REPUBLIC OF TUNISIA
Ministry of Higher Education and Scientific Research
University of Carthage

National Institute of Applied Sciences and Technology

Personal Professional Project

Facility Reservation Platform

Authored by:

Bchini Mohamed Aziz
Ferjani Oussama
Mejri Louay
Zghal Ines

Under the guidance of:

Mr. Mohamed Ali Zormati

RT3 - Groupe 1

Academic Year : 2023/2024

Abstract

This report details the development of a web application designed to streamline the facility reservation system within the National Institute of Applied Science and Technology (INSAT). The application enables club presidents to book classrooms for their events or workshops at specific dates and times. Built using the MERN stack (MongoDB, Express.js, React.js, Node.js) and containerized with Docker, the project leverages modern web technologies to ensure robustness, scalability, and ease of deployment. The web app features a user-friendly interface for booking management, for both club president and admin. This report covers the project objectives, technology stack, system requirements, architecture, implementation details, and future enhancements, providing a comprehensive overview of the development process.

Table of contents

List of figures

1. Introduction

1.1 Background

Managing student life and activities within a college setting can be challenging, especially when it comes to organizing events and workshops. Club presidents at the National Institute of Applied Science and Technology (INSAT) often face difficulties in booking classrooms, leading to conflicts and inefficiencies. Traditional methods of reserving facilities, such as manual booking through administrative offices or basic online sheets, are prone to errors and double bookings. To address these issues, we propose the development of a web application that streamlines the process of reserving classrooms for events and workshops. This project aims to create a user-friendly platform that allows club presidents to book facilities easily and efficiently, and allows admin to easily manage booking requests, ensuring better management of college resources and student activities at INSAT.

1.2 Problem Statement

The current system for booking classrooms within INSAT (Through online reservation sheets and manual approval) is inefficient and prone to conflicts. Club presidents often find it difficult to secure rooms for their events, resulting in scheduling conflicts, and last-minute cancellations. There is a lack of a centralized system that provides real-time availability and booking capabilities, which disrupts the smooth operation of student activities.

1.3 Objectives

The primary objectives of this project are:

- To develop a web application that enables club presidents at INSAT to book classrooms for their events and workshops.
- To provide a real-time availability calendar to avoid booking conflicts.
- To streamline the reservation process, making it more efficient and user-friendly.
- To implement administrative features for managing and approving reservations.
- To improve the overall management of student life at INSAT.

2. Technology Stack

2.1 MERN Stack

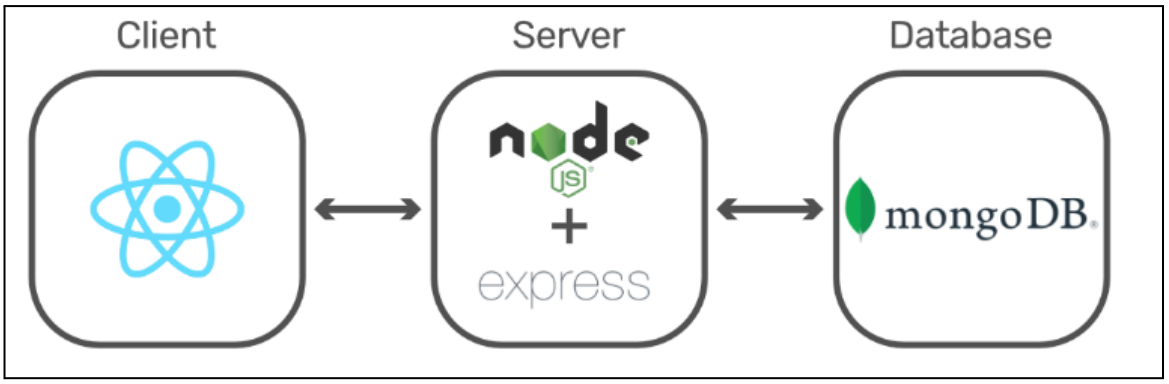


fig. 1

What is the MERN stack ?

MERN Stack is a popular and powerful technology stack used for building full-stack web applications. It includes four technologies:

- MongoDB: MongoDB is a NoSQL database that stores data in JSON-like format. Mongodb is designed such that it handles large amounts of data and provides high-quality scalability and performance. It's a lot more flexible if compared with SQL databases like MySQL.

- Express.js: Express.js is a backend web application framework for Node.js. It simplifies the process of development of a scalable server-side web application. Express.js provides utilities for handling HTTP requests, routing, and middleware which makes it easier to create REST APIs with Express.js.

- React.js: React is a modern JavaScript library utilized for crafting user interfaces (UIs). With React, developing front-end interactive web applications becomes straightforward. Its component-based methodology simplifies the process of building scalable applications. Moreover, React boasts an extensive ecosystem of libraries, further streamlining feature implementation for developers.

- Node.js: Node.js is a runtime environment that enables developers to write JavaScript code on the server side. It uses an event-driven, non-blocking I/O model, making it well-suited for building scalable and real-time applications.

Node.js can handle a high number of concurrent connections without blocking the execution of other tasks.

Why MERN Stack?

- Full-Stack JavaScript: MERN stack allows developers to use a single programming language for the entire application. Whether it be frontend or backend.

- Reusability: With the component-based architecture of React, developers practice one of the most important principles of development which is DRY (Do not repeat yourself). With React developers can create reusable UI elements, reducing repetition and saving time.

- Real-time applications: Node.js's event-driven architecture enables the creation of real-time applications, such as chat applications and collaborative tools.

- Scalability: MongoDB's ability to handle large amounts of data and Node.js's non-blocking I/O make the stack suitable for building scalable applications.

- Large Community Support and Documentation: The MERN stack has strong community support and thorough documentation, aiding developers with resources, issue troubleshooting, and staying current, creating a supportive development ecosystem.

MERN vs Other Technologies?

	MERN Stack	**MEAN Stack**	**Spring + Angular**
Database	MongoDB (NoSQL) - Document-oriented, flexible schema	MongoDB (NoSQL) - Document-oriented, flexible schema	Relational Database (SQL) - Structured data, strong relationships
Backend Language	Node.js (JavaScript) - Event-driven, asynchronous	Node.js (JavaScript) - Event-driven, asynchronous	Java - Object-oriented, mature, robust ecosystem

Front end Framework	React (Component-based) - Declarative, reusable components	Angular (Two-way data binding) - Full-featured, opinionated	Angular (Two-way data binding) - Full-featured, opinionated
Suitable for	Real-time apps, SPAs, APIs, rapid prototyping	Complex, data-heavy SPAs, enterprise-grade apps	Enterprise apps, microservices, complex data modeling
Data Binding	Unidirectional (predictable state management)	Bidirectional (simplifies development but can lead to complexity)	Bidirectional (simplifies development but can lead to complexity)
Popularity	High (Growing) - Modern and versatile	High (Established) - Widely used and mature	High (Established) - Enterprise-backed and stable
Community & Support	Large, Active, JavaScript-focused	Large, Active, JavaScript-focused	Large, Active, Java & JS focused
Performance	Good for real-time (async) - Efficient for handling concurrent requests	Good for complex UIs - Handles data-heavy interfaces well	Good for scalability - Optimized for large deployments
Scalability	Horizontal (easy to add servers) - Scales well with increased traffic	Horizontal (easy to add servers) - Scales well with increased traffic	Vertical (powerful, complex) - Requires careful server configuration
Security	Requires careful implementation (framework-specific)	Requires careful implementation (framework-specific)	Strong focus on security features within Java ecosystem
Testing	Wide variety of JavaScript testing frameworks available	Wide variety of JavaScript testing frameworks available	Wide variety of Java and JavaScript testing frameworks available
Deployment	Wide range of deployment options (cloud platforms, containers)	Wide range of deployment options (cloud platforms, containers)	Deployment options vary depending on chosen server and framework
Cost	Generally lower due to open-source nature of technologies	Generally lower due to open-source nature of technologies	Costs may vary depending on chosen server and licensing (Java)

2.2 NGINX

fig. 2

What is NGINX ?

NGINX is a high-performance web server and reverse proxy server that has gained widespread popularity in the world of web development and deployment. Originally developed to solve the C10K problem (handling 10,000+ simultaneous connections). NGINX has evolved into a robust solution capable of serving static and dynamic content, load balancing, caching, and acting as a reverse proxy for HTTP, HTTPS, SMTP, and other protocols.

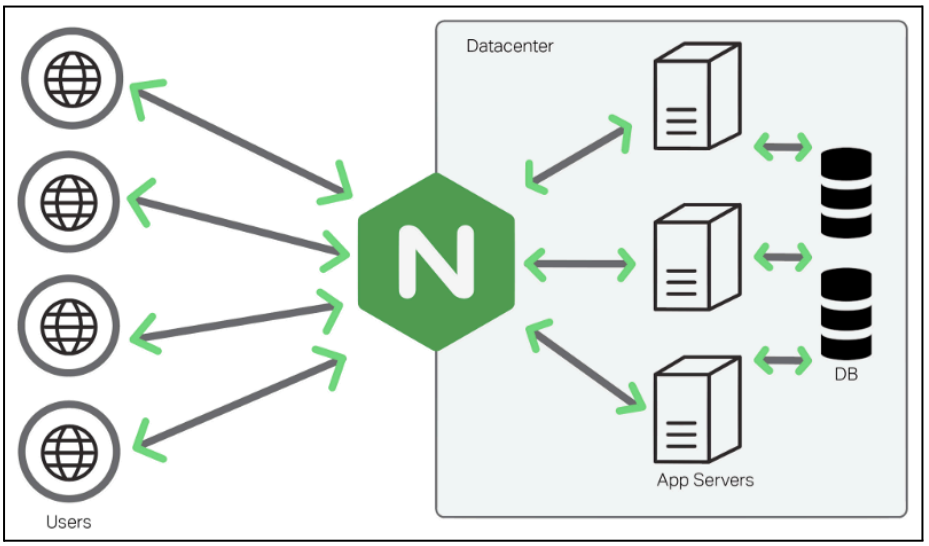


fig. 3

What makes Nginx so powerful?

- Efficient Web Server: Handles high traffic with minimal resources, ideal for static content.
- Reverse Proxy: Distributes incoming requests efficiently.
- SSL/TLS Termination: Robust support for secure connections.
- Caching and Content Delivery: Stores frequently accessed content, improving response times.
- Configuration Flexibility: Fine-grained control over server behavior, adaptable to specific needs.
- Extensibility: Modular architecture allows for custom functionality integration.
- Community and Support: Large user community, abundant documentation and tutorials.
- Scalability and High Availability: Load balancing ensures accessibility during spikes or failures.

2.3 Docker

fig. 4

What is Docker ?

Docker is a platform that uses containerization to streamline software development. It packages applications into containers, which are lightweight and include only essential elements. This ensures applications run consistently across different environments, improving efficiency and portability in software deployment.

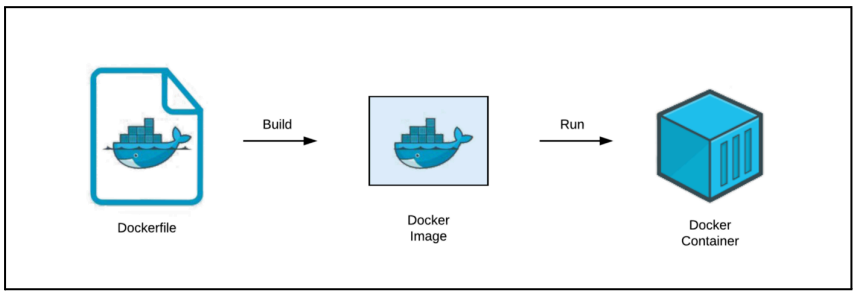


fig. 5

Docker vs Virtualization ?

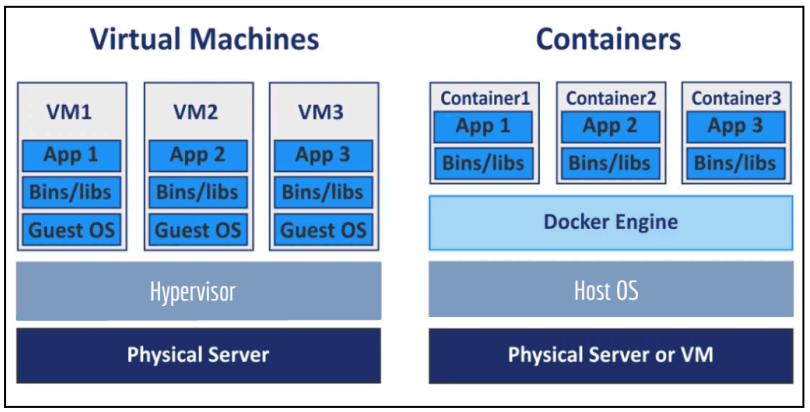


fig. 6

Core concepts

Docker is built on several core concepts that form the foundation of its functionality:

- Docker Images: Blueprints of applications containing everything needed to run software, including code, runtime, libraries, and config files. Immutable once created.

