Web Channel for a Mobile money system

Master degree in Computer Engineering – Mobile Computing

Rúben Filipe Simões Pereira

Leiria, September 2019

Web Channel for a Mobile money system

Master degree in Computer Engineering – Mobile Computing

Rúben Filipe Simões Pereira

Internship Report under the supervision of Professor Patrício Rodrigues Domingues, and Tiago Fonseca Marto.

Leiria, September 2019

Originality and Copyright

This internship report is original, made only for this purpose, and all authors whose studies and publications were used to complete it are duly acknowledged.

Dedication

I want to dedicate this thesis to my family and friends for all the motivation and support to embrace the challenge of proceeding with my academic studies and join the Master degree of Computer Engineering – Mobile Computing. I also would like to thank them for all the patience and understanding that they had during this latest academic year. Thank you so much.

Acknowledgments

I want to thank professor Patrício Rodrigues Domingues for all the help and work dedicated to me and this thesis. Without his guidance, it would not be possible to reach the achieved results.

Another person that had a unique role during the development of the internship project was Tiago Fonseca Marto. His help to understand the business logic and guidance to keep the internship project on the right track was essential.

Also, I would like to thank the M-Pesa development team. In particular to some of my co-workers as Pedro Bento, Leandro Almeida, David Matos and Diogo Correia for all the fellowship, support and insight that they gave me about the M-Pesa project.

Finally, I would like to thank WIT Software for the internship opportunity and the working conditions that they provided.

Abstract

Nowadays, 31% of the population worldwide still classified as Underbanked. These do not have access to mainstream financial services or institutions, and this is a problem that especially haunts the emerging markets. As the mobile phone penetration rate is tremendous, financial companies are investing in the development of mobile banking solutions to solve this issue. M-Pesa is one of those solutions that currently is being developed and deployed in several of these markets as in Tanzania, Mozambique, Democratic Republic of the Congo and others.

The appearing of solutions like M-Pesa allowed its users to have an online bank account and perform operations as deposit, transfer or withdraw money. Despite the innovation and these solutions give them the possibility of pay taxes, bills or services, they continue excluded from the online shopping world. This exclusion happens because payment gateways are not prepared to support users that do not use the traditional banking systems.

Online Payments platform aims to allow M-Pesa users to shop online. This platform is an online payment gateway that recurring to a third-party payment processor allows M-Pesa users to purchase on online stores. Is part of the prototype a web platform, that is composed by the following portals: i) merchant; ii) developer; iii) checkout; iv) documentation and v) demo and the Android M-Pesa application that was used to add support to new features for the Online Payments platform. This platform is a proof of concept developed during an internship at WIT Software, that is the company responsible for the development process of the M-Pesa mobile applications.

Finally, as far as the final phase of the project is concerned, have been obtained proof of concept thought which can see that it is possible to implement it.

Keywords: online payment system, payment gateway, mobile payments, M-Pesa

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

API	Application Programming Interface
ATM	Automated Teller Machine
AWS	Amazon Web Service
CSS	Cascading Style Sheets
CRM	Customer Relationship Manager
CRUD	Create, Read, Update and Delete
DB	Database
DTO	Data Transfer Object
FCM	Firebase Cloud Messaging
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ID	Identifier
JPA	Java Persistence API
JSON	JavaScript Object Notation
JWT	JSON Web Token
ORM	Object-relational Mapping
OTP	One Time Password
POM	POM Project Object Model
REST	Representational State Transfer
SIM	Subscriber Identity Module
SMS	Short Message Service
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
VM	VM Virtual Machine
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USSD	Unstructured Supplementary Service Data
UX	User Experience

1. Introduction

As stated by Holdgson, "Two billion people worldwide do not have a bank account or access to a financial institution via a mobile phone, or any other device" [1]. In 2014, only 62% of people over 15 years old of the worldwide population had an account in his name. The other 38% is classified as Underbanked, which means that for some reason, they do not have access to mainstream financial solutions, and this is a very common problem in emerging markets [2].

As it is possible to see in Figure 1, there might be several reasons for this such as they lack the necessary documentation, they do not have access to the financial mainstream services or simple as the fees imposed by these financial services are too high.

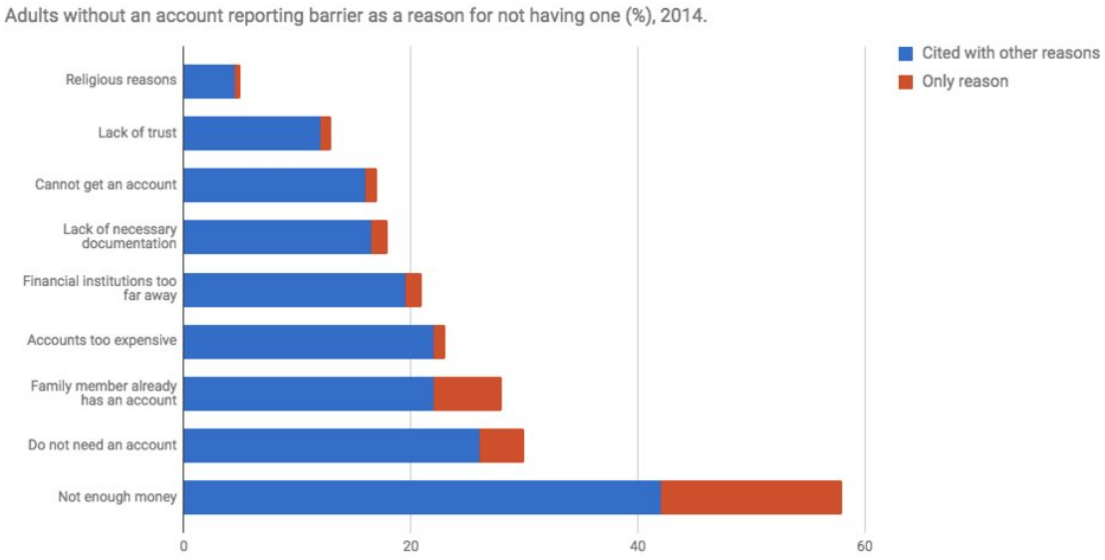

Figure 1 - Most common reasons for not having a bank account **[1]**

In the past years, these numbers have become more optimistic. The percentage of the global population that has a bank account with a financial institution or mobile banking service grew from 62% to 69% [3]. Given the high penetration of mobile phones in these markets, as shown in Figure 2, one of the solutions that have been explored to solve this problem is mobile banking.

Mobile banking is a service provided by a bank or other financial institution that allows its customers to conduct financial transactions remotely using a mobile device.

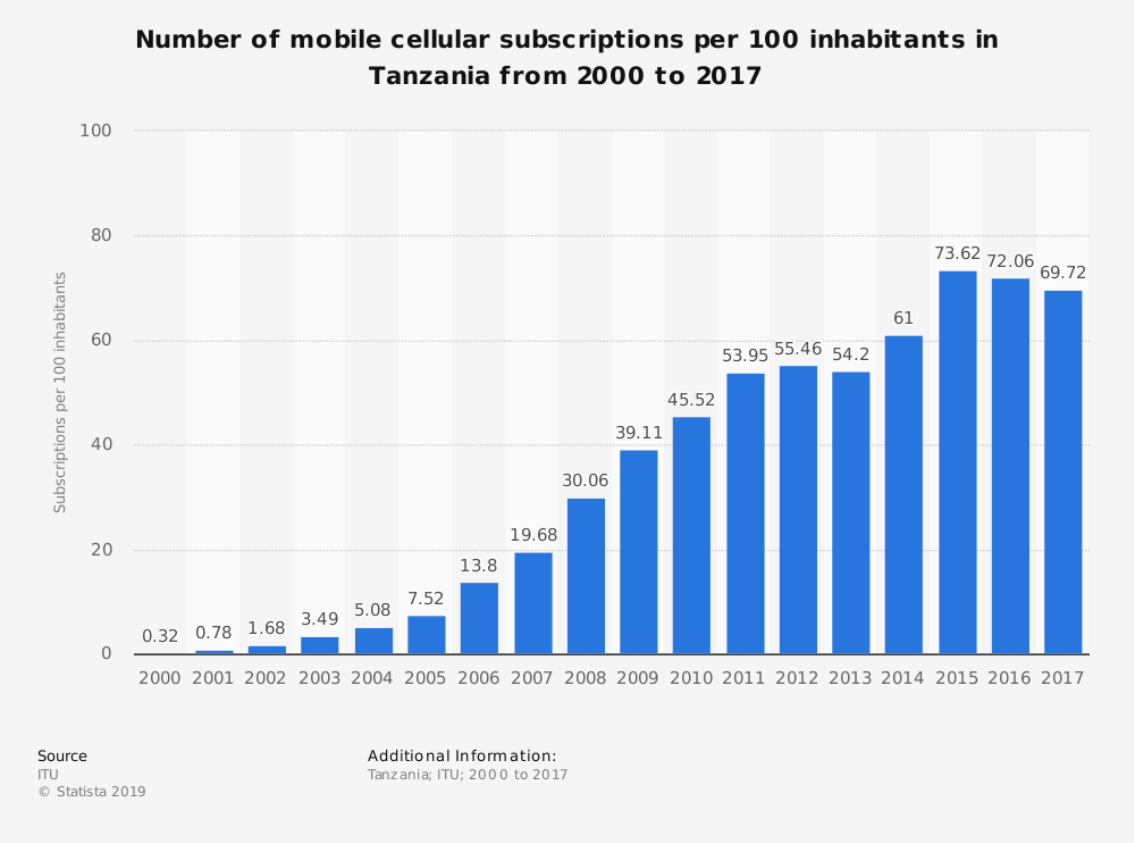

Figure 2 – Mobile phone adoption rate [4]

In the past years, WIT Software has been working and developing one of this solution that is M-Pesa. M-Pesa is a mobile banking solution owned by Vodafone[1] and launched initially for Safaricom[2] and Vodacom[3], the largest mobile network operators in Kenia and Tanzania. M-Pesa project name goes *M* for Mobile and *Pesa* is the Swahili word for money [5].

This mobile banking solution is an innovative payment service and alternative for the financial mainstream services for the Underbanked that allows them to transfer money securely [6]. "M-Pesa allows users to exchange cash for "e-float" on their phones, to send e-float to other cellular phone users, and to exchange e-float back into cash" [6]. By other words, this mobile banking solution allows its users to deposit, withdraw, transfer money, pay goods and services quickly with a mobile device.

[1] https://www.vodafone.pt/
[2] https://www.safaricom.co.ke/
[3] https://vm.co.mz/

Despite this innovative payment solution helps to fix many of its user's problems, there still exists a massive gap between M-Pesa users and the online market. Nowadays, online payment systems require that its users have a credit card or a bank account in order to be able to use them. M-Pesa users do not have any of them what is preventing them from being able to be part of the online market. The Online Payments platform is an online payment system prototype that aims to be a bridge to avoid this difference that M-Pesa users are subject to and include them in the online market.

The work depicted in this report produced a functional prototype of a payment service for M-Pesa. Specifically, the prototype comprises three main modules: i) Web Applications; ii) Backend and iii) some functionalities added to M-Pesa Android Mobile Application. Although M-Pesa currently is deployed in several countries, in this report, we solely focus on the Tanzanian Market that is our target for this demo.

Work on the prototype lasted the whole nine months of the internship that ran on the academic year of 2018/2019 from 7 October of 2018 to 7 July of 2019. Overall, the time spent might be divided into three very distinct stages: i) Research; ii) Development and iii) Refactoring. The research focused on understanding the possible user's necessities and how current existing solutions work and what are their architecture. The development stage centred on the creation of a prototype composed by all web applications, backend and add all functionalities to M-Pesa mobile application. Finally, the refactoring stage was performed within the remaining time of the internship. During this final stage, interfaces were improved, with the software prepared for production stage.

1.1. Motivation

The number of online stores is increasing as the E-Commerce impact and nowadays given the globalization phenomenon online shopping is not a luxury, but a necessity and is essential that everybody has that opportunity. However, to shop online, having a credit card or a bank account is a must and, as the Underbanked do not have any of these, they cannot do it.

With the emergence of mobile banking solutions like M-Pesa and the rapid growth of its number of users, there is an opportunity to create an online payment system that any online store can integrate into its payments options. This payment option would allow M-Pesa users to pay directly from their M-Pesa wallet and creating them the opportunity to shop online

and significantly improve their lives. The problem presented here posed as one of the main motivations to create and implement a prototype for an online payment platform for M-Pesa.

1.2. Main Goals and Contributions

This internship focused in creating a payment system platform to allow M-Pesa users to pay directly from their M-Pesa wallet on any website that integrates the developed platform Application Programming Interfaces (APIs) and provides it as one of the payment options.

The main contributions of this work are as follows:

- Development of the Online Payments frontend
- Development of the Online Payments backend
- Added the following functionalities to M-Pesa application:
    - Receive push notification informing about a new pending payment
    - List pending payments
    - See pending payment details
    - Refuse or pay a pending payment
- Submission of a scientific article, "Segurança no Spring Boot" in the journal Programar, by Rúben Pereira and Patrício Domingues, this can be found on Appendix A
- User experience (UX) / user interface (UI) refracturing based on inputs from the WIT design team

1.3. Host Institution

Figure 3 - WIT Software logo

The internship took place at Leiria's office of WIT Software, from October 2018 to July 2019, to see more details consult the Appendix B.

WIT Software, Figure 3 represents the company logo, is a software company that creates advanced solutions and white-label products for the mobile telecommunications industry [7]. This company has over 300 employees and has five offices in Portugal in Porto, Leiria, Lisbon, Coimbra (Headquarters) and Aveiro. This company was founded in 2001 and has vast experience and deep expertise in areas like mobile communications and network technologies.

1.4. Outline

This document is organized in sequential order. The concepts or discussions presented in each chapter may rely on concepts, ideas or explanations presented in the previous chapters. In addition to chapter one, this is divided into the following six chapters.

Chapter two presents the State of Art related to the current most popular and used online payment systems along with the concept of payment processor and his most common architecture.

Chapter three describes the M-Pesa universe. In particular, it describes in a more detailed way what is M-Pesa and the reasons for having several M-Pesa solutions.

Chapter four focuses on the implemented prototype architecture explaining the measures and decisions taken into consideration which led to the implemented architecture.

Chapter five details the implementation process of Online Payments platform prototype. Specifically, this chapter explains the methodology and how the implemented features work across all the web applications and backend.

Chapter six aims to explain the changes done to the Online Payments platform, namely to the UX/UI to approach it to what user is used to seeing and giving him the best experience. Also, there is described the architecture and features implemented in the M-Pesa mobile application.

Lastly, chapter seven concludes the document. It summarizes the developed work along with critical analysis of the developed solution and the process used to obtain the current prototype. This chapter also lists some possible future work for further improvement.

2. State of the art

Online shopping market is recent, but there are already a lot of different payment systems for different niches of markets and different types of platforms: web or mobile.

An electronic payment system or online payment system is a way of making transactions or paying for goods and services through an electronic medium, without the use of cheques or cash. Usually, these solutions gather all the necessary information and assure that the transaction is reliable and can be completed. As shown in Figure 4, online payments involve several entities that will be responsible for ensuring the success of the transaction. Between the online store's checkout step and the money reach the merchant is necessary to go through the payment gateway and the payment processor, and only then the banking institutions will approve the transactions.

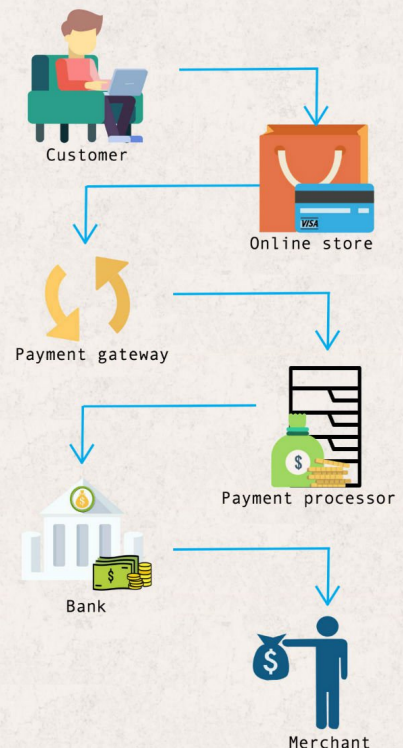

Figure 4 - Online payment process

The payment gateway is a software or a webpage where customers enter their payment details. This software checks the validity and encrypts the information entered by the user and determines the issuing bank and requests to the payment processor to initiate the transaction. There are several kinds of payment gateways, but the most popular is the hosted and the self-hosted. The hosted payment gateways are the ones which redirect the customer away from the website checkout page to the payment service provider page while the self-hosted payment gateway collects customer payment details directly from the merchant's website. Usually, this last kind of payment gateway uses a popup window that opens over the merchant's store.

The payment processor is the entity responsible for assuring that all the payment details have been submitted correctly from both sides and have a corresponding bank or financial institution and the issuer account has the fund at their disposal. This entity is also responsible for work as a mediator between the issuing and acquiring bank.

It is important to clarify that an online payment system does not necessarily need to have a payment processor built-in its architecture. There are several third-party payment processors

which have agreements between them to cover a width variety of banks. Also, there exists a considerable amount of online payment systems that only have a payment gateway and fall back on third-party payment processors to finish the transaction process.

The most recent services follow a trend to try to answer all user's needs. Nowadays, these solutions are equipped with a portal for each kind of user that usually is involved in online payments. Specifically, the users are Merchant, Developer and Customer. Each of these portals has their functionalities and tools and often, they are prepared with the means to satisfy the specific kind of user that it targets.

This chapter reviews online payment systems, focusing on what seemed the most important ones. The selection criteria were popularity and similarity with the M-Pesa system.

2.1. PayPal

Figure 5 - PayPal landing page

PayPal[8] is an online payment system that supports online money transfers. It is a Web solution and probably the most popular among all the selected ones, Figure 5 shows its landing page. It is also the best and the most relevant solution in the market. PayPal offers shoppers the possibility to link bank accounts, bank cards and to move money between their bank accounts and PayPal wallets. It also allows users to send and request money between them.

In case of merchants, PayPal offers all the mentioned features above, but also provides some useful tools that can be very handy for merchants like analytics, mass payment, calculators for shipping cost and sales tax and among others.

It also is prepared for giving developers all the assistance that they need. PayPal has a dedicated portal[4] only for developers that contains documentation and code samples to ease the implementation process. It also contains some practical examples where the developer can edit the code and see how it will change the result[5].

PayPal still has one crucial advantage over all the other competitors: it was the first online payment solution with direct integration to M-Pesa[6], although this integration is solely operational in Kenia, it already allows users to transfer money between PayPal and M-Pesa wallet and shop online paying from theirs PayPal wallet.

PayPal also handover a mobile application for shoppers[7] and another one for business[8]. Both of the applications are very well classified in the Google Play Store and have available the most useful and commonly needed features for each of the users.

[4] https://developer.paypal.com/
[5] https://demo.paypal.com
[6] https://www.paypal-mobilemoney.com/m-pesa
[7] https://play.google.com/store/apps/details?id=com.paypal.android.p2pmobile
[8] https://play.google.com/store/apps/details?id=com.paypal.merchant.client

2.1.1. Venmo

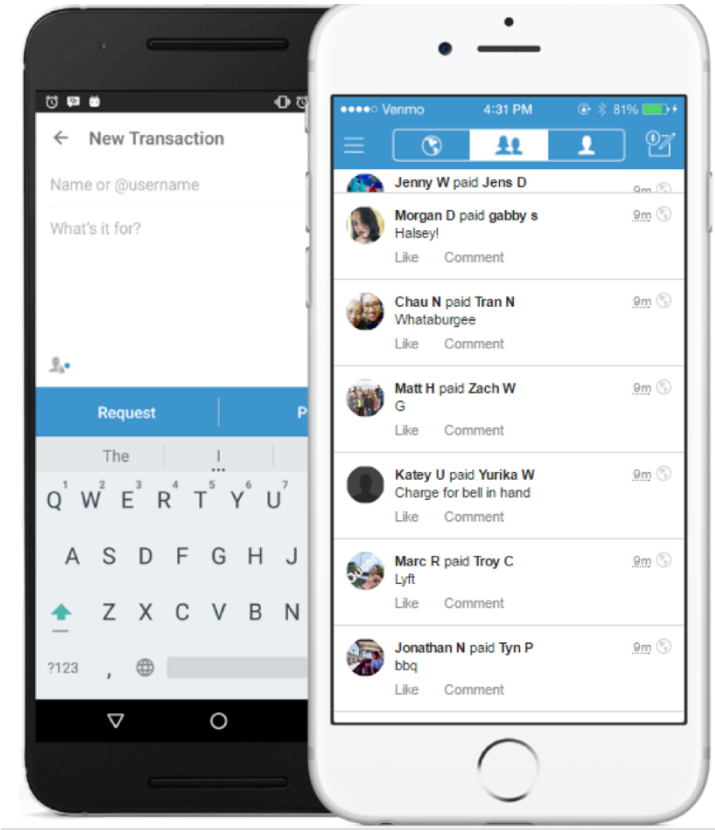

Figure 6 - Venmo mobile app

Venmo[9] is a mobile payment service and a subsidiary of PayPal. This payment service has a social component and includes social networking interaction that allows users to share information about what they just bought with their friends or even split bills between users.

Unlike the other system reviewed so far, Venmo does not appear to be prepared for the three types of users considered in this work. Indeed, Venmo only provides the mobile app[9], as seen in Figure 6, for users and a webpage for the developers[10]. Merchants have to use PayPal merchant page one time that Venmo payment option is integrated with PayPal checkout.

[9] https://play.google.com/store/apps/details?id=com.venmo
[10] http://developer.venmo.com/

2.2.Pesapal

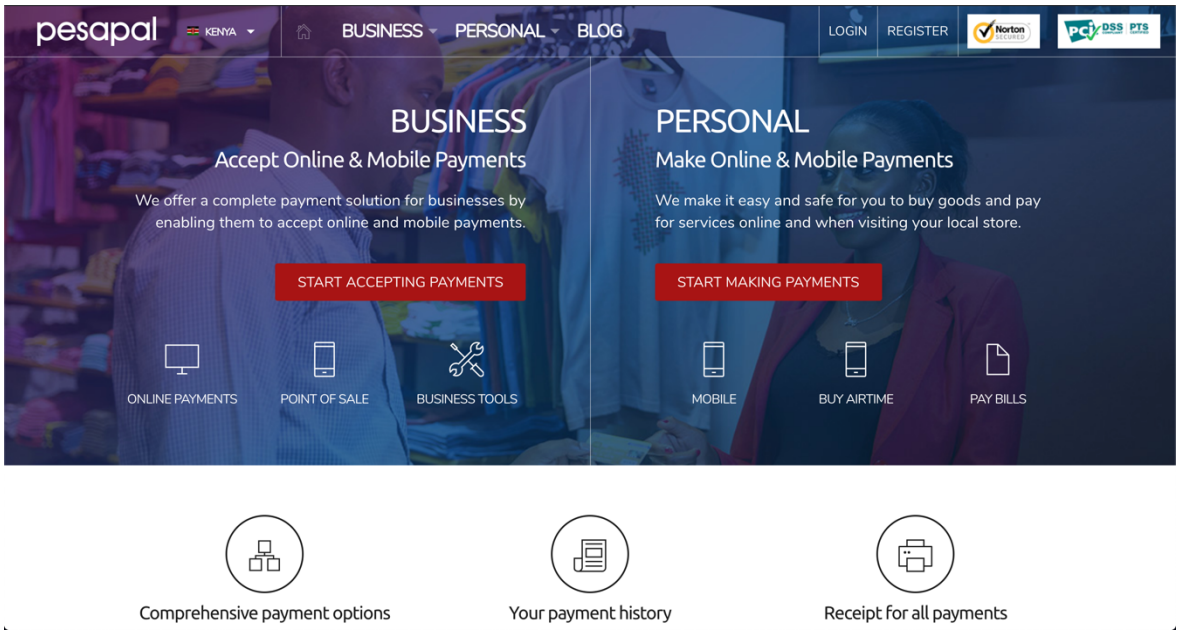

Figure 7 – Pesapal landing page

Pesapal[10], as can be seen in Figure 7, is a payment platform developed for allowing mobile banking users, like the ones who use M-Pesa, to do online shopping and perform online payments.

For the purpose of this survey, we could not experiment Pesapal, since this required a phone number from one of the supported countries to complete the registration. Nonetheless, due to its importance, was include Pesapal in this review.

Additionally, the checkout experience is really slow and confusing. Specifically, on checkout, Pesapal requests users to perform a transaction to them with the purchase amount and then fill a form with the buyer's mobile phone number and the transaction confirmation code. After Pesapal receives the form submission, they acknowledge the transaction through the confirmation code and then perform a transaction to the merchant store of the amount left after applying their taxes and commission.

Pesapal also provides a mobile application[11]. The application has some interesting features like book flights and holidays, buy event tickets and order food. Although, some of the features like pay bills, buy air time M-Pesa application already have it.

2.3.JumboPay

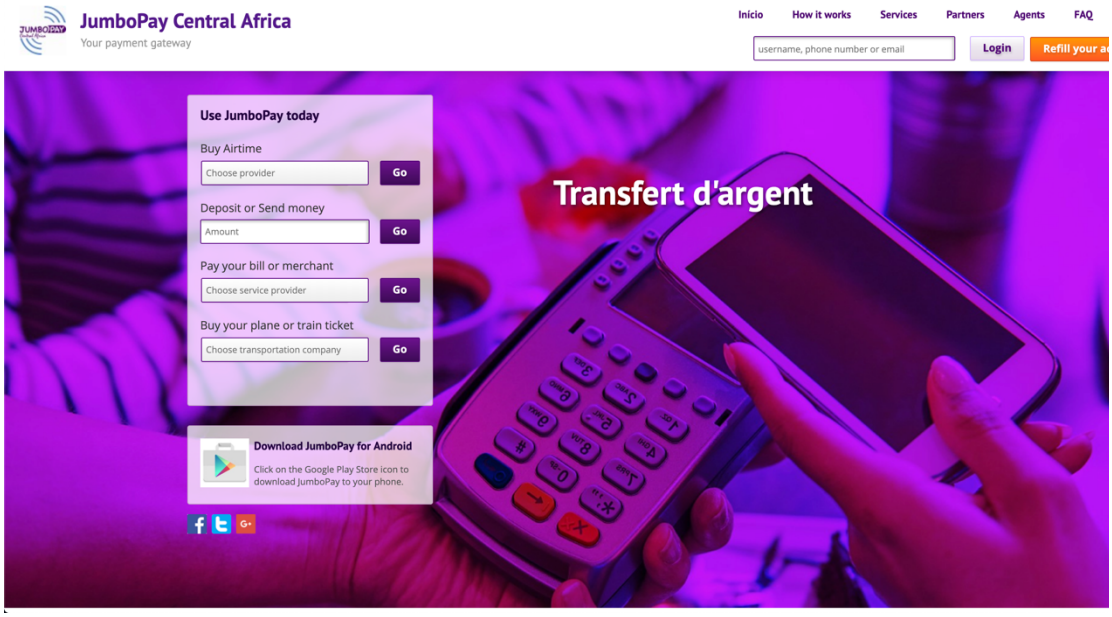

Figure 8 - JumboPay landing page

JumboPay[11] is also another payment platform, quite similar to Pesapal, both platforms processing the payments in the same way.

JumboPay also provides a portal for merchants and another one for buyers. Although it does not have a developer portal, it provides a web page with downloadable material regarding their APIs.

JumboPay's web interface, as can be seen in Figure 8, is one of its biggest fragilities because almost all of the navigation buttons do not work, and the web site provides few information regarding how the service works.

[11] https://play.google.com/store/apps/details?id=com.pesapal.mobile

2.4.Alipay

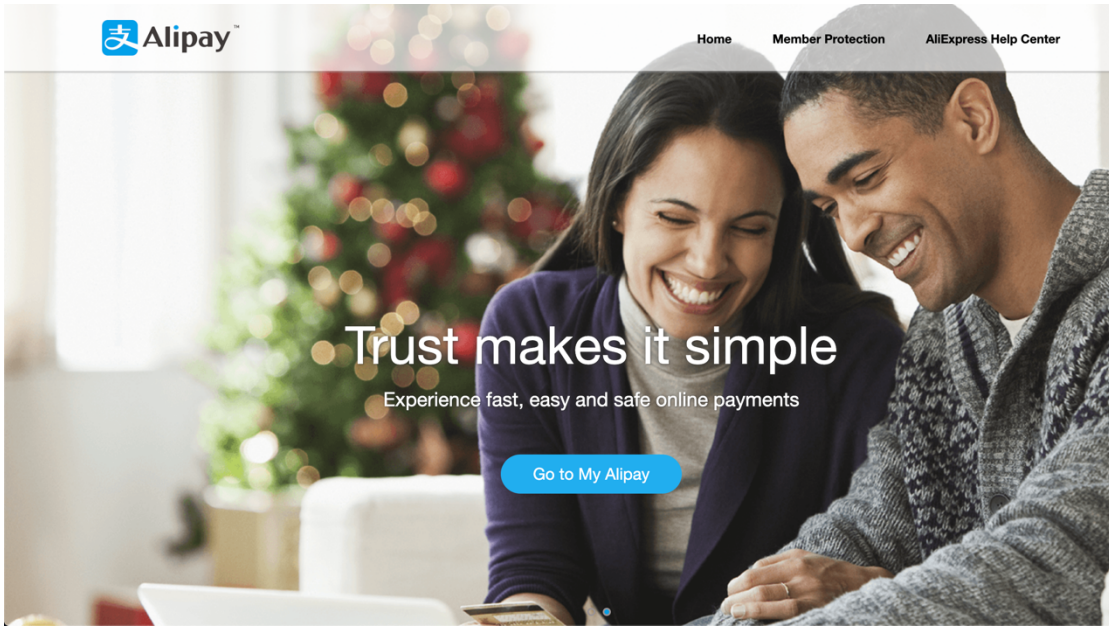

Figure 9 - Alipay landing page

Alipay[12] is regarded as the world's largest mobile payment platform[13], providing for nearly half of third-party online payments in China. Alipay is a third-party mobile and online payment platform. Alike to PayPal, Alipay provides three portals. In particular, as can be seen in Figure 9, shopper's portal[12] is basic, solely providing the functionality to link a credit card to perform payments and consult recent transactions.

We cannot elaborate on the merchant portal[13], since it requires an account linked to an existing store.

Developer's portal[14] solely contains documentation with some explanations but fail to provide code samples.

[12] https://intl.alipay.com/
[13] https://global.alipay.com/index.htm
[14] https://intl.alipay.com/doc/thirdpartyqr/zzo26b

2.5.Skrill

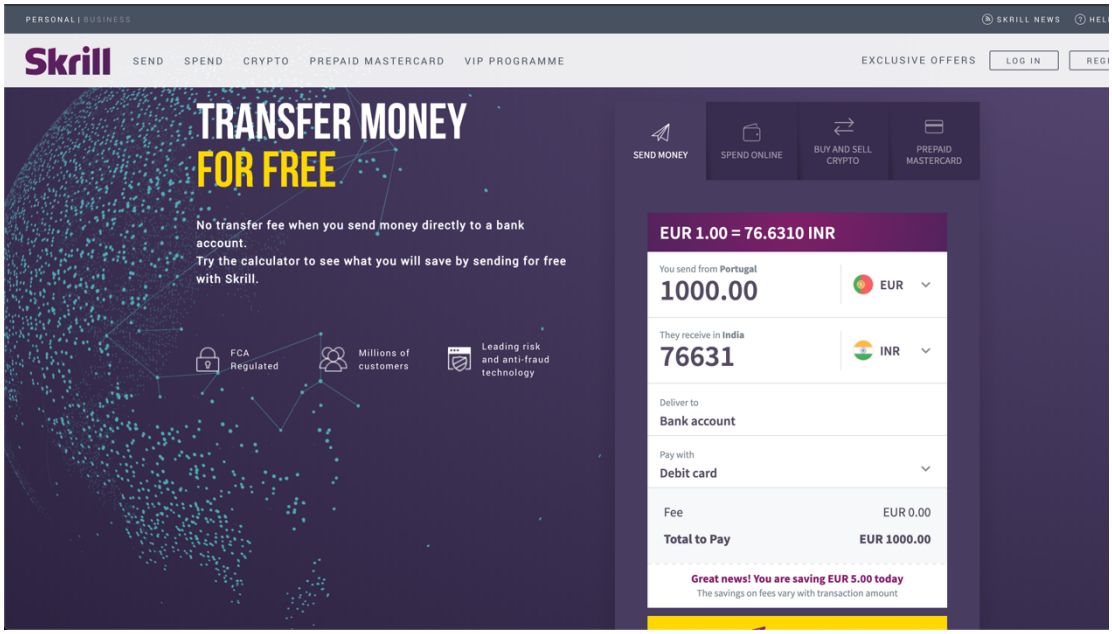

Figure 10 - Skrill landing page

Skrill[14], represented in Figure 10, is an online payment system that also allows transfer money. Skrill, like PayPal, is one of the most complete solutions for standard users and allow them to perform the same thing except request money to other users.

On the other hand, Skrill has no tools for merchants than the ones that they offer to regular users.

Developers' portal[15] only contains an option to download the guidelines that contain the documentation and code samples. They have not any demo portal to try its checkout experience.

[15] https://www.skrill.com/en/business/integration

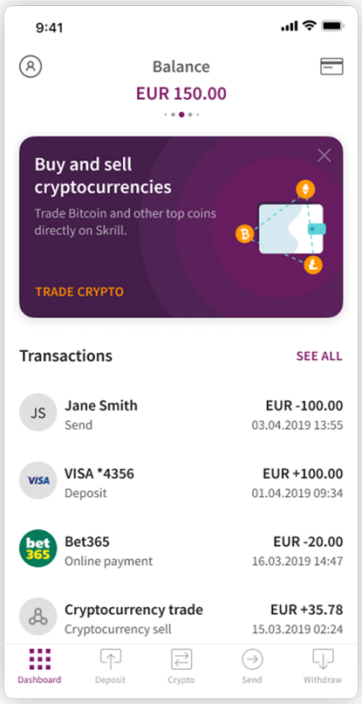

Figure 11 - Skrill application

Skrill also provides a mobile application[16], as shown in Figure 11. The application is focused on allowing users to manage Skrill wallet with standard functionalities like check the balance, see recent transactions, deposit and withdraw funds, send money and being notified when receiving money. The Android version is rated as 3.9 as of July 2019, with some reviewers complaining about the impossibility of withdrawing funds from their accounts.

[16] https://play.google.com/store/apps/details?id=com.moneybookers.skrillpayments

2.6. AmazonPay

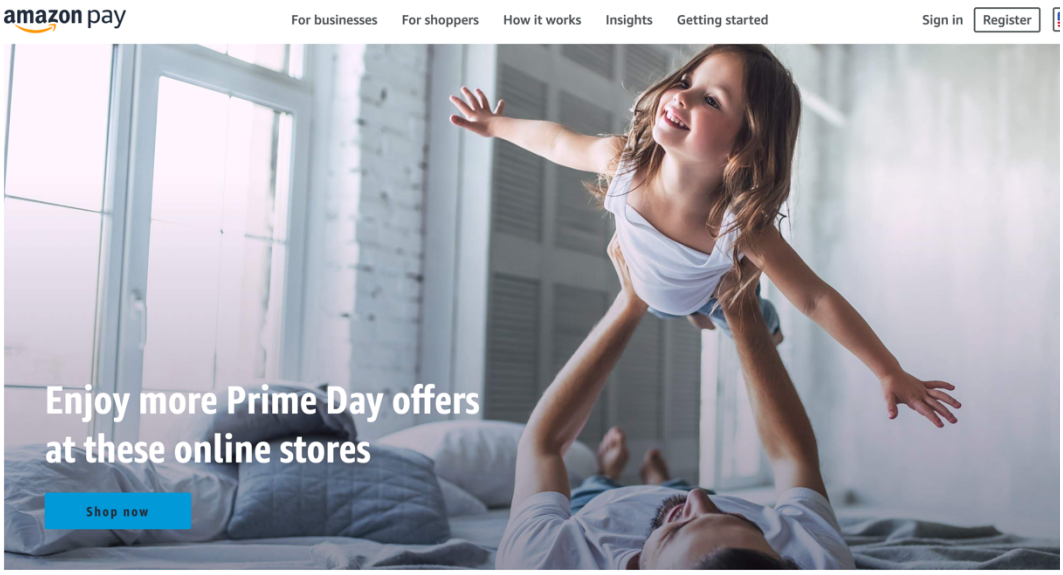

Figure 12 - AmazonPay landing page

AmazonPay[15] is an online payment service that is owned by Amazon. This payment service allows users to pay with their Amazon account on external merchant websites. The portal for normal[17] users, represented in Figure 12, is pretty simple, only containing information like a list of orders and agreements with merchants. It creates the bridge for paying in external websites from the wallet or linked credit cards in Amazon.

