People's Democratic Republic of Algeria
Ministery of Higher Education and Scientific Research
Mohamed El Bachir El Ibrahimi University of Bordj Bou Arréridj
Faculty of Mathematics and Informatics
Informatics Department

DISSERTATION

Presented in fulfillment of the requirements of obtaining the degree

Master in Informatics

Specialty: Information and communication technologies

THEME

Loyalty Management system (mobile & web) based on Microservices architecture

Presented by :

MOHAMED NABIL AGUIDA
ACHREF BECHANE

Publicly defended on: 22/06/2023

In front of the jury composed of:

President : OUSSAMA SENOUCI
Examiner: RAMLA BELALTA
Supervisor: SAFA ATTIA

2022/2023

DEDICATION

"Oh my parents, your favor surrounds every bit of me.

Every grief we met caused you more grief.

Everything we have achieved is a result of your efforts.

Oh father, you were my supporter in hardships.

Oh you, who has Paradise under her feet.

All my words and all my thanks are chained.

Either I gather all the meanings in all the tongues.

I still cannot thank them and I never will.

As always daddy << with the left hand >>.

Dr.Miloud Naija, OH teacher Despite the few hours I spent with you, you left in me things that always remind me of you. You were the best, and you will remain the best. May God have mercy on you, sir"

Mohammed Nabil Aguida

DEDICATION

First of all, I thank ALLAH the most powerful who helped us and gave us the patience and courage to work hard and achieve the desired results.

To my beloved parents, who have nurtured and cherished me since the day I was born, your unconditional love and sacrifices have shaped me into the person I am today. I am eternally grateful for the countless sacrifices you have made to ensure my happiness and success. Your unwavering encouragement has given me the strength to persevere, and I dedicate this achievement to your boundless love and support.

To my amazing friends, who have generously shared their time, expertise, and insights, your invaluable contributions have played a pivotal role in the success of this project. Your unwavering support, creative ideas, and relentless dedication have not only elevated the quality of our work but also made the entire journey enjoyable and memorable.

Achref Bechane

ACKNOWLEDGEMENTS

First, we thank ALLAH the almighty for giving us the will to begin and complete this work in good conditions.

We also present our deep and sincere thanks to Mrs. Safa Attia our overseer on this thesis, for the considerable time she has granted us, for her motivation, precious orientations and her relevant criticisms that helped us a lot to complete this thesis.

We also thank Mrs. H ALIMA AGUIDA, MOHAMED AMINE SEDIIKI for all the time we spent together throughout this work, and for all her provided support to us.

Finally, I thank everyone who contributed from far or near to the realization of the presented work.

Abstract

Computers & mobile devices have become an essential aspect of our everyday routines as they serve as a gateway to various online services. In this work, we design and implement a novel microservice architecture to manage loyalty services. Our work helps business owners to provide a smooth and easy way to manage their customers loyalty. We implement both web & mobile applications based on a microservices architecture. To achieve this, we opt for UML as the modeling language for application design. For the BACK-END, we use the spring boot framework (JAVA) deployed on docker, for the interfaces (FRONT-END), we employe the Flutter framework (mobile) and Html, CSS, Java script (web).

Keywords: web & mobile apps, online services, microservices architecture, loyalty management.

Résumé

Les ordinateurs et les appareils mobiles sont devenus un aspect essentiel de nos routines quotidiennes, car ils servent de passerelle vers divers services en ligne. Dans ce travail, nous concevons et mettons en œuvre une nouvelle architecture de microservices pour gérer les services de fidélisation. Notre travail aide les propriétaires d'entreprises à fournir un moyen simple et facile de gérer la fidélité de leurs clients. Nous mettons en œuvre des applications web et mobiles basées sur une architecture microservices. Pour ce faire, nous optons UML comme langage de modélisation de l'application. Pour le BACK-END, nous utilisons le framework spring boot (JAVA) déployé sur docker, pour les interfaces (FRONT-END), nous employons le framework Flutter (mobile) et Html, CSS, Java script (web).

Mots-clés : applications web et mobiles, gestion de fidélité, services en ligne, architecture microservices.

ملـــخص

أصبحت أجهزة الكمبيوتر والأجهزة المحمولة جانبًا أساسيًا من روتيننا اليومي لأنها تعمل كبوابة للعديد من الخدمات عبر الإنترنت. في هذا العمل ، سنقوم بتصميم وتنفيذ بنية خدمات مصغرة جديدة لإدارة خدمات الولاء. يساعد عملنا أصحاب الأعمال على توفير طريقة بسيطة وسهلة لإدارة ولاء العملاء. و سنقوم بانشاء تطبيقات الويب والجوال بناءً على بنية الخدمات المصغرة. للقيام بذلك ، اخترنا UML كلغة نمذجة لتصميم التطبيق. بالنسبة لـ BACK-END ، سنستخدم إطار عمل Spring boot (JAVA) المستضاف على Docker ، للواجهات (FRONT-END) ، سنستخدم إطار عمل Flutter (المحمول) و Html و CSS و Java script (الويب).

الكلمات المفتاحية : الخدمات عبر الإنترنت - بنية خدمات مصغرة - إدارة ولاء العملاء

CONTENT TABLE

ABBREVIATIONS LIST

UML	Unified Modeling Language
Pos	Point of sale
CRM	Customer Relationship Management
KPI	key performance indicator
NIST	National Institute of Standards Technology
SAAS	Software as a Service
PAAS	Platform as a Service
IAAS	Infrastructure as a Service
VMM	virtual machine monitor
VMs	virtual machines
IT	Information Technology
API	Application Programing Interface
BFF	Backend for frontend
UP	Unified Process
B-O	Business Owner
QR	quick response
Ms	Microservice
IAM	Identity and access management
DB	Data Base
SSPL	Server-Side Public License
ERP	Enterprise Resource Planning
CIB	Corporate and investment Banking

FIGURES LIST

TABLES LIST

CHAPTER I

INTRODUCTION AND PROBLEMATIC

A. Contexts

1. Business context

1.1 Overview

Businesses now frequently use loyalty programs to boost the loyalty of their customers. who are rewarded with incentives and rewards for their continued patronage of a business. There are many forms and shapes for loyalty programs, such as point systems, tiered membership programs, or exclusive discounts. The retail industry, in particular, has seen a significant growth in the use of these programs, by providing these benefits, retailers aim to foster long-term relationships between businesses and customers, leading to increased customer satisfaction and increased revenue for the business.

Loyalty programs particularly apply to high-volume businesses that thrive on return customers, furthermore, since selling to an existing customer is cheaper than acquiring new ones, the possibility of building a loyal following is essential to providing value. When properly implemented, repeat customers will help recruit new ones at a fraction of the cost of traditional marketing methods.

In addition to rewarding customers for their repeated patronage, loyalty programs provide the issuing business with a wealth of consumer information and data While businesses can evaluate anonymous purchases, the use of these programs offer additional details on the type of products that may be purchased together, and whether certain incentives are more effective than others.

These initiatives may foster genuine brand loyalty when they are included into the customer's regular routine. Customers frequently become engaged in the program, and more than anything else.

1.2 Loyalty program types

1.2.1 Points-based loyalty program

Points-based programs reward customers for their purchases by awarding them points. The customers can then redeem these points for rewards or discounts. This type of loyalty program is designed to encourage customers to make repeat purchases, making them popular in retail environments, like restaurants. When customers reach a certain number of points, they can cash those points in to get a product or receive a discount [1].

1.2.2 Tiered loyalty program

Tiered loyalty program offers different levels of rewards and benefits based on the frequency and number of purchases made by the customer. The program is divided into different tiers, with each tier offering progressively better rewards and benefits, some programs use the names of precious metals -- silver, gold and platinum -- or other naming conventions to motivate customers to spend more and reach higher tiers for increased rewards [1].

1.2.3 Subscription based loyalty program

Subscription-based loyalty programs require customers to pay an upfront to become a member and receive benefits such as discounts, special promotions, and exclusive access to products or services. The goal of this type of loyalty program is to provide customers with a personalized experience and incentivize them to continue their patronage of the business. Subscription-based loyalty programs are often used in industries such as hospitality, entertainment, and retail [1].

2. Technical context

2.1 Overview

Cloud computing is a model for delivering computing services including servers, storage, databases, processing units, networking, software, analytics, and intelligence over the Internet “the cloud” to offer faster innovation, flexible resources, and economies of scale.

The impact of cloud computing on software development has been significant and far-reaching. Cloud software development is forever altering the way humans and technology interact. Businesses and individuals benefit from enhanced accessibility while reducing complexity. With cloud computing organizations can develop and deploy applications within a web browser. And your clients have the potential to use these apps while bypassing an unwelcome download and installation process.

When the software architecture design is the process of defining the structural elements of a software system and their relationships, as well as the behavior of the system. Two of the most common software architecture designs stand in a challenge of which is the suitable architecture for software dev, monolithic and microservices architecture.

2.2 Monolithic architecture

Monolithic architecture is a traditional software architecture style where all components of an application are combined into a single, tightly coupled unit. In a monolithic architecture, the application is built as a single, large executable, with all components being tightly integrated and dependent on each other, The word “monolith” is often attributed to something large and glacial, which isn’t far from the truth of a monolith architecture for software design, some of its advantages mentioning:

- Easy to develop: Monolithic architecture is relatively simple to develop, as all components of the application are combined into a single unit. This makes it easier for developers to understand the code, as well as to write, test, and debug the application.

- Easy to deploy: Because all components of a monolithic application are combined into a single unit, it is easier to deploy and run the application. There is no need to manage separate components or worry about dependencies between different parts of the system.
- Improved performance: Monolithic applications can be optimized for performance, as all components of the application can be designed to work together in a harmonious way. This can result in improved speed and efficiency.
- Reduced complexity: Monolithic architecture can help reduce complexity, as there are fewer components to manage and fewer dependencies to worry about. This can make it easier to maintain the application and ensure that it continues to work correctly over time.

However, as monolith architecture has some advantages in a small scale, when the app grows in complexity and size a monolith architecture may face some difficulties:

- Scalability: Monolithic architecture can be difficult to scale, as all components of the application are combined into a single unit. This makes it harder to add new features or handle increased traffic, as the entire system must be deployed and tested in its entirety.
- Maintenance: Monolithic architecture can be more difficult to maintain, as changes to one part of the system can affect other parts. This can lead to increased maintenance costs and a higher risk of bugs and other issues.
- Deployment: Monolithic architecture can be difficult to deploy, as all components of the application are combined into a single unit. This can result in longer deployment times and increased downtime, as the entire system must be deployed and tested in its entirety.
- Flexibility: Monolithic architecture can be less flexible than other approaches, as all components of the application are combined into a single unit. This can make it harder to adapt to changing requirements or add new features and functionality.

These difficulties or challenges of scalability and resilience effect growing up with businesses, software architects design a modern architecture which is microservices architecture to be able to overcome those challenges [2].

2.3 Microservices architecture

Microservice architecture is a software design pattern in which a large monolith application is broken up into a collection of small, independent loosely coupled services that communicate with each other over well-defined APIs. Each microservice is responsible for a specific business capability and runs in its own process, often on its own server. Here are some of microservices architecture’s strengths:

- Scalability: Individual microservices can be scaled up or down as needed, which helps to improve the performance of the overall application.
- Flexibility: With microservices, different parts of the application can be built using different programming languages and technologies.
- Resilience: If one microservice fails, it won't bring down the entire application. This helps to increase the overall stability and availability of the system.
- Continuous Delivery: Microservices can be developed and deployed independently, which enables faster delivery of new features and improvements.
- Improved Deployment: Microservices can be deployed to different environments and resources, making it easier to test and deploy new features.

However, implementing a microservice architecture can be challenging, as it requires careful planning and coordination to ensure that services can communicate with each other effectively and the overall system remains secure and performant [2].

The next figure (Figure I.1) represents the overall architectures (Monolothic vs Microservices)

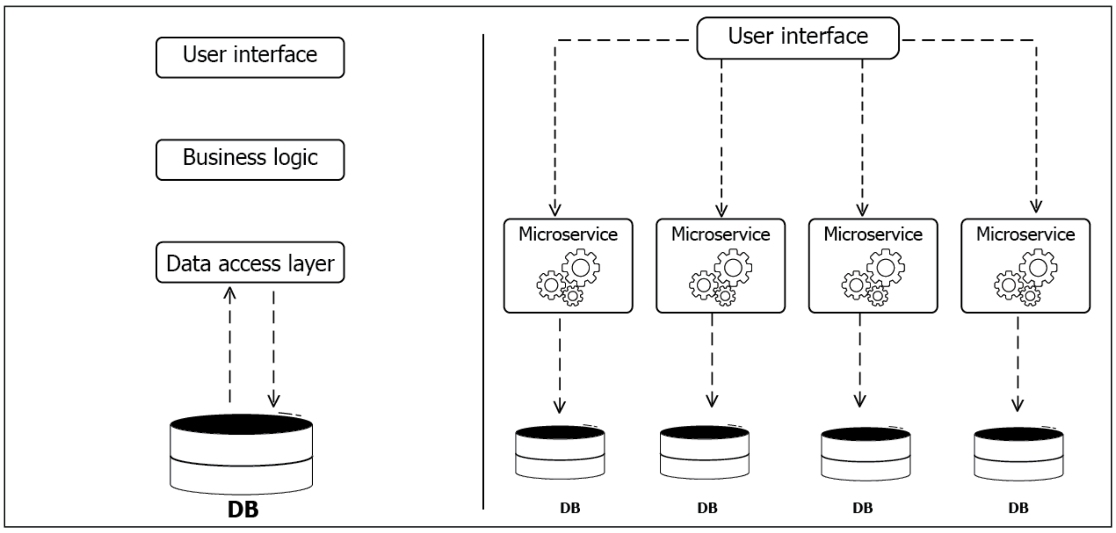


Figure I.1.Microservices vs Monolithic architectures [5]

B. Problematic

Managing a loyalty program can be a complex and challenging task, particularly for businesses with a large customer base. Some of the common issues faced in managing loyalty programs include:

- Complex Rewards Structures: With many loyalty programs offering a range of rewards, it can be challenging for businesses to manage the complex structures and ensure that customers receive the correct rewards in a timely manner.

- Redeeming Rewards: Some customers may find it difficult to redeem rewards, especially if they are not aware of the redemption process, if the process is overly complex, or lost, forgotten of the physical cards. This can lead to a lack of motivation to participate in the program.
- Fraud and Misuse: Loyalty programs can also be vulnerable to fraud and misuse; such as multiple accounts or cards being used by a single customer or rewards being redeemed for non-eligible items.