- Containers: Runtime instances of Docker images, isolating applications from their environment for consistent performance across different stages.
- Dockerfiles: Text scripts automating Docker image creation, ensuring repeatability and consistency by defining a series of commands.
- Docker Hub (Registry): Cloud-based registry service for Docker images, facilitating sharing and access to a vast library of images contributed by Docker and its user community for distribution and version control.

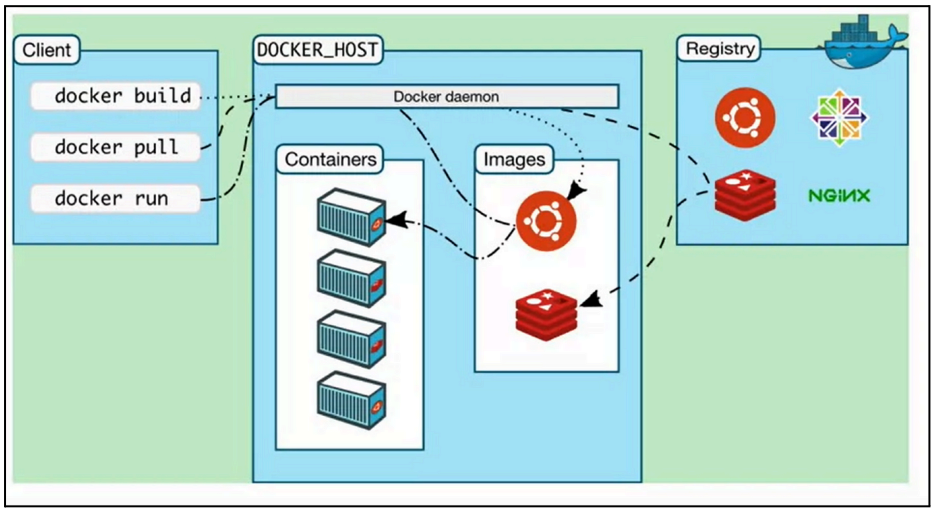


fig. 7

Why Docker?

- Consistent & Isolated Environment: Docker ensures each container accesses resources independently, reducing dependencies and risks like downtime. It maintains consistency across development stages.
- Rapid Application Deployment: Docker's containerized processes skip booting an OS, saving time. With standardized environments, app deployment becomes faster and more efficient, ideal for CI/CD workflows.
- Scalability & Flexibility: Docker enables easy scaling across servers. Changes can be made, tested, and rolled out swiftly, allowing for efficient application management without downtime.
- Better Portability: Docker containers are highly portable, running on various platforms without hassle. This flexibility streamlines development, reduces setup time, and enhances agility.
- Cost-Effective: Docker optimizes resource usage, allowing multiple containers on one server, reducing infrastructure and workforce costs while maintaining quality.

- In-Built Version Control System: Docker facilitates version control, enabling easy rollbacks to stable versions. Containers maintain configurations and reuse components for efficiency.
- Security: Docker provides isolation, enhancing application security. Each container has limited access, reducing the risk of unauthorized data access. However, comprehensive security measures beyond Docker are necessary for overall protection.

3. System Architecture

3.1 MVC Model

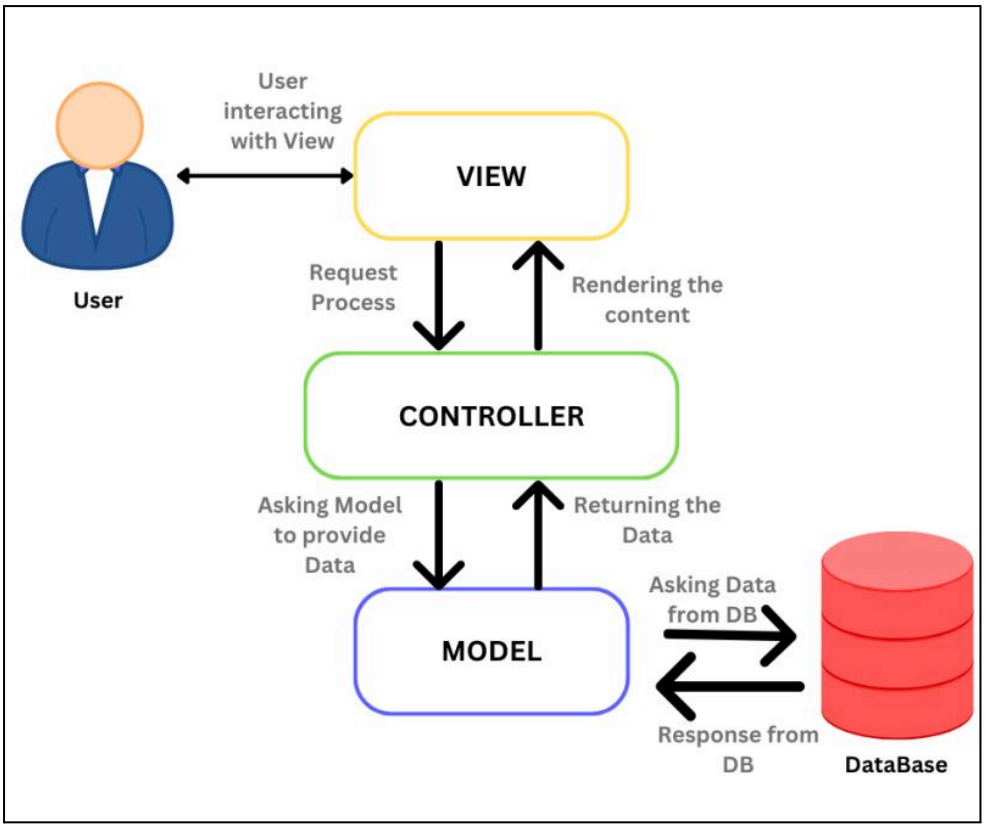


fig. 7

What is MVC ?

The Model-View-Controller (MVC) pattern is an architectural pattern that separates the concerns of data management, user interface, and user input control. It consists of three main components:

- Model: Represents the data and business logic of the application. (handled by mongoDB)
- View: Represents the user interface and displays the data from the model. (handled by React js)
- Controller: Handles user input and updates the model and view accordingly. (handled by Express.js)

