instituto
superior de
engenharia
de lisboa

Department of Electronical Engineering, Telecommunications and Computers

Remote Lab

50565: Ângelo Filipe Maia Azevedo (a50565@alunos.isel.pt)
50539: António Miguel Alves (a50539@alunos.isel.pt)

Report for Project and Seminar Class
of Computer Science and Computer Engineering BSc

Advisor: Prof. Pedro Miguens Matutino

June 2025

Lisbon School of Engineering

Remote Lab

50565 Ângelo Filipe Maia Azevedo

____

50539 António Miguel Alves

____

Advisor: Prof. Pedro Miguens Matutino

____

Report for Project and Seminar Class of Computer Science and Computer Engineering
BSc

June 2025

Abstract

The design, development, implementation, and validation of digital systems require, in addition to simulators, the use of hardware for verification of their implementation in real devices. However, access to these real devices is sometimes restricted, not being available 24/7. In the current teaching paradigm where face-to-face time is reduced and remote and autonomous work is increased, it is necessary to create alternatives to the usual model.

The Remote Lab project aims to provide an online laboratory with access to remote hardware. This remote workbench consists of a web application running on an embedded system. The web application, accessed through a website, aims to provide a dashboard where users can join a laboratory. This is where users can control the remote hardware. A hierarchy system will be implemented to provide different roles, each with their own permissions relative to how users can browse the information provided by the web application.

This project will implement the infrastructure to support the configuration, manipulation and visualization of remote hardware. Based on an architecture with back-end (database and Web API) and front-end (Web App, with a dashboard).

Resumo

A conceção, desenvolvimento, implementação, e por fim a validação de sistemas digitais requerem para além dos simuladores, a utilização de hardware para uma verificação da sua concretização em dispositivos reais. No entanto, o acesso a esses dispositivos reais é por vezes restrito, não estando acessíveis 24h/7. No atual paradigma de ensino em que se reduz o tempo de contacto presencial, aumentando-se o trabalho remoto e autónomo, é necessário criar alternativas ao modelo habitual.

O projeto Remote Lab tem como objetivo fornecer um laboratório online com acesso a hardware remoto. Este laboratório consiste numa aplicação web executada num sistema embebido. A aplicação web, acedida através de um website, visa fornecer um painel de controlo onde os utilizadores podem aderir a um laboratório. Os utilizadores podem controlar o hardware remoto. Será implementado um sistema hierárquico para fornecer diferentes funções, cada uma com as suas próprias permissões relativamente à forma como os utilizadores podem navegar pela informação fornecida pela aplicação web.

Este projeto implementará a infraestrutura de suporte à configuração, manipulação e visualização de hardware remoto. Baseado numa arquitetura com back-end (base de dados e Web API) e front-end (Web App, com um dashboard).

Contents

List of Figures

Listings

Acronyms

API Aplication Programming Interface

BSc Bachelor of Science

ER Model Entity-Relationship Model

HTTP HyperText Transfer Protocol

ISA iLab Shared Architecture

MIT Massachusetts Institute of Technology

URI Uniform Resource Identifier

Chapter 1

Introduction

In recent years, the need for remote access to laboratory resources has grown significantly, driven by the expansion of online education, increased research collaboration, and the growing complexity of experimental setups. Traditional laboratories often require physical presence, which can limit accessibility and flexibility for students, researchers, and professionals. This limitation has become particularly evident in situations where geographical constraints, time restrictions, or extraordinary circumstances (such as global pandemics) prevent direct access to laboratory facilities.

The project described in this report addresses these challenges by developing a comprehensive solution for remote laboratory access that maintains the quality and integrity of hands-on experimentation while providing the flexibility of remote operation.

1.1 Objectives

Considering these emerging needs, a platform was proposed and implemented to enable secure, efficient, and user-friendly remote access to laboratory equipment and resources. The design of the platform was divided into two distinct phases. In the first phase, a database, an Aplication Programming Interface (API), and a web application were designed and implemented. This phase also encompassed the deployment architecture of the platform, including containerization, orchestration, and other essential configurations. In the second phase, the communication protocols between the platform and laboratory hardware were designed and implemented.

To design and implement a scalable platform for remote laboratory access, the following main objectives were established:

- Web API to ensure comprehensive user, laboratory, and hardware management;
- Web application that provides an intuitive and user-friendly interface;
- Secure authentication and authorization mechanisms;
- Role-based access control;

- Robust data persistence through a well-designed database system;
- Remote manipulation capabilities via terminal access to laboratory devices.

Additionally, several optional objectives were identified to enhance the system's functionality:

- Laboratory scheduling and reservation system;
- Real-time visual monitoring of laboratory hardware;

1.2 State of the Art

In recent years, numerous initiatives have emerged to provide remote access to laboratory resources, particularly in higher education and research contexts. Pioneering projects such as MIT's iLab [1] and LabShare [2] have demonstrated both the feasibility and substantial benefits of remote laboratories, enabling students and researchers to conduct experiments from anywhere in the world. These platforms typically emphasize secure access protocols, intelligent scheduling systems, and seamless integration with diverse laboratory equipment.

The existing literature underscores the critical importance of usability, scalability, and security in the design of remote laboratory systems. Key challenges identified include ensuring real-time interaction capabilities, maintaining hardware integration reliability, and providing adequate user support and training.

Several systems currently offer functionalities similar to those proposed in the Remote Lab project. The ISA, originally developed by MIT, represented an early successful implementation but is no longer operational and unavailable for public access [1]. Similarly, LabShare, another notable platform, is currently inactive [2]. Contemporary systems such as WebLab-Deusto focus on specialized domains like electronics and instrumentation, providing tailored interfaces and tools for remote experimentation in specific fields [3].

