



Département Génie Informatique

Réf :

# Summer Internship Report

*Of*

## Second year in Computer Engineering

Presented and publicly defended on ../../..

*By*

*Najjar Mohamed Nadhir*

---

Development and deployment of a server provisioning and  
management app

---



Composition of the jury

Mr  
Ms

...  
...

President  
Supervisor

Academic Year: 2024-2025

# Contents

<b>List of Figures</b>	<b>iii</b>
<b>List of Tables</b>	<b>iv</b>
<b>Acknowledgments</b>	<b>v</b>
<b>General introduction</b>	<b>1</b>
<b>1 Introduction to the company and general project framework</b>	<b>2</b>
1.1 Introduction . . . . .	2
1.2 Company Overview . . . . .	2
1.2.1 BH Bank overview . . . . .	2
1.3 Context of the project . . . . .	3
1.4 Analysis and Evaluation of Existing Solutions . . . . .	4
1.4.1 Existing solutions . . . . .	4
1.4.2 Comparison table of Existing Solutions . . . . .	6
1.5 Proposed solution . . . . .	7
1.6 Conclusion . . . . .	7
<b>2 Analysis and specification of requirements</b>	<b>8</b>
2.1 Introduction . . . . .	8
2.2 Requirements specification . . . . .	8
2.2.1 Functional requirements . . . . .	8
2.2.2 Non functional requirements . . . . .	9
2.3 Analysis . . . . .	10
2.3.1 Actors identification . . . . .	10
2.3.2 Global use case diagram . . . . .	10
2.3.3 Refinement of use cases . . . . .	12
2.4 Conclusion . . . . .	15
<b>3 Étude conceptuelle</b>	<b>16</b>
3.1 Introduction . . . . .	16
3.2 Architectural model . . . . .	16
3.3 Class diagram . . . . .	17
3.4 Conclusion . . . . .	18
<b>4 Implementation</b>	<b>19</b>
4.1 Introduction . . . . .	19
4.2 Hardware environment . . . . .	19

4.3	Software environment . . . . .	19
4.4	Application interfaces . . . . .	21
4.4.1	Register . . . . .	21
4.4.2	Login . . . . .	21
4.4.3	Create Request . . . . .	22
4.4.4	Request Details . . . . .	22
4.4.5	Incoming Requests . . . . .	23
4.4.6	Admin Dashboard . . . . .	23
4.4.7	Server Details . . . . .	24
4.4.8	Servers list . . . . .	24
4.4.9	User Details . . . . .	25
4.4.10	Users List . . . . .	25
4.5	Conclusion . . . . .	26
<b>5</b>	<b>Deployment</b>	<b>27</b>
5.1	Introduction . . . . .	27
5.2	Provisioning VMs . . . . .	27
5.3	Containerization . . . . .	27
5.3.1	Backend Tier - Node.js . . . . .	28
5.3.2	Frontend Tier - Angular . . . . .	28
5.3.3	Database Tier - MongoDB . . . . .	28
5.4	CI/CD VM Configuration . . . . .	29
5.5	CI/CD pipeline . . . . .	29
5.6	Conclusion . . . . .	30
	<b>General conclusion</b>	<b>31</b>

# List of Figures

1.1	BH Bank, Mohamed V street . . . . .	3
1.2	Ansible . . . . .	4
1.3	Terraform . . . . .	4
1.4	Jira service management . . . . .	5
1.5	ServiceNow . . . . .	5
2.1	Global use case diagram . . . . .	11
2.2	Refinement of use case « Manage users » . . . . .	12
2.3	Refinement of use case "Answer requests" . . . . .	14
3.1	MVC Architectural Model . . . . .	16
3.2	Class Diagram of the System . . . . .	17
4.1	Register Interface . . . . .	21
4.2	Login Interface . . . . .	22
4.3	Create Request Interface . . . . .	22
4.4	Request Details Interface . . . . .	23
4.5	Incoming Requests Interface . . . . .	23
4.6	Admin Dashboard Interface . . . . .	24
4.7	Server Details Interface . . . . .	24
4.8	Servers list Interface . . . . .	25
4.9	User Details Interface . . . . .	25
4.10	Users List Interface . . . . .	26

# List of Tables

1.1	Comparing existing solutions . . . . .	6
2.1	Textual description of the use case 'Manage users' » . . . . .	13
2.2	Textual description of the use case « Answer requests » . . . . .	14
4.1	Machine Specifications . . . . .	19

# Acknowledgments

I would like to convey my heartfelt gratitude to the entire system administration and security team at BH Bank for their invaluable support and guidance throughout my summer internship. I extend special thanks to Mr. Hatem Langar for his outstanding supervision and unwavering encouragement; his insightful feedback and mentorship significantly enhanced my learning experience. I am also deeply appreciative of Mr. Akram Rouis for his exceptional guidance and support. My sincere thanks go to the jury members who took the time to evaluate my work with care and professionalism. Finally, I want to express my appreciation to everyone who supported me during this period, making my internship both enriching and fulfilling.

# General introduction

The rapid advancement of technology has brought significant changes to various industries, particularly in the way services are managed and delivered. Traditional methods are becoming less efficient in meeting the growing demands for speed, flexibility, and scalability. This shift has created an increasing need for innovative solutions that can automate and streamline processes to improve overall efficiency and reduce operational complexities.

In response to these challenges, many organizations are looking for ways to modernize their workflows, improve resource management, and enhance customer satisfaction through digital transformation. The goal is to provide solutions that not only simplify complex tasks but also ensure that services are delivered in a secure, standardized, and timely manner.

Our project aims to address these needs by developing a comprehensive system that optimizes service delivery processes. The solution offers features such as an intuitive user interface, automated workflows, and tools that allow seamless interaction between service providers and their clients. By leveraging modern technologies, the system enhances the efficiency of operations, reduces the potential for errors, and supports the scalability required to meet future demands.

This report outlines the key stages of the project's development. It is divided into several chapters, each covering a different aspect of the solution, from initial analysis and requirement specification to conceptual design and implementation to the deployment on azure virtual machines. The report also highlights the benefits of the system in improving service management and aligning with the broader trends of digital transformation in today's fast-evolving technological landscape.

# Chapter 1

## Introduction to the company and general project framework

### 1.1 Introduction

This first chapter is dedicated to introducing BH Bank and outlining the general framework of the project. We will begin with a brief overview of BH Bank, followed by a discussion of the project's context. Next, we will examine the current state of existing solutions, present the proposed solution, and conclude with the development methodology employed.

### 1.2 Company Overview

In this short section, we will be introducing BH Bank.

#### 1.2.1 BH Bank overview

BH Bank, or "Banque de l'Habitat", is a prominent financial institution based in Tunisia, specializing in housing and real estate financing. Established in 1976, the bank has a long-standing reputation for providing tailored financial solutions to support housing projects and homeownership. BH Bank offers a comprehensive range of services, including personal and business banking, mortgages, and investment products. The bank plays a crucial role in the Tunisian financial sector by facilitating access to housing loans, supporting real estate development, and contributing to the economic growth of the region. With a network of branches across the country, BH Bank is committed to delivering personalized customer service and innovative financial solutions to meet the evolving needs of its clients.





Figure 1.1: BH Bank, Mohamed V street

### 1.3 Context of the project

The rapid evolution of IT infrastructure needs within enterprises has significantly transformed the landscape of server management and provisioning. Traditional manual processes are becoming increasingly inadequate as organizations demand more efficient, automated solutions to manage their server resources. This shift highlights an urgent need for advanced technological solutions that can streamline server provisioning, enhance efficiency, and reduce operational overhead. In this context, organizations are seeking ways to automate and optimize their server provisioning processes to better meet their operational requirements. The challenge is to provide a solution that simplifies the request and approval workflow for server resources, while ensuring secure and standardized provisioning across various cloud platforms. This requires a seamless integration of tools that handle infrastructure management, user requests, and administrative workflows. Our project addresses these needs by developing an Automated Server Provisioning System. This system is designed to improve the efficiency and effectiveness of server management within enterprises. It offers key functionalities such as a user-friendly interface for request submission, a multi-level approval process involving different administrative roles (General Admin, Network Admin, Security Admin, and Super Admin), and automated server provisioning using Infrastructure as Code (IaC) tools like Terraform or Ansible. Additionally, the system includes features for tracking request status, managing approvals, and ensuring compliance with organizational policies. The project is aligned with the broader trend of digital transformation in IT infrastructure, aiming to provide a robust solution that bridges the gap between server provisioning needs and modern technological capabilities. By automating and streamlining these processes, the system enhances operational efficiency, reduces manual errors, and ensures a scalable approach to managing server resources.