AmazonPay also has a merchant portal[18] integrated with Amazon online store, but it was not possible to interact with it since it required a valid business to complete the registration. In contrast, the developer portal[19] provides a wealth of documentation and code samples to exemplify and help on the implementation process.

[17] https://pay.amazon.com/
[18] https://sellercentral.amazon.com/
[19] https://developer.amazon.com/docs/amazon-pay/intro.html

2.7.Trustly

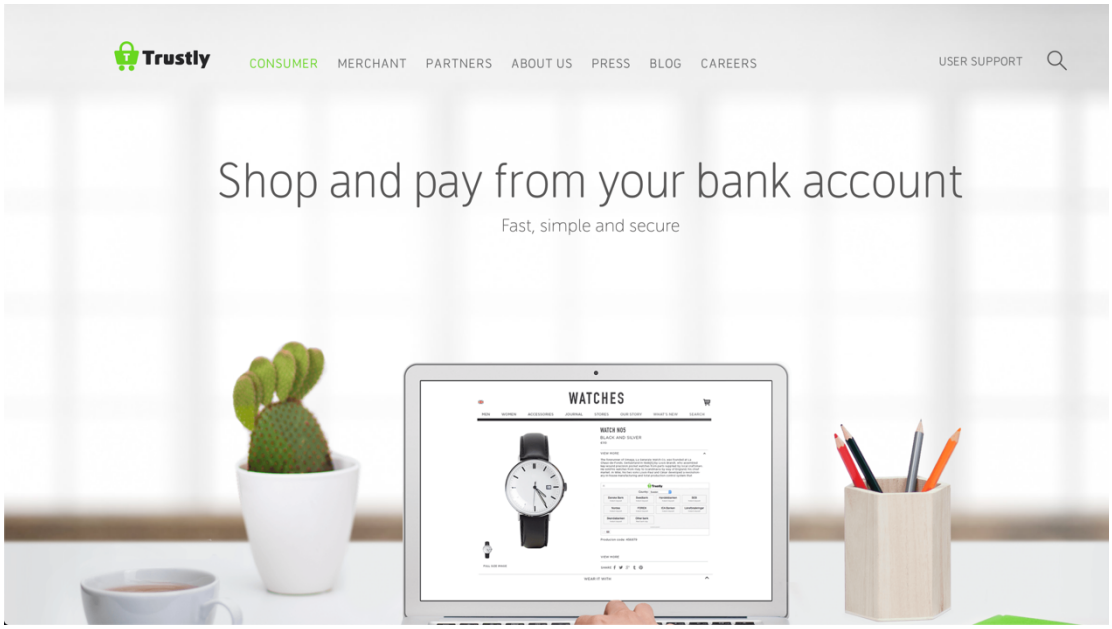

Figure 13 - Trustly landing page

Trustly[16] is an online banking ePayments solution that departs from the other payment platforms reviewed so far. Indeed, it has an entirely different approach, since the client logs into Trustly portal using her/his bank account credentials. At checkout, Trustly retrieves accounts information for the user to choose from which account pay. The Figure 13 shows its landing page.

Although Trustly has a backoffice portal for merchants[20], we cloud no check its functionalities because it was mandatory to contact the service support with a valid business to get access credentials. The developer's portal[21] offers access to documentation and code samples.

[20] https://backoffice.trustly.com/
[21] https://trustly.com/en/developer/api#/introduction

2.8.Stripe

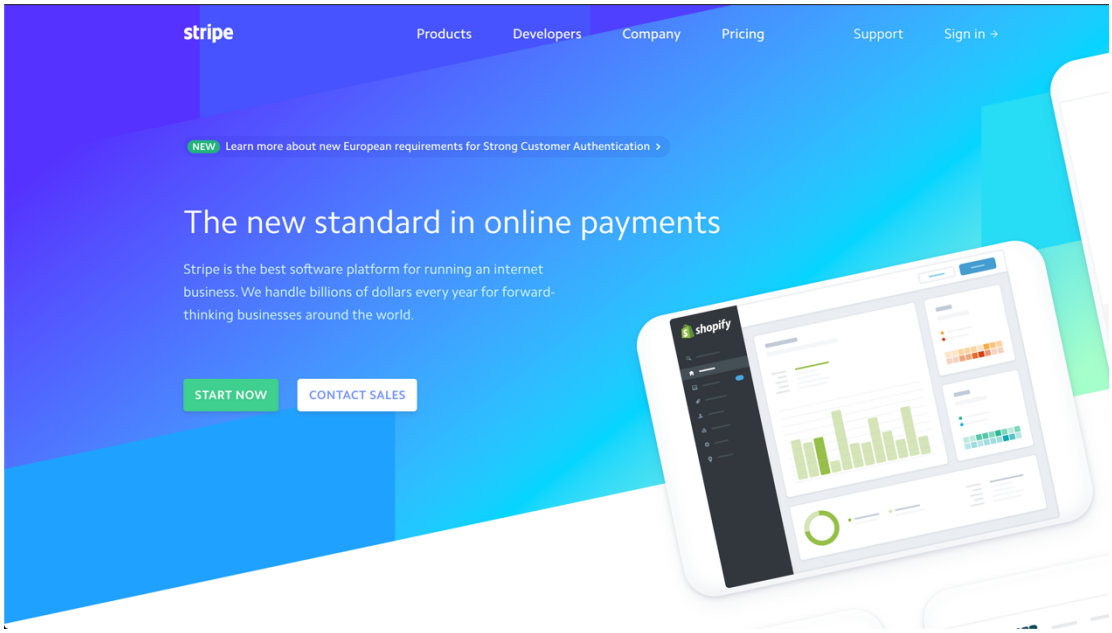

Figure 14 - Stripe landing page

Stripe[17] is a web online payment system that allows users to receive payments over the Internet, and Figure 14 shows its website landing page. This payment system is a specific platform for merchants and does not have a portal for non-merchant users. It is quite simple to use and has all the essentials tools to support an online business. Specifically, Stripe offers merchants the possibility to link bank cards and accounts and move money between bank accounts and Stripe wallets.

Regarding the developer's portal[22], it has plenty of documentation and provides samples that developers can change code and see the impact of the transformation. The developer portal also provides tools to diagnose the status of solutions created by developers.

[22] https://stripe.com/docs

2.9.WePay

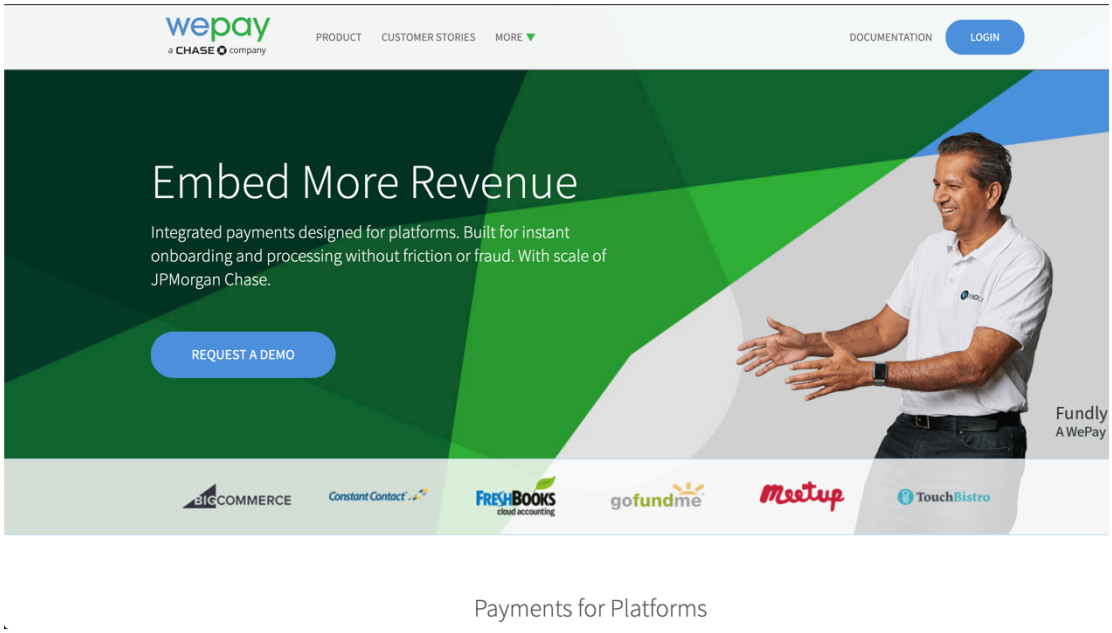

Figure 15 - WePay landing page

WePay[18] is an online payment service that provides an integrated and customizable payment solution through its APIs. Figure 15 represents the website landing page.

This solution is prepared for the three types of users, but it was not possible to check the merchant's portal. Although, the portal for buyers[23] was pretty simple and only allow the user to withdraw money from the website and link credit cards or bank accounts.

The developers had in their portal[24] a lot of documentation as code samples and had access to diagnostic API tools.

[23] https://www.wepay.com/login
[24] https://developer.wepay.com/

2.10. 2checkout

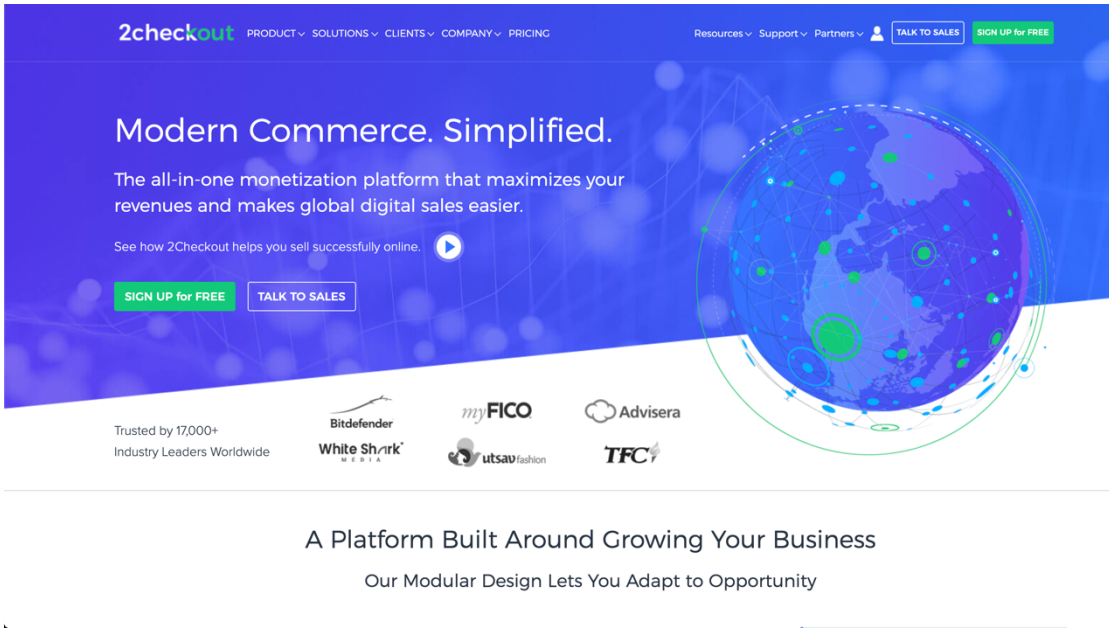

Figure 16 - 2checkout landing page

2checkout[19], in Figure 16, is a payment platform that allows companies to accept online and mobile payments from buyers worldwide.

This platform is similar to Stripe, only targeting merchants, and thus does not provide a shopper portal. The tools that they offer are limited: the merchant is only provided with a wallet, a list of items that the merchant is selling with a Customer Relationship Manager (CRM) System. 2checout also provides a list of recent transactions as a list with the orders made by merchant's customers.

Although it does not have the best merchant portal of the reviewed platforms, 2checkout has a good developer's portal[25], which provides appropriate documentation along with code samples. It also supports a diagnostic API and some other advanced features such as webhooks.

[25] https://knowledgecenter.2checkout.com/API-Integration

2.11. MBWay

Figure 17 - MBWay application across the several platforms

MBWay[20] is a mobile application based on an Automatic Teller Machine (ATM) network and, as can be seen in Figure 17 it is available in more than one platform. It allows users to perform transactions between them, as well as it allows users to purchase on online stores. MBWay is linked to the user's bank account through ATM. This service provides a mobile app that can generate virtual banking cards to shop online and pay e-services. MBWay allows web stores to integrate their API, with the shopper solely having to provide the phone number associated to the service and receiving back in the smartphone a push notification to confirm whether wants to accomplish the transaction.

The MBWay ecosystem has three main components: i) mobile application for shopper[26]; ii) portal for merchant and iii) portal for developer[27]. The developer's portal provides documentation to implement access to the service and also on how to use the sandbox environment.

We could no assess the merchant portal since access requires a real business.

2.12. Summary

In summary, there are a wealth of mobile-oriented payment platforms. A significant number of these platforms provide useful functionalities. However, many of the platforms were clearly built targeting shoppers that have access to regular bank and financial services.

From all the reviewed solutions, PayPal is probably the most complete one. It is also the only one that appears to threaten the M-Pesa purchase online ecosystem.

Overall and after comparing all platform functionalities, comparison summary in Appendix C, it was possible to notice that most of the reviewed solutions already provide a portal for each type of user. Specifically, a merchant portal focuses on helping merchants to keep track of sales and manage their business. Developer's portal, on the other hand, mostly focuses on providing documentation to explain how to integrate the payment gateways Software Development Kit (SDK) on checkout. Some of the solutions provided dedicated tools to turn their SDK implementation easier and, in some cases, there was even a sandbox environment to test the integration and keep sure that everything was working fine before going live.

Lastly but also important, the checkout portal in most of the solutions provides for simple and hassle-free checkout experience.

Based on analysed solutions, was arranged a list of the most relevant and possibly useful features. Table 1 lists these features grouped by roles.

Table 1 – Selected features

Role	Features

[26] https://play.google.com/store/apps/details?id=pt.sibs.android.mbway
[27] https://www.mbway.pt/developers/

Seller	• Virtual Wallet • Send Money • Request Money (e.g. Create/Send an invoice) • Transaction Summary • List of Items for Invoice • Address Book • Analytics/Reports • Mass Payment • Shipping Calculator • Sales Tax • List of Orders • Refund Order • Webhooks
Developer	• Documentation • Sandbox Environment • Code Samples • API Logs • List of Apps & Credentials • Webhooks
Shopper	• List the Purchase Products • Shopper Authentication

3. M-Pesa Universe

M-Pesa concept was born in the mid-2000s in Kenya and was encouraged and developed by Safaricom, Vodafone's associate in Kenya. The main goal of this project was to develop services to help the underbanked. Originally, the idea was to provide microfinance loans using mobile phones. However, it soon was found in practice that people who received the loans were sending the money to other people hundreds of miles away. In Kenya, it is usual for breadwinners to travel to urban areas for work while family stays behind. This create the need for sending money from the breadwinner to their families. Since, often these people did not have access to proper financial services, M-Pesa becomes a solution to solve the problem of sending money back to their homes. This way, the loan systems was reengineered to provide for directly sending money between phones. This give birth to a huge network of agents hoping that M-Pesa customers would visit those agents with cash, then, through an Unstructured Supplementary Service Data (USSD) menu and series of Short Message Service (SMS) messages, convert that money into electronic funds that securely sat on their Subscriber Identity Module (SIM) cards. Then, via SMS, they could transfer that money to anyone else, who would then visit their local agent and withdraw that cash again [21].

Figure 18 – Withdraw money at early begin of M-Pesa

The M-Pesa project has now almost 20 years and has never stopped growing. Interface-wise, the project evolved from a small menu that could fit in a 32Kb storage of SIM card and sent money over SMS, as in Figure 18, to what we can found nowadays, a mobile application for smartphones (Android and IOs), as shown in Figure 19.

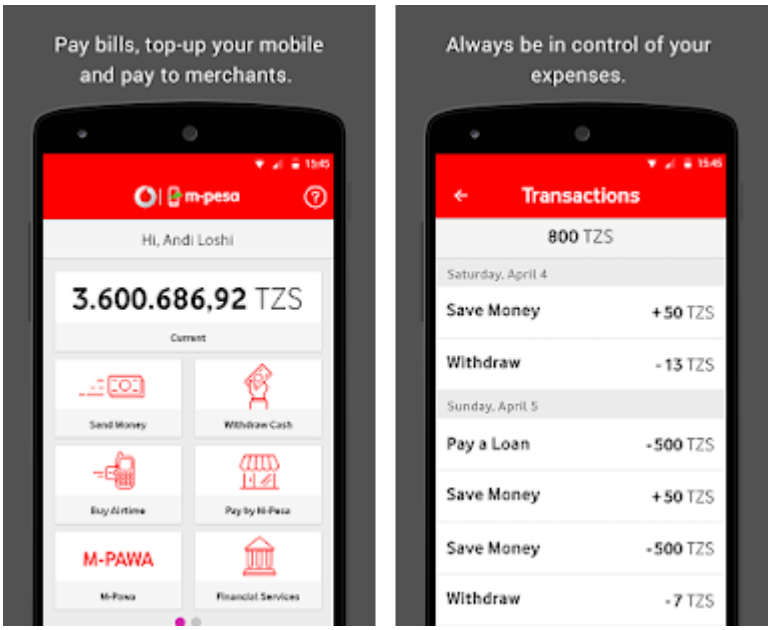

Figure 19- Current M-Pesa application interface

M-Pesa intellectual rights belong to the Vodafone Group. They keep investing in the evolution of the smartphone application, although one of their associates, Safaricom in Kenya, kept using USSD menus and investing instead in the development of new features that suited best their customer's needs. As a reflection of Vodafone Group and Safaricom taking different paths, nowadays, there are two different M-Pesa solutions, although both solutions have the same concept and similar functionalities. The solution logos can be seen in Figure 20, M-Pesa by Vodafone Group[28] and M-Pesa Kenya[29].

[28] https://vodacom.co.tz/en/getting_started_mpesa/
[29] https://www.safaricom.co.ke/personal/m-pesa

Figure 20 - M-Pesa solution logos

M-Pesa Kenya keeps investing in new functionalities which made them have features that M-Pesa by Vodafone Group has no support yet. One of the features that most distinguishes these solutions is that M-Pesa Kenya invested in the development of a developer portal for the creation of an open API[30] to allow third-party to develop their solutions. This way, online stores can implement this API and have M-Pesa as a payment option at checkout. As a result of this investment, M-Pesa Kenya already have is first partner, AliExpress, that offers the possibility to pay orders at checkout directly from user's M-Pesa wallet [22].

Not only AliExpress but also PayPal, as mentioned before, take advantage of this open API and integrated it in their systems to allow M-Pesa Kenya users to exchange money between their M-Pesa wallet and PayPal account [23].

All the work developed during the internship posed to create a demonstration of a working solution to introduce to Vodafone ground and how M-Pesa Kenya could stop using USSD menus and start using a mobile application.

[30] https://developer.safaricom.co.ke

4. Architecture

In this chapter, technologies are review as well as the architecture used to implement the solution approached in this report. As mentioned earlier, the platform Online Payments is an online payment system for M-Pesa project that will be integrated on the current M-Pesa project architecture, as shown in Figure 21. The main objective of this platform is to allow M-Pesa users to purchase from online stores and pay directly from their M-Pesa wallet.

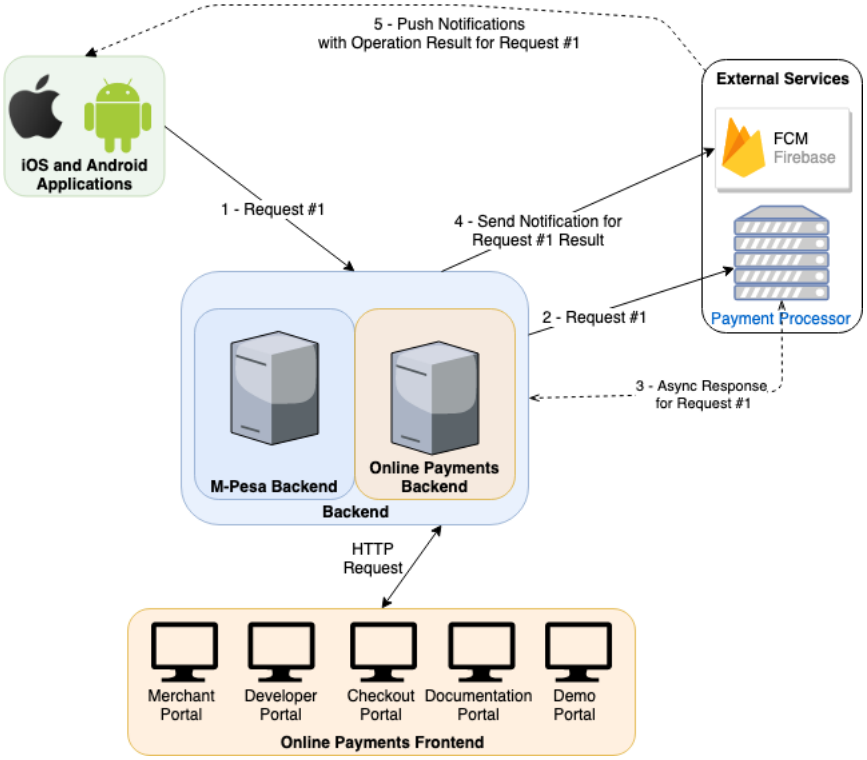

Figure 21 – Final solution architecture

As mentioned previously, to archive an online payment from any online store, the system requires a payment gateway and a payment processor. The M-Pesa application already uses a third-party payment processor, so the missing key of the puzzle is a payment gateway.

To create the intended system and deliver an uncomplicated and straightforward checkout experience, a checkout portal was created. The checkout portal works as the payment gateway. Online stores can use this portal to redirect M-Pesa users at payment moment to authenticate themselves and confirm their intention of purchasing the order.

Despite appearing to be a simple solution, this platform needs to support various kinds of users besides the shopper. Indeed, the M-Pesa payment option at checkout requires a

developer and documentation portal. Both portals eases the integration with any online store and helps developers to monitor and tests their implementation before going live. Furthermore, it is necessary to have someone in charge of selling things and receiving the order placements. For this reason, the system also needs a merchant portal to help the business owner to manage the business and to keep track of the online orders.

The platform needs to be prepared to satisfy user's needs. For thus and for that is part of the solution five web applications and a multi-module backend server with a Representational State Transfer (REST) web service. This web service deals with all the actions performed in web applications and insure the safety of redirected users from online stores to the checkout portal.

4.1. Technologies

The technologies to develop the proposed solution were carefully chosen to keep up in accordance with all the previous work carried by M-Pesa development team and also to simplify the integration of this solution with the current M-Pesa backend.

Given this, the chosen technologies are listed in Table 2:

Table 2 - Used technologies

Usage	Technology
Frontend	Angular[24]
	React[25]
Backend	Spring Boot[26]
Database	Oracle[27]
	Hazelcast[28]
Mobile Application	Android[29]
Deployment	Jenkins[30]

Among the technologies listed in Table 2, most of them are very well-known, except for Hazelcast and Spring Boot. Next, we provide a brief summary of these two technologies.

Hazelcast is an open-source in-memory data grid based on Java. It allows to store data in-memory and share it between different nodes of a cluster, and therefore contributes to reduce the query load on the database (DB) and improve the fetching speed.

Spring Boot is an open-source Java-based framework to create standalone applications. This framework is built on top of Spring framework and provides a simple and faster way to set up, configure and run developed applications. Spring Boot, unlike of Spring, does smart management of dependencies and avoid the necessity of setting all the configuration files.

4.2.Frontend

The Online Payments frontend has been developed using the Angular framework. Since the beginning, one of the main problems has been the number of portals required for the whole solution. The need of a portal for: i) merchant; ii) developer; iii) checkout; iv) documentation and v) demo, where each one has its specific UI would lead to the creation of five different Angular projects. This would contribute to the duplication of code and components that could be reused among the portals. Also, the deployment would be significantly harder and longer than if all of them were fit in a single project.

To address this issue, we resorted to an architecture that combines multiple Angular applications into one. The chosen architecture allows multiple projects in the root of a parent project and each one with its UI [31]. This is possible recurring to loadChildren property of Angular router. This property is part of Angular mechanism for lazy loading modules and loads routes from the submodules on demand. Figure 22 shows the frontend project structure with all the portals inside the same project.

Notwithstanding the number of web applications fit in a single project, the usage of lazy loading on the router gives the user the same experience as the projects were split. The interaction with the portal is smooth, with only the needed submodules being loaded.

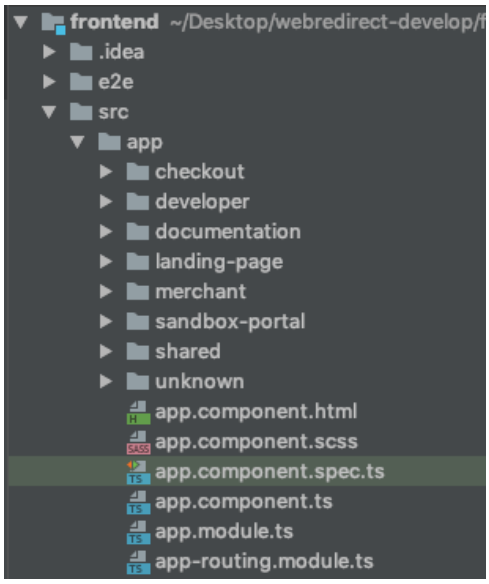

Figure 22 - Frontend project structure

The communication between the frontend and the backend is performed using Hypertext Transfer Protocol (HTTP) requests. The backend has a security layer that filters HTTP requests, on protected resources, and searches for the authorization token in the header of the request. This token is created at user login and its function is to authenticate itself.

The requests sourced from the portals go through the HTTPInterceptor and is added the authentication token of the logged user to the request header. After this, the request goes through a proxy that redirects it to the proper backend module. This proxy is configured on the root of the project in order to redirect each request from its web application to the correspondent backend module, as can be seen in Figure 23.

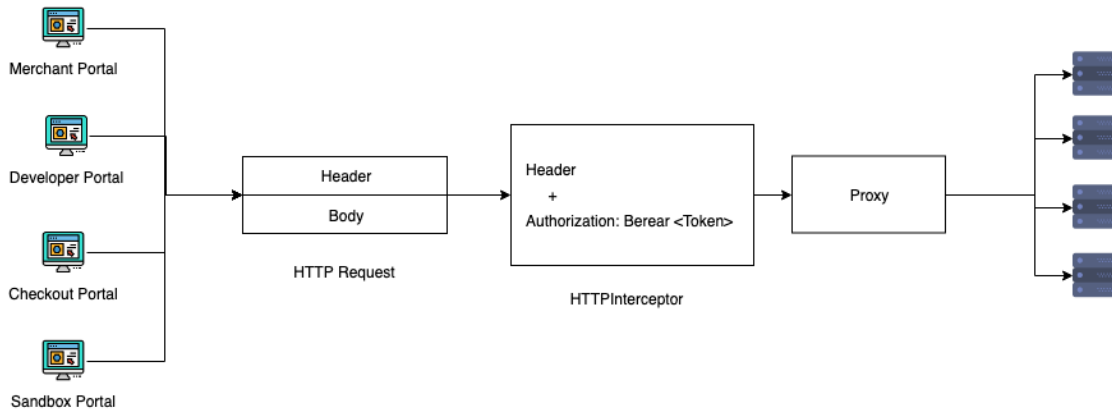

Figure 23 - Request path from the portals to the backend

To keep everything together and to avoid breaking up information, a landing page is provided. The landing page helps to connect all portals and allow users to move between them.

Each of the web applications follows the Feature Services Pattern.[31] This pattern allows to organize the code and to keep a certain level of abstraction. This helps to improve the application. In practice, one service is imported once for each feature, with all logic kept, for a given feature, in the associated service. This pattern allows to organize the project as documented in Figure 24. Additionally, data are only fetched and stored when needed.

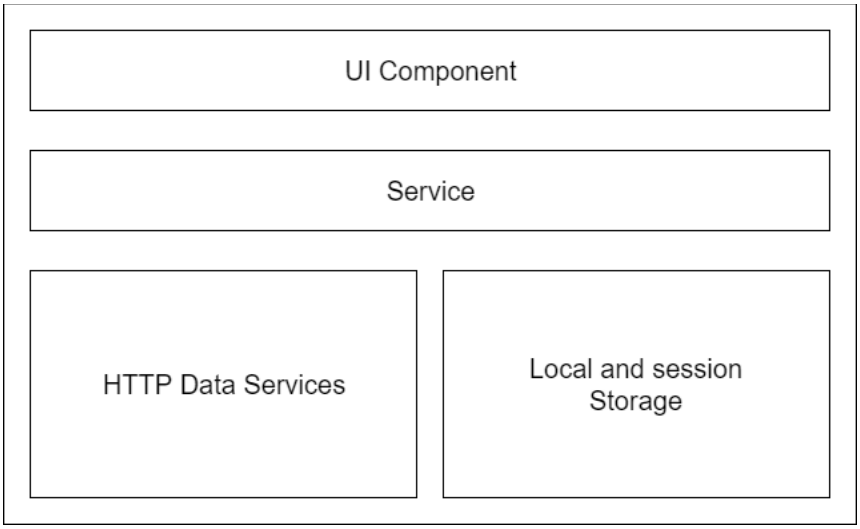

Figure 24 - Individual application architecture

On Angular, each frontend page is comprised of three things: i) component class ii) Hypertext Markup Language (HTML) template and iii) component-specific stylesheet or a Cascading Style Sheets (CSS). The component class is responsible for focusing on

[31] https://coryrylan.com/blog/angular-design-patterns-feature-services

presenting data and delegate data access to a service. According to the Angular documentation[32], the service is the class that should be responsible for fetching data and deal with its storage operations.

The feature-service pattern states that for each feature, all the HTTP request logic as well as the storage operations related to it, should be written in a service and then injected into the necessary component classes. This pattern helps to keep the code organized, well-structured and avoid code duplication.

4.3. Backend

The backend has been developed using Spring Boot. As stated earlier, Spring Boot is a Java-Based framework for creating standalone applications. The existence of different portals and the fact that they are independent between them in terms of functionalities led to the creation of a multi-modular backend. Spring Boot gives the possibility of creating a multi-module project that is a solution distributed over several modules. The advantage of the usage of a multi-module project is when a specific module needs to have new functionality or bug fix, it is possible to focus on that specific module only. Also, in a production scenario, it will not be necessary to stop the whole platform to perform a deploy, since in this approach, each module is independent and thus can be deployed/stopped without interfering with the other ones. This approach allows creating modules that may behave either as an application or a library, and which other modules may import in their dependencies. In a multi-module project, there is a parent Project Object Model (POM) file that works as an aggregator and is where the application modules are declared as the common dependencies across the children modules.

As can be seen in Figure 25, the backend consists of six modules: i) merchant; ii) developer; iii) checkout; iv) sandbox; v) mobile and vi) core.

[32] https://angular.io/tutorial/toh-pt4

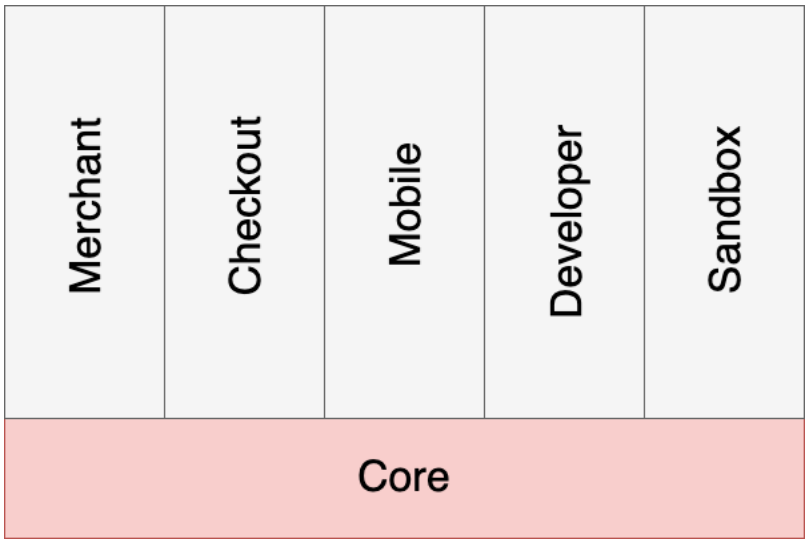

Figure 25 - Backend modules

Each of the modules earlier presented is responsible for its web or mobile application. The merchant module is responsible for merchants' authentication as ensure that they are part of the business that they are trying to access. This module deals with all the requests that come from the merchant portal. Just like the merchant module, the developer module is in charge of the requests that came from the developer portal. On the other hand, the checkout module is responsible for all the checkout process as also generate tokens to identify an online store checkout session. The sandbox module is a playground for developers. This module has the same endpoints that checkout module, but in this scenario, they work with mocks to test if the solutions that developer are implementing are well structured. However, this module also holds the backend of the demo store where developers can use it to test and see the checkout experience. The mobile module is responsible for all mobile applications requests. The core module is a library and imported by all the other modules. It is used to keep the business logic that is required in more than one module. This module also keeps all models and classes that are present in more than one module. This allows to avoid code repetition and creating a higher level of abstraction. Also, the core keeps all the services as the Firebase Cloud Messaging (FCM), SMS and Email Service to allow the other modules to use these services.

All the five modules have the core module as a dependency, and all of them follow the same architecture, as shown in Figure 26. The layers that build the modules are: i) Security; ii) Resources; iii) Service; iv) Domain; v) Repository; vi) Data Mapping and vii) Infrastructure.

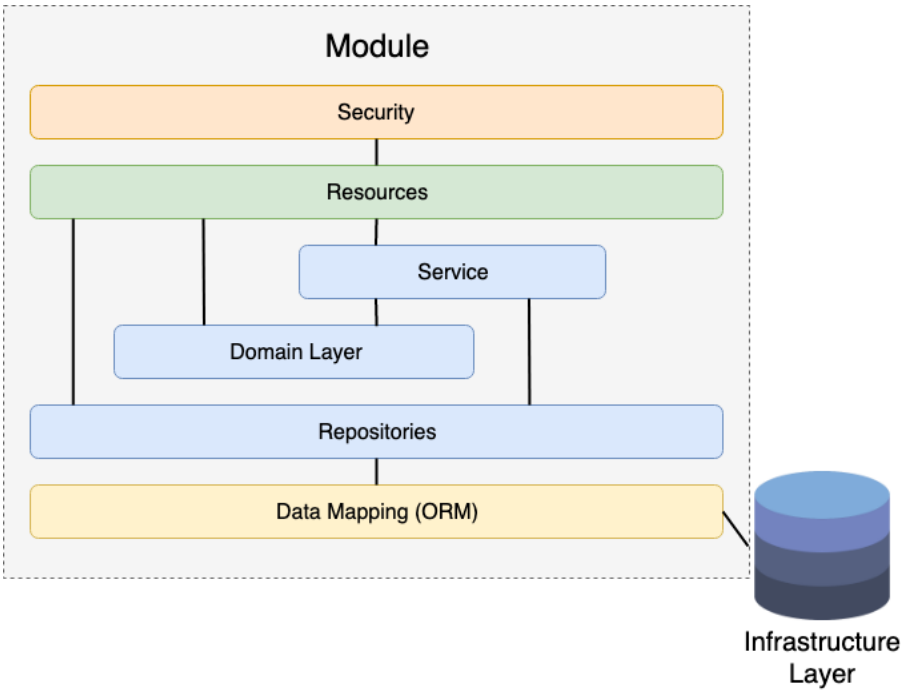

Figure 26 – Module layer architecture

Each of the layers presented before has a specific function that is:

- **Security Layer**: This layer deals with user authentication and authorization to access protected resources.
- **Resource Layer**: This layer holds the REST controllers and where the endpoints are declared.
- **Service Layer**: This layer retains the business logic and establishes a connection to the database.
- **Domain Layer**: The domain layer represents the mapping layer. This layer is where the inner domain classes are converted to the output format that is sent to the client applications.
- **Repositories Layer**: The Repositories are powered by String Data which means that basic Create, Read, Update and Delete (CRUD) operations are supported out of the box. Spring Boot is also capable of translate method names into queries.
- **Data Mapping Layer**: The Mapping Layer contains all plain annotated classes and their Object-Relational Mapping (ORM) relations.
- **Infrastructure Layer**: This layer represents the storage mechanisms used to persist data.