Addition to these issues, implementing traditional loyalty programs systems in Algeria face more challenges including:

- Lack of Integration: Loyalty programs that are not integrated with other systems, such as point-of-sale (POS) systems, customer relationship management (CRM) systems, and marketing automation systems, can result in a disjointed customer experience. This can lead to data inaccuracies, inconsistent rewards, and a lack of tracking and analysis capabilities.
- With the rare use of credit/debit cards in Algeria, the use of physical cards is uncommon to Algerians.
- poor data quality: absence stores management systems or offline solutions led to lack of data concerning retails, product reviews, successful loyalty programs patterns

Digitalization can help address these challenges and make loyalty program management more efficient and effective. Digital tools and systems can automate many of the manual processes involved in managing a loyalty program, such as tracking customer purchases and rewards, validating reward redemptions, and communicating with customers.

Deciding whether to use a monolithic or microservice architecture for a given tool or system can be a challenging problem.as we saw both approaches have their own advantages and disadvantages, and the best choice will depend on the specific requirements and constraints of the project, and the trade-offs between scalability, complexity, time to market, maintenance and deployment.

How we can implement a resilient and flexible solution providing real-time analytics and reporting capabilities, enabling businesses to monitor program performance with the capability of scale up?

C.Objectives and Contributions

Since data has become a powerful tool in the business industry, allowing business owners to gain valuable insights into customer behavior and preferences, as well as to optimize operations and improve the overall customer experience, the main objective of our project is to build a cross-platform (Android, Ios, and Web) application named <<Cardilla>> to contribute with:

- Business owners: help the business to grow up by providing them with tools enabling them to:
  - Create and manage Loyalty programs, by making the loyalty program more convenient and accessible through a digital platform, businesses can improve the overall customer experience.
  - A client dashboard for a clean visualization.
  - Real time KPI's enable them tracking the orientation curve their businesses.
- Customers: offer customers a convenient, personalized, and engaging loyalty experience
  - Storing loyalty card electronically will make customers no longer need to carry physical loyalty cards or remember multiple loyalty numbers.
  - offer immediate rewards for in-store purchases, including points or discounts. This leads to greater customer satisfaction.
  - Real time tracking of the various transactions and rewards, providing a clear understanding of their loyalty status.
- Our business perspective: our goals that we aspire to achieve from building this platform are:
  - Building a large database gathering small and medium businesses in Algeria.
  - Study and identifies trends, patterns, and relationships between customers and businesses in Algeria.
  - Assisting the advancement of the digital economy and the spread of technological solutions in Algeria.

With data analysis we can identify new opportunities, respond quickly to market changes, making our data among the most priceless resources for production businesses and marketers.

D. Work Plan

To cover both business and technical aspects of our project we decided to divide our thesis to four chapters:

The first chapter <<Introduction & Problematic>> of which we made an introduction to the loyalty programs, their types and their benefits on the growth of businesses, in the other hand we have defined a technical context putting two of the most common software dev architectures "Monolith" and "Microservices" architectures In a briefed comparison also presented the different problems and solutions which our platform is based on.

The second chapter <<State of Art>>in the part one, we have dived deeper defining the cloud computing, its characteristics, its overall architecture and why we need the cloud. Moving to microservices architecture, its components, the communication inter-services and finally the deployment. While the part two we have notating some of the existing models with a similar context.

The third chapter <<Conception & Methodology>> is devoted to the project framework, methodology, analysis and design, where we described our project, the UML formalism and the UP approach. We have also listed and established different UML diagrams related to the design of our application, the use cases, sequences, activities and class diagrams.

While the app development will be the subject of the fourth and last chapter<<Implementation>>in which we have defined the used development tools and illustrated some interfaces of the implemented platform <<Cardilla>>.

Finally, we concluded this work by summarizing the knowledge acquiredduring the realization of this project as well as some perspective.

E. Conclusion

In the first chapter, we discussed the business context, including an overview of loyalty program types, and the technical context. We identified the need for a robust loyalty program solution and outlined the objectives and contributions of this work. In the next chapter we will deep dive in explaining cloud computing and microservices architecture

CHAPTER II

STATE OF ART

A.Part 01

1. Introduction

Building flexible application that can adapt to changing business needs is the is the primary goal of the software architects, therefore combining two powerful technologies “Microservices architecture” and “Cloud Computing” enable architects to build highly adaptable, resilient, and cost-effective applications that can respond quickly to changing business requirements, while also improving performance, reliability, and maintainability.

In this chapter we will explore these two technologies in general highlighting their basic components.

2. Cloud Computing

2.1 Definition :

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. (The National Institute of Standards and Technology (NIST)) [3].

2.2 Characteristics

This model includes five essential characteristics, three service models, and four deployment models:

2.2.1 Essential Characteristics:

- On-demand self-service: users can request and access computing resources, such as storage or processing power, whenever they need it
- Broad network access: the ability of users to access computing resources from a variety of devices and locations
- Resource pooling: resources are combined into a shared infrastructure and dynamically allocated to multiple users, maximizing efficiency and scalability.
- Measured service: resource usage is monitored, measured, and reported back to users, allowing for accurate and transparent billing based on actual usage, as well as providing insights into resource optimization and planning [3].

2.2.2 Service Models

- Software as a Service (SaaS): model where software applications are hosted by a third-party provider and made available to users over the internet, eliminating the need for local installation and maintenance. Users typically pay for SaaS applications on a subscription basis.
- Platform as a Service (PaaS): model where a third-party provider offers a platform and tools for application development, testing, and deployment, eliminating the need for users to build and maintain their own infrastructure. Users typically pay for PaaS services on a pay-as-you-go basis.
- Infrastructure as a Service (IaaS): model where a third-party provider offers virtualized computing resources, such as servers, storage, and networking, that users can access and use as needed, without the need to purchase and maintain their own physical infrastructure. Users typically pay for IaaS services on a pay-as-you-go basis.

The next figure (Figure II.1) represents the main layers accessed in each service model.

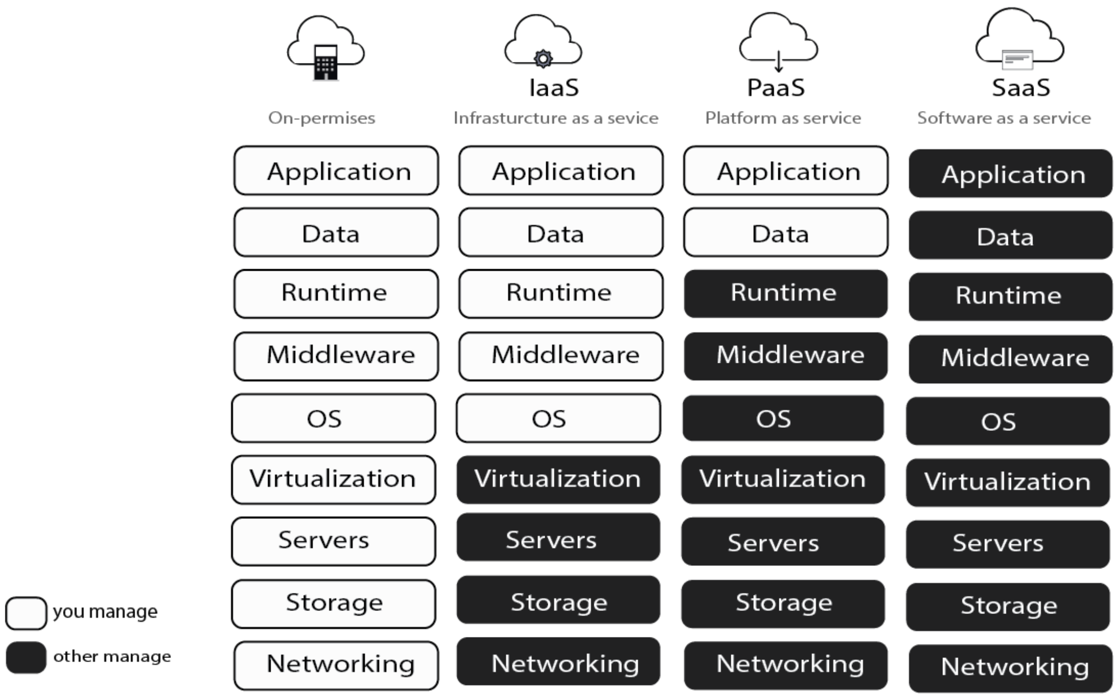
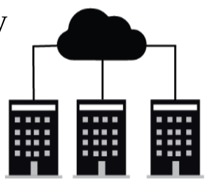


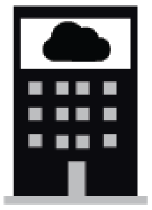
Figure II.1 Cloud computing service models [21]

2.2.3 Deployment Models

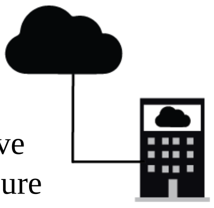
- Public cloud: computing resources and services are provided by third-party providers over the public internet. Public cloud services are typically offered on a pay-as-you-go basis, providing users with scalability, flexibility, and cost-effectiveness.



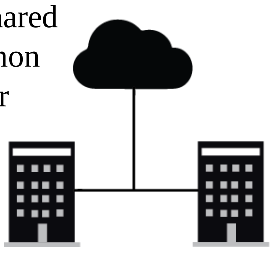
- Private cloud: computing resources and services are dedicated to a single organization or user, providing greater control, security, and customization. Private cloud services can be hosted either on-premises or by a third-party provider and can offer similar benefits to public cloud services, with the added advantage of being isolated and dedicated to a specific organization or user.



- Hybrid cloud: combines the use of both public and private cloud services, allowing users to take advantage of the benefits of both models. This approach enables organizations to optimize their computing resources by using public cloud services for non-sensitive workloads, while keeping critical data and applications in a secure private cloud environment.



- Community cloud: A model in which a cloud infrastructure is shared among a specific group of organizations or individuals with common interests, such as a particular industry, government agency, or geographic location. This model offers the benefits of public cloud services while addressing the concerns around security, privacy, and compliance of sensitive data by restricting access only to the authorized members of the community [3].



2.3 Cloud architecture

Cloud architecture is the structure of a cloud computing system, which is composed of various components that work together to provide computing resources and services to users. At a high level, cloud architecture can be broken down into four main layers:

1. Cloud Infrastructure Layer: This layer includes the physical resources that make up the cloud, such as servers, storage devices, and networking equipment. These resources are typically housed in data centers and are managed by cloud providers.
2. Cloud Platform Layer: This layer includes the various software platforms and tools that are used to manage and orchestrate the cloud infrastructure.
3. Cloud Services Layer: This layer includes the various services that are provided to users over the cloud, such as computing, storage, networking, and securityservices. These services can be customized and configured to meet the specific needs of individual users.
4. Cloud Applications Layer: This layer includes the applications and services that are built on top of the cloud infrastructure and platform layers. These applications can be developed and deployed by users, or they can be provided by third-party vendors.

Each cloud architecture layer consists of key cloud components that work together to provide a comprehensive and scalable cloud computing environment [4].

Layer 1	Layer 2	Layer 3
• Virtualization • Computing resources • Storage • Networking	• Management tools • Automation tools • Orchestration tools	• Data storage services • Content delivery services • Security services

Table II.1 Cloud architecture layers

After discussing the four layers of cloud computing architecture, it's important to consider the underlying technology that makes it all possible: virtualization

2.4 Virtualization

Virtualization is the process of creating a virtual version of something, such as an operating system, a server, a storage device, or a network resource. Virtualization allows multiple operating systems or applications to run on a single physical machine, or for multiple physical resources to appear as a single virtual resource [6].

2.4.1 Virtualization types

There are different types of virtualizations, each with its own unique benefits and use cases. Here are some of the most common types:

- Server virtualization: This involves creating multiple virtual servers on a single physical server, allowing for more efficient use of resources and better management of workloads.
- Desktop virtualization: This involves creating virtual desktops that can be accessed remotely by end users, allowing for greater flexibility and mobility.
- Network virtualization: This involves creating virtual networks that can be used to connect different physical networks or to partition a single physical network into multiple virtual networks.
- Storage virtualization: This involves creating a virtual layer between physical storage devices and the applications that use them, allowing for more efficient storage management and easier data migration [7].

2.4.2 Hypervisor

A hypervisor, also known as a virtual machine monitor (VMM), is a software layer that creates and manages virtual machines (VMs) on a physical server. The hypervisor sits between the physical server's hardware and the virtual machines, providing a layer of abstraction that allows multiple operating systems to run on the same physical machine without interfering with one another. There are two main types of hypervisors (Figure II.2):

- Type 1 hypervisors, also known as bare metal hypervisors, run directly on the physical hardware and are often used in server virtualization.
- Type 2 hypervisors run on top of a host operating system and are typically used in desktop virtualization.

Hypervisors play a critical role in virtualization and are a key component of many cloud computing architectures. By enabling the creation and management of virtual machines, hypervisors allow for more efficient use of computing resources and greater flexibility in managing workloads [8].

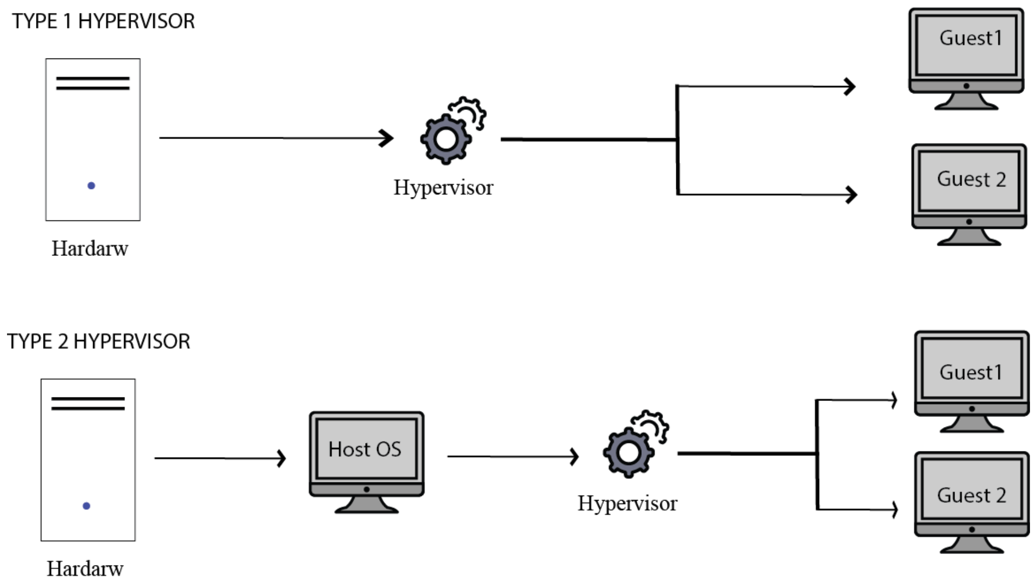


Figure II.2 Hypervisor types [9]

2.4.3 Virtualization impacts on the cloud

Virtualization has a significant impact on cloud computing by enabling the creation of virtualized environments that can be dynamically provisioned, scaled, and managed in response to changing workloads and business needs. By abstracting physical computing resources and creating virtual machines that can run on shared hardware, virtualization has enabled the creation of multi-tenant cloud computing environments that offer a range of benefits, including:

- Greater efficiency: With virtualization, multiple virtual machines can run on a single physical machine, enabling more efficient use of computing resources.
- Improved scalability: Virtualization enables cloud providers to rapidly provision new virtual machines in response to changing workloads, enabling greater scalability and flexibility.
- Lower costs: By reducing the need for dedicated hardware and enabling greater resource utilization, virtualization has helped to lower the costs of cloud computing.
- Improved reliability: Virtualization makes it easier to manage and maintain cloud environments, reducing the risk of downtime or service interruptions.