## 1.4 Analysis and Evaluation of Existing Solutions

In this section, we will be introducing existing tools and solutions and their offered features.

### 1.4.1 Existing solutions

- **Ansible [1]** : Primarily used for configuration management and automation. It can manage server provisioning when combined with external request management systems but lacks built-in request and approval workflows. 1.2.



Figure 1.2: Ansible

- **Terraform [2]** : Focuses on infrastructure provisioning through code. Like Ansible, it can automate server provisioning but needs integration with other tools for request management and approval workflows. 1.3.



Figure 1.3: Terraform

- **JIRA Service Management [3]** : Provides a comprehensive platform for handling IT service requests, including server provisioning. It includes built-in approval workflows and can integrate with automation tools for provisioning.



Figure 1.4: Jira service management

- **Service Now [4]** : A robust ITSM platform that manages service requests, including server provisioning. It offers built-in request management, approval workflows, and integration capabilities for provisioning tools like Ansible and Terraform.



Figure 1.5: ServiceNow

## 1.4.2 Comparison table of Existing Solutions

At the end of our analysis of current solutions, we identified the shared features and important criteria for comparing and evaluating the different applications. Table 1.1 below presents a comparison of these solutions based on these criteria

Table 1.1: Comparing existing solutions

Criteria \ Tool	Ansible	Terraform	Jira Service	Service Now
<b>Primary Function</b>	Configuration management and automation	Infrastructure provisioning and management	IT service request management	IT service management and automation
<b>Integration with Provisioning</b>	Automate server provisioning and configuration using playbooks	Automates infrastructure provisioning with code	Provisioning actions through integrations or automation plugins	Can trigger provisioning actions through integrations
<b>Automated Provisioning</b>	Yes, through playbooks and roles	Yes, through Terraform configurations	Can be integrated with automation tools for provisioning	Can integrate with provisioning tools and trigger actions
<b>User Interface</b>	Command-line interface or GUI tools like Ansible Tower	Command-line interface or GUI tools like Terraform Cloud	Web-based interface for managing requests and approvals	Web-based interface for managing requests and approvals
<b>Cost</b>	Open-source (Ansible Tower has enterprise licensing)	Open-source (Terraform Cloud has enterprise licensing)	Subscription-based, with various pricing tiers	Subscription-based, with various pricing tiers
<b>Approval Workflow</b>	Requires external tools for approval workflows	Requires external tools for approval workflows	Built-in approval workflows and automation rules	Built-in approval workflows and automation rules

As illustrated in Table 1.1, Ansible, Terraform, Jira Service Management, and ServiceNow vary significantly in their functionalities and integrations. Ansible and Terraform are primarily focused on different aspects of infrastructure management—Ansible excels in configuration management while Terraform is dedicated to infrastructure provisioning. Neither tool offers built-in approval workflows or direct request submission features, nor

do they support automated provisioning of requests. In contrast, Jira Service Management and ServiceNow are tailored for IT service management and feature web-based interfaces. Both support request and approval workflows and allow direct submission of service requests. However, they do not provide native support for automated provisioning tasks, a key feature of tools like Ansible and Terraform. Notably, Jira Service Management and ServiceNow offer robust interfaces for managing IT services, but neither integrates directly with automated provisioning, which is a distinctive capability of the infrastructure-focused tools. Thus, while Ansible and Terraform are suited for backend provisioning and configuration, Jira Service Management and ServiceNow are geared towards broader IT service management and request handling.

## 1.5 Proposed solution

We envision the development of an innovative web application, "Automated Server Provisioning," tailored to revolutionize server provisioning within enterprises. This sophisticated platform will provide users with a streamlined interface for submitting detailed server requests, specifying essential attributes such as CPU, memory, storage, and other specific requirements. Upon submission, requests will be routed through a well-defined approval workflow, where various administrators—including General Admins, Network Admins, Security Admins, and Superadmins—will review and approve them based on their designated roles, ensuring compliance with organizational standards. Once a request is approved, the application will utilize advanced automation tools like Ansible and Terraform to efficiently manage the provisioning process. This includes configuring network settings, deploying required software, and setting up server environments accurately and promptly. Furthermore, the application will offer robust server management capabilities, allowing users to oversee, update, and decommission servers as necessary. Detailed server information, such as IP addresses, usernames, and passwords, will be readily accessible, providing transparency and control over server resources. The platform will feature a dynamic dashboard for real-time tracking of request statuses, historical data, and notifications regarding request approvals, rejections, and provisioning updates. Seamless integration with existing CI/CD pipelines will ensure that server provisioning aligns smoothly with development workflows. "Automated Server Provisioning" is designed to enhance efficiency, control, and visibility in server management, addressing the complexities of server provisioning while delivering a user-centric experience that meets both operational needs and organizational objectives.

## 1.6 Conclusion

In this chapter, we introduced BH Bank, provided an overview of the entire project, and conducted a comparative analysis of the existing solutions, highlighting their various features. In the next chapter, we will discuss the analysis and specification of requirements.

# Chapter 2

## Analysis and specification of requirements

### 2.1 Introduction

At this point, we will provide a detailed specification of the functional and non-functional requirements for the project. We will begin by identifying the key Actors involved, then proceed to present a comprehensive use case diagram, along with further refinement of specific use cases.

### 2.2 Requirements specification

Before starting the development of an application, it is essential to go through the requirements specification phase. In this section, we outline the functional and non-functional requirements of our application to better understand the system's context.

#### 2.2.1 Functional requirements

Functional requirements define what the product or service developed within the project must achieve or provide. The system being designed must allow for:

##### 2.2.1.1 For server requesters:

- **Register and authentication** : Requesters should be able to create an account using their employee ID and email, and securely log in afterward.
- **Request a server** : Each user should have the ability to request a server by filling out a form with the desired specifications and submitting it to the administrators for approval.
- **Communication intégrée** : L'application doit offrir des moyens de communication, tels que des messages et des appels vocaux, pour faciliter les échanges entre clients et prestataires.
- **View the request's response** : Users should be able to view the status of their request at any time, and once approved, they can access the server via SSH using the credentials provided in the response.

- **Request server removal** : When a server is no longer needed, users can submit a request to delete it. The request is sent to the administrators, who will manually stop and delete the server after completing the necessary verifications.

#### 2.2.1.2 For admins :

- **View requests** : General admins should be able to view all requests, check general specifications details.
- **Approve or reject requests** : After reviewing the general details, general admins should have the ability to either approve the request or reject it if the server specifications are deemed inefficient or unnecessary for the intended purpose.
- **Set network configurations** : After the server request has been accepted by general admin, the request is forwarded to the network admin that will work on setting up the necessary network configurations depending on the department, the role of the requester etc..
- **Trigger server creation** : when network configuration is done, network admin triggers the creation of the server via a pre-configured Ansible playbook template that will call for Azure cloud APIs and create the server with the specified configurations.
- **Manage servers** : Admins should be able to view all servers with their corresponding details, filters them by the creation date, the CIDR range and the name of the requester etc.

#### 2.2.1.3 For super admin :

- **Managing users** : Super admin should be able to create, delete a user, assign, change or remove a role from any user.
- **Manage requests** : Super admin should be able to view requests with details, accept or reject a request.
- **Manage servers** : Super admin can view servers with details, delete a server at any time. He should be able also to check for removal requests.
- **View general statistics** : Super admin should have the ability to check all the general statistics and logs including number of users per departments, number of running servers per department, number requests arranged by their status, and much more, inside a comprehensive dashboard.

### 2.2.2 Non functional requirements

Non-functional requirements are vital criteria for users, as they ensure the system operates effectively and efficiently. It is essential for the developed system to fulfill these requirements.

