success. \*

*Marcar apenas uma oval.*

Success

Fail

25. Difficulty level. \*

*Marcar apenas uma oval.*

1    2    3    4    5

---

---

Task 6 - Send 0.01 Bitcoin (BTC) to the phone number 966666666.

26. Number of clicks. \*

---

Figure 107: Usability testing page 6

27. Task success. \*

*Marcar apenas uma oval.*

Success

Fail

28. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 7 - Generate a QR code to request 0.01 Bitcoin (BTC).

29. Number of clicks. \*

---

30. Task success. \*

*Marcar apenas uma oval.*

Success

Fail

31. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure 108: Usability testing page 7



8/31/2021

Usability Test for Crypta Project - Pre Questionnaire

Task 8 - Create a market sell order for 0.01 Bitcoin (BTC) that is only good until tomorrow.

32. Number of clicks. \*

---

33. Task success. \*

*Marcar apenas uma oval.*

Success

Fail

34. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 9 - Create a limit buy order for 0.01 Bitcoin (BTC) at 14.42€ that is good until cancelled.

35. Number of clicks. \*

---

36. Task success. \*

*Marcar apenas uma oval.*

Success

Fail

[https://docs.google.com/forms/d/1Lpvj9x7ah783H-Hic9S\\_k-sEU66fD\\_wbJGLTSMJqbfg/edit](https://docs.google.com/forms/d/1Lpvj9x7ah783H-Hic9S_k-sEU66fD_wbJGLTSMJqbfg/edit)

8/15

Figure 109: Usability testing page 8

37. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 10 - Check your open and then closed orders.

38. Number of clicks. \*

---

39. Task success. \*

*Marcar apenas uma oval.*

Success  
 Fail

40. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 11 - Cancel the market sell order for 0.01 Bitcoin (BTC).

41. Number of clicks. \*

---

Figure 110: Usability testing page 9

8/31/2021

Usability Test for Crypta Project - Pre Questionnaire

42. Task success. \*

*Marcar apenas uma oval.*

Success

Fail

43. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 12 - Edit your personal information (full name).

44. Number of clicks. \*

---

45. Task success. \*

*Marcar apenas uma oval.*

Success

Fail

46. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 13 - Edit your security settings (PIN code).

[https://docs.google.com/forms/d/1Lpvj9x7ah783H-Hic9S\\_k-sEU66fD\\_wbJGLTSMJqbfg/edit](https://docs.google.com/forms/d/1Lpvj9x7ah783H-Hic9S_k-sEU66fD_wbJGLTSMJqbfg/edit)

10/15

Figure 111: Usability testing page 10

47. Number of clicks. \*

---

48. Task success. \*

*Marcar apenas uma oval.*

Success

Fail

49. Difficulty level. \*

*Marcar apenas uma oval.*

1    2    3    4    5

---

---

Task 14 - Check the instructions to reset your password.

50. Number of clicks. \*

---

51. Task success. \*

*Marcar apenas uma oval.*

Success

Fail

Figure 112: Usability testing page 11

8/31/2021

Usability Test for Crypta Project - Pre Questionnaire

52. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 15 - Visit the "About" section of the application.

53. Number of clicks. \*

---

54. Task success. \*

*Marcar apenas uma oval.*

Success  
 Fail

55. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 16 - Contact the application support team and send a message with subject "Greeting" and message "Hello team."

56. Number of clicks. \*

---

Figure 113: Usability testing page 12

57. Task success. \*

*Marcar apenas uma oval.*

- Success
- Fail

58. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Task 17 - Logout of the application.

59. Number of clicks. \*

---

60. Task success. \*

*Marcar apenas uma oval.*

- Success
- Fail

61. Difficulty level. \*

*Marcar apenas uma oval.*

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure 114: Usability testing page 13

8/31/2021

Usability Test for Crypta Project - Pre Questionnaire

Usability Test for Crypta  
Project - Pos  
Questionnaire

Finally, fill the following questionnaire, so we can get some feedback of how you feel about using our mobile web application.

62. The application was easy to understand. \*

*Marcar apenas uma oval.*

- Strongly Disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

63. The application was easy to use (complete all the tasks). \*

*Marcar apenas uma oval.*

- Strongly Disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

64. If you would open the application tomorrow, you could still find and use all the functionalities. \*

*Marcar apenas uma oval.*

- Strongly Disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

Figure 115: Usability testing page 14

65. You found the application UI (user interface) pleasant to use. \*

*Marcar apenas uma oval.*

- Strongly Disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

66. Recommendations to improve the application.

---

---

---

---

---

Thank you so much for your time and help!

---

Este conteúdo não foi criado nem aprovado pela Google.

Google Formulários

Figure 116: Usability testing page 15



This page is intentionally left blank.