4.3.1. Data Persistence

The type of database to use is one of the most important decisions, since in a situation with multiple users, it impacts on the solution performance. At an early step, a sketch was designed and created on how the platform entities relationship could be. It was essential to decide between using a relational database or a non-relation database. This sketch led to the conclusion that entities would have many relations between them and, thus the best option would be to resort to a relational database. As can be verified in Figure 27, it is easy to understand that data persisted is highly relational.

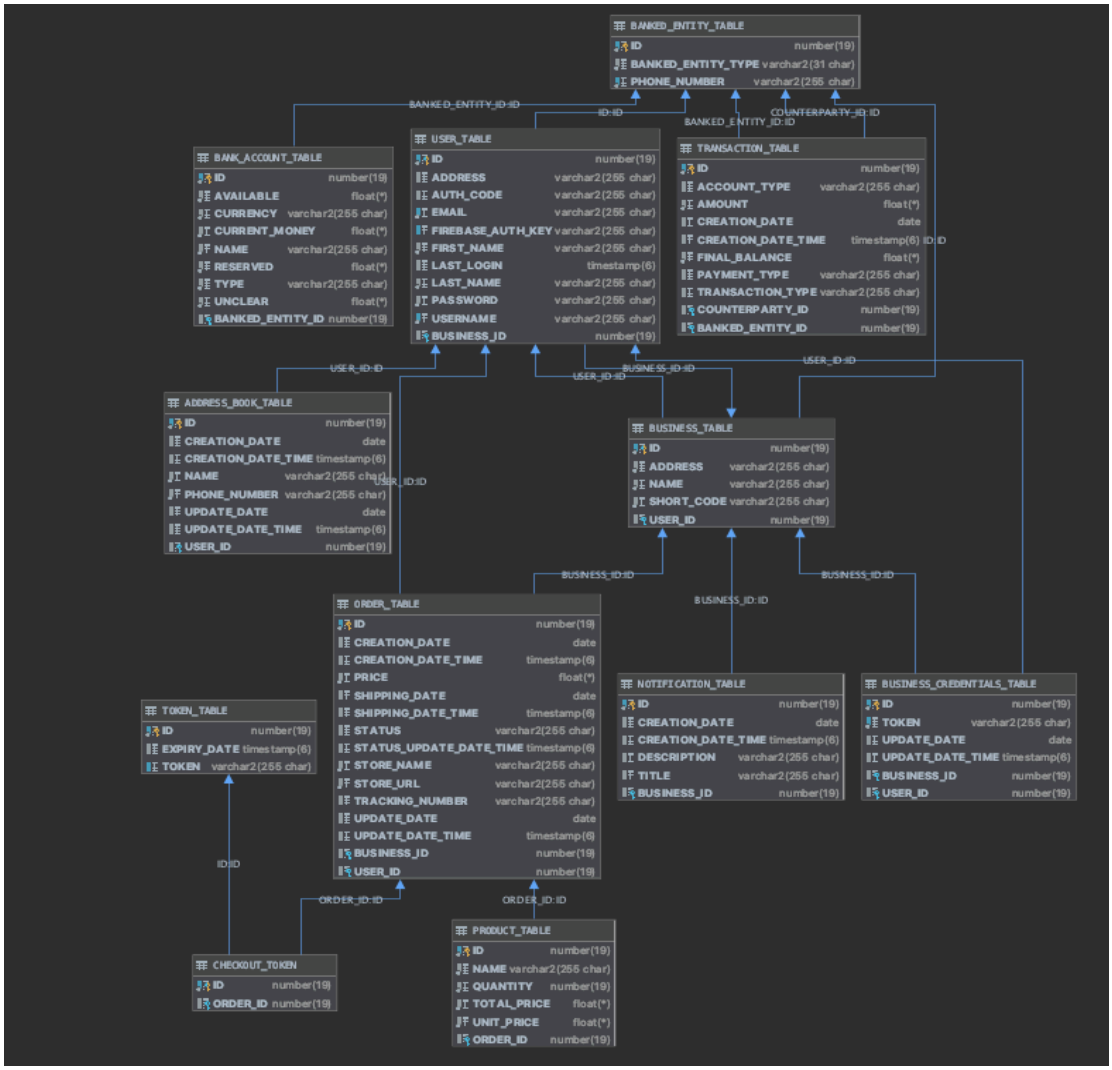

Figure 27 – Merchant entity relationship diagram

The creation of the demo portal along the demo store originated a source of dummy data that should not mix with the one generated by the rest of the platform. To avoid data contamination, the database has a separate schema used to keep the data generated from the demo portal.

Another problem that was addressed, is scalability. Indeed, this platform in a scenario with a high number of concurrent users performing requests would generate a high query load on the database. So, after a careful analysis, it was noticeable that some of the stored data were shared across several modules and often, the same data would be queried in a short space of time from the database. As a mitigation and proposal to solve this problem, we resorted to Hazelcast that stores the data in-memory in a HashMap and thus allow to share that data across the several modules. The usage of Hazelcast allows to reduce the number of queries to the database and increase the speed of fetching data, effectively reducing the response time.

4.4. Deployment Environment

The Online Payments platform was deployed to a virtual machine (VM) running CentOS[33] in Amazon Web Service (AWS). It was necessary to do all the VM setup because the VM only had the default packages installed. The project was deployed using NGINX[34], that is an HTTP Server and works as a proxy on port 80, redirecting the traffic based on the context path. Additionally, an Oracle DB was installed to persist data.

In this stage, Spring Boot provided the leverage, dispensing the use of a java web server to deploy the backend. Indeed, Spring Boot web dependencies already have a built-in Tomcat[35].

[33] https://www.centos.org/
[34] https://www.nginx.com/
[35] http://tomcat.apache.org/

5. Implementation

The implementation stage was long. It also needed several preparation steps before the effective start of the implementation. The starting point was market research to understand the needs and how similar solutions work. This study allowed to gather a list of requirements and create the user stories, Appendix D, to guide all the work. Besides the feature gather, we also performed a UX/UI analysis to understand how usually these solutions are in terms of user experience.

As we shall in this chapter, we relied on a derived agile methodology. All the steps were carefully planned and analyzed before the start of every sprint. The implementation process was supervised and followed by the internship mentor along with the company Staff to assure that it did not lose the track.

5.1. Methodology

During the internship, all the work developed was under agile methodology. It had nine months length and was organized in sprints of one month, hence giving a total of nine sprints. A meeting to review the work done and discuss next steps were performed at the end of every month. The development followed the GitFlow[36] rules that ensure that a stable version of the project ready to be shown always exists. Whenever a new feature was started, a new branch was specifically launched and merged to the development branch when the feature was considered complete. This way, the master branch was solely used by the maintainer of the repository to release new versions of the project in specific points of the internship.

The Redmine[37] tool was used for project management and issue tracking. The usage of this tool had the advantage to keep all the work log and time spent in each task as the bugs tracking. Figure 28 shows all the tasks carried out distributed over time with the respective sprint date.

[36] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
[37] https://www.redmine.org/

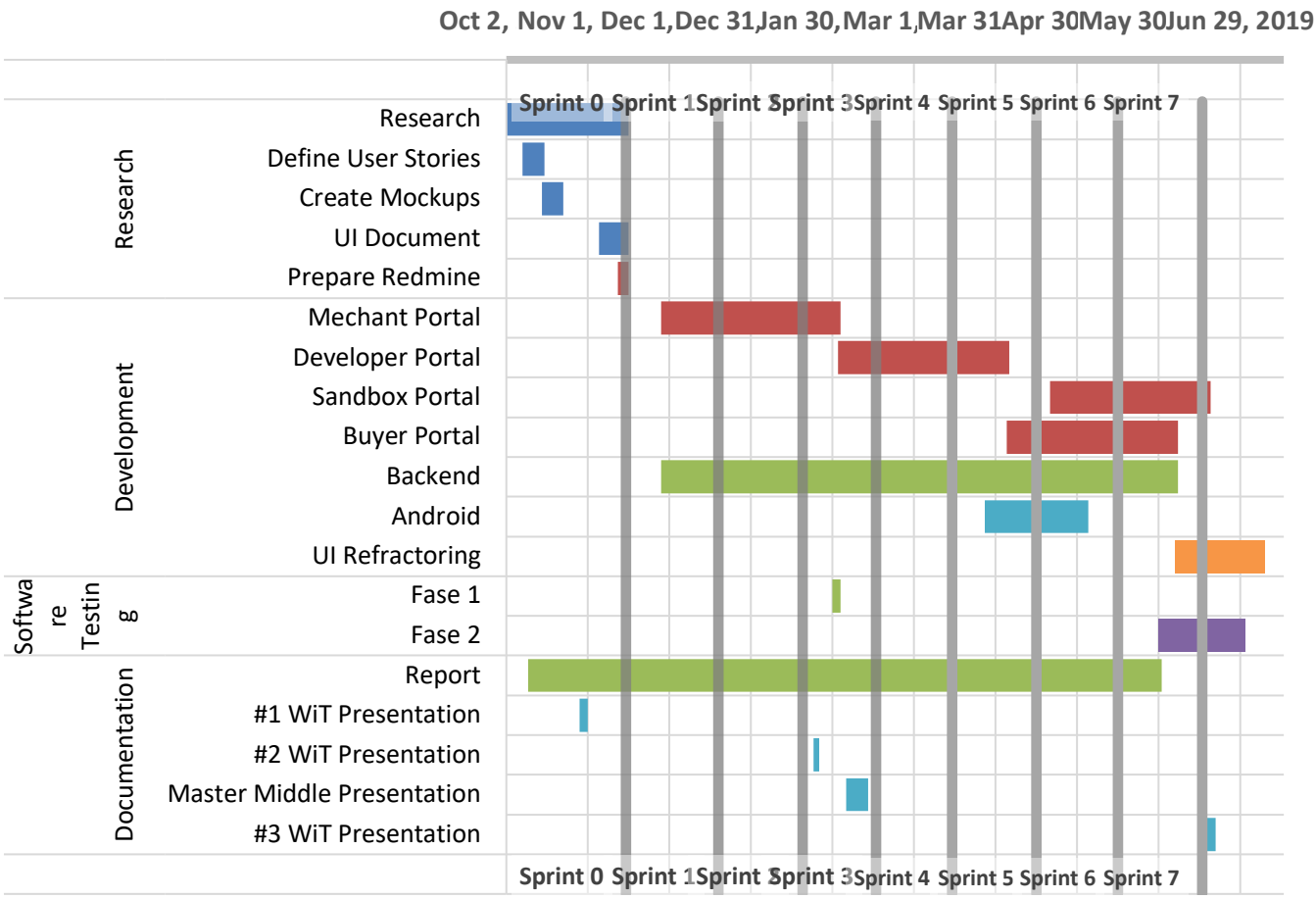

Figure 28 - Internship timeline

During the internship, WIT Software required several presentations on distinct points. These presentations gathered all the company trainees and internship mentors along with some of the WIT Staff to discuss the work carried on till the moment. Usually, these presentation sessions had a duration of 15 minutes, where ten were to present and the other five were to discuss ideas. These presentations worked as a brainstorm and sometimes spawned exciting ideas.

For this project, three presentations were performed. Each presentation had a distinct objective. The first one occurred during the first month and its goal was to show that business knowledge had been acquired and the trainee had already conduct market research. The second presentation was somewhere around the middle of the internship and its objective was to show a working prototype of what has been done until then. The last one occurred in the last week of the internship, to resume what had been learned during the internship and show the final state of the developed prototype.

5.2. Backend

The backend implementation was done according to the explained in the previous chapter. All modules that are part of the solution follow the architecture presented in Figure 26. Modules comprises six layers that will now be approached.

5.2.1. Security Layer

The security of any developed application is a huge concern nowadays. This project is no exception, and hence a security layer exists in all of the Online Payments modules. Since the Spring Boot framework already has its security mechanisms, the security layer was built on top of it. This layer works as a filter that only allow authorized users to access protected resources.

The security layer allows configuring how the application should behave when a request reaches the application. This layer specifies the endpoints that do not require any authorization, that is, the exceptions to the security layer. These exceptions are the ones used for authentication purpose. Also in here, it is specified how it should behave when a request without authorization tries to access a protected endpoint.

To authenticate and identify the user who is performing a request, this layer has a class that behaves as a filter and its function is search, extract and validate the authorization token from the header of the requests. This authorization field has in it a JSON Web Token (JWT) and follows the structure *Bearer JWT*.

A JWT is a compact and self-contained way for securely transmitting information between parties as a JavaScript Object Notation (JSON) object. It is a stateless authentication mechanism given that nothing needs to be stored in memory. An example of a situation where it can be used is authentication. Once the user logs in, each subsequent request will include the JWT, thus allowing the user to continue accessing services and resources that are only released with that token. This information can be verified and trusted because it is digitally signed. The JWT consists of three parts: i) header; ii) payload and iii) signature.

The header comprises of two parts: i) *type* and ii) *alg*. The field type identifies the type of the token, while the *alg* field identifies the used signing algorithm.

Payload forms the second part of JWT, which in turn contains Claims. Claims are information about an entity, usually the authenticated user.

There are three types of Claims: reserved, public and private. Reserved is a predefined set that is not required but is recommended for use as it is used in validating tokens. Examples are *iss* (issuer), *exp* (expiration date) and *sub* (subject).

Public type claims can be freely defined and used for public consultation, but in order to avoid a collision, they must be defined in the Internet Assigned Number Authority (IANA) repository or as Uniform Resource Identifier (URI) containing a collision resistant namespace (such as the domain name). Finally, private claims are customized and are used for sending data between two entities and usually contain sensitive data.

The signature allows assuring that the token was not changed during communication between the two entities. The header and payload must be used as input to the hash algorithm to generate the signature. The hash algorithm to be employed is defined in the header. For example, the use of the HS256 algorithm (HMAC SHA256) is indicated by the header as in Figure 29.

Figure 29 - JWT header

In addition to validating the integrity of the message and thus detecting whether the message has been altered or not during transmission, the signature also allows verifying the authenticity of the sender.

On this project is created a new JWT every time that a user logs in. The JWT brings in the claims with the crucial user information that will allow to authenticate and acknowledge the user. The token generated has inside the username, user role and the token expiry date. The token date expires according to the login options, for example, if the user chooses the option “remember me” the token is generated to last one month, otherwise, it will only last four hours. An example of a payload is shown in Figure 30. This token consists of the reserved claims, *sub*, the entity to whom the token belongs to, *iss*, the entity issuing the token, *iat*, the timestamp of when the token was created, and, finally, the *exp* field, a timestamp holding the token expiration time. It also comprises the private claim “scopes” which contains the information related to the user’s role.

Figure 30 – Online Payments login JWT payload

This layer is also where the Cross-Origin Resource Sharing (CORS) Filter is configured, that is, the allowed HTTP methods and the trusted origins with which backend is allowed to communicate with.

5.2.2. Resource Layer

The resource layer is where the application endpoints are defined along with their input parameters. In REST, a consistent and robust endpoint resource naming strategy is fundamental for good API design. In this platform, we adopted the nomenclature shown in Figure 31.

Figure 31 - API naming nomenclature

For the cases where the web application and the backend shares the same machine, it must be possible to distinguish between the two Uniform Resource Locator (URL) paths to avoid a collision. Considering the following URL *http://localhost:80* that is the base path and in a standard situation, it would be occupied with the landing page of the web application. One time that this URL will identify and API, the ideal would be to add the term API to the path of the URL getting the following *http://localhost:80/api*. The versioning of the API is a good practice so it that had been adopted. The API versioning allows to changing the APIs and keep with the legacy versions while they are needed. Continuing with the versioning and completing the URL, it looks like *http://localhost:80/api/v1*. The rest of the path is allusive

to what we want to do. Usually, an API is relative to an entity along with an action. Also, sometimes a parameter as a query string and adding all of it will give the result presented previously.

In this layer, the endpoint inputs are validated before delegating a task to the service layer. This validation is accomplished through the usage of the annotations provided by the dependency Web Starter. These annotations allow specifying the values that a parameter may assume to be valid.

5.2.3. Service Layer

This layer is relative to all the services used in Online Payments. This layer contains all business logic, repository layer requests and calls to external services (Email, FCM, SMS). The usage of a service layer provides code modularity and allows business logic and rules to be specified in only one place. The service layer will expose the CRUD functionality provided by this repository layer through its interface and implementation.

The service layer consumes the following three external services: i) email; ii) FCM and iii) SMS service. All of these services calls are asynchronous, which means that they will be executed in a different thread, so the thread in charge for the user request does not need to wait for the answer of the external service.

The Email Service is a service created to send emails with the purpose of i) recover password and ii) confirm email at the registry. The email sent is built with a template as a base and recurring to Thymeleaf[38] that allow filling HTML tag values with values held into variables.

The FCM service is a service built using the legacy REST API from the FCM Google service and allows to send push notification to users M-Pesa app.

The SMS service is used to send authentication codes and One Time Password (OTP). This service sends SMSs recurring to an external service that is Nexmo[39].

The service layer also holds the scheduled tasks. These tasks run at a specific date/time and within specific intervals of time. These tasks are used to purge the expired tokens that are stored in database.

[38] https://www.thymeleaf.org/
[39] https://www.nexmo.com/

Error handling is done by raising exceptions. For every error that occurs within the service layer, the code throws an exception that prevents the tasks of going further and forces a rollback on the actions already performed. The exception is later caught in the domain layer and transformed in an appropriate response from the backend.

5.2.4. Repository Layer

The repository layer keeps all the repositories, as well as the database connection. This layer provides a set of methods to read, persist, update and remove an entity from the underlying data store. This layer allows the service layer to perform write and read operations at a logical level. A repository is simply a class that provides data store specific implementation to each interface method and resorts to Java Persistence API (JPA) to generate standard repository implementation alone. In more complex queries as the ones used to search/filter resources are used Specifications along with JPA. Specifications is a query language that builds a query according to the inputs provided to it.

5.2.5. Data Mapping layer

The data mapping layer is the crux of the application. A data mapping layer model is a representation of the data storage types required by the business logic. It describes the various domain objects (entities), their attributes, roles, and relationships, plus the constraints that govern the problem domain. For this layer, we use the Hibernate[40] that is an ORM for Java. This framework primary feature is mapping from Java classes to database tables and the opposite.

5.2.6. Domain layer

The domain layer is where the response to the request is formatted to be exposed. This task is performed with Data Transfer Objects (DTO) that are classes identically to their correspondent entity but without the attributes that reveal sensitive information like that entity identifier (ID). The mapping between the entities and DTOs objects is performed with ModelMapper[41], which is an object mapper. This tool copies the attributes from one object to the other, requiring no special instruction.

[40] http://hibernate.org/
[41] http://modelmapper.org/

This layer also harbours a class that catches and deals with all errors that are triggered in runtime. For this purpose, this class catches the exceptions that are raised and converts it into an error response sending back with the format, as shown in Figure 32.


```
{
    "status": {
        "httpCode":401,
        "message": "The Order you tryed to access with Id 1367 does not belong to you."
    },
    "requestSubmitted": "false"
}
```


Figure 32 - Backend error response

5.2.7. Infrastructure Layer

This layer represents the data mechanisms used to persist data, holding databases and Hazelcast. The Hazelcast to work is required to create a so called Hazelcast Node. Each module of the backend will be a Hazelcast Member and these automatically join together to form a cluster. This automatic joining takes place supported by various discovery mechanisms used by members to find each other. The data is persisted in distributed HashMaps, and that is how it is shared across all the Hazelcast members.

5.3. Frontend

The Online Payments portals are split accordingly to user roles and necessities. A landing page helps the different types of user to navigate easily through the web application.

The creation of a web application with multiple portals along with a multi-modular backend presents a problem regarding the URL path to be used to address each portal. The used approach resorted to a proxy located at the root of the project to catch all requests and redirecting them to the correspondent backend module. Moreover, in order to append the authentication token used to free the protected resources, an HTTP Interceptor is used. It catches the requests before they can reach the proxy, adding correspondent JWT token to the request. The routes are protected with a so called AuthGuard. The AuthGuard is a specific class that allows specifying conditions to unlock that path to the user. Also was used the JWT token that had the information regarding the user role to unlock the correspondent role routes.

Across all the portals, the used storage mechanism comprises local and session storage. Most of the data are stored in Services classes which are kept in memory. However, some data needs to be persisted upon the application refresh or at close event, as is the case for JWT that allows a user to re-authenticate again without the need to log in. Each of the storage mechanism is used in different situations. Both of the storages persist data in the browser memory, but while local storage keeps it there, session storage works like a volatile memory that vanishes when the browser tab is closed. Therefore, persistent storage is used for users that want a persistent session, while remember-me-not users have their sessions solely kept in volatile memory. Every time that developer or merchant portal is accessed, the software verifies whether there is any JWT in storage. If it exists, the web application will check its validity through the backend. If valid, the user is automatically logged in. Otherwise, the token is erased from memory storage and redirects the user to the login page.

5.3.1. Merchant Portal

The merchant portal is the portal dedicated for Merchants. In this portal, merchants can find tools to manage their business. Specifically, merchants have the following tools: i) Send money; ii) See transactions list; iii) See order list; iv) Manage order; v) Reports; vi) Manage business credential and vii) Contacts. Figure 33 shows the merchant portal organization.

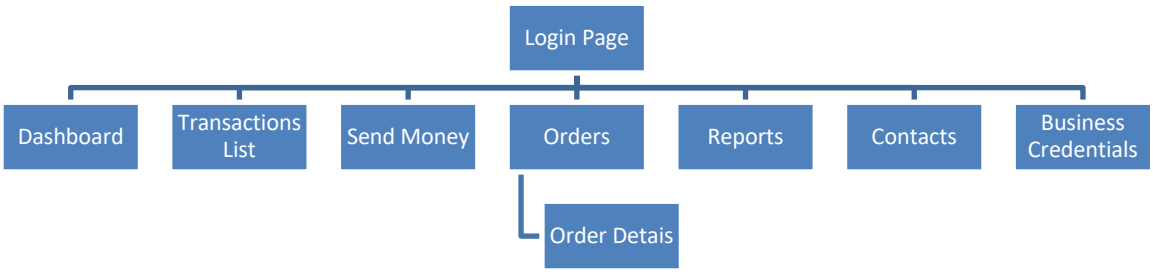

Figure 33 – Merchant portal sitemap

A merchant needs first to register as an M-Pesa Merchant by going to an Agent of the M-Pesa network. This way, the registration page is not necessary since the user has already the

credentials to log in. The login page, Figure 34, in the merchant portal is the only one that a user can access without being authenticated. Indeed, the merchant can authenticate by providing the short code that identifies the business, along with the registered phone number. Note that a business entity can have more than one worker. All business employees can access the business's portal as long as they had been enrolled in the business by the owner.

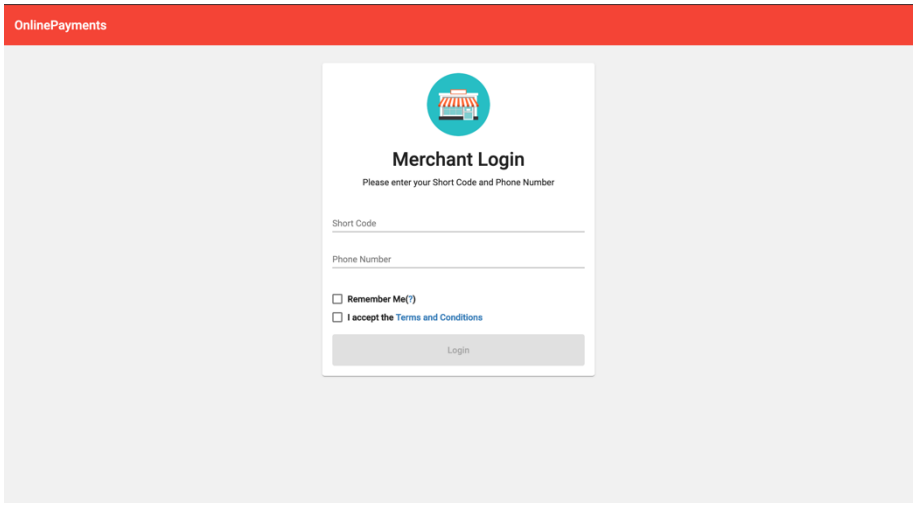

Figure 34 - Merchant portal login

After providing credentials to the backend, it validates the enrollment of the provided phone number along with business that corresponds to the short code. If the merchant belongs to the list of employees, the system sends an SMS with an OTP to complete the login. After having completed the previous step, and if everything was successful, the merchant is authorized to log in, with the system returning a JWT token. The returned token is then saved in the respective memory storage mechanism according to the option "remember me" at login. After login, the merchant is redirected to the dashboard of the merchant portal, represented in Figure 35. Since managing a business is often hard. To help merchants, the system provides a dashboard, which allows merchants to access useful information and quickly overview what has been happening in the business. For instance, the merchant can quickly glance at data regarding the bank account and the current balance. Merchant also can check the activity of the last 7-day orders across the several statuses, and to see recent activities like for example payment sent, received and orders and demanded refunds.

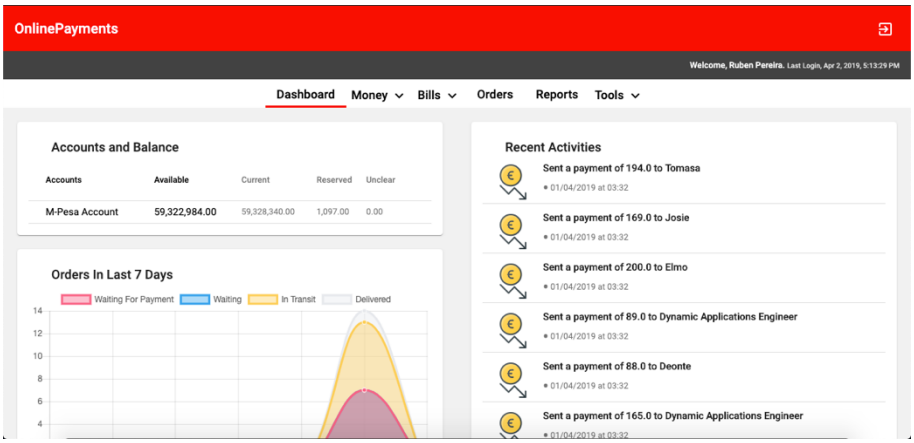

Figure 35 - Merchant dashboard

Keeping track of money in a business is essential. For this reason, the software creates transactions list, a merchant can filter the business's transactions with the following parameters: i) transaction ID; ii) transaction type and iii) within a given range of dates. The filter options are sent to the server, which queries the appropriate data from the DB. The transaction list resorts to a paginator to fetch solely the data to be displayed in the visible area. The Figure 36 shows the transaction list.

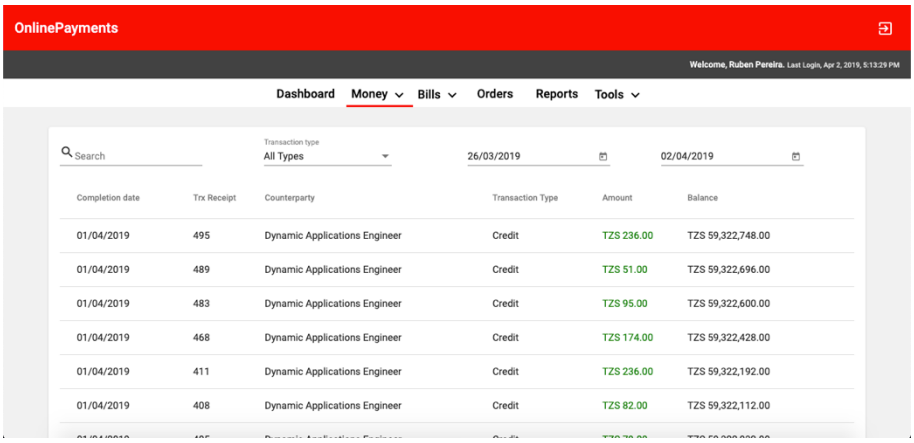

Figure 36 - Merchant transaction list

Having a business implies to have some expenses too and thus the need to pay bills. For this purpose, the software has an area where a merchant can send money from her/his account to other M-Pesa users. To perform a money send operation, the merchant needs to provide the receiver phone number and the amount to send. Often, these transactions are to frequent users – regular suppliers, for instance. To this end, the system offers an autosuggest that is updated accordingly to the contacts saved by the merchant. Note that the contacts can be searched either by phone number or by name. In Figure 36, it is possible to see the list of suggestions based on merchant input.

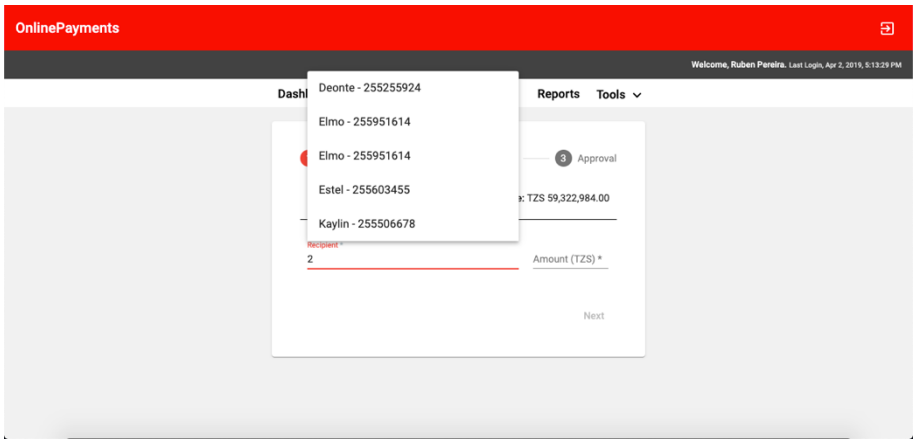

Figure 37 - Send money choose receiver

After providing the phone number and the amount of money, the backend validates the phone number to assure that the destination phone number belongs to an M-Pesa user. When validation is successful, the next button is unlocked, allowing the user to proceed with the transaction. Figure 38 represents the validation step.

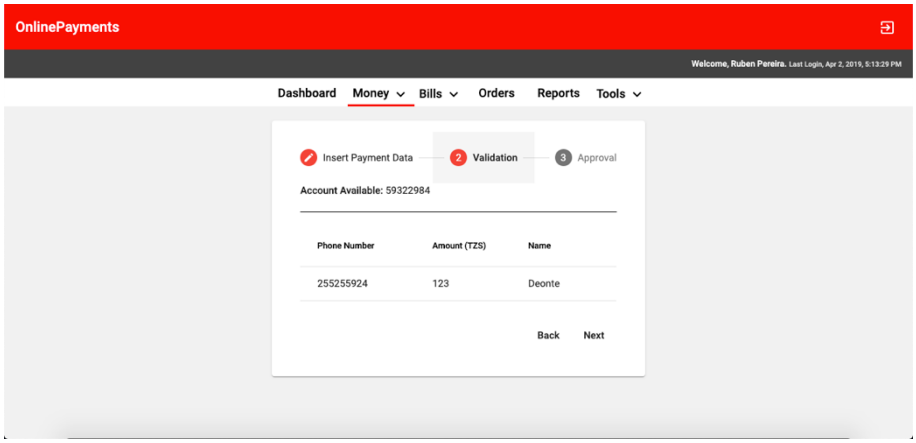

Figure 38 - Send money server validation

The final step of the payment operation corresponds to the result where the merchant can check the status of the transaction, as can be seen in Figure 39.

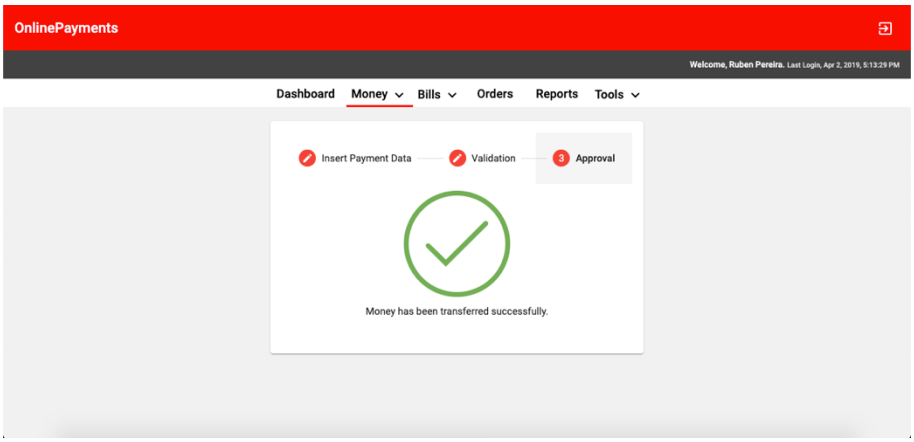

Figure 39 - Send money status

For a business with more than a few employees, paying wagers individually to all employees is inefficient from the human point of view. To speed up this use case, the system allows merchants to perform transactions in bulk. For that, the merchant only needs to upload a CSV file holding the payment data.

Merchant has access to a list of orders, Figure 40. This list alike the transactions list allows the merchant to filter the orders as he wants to.

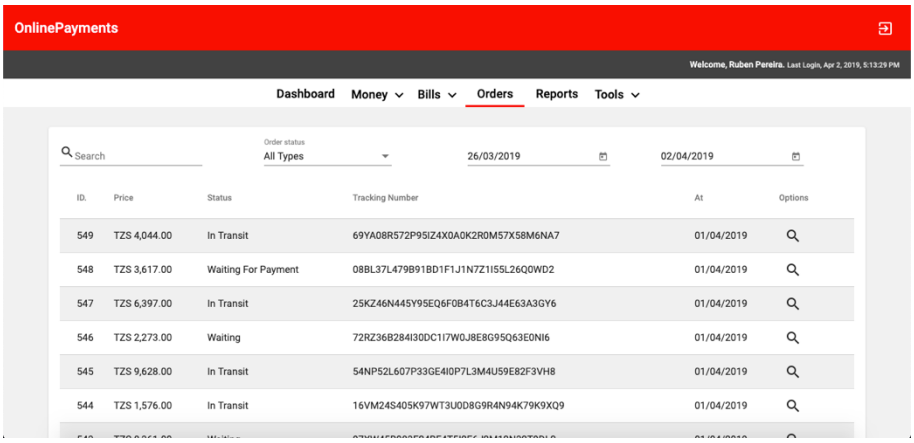

Figure 40 - Order list

Merchant can see and manage the business orders, accessing the details page of it. In the details page, the merchant can consult the customer information and check or edit the order status as well as the tracking number. Merchant is also able to refund the order or request to print a label for the order package. Figure 41 shows the order summary.

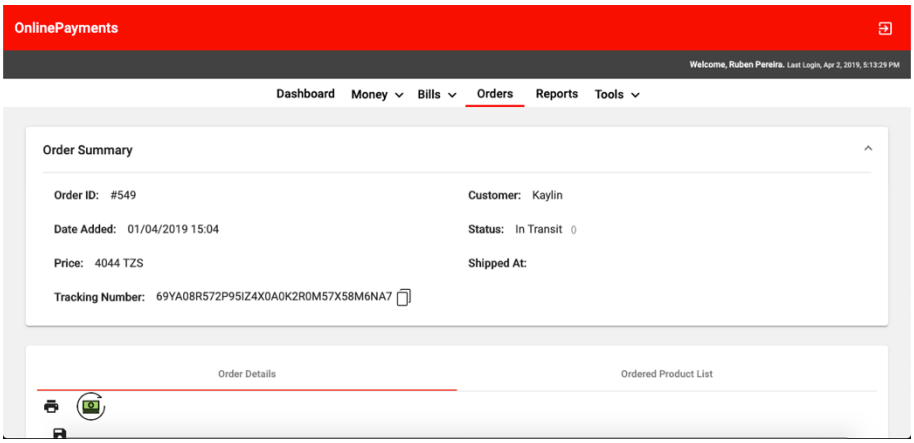

Figure 41 - Order details

Additionally, on this same page, the merchant can find the list of products that had been ordered by the customer, as it is possible to see in Figure 42.