- **Performance** : The application should handle a large number of concurrent users submitting requests without significant delay.

- **Security** : Role-based access control (RBAC) should be implemented to ensure that only authorized users (admins) can approve or reject server requests. Also user password should be stored in an encrypted state in the database.
- **Usability** : The application should provide a clean and intuitive user interface for both end-users (requesters) and admins. Error messages should be clear and provide sufficient detail for users to understand the issue and take corrective action.

## 2.3 Analysis

The analysis aims to identify all Actors involved in the application's system and to illustrate their interactions through a use case diagram.

### 2.3.1 Actors identification

This application includes 4 Actors :

- **Super admin** : Super admin is charged of the global management of the application, his main role is to supervise and view all the general stats and logs concerning created requests and their status, created VMs, users and their roles, he has the absolute privileges to create users, remove users, view requests, view servers, assign roles and remove roles etc.. .
- **General admin** : The general admin reviews the request submitted by the requester. If the request meets the approval criteria, the admin approves it; otherwise, the request is rejected.
- **Network admin** : Once the request is approved, it is forwarded to the network admins, they will have to set the network configuration such as private IP address, subnet mask, firewall etc., after that, he triggers the creation of the vm using a preset Ansible playbook, Network admin can also make configurations on the playbook before execution.
- **Requester** : The requester can fill a form specifying the server he needs, some of the specifications are : Machine name, CPUs, disk space, OS, RAMs, purpose, open ports, services etc.. After submitting, he should wait for the approval process to be launched. If the request is approved and the server is created with success, he should find his new server in his dashboard with the credentials needed to ssh into the machine.

### 2.3.2 Global use case diagram

The general use case diagram allows for the analysis of the system's various functionalities and the identification of use cases, specifying the actors involved. Figure 2.1 illustrates the global use case diagram of our system.



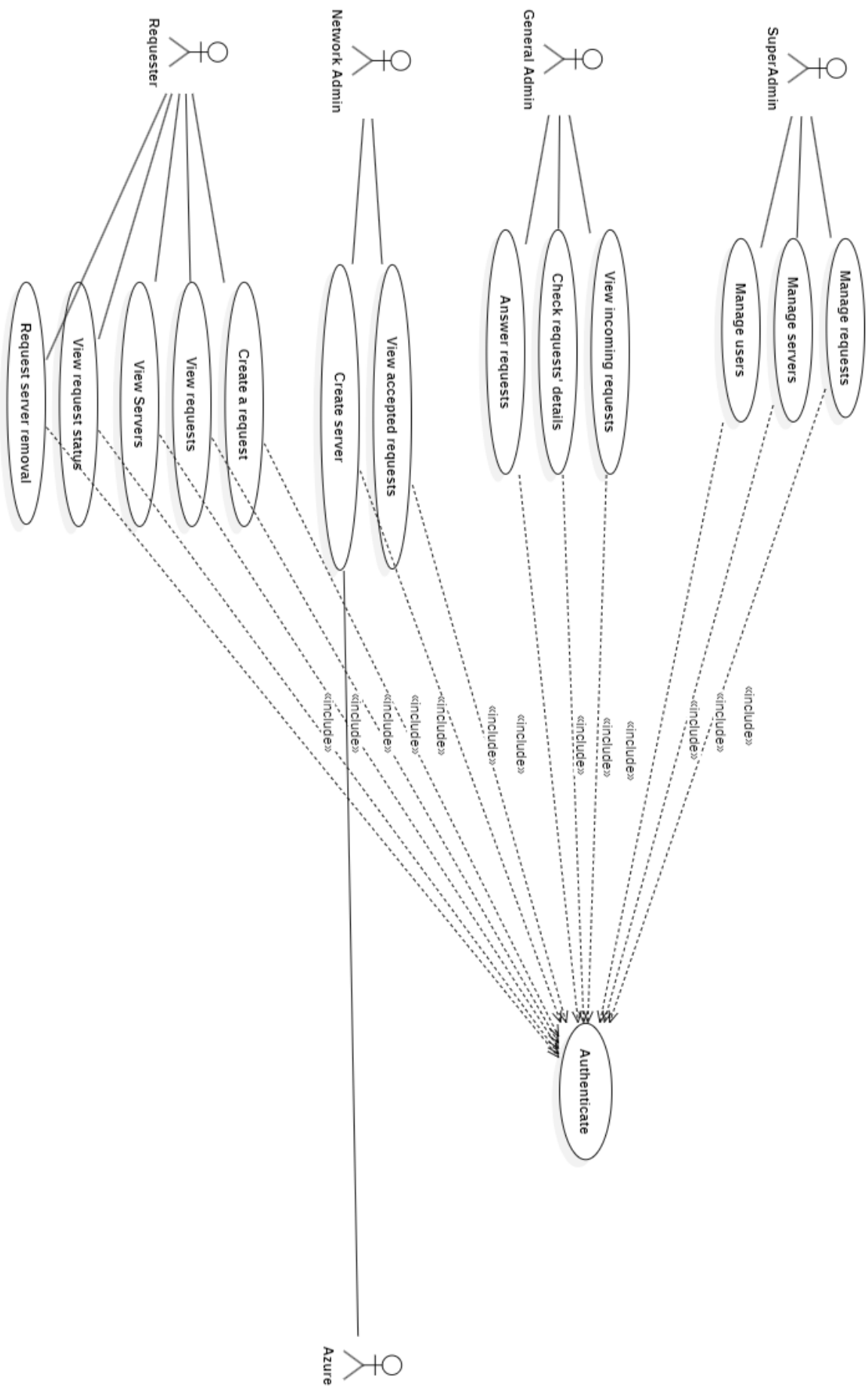


Figure 2.1: Global use case diagram

### 2.3.3 Refinement of use cases

In this section, we conduct an in-depth analysis of selected use cases, illustrating them with detailed diagrams accompanied by textual descriptions.

#### 2.3.3.1 Refinement of the use case « Manage users »

##### -Use case diagram « Manage users »

The figure below shows the use case « Manage users ». The super user is allowed to create a user, specifically a new admin (admins cannot create their own accounts, either they are created by a super admin, or they register as normal users, and a admin role is assigned to them by the super admin also). He can also assign roles to any user, or delete a user.

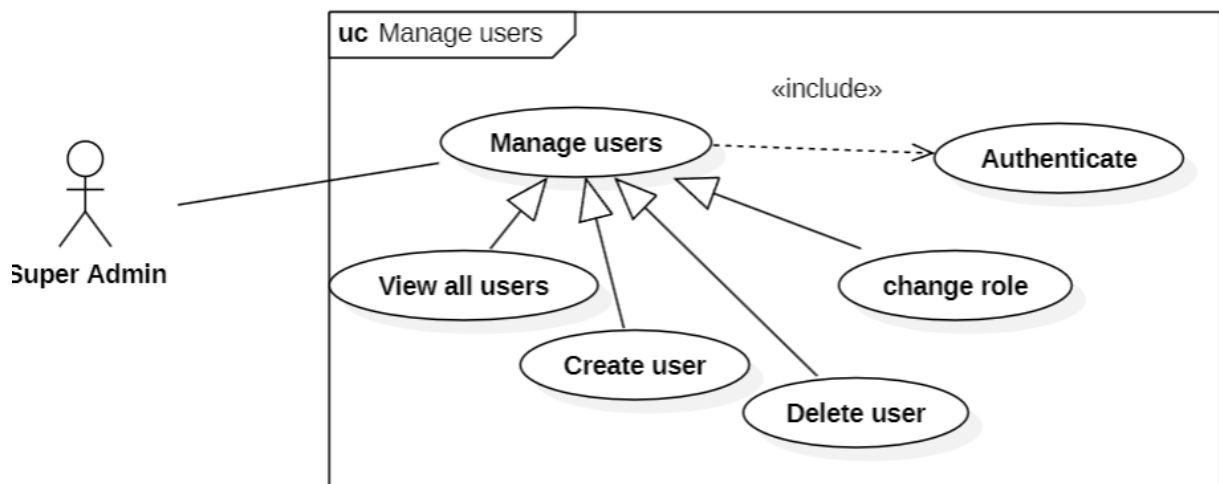


Figure 2.2: Refinement of use case « Manage users »