3.2 Diagrams

3.2.1 MERN Stack Architecture Diagrams

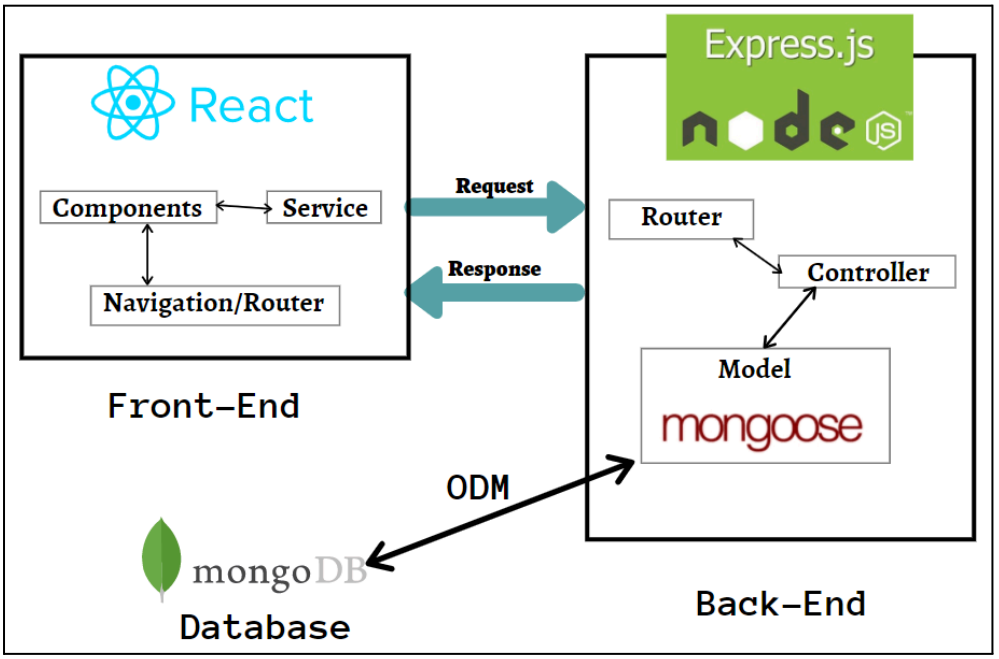


fig. 8

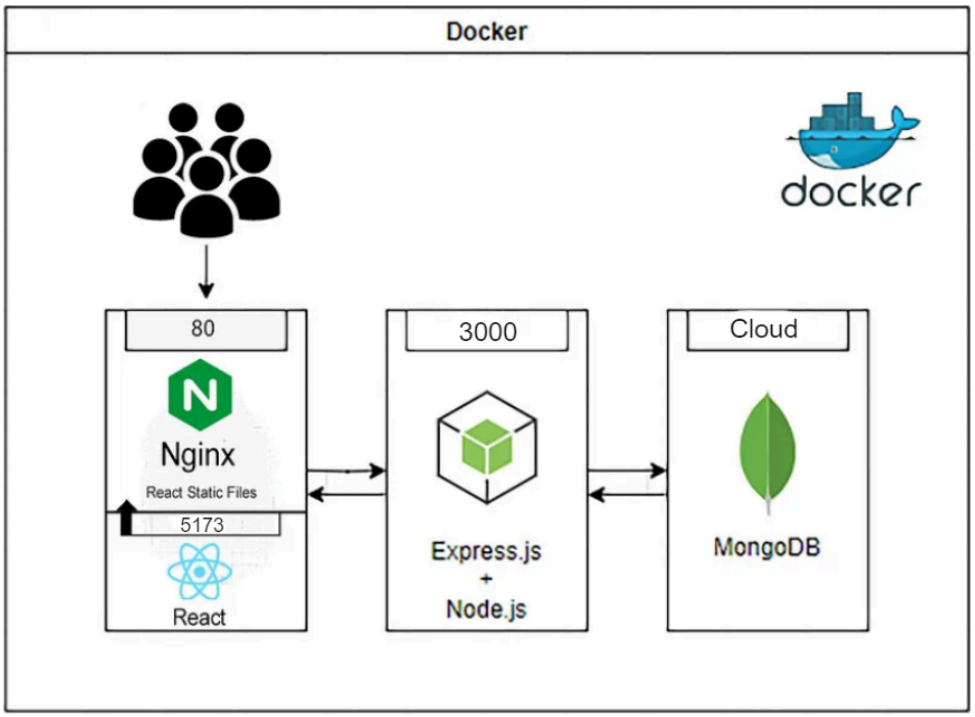


fig. 9

3.2.2 Database Schema

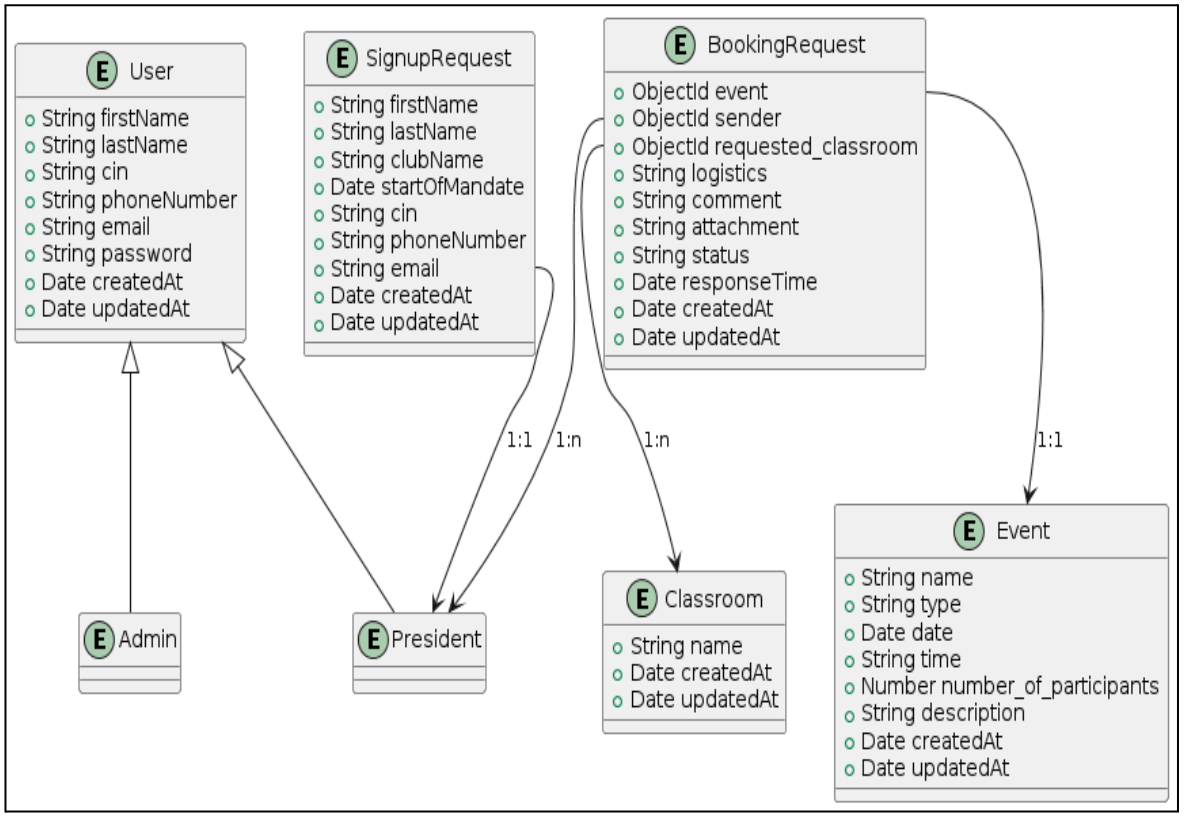


fig. 10

3.2.3 Use Case Diagram

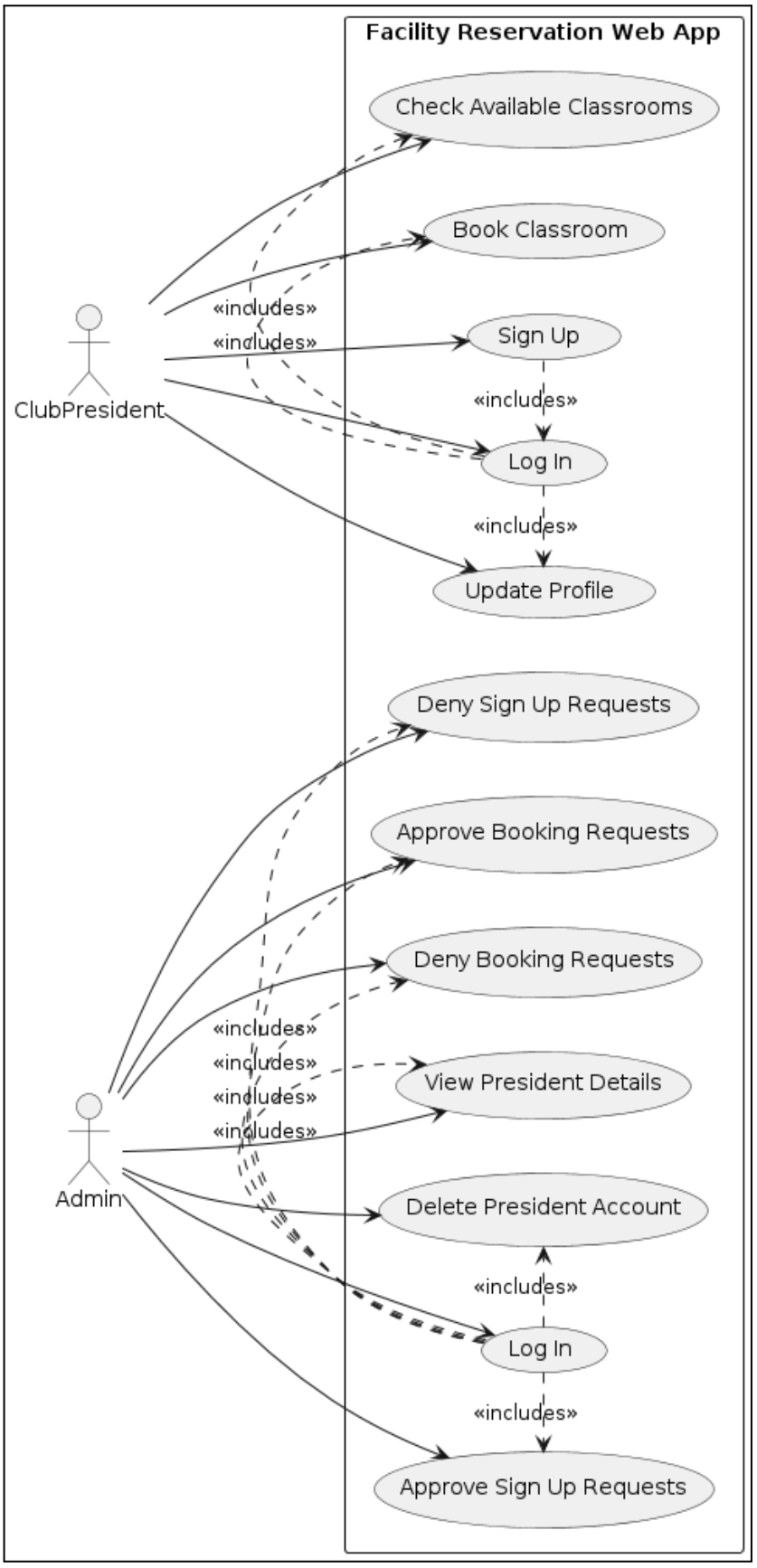


fig. 11

3.2.4 Entity Relationship Diagram (ERD)

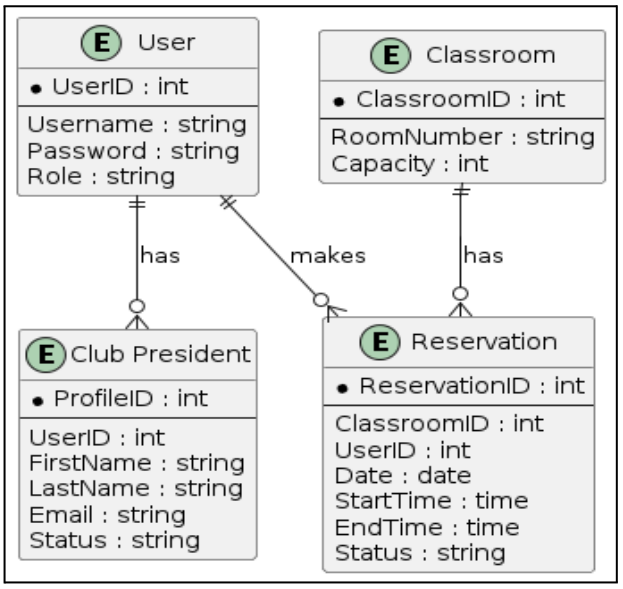


fig. 12

3.2.5 Sequence Diagram

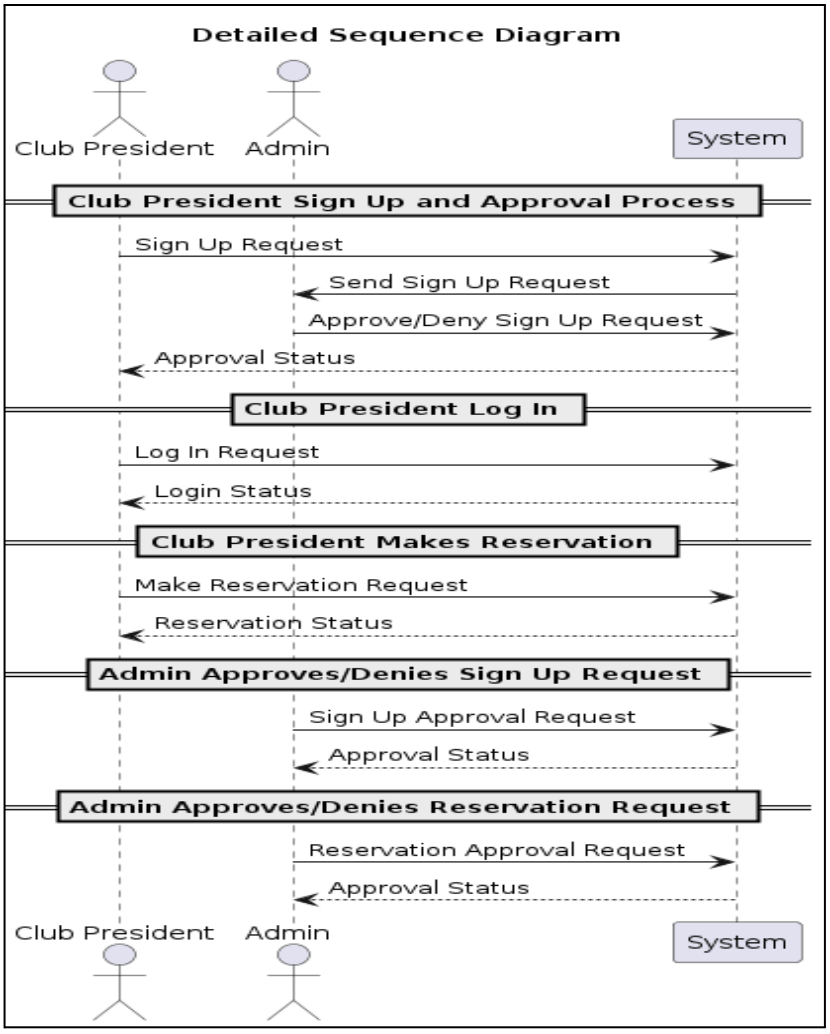


fig. 13

4. Development Phases

The development of the INSAT facility reservation web app progressed through several distinct phases, each contributing to the overall construction and refinement of the system.

4.1. Planning

4.1.1 Gathering Information

Work together with club committee members and college administration to gather all the necessary requirements for the facility reservation system. This involves holding meetings, conducting surveys, and discussing to identify the most important features and functionalities needed.

4.1.2 Project blueprint

Create a detailed project plan (based on the gathered requirements) that includes:

- Timelines: Set achievable deadlines for each part of the project.
- Choice of technologies and tools: Choose which programming language we are going to use, IDE, version control tools, etc…
- Milestones: Define key milestones to track progress and keep the project on schedule.

4.2. Requirements Analysis

4.2.1 Functional Requirements

- User Authentication: Users should be able to register, login, and logout securely.
- Account Management: Club presidents should be able to edit their accounts or delete them.
- Facility Reservation: Club presidents should be able to view available facilities, select a facility, and reserve it for a specific date and time.
- Dashboard: Club presidents should have access to a dashboard displaying their current bookings and available facilities.
- Admin Panel: Administrators should have the ability to manage facilities, user accounts, and booking requests.
- Email Notifications: Club presidents should receive email notifications for successful reservations, and cancellations.

4.2.2 Non-Functional Requirements

- Performance: The application should be responsive and provide fast loading times.
- Security: The application should implement secure authentication mechanisms and protect user data from unauthorized access.
- Scalability: The system should be able to handle a large number of concurrent users and scale horizontally as the user base grows.
- Reliability: The application should be highly available and resilient to failures, with built-in redundancy and failover mechanisms.
- Usability: The user interface should be intuitive and user-friendly, with clear navigation and error handling.

4.3. Design and Architecture

We define the web app's overall structure, including the technologies, frameworks, and methodologies we'll employ. We also design the database schema, focusing on data organization, relationships, and how data will be accessed. Additionally, we create mockups to visualize the user interface, mapping out layout and interaction pathways.

4.4 Front-End Development

- Initialize a new React.js project and configure the development environment.
- Develop components and make sure they look good on different devices, are easy to use for everyone, and match the design requirements.
- Connect front-end components with back-end APIs to enable data exchange and interaction with the server.

4.5. Back-End Development

- Create and configure the Node.js server using Express.js, defining routes, controllers, models and middleware for handling HTTP requests.
- Integrate the back-end server with MongoDB Atlas, implementing data access and manipulation logic using Mongoose.
- Create RESTful API endpoints.

4.6. Debugging

- Testing all the web application functionalities, debug, and handle errors efficiently.

4.7. Containerization and Deployment

- Containerize the application using Docker, creating Dockerfiles and Docker Compose configurations for the client side and server side.
- Test the Dockerized application locally to ensure compatibility, functionality, and performance in a containerized environment.
- Deploy the Docker containers to the AWS production environment to ensure scalability, availability, and reliability.