Overall, virtualization has been a key enabler of cloud computing and has played a critical role in making cloud computing more scalable, flexible, and cost-effective.

While virtualization offers many benefits, it also presents several challenges. These include increased complexity and Compatibility (Some software may not be compatible with virtualized environments). Proper planning and management are critical to addressing these challenges [10].

2.5 Why the cloud

The Cloud is an attractive option for organizations looking to improve their IT infrastructure and stay competitive in today's fast-paced business landscape due to its numerous benefits including:

- Scalability: Cloud computing allows organizations to quickly scale their computing resources up or down as needed, without having to make significant capital investments in new hardware.
- Cost Savings: Cloud computing can be more cost-effective than traditional on-premises computing because it eliminates the need for organizations to purchase, maintain, and upgrade hardware and software.
- Accessibility: Cloud computing enables users to access their applications and data from anywhere with an internet connection, making it easier to work remotely or collaborate with others.
- Flexibility: Cloud computing offers a wide range of services and deployment models, allowing organizations to choose the services and configurations that best meet their needs.
- Reliability: Cloud providers offer high levels of redundancy and fault tolerance, which ensures that applications and data remain available even in the event of a hardware failure or other issue.
- Security: Cloud providers typically invest heavily in security measures to protect their infrastructure and customer data, which can be more secure than on-premise solutions.
- Innovation: Cloud computing allows organizations to quickly adopt new technologies and services, enabling them to stay competitive in a rapidly evolving marketplace.

In summary, cloud computing offers a wide range of benefits that can help organizations improve their operational efficiency, reduce costs, and foster innovation [11].

3. Microservices

3.1 Definition

Microservices architecture is a way of designing and developing software applications as a collection of small, independent, and loosely coupled services, each running in its own process and communicating with other services over a network. [12]

3.2 Microservices architecture components

Here are some of the key components of a microservices architecture:

1. Microservices: A microservice is a small, independent, and loosely coupled service that performs a specific function within an application. Each microservice runs in its own process and communicates with other microservices through a network. Microservices can be developed, deployed, and scaled independently, which makes them more flexible and resilient than monolithic architectures.
2. API Gateway: An API Gateway is a service that acts as a single-entry point for all client requests to the microservices. It provides a layer of abstraction between the clients and the microservices, which helps to simplify the client code and improve the scalability and security of the microservices architecture.
3. Message Broker: A Message Broker is a service that facilitates communication between microservices. It enables messaging between microservices, which helps to decouple them and make them more resilient to failures. BFF (Backend for frontend): a microservice designed to handle requests and aggregating responses from the different microservices to send it back to the frontend.
4. BFF (Backend for frontend): a microservice designed to handle requests and aggregating responses from the different microservices to send it back to the frontend.
5. Service Registry/Discovery helps in locating services based on a database of all available microservices, their endpoints, and metadata.
6. Load Balancer: A component that distributes incoming requests to the appropriate service instances based on their availability
7. Configuration Server: A centralized repository for storing configuration information that is accessible to all microservices.
8. Service Monitoring: A system that tracks the health and performance of the microservices and generates alerts in case of failures or performance degradation.
9. Circuit Breaker: A mechanism that helps prevent cascading failures by interrupting communication between services when one service is not responding.
10. Service orchestration: A layer that coordinates communication and interactions between microservices to ensure that they work together correctly [13].

3.3 Communication Inter-microservices

Effective communication between microservices is critical for the success of a microservices-based application, Asynchronous and synchronous communication are two approaches to exchanging information between microservices.

Synchronous communication: involves a request-response model where the client makes an API call and waits for the server to respond. In this approach, the client has to wait until it receives a response from the server before it can proceed with its task. Synchronous communication can be simpler to implement but can also lead to scalability issues, as many requests can increase response times.

Asynchronous communication involves a message-passing model where the client sends a message to the server through message queues for example, and the server processes the message at its own pace. In this approach, the client does not wait for a response and can continue with its task. Asynchronous communication can improve scalability and fault tolerance, as messages can be queued and processed later, and failures can be handled more gracefully in the meantime async communication can cause more complexity.

Both async and sync communication have their benefits and drawbacks, and the choice between them depends on the specific use case and requirements of the microservices architecture. In general, synchronous communication is better suited for use cases that require immediate responses, while asynchronous communication is better suited for use cases that can tolerate delays and require high scalability and fault tolerance [14].

3.4 Microservices Deployment

Microservices deployment is typically done using containerization technologies such as Docker or Kubernetes. Each microservice is packaged as a container and deployed separately, making it easier to manage and scale individual components of the application [15].

The deployment process for microservices involves several stages we should highlight, but first we have to define containerization.

Containerization: a technique used in software development and deployment that allows applications to be packaged with all their dependencies and run consistently in any environment. Containers are lightweight, standalone executable packages that contain all the necessary components to run an application, such as the code, libraries, and configuration files. It provides several advantages in a microservices architecture [16]:

- Isolation: Each microservice can be packaged into its own container, providing isolation from other microservices and ensuring that they can run independently of each other.

- Scalability: Containers can be easily scaled up or down to meet the demands of each microservice, allowing for greater flexibility and efficiency.
- Portability: Containers can be easily moved between environments, such as from development to testing to production, without any changes to the underlying application code.
- Fast deployment: Containers can be deployed quickly and easily, allowing for faster release cycles and quicker time to market.
- Resource optimization: Containers use fewer resources than traditional virtual machines, allowing for greater resource optimization and cost savings.
- Consistency: Containers ensure that each microservice runs in the same environment, providing consistent behavior and reducing the risk of errors.

Overall, containerization provides a flexible and efficient way to deploy and manage microservices in a modern architecture, allowing for faster development cycles and stages, Noting some deployment stages:

1. Building and packaging each microservice into a container image.
2. Deploying the container images to a container registry or repository.
3. Using an orchestration tool to deploy and manage the microservices in a containerized environment.
4. Configuring load balancers and service discovery tools to ensure that requests are routed to the appropriate microservices.
5. Monitoring the performance of the microservices and making adjustments as needed to optimize the application's overall performance [15].

B. Part 02

1. Introduction

After conducting research, we have discovered some applications that offer different loyalty cards management services. In this section, we will present the applications we have found, outlining some of their strengths and weaknesses, similarities, and other relevant details. We will then use this information to optimize our own platform to the best of our abilities.

During our research, we did not come across any application in Algeria that offers the same set of services as our proposed application. This presents a unique opportunity for us to fill a gap in the market and become the go-to platform for businesses and customers in Algeria looking for reliable and convenient loyalty management services. By capitalizing on this opportunity, we can establish ourselves as a leader in the industry and pave the way for future expansion and growth.

2. Existing Templates

2.1 Stocard

Stocard (Figure II.3) is a digital wallet app that allows users to store all their loyalty cards in one place. It was founded by BjörnGoß and David Handlos in Germany in 2011.

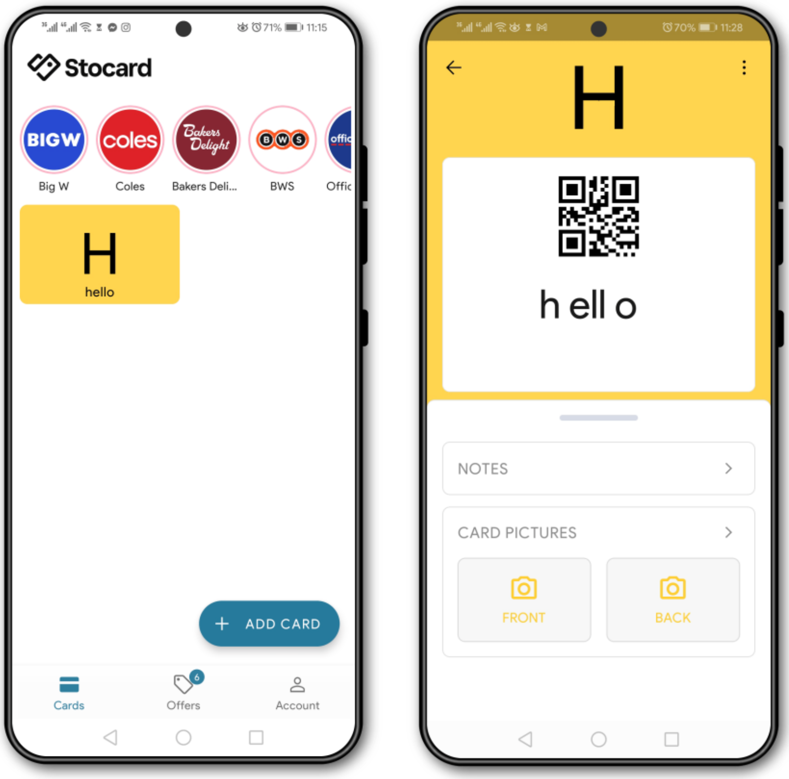


Figure II.3 stocard app *[30]*

Strengths:

- User-friendly interface
- Ability to scan cards using the phone camera
- Availability of special offers and deals

Weaknesses:

- Limited functionality beyond storing loyalty cards
- Lack of features such as expense tracking
- Issues with the app's card scanning feature, which can be unreliable or difficult to use.

2.2 Yollty

Yollty (Figure II.4) is a mobile app that allows users to collect loyalty points and rewards from their favorite businesses, such as cafes, restaurants, and shops. The app aims to simplify the loyalty program experience for users, making it easy to earn and redeem rewards across a variety of businesses. It was founded in 2014 by Michael Judeh and Mario Hassan, who are both based in Switzerland.

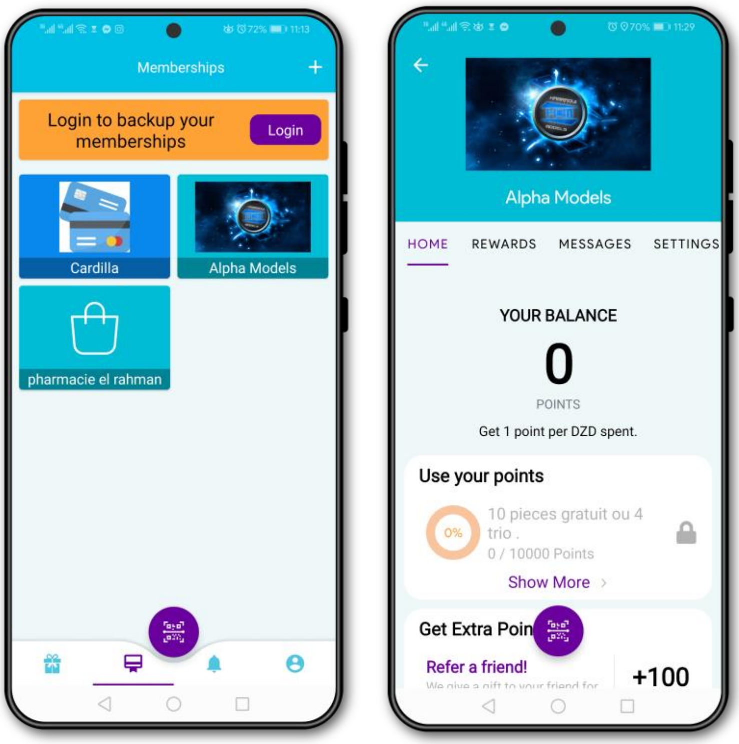


Figure II.4 yolltyapp *[31]*

Strengths:

- Easy to use interface: Yollty's user-friendly interface makes it simple for users to collect loyalty points and redeem rewards.
- Variety of businesses: Yollty works with a wide range of businesses, giving users plenty of options to earn and redeem rewards.

- Social media integration: Yollty allows users to share their rewards and experiences on social media platforms, which can help promote the businesses they visit.

Weaknesses :

- Privacy concerns: Some users may be hesitant to share their personal information and location data with the app, which could impact their willingness to use it.

2.3 Catima

Catima Loyalty Card Manager (Figure II.5) is a mobile app that allows users to manage their loyalty cards and rewards programs from a single platform. Users can add their loyalty cards to the app, it has almost the same features with yollty app.

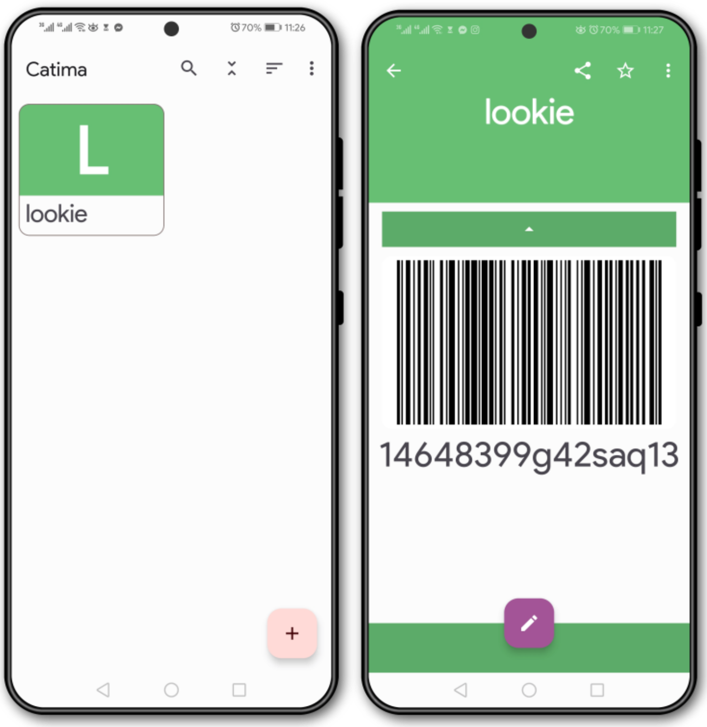


Figure II.5 catima app *[32]*

3 Model Description

Through our research on similar loyalty card management apps, our goal was to gather information that would help us create a platform that would not only address the strengths of existing apps, but also compensate for their weaknesses. This involved analyzing the features and functionalities of existing apps, as well as evaluating user feedback and reviews to identify areas where improvements could be made. The ultimate goal was to create a platform that would provide a new experience in Algeria helping with the process of digitalizing alongside, we are aiming to achieve the following goals:

Business:

- Track business performance KPI.
- Promoting the business with cost-effective marketing.
- Increase customer loyalty.
- Enhanced customer data and insights.