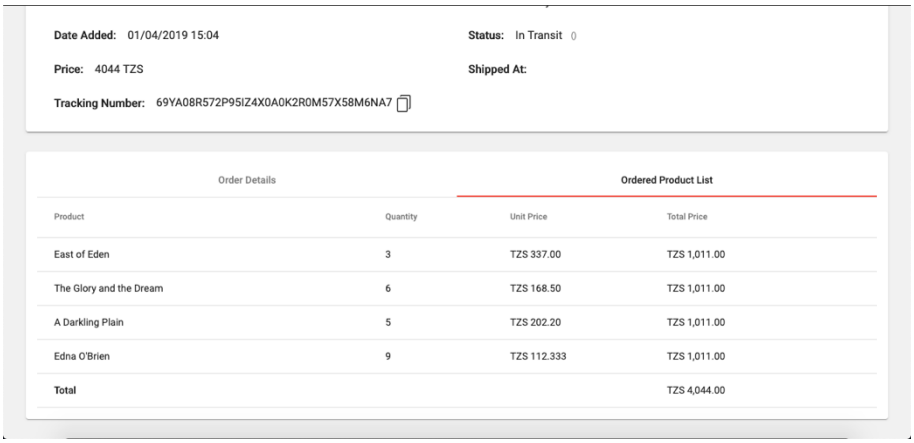

Figure 42 - Order details product list

Behind the dashboard that is equal for all merchants, each merchant has her/his individual report area where they can customize and choose a variety of widgets to be displayed in their customized report. The reports area is highly customizable, as can be seen in Figure 43, and allows to move, add, remove or resize the widget as the merchant wants to. The settings for the merchant-customized reports are stored in the DB as a JSON object. The list of widgets available for report customization is:

- **Average order value**
- **Bank account available money**
- **Financial list**
- **Last seven days balance resume**
- **Last seven days orders resume**

- **Lifetime sales amount**
- **Total orders**

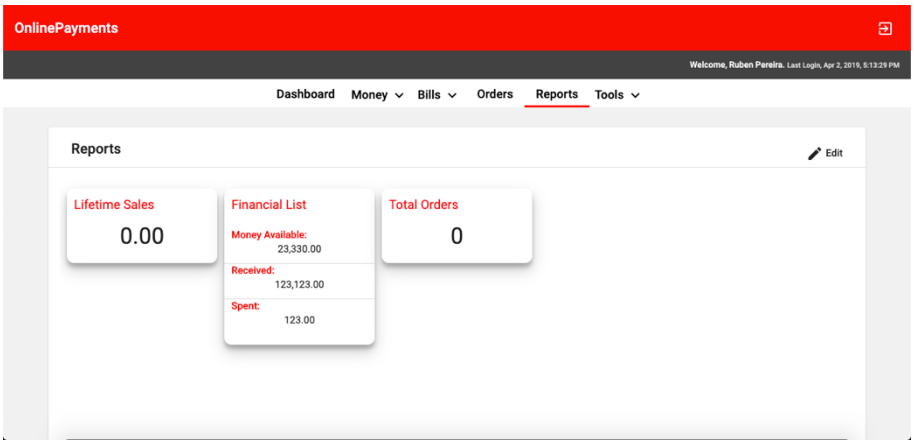

Figure 43 - Reports area with widgets

As already mentioned, the merchant has a contact list, Figure 44, where can add, edit or remove contacts. This contact list keeps the most frequent phone numbers to whom merchant send money. This way, the contacts of this list will appear as a suggestion while fills the receiver phone number in a transaction.

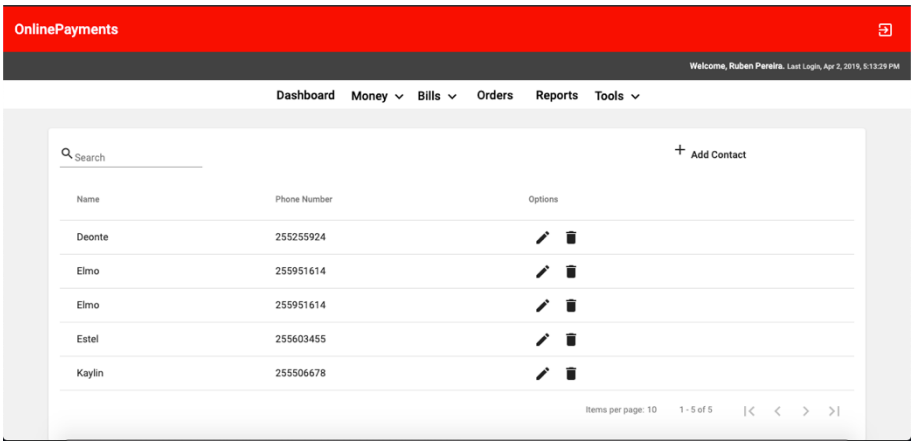

Figure 44 - Contacts list

The last area of the merchant portal, Figure 45, is also one of the most important: credential settings area. This area allows the merchant to get/ request to renew her/his identifier token in the Online Payments checkout API.

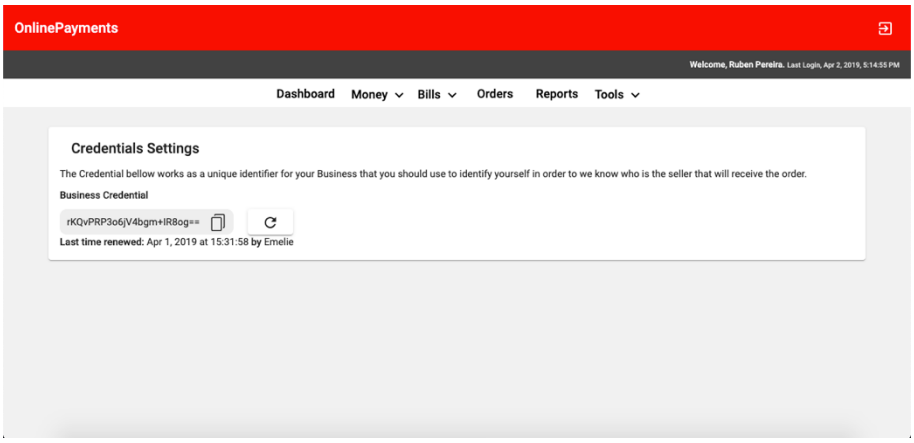

Figure 45 - Credentials settings

5.3.2. Developer Portal

The developer portal is an area where developers can easily access their applications information and monitor their implementations. Like the merchant portal, the developer portal also has a login area where developers sign in and authenticate themselves with their email and password. Otherwise than merchants that need to create a merchant account within a M-Pesa agent, developers simply need to register themselves. To sign up, as can be seen in Figure 46, developers need to provide their email address, a username and define a password.

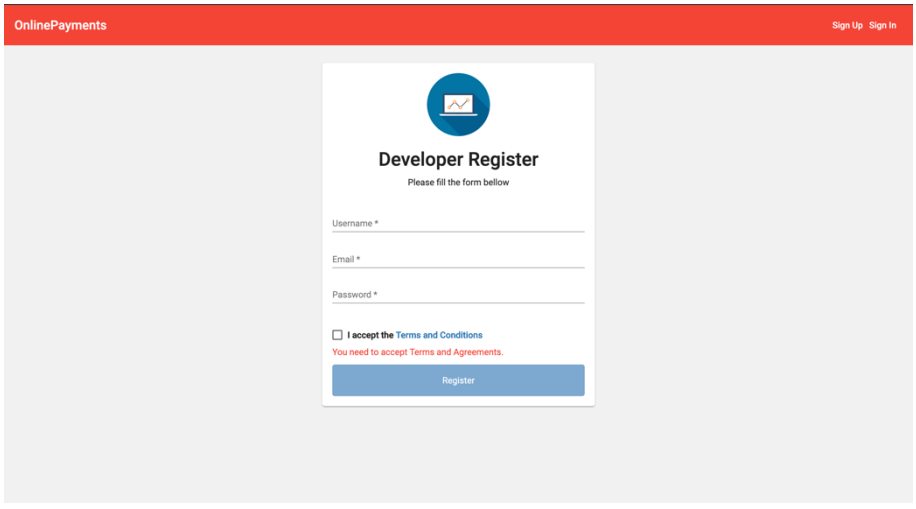

Figure 46 - Developer register

After having filled the required data, the developer receives an email in the provided email address, as shown in Figure 47, with a confirmation button to effectively complete the registration.

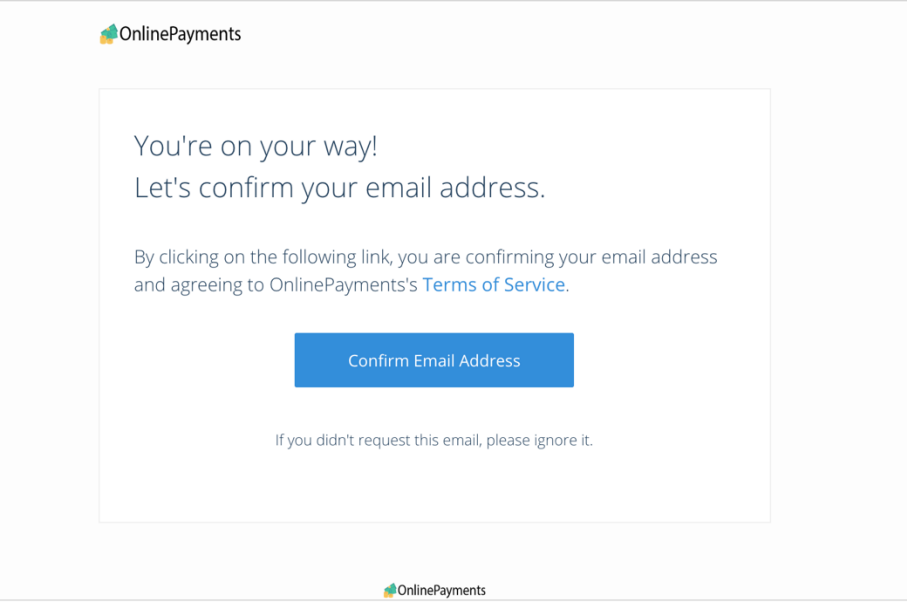

Figure 47 - Developer register email

The registration email is generated with HTML as base and the button that user clicks redirect her/him to the portal with a query string in the URL that is a unique token generated with the objective of identify the developer in this moment. To avoid brute force attacks the token generated is an UUID and 24 hours after its creation it will became invalid.

After the developer has confirmed the registration, she/he is redirected to the login page to authenticate her/himself and enter in the account details. Note that if by any mean the developer forgot his account password, she/he can recover it by initialize the recover password process. For password recovery, all that needs to be done is to provide the correct email address, with the system sending a reset-through-click URL in a recovery email. Figure 48 represents the email sent.

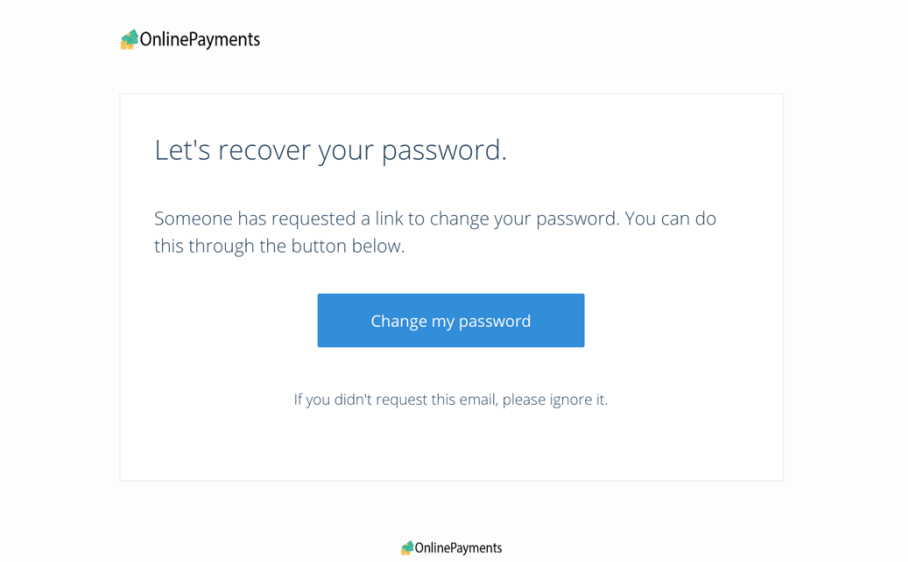

Figure 48 - Developer recover email

The developer's dashboard gathers what is considered essential data. Specifically, the dashboard has a plot and a table with data of the last 48 hours. The plot, as in Figure 49, shows the number of calls to the applications, separating between successful and unsuccessful ones. The table lists the last error of each application, the number of incidences that occurred in the last 48 hours and the time elapsed since the last time the error occurred.

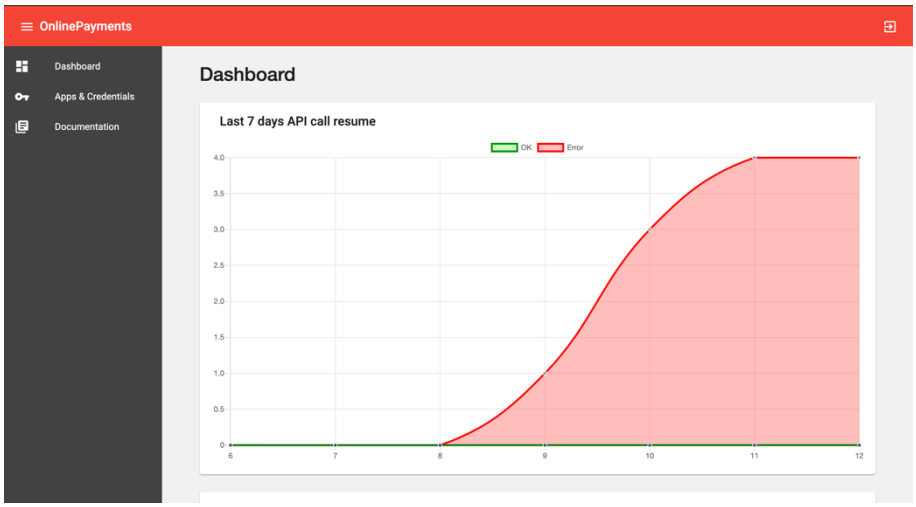

Figure 49 - Developer dashboard graph

Developers can create applications at their will. An application is an online store identifier as an entry point as authorization to the API at checkout. An application, as can be seen in Figure 50, is also a hub of information of the previous calls and credentials that should be used to create an authorization token to access the protected resources at checkout. Each application has two environments: i) sandbox and ii) live. The sandbox environment is like

a playground for developers. It works as an environment for testing purpose and to help the developer to check whether everything is working how it should be.

Conversely, the live environment is used when the API implementation is finished and ready to be used. Note that the status of an application – sandbox vs live – can be changed at any moment, meaning that developers can freely change the application status between sandbox and live. Taking into consideration that the reason for the change of status from live to sandbox could be originated for security reasons all the transactions in progress are not possible to be completed while the application remains in sandbox mode.

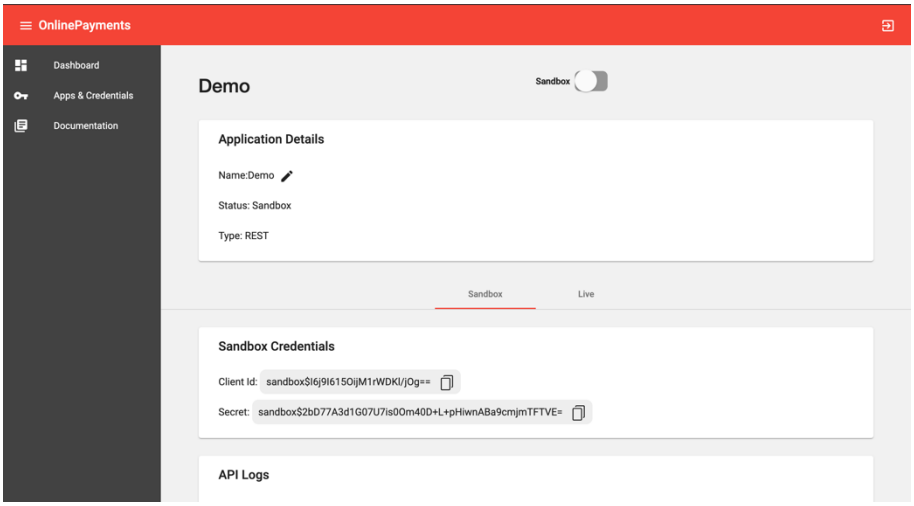

Figure 50 - Developer application details

To create an application, the user needs to provide a name and select the desired protocol to communicate with the application: REST or Simple Object Access Protocol (SOAP).

The application live credentials only are unlocked when the user publishes the application for the first time. Until then, the credentials and all information in the live area are kept hidden.

On both environments, the developer has a small report area with all the errors that occurred in the last 48 hours. The developer can also see the body of the request to understand what effectively happened, as in Figure 51.

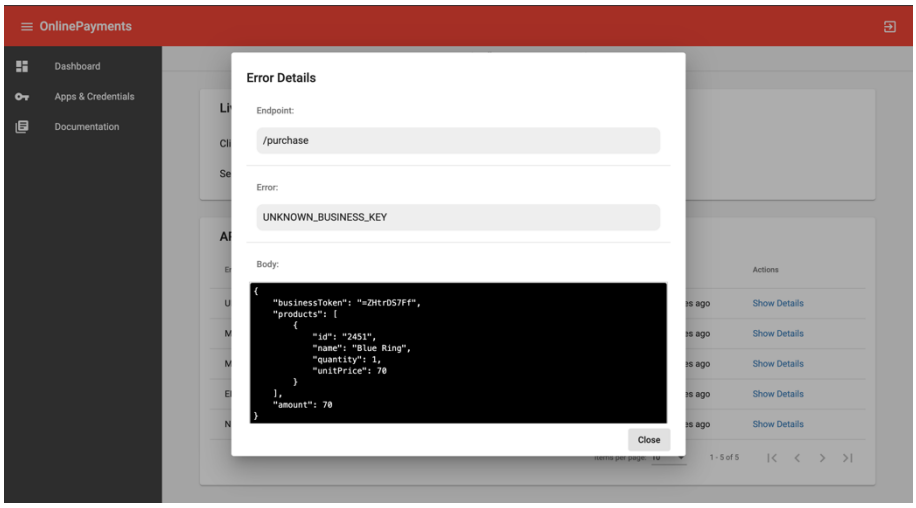

Figure 51 - Application error details

The sandbox environment would not be much of a help if it just accepted any input in the fields. Therefore, for testing purposes, the developer portal has a dedicated section to create sandbox credentials. The developer can create Merchant as well as Shopper credentials. These credentials can be used to test the integration and implementation of the API. Figure 52 exhibits both of the credentials list.

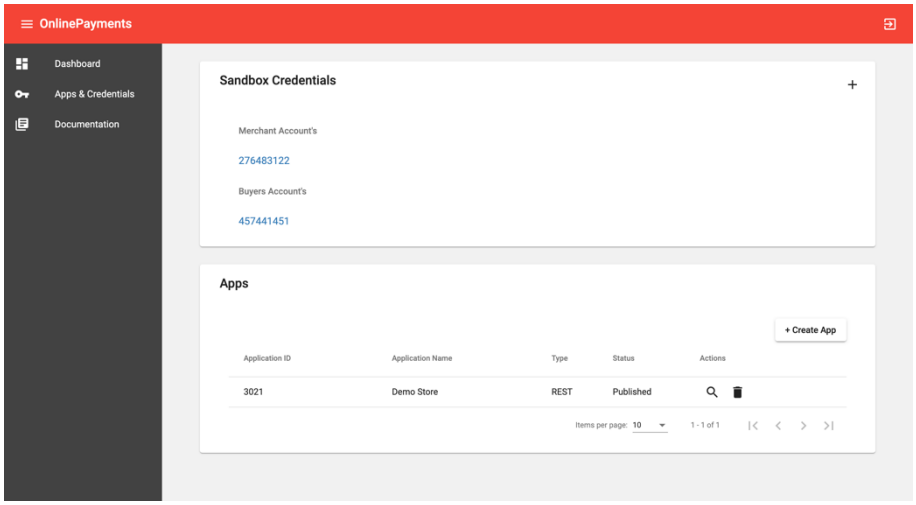

Figure 52 - Sandbox credentials

5.3.3. Checkout Portal

The checkout portal is the portal where the user is redirected after having selecting *pay with M-Pesa* on any online store. Simply put, this portal is the payment gateway. Despite the existence of this checkout portal, online stores do not obligatorily need to redirect their customer to this portal. All the required checkout APIs for this process are public and allows online stores to create their portals or to integrate it directly on their store. To be possible to

use Online Payments APIs developers must create an application on their account. The application is used to identify the origin of transactions and provide the credentials required to generate a token to access the protected APIs.

On the checkout, after the customer chooses the option to pay with M-Pesa is requested to the Online Payments server to generate a JWT token. This token will work as an identifier for the current transaction and will be used as an authorization key to access the server APIs. JWT is generated using the application ID and secret that is extracted from the developer's application details, as shown in Figure 50.

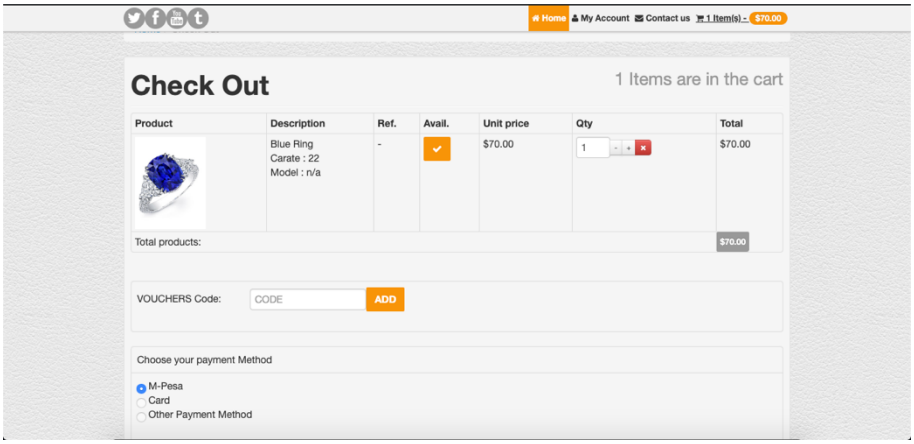

Figure 53 - Demo store checkout

The backend returns a JWT which is used during all the checkout. This token identifies the current application being used along with the transaction. After receiving the token, the online payment store sends the purchase information, as shown in Figure 54, to store on the backend.

```
{
    "businessToken": "quDkdXLDLFeUX+wJuVeOTw==",
    "productList": [
        {
            "name": "Blue Ring",
            "quantity": 1,
            "unitPrice": 70,
            "totalPrice": 70
        }
    ],
    "amount": 70
}
```


Figure 54 - Store order JSON structure

The backend will validate the information sent to it and in case of any error in the structure of the JSON, it will append an error in the application and store the JSON body to further analysis of the developer. The order is stored at the database and the backend returns a unique identifier token for that transaction. That token identifies the order at checkout and will last for 24 hours. After the 24 hours, the token expires. Assuming that the user is redirected to our checkout portal, Figure 55, she/he will be presented with the order list and a field to fill in her/his M-Pesa phone number.

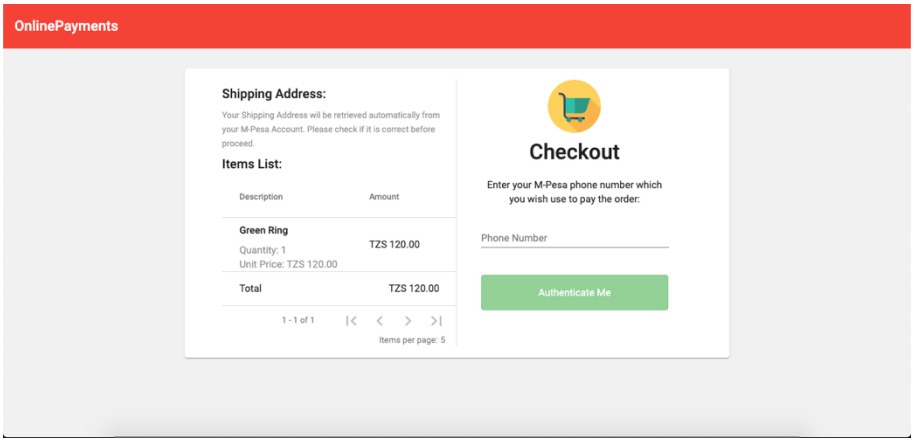

Figure 55 - Checkout portal

Note that when the user submits her/his phone number, the order is automatically bound to her/him. Furthermore, she/he will receive an SMS with the OTP, as in Figure 56.

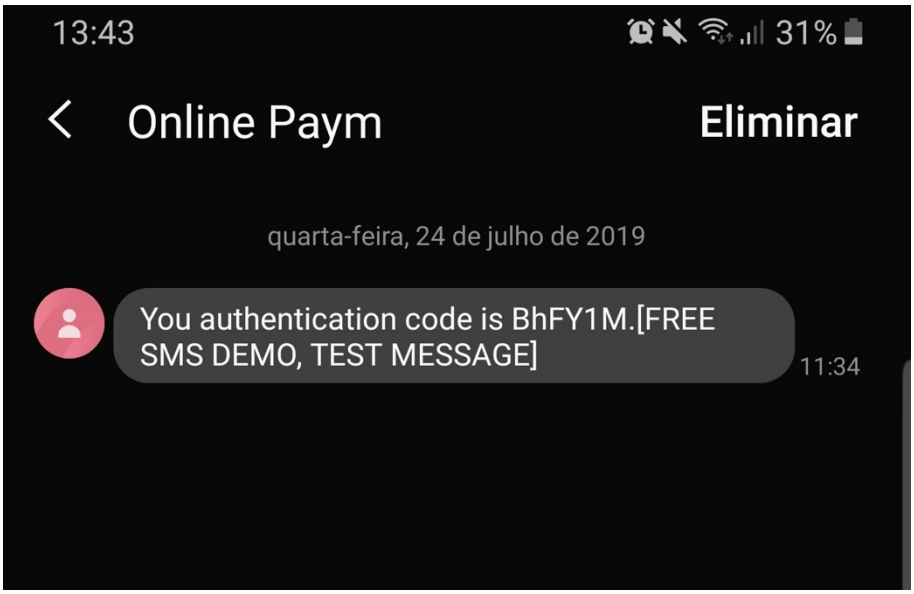

Figure 56 - OTP to finish checkout payment

To authenticate, a user only needs to insert the authentication code in the checkout portal, as in Figure 57, and if everything matches, the purchase is considered to be complete.

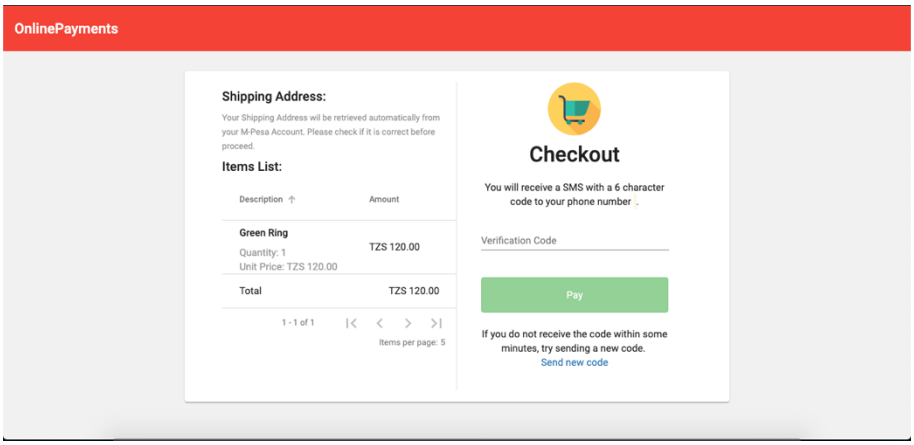

Figure 57 - Checkout portal waiting for OTP

In order to make all this process much simpler has been developed a simple SDK to the Angular framework that Developers can implement to simplify the implementation process. The SDK deals with all API calls that are necessary at checkout and redirects the user to the Online Payments checkout portal. The SDK also provides interface classes that allow developers to implement the interfaces and know the exact format that their objects should have before requesting the checkout. The SDK helps to prevent errors.

5.3.4. Demo Portal

The demo portal, Figure 58, is a portal to show the functioning of the Online Payments portal, as well, the checkout experience. This portal consists of an online store that interacts with the Online Payments.

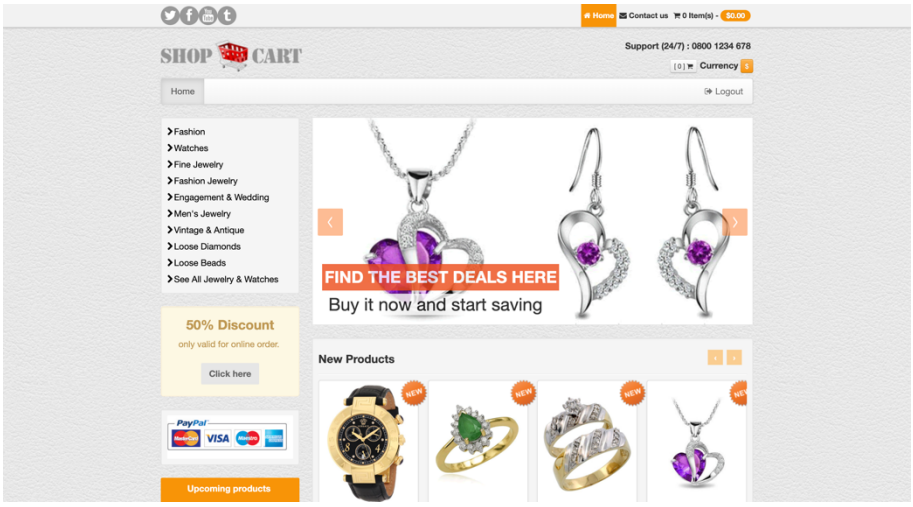

Figure 58 - Demo portal

It is possible to register new accounts (Merchant and Customer) in this online store. To simulate a real scenario of an online store, merchants have a personal area where they can set up their Online Payments business token to be identified along with the API. All the data generated in this platform will trigger real events on the Online Payments platform.

5.3.5. Documentation Portal

Implementing a payment solution is not an easy task. The documentation portal, Figure 59, exists to provide help and guidance. It holds code samples, and also all the API fields explanations needed to implement it easily. The documentation portal is a simple portal based on statically HTML pages. The ultimate goal of the documentation portal is to work as an information hub to developers, so they can easily understand how everything works.

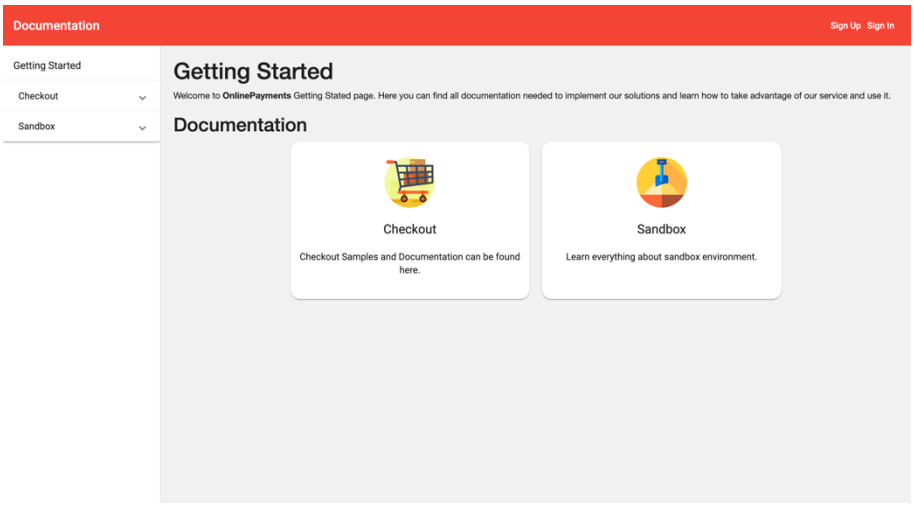

Figure 59 - Documentation portal

The documentation pages, as can be seen in Figure 60, have the information regarding all the API request and answers as the JSON objects samples related to the topic.

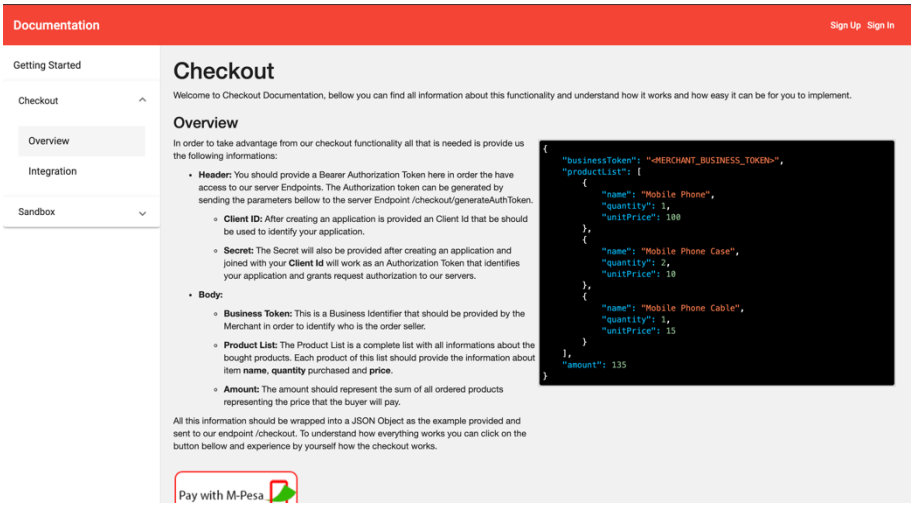

Figure 60 - Documentation page

5.4.Security

The security of the Online Payments users is one of the main concerns of the project. Security includes, besides the security layer, well known practices. Examples include the ban in using incremental IDs in URL with query parameters. Instead, UUIDS are used to increase the difficulty of guessing the right one by malicious individuals. Moreover, identifiers are grouped with tokens at the backend, allowing for time-based expiration of identifiers. Another security-oriented measure is the use of encryption at the database for sensitive data. Note that although relying on local storage can security-wise be dangerous, the project

follows the traditional security approach of resorting to cookies to store JWT data. Finally, another security-wise example is the five-minute timeout that is triggered when no interaction occurs for the last five minutes. Both merchant and developer portals have this feature enabled. Moreover, if the “remember me” option is off, the user is automatically logged off as a consequence of the timeout.

5.5. Deployment

The deployment of a solution with more than one backend module along with a web application is not simple and rather time consuming. To speed up the task, the process was automated through a script that locally generate a build for deploy of frontend and backend modules. The deployment script resorts to the parent Maven .pom file to build the backend modules all at once, as well as the web application given that they were all wrapped up in a single project. This script is also used to configure Jenkins and generate automatic builds every time that a merge operation occurs in the development source code.

The deployment of the backend into the VM is performed through the embedded web server that already come on built modules with the *nohup java -jar* command and redirecting the logs of each .jar file to a specific log file. For the deployment of the web applications, the project relies on the Nginx web server. The VM used for deployments only has two open ports port 22 for both Secure Shell (SSH) and Secure File Transfer Protocol (SFTP) and port 80, which is the well-known port for web communication Note that Nginx is also used as a proxy that redirects the API request to the correspondent backend module.

6. UX/UI Refactoring

Software quality is assessed by the used through what is often called the user experience. User experience is mostly the results of the quality of the user interface and on how it suits users' needs. The last stage of the internship was devoted to carefully plan and analyze the interface and how to provide the best user experience.