5. Implementation

5.1 Front-End

The key components of the INSAT facility reservation web app are:

- /**home**:Home.jsx: Home page.
- **/signup:** SignUp.jsx: Sign up page.
- **/login:** Login.jsx: Login page.
- **/account-details:** ProfileDetails.jsx: Club president can edit his account here.
- **/request/list:** RequestList.jsx: Renders all the booking request list (of all club presidents if you are logged in as admin, or just your requests if you are logged in as a club president).
- **/request/details:** RequestDetails.jsx: Display details of a specific request.
- **/users:** User.jsx: A menu displaying all registered users and all signup requests.
- **/users/president/details:** PresidentDetails.jsx: Display details of a specific club president.
- **/users/signup/request/details:** SignupRequestDetails.jsx:Display details of a specific sign-up request.
- **/dashboard/president**: DashPresident.jsx: Dashboard for the club president.
- **/dashboard/admin:** DashAdmin.jsx: Dashboard for the admin (Admin panel).
- **/bookingProcess:** BookingProcess.jsx: The booking request starts from here, and it's divided into three consecutive steps.
- ***: NotFound.jsx: Component for displaying a 404 - Not Found page.

5.2 Back-End

→ The main API endpoints are:

- **Booking Requests Routes**

Prefix: /api/booking/request

POST /step1: Initiate step 1 of the booking process.

POST /step2: Proceed to step 2 of the booking process.

POST /step3: Complete step 3 of the booking process.

GET /list: Retrieve a list of all booking requests.

GET /details: Retrieve details of a specific booking request.

PATCH /details: Approve or deny a specific booking request.

GET /count: Get the total count of all booking requests.

- **Club Presidents Routes**

Prefix: /api/users/president

GET /list: Retrieve a list of all presidents.

GET /details: Retrieve details of a specific president.

DELETE /details: Delete a specific president.

POST /add: Add a new president.

GET /count: Get the total count of all presidents.

GET /president: Show the president's dashboard.

- **Signup Requests Routes**

Prefix: /api/users/signup/request

GET /list: Retrieve a list of all signup requests.

GET /details: Retrieve details of a specific signup request.

DELETE /delete: Delete a specific signup request.

- **Authentication**

Prefix: /api/user

POST /login: User login.

POST /signup: User signup.

GET /profile: Get the profile of the logged-in user (requires authentication).

PATCH /profile: Update the profile of the logged-in user (requires authentication).

- **Admin Routes**

Prefix: /api/admin

GET /list: Retrieve a list of all admins.

GET /details: Retrieve details of a specific admin.

DELETE /details: Delete a specific admin.

POST /add: Add a new admin.

GET /count: Get the total count of all admins.

GET /admin: Show the admin's dashboard.

→ This is a summary of Common Endpoints

List Endpoints (/list):

- Retrieves lists of entities (admins, presidents, booking requests, signup requests).

Details Endpoints (/details):

- Retrieves details of a single entity.

- Allows for modification (e.g., PATCH for booking requests, DELETE for admins/presidents).

Count Endpoints (/count):

- Provides the count of respective entities.

Add Endpoints (/add):

- Adds a new entity to the database.

Authentication and Profile Management:

- /login: User login.
- /signup: User signup.
- /profile: Profile management (GET for retrieval, PATCH for update).

5.3 Database

We have created 6 models:

- President: id, firstName, lastName, cin, phoneNumber, email, password, clubName
- Admin: id, firstName, lastName, cin, phoneNumber, email, password
- classroom: id, name
- Signup_request: id, firstName, lastName, clubName, startOfMandate, cin, phoneNumber, email, submissionTime, status
- Event: id, name, type, date, time, number_of_participants, description
- Booking_request: id, event, sender, requested_classroom, logistics, comment, attachment, submissionTime, responseTime, status

5.4 Setting-up nginx

```
#nginx.conf
server {
    listen 80;
    server_name facility_reservation_app.com;

    location / {
        proxy_pass http://frontend_server:5173;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api/ {
        proxy_pass http://backend_server:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
```

```
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

5.5 Containerization

A- Client-Side Dockerfile (client/Dockerfile)

```
# Stage 1: Build client
FROM node:16 as client-build
WORKDIR /app
COPY package.json package-lock.json vite.config.js /app/
RUN npm install
COPY . /app/
RUN npm run build

# Stage 2: Serve client with Nginx
FROM nginx:latest
COPY --from=client-build /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

B- Server-Side Dockerfile (server/Dockerfile)

```
# Setup server
FROM node:16
WORKDIR /app
COPY package.json package-lock.json /app/
RUN npm install
COPY . /app/
EXPOSE 3000
CMD ["npm", "start"]
```

C- Docker Compose File (docker-compose.yml)

```
version: '3.8'

services:
  frontend:
    build:
      context: ./client
      dockerfile: Dockerfile
      target: client-build
    ports:
      - "80:80"

  backend:
    build:
      context: ./server
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
```

→ This setup allows the client-side and server-side to run in separate containers, communicating through the specified ports.

5.6 Deployment

5.6.1. Local deployment

For local deployment, we have utilized Docker Compose to orchestrate the containerized application components. By running a single command, we spin up the entire application stack, including the front-end, back-end, in a local development environment.

We use the command: `docker-compose up`

5.6.2. Production deployment (To be added in the future)

For production deployment, we will utilize AWS cloud services.

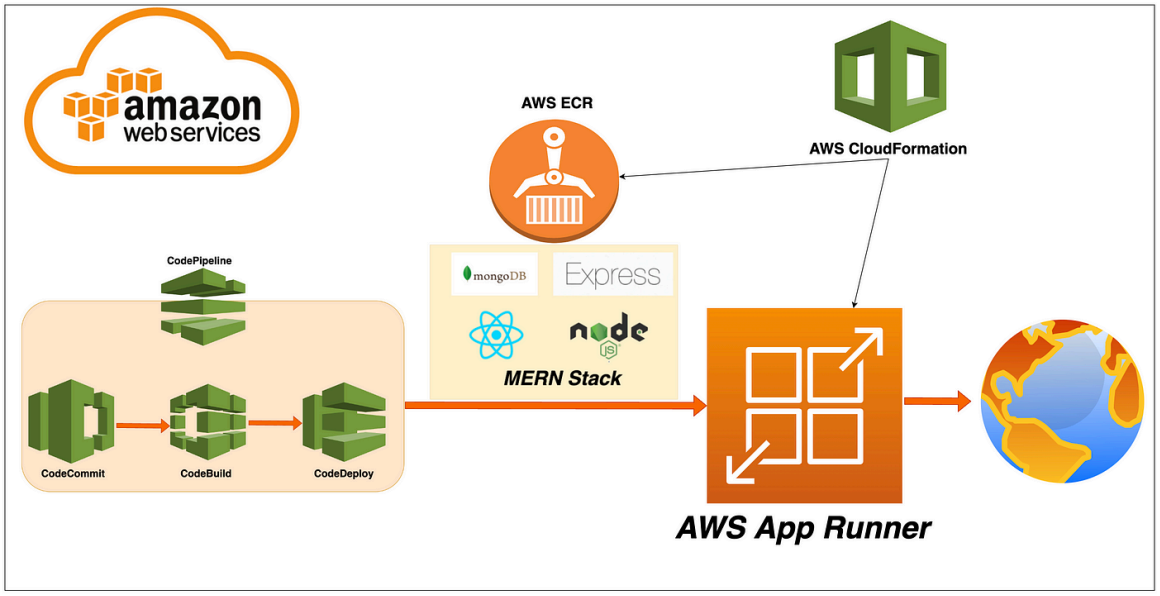


fig. 14

6. Navigating

6.1. Main features (for each user role)

A- Admin

- Access to the admin panel upon login.
- Modify the club presidents' accounts.
- See the full list of signup requests.
- Approve or deny signup requests.
- See the full list of registered club presidents.
- Check the complete list of reservation requests and their status.
- View the reservation calendar.
- Approve or deny reservation requests with optional comments.
- Modify or delete their own account.
- Generate PDF from signup request.

B- Club's President:

- Register via provided sign-up form.
- Receive approval email with generated unique password.
- Login with institutional email and pre-set password.
- Edit their account details.
- Delete their account.
- Create reservation requests.
- View complete list of their requests and their status.
- Modify requests before approval by admin.
- Delete their own requests.
- View reservation calendar.

6.2. Tour in the application

6.2.1 - Dashboards

- **President Dashboard:**

  Path: /dashboard/president?id=<president_id>

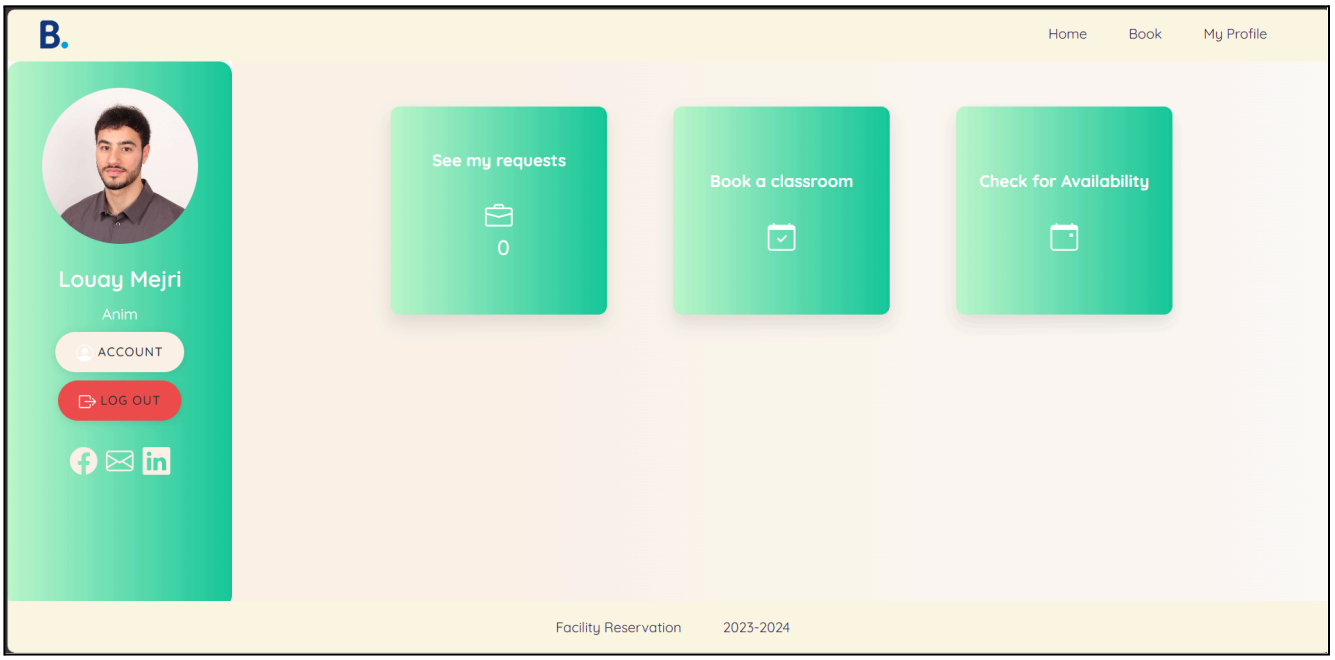


fig. 15

Account: This button leads to a page where users can view their information and make changes. Logged-in users can make reservations and manage account settings.

Log Out: This button, located on the same page as the Account button, logs the user out and redirects them to the Login page.

Requests: This button allows users to view their list of requests for using the club's facilities.

Booking: This button allows users to make reservations for the club's facilities.

See Availability: This button allows users to view the availability of the club's facilities for reservations.