##### -Textual description of the use case 'Manage users'.»

Table 2.1: Textual description of the use case 'Manage users' »

<b>Title</b>	Manage users
<b>Actor</b>	Super user
<b>summary</b>	a use case that allows super admin to manage users in the admin space.
<b>Pre-conditions</b>	User must be authenticated and has the super admin privileges.
<b>Post-conditions</b>	Super admin can view all users with details, open their profiles, change a user's role and permissions, or delete a user.
<b>Standard scenario</b>	<ol style="list-style-type: none"> <li>1. Super admin logs into his account.</li> <li>2. Super admin selects a user from the users list and opens his profile.</li> <li>3. Super user chooses either to change the user's role, view his account details, or delete his account permanently.</li> </ol>
<b>Errors flow</b>	<p>E1 : User is not authenticated.</p> <ul style="list-style-type: none"> <li>- The system redirects the user to the login page.</li> </ul> <p>E2 : User is authenticated but does not have super admin privileges</p> <ul style="list-style-type: none"> <li>- An error message is displayed to tell that user does not have enough permissions.</li> </ul> <p>E2 : Super user tries to delete a none existing or deleted user.</p> <ul style="list-style-type: none"> <li>- Error message that indicates that the user is not found.</li> </ul>

#### -Use case diagram of « Answer request »

The diagram below illustrates the use case titled "Answer Request." In this scenario, the general administrator has the authority to respond to server requests based on various criteria. The administrator can choose to accept the request if it aligns with the organization's standards regarding resource allocation, the requester's historical usage patterns, and the purpose for which the server is needed. Conversely, the request may be rejected if it fails to meet these standards, such as if the requested resources exceed available capacity, the requester has a poor history of utilization, or the reason for the server usage does not comply with the company's operational guidelines.

The decision-making process ensures that server resources are allocated efficiently and in a manner that upholds the enterprise's policies and priorities, ultimately contributing to effective resource management and organizational compliance.

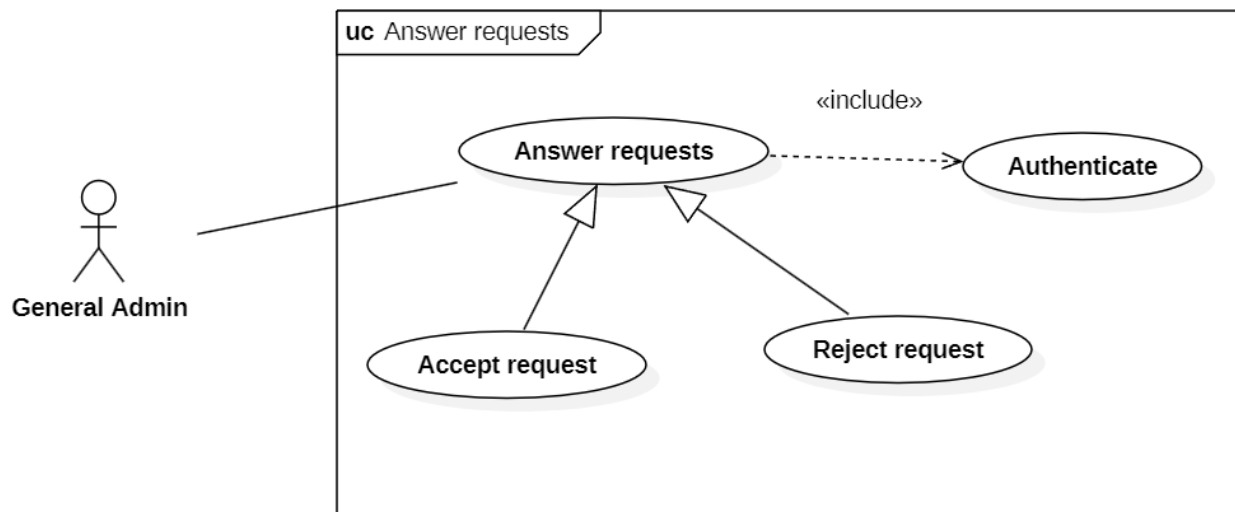


Figure 2.3: Refinement of use case "Answer requests"

#### -Textual description of the use case « Answer requests »

Table 2.2: Textual description of the use case « Answer requests »

<b>Title</b>	Refinement of the use case « Answer requests »
<b>Actor</b>	General admin
<b>Summary</b>	Use case that allows general admin to deal with the requests either by accepting them or rejecting them after checking server specifications and user profile.
<b>Pre-conditions</b>	User should be authenticated and have general admin privileges.
<b>Post-conditions</b>	The request should be either accepted or rejected for some reason.
<b>Standard scenario</b>	<ol style="list-style-type: none"> <li>1. User logs in as a general admin.</li> <li>2. General admin finds all new incoming requests detailed in a table and chooses a request.</li> <li>3. General admin views the details of the request.</li> <li>4. If all conditions are verified and server specifications are justified, the general admin accepts the request, otherwise, he rejects it, and a response is sent to the requester.</li> </ol>
<b>Errors flow</b>	<p>E1: User is not authenticated.</p> <ul style="list-style-type: none"> <li>- The system redirects him to the login page.</li> </ul> <p>E2: The user does not have the general admin privileges.</p> <ul style="list-style-type: none"> <li>- An error message pops up indicating that current user is not allowed to perform this action.</li> </ul>

## 2.4 Conclusion

In this chapter, we explored the analysis and definition of the project's functional and non-functional requirements to better understand the problem and identify the necessary features. We also identified the architectural model of the app, the class diagram and explained different entities. The next chapter will be dedicated to the implementation phase.

# Chapter 3

## Étude conceptuelle

### 3.1 Introduction

The conceptual study is a crucial step in the planning and execution of a project. It involves analyzing, organizing, and designing the various components of the system to ensure a clear, scalable, and easy-to-maintain architecture. In this chapter, we will explore the software architecture adopted for our project, particularly the Model-View-Controller (MVC) model, as well as the static structure of the system through a class diagram. This approach allows the code to be structured into distinct components, facilitating the management and evolution of the application over time.

### 3.2 Architectural model

In our project, we follow the Model-View-Controller (MVC) architecture, which organizes an application into three distinct components: the model, view, and controller. This separation of concerns enhances code management and scalability, with each component serving a specific role:

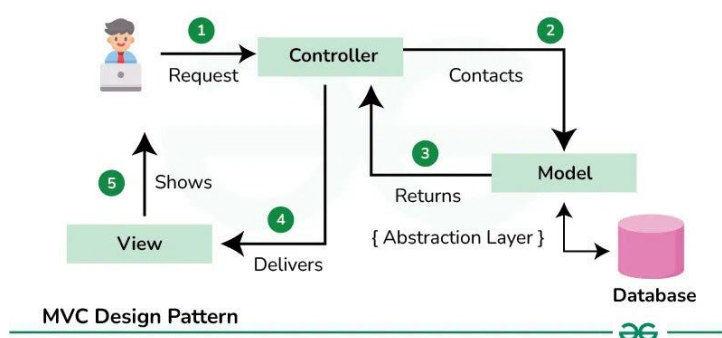


Figure 3.1: MVC Architectural Model

- View: Responsible for presenting the user interface, the view retrieves data from the model and displays it visually to the user. It handles only the presentation layer, without any business logic.

- Controller: Acting as a mediator between the view and the model, the controller processes user inputs from the view, applies the necessary business logic, and updates both the model and view. It orchestrates the interaction between these components.

- Model: The model represents and manages the application's data. It handles data retrieval, processing, and validation, focusing on business rules and data integrity, independent of how the data is displayed in the view.

### 3.3 Class diagram

A class diagram visually represents the classes, attributes, and relationships within a software system. It depicts the system's static structure by focusing on the main entities and how they interact with one another.