The effort for a good user experience started at the beginning of the project, and the interface had continual improvements, with WIT Design Team providing UX/UI reviews and valuable feedback. A set of documentation was created to help WIT Design Team to grasp the main concepts and goals of the platform (Appendix E).

The result of WIT Design Team review yielded not only changes in the UI and user experience, but also the request to use a different frontend framework: dropping Angular to instead use React. To ease the checkout experience, the Design Team also requested to include the M-Pesa mobile application in the solution.

6.1. Android

The M-Pesa project already had a customer application, which is quite popular among users. To leverage this user base, we decided to extend this customer application with the new functionalities. The changes focused on simplifying the checkout process and, at a larger scale, to improve user experience.

6.1.1. Architecture

Extending the M-Pesa customer Android application did not require changes at the architecture level. Only some changes were performed regarding the application flow, to better integrate the changes.

It should be pointed out, that M-Pesa Android APP is no ordinary application. Despite the typical behaviour of mobile applications, this one is built based on a configuration that comes from a backoffice, as shown in Figure 61. This configuration is a JSON file that is loaded at application startup and brings in it all the application views and behaviours. This kind of approach allows the application to become more versatile, since it is possible to change at any moment the screens. In a way, the application acts similarly to a web-based application.

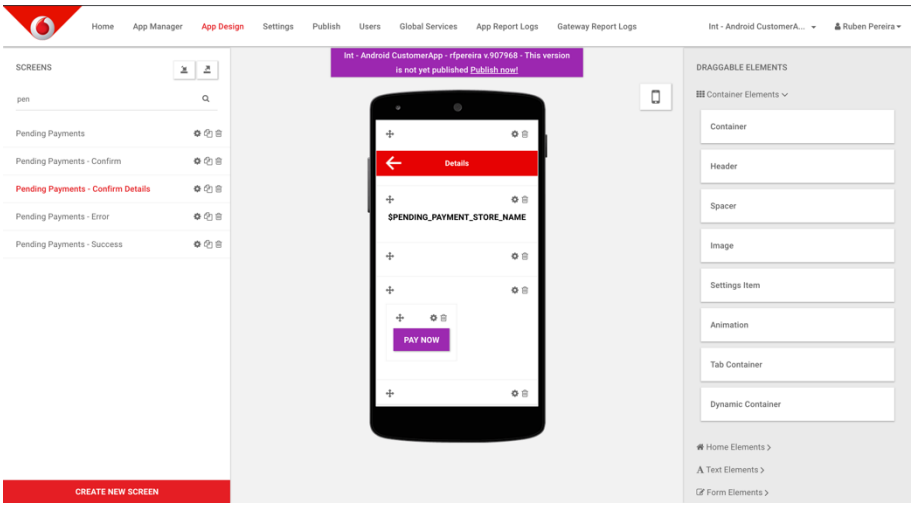

Figure 61 - Backoffice application screen configuration

The communication between the application and the backend is performed using two different protocols. Specifically, the communications between the application and the backend are done using REST, while the communications from the backend to the application uses REST and Firebase Cloud Messaging (FCM) push notifications. This approach is mostly dictated by the fact that the backend only works as an SDK for the mobile application requests. All requests that reach the backend are redirected to the payment processor for an answer, and can therefore suffer a meaningful delay and consequently causes the system to timeout. To minimize timeouts and resource usage, namely the connection socket, we devised the following approach: requests arrive at the backend and are added to a queue to be processed, while a synchronous response is immediately sent to the application, informing it that its request is awaiting to be completed. Then, when the backend receives a response from the payment processor, it sends the requested data through the FCM push notification. Otherwise, if the application only requests the order list, the order list is queried from Online Payments backend, with the response being sent through a synchronous response.

The features added to the application were developed and tested, supporting from Android 4.1 (SDK 16) to the last version Android 9 (SDK 29).

6.1.2. Implementation

The features developed aimed to make the checkout experience more user friendly, giving the possibility for users to manage their orders directly from the application. This change required the addition of a dedicated section to online shopping, where customers could manage their pending purchases. With these changes, the customer introduces his/her phone

number at checkout portal unlike previously where he/she would always receive a SMS with the *auth* code. In the changed version, if the user had already opened the M-Pesa application, an FCM token is stored in the database, so that a notification is sent regarding the new pending payment (Figure 62). Users that access the system without the M-Pesa Customer application receive an SMS with the code that allow to complete the purchase directly at the website.

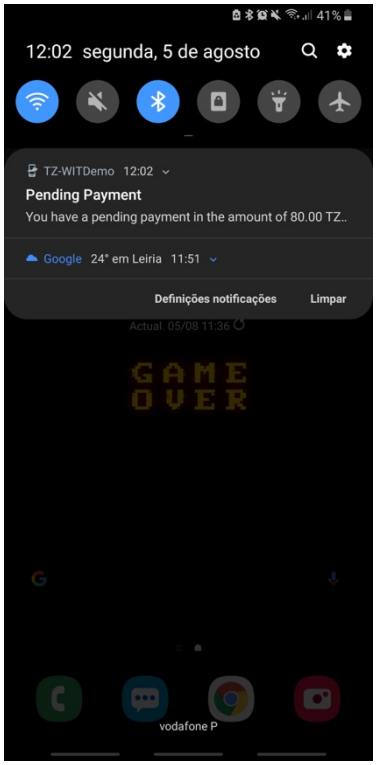

Figure 62 - New pending payment notification

To avoid performance costly subsequent requests to gather all data regarding the purchase that customer has performed, the push notification brings all the order details. Figure 63 is a sample of the JSON object sent to the FCM that triggers the push notification. As it is possible to see, the JSON structure has fields named *to*, *notification* and *data*. The field *to* identifies the user to whom the push notification should be sent. The *notification* field represents the notification that to be shown to the user, and finally, the field *data* represents the data of the order that is sent along the push notification.

```
{
   "to":"<TOKEN_FCM>",
    "notification" : {
        "body" : "You have a Pending Payment",
        "title" : "Store is waiting for you to finish your Pending Payment."
    },
    "data": {
        "responseType":"CM_PENDING_PAYMENTS",
      "payload":{
         "pendingPaymentData":{
            "pendingPayment": {
                  "id":"1547",
                  "storeName":"Store61",
                  "numberOfItems":7,
                  "amount":4461.0,
                  "expiresAt":"2019-04-30T11:35:53.532",
                  "orderId":1547
               }
         }
      }
    }
}
```


Figure 63 - JSON sample trigger push notification

Google services that are running in the background receive the push and trigger the notification to the user. They also extract and append the field *data* to the *MainActivity* extras. This allows serializing the field *data* to an object on the next startup of the application. Note also that when the application is opened in response to a notification, it does not request the user to authenticate, thus saving the user from having to provide its authentication PIN. Instead, the customer goes directly to the pending payment screen (Figure 64).

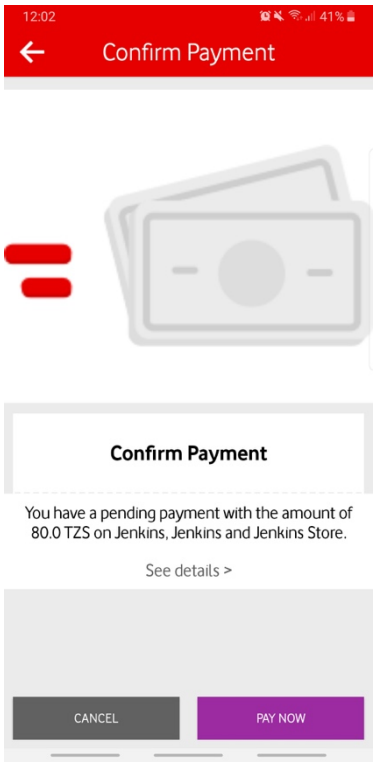

Figure 64 - Online payment

As the name implies, in the Pending Payments screen, the customer can see the pending payment details, and either pay or cancel the order (as seen in Figure 66). One of the problems was the size of data to send using the FCM. Indeed, the pending payment details could have several elements, overflowing the maximum payload size of the FCM. To solve this issue, a JSON structure was devised to send the data over several push notifications as packets. This is done with a *requestId* that identifies the request and allows to group the several push notifications that are received. There are also fields *packetNumber* and *totalPackets* that identify the number of the packet and the total of packets that application will receive. The structure mentioned previously is shown in Figure 65.

```
{
    "to": "<TOKEN_FCM>",
    "data": {
        "responseType": "CM_PENDING_PAYMENTS",
        "packetNumber":1,
        "totalPackets":1,
        "requestId": 1234,
        "payload": {
            "pendingPaymentList": [
                {
                    "id": "1547",
                    "storeName": "Adidas",
                    "numberOfItems": 2,
                    "amount": 1461.0,
                    "expiresAt": "2019-04-30T11:35:53.532",
                    "orderId": 1547
                },
                {
                    "id": "1563",
                    "storeName": "Nike",
                    "numberOfItems": 1,
                    "amount": 59.0,
                    "expiresAt": "2019-04-31T19:35:52.532",
                    "orderId": 1563
                }
            ]
        }
    }
}
```


Figure 65 - Push notification with pending payments

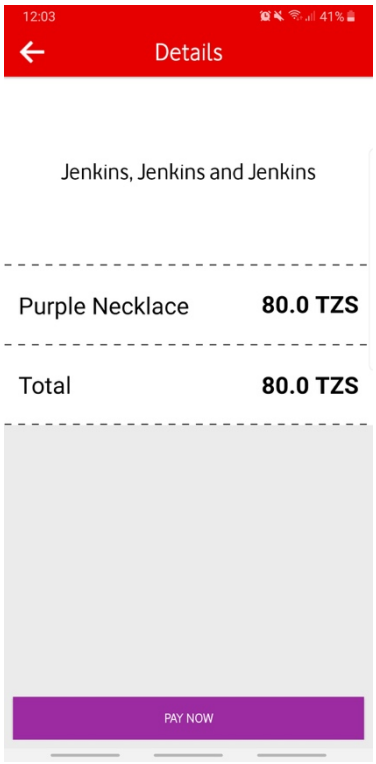

Figure 66 - Pending payment details

In Figure 66, the user can also accept and pay the order. To complete this step, the user needs to insert her/his authentication pin. If the purchase is well succeeded, the purchase invoice can then be downloaded. This download via an Async Task that performs the download and notify the user when the download has been completed (Figure 67).

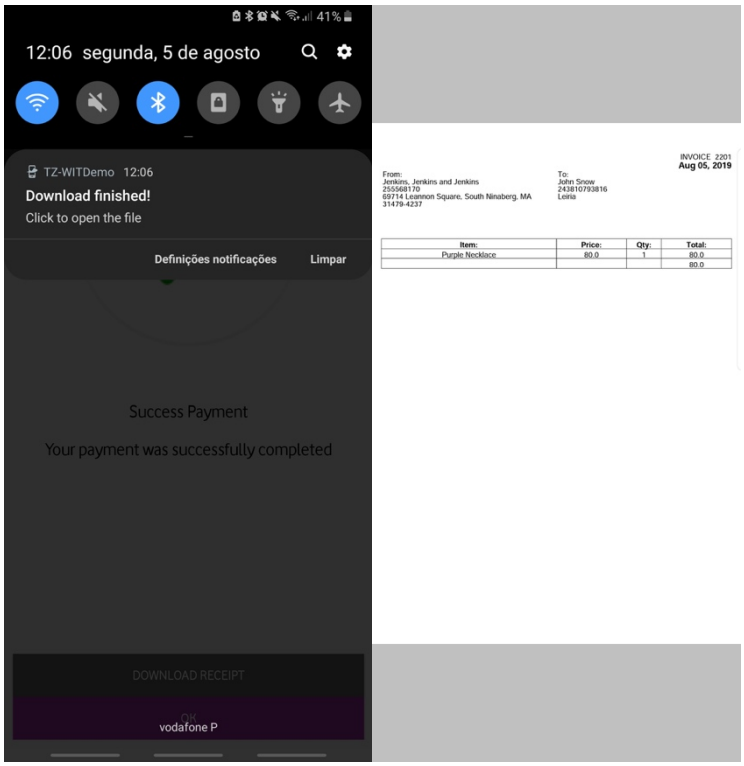

Figure 67 - Download invoice

The user also can consult the list of pending payments from the application menu, as can be seen in Figure 68.

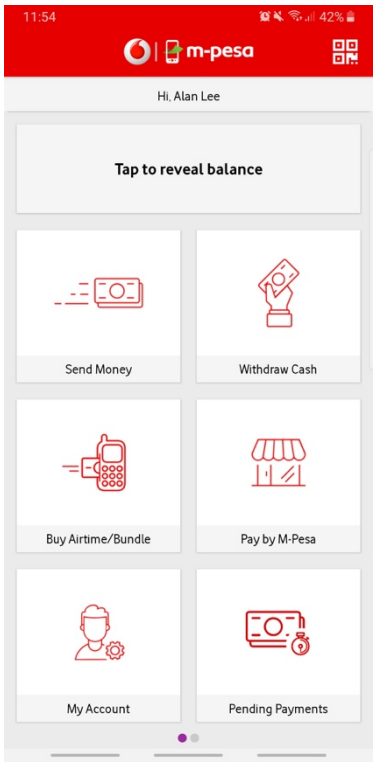

Figure 68 - M-Pesa application menu

By selecting the option *Pending Payments,* a list of pending payments is shown (Figure 69). Each pending payment lists the amount, the store name and the time remaining to perform the payment. Note that a pending payment needs to be completed within 24 hours, or it will be automatically cancelled.

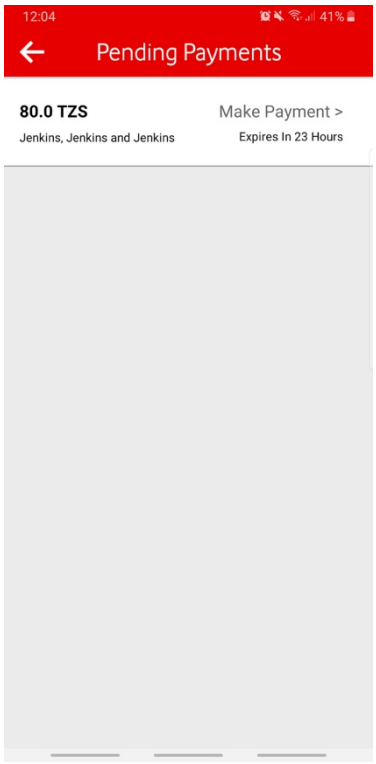

Figure 69 - Pending payments list

6.2. Frontend

The Angular framework was selected for the implementation, mostly because it was the one being used by the development team in other projects. During the internship, the development team started to explore the React framework, adopting it for new projects, a choice also flavored by new members who joined the development team. Moreover, WIT Design Team recommendations also pushed toward the use of the React framework, and thus, the decision was made to refactor the frontend for the React framework.

6.2.1. Architecture

The usage of a new framework forced the creation of a new frontend project. As the framework changed the architecture of the web application, the application itself also incurred some modifications.

The React-based project resorted to the Redux-Sagas pattern. This pattern suits well with React, since it allows i) to describe UI as a function of state, and ii) Redux emits state updates in response to actions. The pattern significantly simplified the code, also enhancing how the side effects sequences were managed. Overall, the whole application architecture had to change, as show in Figure 70.

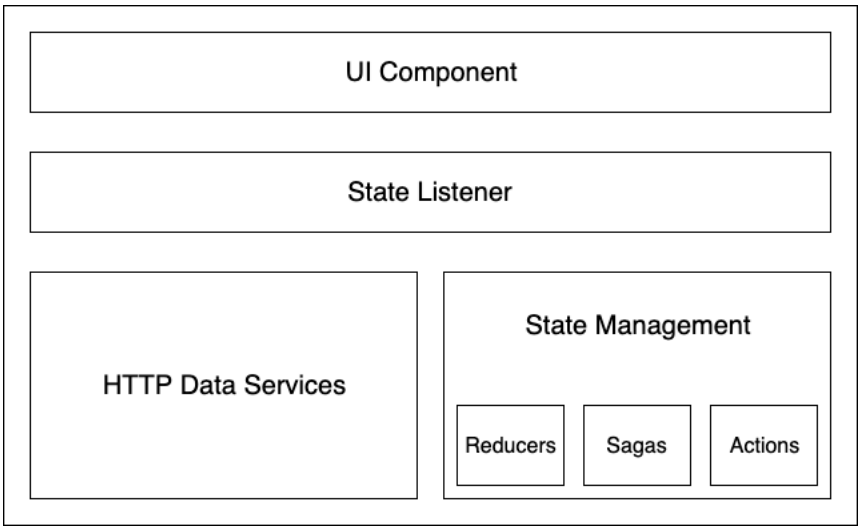

Figure 70 - React web applications architecture

The architecture, along with the used pattern allows the user to trigger an action on the UI that dispatches an event which triggers the invocation of a method that effectively executes an HTTP request. The server response goes through the reducer that triggers a change on the event and stores the server response and updates the UI as a consequence of event changes. This chain of events is depicted in Figure 71.

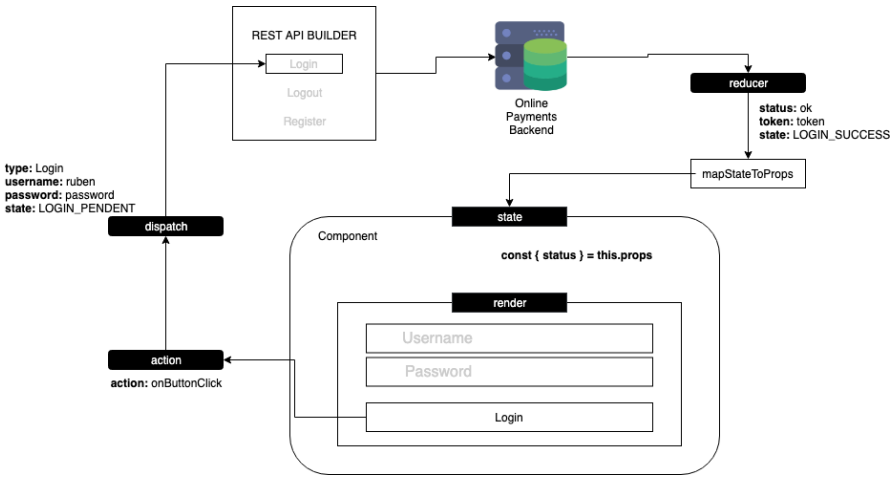

Figure 71 - Redux-sagas event chain

Despite the changes in the architecture, some aspects were kept like for instance, *lazy loading* to avoid loading more assets that the necessary. The HTTP interceptor was also maintained, since it is needed to add the authorization token to the header on request exit.

6.2.2. Implementation

The changes in the UI/UX were significant, and apart the Demo portal, all other portals had to be recreated. The UI changes performed across all the portals follow the same guide lines and use the same UI components.

The merchant portal was one of the portals that had the most changes. The login of the merchant changed and now he/she logs in providing his/her business identifier along with a username and password, as can be seen in Figure 72.

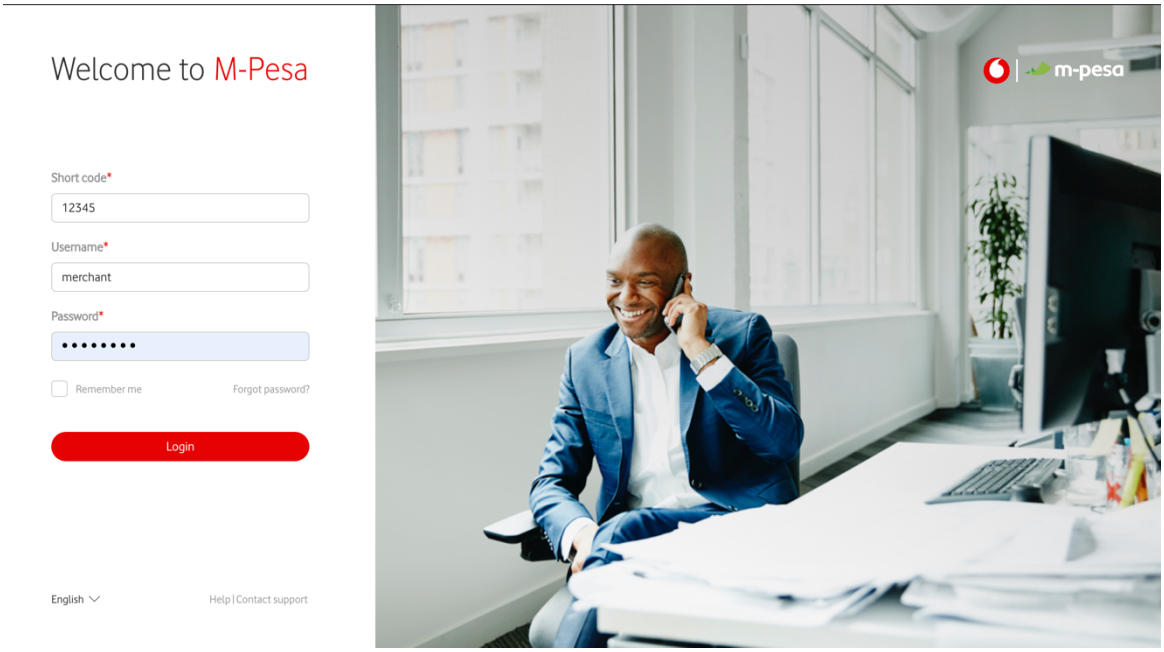

Figure 72 - Merchant login

Figure 73 shows the new UI of the merchant dashboard. Specifically, the dashboard now holds the data regarding the last transactions and the amount that the user still has left in his/her M-Pesa wallet, as well as last transactions that was performed to his/her account.

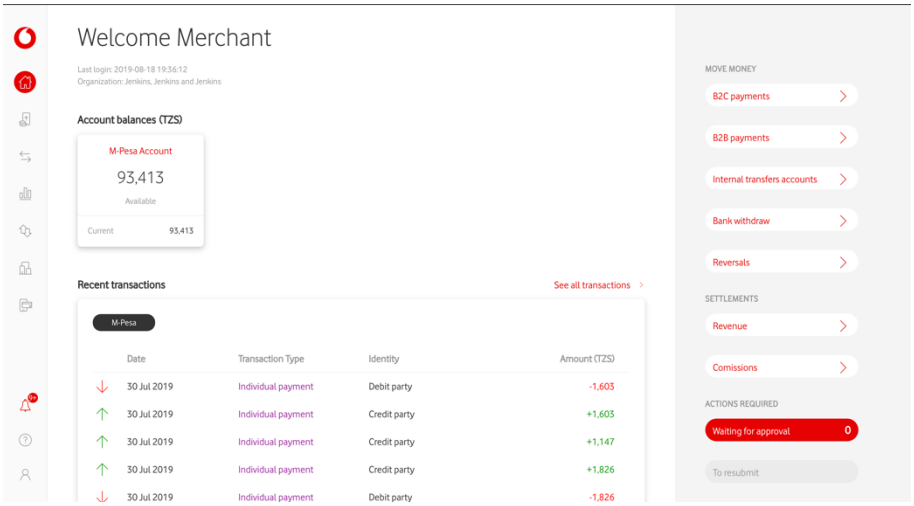

Figure 73 - Merchant dashboard

Other changes at the dashboard included moving some elements to a dedicated area, as shown in Figure 74.

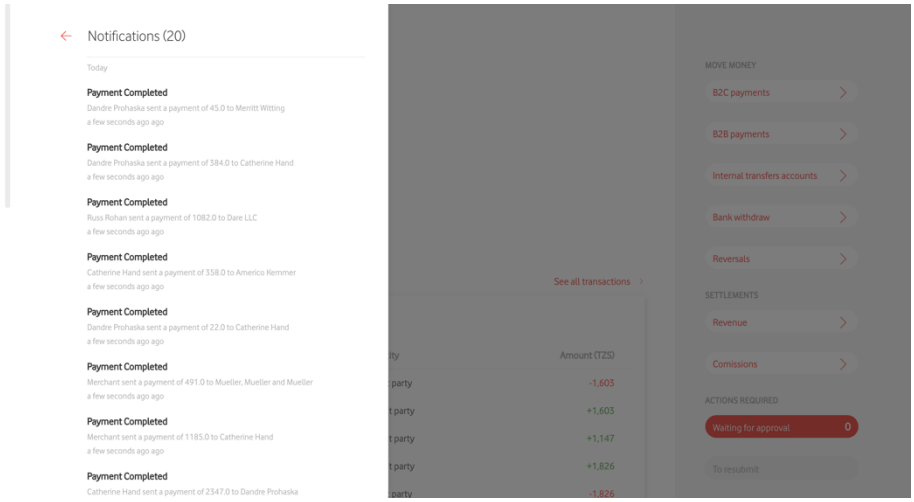

Figure 74 - Merchant notifications

The report area in the merchant portal was also changed; it is no longer customizable. The decision to remove support for customization was based in the fact that it was seldom used by merchants (Figure 75).,

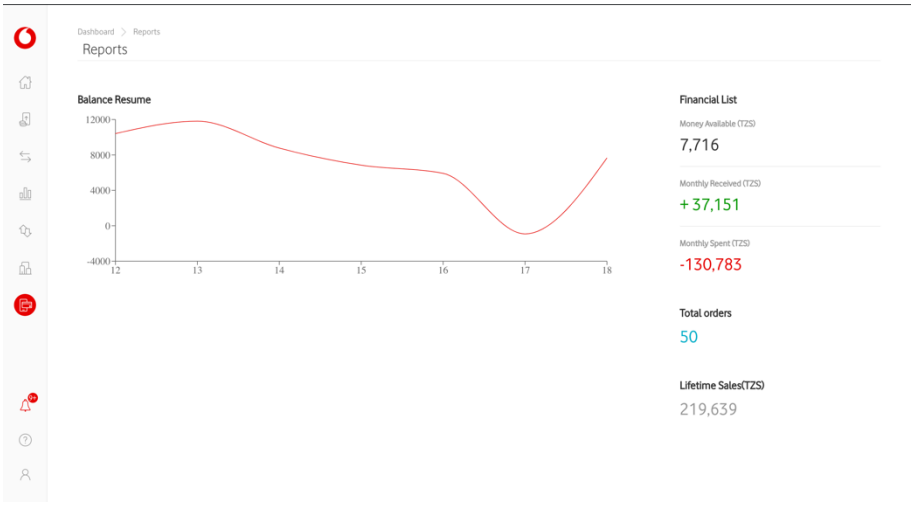

Figure 75 - Report area

Finally, some changes were also done to the checkout portal, mostly at the level of the UX, since the new approach assumes integrations with the Android application. The user is redirected to the checkout portal (Figure 76), and introduces the phone number as previously, in case the user is already registered as an M-Pesa mobile application user he/she receives a push notification, as in Figure 62.

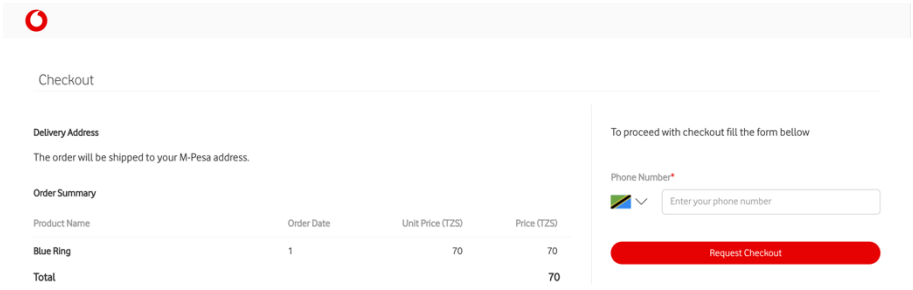

Figure 76 - Checkout portal

The portal shows the status of the operations and tells the user to finish the transaction on her/his application, as represented in Figure 77. If the user does not use the M-Pesa application, she/he will instead receive an OTP as showed previously.

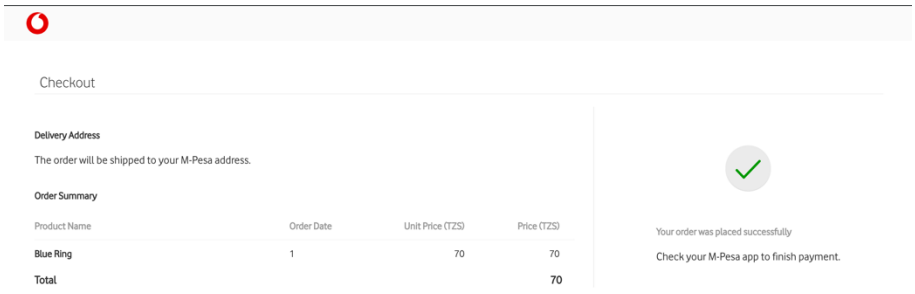

Figure 77 - Checkout portal suggesting user to check application

7. Conclusion

In summary, all the work carried on during the nine months of the internship took place at WIT Software. The work focused on the development of an online payment system to allow M-Pesa users to purchase on online stores and pay directly from their M-Pesa wallet. The developed platform -- Online Payments -- is comprised of a multi-module backend and five web applications. The Online Payments platform aims to reach the different roles -- merchants, developers and customers – that comprises its ecosystem. To respond to the needs of all its users, the following five portals were created: i) merchants; ii) developers; iii) checkout; iv) documentation; and v) demo. The merchant portal aims to help merchants manage their businesses and keep track of orders. The developer portal along with the demo and documentation portal aims to delivery documentation, tools and a test environment for developers to create, develop and test their applications, to iron out bugs and defects at development time, allowing for a smooth deployment. Finally, the checkout portal redirects customers to conclude the checkout process.

One of the main concerns about the developed product was to deliver a good user experience to the customer at checkout. For this purpose, we received the precious guidance of the WIT Design Team, which made a UX/UI review of an early version of the project. The feedback of the WIT Design Team led to a fully renovated user interface, that changed the overall user experience and the addition of functionalities in the customer M-Pesa applications.

Overall, the project was quite ambitious and also very interesting. Indeed, this project is a part of M-Pesa, itself a mega-project that can be, for some populations, a true life enhancer, since it aims to provide financial services to regions where banking might not be able to reach everyone. This contributed to the appeal and challenge of the project.

From a personal perspective, the internship experience at WIT Software was incredibly pleasant and gave me the opportunity of working with very skilled people and get in touch with new technologies as well as work methodologies.

Finally, we believe that main goals of the internship were accomplished, since the developed prototype has demonstrated that the proposed problem can be solved.

7.1. Future Work

There is always space for improvement. This project is no exception to the rule, as there are many things that could be improved in the future to strengthen this solution. Next, we provide some possible future work.

In terms of data persistence and despite the effort and concern in the usage of in-memory databases, there still is much progress that can be achieved. Online Payments is a platform that can reach a huge number of users, thus requiring a fine tuned and optimized database to allow for large scalability. Optimization could be obtained via the traditional routes, such as the creation of indexes in tables to increase the speed of the queries. Performance-wise, it is also important to focus on the removal of the automatically named queries in more complex queries. This kind of queries allow for a more straightforward integration, but in a production environment, they can cause serious trouble because they simply lack optimization. Another useful improvement for DB performance could be the addition of a NoSQL Database to store high volume and non-relational data like the one used to generate statistics.

Although the above suggested improvements could yield an increase in performance, there are other improvements like security questions that need to be addressed. There is still some sensible data in the database that require solid encryption. Another paramount question regarding security is the usage of the JWTs as an authentication mechanism. Indeed, even after the user has logout, it is still valid to login again while it has not expired. To avoid the JWT token from being stolen and used by third party without permission, it should be stored in a “blacklist of already logged out” tokens.

Another significant improvement would be the creation of an SDK for different programming languages that would allow online stores to integrate the Online Payments API easily.

Finally, in addition to the mentioned previously, all of it would not be efficient without the proper testing and quality assurance. For that would be mandatory to add automated tests to warrant the platform quality.

Despite being rather simple improvements, the above identified problems with the appropriated attention and solution could be significant improvements to this platform.

Bibliography

[1] C. Hodgson, “The world’s unbanked population, in 6 charts - Business Insider,” 2017. [Online]. Available: https://www.businessinsider.com/the-worlds-unbanked-population-in-6-charts-2017-8. [Accessed: 04-Feb-2019].

[2] B. Masamila, “STATE OF MOBILE BANKING IN TANZANIA AND SECURITY ISSUES,” *Int. J. Netw. Secur. Its Appl.*, vol. 6, no. 4, 2014.

[3] “New Global Findex: 69 Percent of Global Population Is Banked | Center for Financial Inclusion.” [Online]. Available: https://www.centerforfinancialinclusion.org/new-global-findex-sixty-nine-percent-of-global-population-is-banked. [Accessed: 24-Sep-2019].

[4] A. Holst, “Number of mobile cellular subscriptions per 100 inhabitants in Tanzania from 2000 to 2017.” [Online]. Available: https://www.statista.com/statistics/510624/mobile-cellular-subscriptions-per-100-inhabitants-in-tanzania/.

[5] N. Hughes and S. Lonie, “M-PESA: Mobile Money for the ‘Unbanked’ Turning Cellphones into 24-Hour Tellers in Kenya,” *Innov. Technol. Governance, Glob.*, vol. 2, no. 1–2, pp. 63–81, Apr. 2007.

[6] S. Edwards and S. Johnson, *Mobile Banking: The Impact of M-Pesa in Kenya*, vol. Publisher. University of Chicago Press.

[7] “WiT Software.” [Online]. Available: https://www.wit-software.com/company/why-wit/. [Accessed: 04-Jul-2019].

[8] “PayPal.” [Online]. Available: https://www.paypal.com/.

[9] “Venmo.” [Online]. Available: https://venmo.com/. [Accessed: 12-Nov-2018].

[10] “Pesapal.” [Online]. Available: https://www.pesapal.com/. [Accessed: 08-Feb-2019].

[11] “JumboPay.” [Online]. Available: https://www.myjumbopay.cm/.

[12] “Alipay.” [Online]. Available: https://intl.alipay.com/. [Accessed: 12-Nov-2018].

[13] J. (Business I. Heggestuen, “Alipay Overtakes PayPal As The Largest Mobile

Payments Platform In The World," 2014. [Online]. Available: https://www.businessinsider.com/alipay-overtakes-paypal-as-the-largest-mobile-payments-platform-in-the-world-2014-2.

[14] "Skrill." [Online]. Available: https://www.skrill.com/pt/. [Accessed: 12-Nov-2018].

[15] "Amazon Pay." [Online]. Available: https://pay.amazon.com/us. [Accessed: 12-Nov-2018].

[16] "Trustly." [Online]. Available: https://trustly.com.

[17] "Stripe." [Online]. Available: https://stripe.com/pt. [Accessed: 12-Nov-2018].

[18] "WePay." [Online]. Available: https://go.wepay.com/. [Accessed: 12-Nov-2018].

[19] "2Checkout." [Online]. Available: https://www.2checkout.com/. [Accessed: 12-Nov-2018].

[20] "MBWay." [Online]. Available: https://www.mbway.pt.

[21] M. Joseph, "M-Pesa: the story of how the world's leading mobile money service was created in Kenya," 2017. [Online]. Available: https://www.vodafone.com/content/index/what/technology-blog/m-pesa-created.html. [Accessed: 15-Jul-2019].

[22] Safaricom, "AliExpress M-PESA," 2019.

[23] PayPal, "What is PayPal Mobile Money Service with M-PESA?" [Online]. Available: https://www.paypal.com/ke/smarthelp/article/what-is-paypal-mobile-money-service-with-m-pesa-faq3891.

[24] "Angular." [Online]. Available: https://angular.io/.

[25] "React." [Online]. Available: https://reactjs.org/.

[26] "Getting Started · Securing a Web Application." [Online]. Available: https://spring.io/guides/gs/securing-web/. [Accessed: 11-Feb-2019].

[27] "Oracle." [Online]. Available: https://www.oracle.com/database/.

[28] "Hazelcast." [Online]. Available: https://hazelcast.com/. [Accessed: 28-Apr-2019].

[29] “Android.” [Online]. Available: https://developer.android.com/.

[30] “Jenkins.” [Online]. Available: https://jenkins.io/.

[31] J. Houser, “Combining Multiple Angular Applications into a Single One,” 2018. [Online]. Available: https://medium.com/disney-streaming/combining-multiple-angular-applications-into-a-single-one-e87d530d6527.