Customer:

- Track expenses and transactions.
- Store all loyalty cards in one single app
- Quick access to offers and coupons.
- Varity of businesses and goods

4 Conclusion

In this chapter, we provided a deeper definition of cloud computing and microservices, including their essential characteristics and core concepts, in the second part we reviewed different existing solutions along with their strengths and weaknesses, at the end we showed that our model provides more flexibility and performance

CHAPTER III

CONCEPTION AND METHODOLOGY

1 Introduction

In software development, the use of modeling is a critical practice as it enables the anticipation, prediction, and analysis of system information. A model is typically linked to a specific development approach, and for our purposes, we have selected the UML language in combination with a Unified Process (UP) approach for modeling our application, we aim to create a clear and structured understanding of our system, allowing for effective development.

2. Conception methodology

Conception methodology is an essential process in software development that involves defining, designing, and planning a software system. The methodology helps to ensure that the system performs its intended function effectively. A well-structured conception methodology can save time, reduce costs, and improve the overall quality of the final product. In this context, the UML (Unified Modeling Language) and UP (Unified Process) are two widely used concepts that play a significant role in the conception methodology of software development.

2.1 UML (Unified Modeling Language)

a standardized graphical modeling language used in software engineering to design, document, and communicate software systems. It provides a visual notation for representing different aspects of a system, including its structure, behavior, and interactions with other systems. UML includes a set of diagrams, such as class diagrams, use case diagrams, sequence diagrams, activity diagrams... that are used to model different aspects of a system [17].

2.2Unified Process (UP)

Is an iterative and incremental software development methodology that provides a framework for managing and developing software projects. UP is a flexible and adaptable methodology that can be customized to meet the specific needs of a project. It is based on a set of best practices that cover the entire software development lifecycle, from requirements gathering to deployment and maintenance. The use of UP helps to ensure that the software development process is well-structured, well-documented, and focused on delivering high-quality software [18].

2.3Uml strengths

There are several strength points of UML mentioning:

- UML provides a visual representation of the software system, making it easier to understand and communicate with stakeholders.
- UML is a standardized language, ensuring that UML diagrams can be easily understood and shared by developers worldwide.
- UML is flexible and can be adapted to different software development methodologies and tools.

- UML diagrams can be reused across different software systems, saving time and reducing development costs.

UML diagrams provide clear and concise documentation of the software system.

3 Analyse and Conception

3.1 Use case diagram

A use case diagram is a type of UML diagram that represents the interactions between actors (users or external systems) and a system under consideration. It is a high-level view of the system's functionality and is used to capture the requirements of the system and its intended use [19].

3.1.1 Use Case diagram roles:

The main roles of a use case diagram in software development, presented in bullet points:

- Provides a visual representation of the system's behavior from the perspective of the user or actor.
- Helps developers identify and understand the system's functionality and requirements.
- Serves as a communication tool between stakeholders, including developers, designers, and end-users.
- Can be used as a basis for testing and validating the software system [19].

3.1.2 Use case diagram elements

The following topics describe model elements in use-case diagrams:

- Use cases
  A use case describes a function that a system performs to achieve the user’s goal. A use case must yield an observable result that is of value to the user of the system.
- Actors
  An actor represents the role of a user that interacts with the system that you are modeling. The user can be a human user, an organization, a machine, or another external system.
- Relation
  In UML, a relationship is a connection between model elements. A UML relationship is a type of model element that adds semantics to a model by defining the structure and behavior between the model elements.

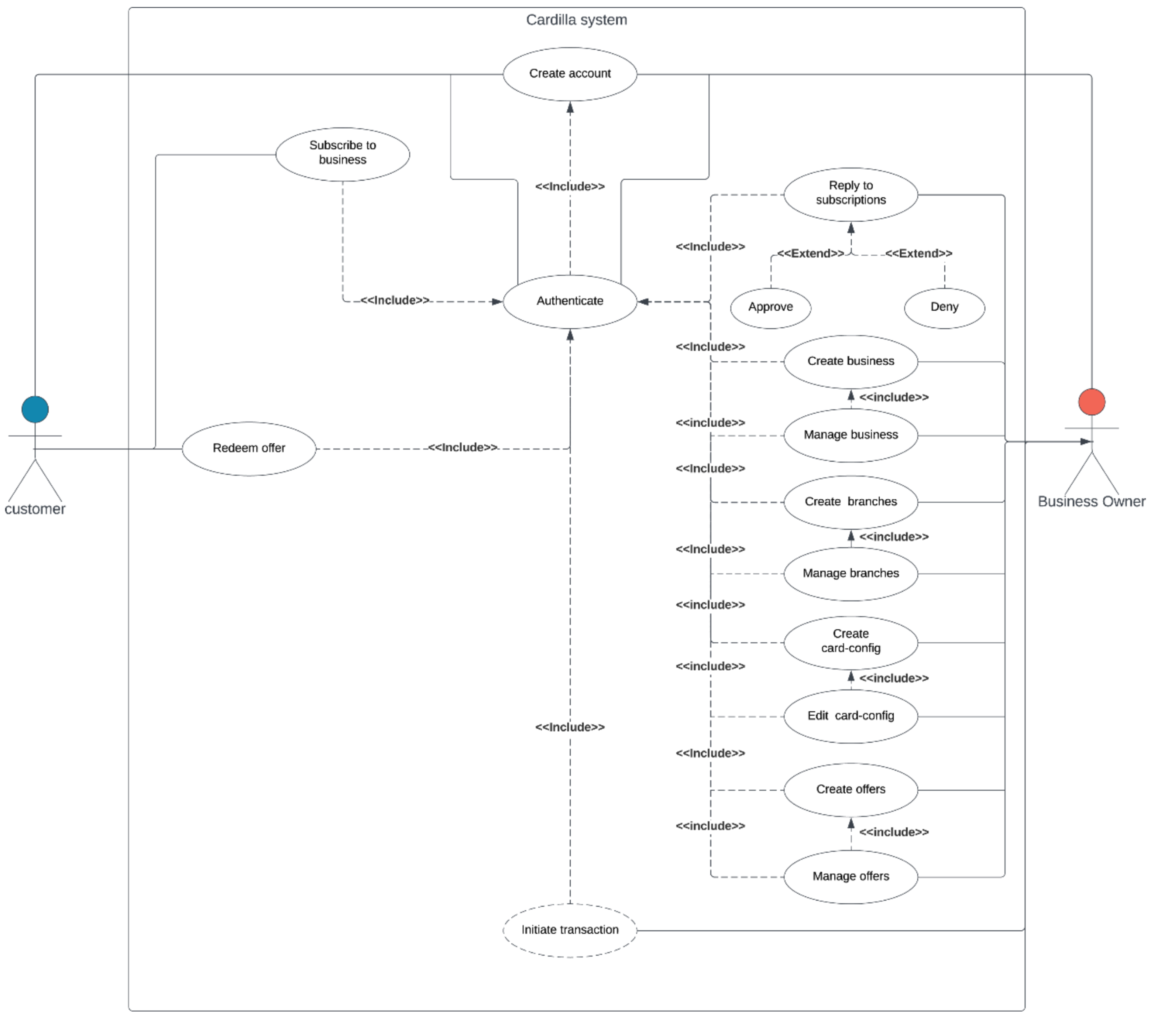


Figure III.1 use case diagram

3.1.3 Use case diagram textual description:

A use case (UC) highlights the functional relationships between actors and the system studied.

- **Precondition**: defines the conditions that must be satisfied so that the UC can start.
- **Postcondition**: defines what should happen when the UC is completed successfully, whether it was a nominal or an alternative scenario**.**

3.1.4 Use case << Create Account >>

This table (Table III.1) illustrates the use case of creating a new account

<u>**Identification**</u> **Use case name:** Create an account. **Goal:** Signup. **Actors:** Customer, Business Owner.
<u>**Sequencing**</u> The user launches the application. **Precondition:** None. **Nominal chains:** • The user accesses the account creation area. • The application requires you to fill out an information form. • The user enters the account information to create and confirm. **Alternative chains:** • Invalid data entered. • The account already exists. **Post-conditions:** Updating the database

Table III.1 Create account textual description

3.1.5 Use case << Authenticate >>

This table (Table III.2) illustrates the use case of Authenticate.

<u>**Identification**</u> **Use case name:** Authenticate. **Goal:** access app resources. **Actors:** Customer, Business Owner.
<u>**Sequencing**</u> The user launches the application. **Precondition:** Signed Up. **Nominal chains:** • The user accesses the login area. • The application requires you to fill out an information form. • The user enters the account information to login. **Alternative chains:** • Invalid data entered. • Incorrect credentials. **Post-conditions:** Access application resources (get access token)

Table III.2 Authentication textual description

3.1.6 Use case << Create & Manage Business | Branches | Card-config | offers>>

This table (Table III.3) illustrates the use case of Create Business | Create Branch | Create Card config | Create offer.

Identification **Use case name:** Create Business | Create Branch | Create Card config | Create offer. **Goal:** create a virtual business | Branch | Card config | Create offer. **Actors: Business** Owner.
Sequencing • The business owner launches the application. • Create a business, branch, card config, offer. **Precondition:** Authenticated. **Nominal chains:** • The user accesses the creation area. • The application requires you to fill out an information form. • The user enters the required information for the creation. **Alternative chains:** • Invalid data entered. • Business | branch | card config | offer already exists **Post-conditions:** Update database

Table III.3 Manage Business textual description

3.1.7 Use case << Subscribe to business >>

This table (Table III.4) illustrates the use case of Subscribe to business.

Identification **Use case name: Subscribe to business**. **Goal:**become a loyal customer to a business. **Actors:** Customer.
Sequencing The user launches the application. **Precondition:** Authenticated. **Nominal chains:** • The customer scans the business code. **Alternative chains:** • a blocked customer. • Customer already loyal customer **Post-conditions:** Update the database

Table III.4 Subscribe to business textual description

3.1.8 Use case << Reply to subscription >>

This table (Table III.5) illustrates the use case of Reply to subscription.

Identification **Use case name:** Reply to subscription. **Goal: reply to the customer requests**. **Actors:** Business Owner.
Sequencing • The user launches the application. • B-O access pending customers interface **Precondition:** Authenticated. **Nominal chains:** • The B-O accepts or decline customer request. **Post-conditions:** • Update the database. • Create client card

Table III.5 Reply to subscription textual description

3.1.9 Use case << initiate transaction >>

This table (Table III.6) illustrates the use case for initiate a transaction.

Identification **Use case name:** initiate a transaction. **Goal: initiate a transaction giveaway points or stamp card**. **Actors:** Business Owner.
Sequencing • The user launches the application. **Precondition:** Authenticated. **Nominal chains:** • The B-O accesses the transaction page. • B-O scans customer card. • The application requires you to fill out an information form. • The B-O enters the required information for the transaction. **Alternative chains:** • Invalid data entered. **Post-conditions:** • Update database

Table III.6 Initiate transaction textual description

3.1.10 Use case << Redeem offer >>

This table (Table III.7) illustrates the use case for Redeem offer.

Identification **Use case name: Redeem offer**. **Goal: redeem business offer** **Actors: Customer**.
Sequencing • The user launches the application. **Precondition:** Authenticated. **Nominal chains:** • The customer accesses the offer details page. • The customer redeems the offer. **Alternative chains:** • Insufficient points to redeem the offer. **Post-conditions:** • Update database

Table III.7 Redeem offer textual description

3.2Sequence diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner [17].

This table (Table III.8) shows the main sequence diagram components.

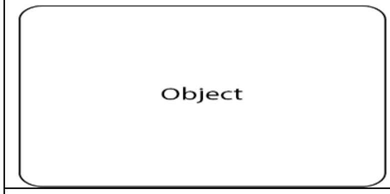
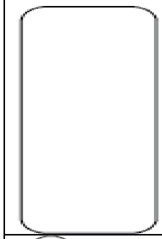
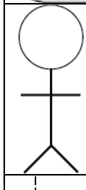
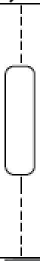
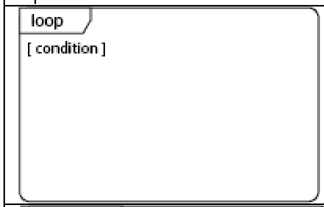
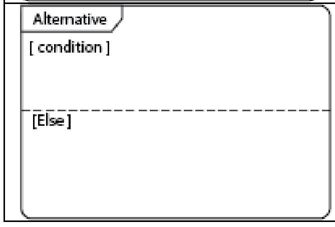
	Represents a class or object in UML. The object symbol demonstrates how an object will behave in the context of the system. Class attributes should not be listed in this shape.
	Represents the time needed for an object to complete a task. The longer the task will take, the longer the activation box becomes.
	Shows entities that interact with or are external to the system.
	Represents the passage of time as it extends downward. This dashed vertical line shows the sequential events that occur to an object during the charted process. Lifelines may begin with a labeled rectangle shape or an actor symbol.
	Used to model if/then scenarios, i.e., a circumstance that will only occur under certain conditions.
	Symbolizes a choice (that is usually mutually exclusive) between two or more message sequences. To represent alternatives, use the labeled rectangle shape with a dashed line inside.

Table III.8 Sequence diagram components

3.2.1 <<Authentication>> sequence diagram

This sequence diagram (Figure III.2) shows the basic flow of a username and password-based authentication process in our client-server application. It demonstrates how the user's credentials are verified. When authenticating the user, three cases may arise:

Data integrity is false, Data integrity is true, and user's credentials are invalid, Data integrity is true, and user's credentials are valid. Which explains the use of the "alt" operator. If the information provided is correct, then the database validates the credentials, if the user's credentials are validated, the system grants access to the appropriate interface.

On the other hand, if the user enters incorrect information, the system generates an error message and redisplays the authentication page from where the "loop" operator is used.

