Ministry of Higher Education
and Scientific Research

ACADEMIC YEAR
2023/2024

EPI
Private International Higher
School of Polytechnic

END OF YEAR PROJECT

Field of study: **Software Engineering**

Entitled

Design and Development of a Loan Application Website

Author
Sami Ayachi

Supervisor
Amen Ajroud

Dedication

At the end of the year, I am dedicating this project

To anyone who accompanied me by a word, a gesture or even a thought,

To all those who have contributed,

Thank you for everything.

This being for the general, now comes the particular

With a well-deserved distinction

For my family

Without them I would never have made it,

To my friends who accompanied me on a long journey Dhia,

Ala and all the others

All of you have been my best cheerleaders.

With Love

Sami Ayachi

Acknowledgment

Primarily, I would thank Allah who gave me grace and blessing to complete this project.

In the term of the end-of-year project., I would like to express my deep gratitude to anyone who has contributed, near or far, to the outcome of this work, in particular:

I would like to express my sincere gratitude to Mr. **Amen Ajroud** for his supervision, insightful advice, and guidance. His close attention to the progress of this project, along with his constructive remarks, have been invaluable. I am also grateful for the confidence he placed in me, and I have greatly appreciated the opportunity to work under his guidance.

List of Acronyms & Glossary

- **API:** Application Programming Interface.
- **HTTP:** Hypertext Transfer Protocol..

Contents

List of Figures

List of Tables

General Introduction

In today's fast-paced digital world, obtaining a loan efficiently and conveniently is crucial for many individuals and businesses. Traditional loan application methods are often lengthy, complex, and burdened with administrative formalities. To address this, our project introduces an innovative web platform designed to streamline and simplify the loan application process. This website offers a user-friendly interface that guides users through each step of their loan application, aiming to demystify the process and make it accessible and straightforward for everyone. Our platform also allows agents to manage loan applications efficiently. Agents can review, approve, or reject applications, communicate with applicants, and oversee the entire loan process from start to finish. Through our platform, we aim to reduce loan processing times and increase approval rates using a streamlined process. This leads to higher applicant satisfaction and more efficient agent management thanks to detailed data insights, comprehensive statistics on the number of clients, loan types, approval rates, and user engagement that provide valuable information for continuous improvement.

It is in this context that our final year project titled "Design and Development of a Loan Application Website" is set. Our final report comprises three main chapters distributed as follows:

In the first chapter, we are going to see the Preliminary Study

In the first chapter, we conduct a preliminary study that lays the foundation for our project. This includes an introduction to the project, a presentation of the project objectives, and an analysis of the problem statement. We critically examine existing

solutions to highlight their limitations and identify the gaps our platform aims to fill. Following this, we present our proposed solution and detail the adapted methodology for the project's execution. Task planning is also covered, outlining the key activities and milestones, and the chapter concludes with a summary of the preliminary findings.

In the second chapter, we are going to see the Analytical and Conceptual Study

The second chapter delves into the analytical and conceptual aspects of our project. It starts with an introduction to the chapter's content and proceeds to specify the functional and non-functional requirements of our platform. We utilize the UML (Unified Modeling Language) for modeling the system, which includes functional modeling to identify actors and develop an overall use case diagram. This is followed by the refinement of select use cases, a comprehensive class diagram, and sequence diagrams that depict the interactions within the system. The chapter concludes with a summary of the analytical and conceptual findings.

In the third chapter, we are going to see the Implementation

The third chapter focuses on the implementation phase of our project. It begins with an introduction to the implementation process and an overview of the application architecture. We describe the hardware and software environment, including the development languages and tools used. A detailed presentation of the user interface is provided, showcasing the design and functionality of the platform. The chapter concludes with a summary of the implementation outcomes and highlights any challenges faced during the development process.

The report concludes with a synthesis of the results obtained and future prospects for our online loan application platform, followed by a bibliography listing the references and resources used during our project.

Chapter 1

General Context

1.1 Introduction

The first chapter sets the work in its general context. It consists of five parts. The first part aims to present the project to be carried out. The second part addresses the problem statement. In the third part, I will examine existing similar applications and, with a critical eye, identify their limitations and take them into consideration in order to propose a more appropriate application. In the fourth part, I will present and defend the methodological approach to be adopted for guiding the project. Finally, I will plan the tasks for its execution.

1.2 Project Presentation

My project involves the study, design, and implementation of a web application aimed at simplifying the presentation of loan applications and streamlining the processing of these applications by banking agents. It will also enable managers to have detailed statistics on the number of clients and types of loans.

1.2.1 Project Challenges

Nowadays, inflation is a major problem affecting individuals and businesses. Inflation refers to the increase in prices of goods and services over time, thereby reducing consumer purchasing power. This situation has led to an increased need for loans as people seek to maintain their standard of living or finance significant purchases. Additionally, businesses may need loans to manage higher operational costs or invest in new opportunities arising from changing market conditions. However, the loan application process has become more complex and lengthy. Financial institutions have implemented stricter requirements to ensure that borrowers can repay their loans, adding to the complexity and length of the application process. Lenders require more detailed financial information and proof of income, which can delay approval times, creating frustrations and challenges for both borrowers and lenders. Moreover, the lack of clarity and visibility in the loan approval process makes it difficult for borrowers to understand what is required and how long it will take. Poor communication between the loan officer and the borrower can lead to misunderstandings and delays. The final decision often rests with the loan officer, whose judgment may not always seem practical or sensitive to the borrower's situation. Therefore, we propose establishing clear and open lines of communication between borrowers and loan officers to help simplify the process and reduce misunderstandings. Financial institutions should be encouraged to simplify documentation requirements without compromising the rigor of their evaluations. Providing borrowers with a clear roadmap of the loan application process and expected timelines can reduce anxiety and improve the overall experience.

1.2.2 Project Goals

Our platform is designed to streamline the credit application process, enhance communication between bank agents and clients, and ensure fair and transparent processing of credit requests. By eliminating unnecessary administrative hurdles and automating

repetitive tasks, clients can submit their applications quickly and hassle-free.

1.3 Study of The Existing

most banks, credit applications are generally submitted by customers directly at the branch, without the possibility of doing so online. This method results in a lack of transparency in the application processing. Furthermore, the evaluation of applications is primarily based on the guarantees provided by the applicants, which, due to the lack of available information, can often be subjective. This leads to ineffective communication between the bank agent and the customer.

There are several websites similar to our website, such as BIAT[2],BTS[1]...

The figure below represents the BTS[1] website.

Figure 1.1: BTS[1] websites.

The figure below represents the BIAT[2] website.

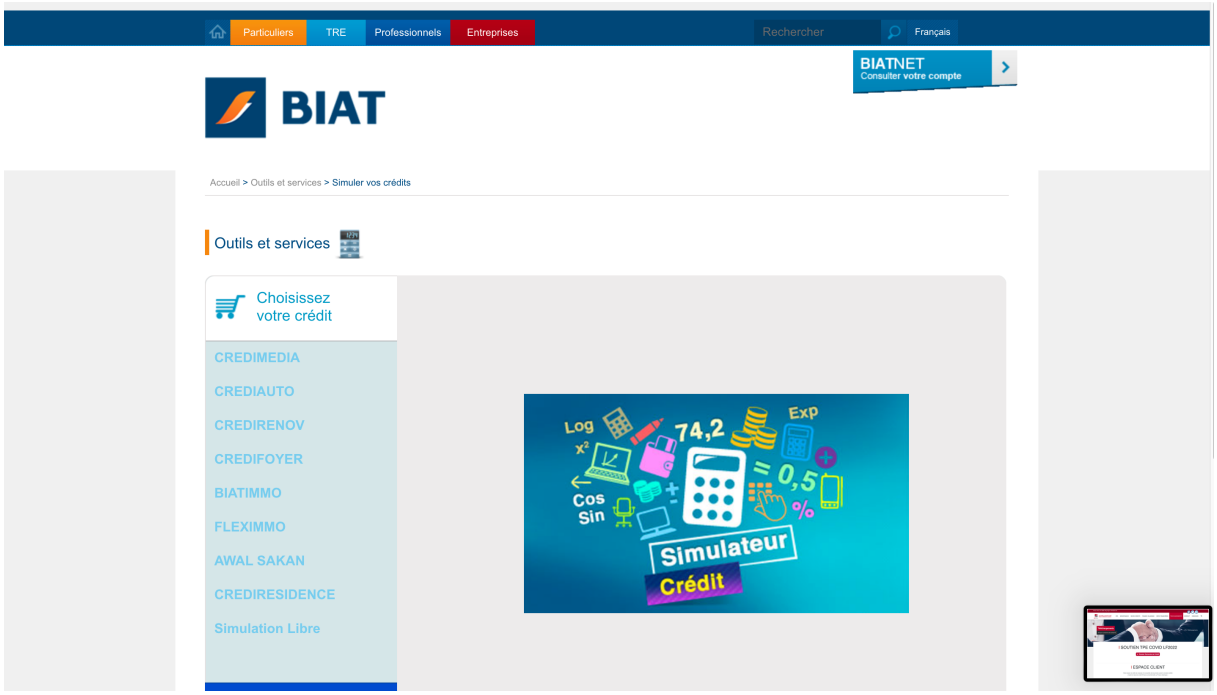

Figure 1.2: BIAT[2] websites.

This table below represents a comparison and the limitations of the two existing applications.

Criteria	**BIAT**	**BTS**
Design	Simple design, lacks modernity	Modern interface, but slow loading
Usability	Complicated navigation, not intuitive	Easy to use, but has some bugs
Functionality	there is not a loan request process in the website	there is not a loan request process in the website
Performance	High page loading time	Overall acceptable performance

Table 1.1: User Interface Comparison between Tayara and Cava

1.4 Proposed Solution

meet the needs of credit applicants and bank agents, we offer an innovative and comprehensive solution. This project involves the design of a website that encompasses the entire

credit application process and facilitates the business relationship between bank agents and credit applicants. Here are the main features of this solution:

Dynamic and Interactive Platform: Providing an online platform accessible to both bank agents and clients. This platform will enable smooth and real-time information exchange, thus facilitating the processing of applications and communication between stakeholders.

Simplified Credit Application Process: The platform will cover all stages of the credit application process, from submission to final approval. Users will be able to track the status of their application at every stage.

Objective Criteria for Credit Allocation: Credits will be allocated based on objective and transparent criteria, ensuring a fair and reliable assessment of credit applications.

Analysis and Reporting Tools: Bank agents will have access to analysis and reporting tools to evaluate credit applications efficiently. This includes customizable dashboards, statistics, and detailed reports.

Data Security: Ensuring the security and confidentiality of user data is a priority. The platform will be equipped with robust security measures to protect sensitive information.

User-Friendly Interface: The interface design will focus on user experience, making navigation intuitive and easy for both bank agents and credit applicants.

In conclusion, this solution aims to optimize the credit application process, improve communication between bank agents and clients, and ensure fair and transparent processing of credit applications.

1.5 Methodology of work

To achieve the project's objectives, it is important to follow an appropriate methodology. We have opted for the V-model methodology, a classic project management approach. Its importance lies in refining strategic visions, meeting client expectations, and adhering to deadlines. There are many traditional methods, and choosing the right one is not easy

because there is no single method for a project or industry. A methodology may work in one situation but not in another. In particular, our project relies on a very high level of predictability of the situation for certain reasons. In this case, the situation can be difficult as it cannot be accurately predicted.

1.5.1 V-Model Methodology

Rigorous Planning: Each project phase must be fully completed before moving on to the next. This allows for well-defined requirements and minimizes the risk of unforeseen changes during development.

Validation and Verification: For each development phase, there is a corresponding testing phase to verify and validate the product. For example, the design of functional specifications is followed by validation tests, and the design of technical specifications is followed by verification tests.

Detailed Documentation: The V-model requires exhaustive documentation at each project stage. This ensures traceability and facilitates the understanding and monitoring of the project by all stakeholders.

Clear Task Separation: Each team member has well-defined responsibilities, and tasks are clearly separated. This helps to avoid conflicts and ensures better organization of the work.

The main phases of the V-model are as follows:

- **Requirements Specification:** Define the project's functional and non-functional requirements in collaboration with the client.
- **Design:** Detail the technical specifications and develop the system design plans.
- **Implementation:** Code the software according to the specifications and design plans.
- **Unit Testing:** Test each code unit individually to ensure it works correctly.

- **Integration Testing:** Verify that the different units work correctly together.
- **Validation Testing:** Ensure that the final product meets the initially defined requirements.
- **Deployment:** Deliver and install the finished product to the client.
- **Maintenance:** Make corrections and improvements post-delivery.

In conclusion, the V-model is particularly suited to projects where requirements are well-defined from the start and where changes during development are limited. It ensures a high-quality final product through systematic testing and rigorous planning.

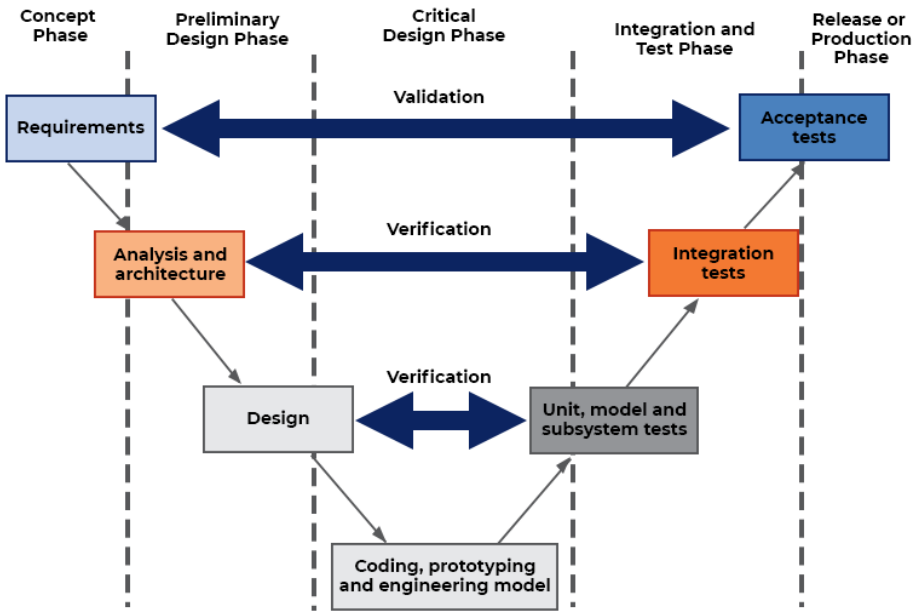

Figure 1.3: V-modal Process. [?]

1.5.2 Planification

A Gantt diagram, or Gantt chart, is a visual project management tool that illustrates a project schedule over time. The chart displays tasks or activities against a timeline, allowing project managers to see start and end dates, dependencies, and overall progress

at a glance. Each task is represented by a horizontal bar, with the length of the bar corresponding to the duration of the task. The Gantt chart helps in planning, coordinating, and tracking specific tasks within a project, ensuring that all parts of the project are on track and deadlines are met.