Appendices

Appendix A

Segurança no Spring Boot

Introdução

Nos dias que decorrem a segurança é um aspeto de grande relevância. É imperativo tentar garantir que as soluções desenvolvidas por nós sejam cada vez mais seguras.

Neste artigo iremos focar o tema Segurança no Spring Boot e entender de que forma podemos proteger os nossos *Endpoints* de utilizadores não autorizados, seja por não estarem autenticados ou por não terem privilégios para tal, através da autenticação dos utilizadores por um *token* JWT. Ainda veremos também como é possível persistir informações cruciais no *token* sobre o utilizador com sessão autenticada.

O artigo assume que o leitor tem conhecimentos básicos em Spring Boot. Para os leitores menos familiarizados com esta *framework* recomenda-se a leitura do artigo "API Rest com Spring Boot (parte 1)" (Revista Programar nº54) e a sua continuação "API Rest com Spring Boot (parte 2)" (Revista Programar nº55).

O que é um Token JWT?

Um JSON Web Token, ou vulgo JWT, é uma maneira compacta e segura de transmitir informação entre duas entidades em forma de objeto JSON. Ou seja, é um mecanismo de autenticação sem estado (*stateless*), visto que nunca é guardado em memoria. Um exemplo de uma situação em que pode ser usado é na autenticação, uma vez que o utilizador efetue o *login*, cada pedido subsequente irá incluir o JWT, permitindo assim que o utilizador continue acedendo serviços e recursos que só são liberados com o tal *token*.

A estrutura do JWT é separada por pontos (.) e dividida em 3 partes, sendo elas:

- **Cabeçalho**
- **Payload**
- **Assinatura**

Obedecendo à seguinte estrutura:

```
Cabeçalho.Payload.Assinatura
```

Listagem 1: Estrutura do token JWT

Cabeçalho

O cabeçalho é constituído por duas partes, sendo elas o tipo de *token* e tipo de algoritmo de HASH utilizado. A estrutura do objeto JSON é encriptado através da utilização do Base64Encoded. O formato do cabeçalho é o seguinte:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Listagem 2: Estrutura do cabeçalho

Este será a primeira parte do JSON codificado para formar o JWT.

Payload

O Payload forma a segunda parte do JWT, que por sua vez contém os Claims. Os Claims são informações sobre uma entidade, normalmente, o utilizador autenticado.

Existem três tipos de Claims: reservadas, públicas e privadas. As reservadas são um conjunto já predefinido que não é obrigatório, mas cuja utilização é recomendada pois são utilizados na validação dos tokens. São exemplos, o iss (emissor), exp (data de validade) e sub (sujeito).

As *claims* do tipo públicas podem ser definidas livremente e utilizadas para consulta pública, mas de forma a evitar colisão, estas devem ser definidas no repositório IANA[1] ou então como URIs que contenham um *namespace* resistente a colisão (como o nome de domínio). Finalmente, as *claims* privadas são customizadas, sendo empregues para o envio de dados entre duas entidades e geralmente contem dados sensíveis.

Um exemplo de um *Payload* é mostrado na Listagem 3. Este é constituído pelas *claims* reservadas, sub, a entidade a quem o token pertence, o iss, entidade que emitiu o token, o iat, o timestamp de quando o token foi criado e por fim o exp, timestamp de quando o token irá

[1] https://www.iana.org/assignments/jwt/jwt.xhtml

expirar. É ainda composto pela claim privada "scopes" que contem a informação relativa ao role do utilizador.

```
{
  "sub": "Ruben",
  "scopes": [
    {
      "authority": " ADMIN"
    }
  ],
  "iss": "http://article-spring-boot.com",
  "iat": 1508607322,
  "exp": 1508625322
}
```

Listagem 3: Estrutura do Payload

Assinatura

A assinatura permite validar que o *token* não foi alterado durante a comunicação entre as duas entidades. Para gerar a assinatura deve ser utilizado como entrada para o algoritmo de *hash*, o Cabeçalho e o Payload. O algoritmo de *hash* a ser empregue encontra-se definido no cabeçalho. Por exemplo, o uso do algoritmo HS256 (HMAC SHA256) é indicado através do cabeçalho indicado na Listagem 4.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Listagem 4: Assinatura do JWT

Para além de possibilitar validar a integridade da mensagem, e portanto detetar se a mesma foi alterada ou não durante a transmissão, a assinatura permite ainda verificar a autenticidade do emissor.

Utilização do JWT

As rotas protegidas no servidor verificam a existência de um JWT valido no campo *Authorization* no cabeçalho do pedido HTTP e apenas se este estiver presente, será permitido o acesso do utilizador as rotas protegidas. Desta forma, sempre que o utilizador quiser aceder a uma rota protegida deve enviar o JWT. Para o efeito, o cabeçalho HTTP deve ter um campo *Authorization*, com o conteúdo similar ao mostrado na Listagem 5.

```
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJBbGV4MTIzIi
wic2N.v9A80eU1VDo2Mm9UqN2FyEpyT79IU
mhg
```

Listagem 5: Exemplo do campo *Authorization* com um *token JWT*

Ferramentas a utilizar

De forma a atingir o nosso objetivo serão utilizadas as seguintes ferramentas:

- **JAVA**: linguagem de programação a utilizar uma vez que Spring Boot é baseado em Java;
- **MySQL**: SGBD empregue para persistir os dados;
- **SQLDeveloper** (opcional): ferramenta que permite realizar pedidos SQL de forma a consultar ou inserir dados na base de dados;
- **Tomcat:** servidor padrão utilizado pelo Spring Boot para correr a aplicação;
- **Maven**: gestor de dependências que iremos utilizar;
- **IntelliJ** (opcional): IDE utilizado para programar;
- **Postman:** empregue para efetuar chamadas as APIs, possibilitando a interpretação dos resultados.

Estrutura do Projeto

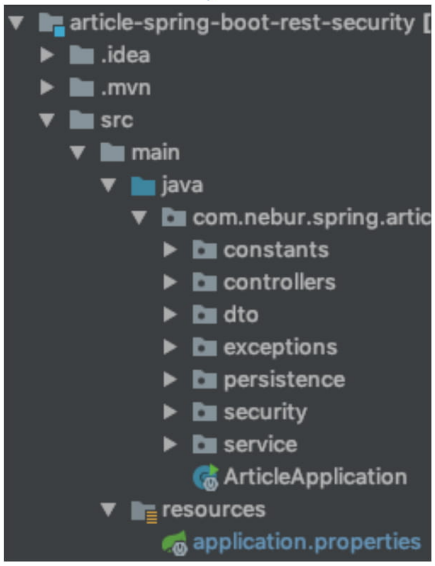

Figura 1 - Estrutura do Projeto

Na Figura 1 é possível encontrar a estrutura do Projeto, sendo que esta se divide pelos seguintes *packages*:

- **Constants**: package que contém o ficheiro com todas as declarações de variáveis constantes;
- **Controllers:** package que contém todos os controladores com os *Endpoints REST* criados por nós;
- **Dtos:** contém todos os *Data Transfer Objects* que são utilizados para transportar os dados para fora da aplicação Spring Boot;
- **Exceptions:** contém todas as exceções passíveis de serem criadas pelo projeto;
- **Persistence:** onde é possível localizar todas as classes de objetos e os repositórios dos mesmos;
- **Security:** onde se encontram todas as configurações de segurança e mecanismos de autenticação necessários;
- **Service**: *package* que guarda todos os serviços e respetivas implementações;

Autenticação na Aplicação

A segurança numa aplicação de Spring Boot que requeira a autenticação do utilizador pode ser obtida através da utilização e configuração das seguintes classes. Cada uma destas classes tem uma função especifica e será explicada ao longo deste artigo:

- **CORSFilter**: classe responsável pela configuração do filtro do *CORS*. Nesta classe é possível indicar as origens de onde se aceitam pedidos, que campos são permitidos no cabeçalho dos pedidos HTTP, bem como os tipos de pedidos que são permitidos chamar (ex.: GET, POST...).
- **JwtAuthenticationFilter**: classe responsável por filtrar todos os pedidos HTTP e verificar a existência de um JWT *token* no cabeçalho, procedendo à autenticação do utilizador consoante a validade do *token*.
- **JwtAuthenticationEntryPoint:** responsável por apanhar as exceções e erros lançados pela classe **JwtAuthenticationFilter** e enviar mensagem de erro ao utilizador sempre que este não estiver autorizado a aceder a algo.
- **JwtTokenUtil**: classe que funciona como uma biblioteca. Contém todas as funções associadas a geração, validação e extração de *claims* de um *token*.
- **WebSecurityConfig:** congrega toda a configuração de segurança do Spring Boot, nomeadamente, como é que os utilizadores são autenticados, bem como indicar os *Endpoints* que são/não são protegidos pelos métodos de segurança implementados, entre outras configurações.

Mecanismo de Autenticação

Como referido anteriormente a autenticação do utilizador é realizada através do envio do *token* JWT no cabeçalho do pedido HTTP. A classe **JwtAuthenticationFilter** é a responsável pela verificação da presença do mesmo e pela autenticação do utilizador em contexto.

Esta classe estende *OncePerRequestFilter* o que garante que este filtro apenas é chamado uma vez por pedido. A função *doFilterInternal* verifica se existe o campo *Authorization* no cabeçalho do pedido HTTP e se este vem preenchido com o *Bearer*.

```
@Override
protected void
doFilterInternal(HttpServletRequest req,
HttpServletResponse res, FilterChain
chain) throws IOException,
ServletException {
    //get HTTP request header
    String header =
req.getHeader(HEADER_STRING);

    String username = null;
    String authToken = null;

    //check if header exist and stats
with the defined TOKEN PREFIX -> Bearer
    if (header != null &&
header.startsWith(TOKEN_PREFIX)) {
```

Listagem 6: Obtenção do cabeçalho do pedido e verificação da existência do *token*

No caso de o cabeçalho ter o campo *Authorization* preenchido, a função remove a palavra *Bearer* isolando assim o *token* JWT. Após isto, tenta extrair o nome do utilizador do *token*. Nesta fase a autenticidade do *token* e a sua validade são garantidas, pois caso o *token* tenha sido assinado com uma outra chave que não a definida neste servidor ao tentar desencriptar e extrair o *username* será gerado uma exceção. O mesmo sucederá caso o *token* já tenha expirado. Caso a autenticação falhe, é enviado ao utilizador uma reposta com o estado 400 e a mensagem *Unauthorized*.

```
//remove the word Bearer in order to
isolate the JWT Token
authToken =
header.replace(TOKEN_PREFIX,"");
try {
    //try to decrypt the JWT TOKEN with
the key with which it was signed
    username =
jwtTokenUtil.getUsernameFromToken(authTok
en);

    //if by any reason it fail will
generate an exception and it will be
throw an Exception
    //where will be possible to find JWT
Tokens signed with different keys or
tokens that already expired
} catch (IllegalArgumentException e) {
    throw new InvalidTokenException();
}
```

Listagem 7: Extração do token e tentativa de extração do nome de utilizador do mesmo

Na eventualidade do passo anterior correr bem e consequentemente não ter sido lançada qualquer exceção, é verificado se foi mesmo extraído um *username* do *token*. Note-se que através do *ClientService* que implementa a classe *UserDetailsService* é possível aceder a base de dados e desta forma obter os dados do utilizador através do respetivo *username*.

```
//check if it was possible to retrieve a
username from the JWT Token and if there
is not already a authentication set
if (username != null &&
SecurityContextHolder.getContext().getAut
hentication() == null) {

    //clientService implements
UserDetailsService which will allow us to
retrieve user credentials from Database
    UserDetails userDetails =
clientService.loadUserByUsername(username
);
```

Listagem 8: Verificação que o nome foi extraído com sucesso e consulta das credencias do utilizador na base de dados

Posteriormente, os dados do utilizador são comparados mais uma vez aos do JWT *token* e caso estejam em concordância, parte-se para a autenticação e persistência da mesma no Spring em memória através dos mecanismos de segurança do Spring Security.

```
//check if data retrieved from token
matches with the one found in Database
if (jwtTokenUtil.validateToken(authToken,
userDetails)) {
    //authentication and persist of user
with Spring Security to keep him
authenticated
    UsernamePasswordAuthenticationToken
authentication = new
UsernamePasswordAuthenticationToken(userD
etails, null,
Collections.singletonList(new
SimpleGrantedAuthority("USER")));
    authentication.setDetails(new
WebAuthenticationDetailsSource().buildDet
ails(req));

SecurityContextHolder.getContext().setAut
hentication(authentication);
}
```

Listagem 9: Persistência do utilizador no mecanismo de autenticação do Spring Boot

O ***JwtTokenUtil***, Listagem 10, é uma classe de métodos uteis que visam simplificar e reduzir o volume de código necessário à a interação com os *tokens*.

```
@Component
public class JwtTokenUtil implements
Serializable {

    String getUsernameFromToken(String
token) {
        return getClaimFromToken(token,
Claims::getSubject);
    }

    private Date
getExpirationDateFromToken(String token)
{
        return getClaimFromToken(token,
Claims::getExpiration);
    }

    private <T> T
getClaimFromToken(String token,
Function<Claims, T> claimsResolver) {
        final Claims claims =
getAllClaimsFromToken(token);
        return
claimsResolver.apply(claims);
    }
```

```
    private Claims
getAllClaimsFromToken(String token) {
        return Jwts.parser()

.setSigningKey(SIGNING_KEY)
                .parseClaimsJws(token)
                .getBody();
    }

    private Boolean isTokenExpired(String
token) {
        final Date expiration =
getExpirationDateFromToken(token);
        return expiration.before(new
Date());
    }

    public String generateToken(Client
client) {
        return
doGenerateToken(client.getUsername());
    }

    private String doGenerateToken(String
subject) {

        Claims claims =
Jwts.claims().setSubject(subject);
        claims.put("scopes",
Collections.singletonList(new
SimpleGrantedAuthority("USER")));

        return Jwts.builder()
                .setClaims(claims)
                .setIssuer("article-
spring-boot")
                .setIssuedAt(new
Date(System.currentTimeMillis()))
                .setExpiration(new
Date(System.currentTimeMillis() +
ACCESS_TOKEN_VALIDITY_SECONDS*1000))

.signWith(SignatureAlgorithm.HS256,
SIGNING_KEY)
                .compact();
    }

    Boolean validateToken(String token,
UserDetails userDetails) {
        final String username =
getUsernameFromToken(token);
        return
username.equals(userDetails.getUsername()
) && !isTokenExpired(token);
    }
}
```

Listagem 10: Classe JwtTokenUtil

Configuração de Segurança

Nesta secção encontra-se as nossas configurações de segurança. Na classe **WebSecurityConfig** a anotação @EnableGlobalMethodSecurity ativa a funcionalidade de segurança e permite que os as funções sejam anotadas com anotações como a @Secured de forma a tornar possível a especificação da autenticação por *Roles* (User, Admin, etc..) ao nível das funções.

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnable
d = true)
public class WebSecurityConfig extends
WebSecurityConfigurerAdapter {
```

Listagem 11: Anotações da classe WebSecurityConfig

São ainda injetadas as dependências *UserDetailsService* e o nosso *JwtAuthenticationEntryPoint*. A primeira dependência referenciada possui a indicação do nosso *ClientService* de forma a ser possível aceder às credenciais do utilizador na base de dados.

```
@Resource(name = "clientService")
private UserDetailsService
userDetailsService;

private JwtAuthenticationEntryPoint
unauthorizedHandler;

@Autowired
public
WebSecurityConfig(JwtAuthenticationEntryP
oint unauthorizedHandler) {
    this.unauthorizedHandler =
unauthorizedHandler;
}
```

Listagem 12: Atributos e construtor da classe WebSecurityConfig

Ainda são injetados mais dois *Beans* que têm como objetivo a transformação em *hash* da palavra-senha antes de a persistir na base de dados e a instanciação do nosso filtro de pedidos que será abordado posteriormente. Note-se que o algoritmo *bcrypt* é um dos recomendados para o armazenamento de *hash* derivadas de palavras-senhas.

```
@Bean
public JwtAuthenticationFilter
authenticationTokenFilterBean() {
    return new JwtAuthenticationFilter();
}

@Bean
public BCryptPasswordEncoder encoder(){
    return new BCryptPasswordEncoder();
}
```

Listagem 13: Atributos e construtor da classe WebSecurityConfig

É através do método configure que se especifica como a aplicação se deve comportar perante o acesso aos

respetivos *Endpoints*, indicando ainda *Endpoints* cujo acesso pode ser feito sem *token* de autenticação.

```
@Override
protected void configure(HttpSecurity
http) throws Exception {
    http.cors().and().csrf().disable().
            authorizeRequests()

.antMatchers("/api/v1/token/**",
"/api/v1/client/signup").permitAll()
            .anyRequest().authenticated()
            .and()

.exceptionHandling().authenticationEntryP
oint(unauthorizedHandler).and()

.sessionManagement().sessionCreationPolic
y(SessionCreationPolicy.STATELESS);

http.addFilterBefore(authenticationTokenF
ilterBean(),
UsernamePasswordAuthenticationFilter.clas
s);
}
```

Listagem 14: Atributos e construtor da classe WebSecurityConfig

```
        .authorizeRequests()
        .antMatchers("/api/v1/token/**",
"/api/v1/client/signup").permitAll()
```

Listagem 15: Definição dos Endpoints que não necessitam de Token

É ainda através do método configure que são especificados os pedidos que requerem obrigatoriamente um *token* de autorização. O configure permite ainda a definir a metodologia de tratamento das exceções, que no caso do exemplo em apreço (Listagem 16), são dirigidas para a classe ***JwtAuthenticationEntryPoint***. Finalmente, através do método *http.addFilterBefore*, pode ser indicado a entidade responsável pela autenticação dos pedidos HTTP, ou seja, o nosso filtro.

```
.anyRequest().authenticated()
.and()
.exceptionHandling().authenticationEntryP
oint(unauthorizedHandler)

http.addFilterBefore(authenticationTokenF
ilterBean(),
UsernamePasswordAuthenticationFilter.clas
s);
```

Listagem 16: Definição da segurança para todos os outros URLs que sejam chamados

Endpoints a testar

Esta secção contempla o caso de uso de três *Endpoints* com finalidades distintas. Assume-se que o servidor encarregue de executar a nossa aplicação localmente, está configurado para o porto 8080. Deste modo, o começo do endereço será http://localhost:8080. Note-se, por uma questão de coerência, segue-se a nomenclatura do artigo "API Rest com Spring Boot (parte 1)" (Revista Programar nº54) para criação e identificação de Endpoints.

No primeiro *Endpoint*, Listagem 17, observa-se a função que permite o registo de um novo utilizador. Note-se que no ficheiro **WebSecurityConfig.java** configura-se este Endpoint de forma a que o utilizador não precise de autenticação para o invocar, permitindo assim que o utilizador se registe sem a necessidade de um *Token* no campo *Authorization*.

```
// EndPoint #1
@RequestMapping(value = "/signup", method
= RequestMethod.POST)
public ResponseEntity<Client>
signupClient(@Valid @RequestBody
ClientDto clientDto) {
    Client client =
clientService.save(clientDto);
    return
ResponseEntity.status(HttpStatus.OK).body
(client);
}
```

Listagem 17: EndPoint #1 - Registo

O segundo Endpoint (Listagem 18) funciona como o *login*: o utilizador envia as respetivas credencias de *login*, *username* e palavra-senha. Caso as credencias sejam válidas, é gerado um *token* JWT que é devolvido ao utilizador.

```
// EndPoint #2
@RequestMapping(value = "/generate",
method = RequestMethod.POST)
public ResponseEntity<String>
register(@RequestBody LoginDto loginDto)
throws AuthenticationException {


authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(login
Dto.getUsername(),
loginDto.getPassword()));

    final Client client =
clientService.findByUsername(loginDto.get
Username());
    final String token =
jwtTokenUtil.generateToken(client);
    return
ResponseEntity.status(HttpStatus.OK).body
(token);
}
```

Listagem 18: EndPoint #2 - Login

O terceiro *Endpoint* (Listagem 19) ilustra como o servidor, através do *token* facultado pelo cliente, pode extrair a informação relativa ao utilizador, autenticá-lo e persisti-lo em memória e ir buscar os seus dados à base de dados.

```
// EndPoint #3
@RequestMapping(value = "/whoami", method
= RequestMethod.GET)
public ResponseEntity<String> whoami() {
    Authentication auth =
SecurityContextHolder.getContext().getAut
hentication();
    if(auth == null) {
        throw new
InvalidTokenException();
    }

    UserDetails userDetails =
(UserDetails) auth.getPrincipal();
    Client client =
clientService.findByUsername(userDetails.
getUsername());

    return
ResponseEntity.status(HttpStatus.OK).body
(client.getUsername());
}
```

Listagem 19: EndPoint #3 Verificar quem é o utilizador autenticado

Todos os restantes *Endpoints* – *Endpoints* 4 até *Endpoints* 8 – (Listagem 20) a serem referenciados já foram abordados no artigo anterior (Revista Programar nº 54), mantendo-se inalterados.

```
// EndPoint #4
@RequestMapping(value = "/{id}", method =
RequestMethod.GET)
public ResponseEntity<Client>
getByClientId(@PathVariable(value = "id")
long clientId) {
    Client client =
clientService.findById(clientId);
    return
ResponseEntity.status(HttpStatus.OK).body
(client);
}

// EndPoint #5
@RequestMapping(value = "/all", method =
RequestMethod.GET)
public ResponseEntity<List<Client>>
getAllClients() {
    List<Client> clients =
clientService.findAll();
    return
ResponseEntity.status(HttpStatus.OK).body
(clients);
}

// EndPoint #6
@RequestMapping(value = "/nif", method =
RequestMethod.GET)
public ResponseEntity<Client>
getByClientNIF(@RequestParam(value =
"nif") String nif) {
    if (!nif.matches("[0-9]+") ||
nif.length() != 9) {
        throw new
InvalidNIFException(nif);
    }
    Client client =
clientService.findByNIF(nif);
    return
ResponseEntity.status(HttpStatus.OK).body
(client);
}

// EndPoint #7
@RequestMapping(value = "/update", method
= RequestMethod.PUT)
public ResponseEntity<Client>
updateClient(@Valid @RequestBody
ClientDto clientDto) {
    Client client =
clientService.update(clientDto);
    return
ResponseEntity.status(HttpStatus.OK).body
(client);
}