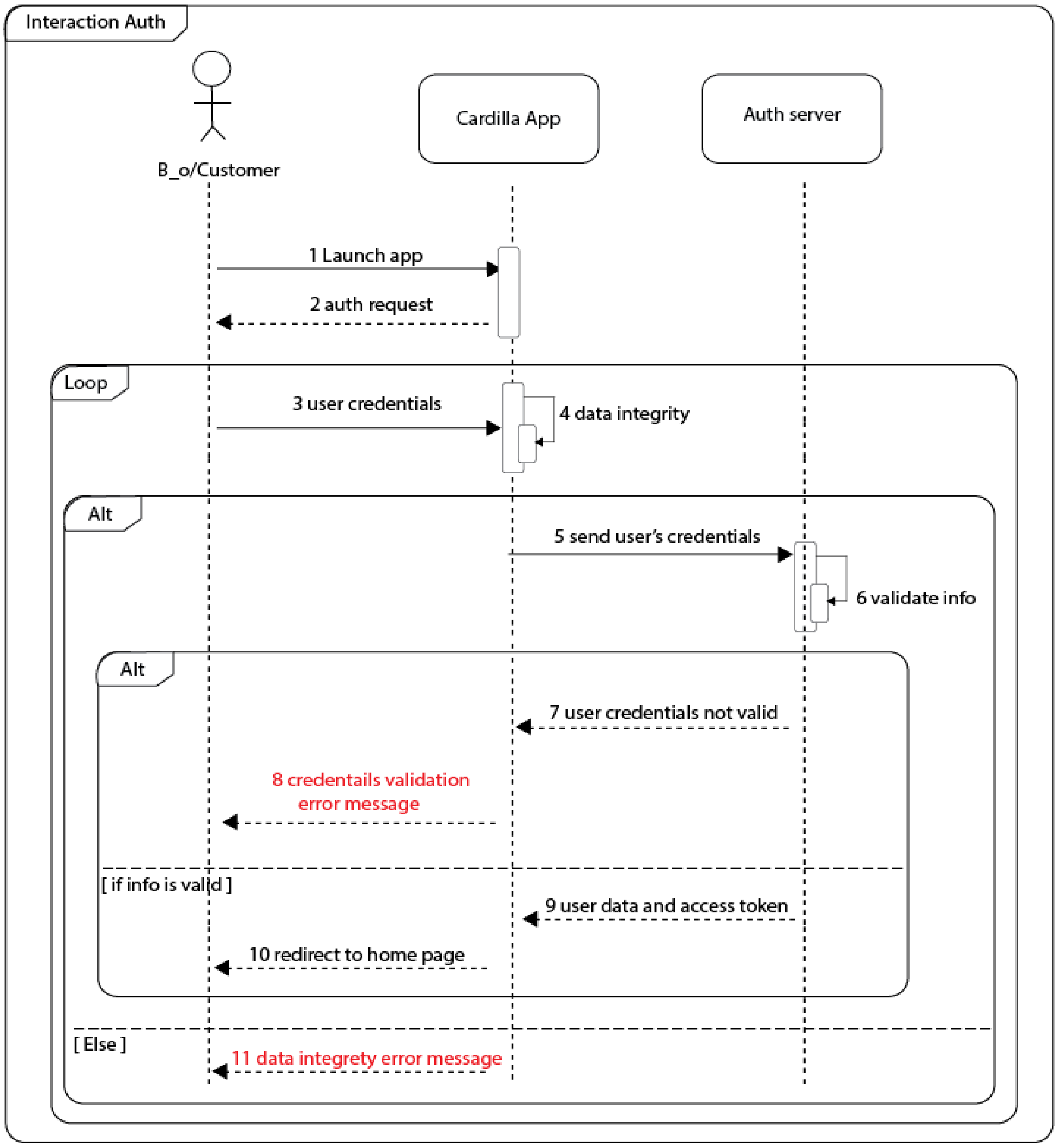


Figure III.2 Authentication sequence diagram

3.2.2 <<Create business >> sequence diagram

This sequence diagram (Figure III.3) demonstrates the steps involved in creating a new business. It shows how the user interface interacts with the server to store the information entered by the user, and how the application responds to the success or failure of the request.

The process of the business creation is almost the same as the offer, card-config, and branches creation.

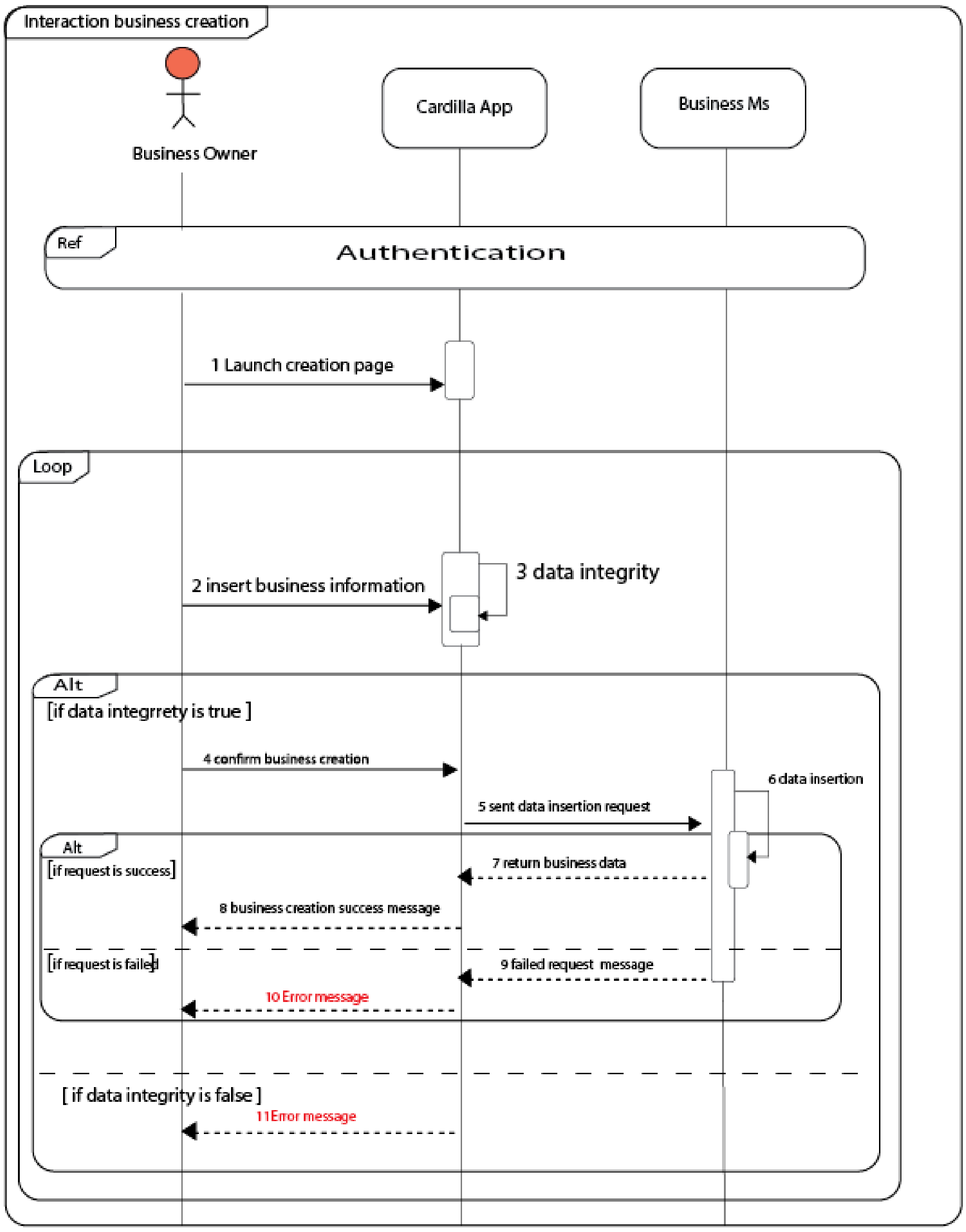


Figure III.3 Business creation squence diagram

3.2.3 <<Subscribe to a business >> sequence diagram

This sequence diagram (Figure III.4) demonstrates how a user can subscribe to a business by scanning the business's QR code. It shows how the application uses the camera toscan the QR code and decode its information, send the request and the different server-side possible responses.

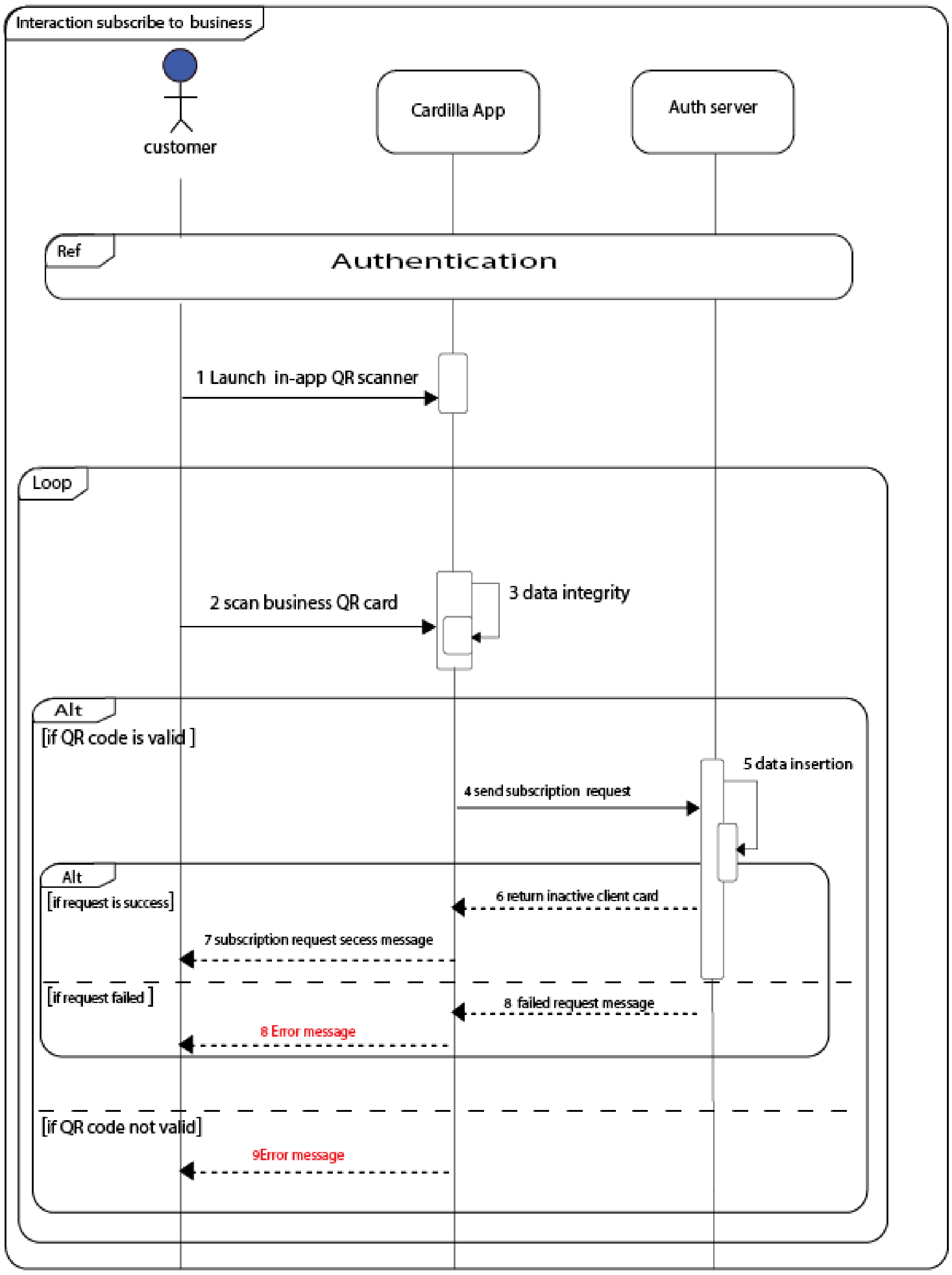


Figure III.4 Subscribe to business sequence diagram

3.2.4 <<Create Customer Card>> sequence diagram

This sequence diagram (Figure III.5) illustrates the high-level steps involved in creating a customer card, including the interactions between the customer, the business owner, the Cardilla app, and the backend system (LCardMs). Depending on the specifics of the system and implementation, additional steps and interactions may be involved, such as linking the card config (card design) with the customer card.

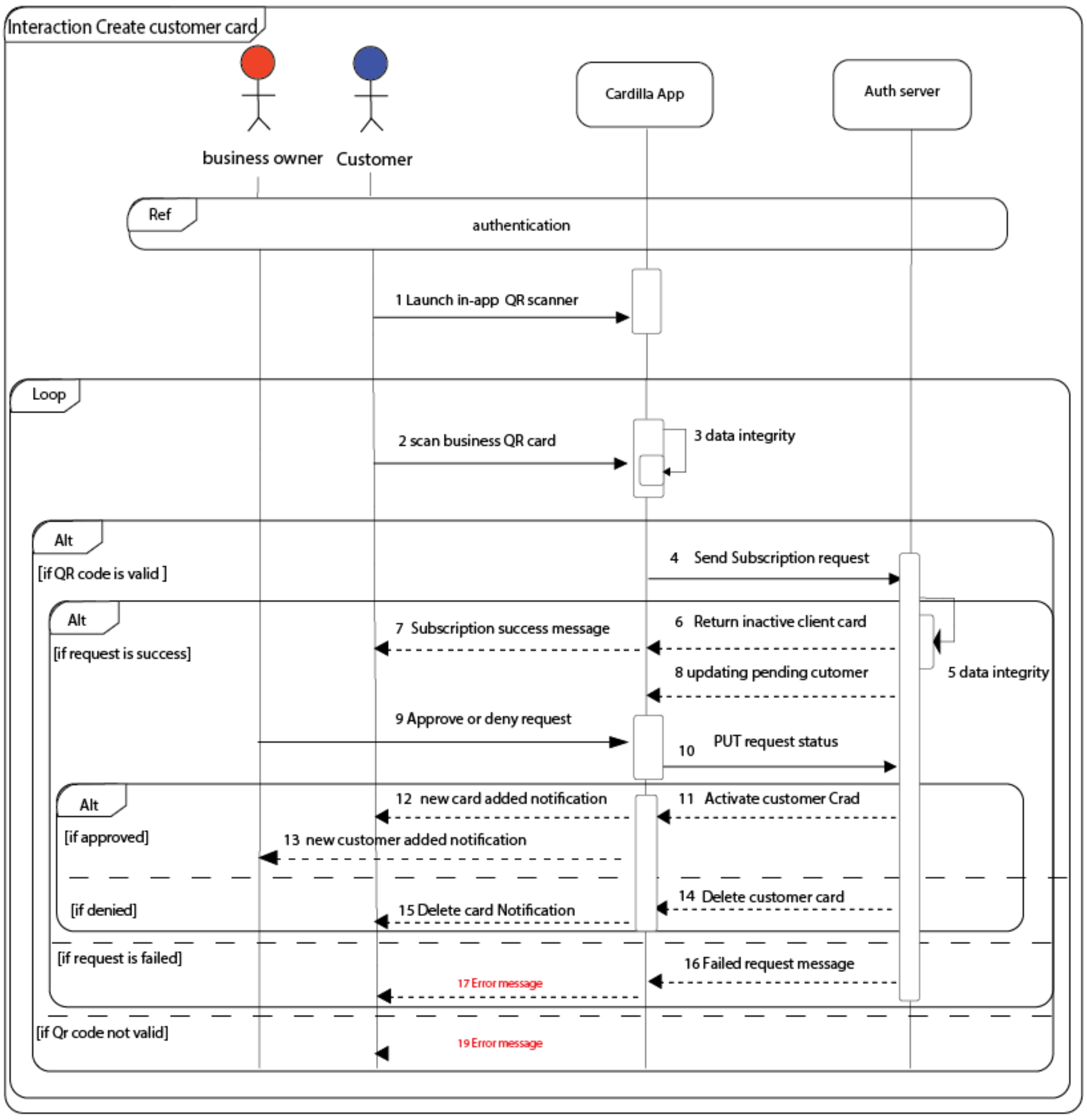

Interaction Create customer card
business owner
Customer
Cardilla App
Auth server
Ref
authentication
1 Launch in-app QR scanner
Loop
2 scan business QR card
3 data integrity
Alt
[if QR code is valid]
4 Send Subscription request
Alt
[if request is success]
7 Subscription success message
6 Return inactive client card
5 data integrity
8 updating pending cutomer
9 Approve or deny request
10 PUT request status
Alt
[if approved]
12 new card added notification
11 Activate customer Crad
13 new customer added notification
[if denied]
14 Delete customer card
15 Delete card Notification
[if request is failed]
16 Failed request message
17 Error message
[if Qr code not valid]
19 Error message

Figure III.5 Create customer Card sequence diagram

3.3 Class diagram

After the detailed study of use cases, sequence and components diagrams we have deduced the overall class diagram of the system. This diagram is considered the final phase of the theoretical design of our system and will be taken as the reference from which the software development and writing the source code will take place.