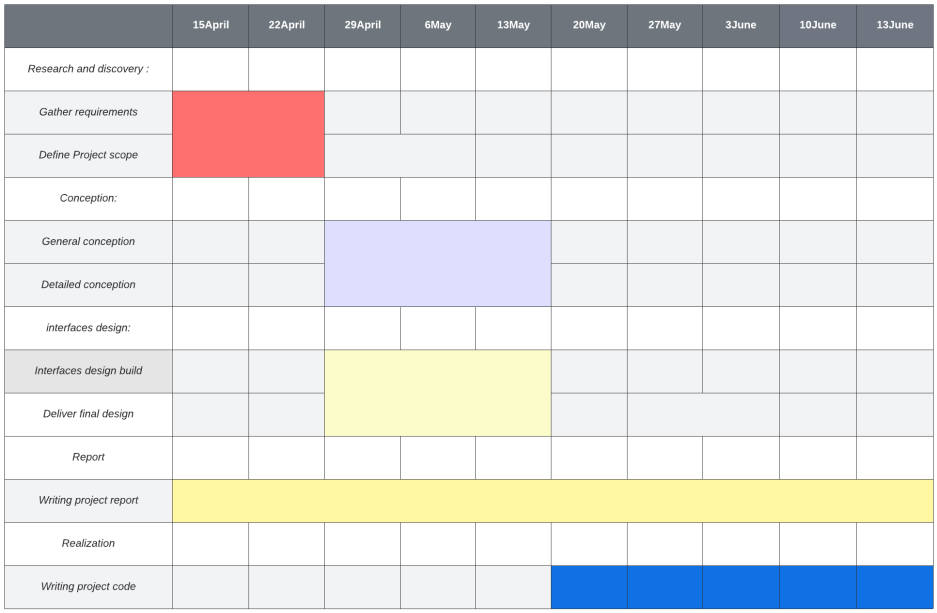

Figure 1.4: Gantt Diagram. [**?**]

1.6 Conclusion

In conclusion, the first chapter provides a comprehensive foundation for understanding the scope and direction of the project. By presenting the project, articulating the problem statement, critically analyzing existing applications, outlining the methodological approach, and detailing the task planning, this chapter sets the stage for a thorough and well-structured exploration of the project at hand. Each part contributes to building a solid framework that will guide the subsequent chapters and ensure a cohesive and informed progression towards achieving the project's objectives.

Chapter 2

chapter 2:Analytical and conceptual study

2.1 Introduction

After outlining the key points of the requirements specification phase, let's now focus on a fundamental phase in the software lifecycle. This phase will provide a clear understanding of the study domain by proceeding with a detailed design based on the UML modeling language.

2.2 Needs analysis

We specify in this section the functional and non-functional needs of the system.

2.2.1 Functional needs

The application should provide users with the essential functional requirements which are listed below:

General Requirement

- **Authenticate:** All users must authenticate to access their respective functions within the system.

Manager Role

- **Validate Account:** Managers have the capability to validate and activate user accounts.
- **View Statistics:** Access to a dashboard to view various statistics such as financial performance, account activities, and other relevant metrics.
- **Manage Governorate:** Ability to manage operations across regional branches or divisions.
- **Manage Subsidiary:** Oversee and manage subsidiary operations and performance.

Agent Role

- **Study Loan Request:** Review and process incoming loan applications.
- **Accept Loan Request:** Approve loan requests based on predefined criteria and verification.
- **Refuse Loan Request:** Deny loan requests with appropriate justification and documentation.
- **Update Profile:** Update personal and professional information within their profile.
- **Create Profile:** Agents can set up new user profiles, complete with necessary credentials and information.

Client Role

- **Create Loan Request:** Clients can submit requests for loans through an online interface.
- **View State of Requests:** Check the status of their submitted loan requests in real-time.
- **Create Quote Loan Request:** Clients can create a request specifically to generate a loan quote.
- **Print Quote:** Capability to print the loan quote directly from the system.

2.2.2 Non-functional needs

The application should respect the following non-functional requirements to ensure a good user experience.

- **Reliability:** the application must run correctly and without containing any bugs. And in case of disruption, the system should ensure its availability and recoverability.
- **Usability:**This application must present a clear interface and hierarchy of information to guarantee the understanding of the different processes.
- **Ease of use:** The use of the app should be simple and does not require any additional skills or knowledge..
- **Security:** The application should ensure the protection of the user's data and information.
- **Extensibility:** The application should be able to integrate new features without any major changes to its basic architecture.

2.3 Functional modeling

2.3.1 Identification of actors

we have three actors for our application:

Manager :he manage the Governorate and banking agency

bank agent :hen manage the loan Request

Client :he can choose whatever loan request he needs and have a quote

2.3.2 Modeling Language

UML[?]: The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system

Figure 2.1: UML Logo.

2.3.3 Global Use Case Diagram

Figure 2.4The global diagram shown below summarizes the basic use cases and provides an overall idea of how our system works.

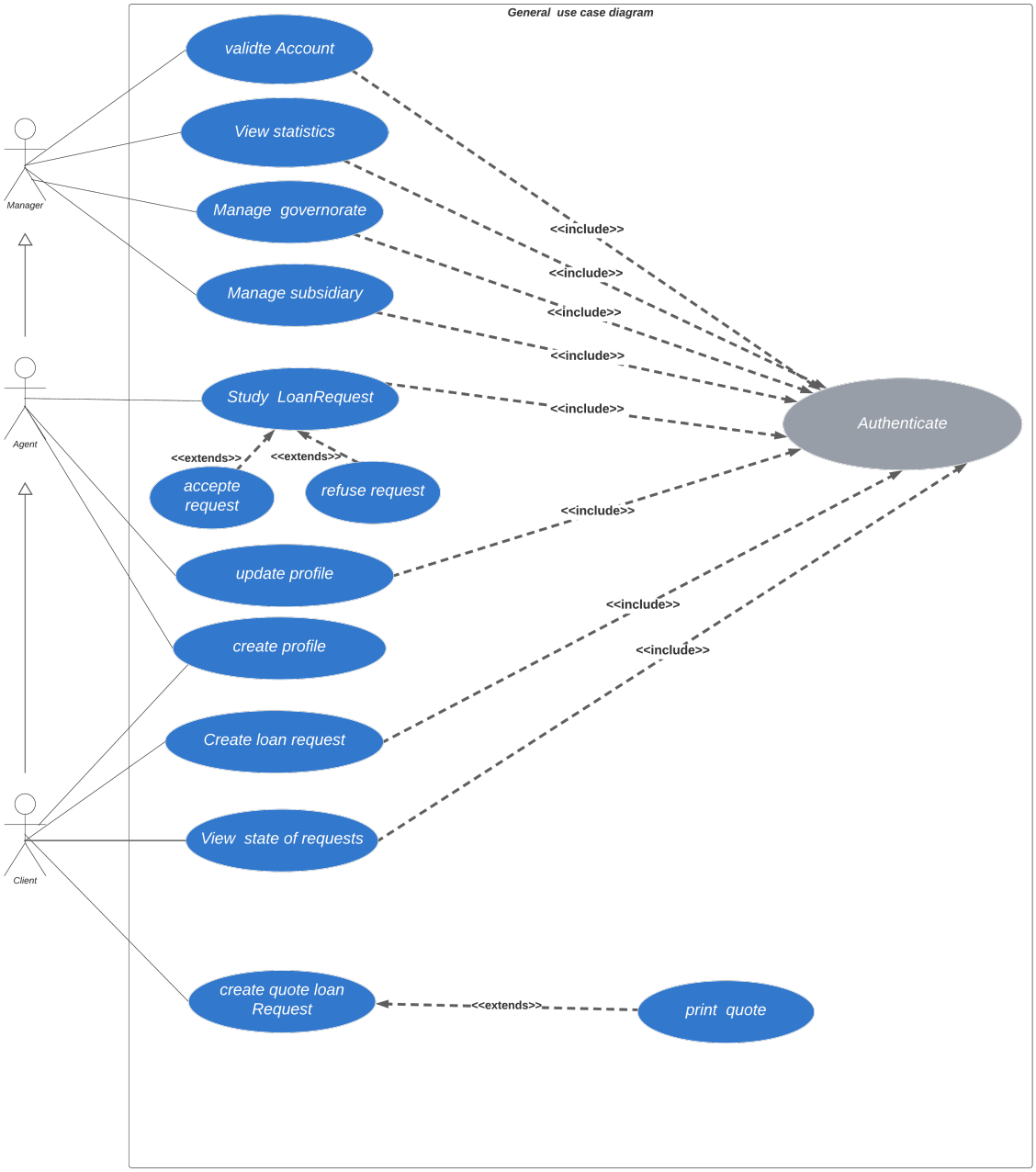

Figure 2.2: Global Use Case Diagram.

2.3.4 Refinement of use cases

This section provides a detailed description of the use cases representing the most important functionalities of our project.

Use Case "Authenticate"

This section details the "Authenticate" use case, pivotal for ensuring secure access to the system. Here, we describe the process by which users prove their identity to gain access to the application features.

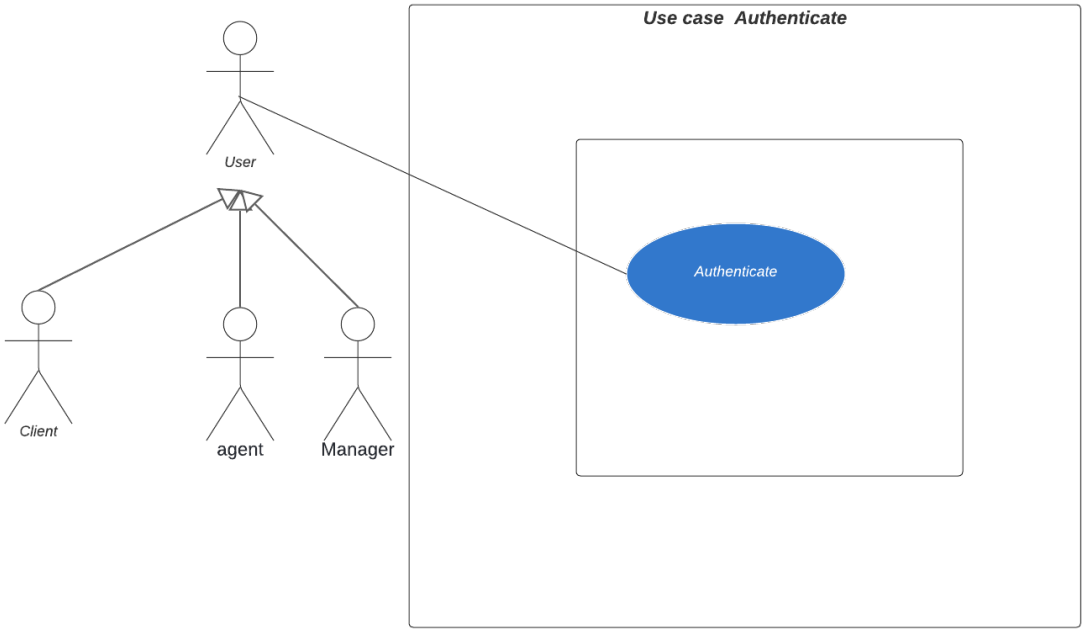

Figure 2.3: Authenticate Use Case Diagram.

- **Textual Description of "Authenticate" Use Case Diagram**

  Table **??** describes the textual description of the "Authenticate" use case

Title	Authenticate
Main Actor	client,Agent,Manager
Summary	The user must authenticate to benefit from more features of the application.

Pre-condition	The user is not authenticated..
Main Scenario	1. An authentication interface will be displayed for the user. 2. The user enters their login and password. 3. The application server checks whether the entered data is correct or not. 4. If positive, the client can access features according to their profile. 5. End.
Alternative Scenario	2.a The user has not filled in both fields. 1. The application will display an alert. 2. Return to step 2 of the nominal scenario to re-enter the login and password. 3.a The values entered by the user are incorrect. 1. The application displays an error.

	2. Return to step 3 of the nominal scenario to re-enter the login and password. -
Post-condition	Authenticated user. Display UI dashboard if the user is of type manager or agent, or the list of requests if the user is of type client.

Table 2.1: Textual Description of Authenticate

Use Case "Create Loan Request"

This section details the "Create Loan Request" use case, which is essential for clients wishing to apply for various types of loans through the application. Here, we outline the process by which clients can specify their loan requirements, choose appropriate loan terms, and submit all necessary documentation. This use case facilitates a structured and efficient loan application process, ensuring that clients can easily and effectively request financial assistance.

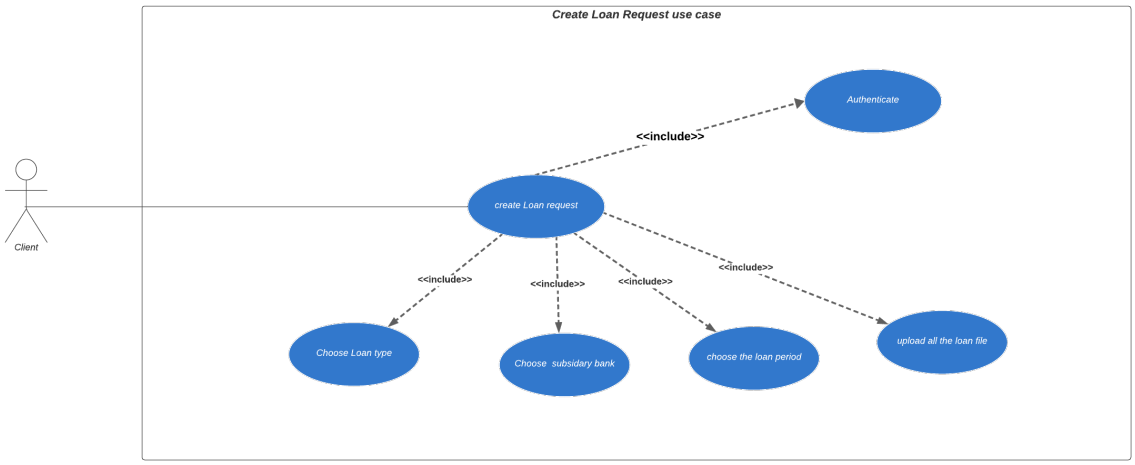

Figure 2.4: create Loan Request Use Case Diagram.