- **Admin Dashboard:**

  Path: /dashboard/admin?id=<admin_id>

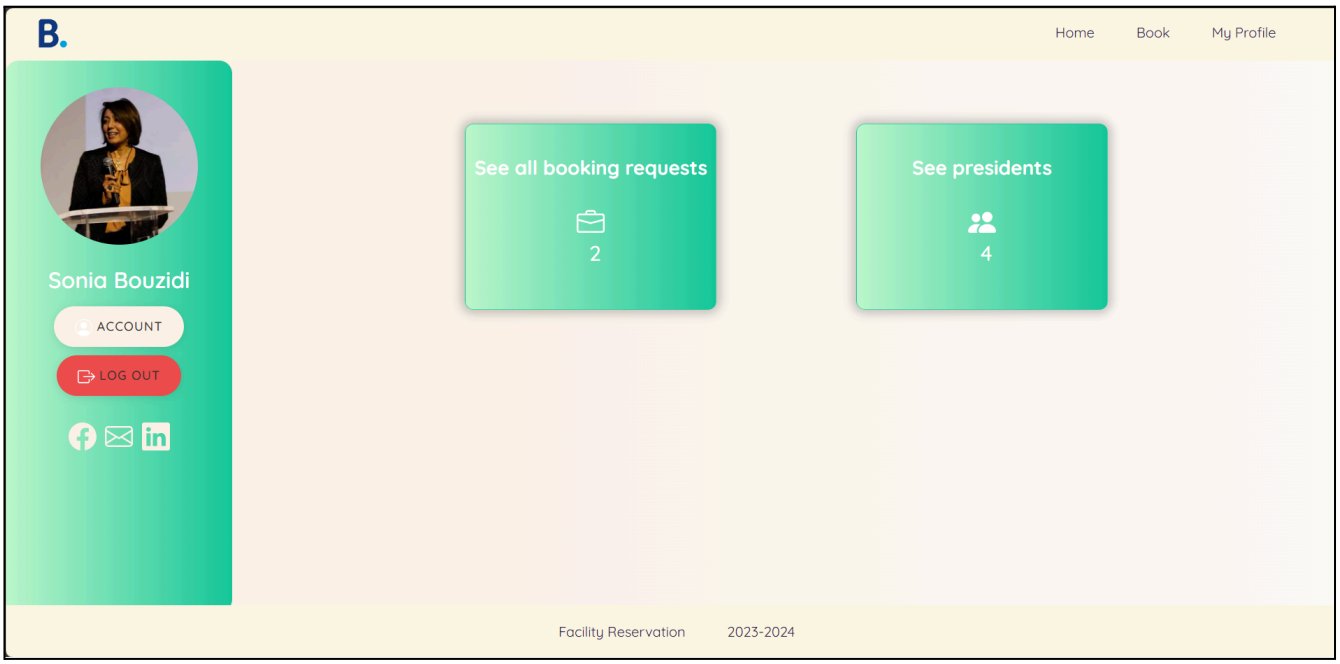


fig. 16

Account: This button leads to a page where users can view their information and make changes. Logged-in users can make reservations and manage account settings.

Log Out: This button, located on the same page as the Account button, logs the user out and redirects them to the Login page.

Requests: This button is part of the facility reservation system and allows the administrator to view, approve, or deny requests for using the club's facilities.

Presidents: This button is part of the facility reservation system and allows admins to view the list of presidents and make changes.

6.2.2 - Booking requests

- **Booking requests list (of all club presidents)**

  Path: /request/list?show=all

*This page is only accessible by admins

*The show query parameter is used to filter results, it can take 4 different values: all, approved, pending and denied.

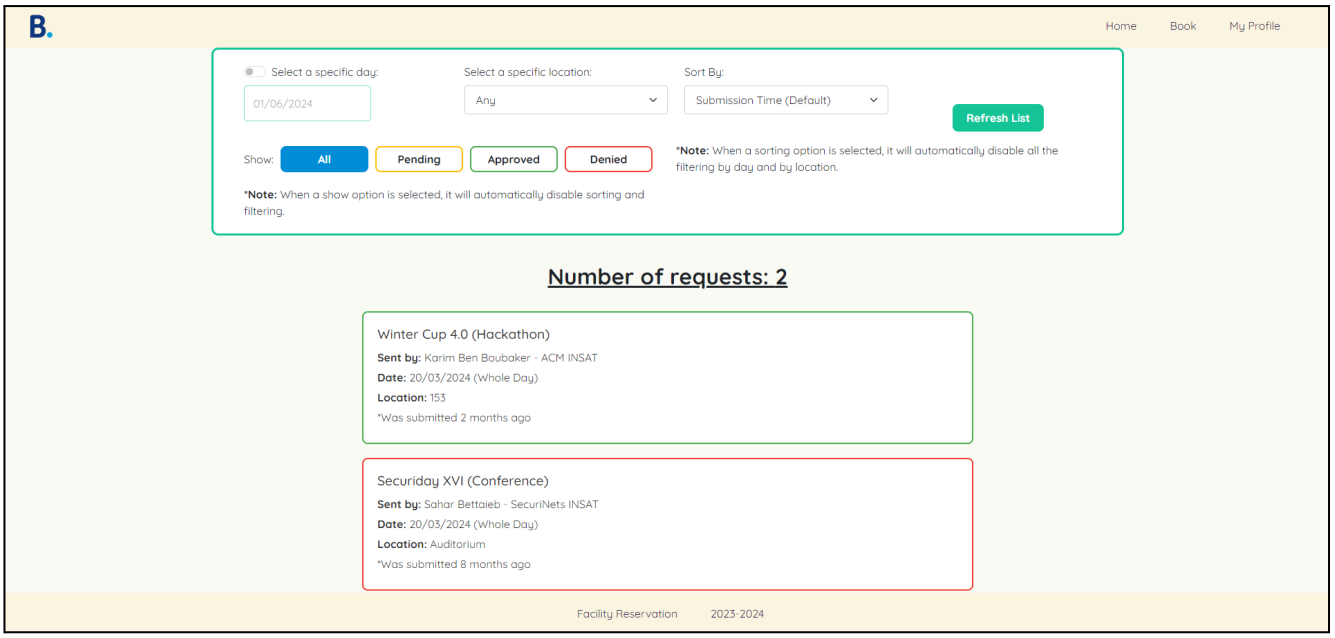


fig. 17

→ It's divided to two main components:

- The request filtering bar
- The rendered request list

→ It allows the admins to see the full booking request list of all the registered users.

→ It shows a preview of the most important details about the request.

→ When clicking on a specific request, the admin will be redirected to another page that contains all the details of that request.

- **Booking requests list (of a specific club president)**

  Path: /request/list?id=<president_id>&show=all

*This page is accessible by admins and to the concerned club president.

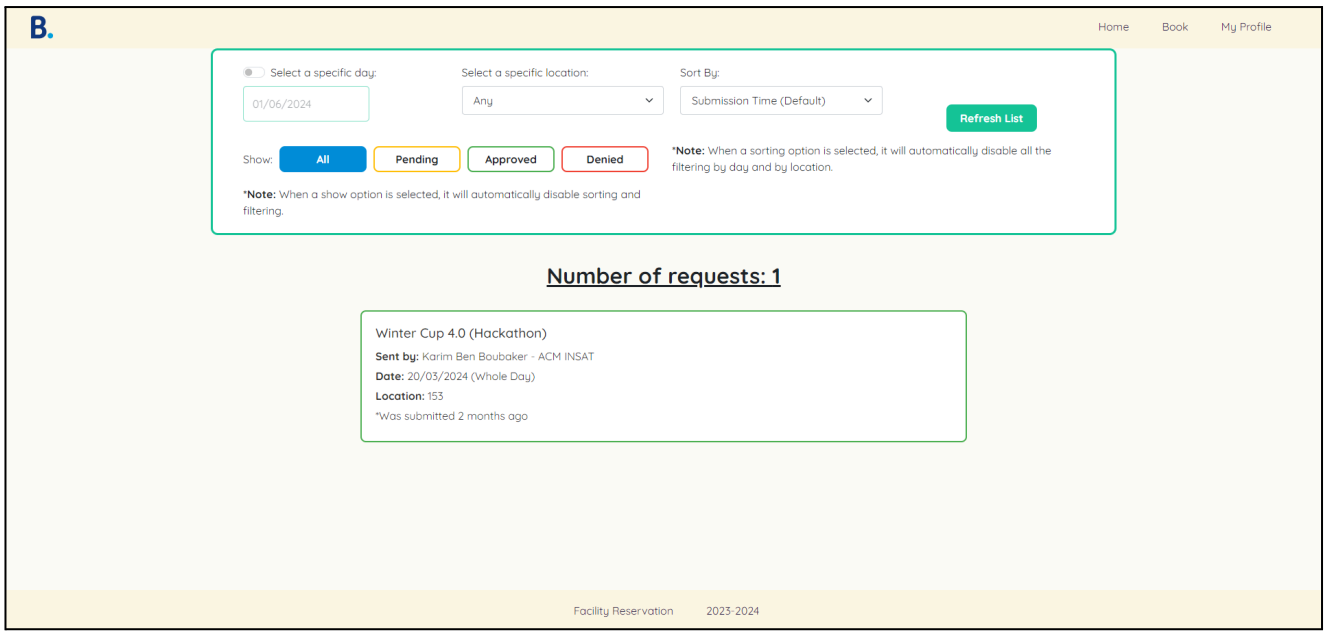


fig. 18

- **Booking request details (of a specific request)**

  Path: /request/details?id=<booking_request_id>

*If you are logged in as the club president (the owner of that request), this page will be displayed with only two buttons.

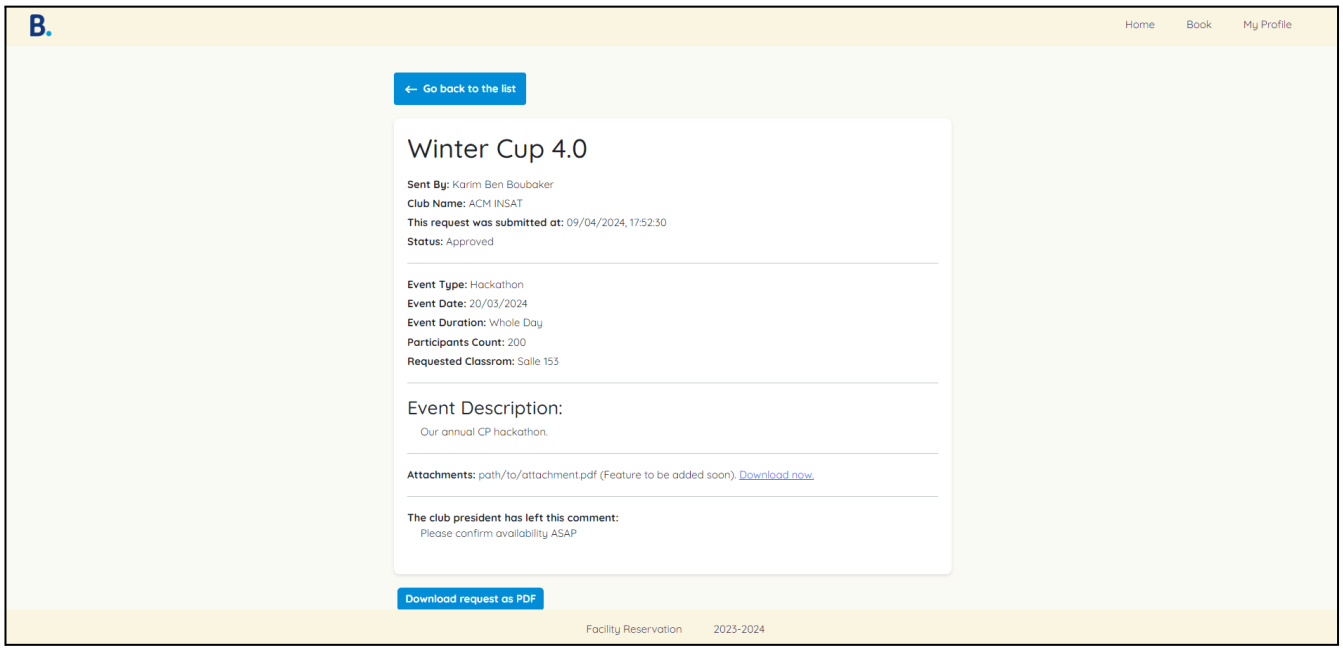
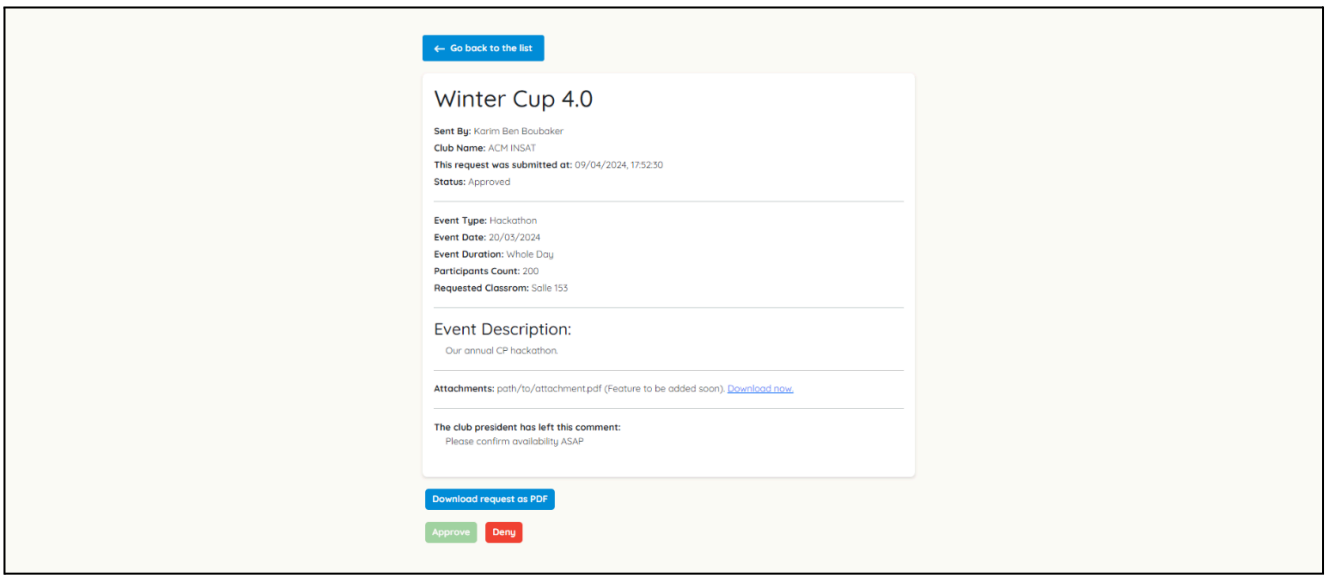


fig. 19

*If you are logged in as an admin, this page will be displayed with two additional buttons: Approve and Deny.