// EndPoint #8
@RequestMapping(value = "/delete/{id}",
method = RequestMethod.DELETE)
public ResponseEntity<String>
deleteClient(@PathVariable(value = "id")
long clientID) {
    clientService.deleteById(clientID);
    return
ResponseEntity.status(HttpStatus.OK).body
("Client deleted successfully.");
}
```

Listagem 20: EndPoint #4 a #8

Testes a API

Para testar a aplicação são empregues duas ferramentas: o **SQLDeveloper** [2] e o **Postman** [3]. O SQLDeveloper permite verificar os dados que se encontram guardados na base de dados, enquanto o

[2] https://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html

[3] https://www.getpostman.com/downloads/

Postman possibilita testar os *Endpoints* da nossa aplicação, observando-se como ela se comporta.

Começamos por executar uma simples consulta SQL à base de dados para comprovar que a tabela Clientes se encontra vazia.

Figura 2 - Consulta de todos os clientes na base de dados

De acordo com o ficheiro de configuração **WebSecurityConfig** ficaram configurados dois *Endpoints* que prescindem de qualquer tipo de autorização. Esses *Endpoints* são o **/v1/cliente/signup** e o **/v1/cliente/login, que possibilitam, respetivamente, o registo e, logo de seguida, o** *login*.

Começamos então por registar um novo cliente enviando- um objeto em JSON para o servidor com os campos relativos ao utilizador. Como é possível verificar, Figura 3, o servidor respondeu que o utilizador foi registado com sucesso, devolvendo o objeto do utilizador com a palavra-senha encriptada.

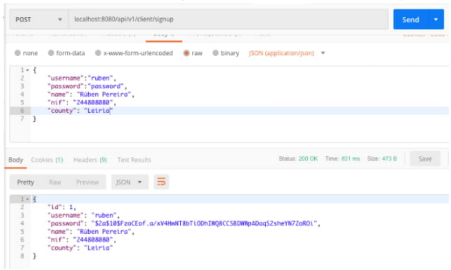

Figura 3 - Registo de um cliente

Assim, nesta fase, já existe um utilizador na base de dados o que possibilita chamar o ***Endpoint* #4**, *endpoint* que permite efetuar uma operação de GET especificando o ID do utilizador. O objetivo deste teste é observar o comportamento da aplicação dado que ainda não existe *token* de autorização a ser passado através do cabeçalho dos nossos pedidos HTTP.

Ao executar este pedido obtém-se a resposta do servidor com a mensagem *Unauthorized* visto que o *Endpoint* pretendido é protegido, e não foi facultado um *token* que nos identificasse.

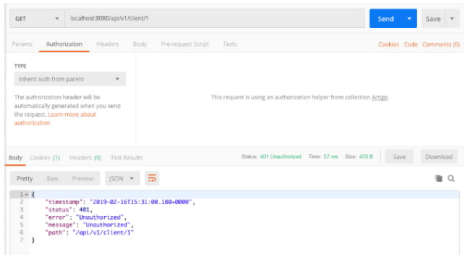

Figura 4 - Consulta do utilizador com o ID 1 sem token

Iremos então identificarmo-nos junto do servidor, efetuando assim *login* para passarmos a ter um JWT *token*. Para o efeito, procede-se ao envio de um objeto JSON com o *username* e palavra-senha, com o servidor a verificar que nos encontramos efetivamente na base de dados, devolvendo um token futuramente ser empregue como identificador nas interações subsequentes com o servidor.

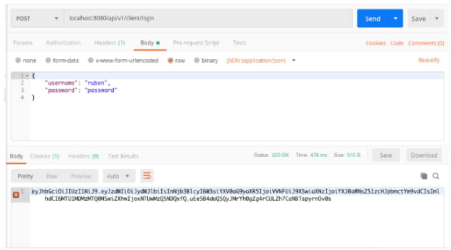

Figura 5 - Login e receção do token

Este token deve ser adicionado ao campo *Authorization* em todos os pedidos HTTP, no Postman, que requerem uma autorização para serem executados, tendo em consideração que *Authorization Type* a selecionar é o "***Bearer Token***", Figura 6.

Figura 6 - Adição do token ao cabeçalho dos pedidos no Postman

Para confirmar que os nossos pedidos já se encontram devidamente autorizados, efetua-se a chamada ao

Endpoint **/v1/cliente/whoami**. O acesso a *endpoint* demonstra que é possível através do JWT *token* identificar um utilizador e consultar os respetivos dados existentes na base de dados.

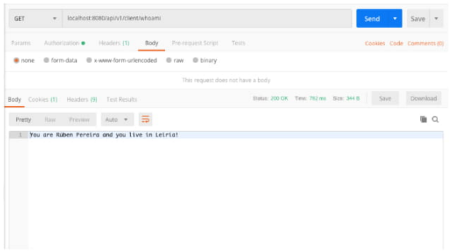

Figura 7 - Verificação de que o servidor sabe quem esta a fazer o pedido com base no token

Voltando agora a chamar o *endpoint* **/v1/cliente/{id}** com o campo ***Authorization*** devidamente preenchido, confirma-se que já se tem acesso aos dados do utilizador.

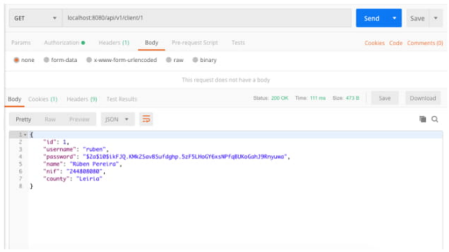

Figura 8 - Consulta do utilizador com o ID 1 com token

Nota final

Este artigo pretendeu passar algumas das noções básicas da segurança no *Spring Boot*. Procurou-se demonstrar como é possível proteger os Endpoints de utilizadores não autorizados, garantindo que apenas utilizadores com os devidos privilégios tenham o devido acesso.

Os leitores interessados encontrarão o código completo do artigo no repositório GitHub de Rúben Pereira: https://github.com/Nebur96/article-spring-boot-rest-security

Bibliografia

(2019, 2 12). Retrieved from JSON Web Tokens - jwt.io: https://jwt.io/

(2019, 2 13). Retrieved from Maven: https://maven.apache.org/

(2019, 2 13). Retrieved from MySQL: https://www.mysql.com/

(2019, 2 13). Retrieved from Postman | API Development Environment: https://www.getpostman.com/

(2019, 2 13). Retrieved from Spring Boot: https://spring.io/projects/spring-boot

Notas biográficas

Rúben Pereira é estudante do Mestrado em Engª Informática – Computação Móvel na Escola Superior de Tecnologia e Gestão (ESTG) do Instituto Politécnico de Leiria (IPLeiria) e licenciado em Engª Informática pela mesma instituição.

Patrício Domingues é professor do Departamento de Engª Informática na Escola Superior de Tecnologia e Gestão (ESTG) do Instituto Politécnico de Leiria (IPLeiria). Tem lecionado, entre outras, as disciplinas de Sistemas Operativos e Programação Avançada da Licenciatura em Engenharia Informática. É ainda docente da Pós-Graduação em Informática de Segurança e Computação Forense da ESTG/IPLeiria e do mestrado em Cibersegurança e Informática Forense (MCIF) do Politécnico de Leiria.

Appendix B

Proposta de Estágio

Mestrado em Engenharia Informática – Computação Móvel (MEI-CM)

Escola Superior de Tecnologia e Gestão

Instituto Politécnico de Leiria

EMPRESA/ORGANIZAÇÃO PARCEIRA: WIT SOFTWARE SA.

QUADRO A - DESCRIÇÃO DO ESTÁGIO

TEMA/TÍTULO	**Web Channel for a Mobile money system**
ÁREA DE ESTUDO	Mobile (Android ou iOS), Java, APIs, REST, Pagamentos Móveis
ÁREA DE ATIVIDADE ECONÓMICA	
OBJETIVOS	Desenvolver o protótipo de um sistema que permita a lojas online a integração com uma plataforma mobile de transferência de dinheiro
BREVE DESCRIÇÃO	

Nos mercados emergentes, apesar da existência de várias plataformas móveis orientadas para a transferência de dinheiro entre pessoas, há ainda uma área pouco explorada: a área de pagamento de compras online. O desenvolvimento de aplicações para esta área é muito apelativo, pelo facto de existir muito valor em transferências relativas a compras de bens ou serviços e de nestas zonas as pessoas não terem acesso a serviços bancários.

Este estágio insere-se num contexto em que se pretende a implementação de um protótipo de um sistema de pagamentos online para eCommerce.

No âmbito deste estágio, será necessário desenvolver um protótipo de um sistema seguro, fácil e rápido para que lojas online possam integrar para facilmente ativar um sistema de pagamentos.

Os use cases principais são:

1. Como consumidor, quero pagar a minha compra online
2. Como consumidor, quero receber uma notificação no meu telemóvel com os detalhes de pagamento
3. Como comerciante, quero saber que o pagamento foi efetuado com sucesso.
4. Como comerciante, quero listar os pagamentos feitos.
5. Como comerciante, quero devolver um pagamento.

Todos os desenvolvimentos efetuados serão documentados e analisados por uma equipa de segurança, que dará feedback sobre as melhores práticas de implementação.

Durante o estágio, o aluno terá o apoio da equipa de pagamentos móveis da WIT, que efetua desenvolvimento com processos já definidos.

Este sistema online deverá ser integrado numa app móvel que irá alterar o cliente da operação. A implementação poderá ser feita em Android ou iOS, à escolha do estagiário. No final deverá ser montada uma demonstração como prova de conceito.

Dada a relevância do tema, este estágio será estratégico para que se reforcem competências nesta área.

COMPETÊNCIAS REQUERIDAS	Conhecimentos básicos de criptografia e segurança online. Conhecimentos de Android ou iOS (Swift) Conhecimentos de Java, RESTfull interfaces, tomcat
COMPETÊNCIAS OBTIDAS AO LONGO DO ESTÁGIO	Experiência no desenvolvimento de software para uma app no mundo real com milhares de utilizadores por dia. Experiência na criação de software de modo a que possa ser testado de forma automática Experiência a estimar e planear tarefas Experiência a desenvolver uma solução segura

QUADRO B - TECNOLOGIAS ENVOLVIDAS NA REALIZAÇÃO DO ESTÁGIO

SISTEMAS OPERATIVOS	Android, Linux (O estagiário poderá, para desenvolvimento, utilizar Windows ou MacOS) Opcionalmente o aluno poderá optar por iOS (Swift)
FRAMEWORKS	Android SDK / iOS Swift
METODOLOGIAS DE DESENVOLVIMENTO	Agile: Scrum
FERRAMENTAS DE GESTÃO DE PROJETO	Redmine entre outras ferramentas internas da organização
OUTROS	

QUADRO C – PLANO DE TRABALHOS

PLANO DE TRABALHOS (9 MESES)		
ITENS DE AÇÃO (ACRESCENTAR LINHAS DE ACORDO COM A NECESSIDADE)	DATA ÍNICIO	DATA FIM
***Setup* e Análise de requisitos**	01/10/2018 (indicativo)	14/10/2018
Arquitetura e submissão da proposta à equipa de segurança	15/10/2018	06/11/2018
Construção de um protótipo funcional	09/11/2018	08/01/2019
Integração com Apps Mobile	11/01/2019	10/02/2019
BackOffice de consulta de transações	24/03/2019	04/05/2019
Testes de performance e escalabilidade	05/05/2019	02/06/2019
Documentação do projeto e relatório de estágio	03/06/2019	30/06/2019

QUADRO D – LOCAL DE TRABALHO

LOCALIDADE	Leiria
SÃO ESPERADAS DESLOCAÇÕES A CLIENTES/FORMAÇÃO NAS SEGUINTES LOCALIDADES	Coimbra
SÃO ESPERADOS PERÍODOS DE DESLOCAÇÕES DE (DIAS/SEMANAS/MESES)	Dia

QUADRO E – RECURSOS ALOCADOS AO ESTAGIÁRIO

COMPUTADOR DE MESA	Monitor + Teclado + Rato
COMPUTADOR PORTÁTIL	Sim
INTERNET	Sim
OUTROS	Serão disponibilizados todos os recursos de software e hardware necessários para a implementação do estágio
RECURSOS ACESSÍVEIS AO ESTAGIÁRIO NO LOCAL DE TRABALHO	
BAR	Copa onde poderá almoçar
REFEITÓRIO	Copa onde poderá almoçar, com fruta gratuita
MÁQUINA DE CAFÉ	Sim. Café gratuito
OUTROS	

QUADRO F – CONDIÇÕES PARA A REALIZAÇÃO DO ESTÁGIO

TEMPO CONCEDIDO AO ESTAGIÁRIO PARA	
ELABORAÇÃO DO RELATÓRIO DE ESTÁGIO (HORAS/SEMANA)	A definir com estagiário
CONTACTO COM O ORIENTADOR NO IPLEIRIA (HORAS/SEMANA)	A definir com orientador

QUADRO G – FORMAÇÃO INTEGRADA

ESTÁGIO CONTEMPLA FORMAÇÃO NA EMPRESA/ORGANIZAÇÃO (HORAS/DIAS)	O estagiário irá receber formação na WIT Academy (que inclui tópicos tais como: IMS, VoIP, SIP, iOS Programming, Android Programming, Arquitecturas de Software, Software Quality, Metodologias de Desenvolvimento de Software).
ESTÁGIO CONTEMPLA FORMAÇÃO EXTERIOR À EMPRESA (HORAS/DIAS)	A definir com estagiário

QUADRO H – CONDIÇÕES MATERIAIS

ESTÁGIO REMUNERADO (VALOR MENSAL)	O estágio será remunerado
APOIOS PARA DESLOCAÇÕES RESIDÊNCIA-TRABALHO	O estágio será remunerado
SUBSÍDIO DE ALIMENTAÇÃO	O estágio será remunerado
SEGUROS (ACIDENTES PESSOAIS, SAÚDE, ETC.)	Sim (seguro escolar)
OUTROS	

QUADRO I – SUPERVISOR NA EMPRESA/ORGANIZAÇÃO

NOME EMPRESA/ORGANIZAÇÃO	WIT Software, S.A.
MORADA POSTAL	Centro de Empresas de Taveiro Estrada de Condeixa 3045-508 Taveiro Coimbra, Portugal
NOME SUPERVISOR	Tiago Marto
MÓVEL SUPERVISOR	913716686
EMAIL SUPERVISOR	tiago.marto@wit-software.com

Sobre a Empresa:
A WIT desenvolve software para Operadores de Telecomunicações de vários continentes, tais como o Grupo Vodafone (Europa), Deutsche Telekom (Alemanha), Reliance Jio (Índia), KDDI, Softbank, NTT Docomo (Japão), Singtel (Singapura), Telstra (Australia), Unitel (Angola), Eir (Irlanda), Telecom Italia (Itália), Orange (França), Telefónica (Espanha), TeliaSonera (Suécia), Belgacom (Bélgica), Post Luxembourg (Luxemburgo), Bell (Canadá), Century Link (EUA) e Everything Everywhere (RU). O software desenvolvido pela WIT está presente em 42 países. A WIT tem escritórios em Portugal e Reino Unido e os seus centros de desenvolvimentos estão localizados em Coimbra, Porto, Leiria e Aveiro.

Sobre a orientação:
O acompanhamento do estágio será feito não só pelo orientador, mas também por um tutor técnico que dará ao aluno todo o apoio necessário. O Orientador define os requisitos do estágio, define as prioridades do Backlog e acompanha os resultados parciais do projecto. O Tutor dá todo o suporte técnico necessário, garante o cumprimento das tarefas e promove as meetings de acompanhamento do cumprimento dos objectivos.

QUADRO J – POSSIBILIDADE DE PERMANÊNCIA NA EMPRESA/ORGANIZAÇÃO

EXISTE A POSSIBILIDADE DE PERMANECER NA EMPRESA/ORGANIZAÇÃO NO FINAL DO ESTÁGIO	No final do estágio, será feita uma avaliação do estagiário e dos conhecimentos adquiridos. Se o resultado for positivo o estagiário será convidado para fazer parte da equipa de desenvolvimento.

Appendix C

Merchant portal functionalities comparison across several platform:

	PayPal	Alipay	Skrill	Venmo	AmazonPay	Trustly	Stripe	WePay	2Checkout
Wallet	x	?	x	-	?	?	x	?	x
Withdraw Money	x	?	x	-	?	?	x	?	
Send Money	x	?	x	-	?	?	x	?	
Request Money	x	?		-	?	?		?	
Link a Bank Account	x	?	x	-	?	?		?	
Link Bank Cards	x	?	x	-	?	?	x	?	
Create/Send Invoice	x	?		-	?	?	x	?	
List of Items	x	x		-	?	?	x	?	x
Address Book	x	?		-	?	?	x	?	
Analytics	x	?		-	?	?	x	?	
Disputes and Chargebacks	x	?		-	?	?	x	?	
Recurring Payments	x	?		-	?	?		?	
Mass Payment	x	?		-	?	?		?	
Buttons Generator	x	?		-	?	?		?	
API Settings	x	?		-	?	?	x	?	
Shipping Calculator	x	?		-	?	?		?	x
Shopping Cart Pushases	x	?		-	?	?		?	
Sales Tax	x	?		-	?	?		?	
Sales Optimizer	x	?		-	?	?		?	
List of Orders		x		-	?	?		?	x

Transaction List	x	x	x	-	?	?	x	?	x
CRM System	x	?		-	?	?	x	?	x
Webhooks		?		-	?	?		?	x
Integration with other Platforms		?		-	?	?	x	?	

Customer portal functionalities comparison across several platforms:

	PayPal	Alipay	Skrill	Venmo	AmazonPay	Trustly	Stripe	WePay	2Checkou
Wallet	x		x	x	x	-	-	x	-
Withdraw Money	x		x			-	-	x	-
Send Money	x		x	x		-	-	x	-
Request Money	x			x		-	-		-
Transactions Summary	x	x	x			-	-	x	-
Create an Invoice	x					-	-		-
Link Bank Cards	x	x	x	x	x	-	-	x	-
Link a Bank Account	x		x	x		-	-	x	-
Automatic Payments	x					-	-		-
Automatic Exchanges	x					-	-		-
Buyer Protection	x					-	-		-

Exchange between Crypto and Normal Currency			x			-	-		-
Split bills between users				x		-	-		-
List of orders					x	-	-		-
Merchants Agreements					x	-	-		-

Customer portal functionalities comparison across several platforms:

	PayPal	Alipay	Skrill	Venmo	AmazonPay	Trustly	Stripe	WePay	2Checkout
Full access to Documentation	x	x	x	x	x	x	x	x	x
Sandbox Environment	x			x	x	x	x		x
Demo Website	x						x		
Code Samples	x		x	x	x	x	x	x	x
List of Apps & Credentials	x								
Webhooks	x						x		x
API Log	x						x		x
Forum	x				x				

Appendix D

Online Payments
User Stories Document

Types of users identified:

- User not authenticated (NA)
- Merchant (M)
- Developer (D)
- Buyer (B)
- Participant (M, D)

User Stories

Unauthenticated user and all roles

U.S 1. As a site visitor I want to access an Initial Page so that I get to know what the website is about and how it works.

U.S 2. As a site visitor I want to access the Developer's Portal as the Developer's Documentation and see practical examples.

U.S 3. As a site visitor I want to be able to register for a new account as Developer and get an email activation.

U.S 4. As an unauthenticated user I want to be able to authenticate myself as Developers with valid credentials so that I can access functionalities available to my role.

U.S 5. As an unauthenticated user I want to be able to recover my password of my Developer account and get a confirmation email.

U.S 6. As an unauthenticated user I want to be able to authenticate myself with an existing M-Pesa account and get an Authentication code so that I can access functionalities available to my role(s).

Merchant Role

U.S 1. As a Merchant, I want to be redirected to Dashboard after login so that I can have an overview of my business.

U.S 2. As a Merchant, I want to be able to see my list of transactions so that I can keep trace of my money.

U.S 3. As a Merchant, I want to be able to filter and sort transaction's list.

U.S 4. As a Merchant, I want to be able to send Money to other Merchants or Customers.

U.S 5. As a Merchant, I want to be able to send Money to a list of persons with different amounts.

U.S 6. As a Merchant, I want to be able to request a Payment, so that I can send a bill to a customer.

U.S 7. As a Merchant, I want to be able to add frequent items that I use in customer's bill to a list.

U.S 8. As a Merchant, I want to be able to edit Bill Items that I had created.

U.S 9. As a Merchant, I want to be able to delete Bill Items that I had created.

U.S 10. As a Merchant, I want to be able to add a Contact to my Address Book.

U.S 11. As a Merchant, I want to be able to edit a Contact from my Address Book.

U.S 12. As a Merchant, I want to be able to delete a Contact from my Address Book.

U.S 13. As a Merchant, I want to be able to check all my numbers and statistics about my orders and revenue.

U.S 14. As a Merchant, I want to be able to choose what information is displayed in my report area so that I can have customizable reports .

U.S 15. As a Merchant, I want to be able to export my data as Transactions, Orders and Reports to a .csv file.

U.S 16. As a Merchant, I want to be able to see a list of orders.

U.S 17. As a Merchant, I want to be able to associate a Tracking Number to each order as update his status.

U.S 18. As a Merchant, I want to access the order details and all related content from any order in list.

U.S 19. As a Merchant, I want a tool to be able to generate a Checkout Button code, so that I will be able to put it my webpage by my own.

U.S 20. As a Merchant, I want a tool to be able to calculate the shipping costs "automatically" so that if the Merchant export to a different country or the order is heavier.

U.S 21. As a Merchant, I want a tool to be able to calculate the taxes to other countries so that I will be able to export to other countries.

U.S 22. As a Merchant, I want to be able to add Webhooks to each API event result, so that I can choose where each user will land after the API response.

Developer Role

U.S 1. As a Developer, I want to try all APIs in a sandbox environment so that I can see how it works.

U.S 2. As a Developer, I want to be redirected to Dashboard after login so that I can have an overview about last calls to my implemented APIs.

U.S 3. As a Developer, I want to see a list of all Merchant names that grant me third party rights to develop using theirs API keys.

U.S 4. As a Developer, I want to be able to create an Application to a Merchant so that I will be able to configure his APIs.

U.S 5. As a Developer, I want to be able to edit a Merchant Application so that I can change any configuration.

U.S 6. As a Developer, I want to be able to define Webhooks to an Application endpoint.

U.S 7. As a Developer, I want to be able to edit any Webhook to an Application endpoint.

U.S 8. As a Developer, I want to assign my ID to API calls to see their statistics.

U.S 9. As a Developer, I want to see statistics (status code, date) of all API request by application so that I can analyze what happened in which one.

Buyer Role

U.S 1. As a Buyer, I want to pay an Order from my M-Pesa wallet in any Shopping Website with support for this.

U.S 2. As a Buyer, I want to see a list with all my orders.