- **Textual Description of "Create Loan Request" Use Case Diagram**

  Table 2.2 describes the textual description of the "Create Loan Request" use case

Title	Create Loan Request
Main Actor	Client
Summary	The client initiates a loan request, chooses the loan type and subsidiary bank, specifies the loan period, and uploads necessary documentation.
Pre-condition	The client must be authenticated. The client must have access to loan application features.
Main Scenario	1. The client selects the option to create a new loan request. 2. The application presents different types of loans available for selection. 3. The client selects the desired loan type. 4. The client chooses a subsidiary bank from a list of available banks. 5. The client specifies the loan period. 6. The client uploads all required documents for the loan application.

	7. The application validates the entered and uploaded information. 8. If all data is correct and complete, the application submits the loan request for processing. 9. The client receives confirmation of the submitted request. 10. End.

Alternative Scenario	– At any step: If the client needs more information about any option, the application provides help or additional details. – Step 6: If the client fails to upload all required documents: 1. The application notifies the client of the missing documents. 2. The client is prompted to upload the necessary files again. – Step 8: If the data is incorrect or incomplete: 1. The application displays an error message. 2. The client is prompted to correct the data.
Post-condition	The loan request is successfully created and submitted for further processing. The client can track the status of the loan request through their dashboard.

Table 2.2: Textual Description of Create Loan Request

2.4 Classes Diagram

This section represents the Class Diagram

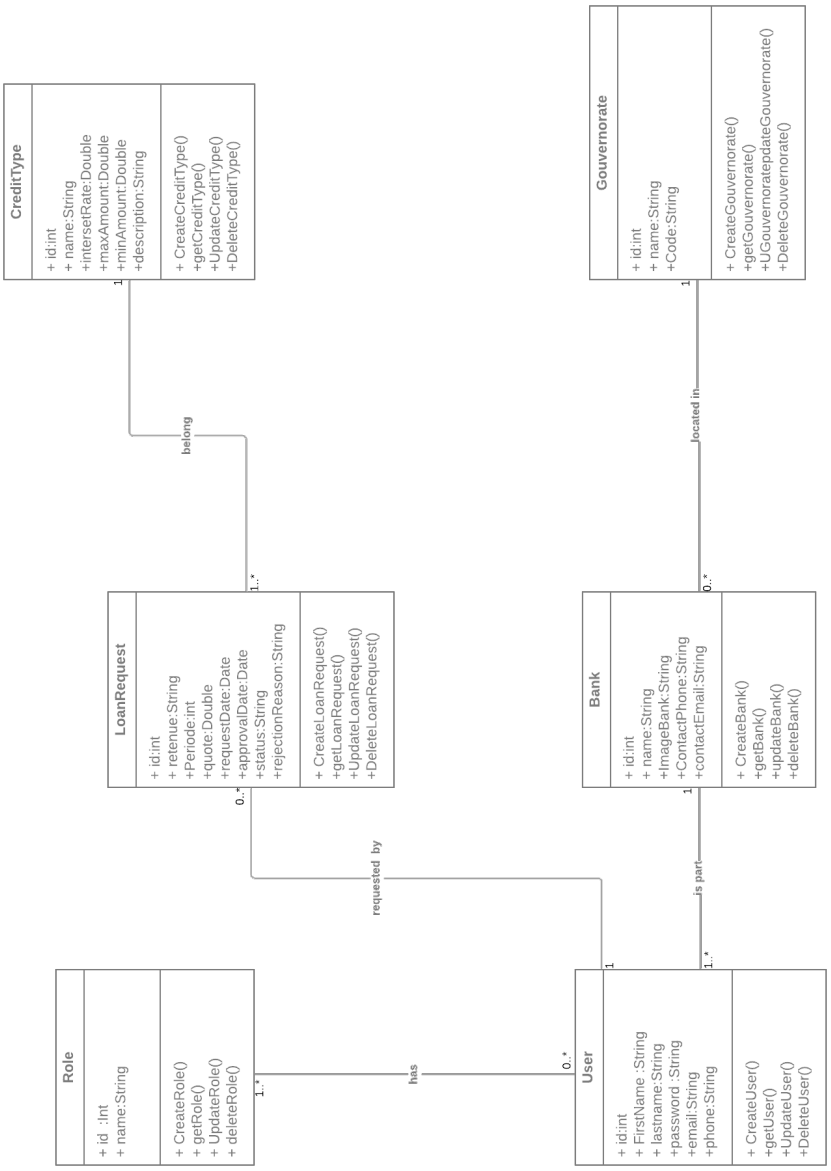

Figure 2.5: Class Diagram.

2.5 Sequence Diagram

2.5.1 Client Authenticate sequence Diagram

This section represents the sequence diagram for authenticating the Client.

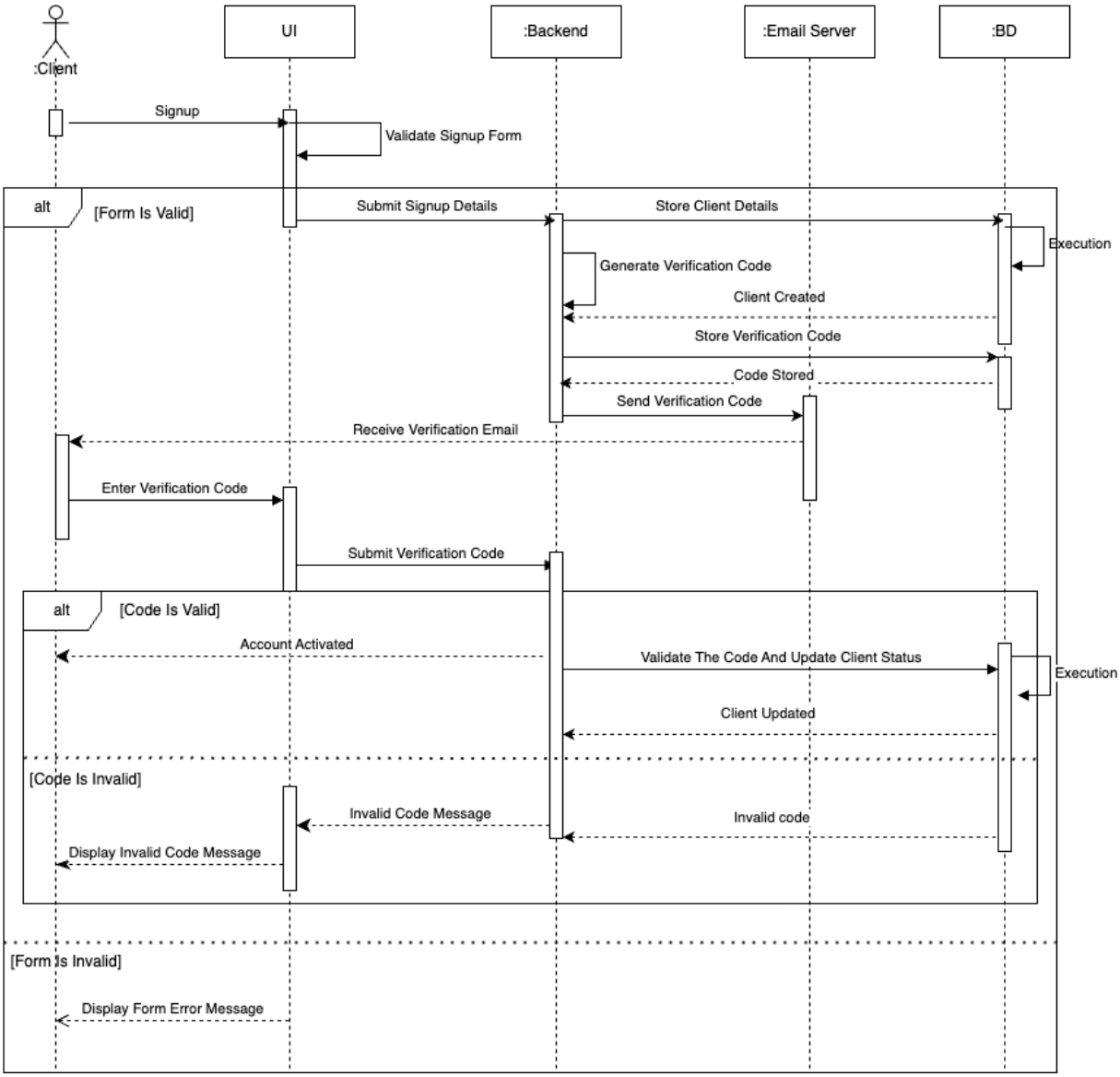

Figure 2.6: Client Authenticate sequence Diagram

2.5.2 agent Authenticate sequence Diagram

This section represents the sequence diagram for authenticating the Agent.

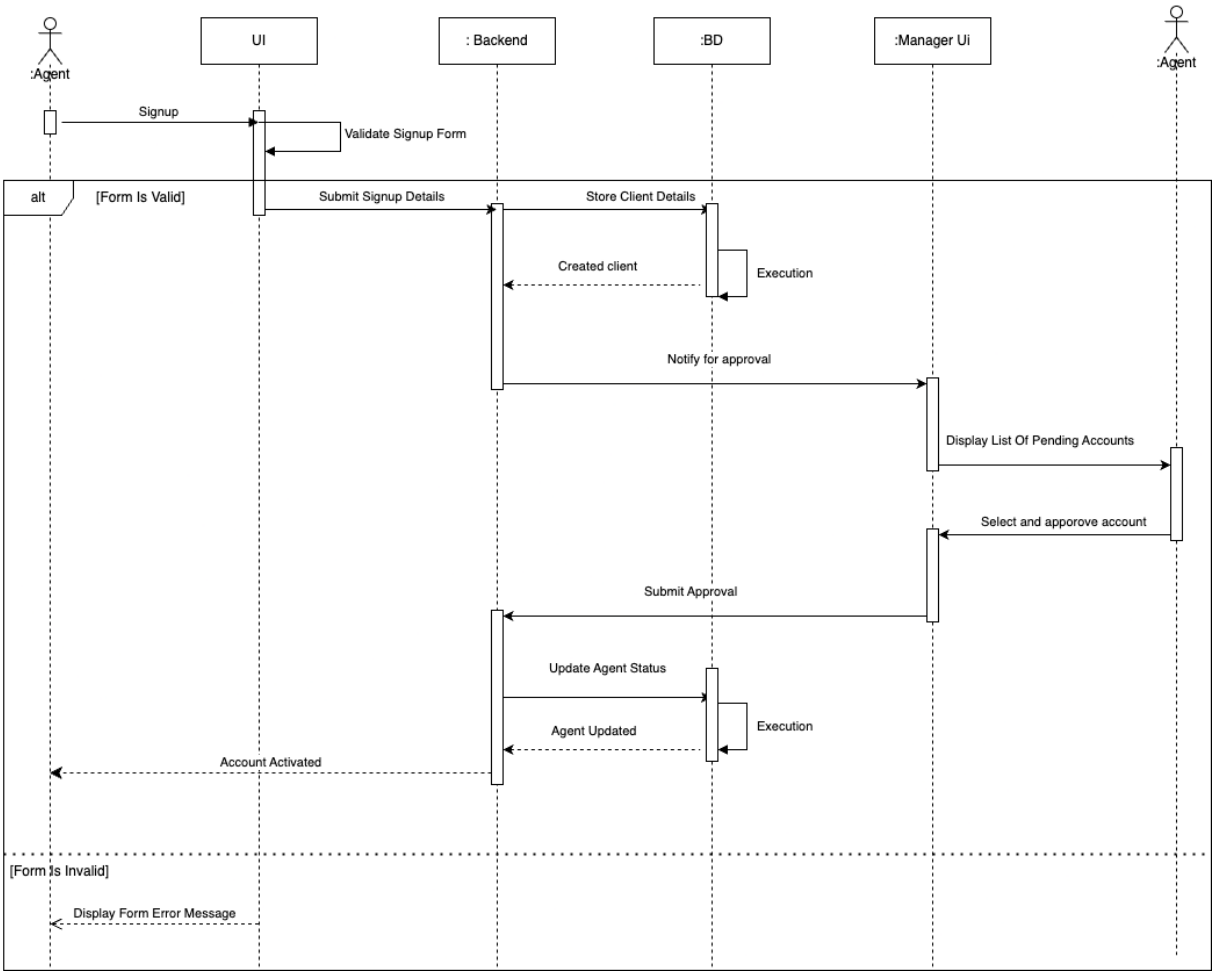

Figure 2.7: agent Authenticate sequence Diagram

2.6 state Diagram

This section represents the state diagram that describes the lifecycle of a loan request.

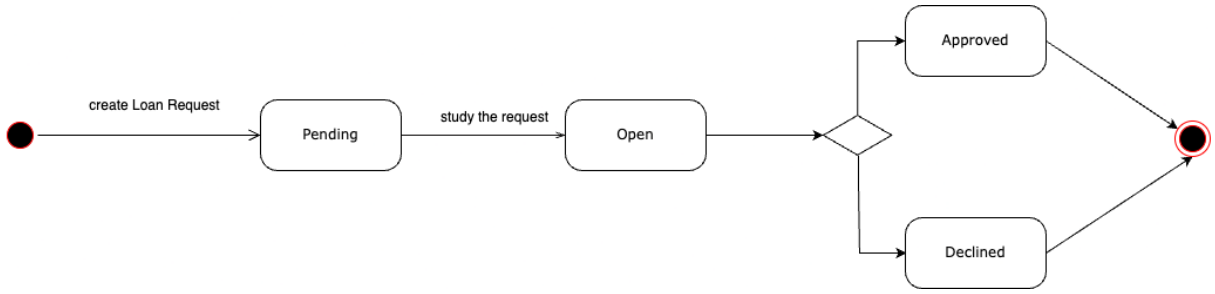

Figure 2.8: state Diagram

2.7 Conclusion

In this chapter, we presented the diagrams that describe the features of my application, leading to implementation with a clear vision of both functional and organizational aspects.

In the following chapter, titled "Implementation," I will execute what has been described during the Design phase.

Chapter 3

chapter 3:Implementation

3.1 Introduction

After completing the specification and design phase, with the solution already chosen and studied, it remains to decide in which environment we worked. Therefore, I dedicated this chapter to presenting the environment used.

Consequently, we exposed the technical choices made and the programming languages adopted, and finally presented the implementation and tests carried out. This chapter is divided into four main parts. Firstly, we began with a description of the software environment and the hardware environment. Secondly, we present the architecture of the proposed solution. Thirdly, we examine the deployment diagrams. Fourthly, we provide a description of the interface that I implemented. Finally, we discuss the continuous integration process for the application, enumerate the difficulties encountered during the realization of this project, and provide a conclusion.

3.2 Environment and Technical Choices

3.2.1 Hardware Environment

To achieve this project, we've used a machine with the following characteristics:

Brand	MacBook Pro
Processor	m1 pro
RAM	8.00 GB
Operating System	Macos
Storage	256
Screen	13 inches

Table 3.1: Computer's Characteristics

3.2.2 Software Environment

In this section, we will define the list of the software environment used throughout the project.