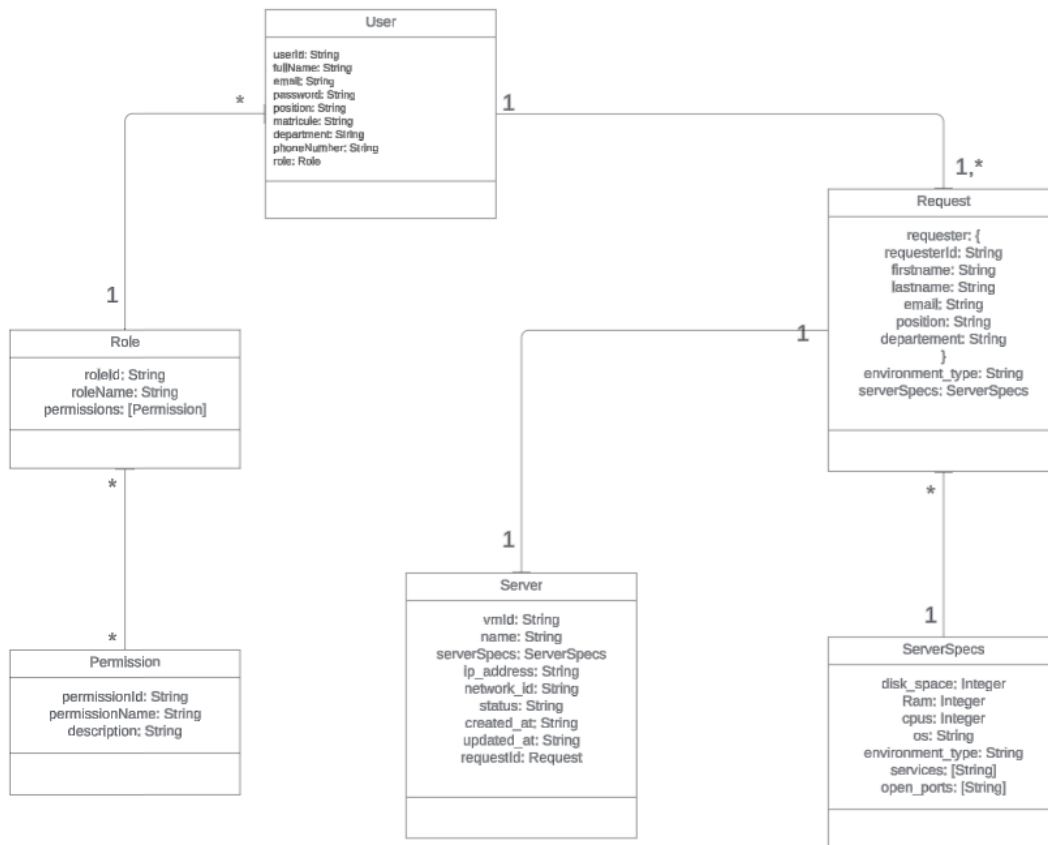


Figure 3.2: Class Diagram of the System

The class diagram of our app includes the following entities:

- **User**: This class defines the four roles available in our application: Requester, General Admin, Network Admin, and Super Admin. It stores key user details such as name, employee ID (matricule), email, password, and other relevant information.
- **Role**: The Role class defines system roles, comprising a unique role ID, the role name, and a list of associated permissions that specify the actions the role can perform.
- **Permission**: The Permission class represents the various permissions within the system. It includes a unique permission ID, a permission name that specifies the

type of permission, and a description that provides additional details about the permission's purpose and functionality.

- **Request:** The Request class models user requests for server provisioning, containing attributes such as a unique request ID, user ID, server specifications, status, and timestamps for submission and updates.
- **Server:** The Server class represents provisioned servers in the system. It includes attributes such as server specifications (serverSpecs), server name, unique identifier (id), IP address, creation timestamp, and associated request ID, allowing for effective management and tracking of server instances.
- **ServerSpecs:** The ServerSpecs class defines the specifications for each server, including attributes like CPU, RAM, storage capacity, and operating system. This class ensures that each server meets the required specifications as outlined in user requests.

### 3.4 Conclusion

The MVC architecture and class diagrams are crucial for ensuring a clear separation of responsibilities within the system, facilitating management, maintenance, and scalability. By structuring components carefully, we lay the foundation for a robust and adaptable system that meets future needs. This conceptual study provides insight into the application's internal organization, ensuring efficient and functionally aligned implementation. In the next chapter, we will explore the implementation phase, translating the design into working code, addressing challenges, and ensuring the system meets performance, security, and scalability requirements.



# Chapter 4

## Implementation

### 4.1 Introduction

In this chapter, we present the hardware and software environment, as well as the technologies used for the development of our application. Then, we will provide details on some of the developed interfaces and features.

### 4.2 Hardware environment

The hardware specifications of the machine used for the development of the application are summarized in the table. 4.1 :

Table 4.1: Machine Specifications

<b>Manufacturer</b>	HP Pavilion
<b>Processor</b>	AMD Ryzen 7 5000 Series
<b>RAM</b>	32.0 GB
<b>Hard Disk</b>	1 TB
<b>Operating System</b>	Red Hat Enterprise Linux (RHEL 8)

### 4.3 Software environment

During the project, we used various software tools, including the following:



**Editor:** Visual Studio Code [5]: Visual Studio Code is an extensible code editor developed by Microsoft for Windows, Linux, and macOS. It features debugging support, syntax highlighting, intelligent code completion, snippets, code refactoring, and integrated Git.



**Design:** Figma [6]: Figma is a vector graphics editor and prototyping tool primarily web-based, with additional offline features enabled by desktop applications for macOS and Windows.



**API Testing Tool:** Postman [7]: Postman is an API development platform that allows developers to efficiently test, debug, and document APIs, offering features such as HTTP request creation, environment management, and report generation.



**Database:** MongoDB [8]: MongoDB is a NoSQL database that stores data in a flexible, JSON-like format, allowing for easy scalability and high performance in handling large volumes of data.



**Frontend:** Angular [9] and Bootstrap [9]: Angular is a platform and framework for building client-side applications in HTML and TypeScript, known for its two-way data binding and modular architecture. Bootstrap is a front-end framework for developing responsive and mobile-first websites, featuring pre-designed components and a grid system to simplify layout design.



**Backend:** Node.js [10] and Express.js [11]: Node.js is a JavaScript runtime built on Chrome's V8 engine, enabling developers to build scalable network applications using JavaScript on the server side. Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications.



**VCS:** Git and GitHub [11]: Git is a distributed version control system for tracking changes in source code during software development. GitHub is a web-based platform that uses Git for version control and enables collaboration among developers.



**Containerization:** Docker [12]: Docker is a platform for developing, shipping, and running applications in containers, ensuring consistency across different environments and facilitating easier deployment.



**Configuration Management as Code:** Ansible [13]: Ansible is an open-source automation tool that helps in configuration management, application deployment, and task automation, using a simple YAML syntax.



**Apache:** [14]: is an open-source web server software that hosts and serves web content. Known for its flexibility and performance, it supports various operating systems and allows extensive customization through modules. Apache is widely used for hosting websites and applications.



**CI/CD:** Jenkins [15]: Jenkins is an open-source automation server that supports building, deploying, and automating projects, providing a robust framework for continuous integration and continuous delivery.

## 4.4 Application interfaces

In this section, we will be displaying some of the main interfaces for the Server provisionner.

### 4.4.1 Register

The Register interface allows users to create an account within the application. It is divided into two sections: the Provider and the Client. Each section has its own unique functionalities while sharing common features like chat and community space. Users can input their details, including a unique username, email, and password, to initiate their registration process. This interface ensures that users are aware of the terms and conditions before completing their registration.