→ It's divided to two main components:

- An article containing all the request details.
- Control buttons: Approve / Deny / Download request as pdf.

→ It allows the admins to see all the details of a specific detail, then choose to approve it or deny it.

→ Admins can also download the details as PDF.

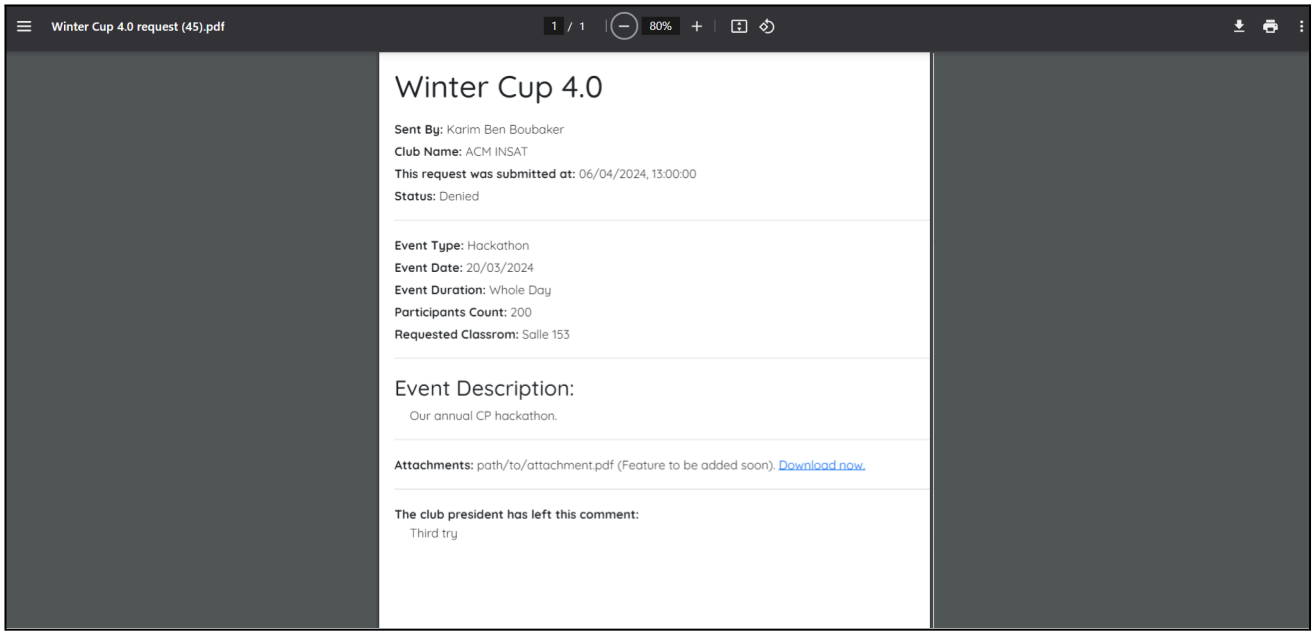


fig. 20

→ You can go back to the full request list by clicking on the “Go back the the list” button above the article.

6.2.3 - Users management

- **The registered presidents page**

  Path: /users

*This page is only accessible to admins

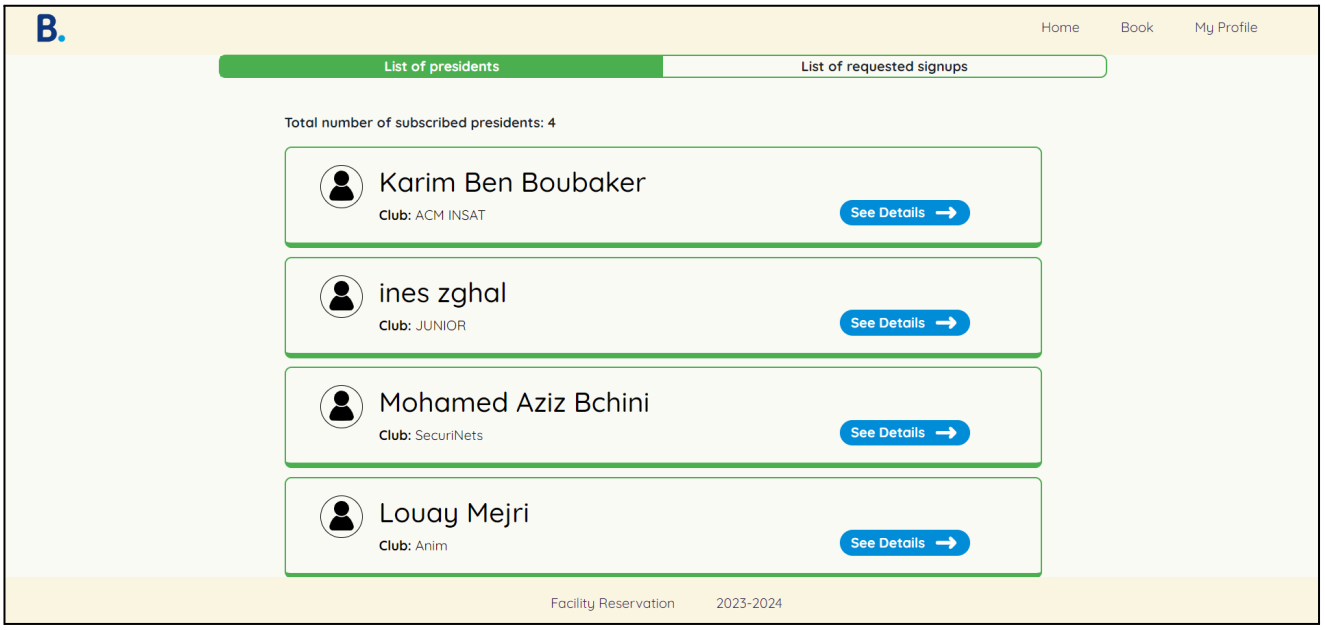


fig. 21

→ It allows the admin to see the full list of the registered presidents .

- **The club president details page**

  Path: /users/president/details?id=<president_id>

*This page is only accessible to admins.

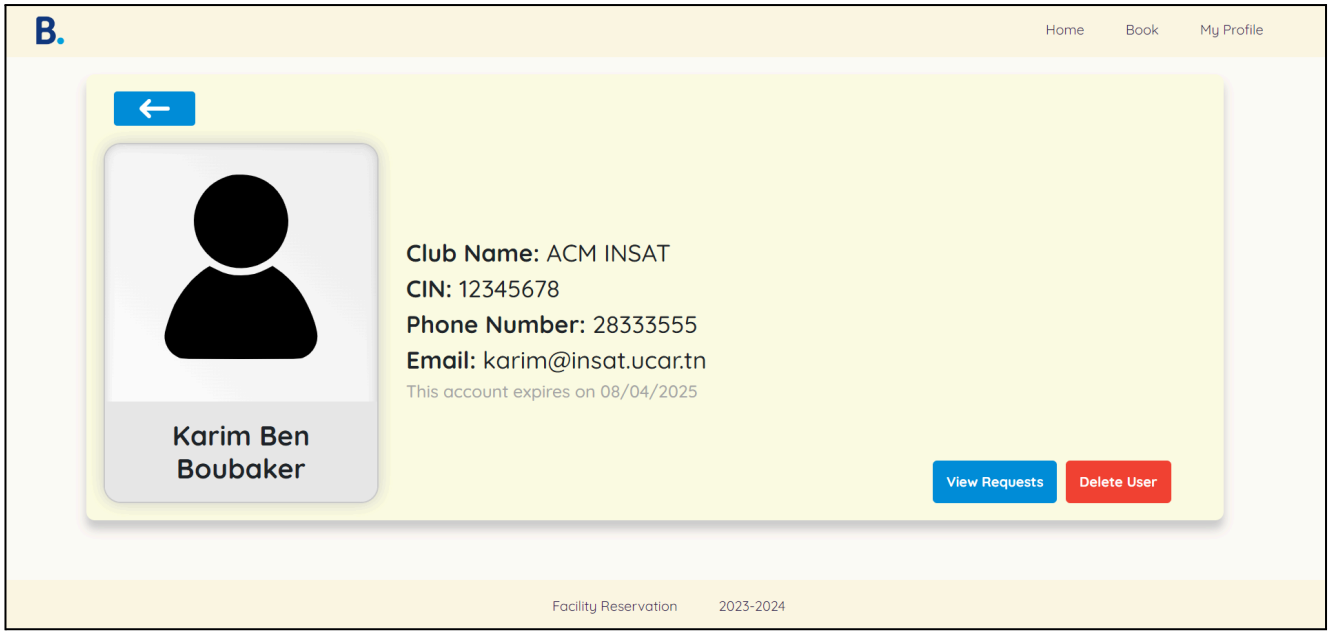


fig. 22

→ It's divided to two main components:

- President details (name, photo, email ...)
- Control buttons (View requests, delete user)

→ This page allows the admins to see all the details of a selected user, and allows them to view all requests submitted by that president, or delete the user account permanently.

→ The "Delete User" action is followed by a confirmation modal: if the deletion is confirmed then the user will be automatically notified by email.

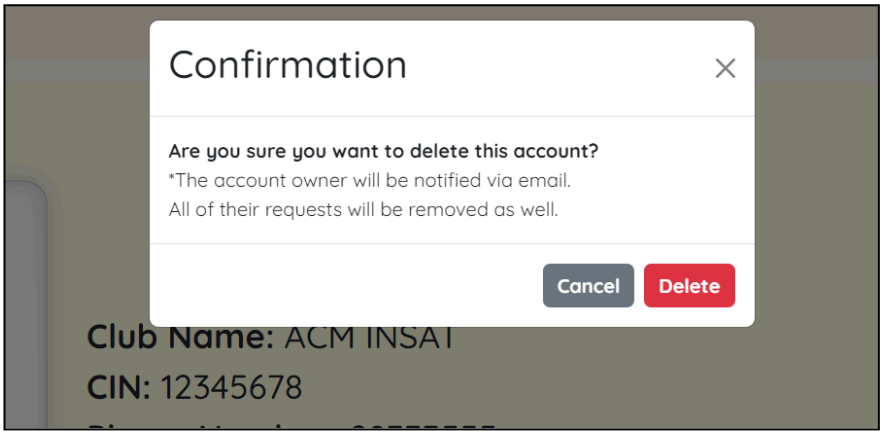


fig. 23

*When the president's account is deleted, all of his requests will be also removed from the database.