Diagrams.net[3]: is free online diagram software for making flowcharts, process diagrams, org charts, UML, ER, and network diagrams.

I used this software to make all conception diagrams.

Figure 3.1: Diagrams.net Logo.

Overleaf[4] and LaTeX[5]: Overleaf is an online LaTeX editor that was used for writing this graduation report. LaTeX is a high-quality typesetting system, it is the de facto standard for the communication and publication of scientific documents.

This software helped me writing a well organized report.

Figure 3.2: LateX Logo.

Visual Studio Code[6]: is a lightweight code editor developed by Microsoft, this editor was used to develop both the web application frontend and backend for its efficiency, simplicity, and lightweight.

Using VS code solve me many problems and saved me a lot of time because of his various tools and features.

Figure 3.3: Visual Studio Code Logo.

IntelliJ IDEA[7]: is a comprehensive IDE developed by JetBrains, used for developing both the backend of web applications. Its efficiency, robust feature set, and user-friendly interface make it an excellent choice.

Using IntelliJ IDEA solved many problems for me and saved a lot of time due to its extensive tools and features.

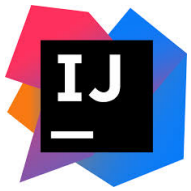

Figure 3.4: IntelliJ IDEA Logo.

Git[8] and Github[9]: Git is a version control system for managing and keeping track of source code history. GitHub is a cloud-based hosting service for managing and

hosting Git repositories.

Git and Github helped me to organize and synchronize the work with deepeye team.

Figure 3.5: Git and GitHub Logos.

Postman[10]: is a free cross-platform desktop application that takes the pain out of interacting with and designing HTTP-based APIs.

I used Postman to test the functionality of my Spring boot REST API.

Figure 3.6: Postman Logo.

Swagger[11]: Swagger is an open source set of rules, specifications and tools for developing and describing RESTful APIs. The Swagger framework allows developers to create interactive, machine and human-readable API documentation.

I used swegger.ui to create a well documented API.

Figure 3.7: Swagger Logo.

Chrome Developer Tools[12]: is a set of web developer tools built into chrome.

You can use them to examine, edit, and debug HTML, CSS, JavaScript and Angular. Chrome Dev Tool is the main tool of testing and debugging front-end functionality.

Figure 3.8: Chrome Logo.

Docker[13]: Docker is an open source software platform to create,deploy and manage virtualized application containers on a common operating system(OS),with an ecosystem of allied tools.

Docker used to reduce the conflict of versions and to set up a unique environment for all development team.

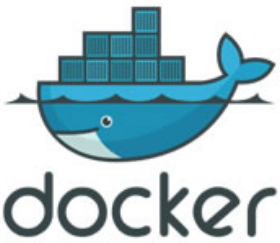

Figure 3.9: Docker Logo.

Jenkins[14]: Jenkins is an open source automation server used to automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery.

We used Jenkins for continuous integration to automate our build and deployment process.

Figure 3.10: Jenkins Logo.

3.2.3 Technical Choices

In this section, we present the technologies used for building this project.

3.2.4 Technical Choices

In this section, we present the technologies used for building this project.

PostgreSQL[15]: PostgreSQL is a powerful, open source object-relational database system. It uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads.

We used PostgreSQL to store and manage the data for our application, leveraging its robust transactional support and extensive indexing capabilities for fast data retrieval.

Figure 3.11: PostgreSQL Logo.

Java[16]: Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.

We used Java to integrate the model and to create back-end functionality.

Figure 3.12: Java Logo.

Spring Boot[17]: Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications.
Spring Boot is used to develop a REST API.

Figure 3.13: Spring Boot Logo.

React[18]: React is a JavaScript library for building user interfaces, maintained by Facebook and a community of individual developers and companies. React allows developers to create large web applications that can change data, without reloading the page. Its key feature is the ability to build components in JSX, a syntax extension that resembles HTML.

JSX[19]: JSX stands for JavaScript XML. It allows us to write HTML in React. JSX makes it easier to write and add HTML in React.

Figure 3.14: JSX Logo.

React[18]: React is a free and open-source front-end JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of

individual developers and companies. React can be used as a base in the development of single-page or mobile applications.

I used React to develop a single-page application web interface and to manage components.

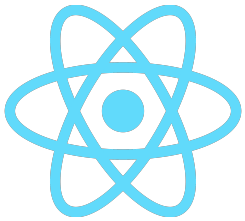

Figure 3.15: React Logo.

Material-UI[20]: Provides React components that implement Google's Material Design.

I used the Material-UI pre-styled React components.

Figure 3.16: Material-UI Logo.

Python[21]: Python is a high-level, interpreted, general-purpose programming language. It is widely used for web development, data analysis, artificial intelligence, and scientific computing.

We used Python along with the Django framework to handle the backend processing of loan requests.

Figure 3.17: Python Logo.

Django[22]: Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It is particularly good at making database-driven websites due to its built-in features.

Django was used to develop the loan request application, providing robust tools to handle web requests and manage data securely and efficiently.

Figure 3.18: Django Logo.

NumPy[23]: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

We utilized NumPy to perform complex calculations required for determining loan eligibility and repayment schedules.

Figure 3.19: NumPy Logo.

Next.js[24]: Next.js is an open-source React front-end development web framework that enables functionality such as server-side rendering and generating static websites for React based web applications. It enhances React applications with better performance and SEO capabilities.

We chose Next.js for the client-side of our application to leverage its server-side rendering features, which improve the SEO performance and the initial load time of our web pages.

Figure 3.20: Next.js Logo.

Advantages of Next.js over React for the Client-Side:

- **SEO Optimization:** Next.js provides server-side rendering by default, which allows search engines to crawl and index the pages more effectively than client-side rendered applications. This is particularly beneficial for content-driven applications where SEO is crucial.
- **Performance Enhancements:** Next.js optimizes the delivery of your web application by automatically splitting the code into smaller bundles and only loading what's necessary for the initial page render. This improves the speed and responsiveness of the application.

3.3 Architecture of the Proposed Solution

The system architecture is structured as an n-tier architecture with distinct layers for the frontend, backend, and the database. Each layer has specific roles and responsibilities, facilitating modular development and maintenance.

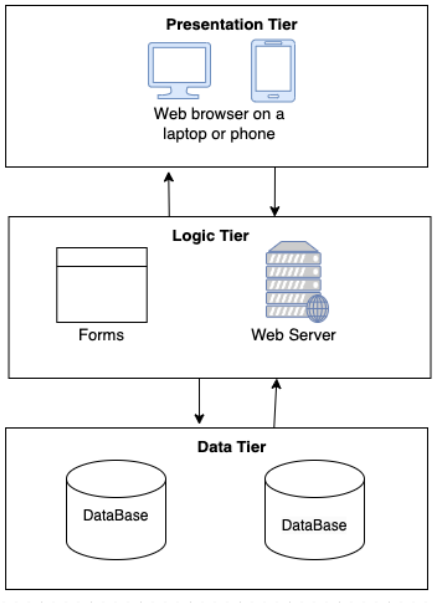

Figure 3.21: n-tier Architecture.

3.3.1 Application Physical Architecture

In our application, the communication between the layers involves HTTP requests and SQL queries. The frontend, consisting of a React dashboard and a Next.js frontend, issues HTTP requests to the backend. These requests are managed by the Spring Boot or Django backend systems.

The Spring Boot backend processes these requests and interacts with the database through SQL queries to fetch or modify data. The SQL responses are received by the Spring Boot backend, which then formats this data into JSON and sends it back to the frontend. This JSON data is subsequently rendered in the browser, showcasing the required information on the React dashboard or Next.js frontend.

Similarly, the Django backend processes HTTP requests from the frontend, engages with the database via SQL queries, and returns JSON responses to the frontend. This architecture supports a modular and scalable application, allowing different backend systems to handle various data processing and management tasks, seamlessly integrating to provide a unified user experience on the frontend.

Figure 3.22 represents the application's architecture.

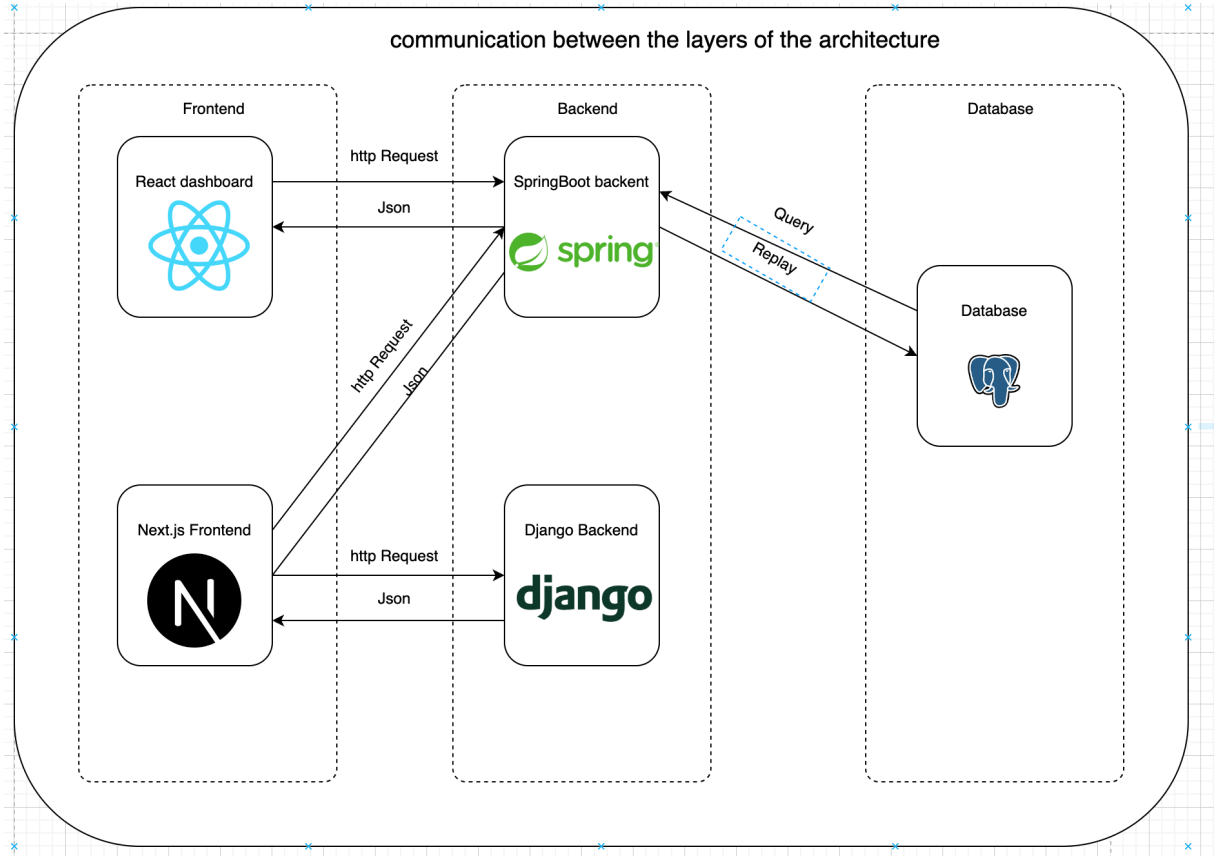

Figure 3.22: Application Physical Architecture.

3.3.2 Design Patterns Applied

MVVM

Allows frontend developers to manage and update the view's logic and behavior without needing to understand the entire model. This is particularly beneficial in complex interfaces where the view behavior can change dynamically based on different states of the application.

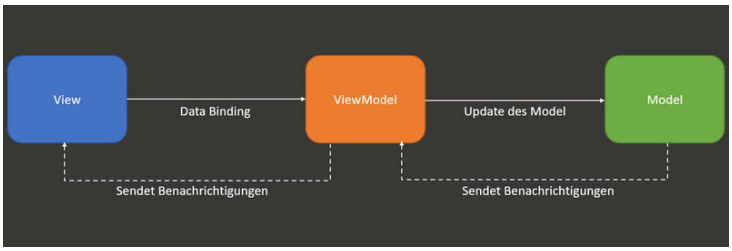

Figure 3.23: mvvm pattern.

Model-View-Controller (MVC)

used within the backends , separating the application into models, views, and controllers, which respectively manage the data, user interface, and application logic.

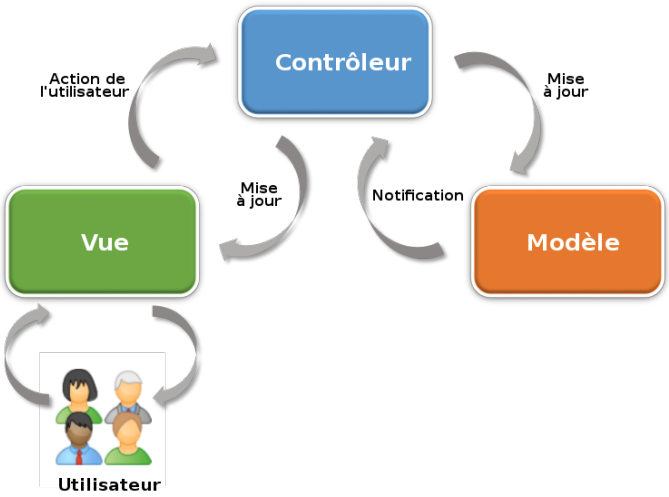

Figure 3.24: mvc pattern.

Benefits of Design Patterns

Maintainability: Each pattern supports maintainability through the separation of concerns:

- **MVVM:** Allows frontend developers to manage and update the view's logic and behavior without needing to understand the entire model. This is particularly beneficial in complex interfaces where the view behavior can change dynamically based on different states of the application.

- **MVC:** Facilitates easy updates and maintenance of the backend logic, as each component (model, view, controller) is developed and maintained independently.

Scalability: The client-server structure allows independent scaling of frontend and backend services. MVVM supports rich interactive and responsive interfaces that can efficiently handle user interactions even on large-scale deployments.

Testability: MVVM enhances testability of the frontend components by allowing developers to write unit tests for the ViewModel without involving the views. MVC supports testability on the backend by allowing separate testing of business logic without interaction with the database or the user interface.