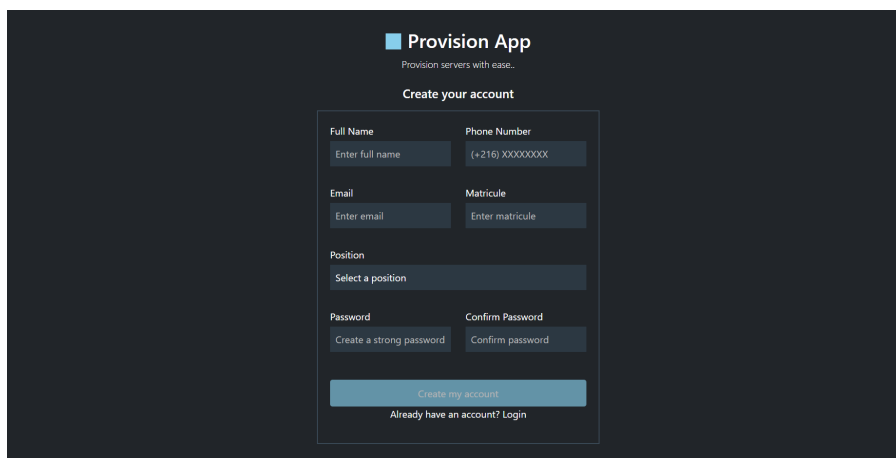
The image shows a dark-themed user interface for a 'Provision App'. At the top, the app's logo and name 'Provision App' are displayed, followed by the tagline 'Provision servers with ease..'. Below this, a section titled 'Create your account' contains a registration form. The form is organized into several fields: 'Full Name' with a placeholder 'Enter full name', 'Phone Number' with a placeholder '(+216) XXXXXXXX', 'Email' with a placeholder 'Enter email', and 'Matricule' with a placeholder 'Enter matricule'. There is also a 'Position' field with a placeholder 'Select a position'. At the bottom of the form, there are 'Password' and 'Confirm Password' sections, each with a placeholder 'Create a strong password' and 'Confirm password' respectively. A large blue button labeled 'Create my account' is positioned below the form, and a link 'Already have an account? Login' is located at the very bottom.

Figure 4.1: Register Interface

### 4.4.2 Login

The Login interface enables users to authenticate their accounts to access the application. Like the Register interface, it is also divided into the Provider and Client sections. Users can input their registered email and password to log in. The interface includes features such as a "Remember Me" option and a "Forgot Password" link to facilitate easy account recovery. It also provides access to shared functionalities like chat and community space.

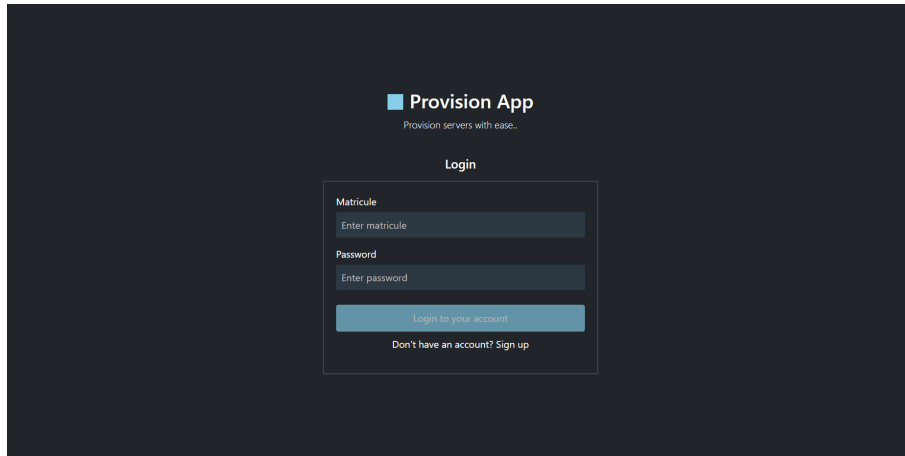


Figure 4.2: Login Interface

### 4.4.3 Create Request

The Create Request interface allows users to submit requests for server provisioning. Users can specify their requirements by filling out necessary details such as server specifications, purpose, and urgency. The interface is designed for simplicity, guiding users through the request submission process and ensuring that all relevant information is captured.

Figure 4.3: Create Request Interface

### 4.4.4 Request Details

The Request Details interface provides users with a detailed view of their submitted requests. This includes information such as the creation date, the user who initiated the request, the admin who accepted it, and the required server specifications. This interface allows users to track the status of their requests and understand the details involved.

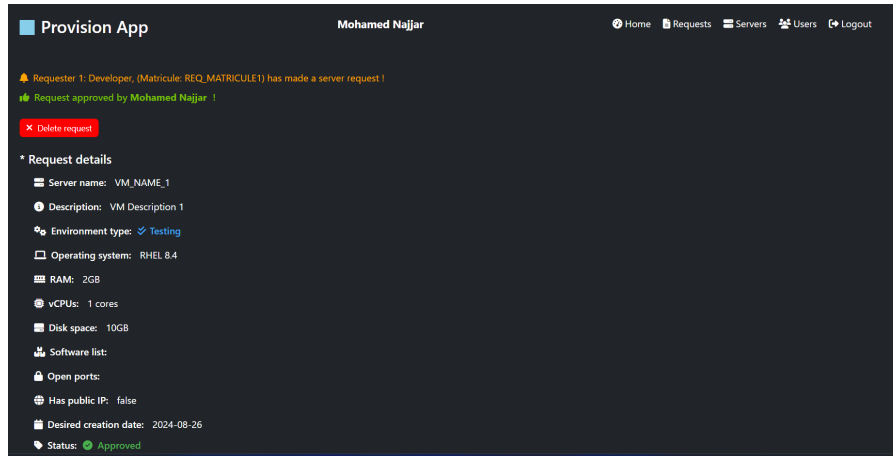


Figure 4.4: Request Details Interface

#### 4.4.5 Incoming Requests

The Incoming Requests interface displays all active requests submitted by users awaiting approval. Admins can view the details of each request, including user information and requested specifications, enabling them to manage and prioritize requests efficiently. This interface serves as a centralized location for handling incoming requests.

The screenshot shows the 'Provision App' interface for user 'Mohamed Najjar'. The 'Incoming Requests' section is active, displaying a table of requests. The table has columns for #, Name, Desired start date, Request created on, Status, and Actions. The data is as follows:

#	Name	Desired start date	Request created on	Status	Actions
1	VM_NAME_1	2024-08-26	2024-08-26	approved	
2	VM_NAME_2	2024-08-26	2024-08-26	pending	
3	VM_NAME_3	2024-08-26	2024-08-26	rejected	
4	VM_NAME_4	2024-08-26	2024-08-26	pending	
5	VM_NAME_5	2024-08-26	2024-08-26	pending	
6	VM_NAME_6	2024-08-26	2024-08-26	approved	
7	VM_NAME_7	2024-08-26	2024-08-26	pending	
8	VM_NAME_8	2024-08-26	2024-08-26	pending	

Figure 4.5: Incoming Requests Interface

#### 4.4.6 Admin Dashboard

The Admin Dashboard provides a comprehensive overview of system activities, including user management, server allocations, and request tracking. Admins can access statistics related to users, servers, requests, departments, and logs. This interface is essential for monitoring the overall health of the system and making informed decisions.

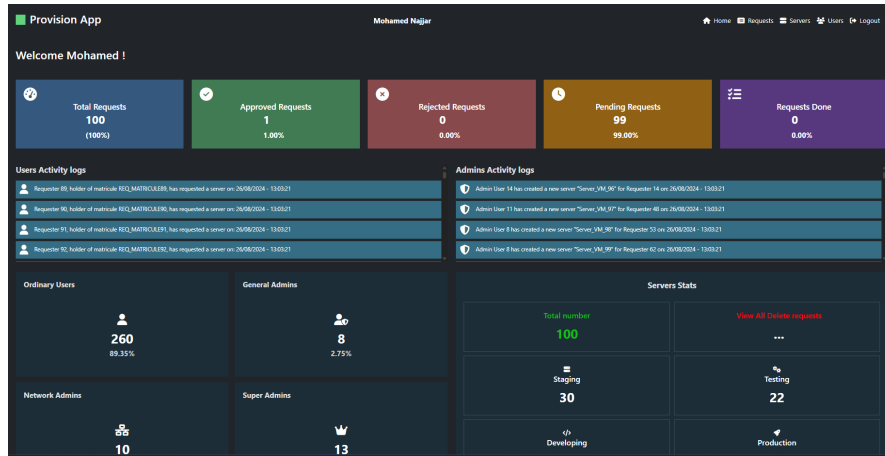


Figure 4.6: Admin Dashboard Interface

#### 4.4.7 Server Details

The Server Details interface showcases the specifics of each created server. Users can view essential information such as server specifications, ownership, creation date, and network configurations. This interface aids in managing server-related details and allows users to monitor their resources effectively.