A class diagram in the Unified Modeling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects.

The architecture of our app is based on microservices, which means that the app's services are split into multiple smaller services. This splitting also results in the classes being divided between these services [17].

The following table (Table III.9) shows each class and which microservice it belongs to:

Class	Microservice
User, Customer, Business_Owner	AuthMs
Business, Branch	BusinessMs
TransactionGain,TransactionRedeem	TransactionMs
ClientCard, CardConfig	LCardMs
Offer	OfferMs

Table III.9 Microservices classes

This division of classes and microservices helps to keep the app's architecture modular and scalable. Each microservice can be deployed independently and can communicate with the other microservices through APIs or message queues, making it easier to maintain and update the app's functionality. The next diagram (Figure III.6) represents the Cardilla class diagram.

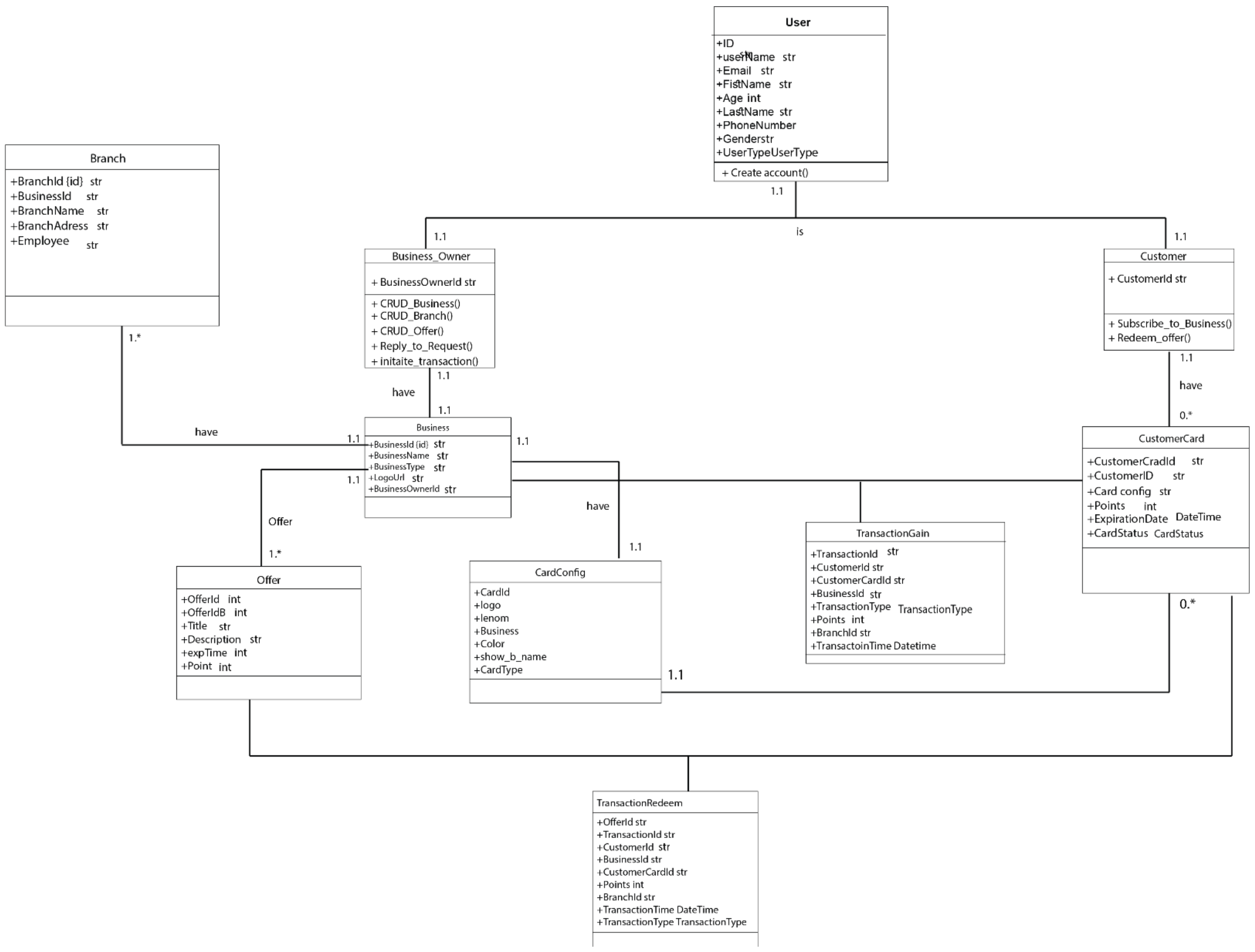


Figure III.6 cardilla class diagram

4. Conclusion

In Chapter three, we provided the design of our solution, we started by explaining the conception methodology, then we explored different diagrams (use case, sequences, class) that explain our solution. In the next chapter we will have an overview of the application in terms of development and walkthrough the interfaces.

CHAPTER IV

IMPLEMENTATION

1.Introduction:

In this chapter, we will provide an outline of the development process of the << Cardilla >> application. We will begin by discussing the development environment we used for building the app, followed by a detailed description of the backend various backend technologies followed by the app global diagram.

Finally, we will provide an overview of the different interfaces of our application, which will allow users to easily navigate through the app and access the different features and functionalities it offers.

2. Development environment

2.1Hardware

This table (Table IV.1) represents the different set of hardware we used to develop our application.

Material	Characteristic
PC 1	Lenovo IdeaPad 320 Processor: Intel(R) Core (TM) i7-7500U CPU @ 2.70GHz Memory: 8.00 GB Operating system: Windows 10 professional 64 bit
PC 2	Lenovo IdeaPad 320 Processor: Intel(R) Core (TM) i7-7500U CPU @ 2.70GHz Memory: 20.00 GB Operating system: Windows 10 professional 64 bit
Mobile simulators	Huawei y9 2019, 4/64 Samsung Galaxy a5 2016, 4/64

Table IV.1 hardware

2.2Software environment

2.2.1 Front-end

***Figma (UI/UX design):**

Figma is a collaborative web application for interface design, with additional offline features enabled by desktop applications for macOS and Windows. The feature set of Figma focuses on user interface and user experience design, with an emphasis on real-time collaboration, utilizing a variety of vector graphics editor and prototyping tools. The Figma mobile app for Android and iOS allows viewing and interacting with Figma prototypes in real-time on mobile and tablet devices [20].

***Mobile Dev**

Flutter

Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, and the web from a single codebase [21].

***Web Dev**

In Web development we use various programming languages and technologies, such as HTML, CSS, JavaScript, Bootstrap, jQuery, also we use VScode editor for coding.

2.2.2 Backend

***Java**

Java is a programming language and computing platform first released by Sun Microsystems in 1995. It has evolved from humble beginnings to power a large share of today's digital world, by providing the reliable platform upon which many services and applications are built. New, innovative products and digital services designed for the future continue to rely on Java, as well [22].

***Spring boot**

Java Spring Boot is an open-source tool that makes it easier to use Java-based frameworks to create microservices and web apps. For any definition of Spring Boot, the conversation must start with Java one of the most popular and widely used development languages and computing platforms for app development [23].

***Keycloak (IAM server)**

Keycloak is an Open-Source Identity and Access Management solution for modern Applications and Services. It allows single sign-on with identity and access management aimed at modern applications and services [24].

***NoSQL DBs**

A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases [25].

***Mongodb**

MongoDB is a scalable and flexible document database with querying and indexing. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server-Side Public License (SSPL) which is deemed non-free by several distributions. MongoDB is a member of the MACH Alliance [25].

***Postman**

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster [26].

***Docker**

Docker is a set of platforms as a service (PaaS) product that uses OS-level virtualization to deliver software in packages called containers. The service has both free and premium tires. The software that hosts the containers is called Docker Engine. It was first started in 2013 and is developed by Docker, Inc [27].

3. Cardilla global architecture

As discussed earlier, our architecture is mainly based on microservices, this latter use different components to interact with each other as depicted below (Figure IV.1). Noting that all microservices communicate together, we have just projected one example of <<subscribing to business>> use case. When a request is made from an endpoint (mobile or web app) to the system it passes through the following stations:

1. **Request from the Endpoint:** The Http request originates from the client application, containing the necessary data and parameters. In our example the client scans the business Qr code and a request is sent with the information of both the business and the client.
2. **API Gateway (Spring cloud Gateway):** The API gateway is the entry point for incoming requests. It receives the request from the endpoint and acts as a mediator between the client and the microservices.
3. **Authentication and Authorization:** The API gateway verifies the authenticity of the (access token) through Keyckloak (auth server). It ensures that the client is authorized to access the requested resources.
4. **Request Routing:** Based on the requested endpoint or URL, the API gateway routes the request to the appropriate microservice.in our case the BffMs which aggregate data and ensure the access of different microservices with just one request.
5. **Service Communication:** after receiving the request the BffMs sends opens canals of communication with:
   a. **BusinessMs: (synchronous communication through OpenFeign)** creates a pending customer in the business database.
   b. **LCardMs: (synchronous communication through OpenFeign)** creates an inactivated customer card in the LCardMs database with appropriate data retrived from the BusinessMs and the Http request, it will be activated whene the business owner accepts the subsctibtion request.
   c. **NotificationsMs: (asynchronous communication through Kafka)** notify the business owner with the subscription request.
6. **Response generation:** After processing the request, the microservice generates a response containing the requested the result of the operation. The response is then sent back to the API gateway. And finally to the original endpoint that initiated the request.

As we noted above, almost all microservices communicates with each other in several ways, and, each request has it's own path in the system.

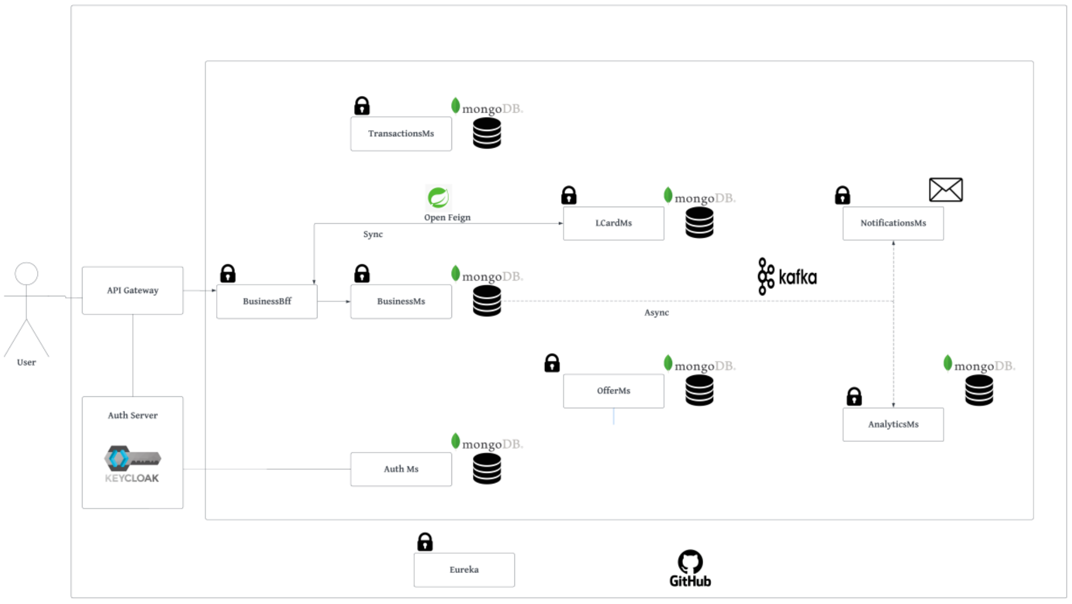
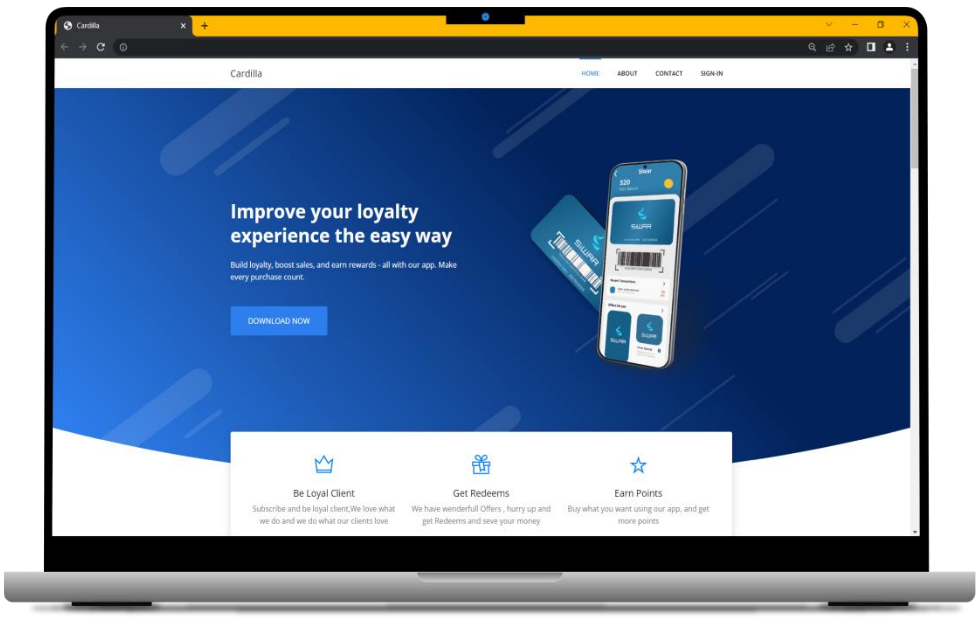


Figure IV**.1 Cardilla global architecture Subscribe to business exaùple**

4. Implementation and deployment

In this section of the thesis, we will showcase the primary characteristics of our application. This will be accomplished through the depiction of several interfaces that will provide a visual representation of the key functionalities and user interactions within the application.