3.4 Deployment Diagrams

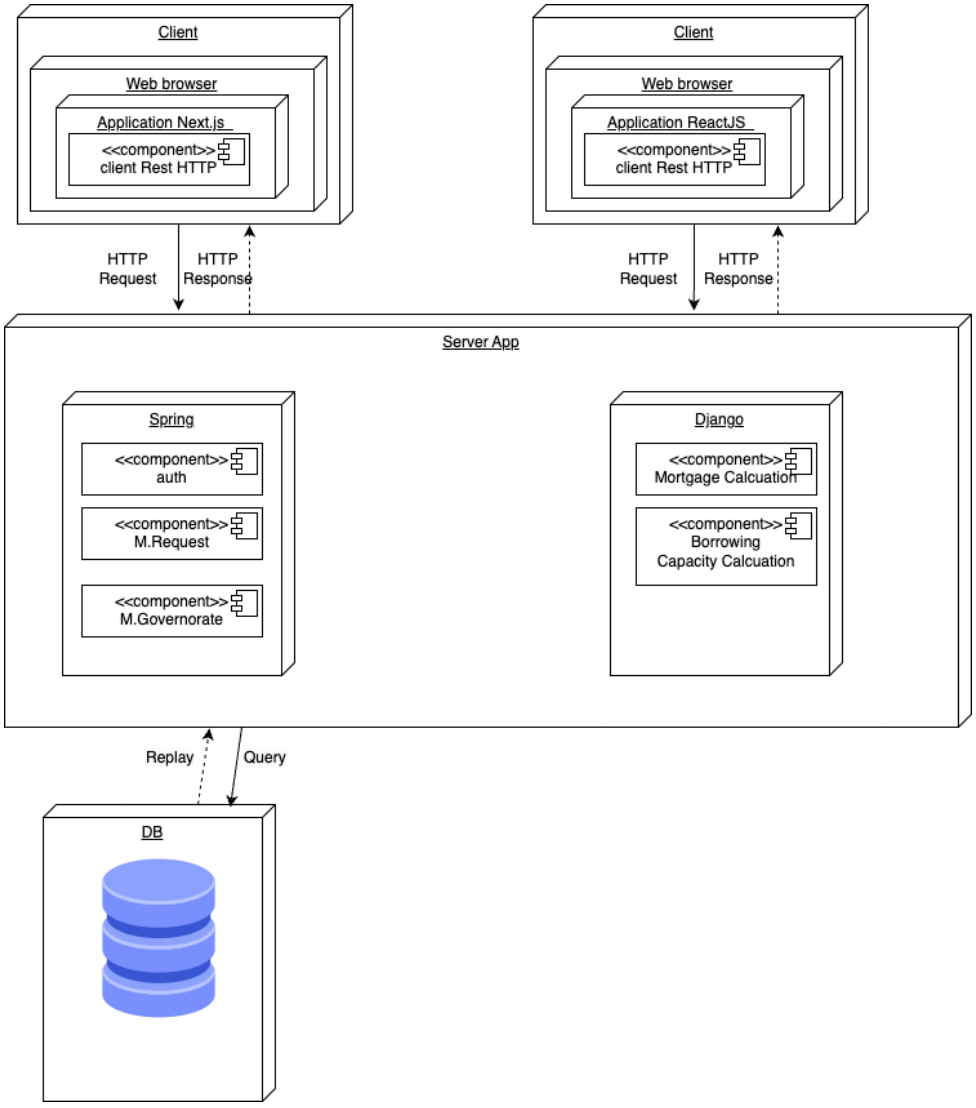

Figure 3.25: Deployment Diagrams.

3.5 Realization

This section is devoted to the exposition of the completed work through screenshots.

3.5.1 Manager Interface

Authentication Interface for Agent and Manager:

The figure below represents the login interface for the Agent and Admin .

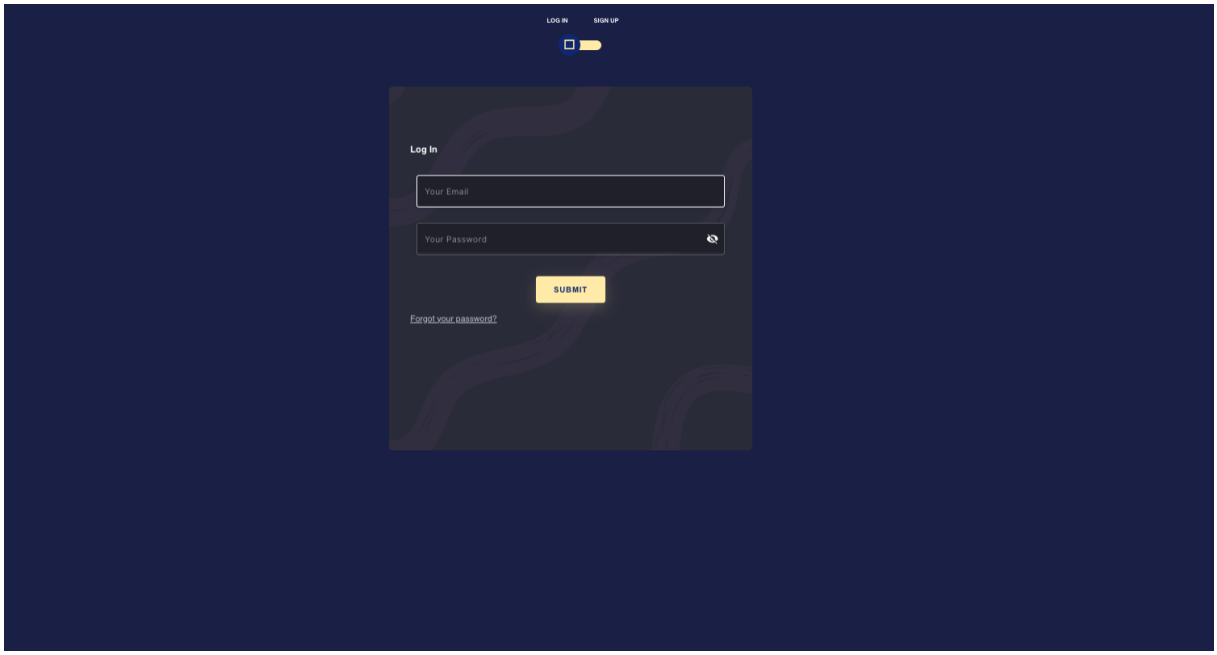

Figure 3.26: Authentication agent and manager Interface

Agent registration interface

The figure below represents the registration interface where the agent can create their account.

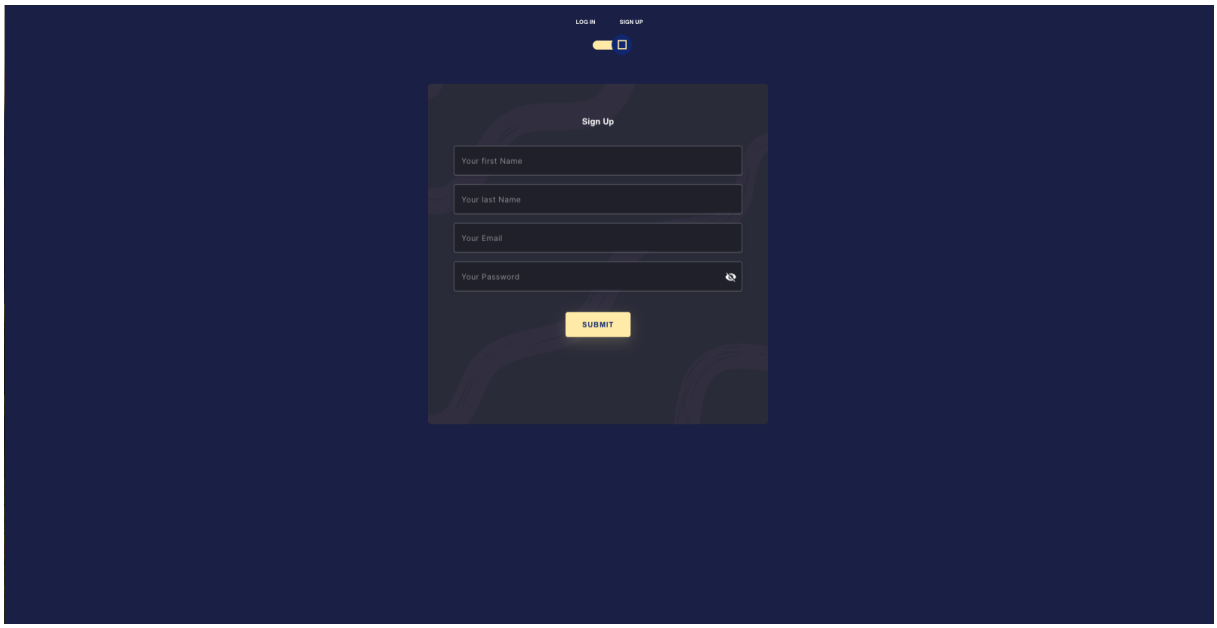

Figure 3.27: Agent registration interface

dashboard interface in dark mode:

The figure below represents the dashboard, displayed in dark mode, where the manager can view the number of users, agents, and bank agencies associated with the website. It also includes a table detailing user information, a pie chart displaying the number of users for each role, and a chart showing the number of loan requests per month.

Figure 3.28: dashboard interface dark

dashboard interface in light mode:

The figure below represents the dashboard, displayed in light mode, where the manager can view the number of users, agents, and bank agencies associated with the website. It also includes a table detailing user information, a pie chart displaying the number of users for each role, and a chart showing the number of loan requests per month.

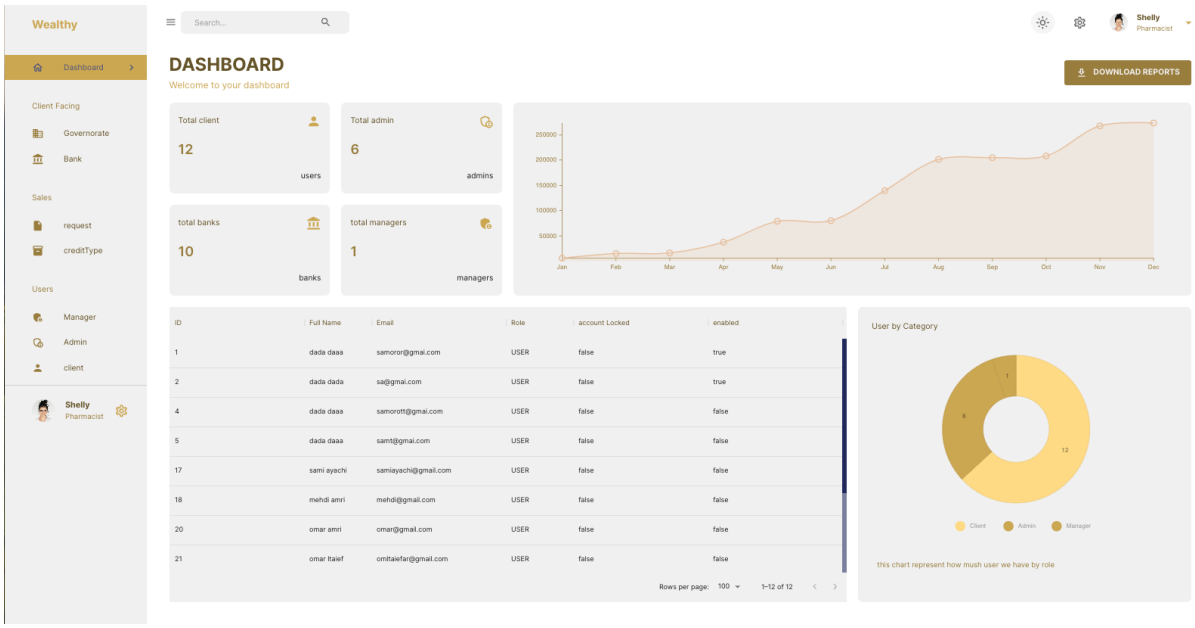

Figure 3.29: dashboard interface

Governorate interface:

The figure below represents the Governorate list where the manager can add, update, or delete Governorates and export the list as an Excel file.

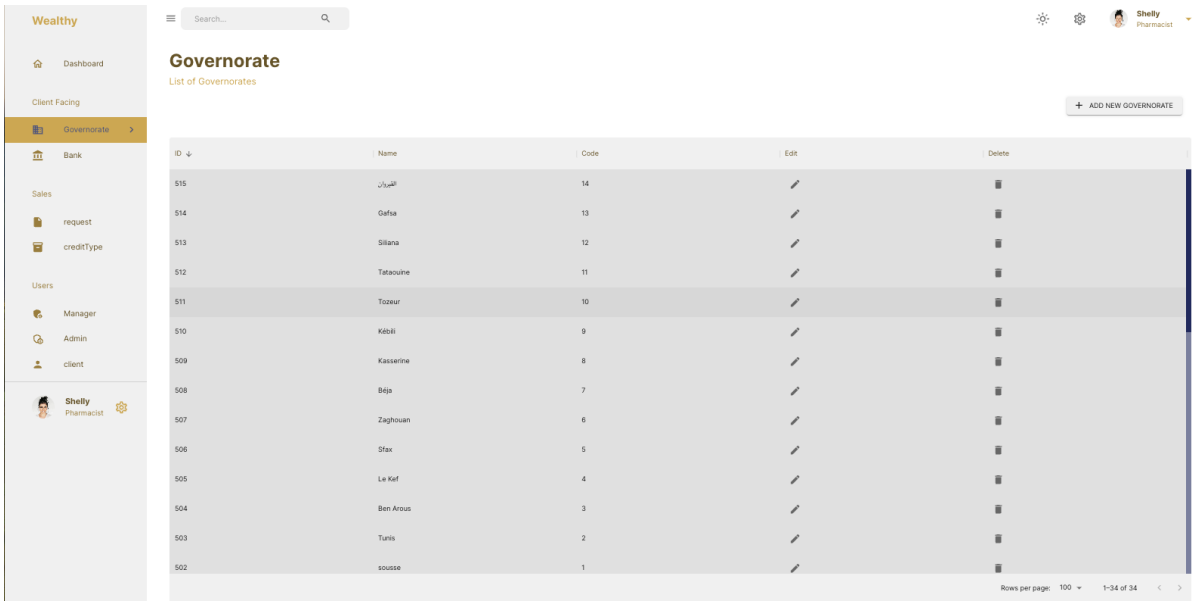

Figure 3.30: Governorate interface

banking agency interface:

The figure below represents the banking agency list where the manager can add, update, or delete bank agencies and export the list as an Excel file.