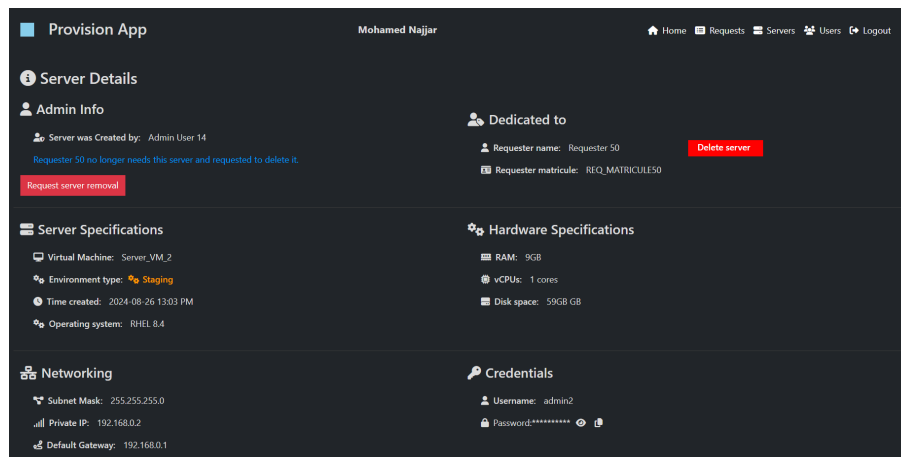


Figure 4.7: Server Details Interface

#### 4.4.8 Servers list

The My Servers interface lists all the servers owned by the logged-in user. This allows users to easily manage and track their servers in one place. They can view server statuses, specifications, and take necessary actions as required.

#	VM Name	Creation Date	Environment	Operating system	CPU (vCPUs)	Memory (GB)	Storage (GB)	Private IP	Actions
1	Server_VM_1	2024-08-26	staging	SUSE Linux Enterprise Server 15	6 cores	7GB	43GB	192.168.0.1	<a href="#">🔗</a>
2	Server_VM_2	2024-08-26	staging	RHEL 8.4	1 cores	9GB	59GB	192.168.0.2	<a href="#">🔗</a>
3	Server_VM_3	2024-08-26	production	Ubuntu 20.04 LTS	2 cores	3GB	25GB	192.168.0.3	<a href="#">🔗</a>
4	Server_VM_4	2024-08-26	staging	SUSE Linux Enterprise Server 15	6 cores	7GB	99GB	192.168.0.4	<a href="#">🔗</a>
5	Server_VM_5	2024-08-26	development	SUSE Linux Enterprise Server 15	6 cores	14GB	36GB	192.168.0.5	<a href="#">🔗</a>
6	Server_VM_6	2024-08-26	production	Debian 11	5 cores	13GB	49GB	192.168.0.6	<a href="#">🔗</a>
7	Server_VM_7	2024-08-26	staging	Debian 11	5 cores	13GB	35GB	192.168.0.7	<a href="#">🔗</a>
8	Server_VM_8	2024-08-26	testing	RHEL 8.4	1 cores	2GB	94GB	192.168.0.8	<a href="#">🔗</a>
9	Server_VM_9	2024-08-26	production	CentOS 8	4 cores	5GB	69GB	192.168.0.9	<a href="#">🔗</a>
10	Server_VM_10	2024-08-26	development	Ubuntu 20.04 LTS	2 cores	10GB	88GB	192.168.0.10	<a href="#">🔗</a>
11	Server_VM_11	2024-08-26	development	CentOS 8	4 cores	12GB	14GB	192.168.0.11	<a href="#">🔗</a>

Figure 4.8: Servers list Interface

#### 4.4.9 User Details

The User Details interface allows admins to view and manage information about users in the system. This includes personal details, roles, and activity logs. Admins can change the user's permissions, or delete the user if they want.

**Provision App** Mohamed Najjar

[Home](#) [Requests](#) [Servers](#) [Users](#) [Logout](#)

### User Details

Full name: Admin User 1  
 Email Address: admin1@gmail.com  
 Account creation date: 24/07/2024, 21:08 UTC  
 Matricule: ADMIN\_MATRICULE1  
 Position: Manager  
 Admin role: GeneralSpecAdmin

Change User Role: Select Role Update role

Delete User Account

- General Admin
- Network Admin
- Super Admin
- Ordinary User

Figure 4.9: User Details Interface

#### 4.4.10 Users List

The Users List interface provides a comprehensive list of all registered users. Admins can filter and search for users based on different criteria. This interface is essential for user management and ensuring the proper oversight of user activities within the application.

**Provision App** Mohamed Najjar Home Requests Servers Users Logout

**Users list**

Find user: Search  Create User

#	Name	Role	Matricule	Email	Actions
1	Mohamed Najjar	SuperAdmin	123456	najjarmohamed443@gmail.com	
2	Admin User 1	GeneralSpecAdmin	ADMIN_MATRICULE1	admin1@gmail.com	
3	Admin User 2	NetworkAdmin	ADMIN_MATRICULE2	admin2@gmail.com	
4	Admin User 3	SuperAdmin	ADMIN_MATRICULE3	admin3@gmail.com	
5	Admin User 4	GeneralSpecAdmin	ADMIN_MATRICULE4	admin4@gmail.com	
6	Admin User 5	NetworkAdmin	ADMIN_MATRICULE5	admin5@gmail.com	
7	Admin User 6	SuperAdmin	ADMIN_MATRICULE6	admin6@gmail.com	
8	Admin User 7	GeneralSpecAdmin	ADMIN_MATRICULE7	admin7@gmail.com	
9	Admin User 8	NetworkAdmin	ADMIN_MATRICULE8	admin8@gmail.com	

Figure 4.10: Users List Interface

## 4.5 Conclusion

This chapter focused on presenting the hardware and software environment used throughout our project, thereby providing an essential overview for understanding our development approach. Additionally, we carefully detailed and analyzed the various interfaces of the application, emphasizing their design and significance in enhancing the user experience.



# Chapter 5

## Deployment

### 5.1 Introduction

This chapter outlines the deployment process of our MEAN (MongoDB, Express, Angular, Node.js) stack application. The deployment leverages modern DevOps practices to ensure automation, scalability, and fault tolerance. Key practices include Infrastructure as Code (IaC) for provisioning and managing infrastructure, Continuous Integration and Continuous Deployment (CI/CD) pipelines to automate testing and releases, and configuration management to maintain consistent environments. Our solution relies on several industry-standard tools, including Azure Virtual Machines (VMs), Docker for containerization, Ansible for automation, Jenkins for CI/CD, SonarQube for code analysis, and Trivy for vulnerability scanning.

This chapter will guide you through the following phases:

- Provisioning VMs on Azure
- Containerizing the application
- Configuring CI/CD on a dedicated VM
- Integrating Jenkins with various tools and processes
- Deploying to Apache

### 5.2 Provisioning VMs

For our infrastructure, we utilized Azure to provision a Red Hat virtual machine. The machine is of type `Standard_B2ms` with 2 vCPUs and 8 GB of RAM to host the CI/CD tools, including Jenkins, Git, SonarQube, Trivy, and Docker, Apache etc... An installation manual that includes all the necessary commands to help installing all of these services was prepared.

### 5.3 Containerization

Containerization is a critical part of our deployment strategy, allowing us to package each tier of the application into portable containers. These containers ensure that the

application behaves consistently across different environments. We utilized Docker to create and manage these containers for the backend (Node.js), frontend (Angular), and database (MongoDB) tiers.

### 5.3.1 Backend Tier - Node.js

For the backend, we created a custom Dockerfile. This Dockerfile defined the steps required to package our Node.js application into a container image. The Dockerfile included:

- Specifying the base image (Node.js LTS version)
- Installing project dependencies using `npm install`
- Copying the source code into the container
- Exposing the necessary port (typically port 3000 for the Node.js server)
- Defining the `CMD` to start the Node.js application

This container ensures that the backend application, including any dependencies and configurations, is encapsulated and ready for deployment.