* Landing Page



Improve your loyalty experience the easy way

Build loyalty, boost sales, and earn rewards - all with our app. Make every purchase count.

Subscribe and be loyal client,We love what we do and we do what our clients love

We have wenderfull Offers , hurry up and get Redeems and seve your money

Buy what you want using our app, and get more points

Figure IV.2 Landing page

* Business Home Page

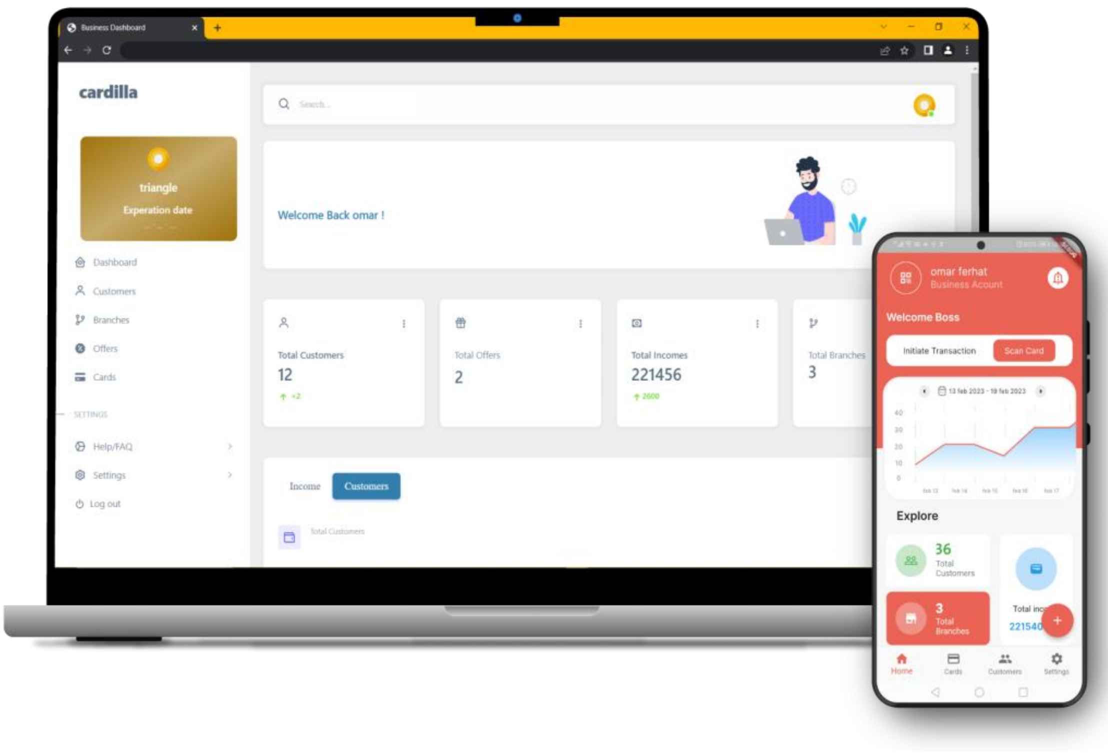


Figure IV.3 Business Home page

This page (Figure IV.3) represents the business side's homepage which features a modern and clean design, with a prominent header section, and a main content area that includes metrics and analysis, a sidebar with links to various pages, such as Customers, Offers and Branches pages, and the loyalty card preview. The web version has a sidebar for navigation, while the mobile version features a bottom navigation bar for easy access to the various pages. The mobile version is designed to be responsive and optimized for small screens, with a focus on delivering the most important information and features in a user-friendly format.

* Edit Card Page

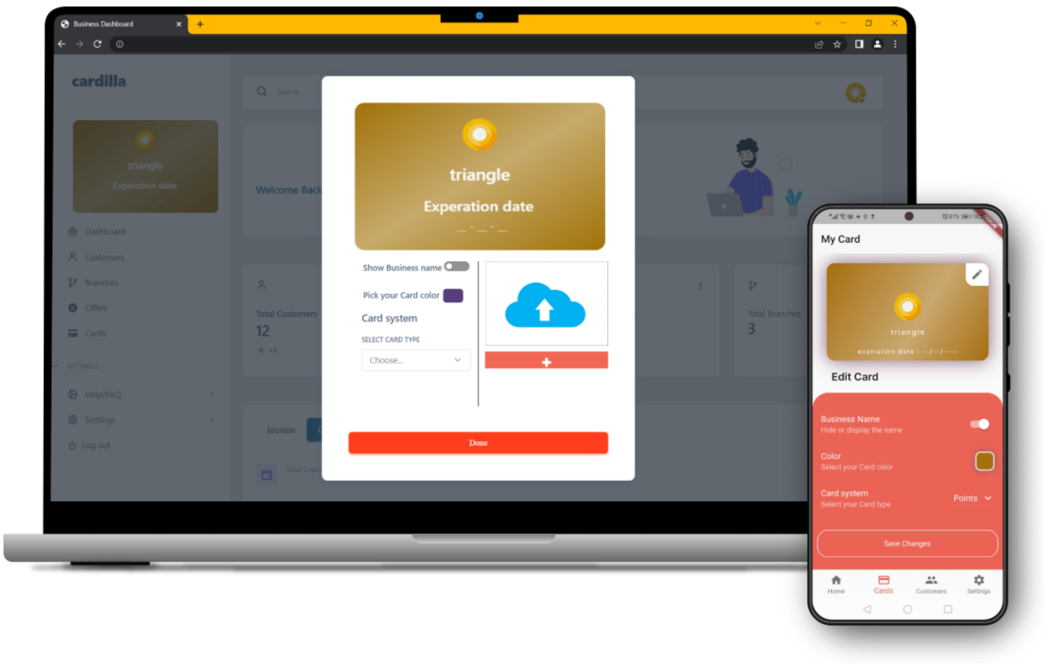


Figure IV**.4 Edit card page**

The Edit Card page (Figure IV.4) allows business owners to customize their loyalty cards by editing the color, logo, card type, and business name appearance. On the web version, the Edit Card page is displayed as a pop-up window, while on the mobile version it is a separate page that can be accessed via the Card section in the bottom navigation bar. Both versions feature a simple and intuitive form with clear labels. Once the business owners have made their changes, they can save the new settings and view their updated loyalty card.

* Set-up business Page

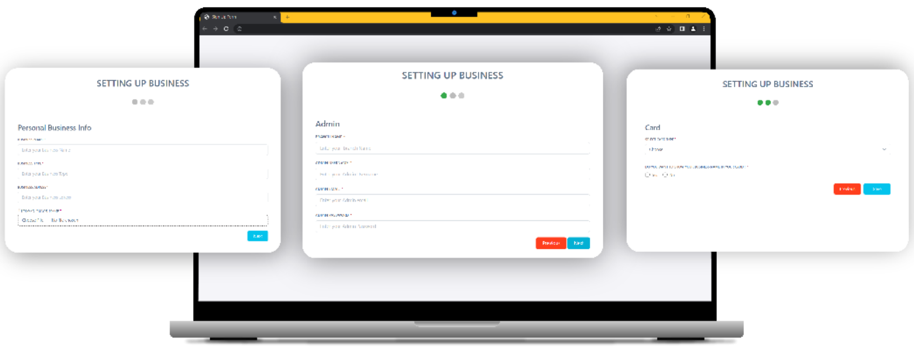
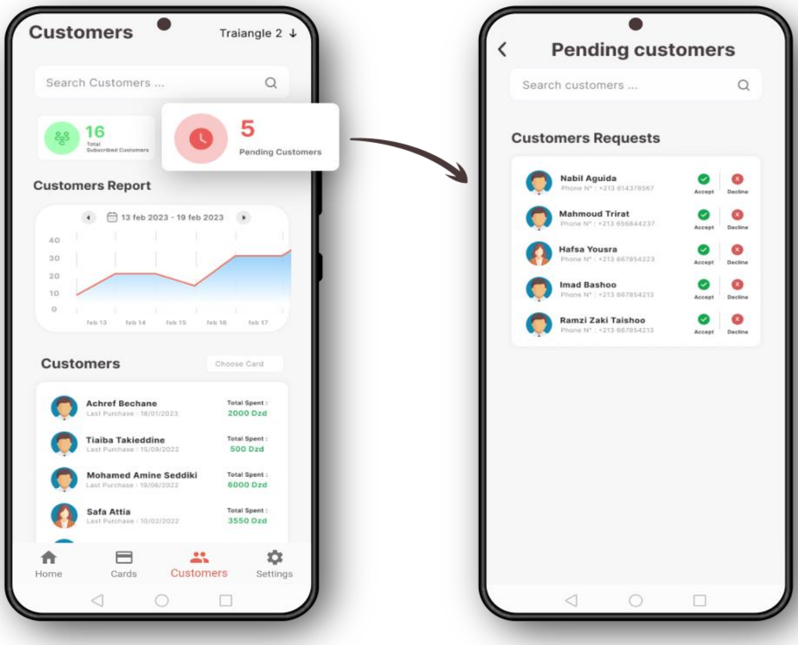


Figure IV**.5 Setup business page**

The Set-Up Business Page (Figure IV.5) allows business owners to effortlessly create and customize their businesses profiles. Whether accessed through the web version or mobile application, the Set-Up Business Page guides users through a seamless step-by-step process. The page is designed as a stepper-based interface, ensuring that every important detail is captured. Business owners can easily fill in essential information such as business name, branches, locations, admins, and customize their loyalty cards.

* Pending customers Page

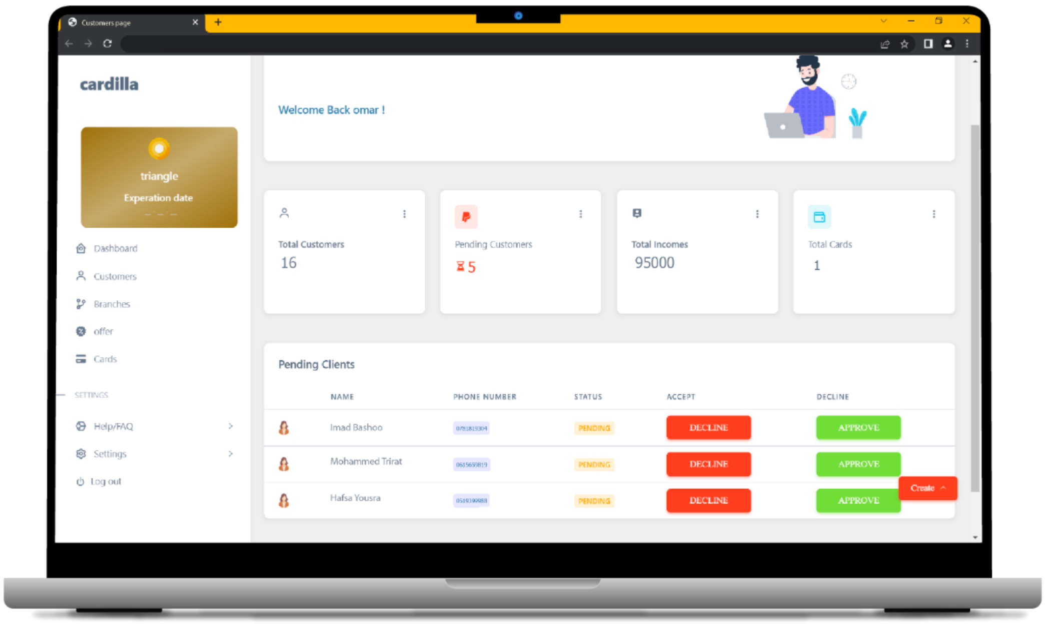

Pending customers

Search customers ...

Customers Requests

Nabil Aguida Phone N° : +213 614378567	Accept	Decline
Mahmoud Trirat Phone N° : +213 656844237	Accept	Decline
Hafsa Yousra Phone N° : +213 667854223	Accept	Decline
Imad Bashoo Phone N° : +213 667854213	Accept	Decline
Ramzi Zaki Taishoo Phone N° : +213 667854213	Accept	Decline

Figure IV.6 Pending customer page for mobile


Welcome Back omar !

Total Customers 16

Pending Customers 5

Total Incomes 95000

Total Cards 1

Pending Clients

NAME	PHONE NUMBER	STATUS	ACCEPT	DECLINE
Imad Bashoo	0781819304	PENDING	DECLINE	APPROVE
Mohammed Trirat	0615659819	PENDING	DECLINE	APPROVE
Hafsa Yousra	0519199988	PENDING	DECLINE	APPROVE

Figure IV.7 Pending customer page for web

The Accept/Decline Customer Request feature (Figures IV.6, IV.7) provides business owners with a straightforward and efficient way to manage customer loyalty requests. Accessible through both the web and mobile versions. When a customer scans the business's Qr-code, a request will be sent to the business owner, they can then accept or decline each request with just a single click.