- **The signup requests page**

  Path: /users

*This page is only accessible to admins

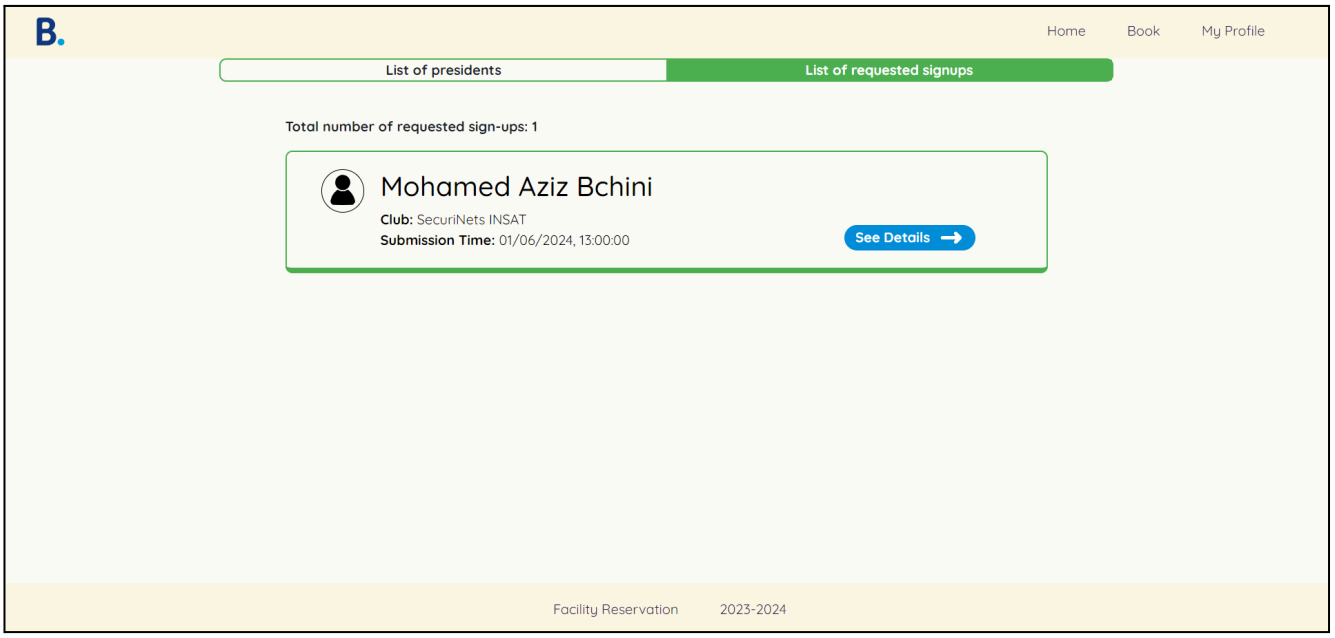


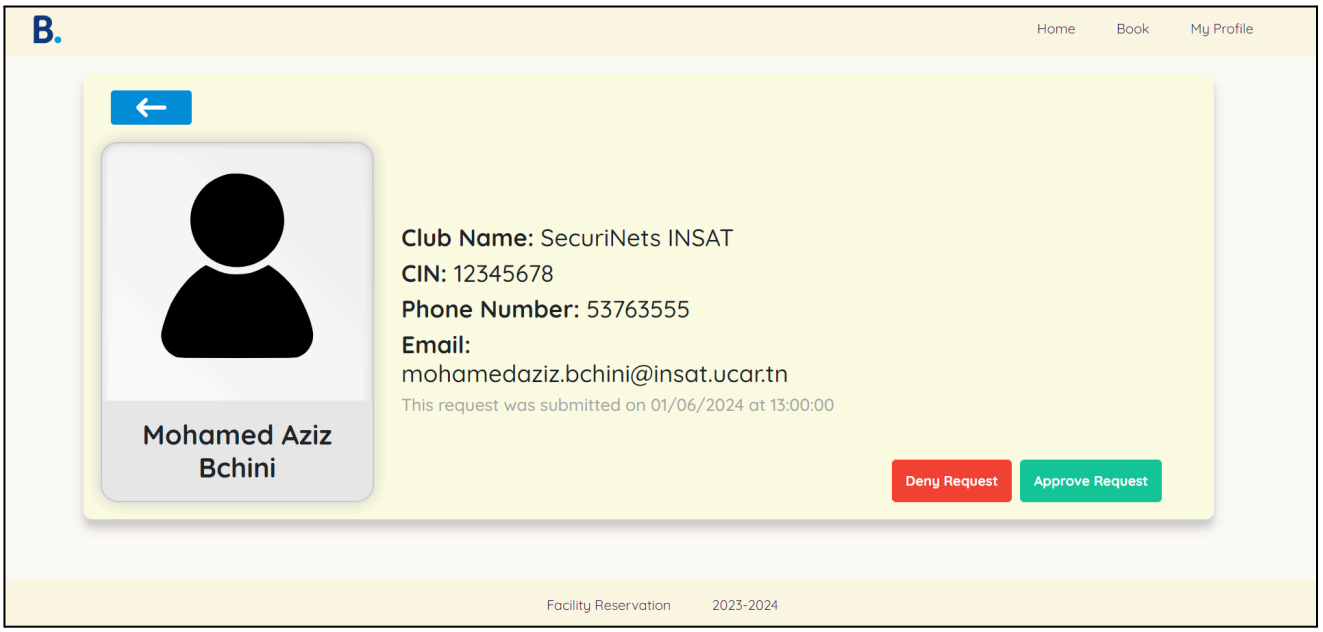
fig. 24

→ It allows the admin to see the full list of the signups requests.

- **The signup request details page**

  Path: /users/signup/request/details?id=<signup_request_id>

*This page is only accessible to admins



→ It’s divided to two main components:

- Signup request details (name, photo, email ...)
- Control buttons (Deny Request, Approve request)

→ This page allows the admins to see all the details of a selected signup request, and allows them to deny or approve that request.

→ The approve or deny actions are followed by a confirmation modal: if either action is taken, the user will be notified by email.

*When approving the request, another president account will be added and filled with the details from the signup request. The account password will be sent to the user by email, like so:

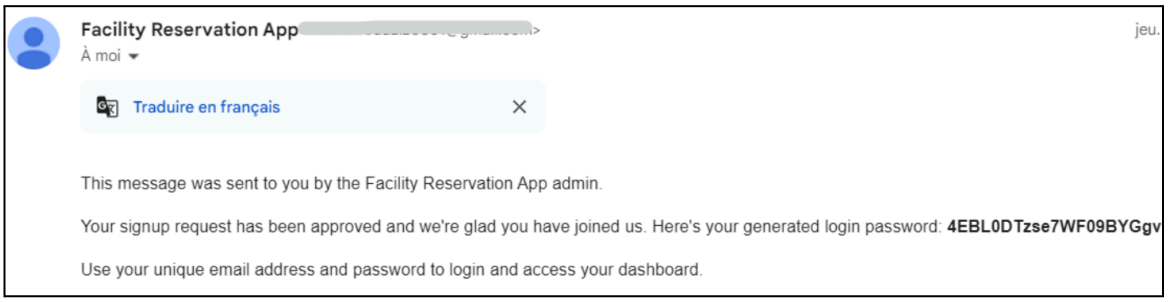


fig. 25

→ If the signup request is denied, it will be removed permanently from the database and the user will be notified by email.

- **Booking requests**

  Path : /bookingRequest

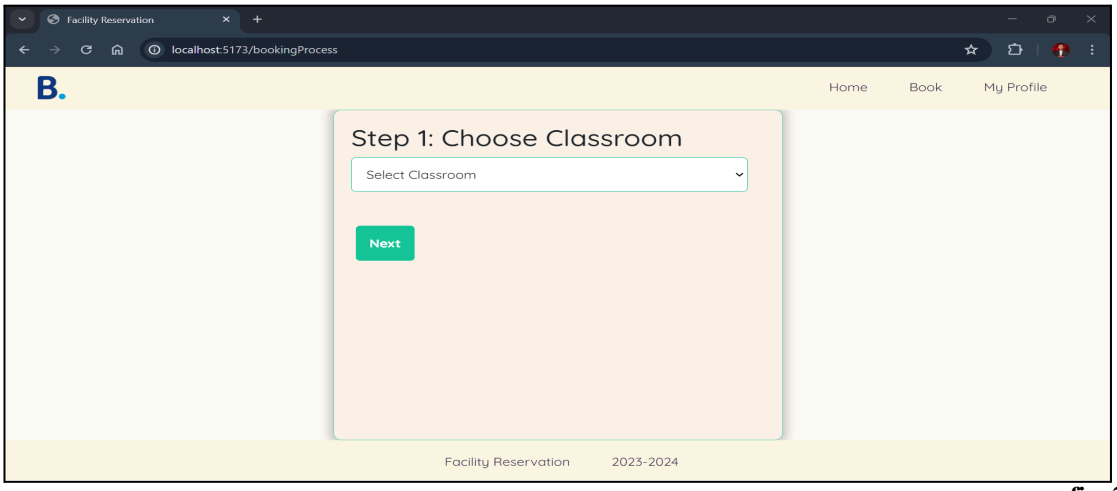


fig. 26

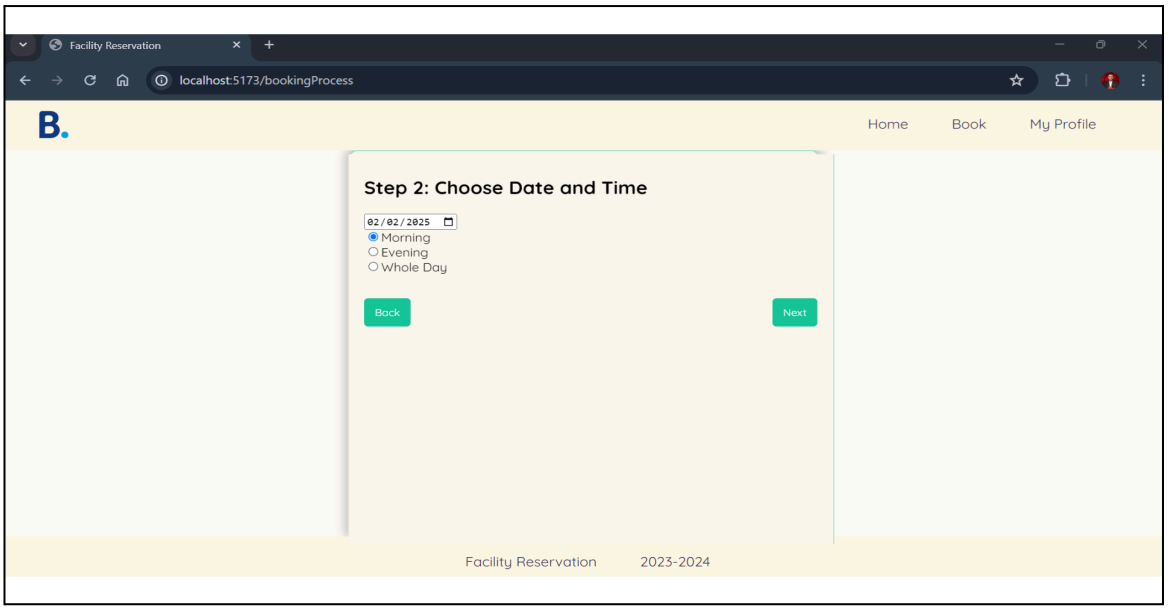


fig. 27

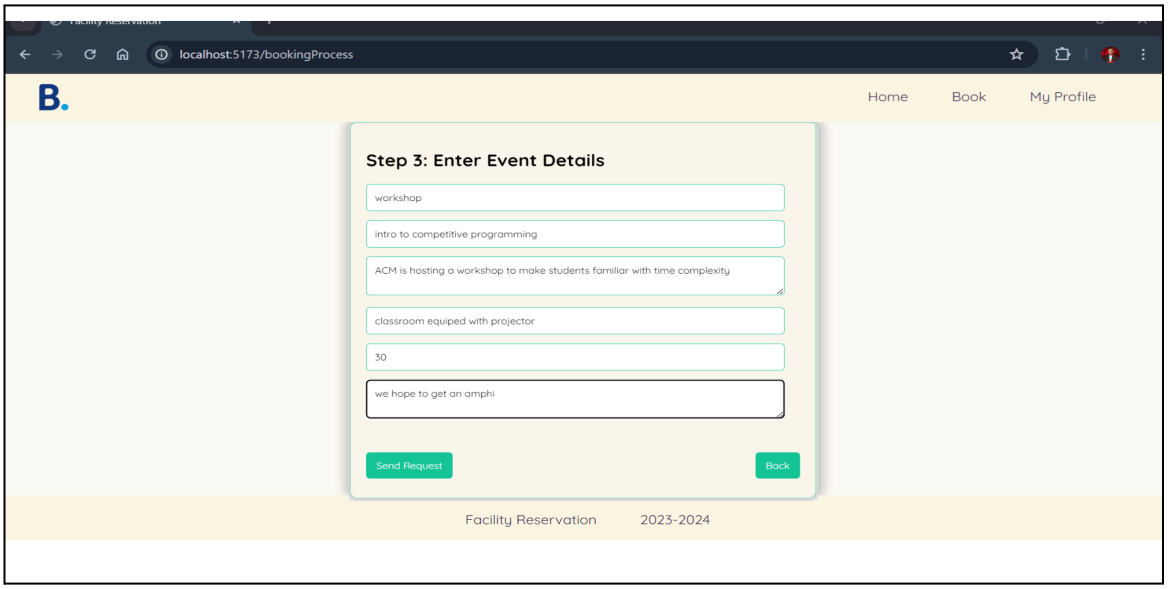


fig. 28

The president submits a classroom reservation request with event details to the campus facilities office.