U.S 3. As a Buyer, I want to see the order details and all related content from any order in the orders list.

U.S 4. As a Buyer, I want to be able to confirm that my order arrived at the destination address.

U.S 5. As a Buyer, I want to be able to request a refund.

Participant Role

U.S 1. As an authenticated user I want to be able to logout so that further access to the platform is made as an unauthenticated user.

U.S 2. As an authenticated user, I want to be able to see the date of my last login.

Appendix E

PQ.01.01.08 v3.1

wit software

OnlinePayments

UI Requirements

Version 1.0
7 September 2019

Table of Contents

List of Figures

1. **Summary**

OnlinePayments is a proof of concept to show to Vodafone Group and it is directly related with M-Pesa project. OnlinePayments platform target are the Emerging Markets where M-Pesa is used as a Mobile Banking solution. Some examples of Emerging Markets are Kenya, Tanzania, Afghanistan, South Africa, India and Albania.
OnlinePayments is like PayPal but instead use bank accounts or cards uses M-Pesa mobile banking solution. This platform is an online payments system that supports online money transfers. This system operates as a payment processor for online vendors and shoppers that use the M-Pesa banking solution for exchanges.
OnlinePayments has 3 portals one for Shopper Checkout, other for Merchants and other for Developers.
The first portal has the objective to allow online shoppers to shop with the one-click experience.
The second one has the objective of allowing Merchants to manage their business giving them information about theirs M-Pesa wallet balance as of the online selling are going for them.
While the third one aims to be prepared to support all developers needs and give them all the help that they need to implement this payment solution. They can count with all documentation needed with code examples by side.

2. Features

The following instructions should be taken into consideration for defining the website UI. There are 4 user profiles and as the screens depend on the user's role, the following nomenclature should be considered:

UA – Unauthenticated User
M – Merchant
D – Developer
B – Buyer

1. OnlinePayments Landing Page (UA)
2. OnlinePayments Developers Landing Page (UA)
3. Login (M, D)
4. Dashboard (M)
5. Transaction Summary (M)
6. Send Money (M)
7. Address Book (M)
    a. Create/Edit Contact (M)
8. Analytics/Reports (M)
9. List of Orders (M)
    a. See Orders Details
10. API Documentation and Code Samples (D)
11. Sign Up (D)
12. List of Apps & Credentials (D)
    a. Create an Application (D)
13. API Statistics (D)
14. Checkout Portal (B)

The next sections will detail the requirements for each listed feature along with the correspondent User Story and mockups to exemplify.

2.1. OnlinePayments Landing Page

User Story:
U.S 1 - As a user not authenticated, I want to be able to navigate through all website presentation pages and understand what it is about.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508911

The landing page is the first screen that an unauthenticated user sees. It should be simple and let the users understand what this platform is about. Should also make easier for users to reach other pages with more detailed information about each of the main functionalities. As it's possible to see below besides the landing page it also as 3 more webpages with detailed information about "Online Shopping" (Figure 2), "Manage Your Business" (Figure 3) and "Developers" (Figure 4).

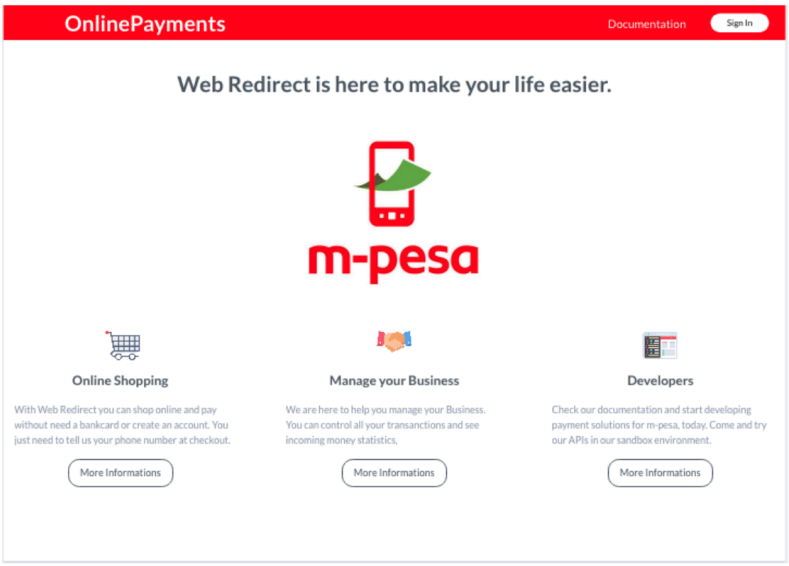

Figure 1 - OnlinePayments Landing Page

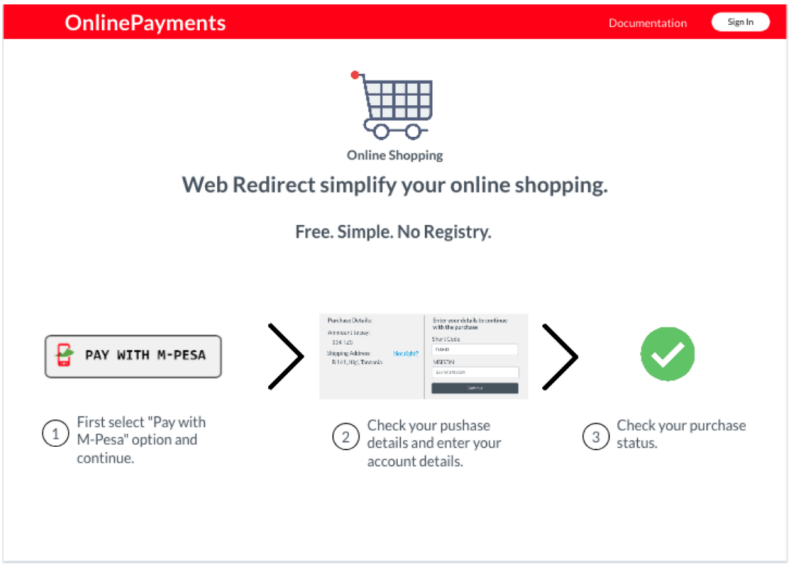

Figure 2 - Online Shopping Information Webpage

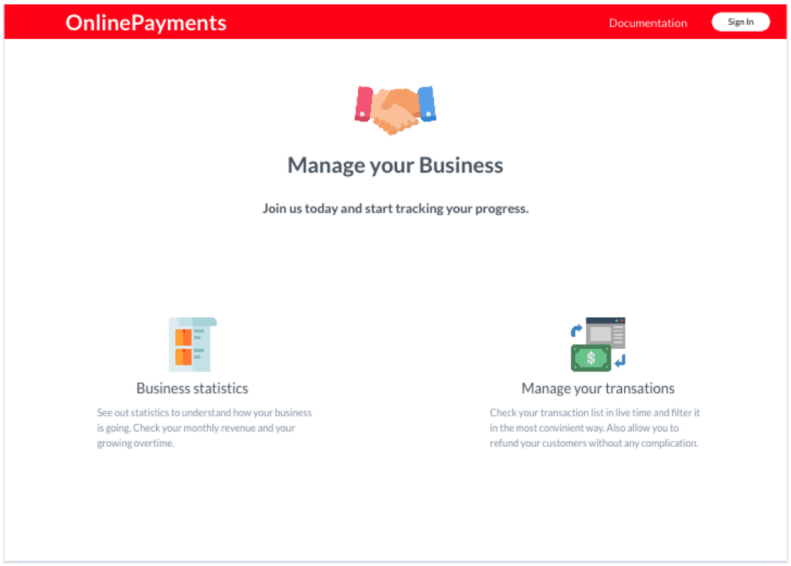

Figure 3 - Manage your Business Information Webpage

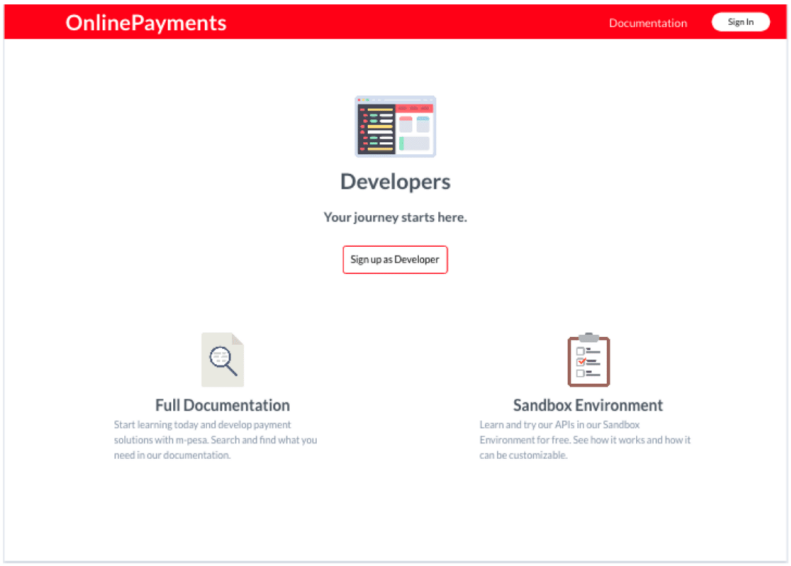

Figure 4 - Developers Information Webpage

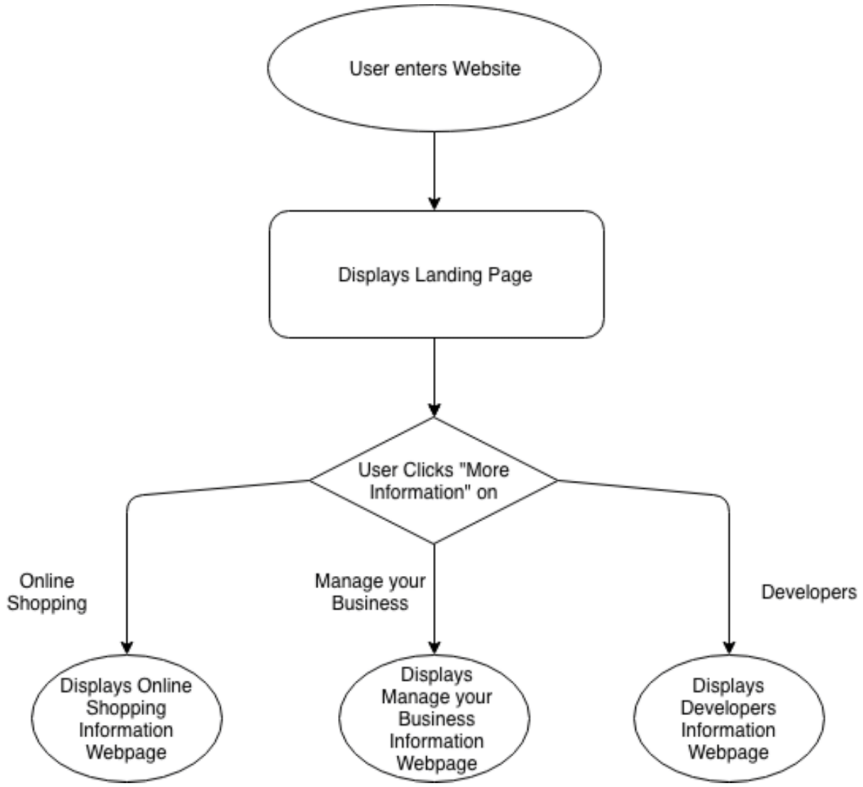

Figure 5 - Landing Page Flow

2.2. OnlinePayments Developers Landing Page

User Story:
U.S 2 - As a user not authenticated, I want to be able to navigate through all developer documentation and find practical examples.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508949

Developers also have their own portal and their own landing page. This page should be really simple, and its function is to let developers decide between login into their accounts or go directly to Documentation Webpage.

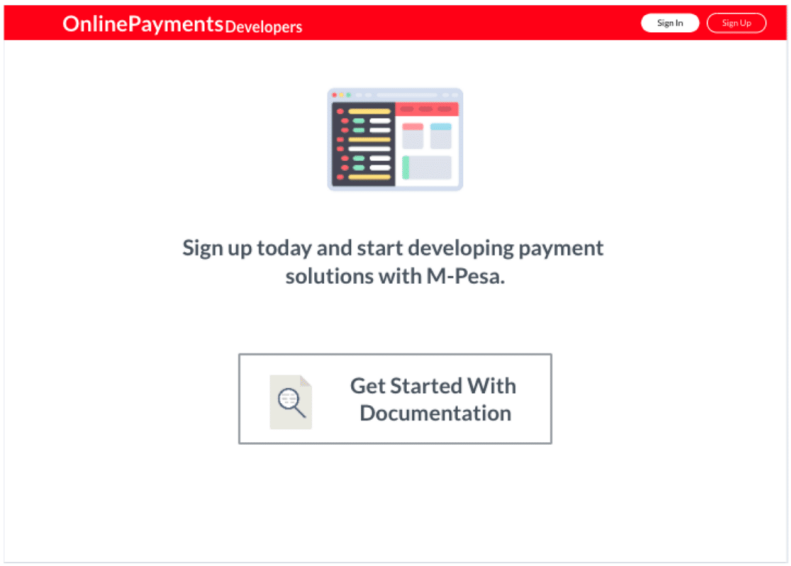

Figure 6 - OnlinePayments Developer Landing page

The Documentation is always composed by explanatory text and then a code sample and it must have a high contrast to make notorious the difference between them as can be seen in the Figure 7.

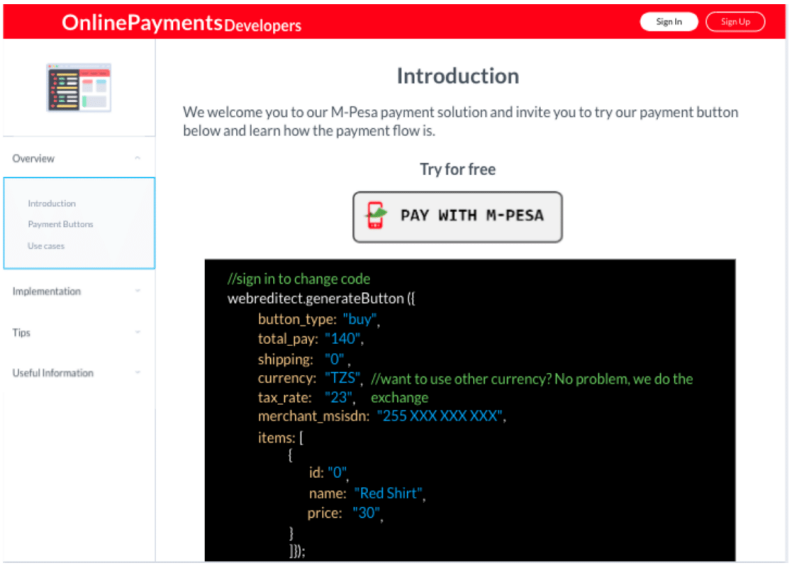

Figure 7 - OnlinePayments Developer Documentation

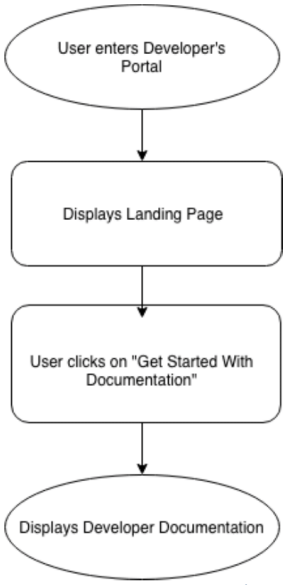

Figure 8 - See Documentation Flow

2.3. Login

In this feature there are 2 different user profiles involved and in each case the usage is different.

Merchant

User Story:
U.S 3 - As a Merchant, I want to be able to sign in using my Phone Number so that I will have full access to all functionalities and my data.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508950

Merchant already have an M-Pesa account, so they don't need to Sign Up to use OnlinePayments platform. This will work like a one account for multiple services. Merchants must login with an M-Pesa account that already exists.
For Sign in, Merchant will need to provide:

- Short Code;
- Phone Number;
- Agree with the Terms and Conditions (if is his first login).

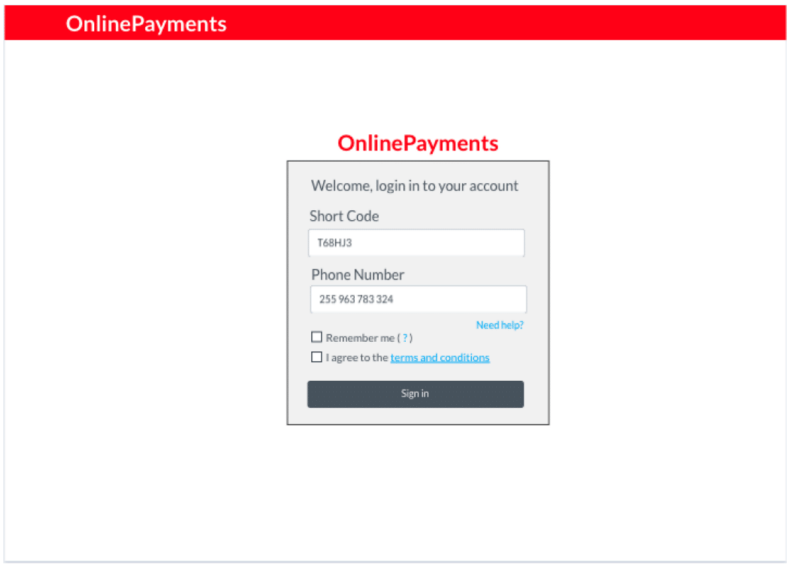

Figure 9 - Merchant Sign in Form

If the short code and the Phone number matches it will be required a 6-digit verification code that will be texted to Merchant at login.

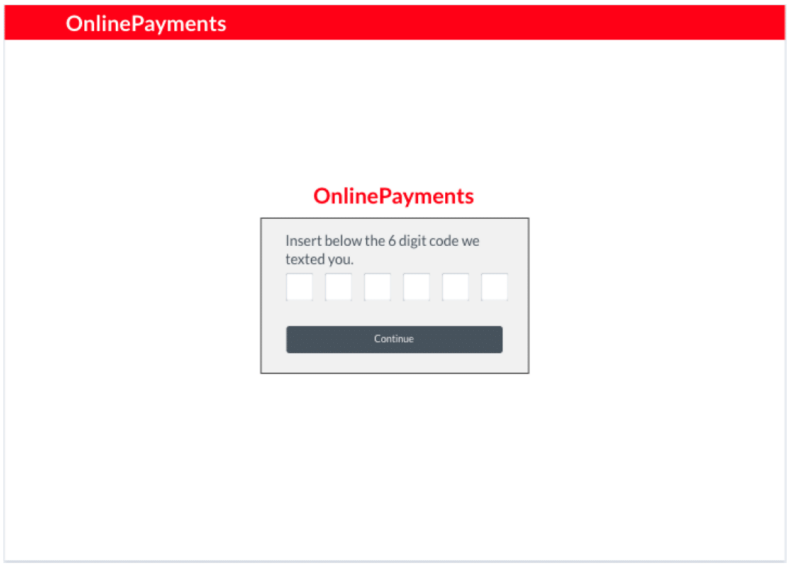

Figure 10 - Merchant Verification Form

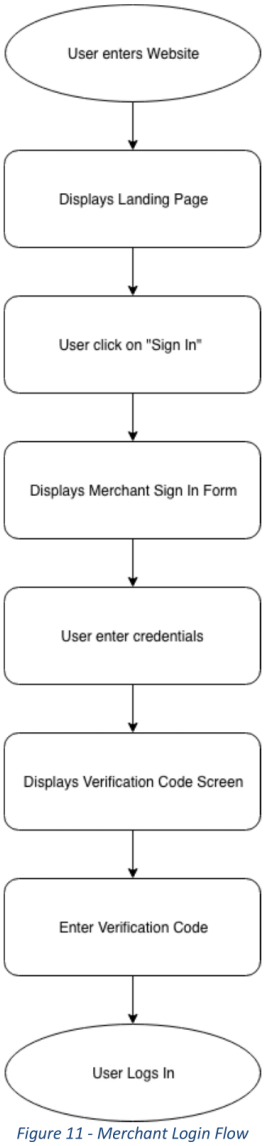

Figure 11 - Merchant Login Flow

Developer

User Story:
U.S 18 - As a Developer, I want to be able to sign in so that I get access to all my data about the developed Payment solutions.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508950

For Developers the story is different. They need to create an account providing a username, a password and a valid email. So, they can only log in with a OnlinePayments account.

For a Developer login successfully, he will need to provide:

- Username or Email;
- Password;

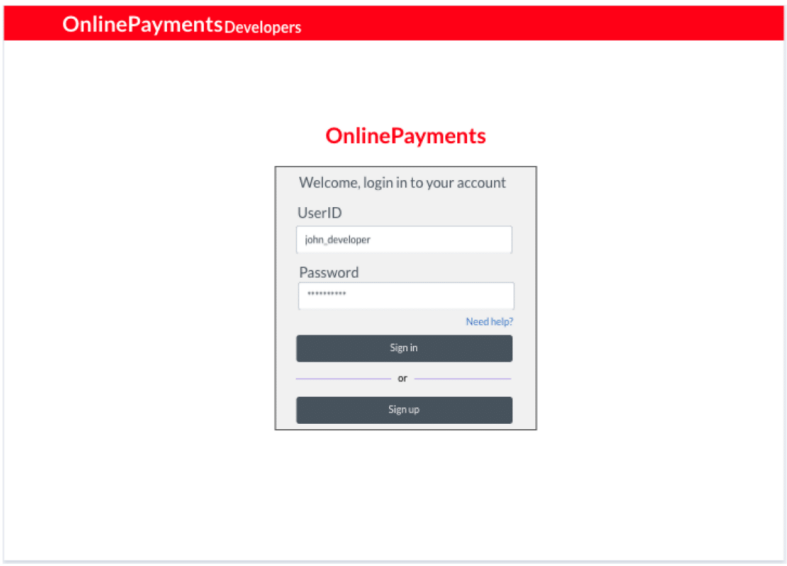

Figure 12 - Developers Sign In Form

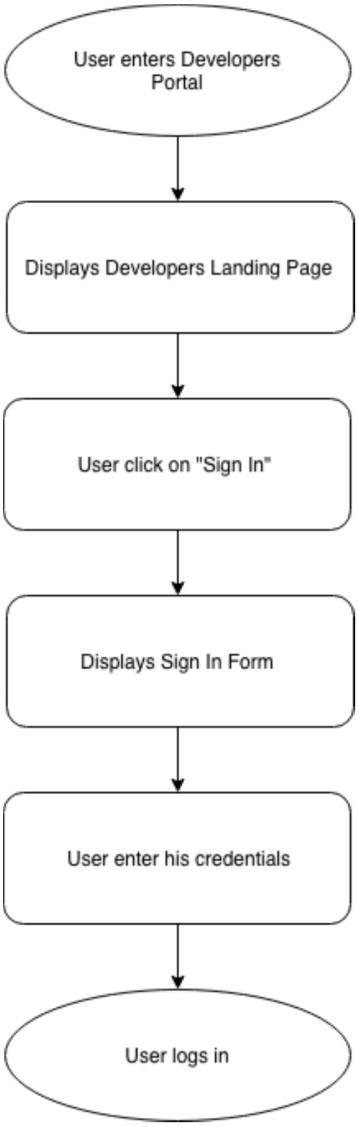

Figure 13 - Developers Login Flow

2.4. Dashboard

User Story:
U.S 4 - As a Merchant, I want to access my home page and it works like a dashboard so that I can easily understand what it is happening with my business.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49509572

After Merchant loggin successfully the dashboard is the first screen that he sees. It must be simple and make easy to undestand what is happening.
Dashboard must show the following informations:

- Merchant M-Pesa Wallet Informtion
- Monthly earnings;
- List with recent activities like:
  - Money Incoming
  - Money Outcoming;
  - Orders Received;
- Chart with orders summary and filter throughout:
  - day,
  - hour,
  - week,
  - month (default option);

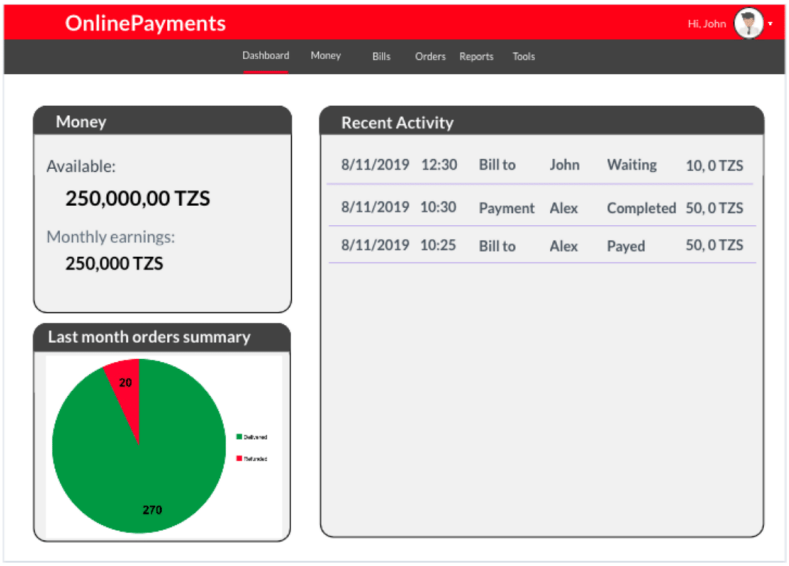

Figure 14 - Merchant Dashboard

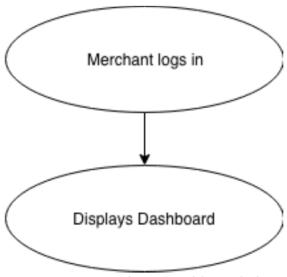

Figure 15 - Merchant Dashboard Flow

2.5. Transaction Summary

User Story:
U.S 5 - As a Merchant, I want to see a list of transactions so that I will be able to keep the trace of my money.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49509395

Transaction Summary is a list of all transactions that Merchant have made or received. Each row must specify the following data:

- Transaction ID;
- Transaction Date;
- Client;
- Amount;
- Type (Incoming, Outcoming, Refunded);
- Final Balance.

Transactions table must allow user to search transactions by:

- Transaction ID;
- Client.

Also, should allow user to order transactions by:

- Date (default option);
- Amount;
- Client;

And filter by:

- Time:
  - Today
  - Last 7 days
  - Last 15 days
  - Last Month
  - Anytime (default option)
- Transaction type:
  - Incoming;
  - Outcoming;
  - Refund;
  - All transaction types (default option).

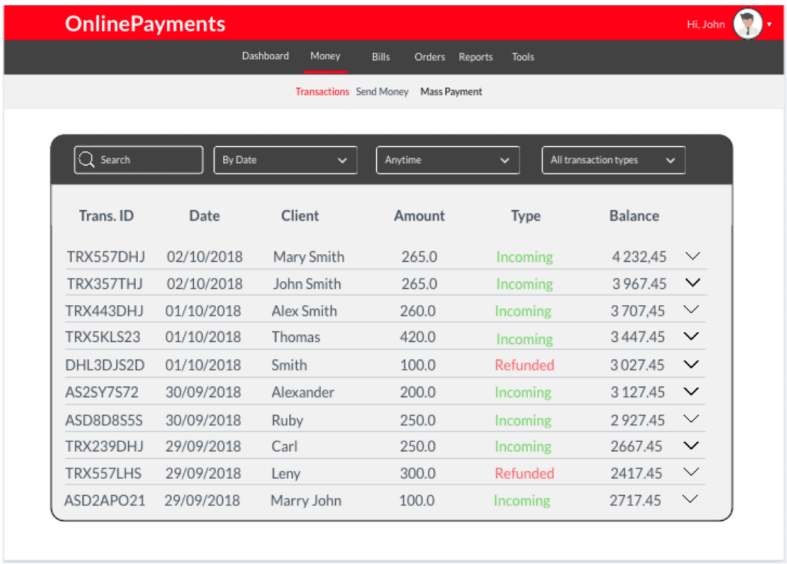

Figure 16 - Transaction List

Each row should also have the option to refund the client that made the order or even view the order details associated with that transaction.

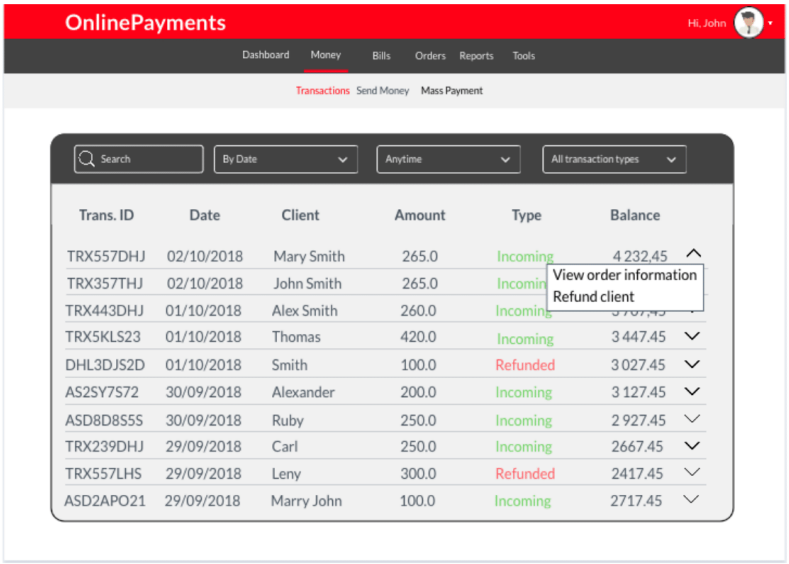

Figure 17 - Transaction List Popup Menu

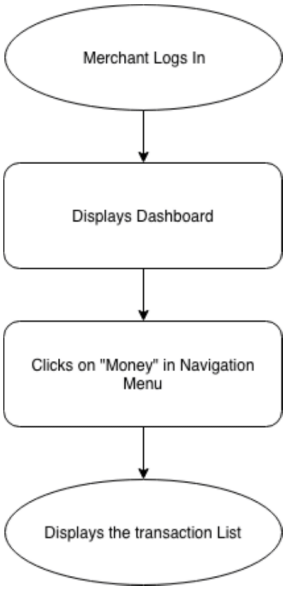

Figure 18 - Transaction List Flow

2.6. Send Money

User Story:
U.S 6 - As a Merchant, I want to be able to send Money.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508929

Merchant can send money to another person. To do this operation Merchant must indicate:

- Phone Number to send it or name (case it is registered in Address Book);
- Amount to send;

The Phone Number is an Edit Text with a autocomplete that will search the number or name that user is inserting in the Address Book.

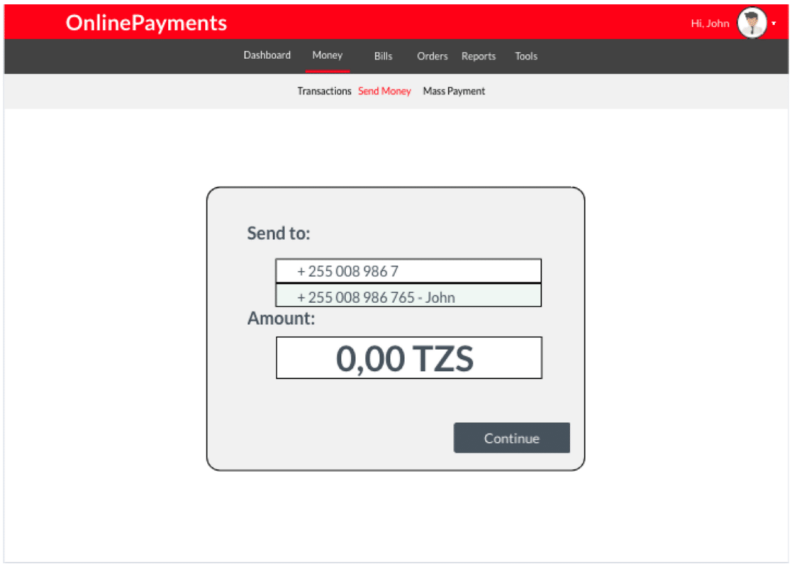

Figure 19 - Send Money

User will have a "Continue" button to confirm the payment and proceed with the operation. Then user must stay informed about the status of the operation. During the operation process it should show a loading animation to user.

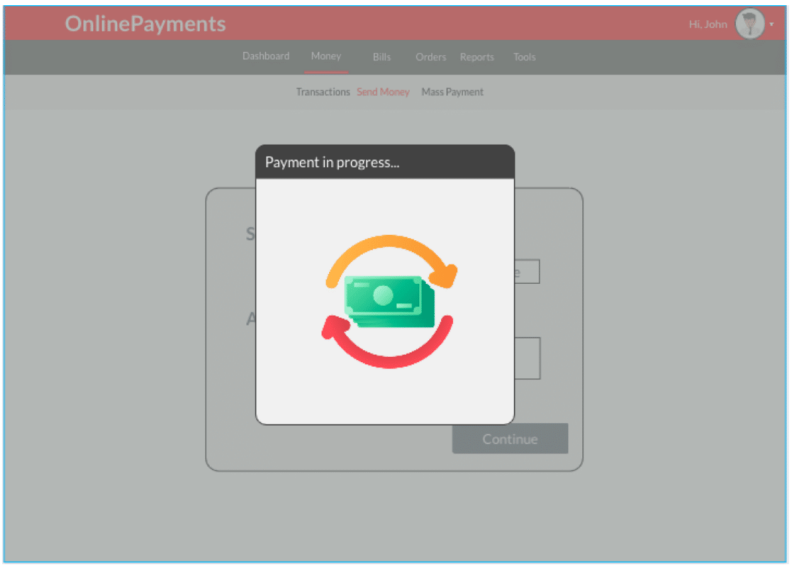

Figure 20 - Send Money in Progress

At the end progress animation should change to the operation result (OK or Failed) so user can know how it over.

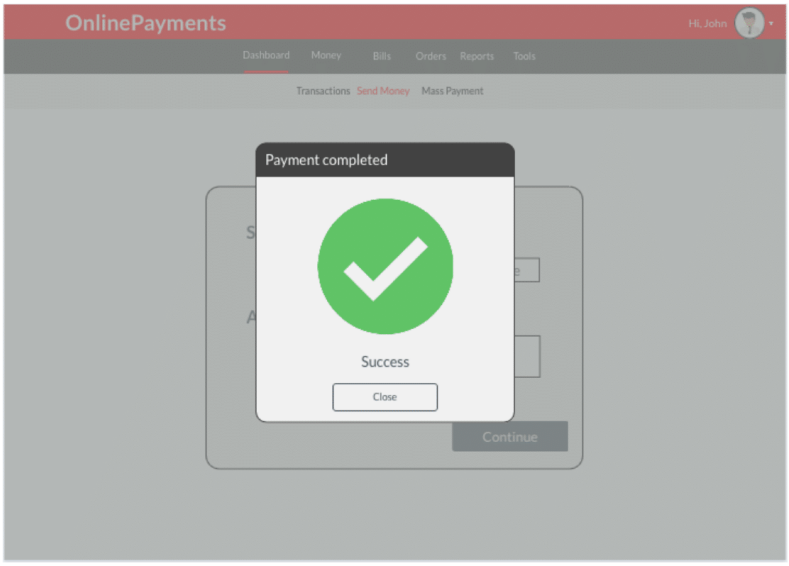

Figure 21 - Send Money Final Status

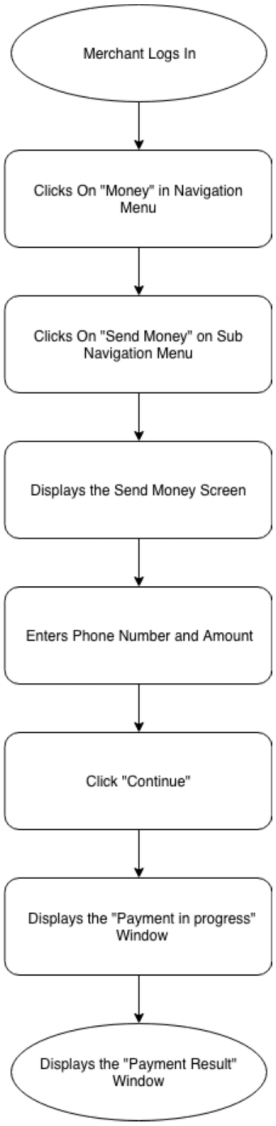

Figure 22 - Send Money Flow

2.7. Address Book

User Story:

U.S 9 - As a Merchant, I want to be able to add a Phone Number to my address list so that I can find them easily later.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508942

Address Book is the list where Merchant can associate Phone Numbers to a Name and make easier to him to find it later. Each row of the table must allow user to edit or remove the contact.

Each item must be displayed in list with the following information:

- Name;
- Phone Number.

Address Book List must allow user to search Contacts by:

- Name;
- Phone Number.

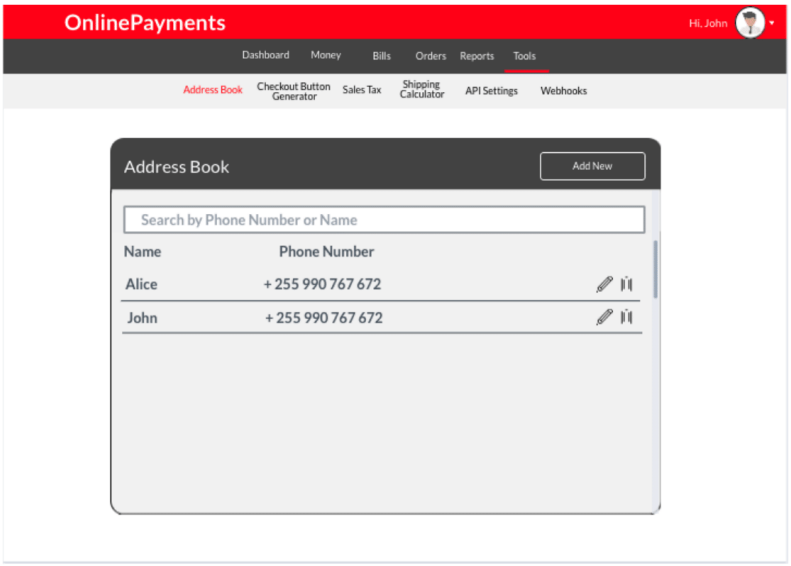

Figure 23 - Address Book List

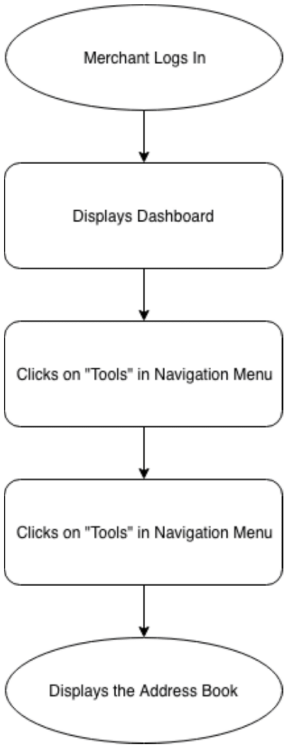

Figure 24 - Address Book Flow

2.7.1 Create/Edit Contact

The Address Book List must have a button near to Merchant click and add a New Contact.
For user create or edit a contact all that user needs to specify is:

- Name;
- Phone number.

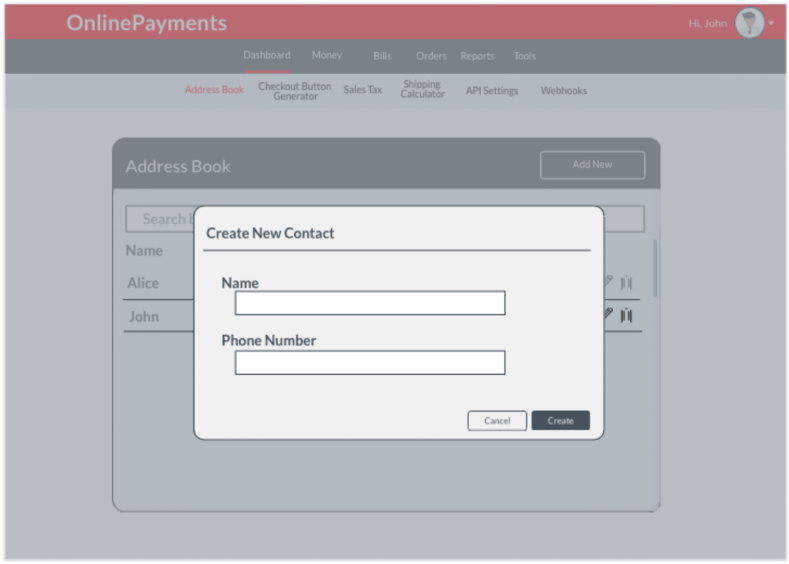

Figure 25 - Address Book Add/Edit

2.8. Analytics/Reports

User Story:
U.S 10 - As a Merchant, I want to be able to check all my numbers and statistics about my orders and revenues.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508944

This is a specific area for Merchants where the can access to their statistics and try to understand how their business are going. This area is dynamic which means user can change the displayed charts and cards and pick another one.

It must display charts like:

- Revenue;
- Average Money Per Order;
- Total Orders this Month;
- Sales Summary (total sales vs refunded and successfully delivered).

Should exist two buttons somewhere around the report's dashboard. One to choose data to export to an .csv and the other to open a settings window to configure which cards are visible in this dashboard.

The window to export data to a .csv should be really simple and contain a dropdown for options of data to export and a button to click to export.

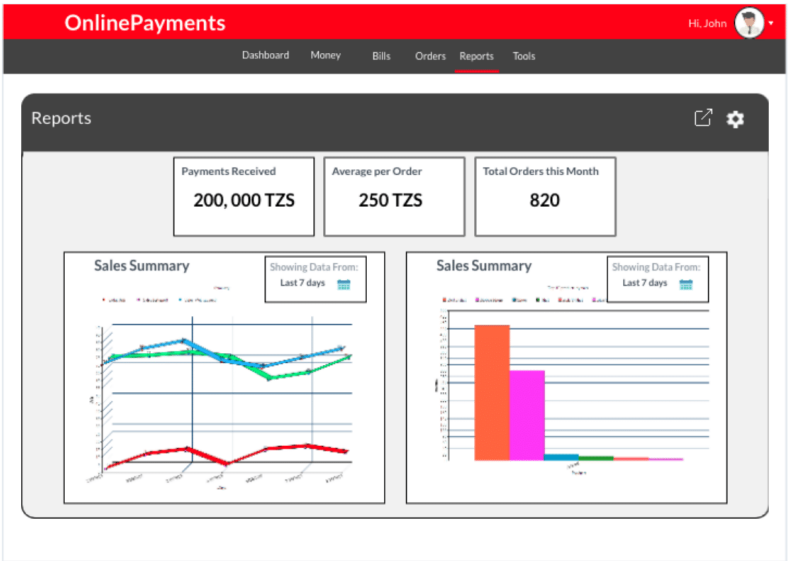

Figure 26 -Reports

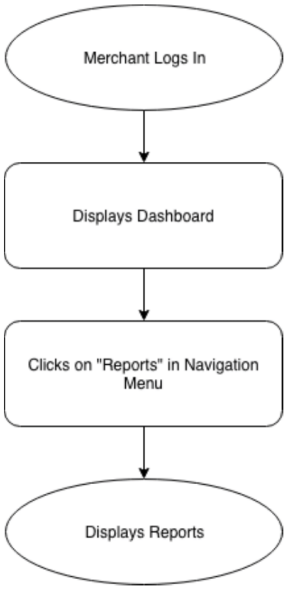

Figure 27 - Address Book

2.9. List of Orders

User Story:

U.S 12 - As a Merchant, I want to be able to see a list of orders so that I can keep track of the status of each order.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508947

This is a list of all orders that Merchant have received. Each row must display the following information:

- Order ID;
- Order Name;
- Order Price;
- Order Status;
- Order Tracking Number.

Besides this, each row must have a button that redirect user to details page of that order.

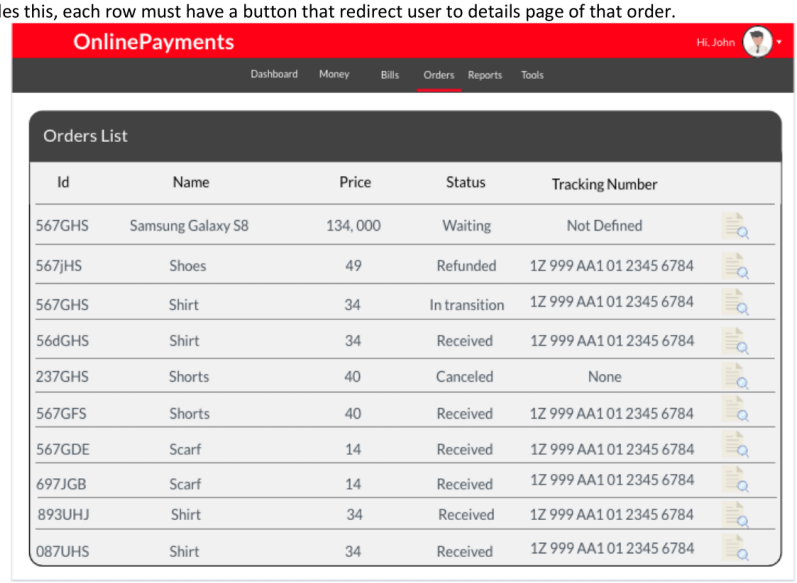

Figure 28 - Orders List

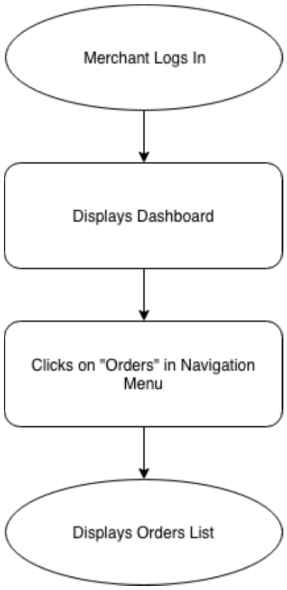

Figure 29 - List of Orders flow

2.9.1 See Order Details

The details page of the orders should give to Merchant information like:

- Order Information:
  - Order ID (read only);
  - List of Items in that order and amount (read only);
  - Amount to Pay (read only);
  - Shipping Cost (read only);
  - Order Current State (Waiting to send, Sent, Cancelled or Refunded);
  - Tracking Number.
- Buyer Info:
  - Name (read only);
  - Address (read only);
  - Mobile Phone Number (read only);

User must have the possibility to edit all fields that are not indicated as read only.

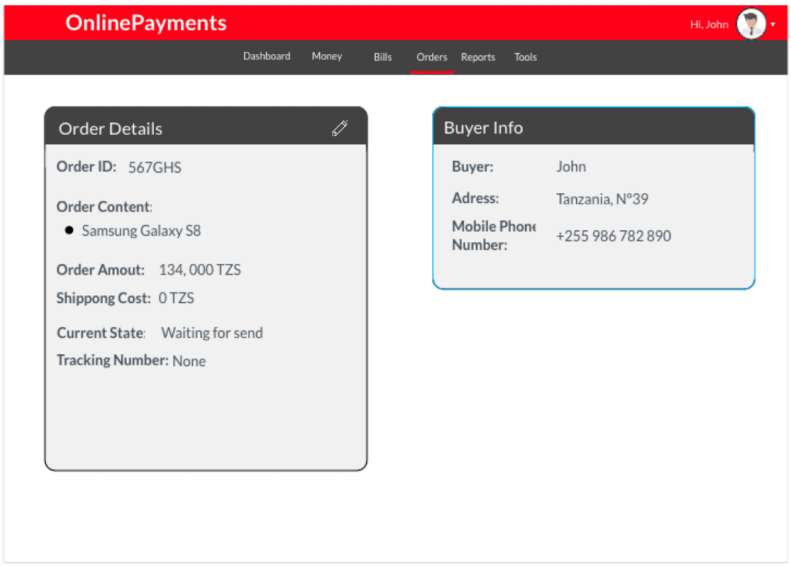

Figure 30 - Orders Detail

2.10. API Documentation and Code Samples

User Story:
U.S 17 - As a Developer, I want to be able to find all APIs documentation so that I will have practical examples of how to implement it.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508954

Developers should find easily all documentation with samples by side. The code samples should have a high contrast compared with the explanatory text. The organization should be similar to the Figure 31 with a Vertical Navigation Menu.

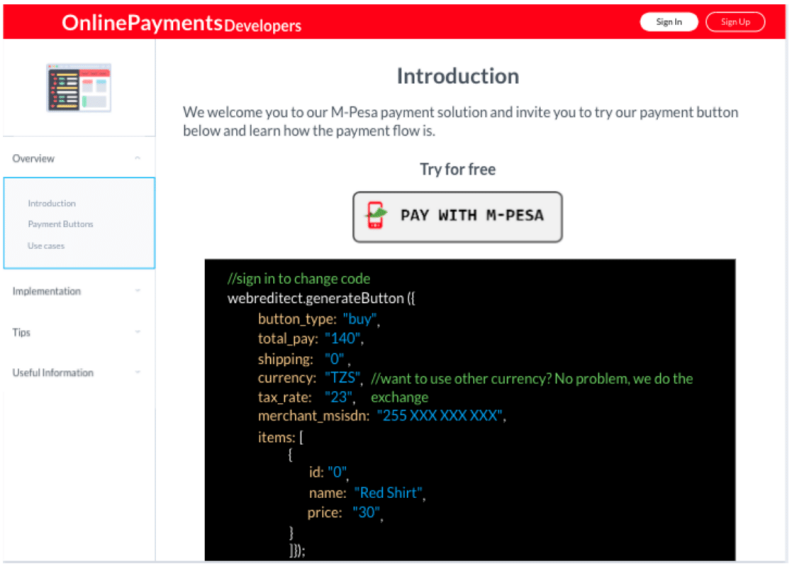

Figure 31 - Code Sample

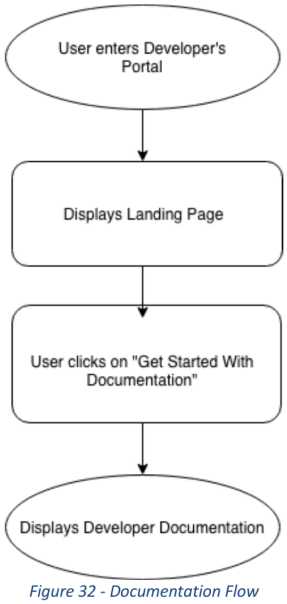

Figure 32 - Documentation Flow

2.11. Sign Up

User Story:
U.S 19 - As a Developer, I want to be able to sign up so that I can store my developed solution on my account.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508951

Developers need to create an account to be able to login into OnlinePayments Developers portal.

For a Developer Sign Up he needs to provide the following mandatory information:

- Email;
- Username;
- Password;
- Accept the Terms and Conditions.

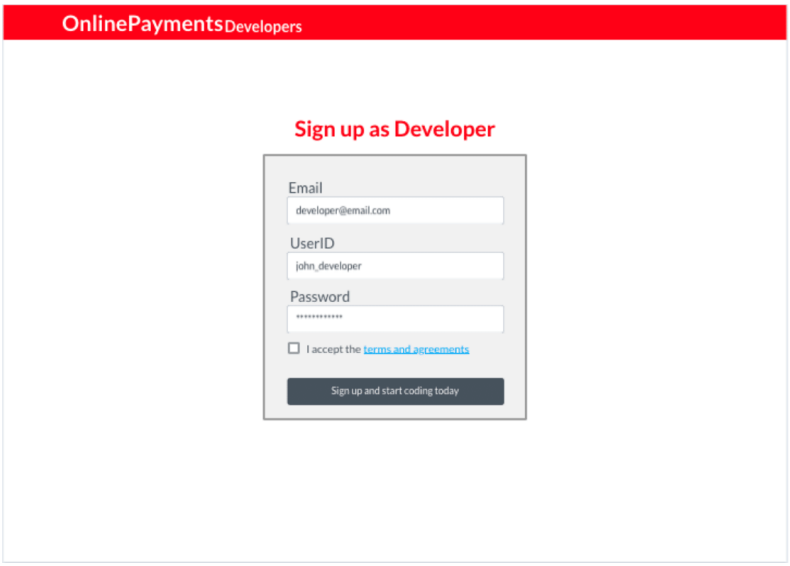

Figure 33 - Developer Sign Up

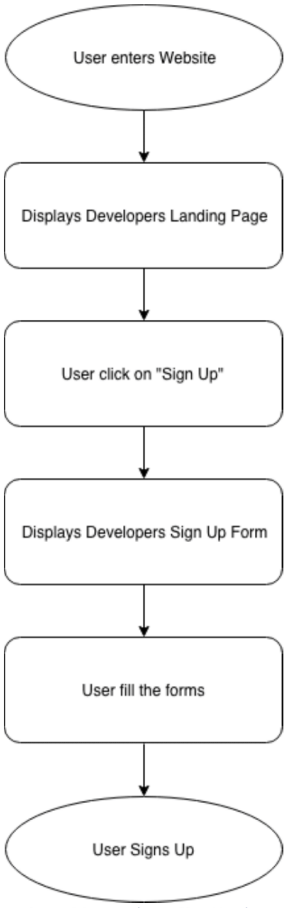

Figure 34 - Developer Sign Up Flow

2.12. List of Apps & Credentials

User Story:
U.S 22 - As a Developer, I want to see on my "List of Apps & Credentials" all Merchant names that grant me third party rights to develop using theirs tokens.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49661435

As a Developer is important to keep track of all Merchants that grant us third party permissions and allow us to develop solutions for them. And it is also important to keep all information from our developed solutions together. For that reason, developers can create an Application that is directly connected to a Merchant and will work like a HUB to keep all configurations and information's about it inside.

It is important to divide the List of Apps from the Credentials in 2 different tables as it is possible to see in Figure 35. One of the tables must contain all applications created by Developer so he can easily select one a seed his configurations. The Apps table items should have the following information:

- Application ID;

- Application Name;
- Merchant Store;
- Status.

The Developer at click on an Application List Item should see the Application Detail Page.

The other table contains all Merchant Credentials that we have authorization to use. Each row must contain the following information:

- Merchant Name;
- Credential;
- Granted at (Date);

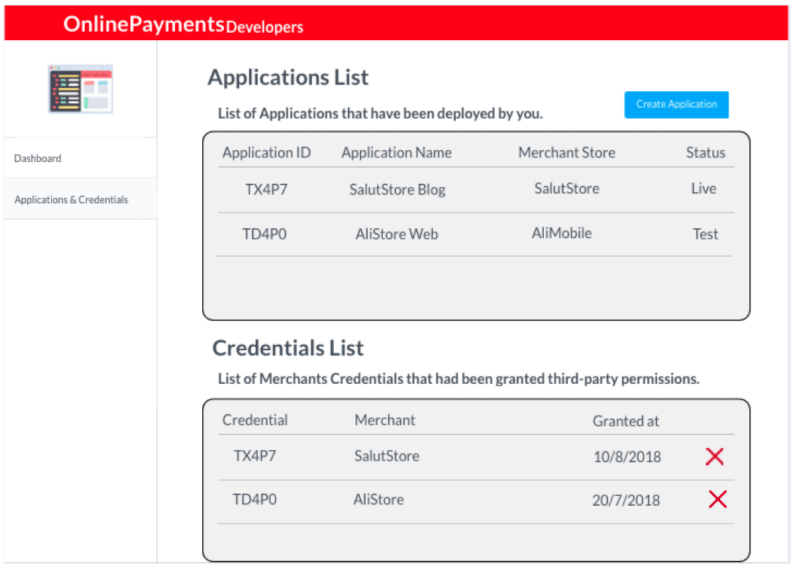

Figure 35 - Apps & Credentials List

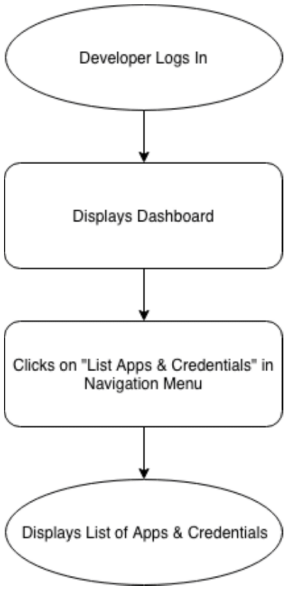

Figure 36 - List of Apps & Credentials Flow

2.12.1 Create an Application

User Story:
U.S 23 – As a Developer, I want to be able to create and Application to a Merchant so that I will be able to configure his APIs.

Developers can find a button near Applications Table in order to create a new one. After clicking should appear a window asking for the following information:

- Application Name;
- Merchant Credentials (dropdown with Merchant Credentials Available);

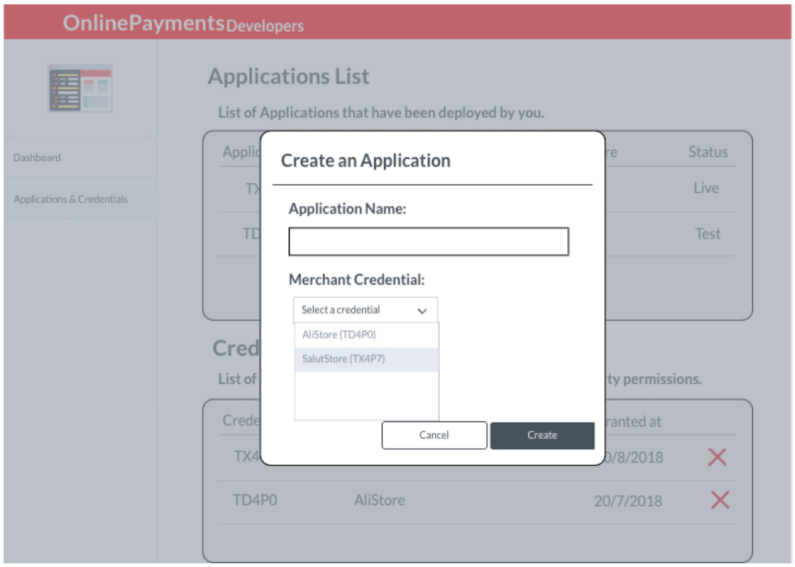

Figure 37 - Create an Application

2.13. API Statistics

User Story:
U.S 21 - As a Developer, I want to sign in an see a dashboard with statistics about last calls to my APIs.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508955

As a Developer is important to keep informed about the API's Call Result to know if some error is occurring. For solving it is stored API Logs, so Developer can see API responses and try to understand what is failing in the implementation.

Statistics should be represented with a line chart with:

- Successful requests (Green)
- Failed Request (Red)

And with a table where is possible to see all errors with more detailed information. Table items must have the following information:

- Application ID;
- Number of times error happened;
- Time passed since last error;
- Error.

Figure 38 - API Statistics

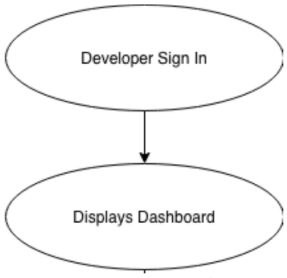

Figure 39 - API Logs Flow

2.14. Checkout Portal

User Story:
U.S 26 – As a Buyer, I want to Pay an order in the simples and secure way possible.

Full Working Example: https://marvelapp.com/a3i9g3e/screen/49508956

The Checkout portal must be as simple as possible. After user choose to pay with M-Pesa he is redirected from the shopping website to our portal where he can find the details of his purchase and a form by side for him to sign in by giving his phone number.

After that user will be texted with a 6-digit authentication code that will be requires in the next step, the verification. User will be asked for that code for verify himself and finalize the purchase.

The purchase details that user must found in the first page are the following ones:

- Amount to pay;
- List of Ordered Items(s);

- Shipping Address.

User also must find a help button, “Not right?” near shipping address so in case the address is not right it will show a tooltip with all information needed to user know what to do.

There is needed also an Input Text for user Phone Number and a Continue Button for Buyer Sign In. In this Sign In the only information required it the Buyer Phone Number as a continue Button.

After introduces the Mobile Phone Number user will receive a message with a verification code to authenticate himself.
The Verification screen must have a Input Text for the 6 digit code and a Continue Button. In the end should be played a loading animation till there’s a final status for the purchase that will be displayer replacing the load animation.

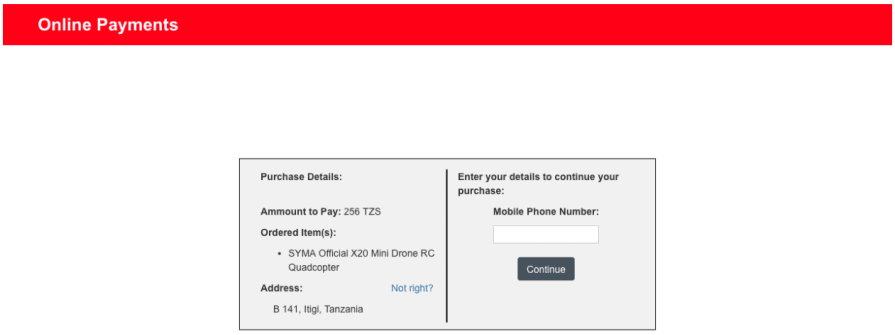

Figure 40 - Checkout Sign in Portal

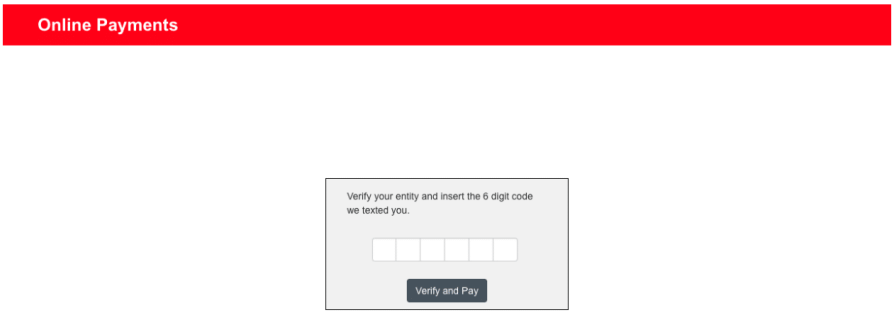

Figure 41 – Checkout Portal User Verification

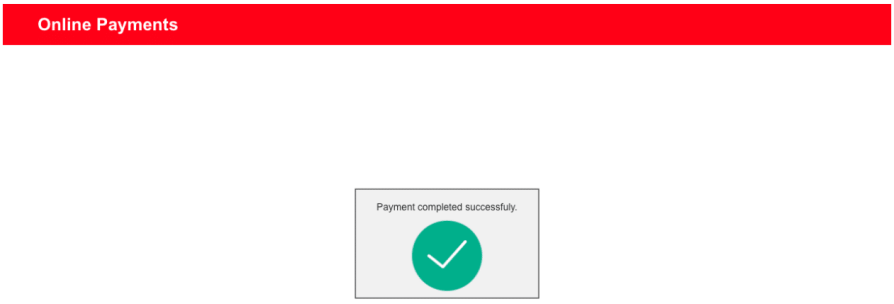

Figure 42 - Checkout Portal Status

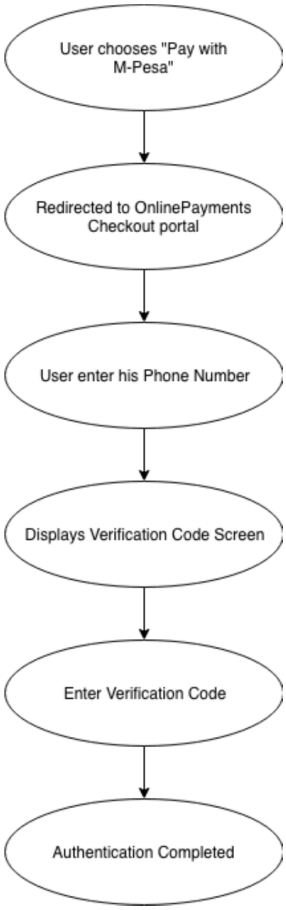

Figure 43 - Checkout Payment Flow