* Initiate transaction

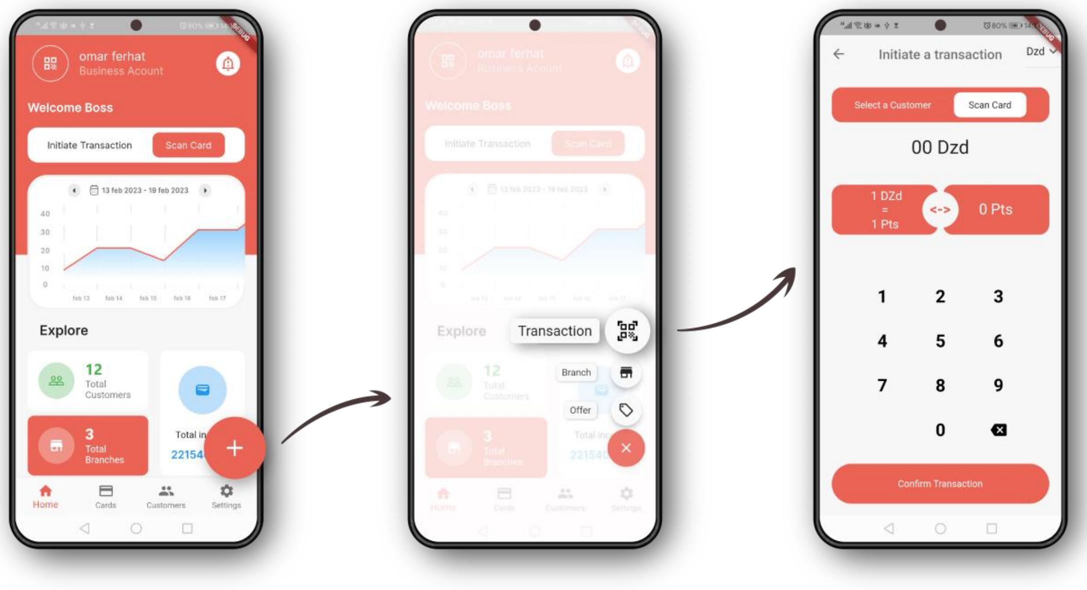


Figure IV.8 Initiate transaction page

The Transaction Page (Figure IV.8), it can be accessed from the business's home page through the initiate transaction button at the top of the home page or transaction section in the floating action button options. The page design is user-friendly, displaying essential details such as the transaction amount (in Dzd or in points) and the customer information. After selecting the customer by scanning his card, the business owner can give away a number of points directly or he can automate the points calculation with a predefined model with the money spent as an input and points as an output.

* <u>Home and card details pages</u>

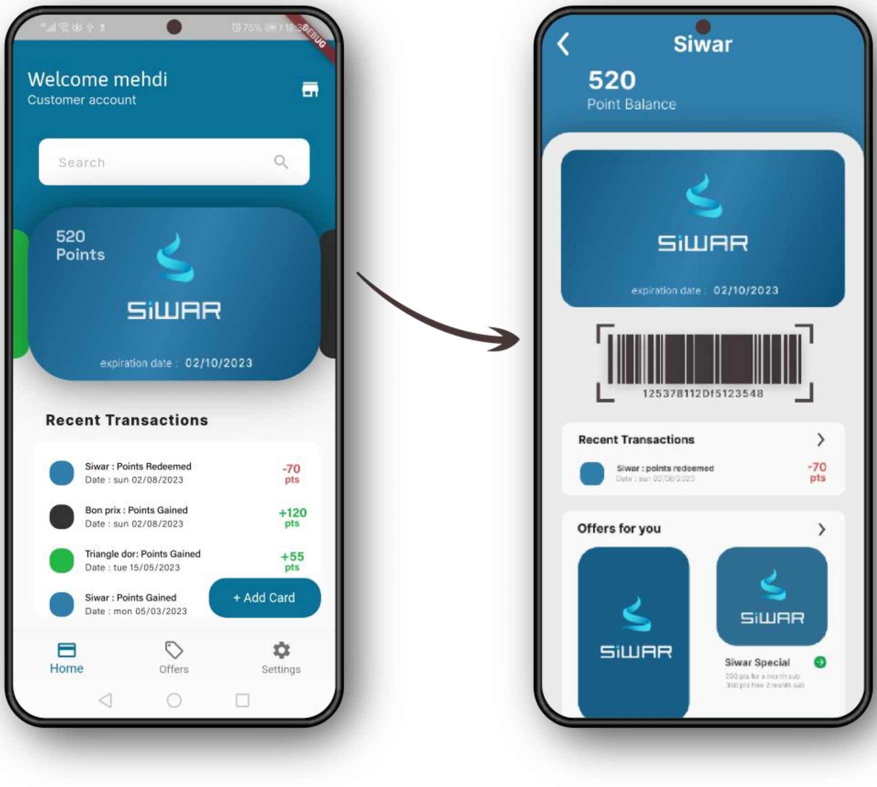


Figure IV**.9 Home & Card details pages**

The customer-side home page of Cardilla (Figure IV.9) presents a simple and organized layout. It consists of two main sections: "Loyalty Cards" and "Transactions." The Loyalty Cards section displays a list of cards associated with the customer's account, each represented by its logo and business name. Tapping on a card navigates the user to the card details page which presents points balance, status, offers, and transaction history. The Transactions section shows a chronological list of recent transactions, featuring business logos, dates, and summaries. A bottom navigation bar offers quick access to other screens (Offers, and Settings), ensuring convenient navigation throughout the app

* <u>Redeem offer page</u>

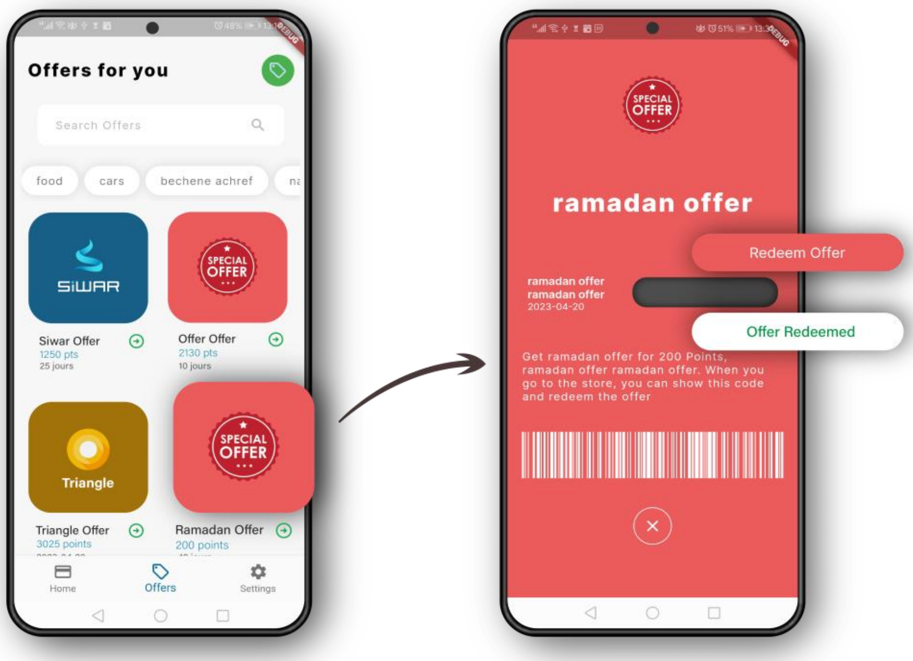


Figure IV**..10 Redeem Offer page**

The offers page (Figure IV.10) showcases a range of offers categorized by different types of businesses. Customers can easily explore the available offers and tap on a specific offer to view its details. The offer details page provides essential information such as the offer description, required points, expiration date, and any applicable terms and conditions. It also includes a status button indicating whether the offer has been redeemed or not. If customers have accumulated enough points to meet the offer's requirements, they can tap the redeem button to claim the offer and enjoy the associated benefits.

5. Conclusion

In this chapter, we provided an overview of the development environment, and we discussed the implementation and deployment of our solution.

GENERAL CONCLUSION

Digitalization has brought forth the widespread use of web applications, and it is impossible to overlook the convenience they offer in simplifying our lives. Computers and mobile devices have become an essential tool for users and consumers across various domains, including services, education, management, and retail. These applications provide a close and accessible platform for individuals to access services, learn, organize their tasks, and engage with content.

Our project falls within the realm of digitalization, focusing on the development and implementation of a web & mobile application centered around virtual loyalty cards for businesses. The aim is to address the evolving needs of customers in their daily shopping experiences and provide them with a seamless solution for accessing and redeeming various offers and rewards. The loyalty app will revolutionize traditional loyalty programs by offering customers a digital platform to manage their virtual loyalty cards, track their accumulated points, and easily redeem exclusive offers and discounts, and give businesses the opportunity to track their customers and business performance.

To accomplish this, we began by introducing the various strategies involved in the development. We provided a comprehensive overview of the project framework and outlined our design methodology, which involved utilizing UML as a modeling language and employing the Unified Process (UP) as our approach. Next, we conducted a preliminary study to identify the key stakeholders who would interact with the system being implemented. We meticulously monitored the specification of functional requirements by creating a use case diagram and performed an in-depth analysis of needs using sequence diagrams. This meticulous approach ensured a thorough understanding of the system's requirements and paved the way for successful implementation.

Finally, we have provided an overview of the app development tools and programming languages utilized in the implementation of our application, The careful consideration of these development tools and languages played a crucial role in successfully bringing our application to life.

The project has proven to be highly advantageous for us, as it has enriched our understanding and expertise in both theoretical and practical aspects. Through this endeavor, we have gained valuable knowledge and insights in the field of mobile and web development. The project served as a platform for us to explore new concepts (Microservices architecture, the cloud, containerization....), acquire new skills, and expand our proficiency in the development techniques.

Looking ahead, we envision the design of a more comprehensive system that offers a wider range of services and functionalities. Our goal is to continuously learn and gather feedback to further enhance the application. The architecture of the app, based on microservices, enables the flexibility and scalability necessary to accommodate the addition of new features seamlessly. This modular approach allows us to independently develop and deploy new services without disrupting the overall system, and give the next students promotions the opportunity to handle a project that is ready for expansion in:

- Data mining: by extracting valuable insights from the vast amount of customer data collected. With data mining techniques, businesses can analyze patterns, trends, and customer behavior to make informed decisions and optimize system.
- ERP (Enterprise Resource Planning): The expansion of the system to cover ERP for small businesses will bring enhanced efficiency and streamlined processes to their operations. By integrating ERP functionalities, small businesses can optimize their resource planning, inventory management, and financial processes, leading to improved productivity and overall business performance.
- Cashless payments: Enabling cashless payments (EDahabia, CIB, prepayed balance) within the system will offer businesses and customers a convenient and secure way to conduct transactions. By incorporating cashless payment options, businesses can streamline the checkout process
- E-commerce: Expanding the system to include eCommerce capabilities will empower businesses to reach a wider customer base and tap into the growing online market.

REFERENCES

[1] R. Chouffani, "techtarget," 19 augest 2022. [Online]. Available: https://www.techtarget.com/searchcustomerexperience/tip/4-types-of-loyalty-programs-and-their-benefits. [Accessed march 2023].

[2] C. Harris, "Développement logiciel," [Online]. Available: https://www.atlassian.com/fr/microservices/microservices-architecture/microservices-vs-monolith. [Accessed march 2023].

[3] P. M. a. T. Grance, "The Nist Definition of Cloud Computing," Gaithersburg, September 2011.

[4] S. Aurangabad, 30 March 2023. [Online]. Available: https://www.geeksforgeeks.org/layered-architecture-of-cloud/. [Accessed april 2023].

[5] L. S. Jonathan Johnson, "BMC," 8 march 2021. [Online]. Available: https://www.bmc.com/blogs/microservices-architecture/. [Accessed mai 2023].

[6] "Vmware," [Online]. Available: https://www.vmware.com/solutions/virtualization.html#:~:text=Virtualization%20relies%20on%20software%20to,of%20scale%20and%20greater%20efficiency. [Accessed april 2023].

[7] K. B. a. B. Kirsch, "TechTarget," [Online]. Available: https://www.techtarget.com/searchitoperations/definition/virtualization. [Accessed april 2023].

[8] "Oracle VM Concepts guide for release 3.3," ORACLE, [Online]. Available: https://docs.oracle.com/cd/E50245_01/E50249/html/vmcon-hypervisor.html. [Accessed april 2023].

[9] J. MACPHERSON, "Park Place," fzbruary 2022. [Online]. Available: https://www.parkplacetechnologies.com/blog/what-is-hypervisor-types-benefits/. [Accessed mai 2023].

[10] Sosinsky.B, "Virtualisation and cloud computing," Auerbach, 2011.

[11] S. Force. [Online]. Available: https://www.salesforce.com/products/platform/best-practices/benefits-of-cloud-computing/. [Accessed mai 2023].

[12] A. S, "Tech target," Augest 2021. [Online]. Available: https://www.techtarget.com/searchapparchitecture/definition/microservices. [Accessed mai 2023].

[13] OptiSol, "8 Core Components of Microservice Architecture," [Online]. Available: https://www.optisolbusiness.com/insight/8-core-components-of-microservice-architecture. [Accessed April 2023].

[14] M. team, "microsoft," 13 april 2022. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture. [Accessed mai 2023].

[15] T. F. D. Ackerson, "semaphore," september 2022. [Online]. Available: https://semaphoreci.com/blog/deploy-microservices. [Accessed mai 2023].

[16] RedHat, "What is containerization," 8 april 2021. [Online]. Available: https://www.redhat.com/en/topics/cloud-native-apps/what-is-containerization.

[17] OMG, "OMG Unified Modeling Language Specification," 2007. [Online]. Available: www.omg.org.

[18] J. O. a. U. Donins, "Science Direct," 2017. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/unified-process.

[19] M. Fowler, UML Distilled A brief guide to the Standard Object Modeling Language.

[20] C. Hope, "Computer Hope," october 2022. [Online]. Available: https://www.computerhope.com/jargon/f/figma.htm. [Accessed march 2023].

[21] f. team. [Online]. Available: https://docs.flutter.dev/. [Accessed mai 2023].

[22] IBM, "IBM," [Online]. Available: https://www.ibm.com/topics/java. [Accessed mai 2023].

[23] "spring," [Online]. Available: https://spring.io/projects/spring-boot. [Accessed mai 2023].

[24] B. Żyliński, "DZone," october 2021. [Online]. Available: https://dzone.com/articles/what-is-keycloak-and-when-it-may-help-you. [Accessed mai 2023].

[25] M. team. [Online]. Available: https://www.mongodb.com/fr-fr/nosql-explained. [Accessed mai 2023].

[26] "Corefy Developer Docs," [Online]. Available: https://corefy.com/docs/integration/postman-collections/. [Accessed mai 2023].

[27] "docker docs," [Online]. Available: https://docs.docker.com/get-started/overview/. [Accessed mai 2023].

[28] P. M. a. T. Grance, "The NIST Definition of cloud computing," Gaithersburg, september 2011.

[29] T. Foley, "2nd watch," augest 2021. [Online]. Available: https://www.2ndwatch.com/blog/back-to-the-basics-the-3-cloud-computing-service-delivery-models/. [Accessed March 2023].

[30] Stocard, [Online]. Available: https://play.google.com/store/apps/details?id=de.stocard.stocard&hl=fr&gl=US. [Accessed march 2023].

[31] Yollty S.A., [Online]. Available: https://play.google.com/store/apps/details?id=com.yollty.yollty&hl=fr&gl=US. [Accessed mai 2023].

[32] Sylvia van Os, [Online]. Available: https://play.google.com/store/apps/details?id=me.hackerchick.catima&hl=fr&gl=US. [Accessed mai 2023].

[33] "Spring," [Online]. Available: https://spring.io/projects/spring-boot.