- **Sign-Up**

  The sign-up page is designed for club presidents to create an account. It includes fields for entering personal information such as first name, last name, email address, phone number, Citizen Identification Number (CIN), and club name.

Additionally, users need to specify the start of their mandate. Once the form is filled out and submitted, a password will be sent to the provided email address. This ensures that users receive their login credentials securely. After receiving the password, users can log in to their account**.**

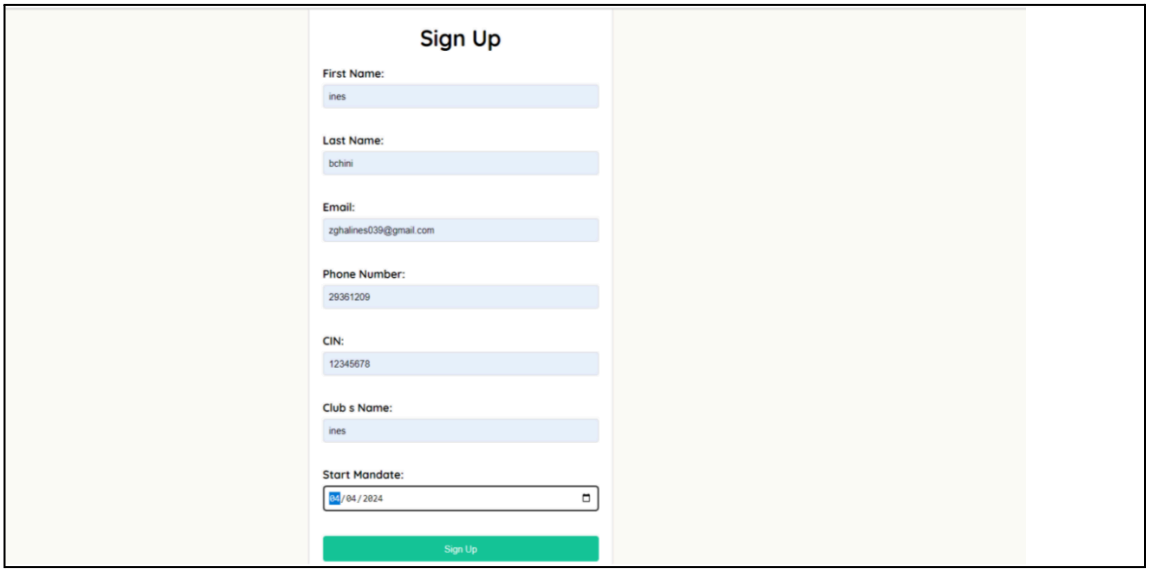


fig. 29

- **Login**

After receiving the password, users can log in to their account.The login page is designed for users to access their accounts. It contains a field for entering the user's email address and a password field with an option to show or hide the entered password.

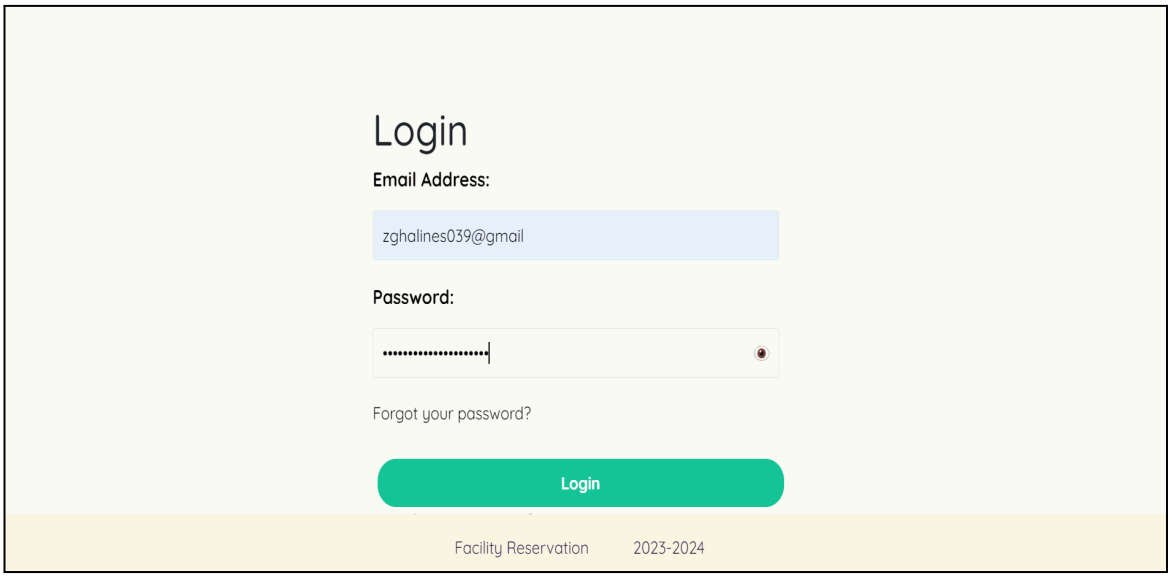


fig. 30

- **Profile Page**

  This page allows users to update their personal information (first name, last name, email, phone number, and Citizen Identification Number (CIN)). Users also have the option to upload a profile picture. After making the desired changes, they can click the "Save Changes" button to save the updates.

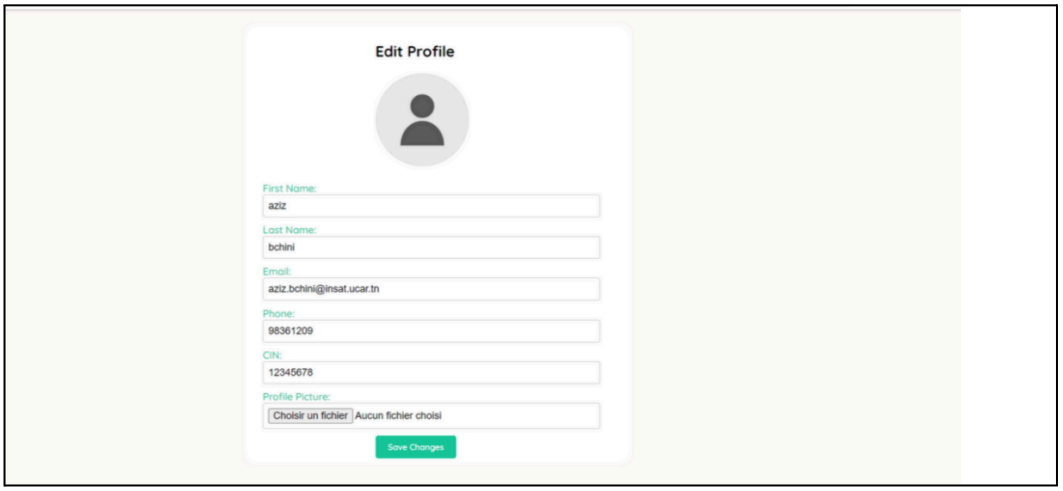


fig. 31

- **Home**

  The home page prominently displays the current date and lists all scheduled events along with their locations. For example, today's "SecuriDay" event is in the auditorium, and the "Bootcamp" event is in the conference room. Each room's section includes an image, the room name, and event details, offering a clear overview of the day's activities and their venues.

fig. 32

7. Tools used

During the development process, we have utilized a variety of tools to enhance productivity, collaboration, and efficiency.

7.1. Visual Studio Code

Our go-to IDE, providing essential features like syntax highlighting, code completion, and debugging. Its extensions made customization easy.

7.2. GitHub

Central to our workflow for version control, enabling seamless collaboration, code management, and review processes through branching and pull requests.

7.3. Postman

Facilitated API endpoint testing with its user-friendly interface, aiding request sending, response inspection, and test automation, all organized efficiently with collections and environments.

7.4. Figma

For UI mockups and prototypes, Figma's collaborative platform allowed real-time teamwork, iterative design improvements, and feedback gathering, including interactive prototyping.

7.5. Docker

Streamlined application packaging, distribution, and deployment with containerization, ensuring consistency and compatibility across diverse environments, crucial for scalable solutions.

7.6. MongoDB Atlas

Managed cloud database service simplifying database deployment, management, and scaling, with features like automated backups, monitoring, and flexible deployment options.

7.7. StarUML

Used for UML diagram creation and software design visualization, aiding effective communication and collaboration on design decisions through its feature-rich modeling tools.

7.8. Notion

An all-in-one workspace combining note-taking, project management, and collaboration features, fostering team productivity, and documentation across different projects and workflows.

7.9. NPM (Node Package Manager)

Essential for managing project dependencies, scripts, and packages, facilitating efficient project setup, maintenance, and updates with a vast registry of reusable code modules.

7.10. ESLint

ESLint is a pluggable tool for identifying and reporting patterns in JavaScript code to ensure consistency and avoid bugs. You can extend its functionality with additional rules, plugins, configurations, and parsers.

8. Future Enhancements

8.1 AWS Deployment

Implement deployment to AWS using services such as Elastic Beanstalk or ECS (Elastic Container Service) for better scalability and management of containerized applications.

8.2. Security Enhancements

Implement security measures such as encryption of sensitive data, role-based access control (RBAC), and protect against vulnerabilities and unauthorized access.

8.3 Continuous Integration and Deployment (CI/CD)

Set up CI/CD pipelines using tools like Jenkins or AWS CodePipeline to automate the build, test, and deployment processes, ensuring rapid and reliable delivery of updates to the production environment.

8.4 Scalability Improvements

Optimize the architecture for horizontal scalability by implementing microservices with Kubernetes, or serverless computing with AWS Lambda to handle increased workload efficiently.

9. Conclusion

In conclusion, the development of the facility reservation web application for INSAT has provided us with valuable technical insights and learnings throughout the entire journey. From the initial planning stages to the final deployment, we encountered and addressed various technical challenges, enhancing our skills and expertise in software development.

One significant aspect of our learning journey was the implementation of the MERN stack for building the application. This allowed us to leverage JavaScript throughout the entire development process, enhancing code consistency and reusability. We gained a deeper understanding of each component of the stack and how they interact to create a cohesive and scalable application architecture.

Additionally, the adoption of Docker for containerization proved to be a valuable learning experience. Docker enabled us to encapsulate our application components into lightweight, portable containers, ensuring consistency across different environments and simplifying the deployment process. We learned how to create Dockerfiles, manage container images, and utilize Docker Compose for orchestrating multi-container applications.

Throughout the development process, we encountered various technical challenges, such as securing API endpoints, implementing efficient front-end user interfaces, etc…
By overcoming these challenges, we deepened our understanding of software design patterns, performance optimization techniques, and security best practices.

In conclusion, the development of the facility reservation web application has been a rewarding technical journey, allowing us to apply and expand our skills in software development. We look forward to applying these learnings to future projects and continuing our growth as technical professionals.

References

MERN Stack Explained: https://www.mongodb.com/resources/languages/mern-stack

MERN vs Other Stacks: https://www.geeksforgeeks.org/mern-stack-vs-other-stacks-a-comparative-analysis/

What is Nginx? :https://medium.com/@sami.alesh/what-is-nginx-7db76b2e79f8

Docker vs VM: https://www.ubackup.com/enterprise-backup/docker-vs-vm.html

Deploy MERN Stack Application Using Docker: https://amanpathakdevops.medium.com/deploy-mern-stack-application-using-docker-docker-compose-d3eec3d9d1eb

MVC model: https://developer.mozilla.org/en-US/docs/Glossary/MVC

Dockerfile code samples: https://github.com/docker/awesome-compose

- Documentations

Nginx: https://nginx.org/en/docs/index.html

MongoDB: https://docs.mongodb.com/manual/

Express.js : https://expressjs.com/en/starter/installing.html

React JS: https://reactjs.org/docs/getting-started.html

Docker: https://docs.docker.com/

MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web

React Bootstrap: https://react-bootstrap.netlify.app/

NodeMailer: https://www.npmjs.com/package/nodemailer

How to write Markdown: https://www.markdownguide.org/getting-started/

Here's the GitHub link for the project: **Facility Reservation Platform**