### 5.3.2 Frontend Tier - Angular

The Angular frontend was similarly containerized. A custom Dockerfile was created to:

- Use a Node.js base image to build the Angular project
- Install Angular CLI and dependencies
- Build the Angular project to generate production-ready static files
- Use Apache as HTTP server to serve the static files
- Expose port 80 for accessing the Angular application

This Dockerfile ensures that the frontend is served efficiently and can be seamlessly deployed alongside the backend.

### 5.3.3 Database Tier - MongoDB

For the database tier, we opted to pull the official MongoDB image directly from DockerHub. This image provides a pre-configured MongoDB server, reducing the need for a custom Dockerfile. The MongoDB container is configured to:

- Expose the default MongoDB port (27017)
- Persist data using Docker volumes, ensuring that data is stored even if the container is restarted

By pulling the official MongoDB image, we minimize configuration overhead and ensure that we are using a well-maintained and secure image.

## 5.4 CI/CD VM Configuration

The dedicated CI/CD VM was configured to host multiple tools that automate the deployment pipeline. We began by SSHing into the VM and installing the necessary software:

- **Jenkins:** Installed as the primary automation server for CI/CD.
- **SonarQube:** Set up for static code analysis and quality gate checks.
- **Docker:** Installed to manage container builds and image deployments.
- **Trivy:** Integrated for scanning Docker images for vulnerabilities.
- **Git:** Installed to manage version control for the application code.

Network configurations were also applied, allowing specific ports for the various services:

- Port 8080 for Jenkins
- Port 80 for HTTP (serving the Angular app)
- Port 9000 for SonarQube

These configurations ensure that the necessary services are accessible, and that the application can be deployed and managed remotely.

## 5.5 CI/CD pipeline

The CI/CD pipeline, managed through Jenkins, automates the process of testing, building, and deploying the application. The pipeline includes several stages, each designed to ensure code quality and streamline the deployment process:

- **Git Checkout:** Jenkins pulls the latest code from the Git repository.
- **SonarQube Analysis:** Static code analysis is performed to identify potential bugs and vulnerabilities.
- **Trivy Scan:** Docker images are scanned for known vulnerabilities before deployment.
- **Build:** The application code is compiled, and any necessary artifacts (e.g., production builds of the frontend) are generated.
- **Docker Build:** Docker images are created for both the frontend (Angular) and backend (Node.js).
- **Push Images:** The Docker images are uploaded to Docker Hub.
- **Ansible Configuration:** A pre-configured Ansible playbook was used to automate the setup of the RHEL8 Azure VM, install needed services (like Apache web server), and deploy the app. After the playbook successfully ran, the application was up and running.

The CI/CD pipeline not only automates deployment but also ensures that every change to the codebase is thoroughly tested, analyzed, and secured before reaching production.

Despite these challenges, the deployment process highlighted the importance of careful planning, automation, and the use of robust DevOps tools.

## 5.6 Conclusion

The deployment of our application was a success, thanks to the strategic use of containerization, and automated CI/CD pipelines. By leveraging modern DevOps practices and tools, we ensured a stable and secure deployment of the Server provision application on Azure infrastructure.

# General conclusion

This project successfully encompassed the conception, implementation, and deployment of a web application designed for server provisioning and management, utilizing the MEAN stack (MongoDB, Express.js, Angular, and Node.js) for a robust and dynamic user experience. The project aimed to streamline the process of server management while ensuring a user-friendly interface for various stakeholders within the enterprise.

The initial conception phase involved thorough requirements gathering, where we identified the key functionalities needed to support server provisioning workflows. This laid a solid foundation for the implementation phase, where the MEAN stack was effectively utilized to develop a responsive and efficient web application. The combination of Angular for the frontend and Node.js with Express.js for the backend provided a seamless experience, while MongoDB served as a flexible and scalable database solution.

For deployment, we adopted a comprehensive DevOps approach, employing Jenkins to automate our Continuous Integration and Continuous Deployment (CI/CD) pipelines. This facilitated efficient collaboration between development and operations teams, ensuring rapid and reliable releases of new features and updates. The use of Docker allowed us to containerize our application, promoting consistency across different environments and simplifying dependency management. Additionally, Kubernetes (K8s) provided robust orchestration for our containerized application, enabling dynamic scaling and high availability.

The integration of DevOps tools such as Trivy for vulnerability scanning and SonarQube for continuous code quality assessment further strengthened our deployment pipeline, ensuring both security and maintainability of the application. These tools enabled proactive monitoring and quality assurance throughout the development lifecycle.

In summary, this project exemplifies the synergy between modern web development practices and DevOps methodologies. The successful implementation and deployment of the server provisioning and management application not only enhanced operational efficiency but also fostered a culture of continuous improvement and collaboration within the team. The insights gained from this endeavor will inform future projects and further advancements in our DevOps practices, ensuring we remain agile and responsive to the evolving needs of our enterprise environment.

# Netography

1. **Ansible** [1]  
Ansible. "Ansible - Simple IT Automation." <https://docs.ansible.com/ansible/latest/index.html>
2. **Terraform** [2]  
HashiCorp. "Terraform: Infrastructure as Code." <https://www.terraform.io/intro>
3. **Jira** [3]  
Atlassian. "Jira Software: Your Project Management Tool." <https://www.atlassian.com/software/jira/guides>
4. **ServiceNow** [4]  
ServiceNow. "What is ServiceNow?" <https://www.servicenow.com/products/what-is-servicenow.html>
5. **Visual Studio Code** [5]  
Microsoft. "Visual Studio Code." <https://code.visualstudio.com/docs>
6. **Figma** [6]  
Figma. "Figma: The Collaborative Interface Design Tool." <https://www.figma.com/resources/learn-design/>
7. **Postman** [7]  
Postman. "Postman - The Collaboration Platform for API Development." <https://www.postman.com/api-documentation-tool/>
8. **MongoDB** [8]  
MongoDB, Inc. "MongoDB: The Database for Modern Applications." <https://www.mongodb.com/what-is-mongodb>
9. **Angular** [9]  
Angular. "Angular - The Modern Web Developer's Platform." <https://angular.io/guide/architecture>
- Bootstrap** [9]  
Bootstrap. "Bootstrap - The World's Most Popular Framework for Building Responsive, Mobile-First Sites." <https://getbootstrap.com/docs/5.0/getting-started/introduction/>
10. **Node.js** [10]  
Node.js. "Node.js - JavaScript Runtime." <https://nodejs.org/en/docs/guides/>
- Express.js** [11]  
Express. "Express - Fast, unopinionated, minimalist web framework for Node.js." <https://expressjs.com/en/starter/installing.html>
11. **Git** [11]  
Git. "Git - the simple, yet powerful version control system." <https://git-scm.com/doc>

**GitHub** [11]

GitHub. "GitHub: Where the world builds software." <https://docs.github.com/en/get-started/quickstart>

12. **Docker** [12]

Docker. "Docker - Empowering developers to build, share, and run applications." <https://docs.docker.com/get-started/>

13. **Apache** [14]

Apache Software Foundation. "Apache HTTP Server." <https://httpd.apache.org/docs/current/>

14. **Jenkins** [15]

Jenkins. "Jenkins: An open-source automation server." <https://www.jenkins.io/doc/>

## Abstract

The project of developing a server provisioning application aims to simplify and speed up servers management and provisioning in an enterprise environment. The application includes several functionalities like: creating new server request, creating servers, managing users, dashboard for logging etc... The application deployment on Azure virtual machine was automated using a CI/CD pipeline made with Jenkins and many other DevOps tools.

## Résumé

Le projet de développement d'une application de provisionnement de serveurs vise à simplifier et à accélérer la gestion et le provisionnement des serveurs dans un environnement d'entreprise. L'application inclut plusieurs fonctionnalités telles que : la création de nouvelles demandes de serveurs, la création de serveurs, la gestion des utilisateurs, un tableau de bord pour la journalisation, etc. Le déploiement de l'application sur une machine virtuelle Azure a été automatisé à l'aide d'une pipeline CI/CD réalisée avec Jenkins et de nombreux autres outils DevOps.