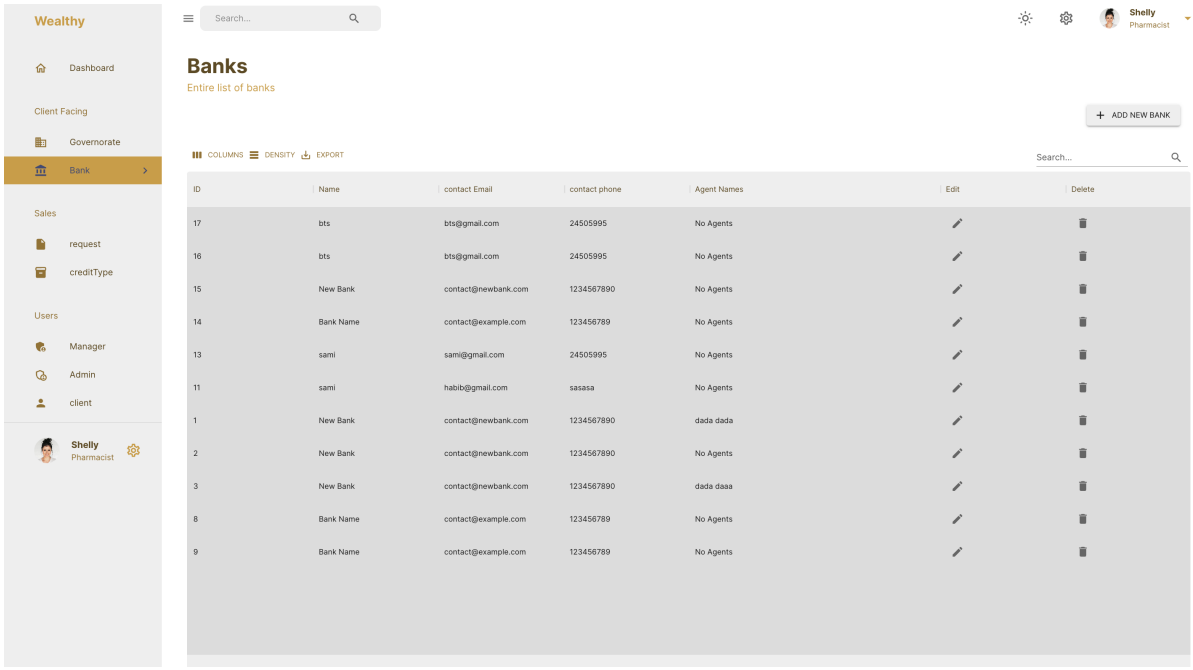

Figure 3.31: banking agency interface

add banking agency interface:

The figure below illustrates how the manager can add a banking agency.

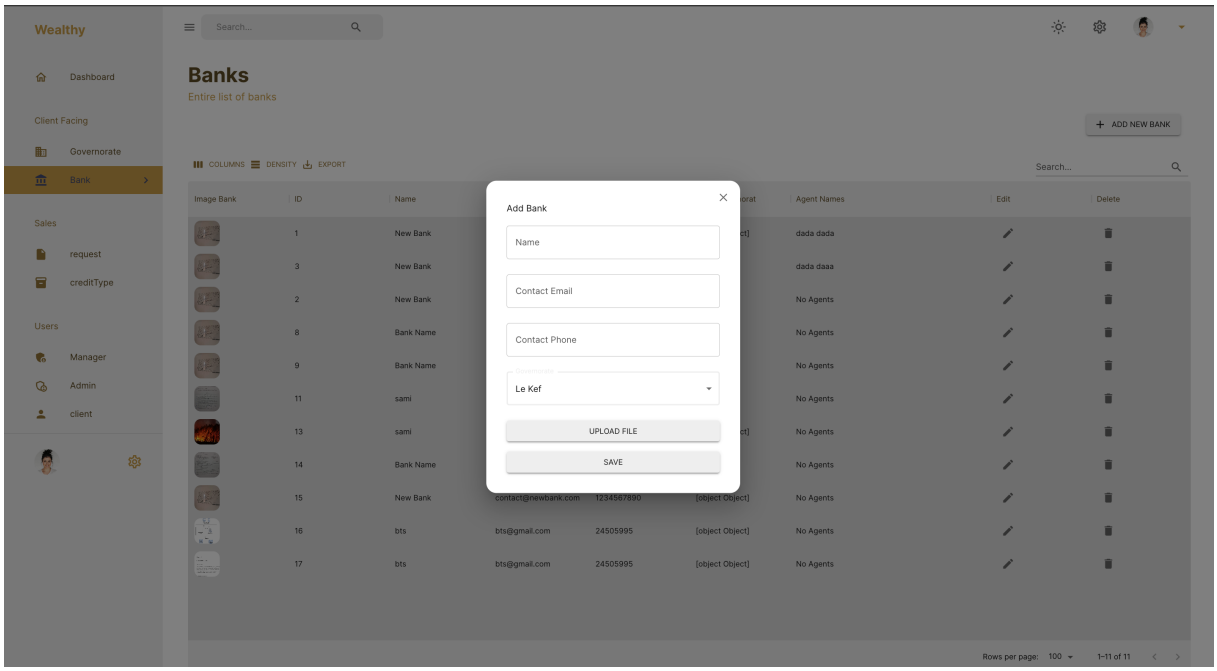

Figure 3.32: add banking agency interface

update banking agency interface:

The figure below illustrates how the manager can update a banking agency.

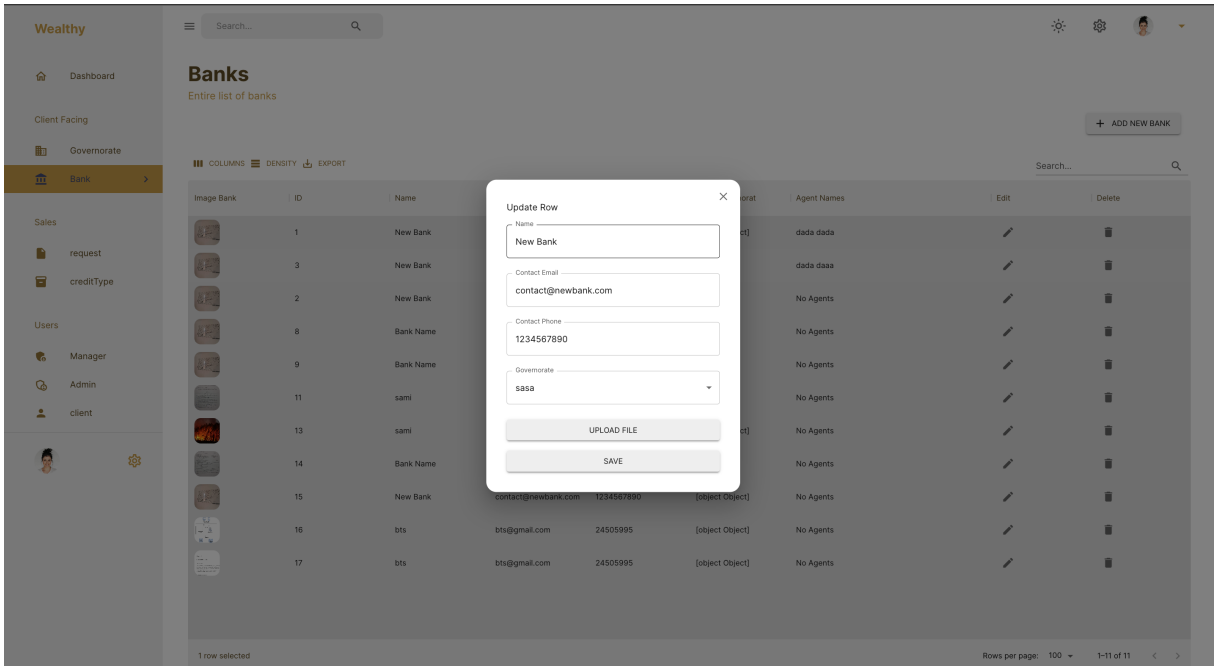

Figure 3.33: update banking agency interface

delete banking agency interface:

The figure below illustrates how the manager can delete a banking agency.

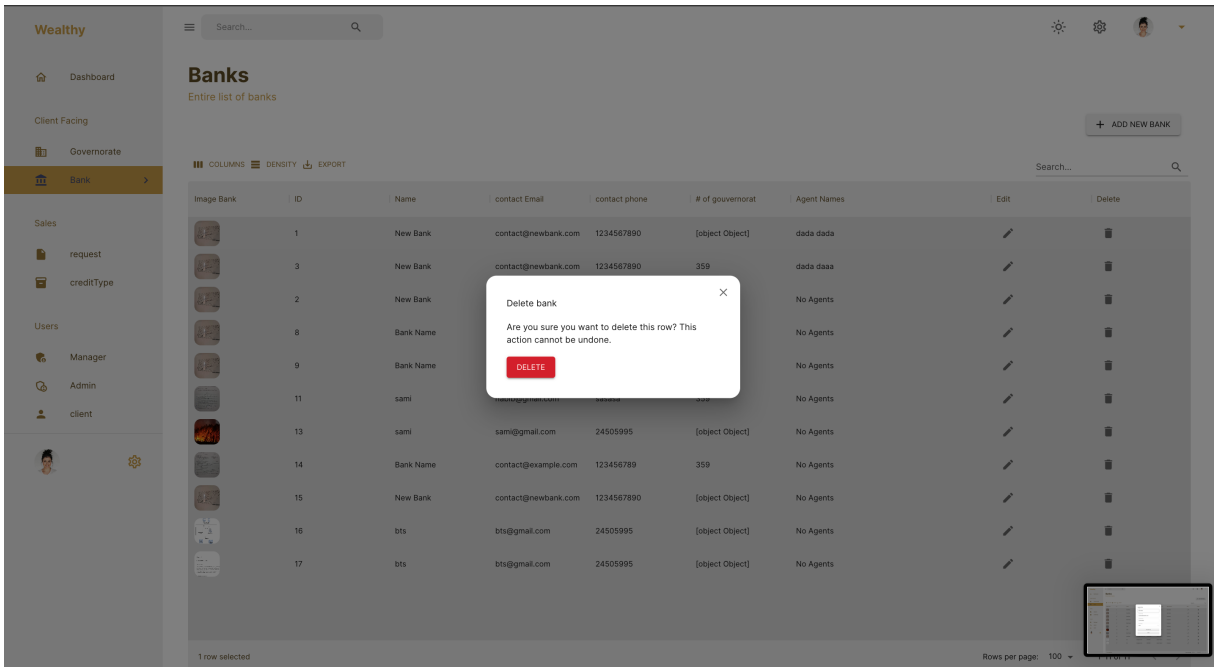

Figure 3.34: update banking agency interface

loan Type interface

The figure below represents the loan type list where the manager can add, update, or delete bank agencies and export the list as an Excel file.

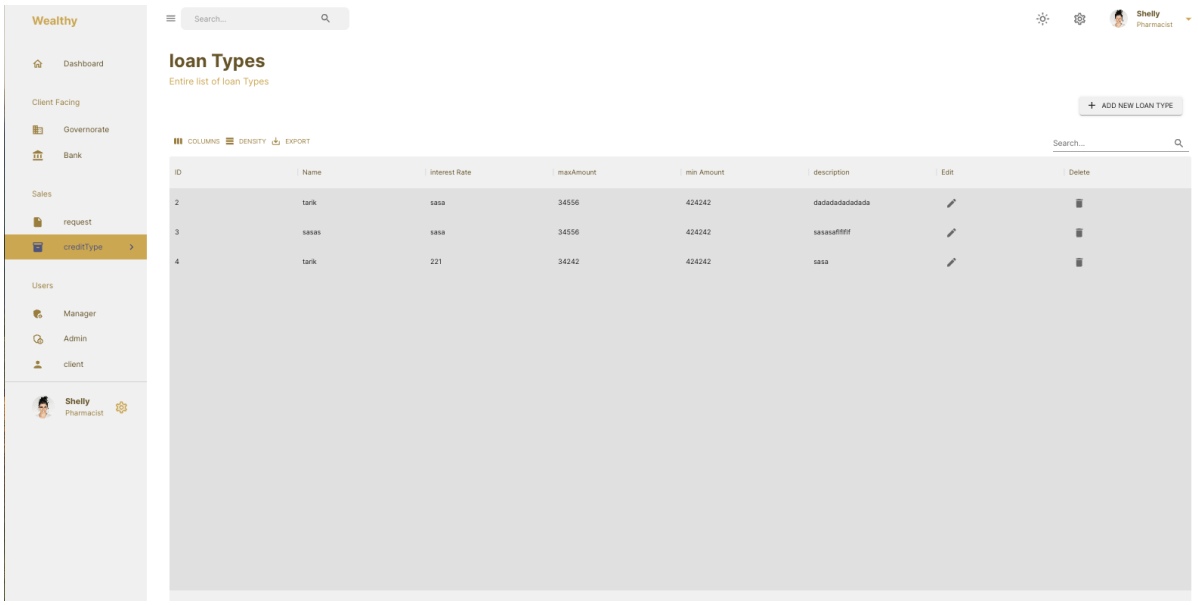

Figure 3.35: loan Type interface

loan request interface

The figure below represents the loan request list, where the agent can approve or refuse a loan request.

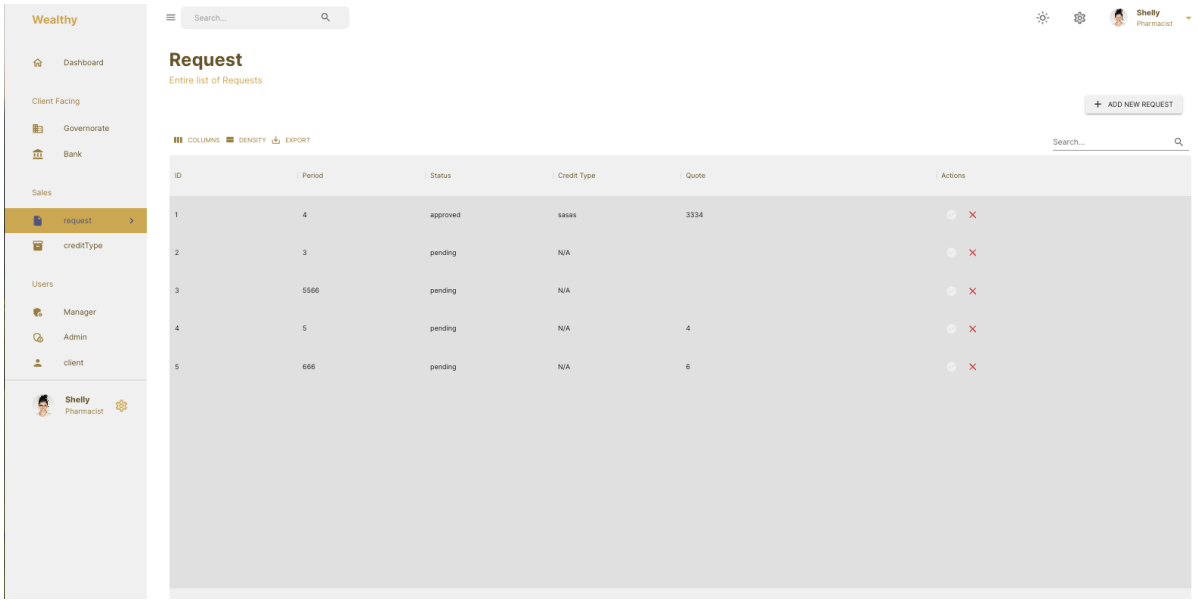

Figure 3.36: loan request interface

Agent interface

The figure below represents the loan agent list, where the manager can approve or blocked the agent account.

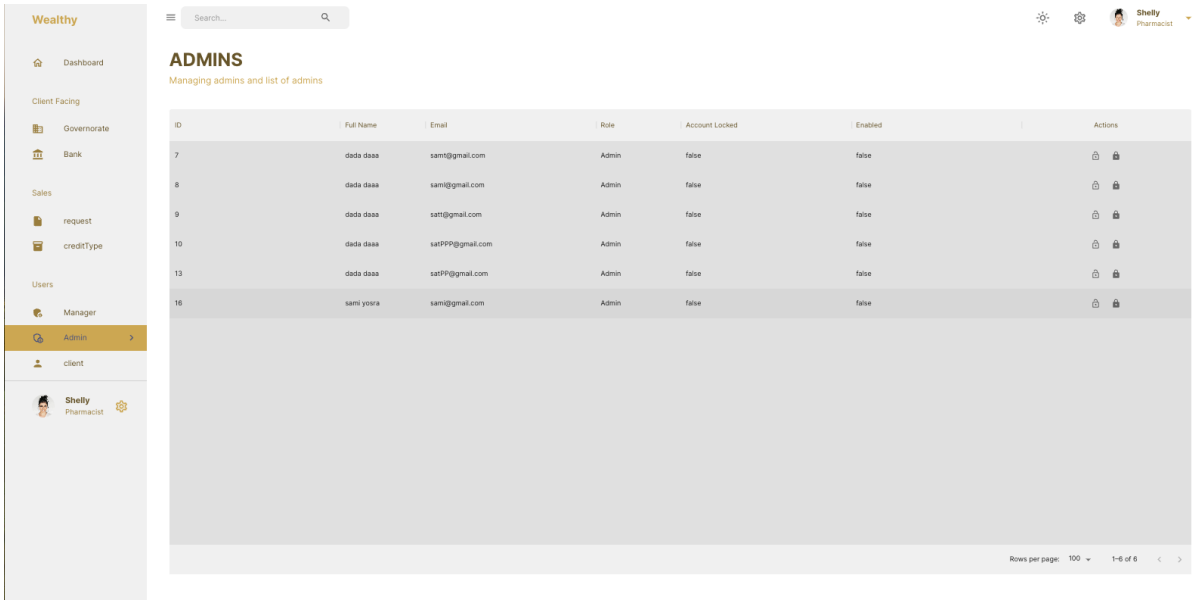

Figure 3.37: Agent interface

client interface

The figure below represents the loan agent list, where the manager can approve or blocked the Client account.

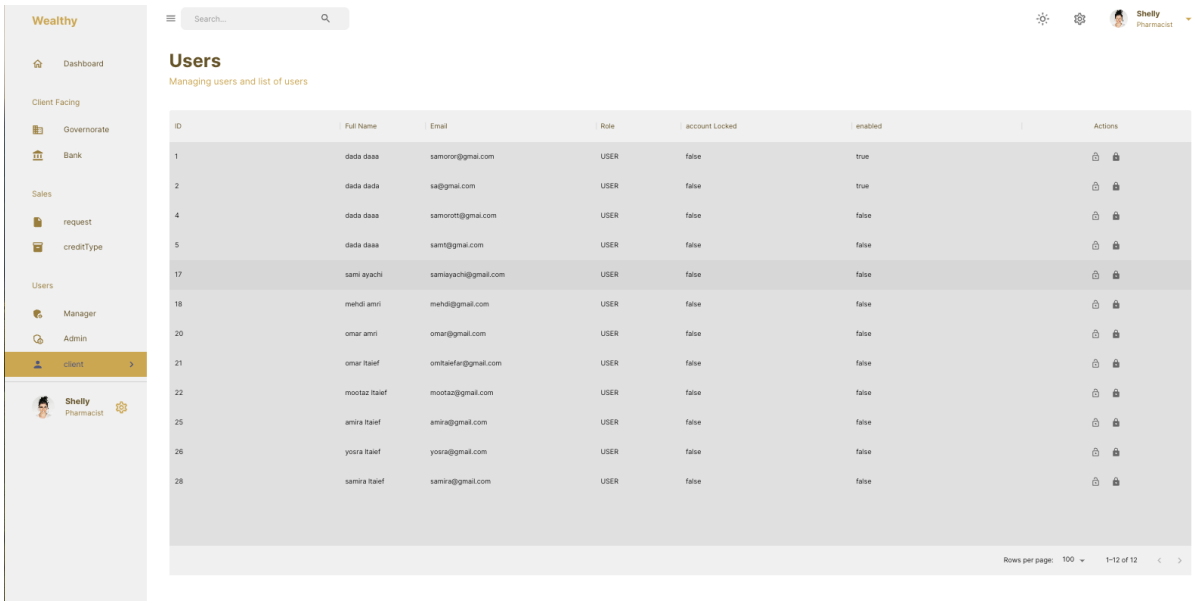

Figure 3.38: client interface

3.5.2 Client Interface

Client registration interface:

The figure below represents the registration interface for the Client .

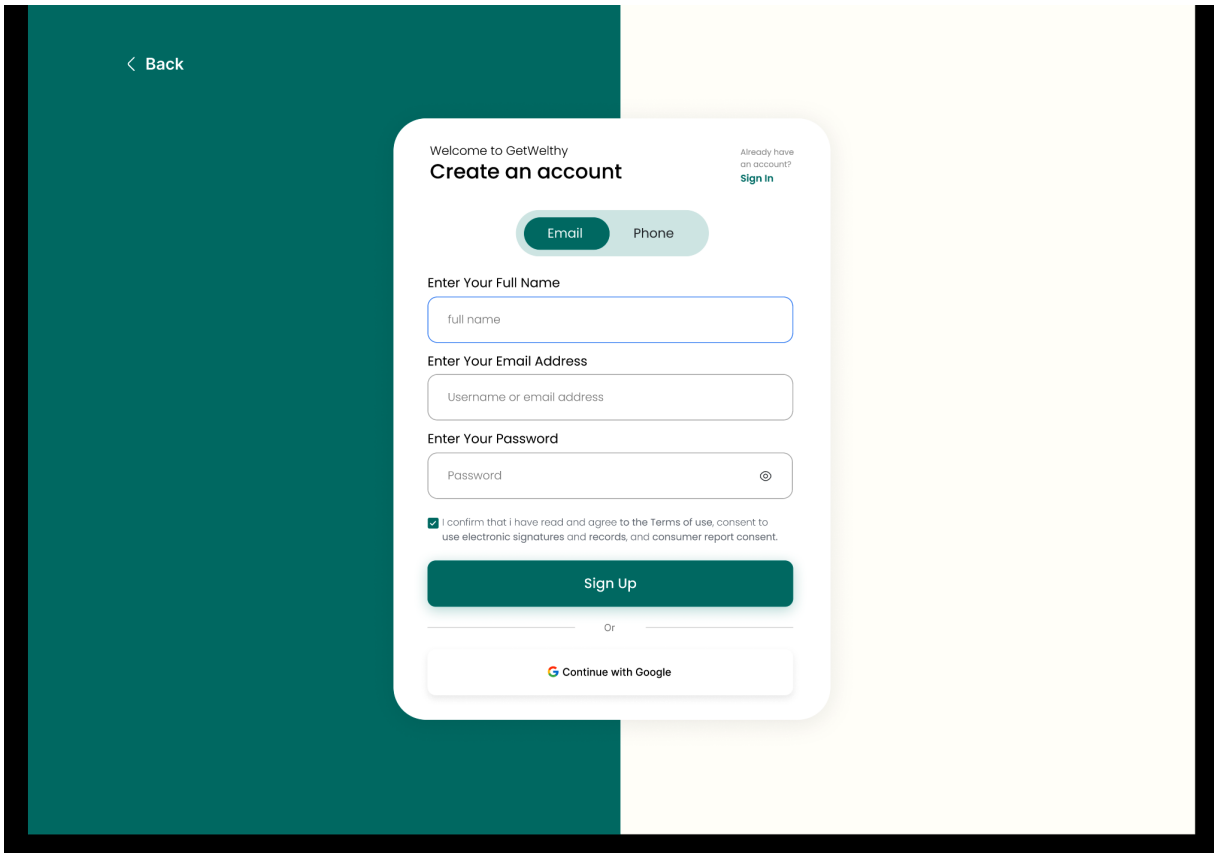

Figure 3.39: Client registration interface

Authentication Interface for client:

The figure below represents the login interface for the client .

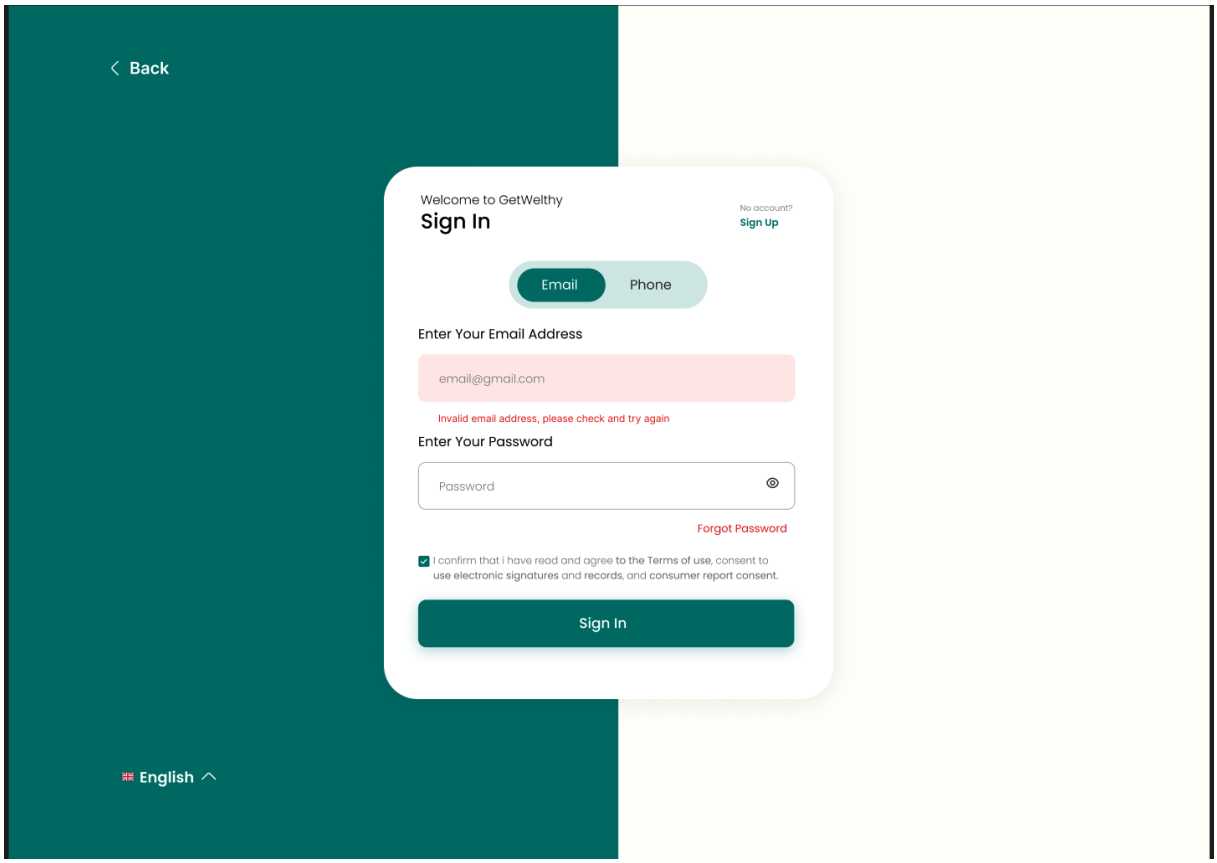

Figure 3.40: Authentication Interface for client

Borrowing capacity calculator Interface:

The figure below represents the Borrowing Capacity Calculator Interface, where clients can determine if they qualify for a loan based on whether their monthly payments can cover the requested loan amount.

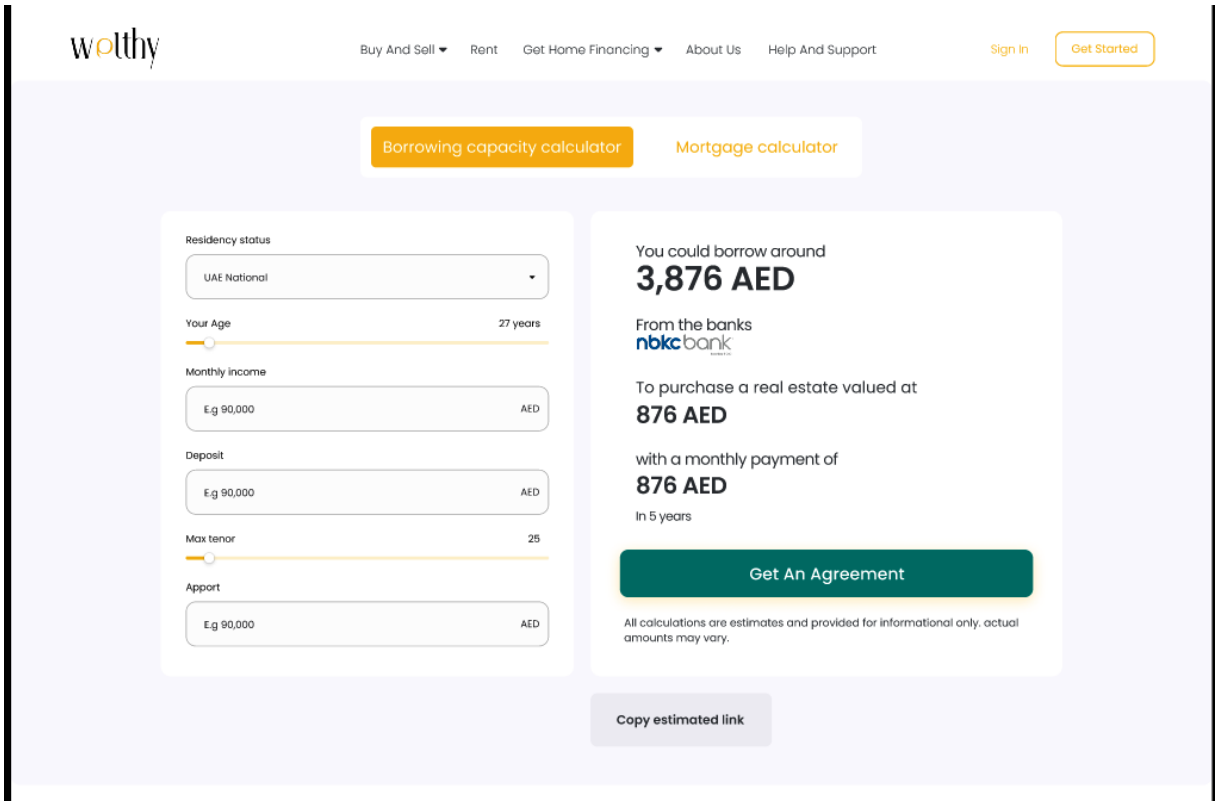

Figure 3.41: Borrowing capacity calculator Interface

Mortgage Calculator Interface:

The figure below represents the Mortgage Calculator Interface, where the client can find out the rate of the payment request and get an idea of how much they will pay monthly.

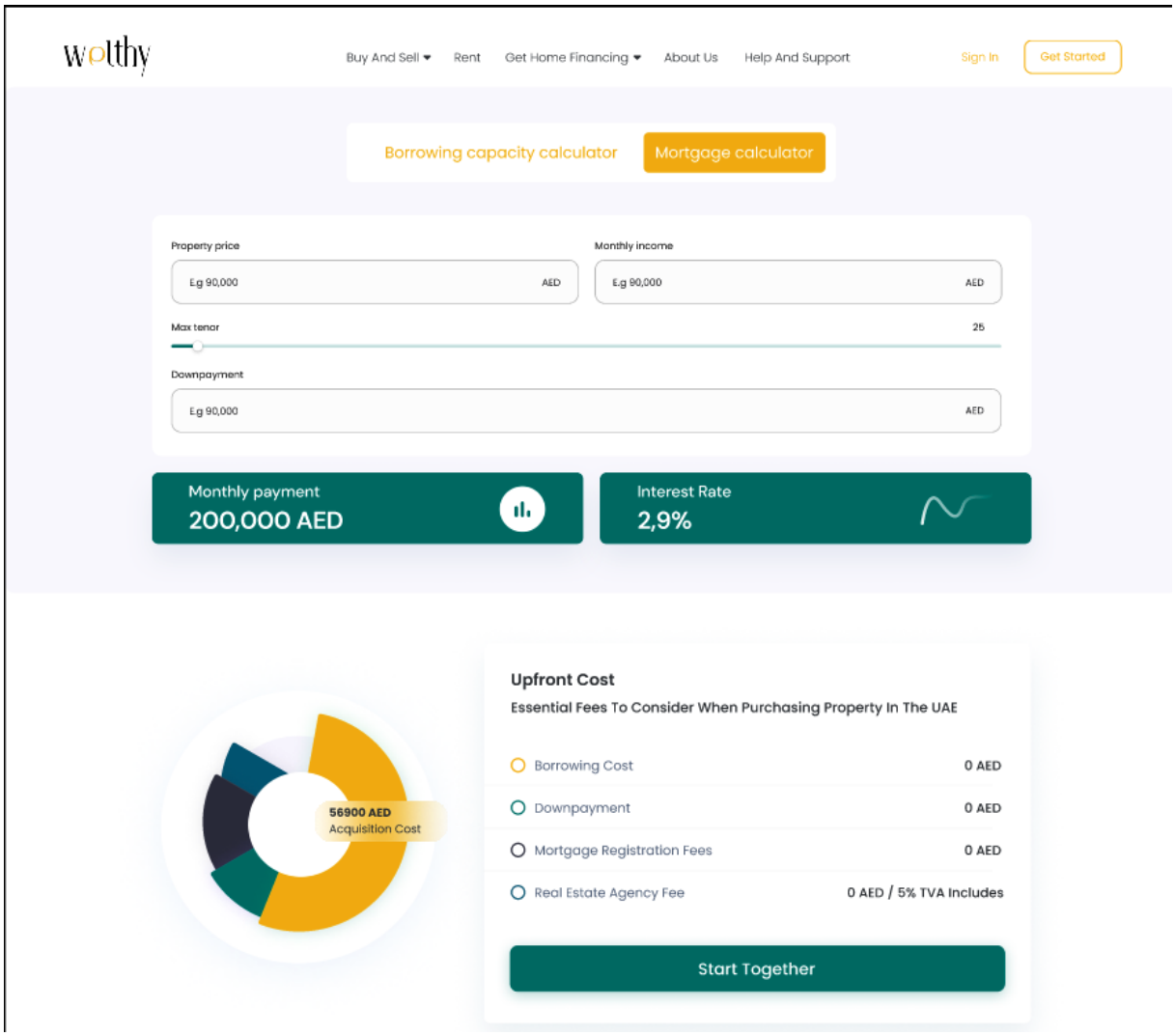

Figure 3.42: Mortgage Calculator Interface e

3.6 Continuous Integration

Continuous Integration (CI) is a software development practice where developers frequently merge their code changes into a central repository, followed by automated builds and tests. The primary goal of CI is to identify and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

3.6.1 Advantages of Continuous Integration:

- **Early Bug Detection:** Continuous Integration helps in catching bugs early in the development cycle, which reduces the cost of fixing them.
- **Reduced Integration Problems:** Regular merging and testing reduce integration issues, which can become significant hurdles at later stages of development if not addressed early.
- **Faster Release Rate:** With automated tools handling integration and testing, the amount of manual work is reduced, enabling faster iterations on product development.
- **Improved Developer Productivity:** Automating the tedious parts of development (like testing and integration) allows developers to focus more on actual coding and problem-solving, boosting their productivity.
- **Enhanced Transparency:** Continuous feedback loops create a transparent culture where every member of the team is aware of the development progress, changes, and issues.
- **Better Test Coverage:** Automated tests run with every change provide a safety net that helps in maintaining and even improving the test coverage of the software.

3.6.2 Dockerizing Application

Front end Image

Just like in a local environment, our Docker container will need React, node modules, packages, Nginx, and other dependencies. In our case, we need a base image with React , and then we write our own Dockerfile.

The image got React and all needed extensions, but we still need to install composer and copy all files to the container. As the next figure shows, we will use a Dockerfile to

install these dependencies:

```
1  FROM node:18-alpine
2
3  WORKDIR /app
4
5  COPY package.json .
6
7  RUN npm install
8
9  COPY . .
10
11 RUN npm run build
12
13 EXPOSE 3000
14
15 CMD [ "npm", "run", "preview" ]
```

Figure 3.43: Front-end Dockerfile for React Application

Back end Image

Our backend system is built using Spring, a powerful and versatile framework for creating Java-based enterprise applications. Just like our front-end, the backend system requires a specific environment setup within our Docker container. We need a base image that supports Java and Spring Boot, along with any necessary libraries and tools such as Maven for dependency management.

The following Dockerfile snippet illustrates the necessary steps to set up our Spring-

based backend environment:

```
FROM maven:3.8.7-openjdk-18 AS build
WORKDIR /build
COPY pom.xml .      You, last month • feature: init
RUN mvn dependency:go-offline
COPY src ./src
RUN mvn clean package -DskipTests
# Runtime stage
FROM amazoncorretto:17
ARG PROFILE=dev
ARG APP_VERSION=3.2.2

WORKDIR /app
COPY --from=build /build/target/backend-pfa-${APP_VERSION}-SNAPSHOT.jar /app/backend-pfa-${APP_VERSION}-SNAPSHOT.jar

EXPOSE 8088

ENV DB_URL=jdbc:postgresql://postgres-sql-bakend:5432/pfaDatabase
ENV MAILDEV_URL=localhost

ENV ACTIVE_PROFILE=${PROFILE}
ENV JAR_VERSION=${APP_VERSION}

CMD java -jar -Dspring.profiles.active=${ACTIVE_PROFILE} -Dspring.datasource.url=${DB_URL} backend-pfa-${JAR_VERSION}-SNAPSHOT.jar
```

Figure 3.44: Backend Dockerfile for Spring Application

3.6.3 Pipeline

Jenkins Pipeline Stages and Post-Build Actions

Stages

- **Checkout:**
  - Retrieves the latest source code from the SCM repository that Jenkins is configured to monitor.
- **Docker Auth:**

  - Executes a shell script to log into Docker Hub using credentials stored in Jenkins. This is necessary for pushing the built image to Docker Hub.

- **Build Docker Image:**

  - Builds the Docker image using the Dockerfile found in the SCM repository. The image is tagged with the build number to maintain version control.

- **Push Docker Image:**

  - Pushes the tagged image to Docker Hub, making it available for deployment or further use.

- **Cleanup Docker Images:**

  - Removes the tagged Docker image from the Jenkins agent to free up disk space. Also logs out of Docker to ensure security.

Post-Build Actions

- **Success:**

  - If the pipeline completes successfully, it sends an email notification to specified recipients indicating success. This step includes a conditional check to send this notification only if the build was triggered from the master branch.

- **Failure:**

  - If any stage of the pipeline fails, it sends an email notification to specified recipients indicating the failure, urging them to check Jenkins console output for details.

```groovy
pipeline {
    agent any
    environment {
        DOCKERHUB_CREDENTIALS = credentials('dh_cred')
        IMAGE_NAME = "${DOCKERHUB_CREDENTIALS_USR}/back-pfa"
    }
    triggers {
        pollSCM('*/5 * * * *') // Check every 5 minutes
    }
    stages {
        stage('Checkout') {
            steps {
                echo "Getting source code"
                checkout scm
            }
        }
        stage('Docker Auth') {
            steps {
                script {
                    sh '''
                    echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin
                    '''
                }
            }
        }
        stage('Build Docker Image') {
            steps {
                script {
                    sh """
                    docker build -t ${IMAGE_NAME}:${env.BUILD_NUMBER} .
                    """
                }
            }
        }
        stage('Push Docker Image') {
            steps {
                script {
                    sh """
                    docker push ${IMAGE_NAME}:${env.BUILD_NUMBER}
                    """
                }
            }
        }
        stage('Cleanup Docker Images') {
            steps {
                script {
                    sh """
                    docker rmi ${IMAGE_NAME}:${env.BUILD_NUMBER}
                    docker logout
                    """
                }
            }
        }
    }
    post {
        success {
            script {
                if (env.BRANCH_NAME == 'master') {
                    emailext subject: 'Build Success: back-pfa',
                             body: 'The pipeline successfully built and deployed to Docker Hub.',
                             recipientProviders: [developers()],
                             to: 'semiayachi.contact@gmail.com'
                }
            }
        }
        failure {
            emailext subject: 'Build Failure: back-pfa',
                     body: 'The pipeline failed to build. Please check the Jenkins console output for more details.',
                     recipientProviders: [developers()],
                     to: 'semiayachi.contact@gmail.com'
        }
    }
}
```

Figure 3.45: Pipeline

verify running pipeline:

The figure below represents the Jenkins interface, where we can see all the stages of the pipeline working correctly.

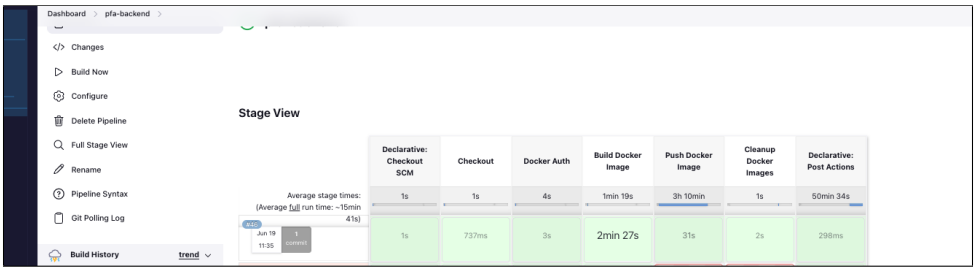

Figure 3.46: verify running pipeline

3.7 Conclusion

In this chapter, we reviewed the environment and methods used to develop our application. We detailed the software and hardware setups, the system architecture, and the deployment processes. The interface we designed and implemented reflects our focus on usability and efficiency. Our continuous integration practices have enhanced our development workflow and maintained high code quality.

Despite some challenges, each was an opportunity to improve our skills and the project's outcomes. These experiences will guide our future work, helping us to continuously refine and innovate. This project underlines our commitment to delivering software that fulfills the evolving needs of our stakeholders.

This streamlined conclusion succinctly wraps up the chapter, emphasizing both the achievements and the ongoing commitment to improvement.

General Conclusion and Perspectives

The work detailed in this report is part of the final year project. It concerns the development of a credit application website, aimed at simplifying and enhancing the interactions between bankers and credit applicants. This website not only facilitates the submission and tracking of credit applications but also plays a crucial role in providing innovative tools to banking agents. These tools enable the processing of applications in a more objective and transparent manner, ensuring greater clarity in credit procedures.

By integrating features such as automated credit assessments, personalized loan offers based on analytical data, and secure management of customer data, this site contributes to an effective digital transformation within the banking sector. Furthermore, the platform promotes smooth and direct communication between applicants and bankers, significantly improving user experience and optimizing decision-making processes.

In the future, this project could be expanded to incorporate advanced technologies such as artificial intelligence to predict credit trends or detect fraud, representing a step forward in the modernization of banking services.

References

[1] BTS,https://www.bts.com.tn/telechargements-2/(Visited 30-03-2024).

[2] BIAT, https://www.biat.tn/biat/Fr/simuler-vos-credits_77_126 (Visited 30-03-2024).

[3] Diagrams.net, https://diagrams.net (Visited 01-06-2022). https://fr.overleaf.com/project

[4] overleaf,https:www.overleaf.com (Visited 01-06-2022).

[5] latex,https:www.overleaf.com (Visited 01-06-2022). https://diagrams.net (Visited 01-06-2022).

[6] Visual Studio Code, https://code.visualstudio.com/ (Visited 21-06-2024).

[7] IntelliJ IDEA, https://www.jetbrains.com/idea/ (Visited 17-06-2024).

[8] Git, https://git-scm.com/ (Visited 23-06-2024).

[9] GitHub, https://github.com/ (Visited 19-06-2024).

[10] Postman, https://www.postman.com/ (Visited 15-06-2024).

[11] Swagger, https://swagger.io/ (Visited 20-06-2024).

[12] Chrome Developer Tools, https://developer.chrome.com/docs/devtools/ (Visited 22-06-2024).

[13] Docker, https://www.docker.com/ (Visited 18-06-2024).

[14] Jenkins, https://www.jenkins.io/ (Visited 24-06-2024).

[15] PostgreSQL, https://www.postgresql.org/ (Visited 25-06-2024).

[16] Java, https://www.oracle.com/java/ (Visited 16-06-2024).

[17] Spring Boot, https://spring.io/projects/spring-boot (Visited 14-06-2024).

[18] React, https://reactjs.org/ (Visited 13-06-2024).

[19] JSX, https://reactjs.org/docs/introducing-jsx.html (Visited 11-06-2024).

[20] Material-UI, https://mui.com/ (Visited 12-06-2024).

[21] Python, https://www.python.org/ (Visited 10-06-2024).

[22] Django, https://www.djangoproject.com/ (Visited 26-06-2024).

[23] NumPy, https://numpy.org/ (Visited 27-06-2024).

[24] Next.js, https://nextjs.org/ (Visited 28-06-2024).