Other significant contributions to the field include the Remote Laboratory Management System (RLMS) [4] and the Labshare Sahara framework [5], which have influenced modern approaches to remote laboratory architecture and user experience design. These systems serve as valuable references for the development of the Remote Lab platform, informing critical decisions related to system architecture, user experience optimization, and hardware integration strategies.

1.3 Document Structure

This report is organized as follows: Chapter 2 presents and describes the proposed system architecture, including detailed specifications and core functionalities. Building upon the understanding of system components, the database design, web API implementation, and web application development are described in Chapters 3, 4, and 5, respectively. Chapter 6 details the project organization methodology and deployment strategies employed. Chapter 7 presents comprehensive testing procedures and validation results for the implemented system. Finally, Chapter 8 provides conclusions regarding the system's performance and outlines potential directions for future development and enhancement.

Chapter 2

Proposed Architecture

Having the objective to implement a platform to provide remote laboratories, figure 2.1 introduces a simple architecture of the system. A user, remotely, can access the platform. The server hosting the platform communicates with an external authentication service to authenticate the user. Then a user can remotely manipulate hardware.

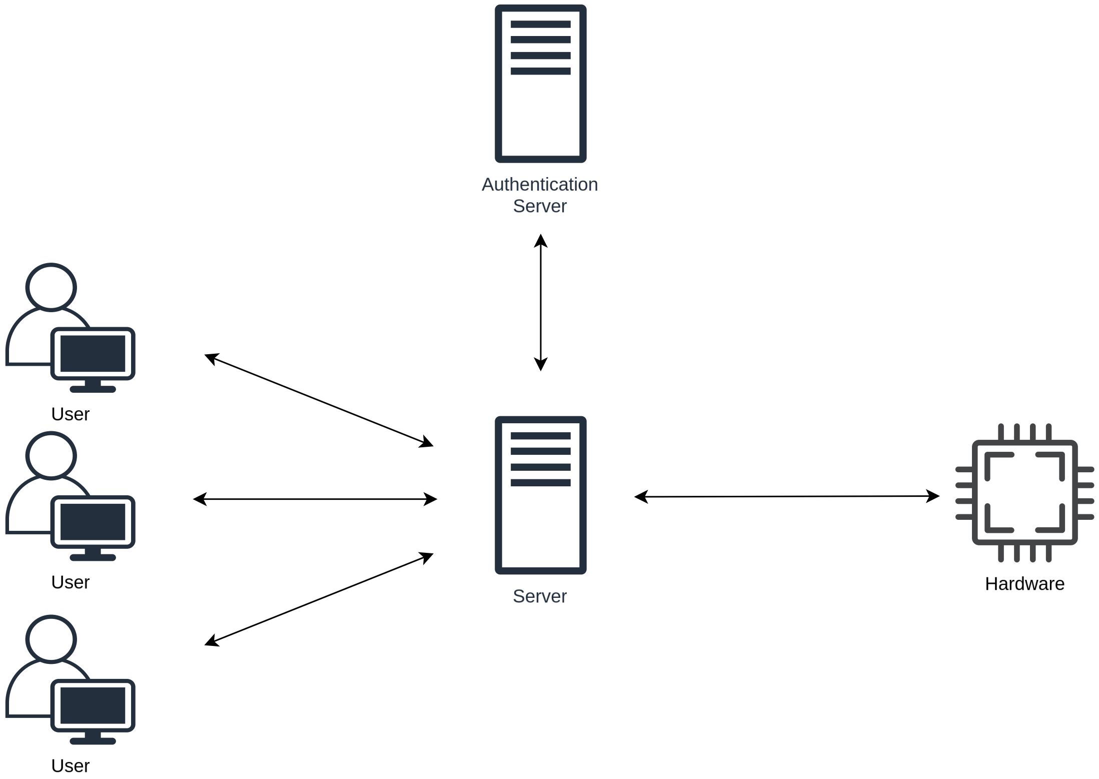

Figure 2.1: High-level Architecture

Section 2.1 describe the different types of users as well their possible interactions with the system. In section 2.2, the system functionalities are introduced and to conclude, in section Y, it is described the system architecture with a introduction to it's components.

2.1 System Users

In order to separate functions by user, it is considered three types of user:

Student can access laboratories and manipulate it's hardware.

Professor can create laboratories and assign hardware and groups, which he can also create.

Administrator can managed system users and administrative aspects.

This separation of user types, gives a reliable control in the system, for example if a user can perform certain actions or not. This is further elaborated later as one progresses through, introducing a hierarchy and RBAC (COLOCAR REF) systems.

2.2 System Functionalities

The system supports user authentication and authorization. This is further utilized in the Web API, which exposes endpoints that offers resources, their management and applies the business logic. The endpoints are offered as Uniform Resource Identifier (URI). The Web API communicates with the database which were design to meet the requirements of the system, such as holding the necessary information. The hardware abstraction layer is also in communication with the Web API, translating instructions to the specefic hardware. The Web Application makes requests to the Web API. This provides:

- Access user information;
- Laboratory access and creation;
- Group creation;
- Hardware information creation;
- User management.

2.3 System Components

As mentioned before, the system is composed with a Web Application, Web API, database and a hardware abstraction layer. Figure 2.2 illustrates a more detailed architecture of the proposed system as well the tecnologies taken intro consideration when proposing it:

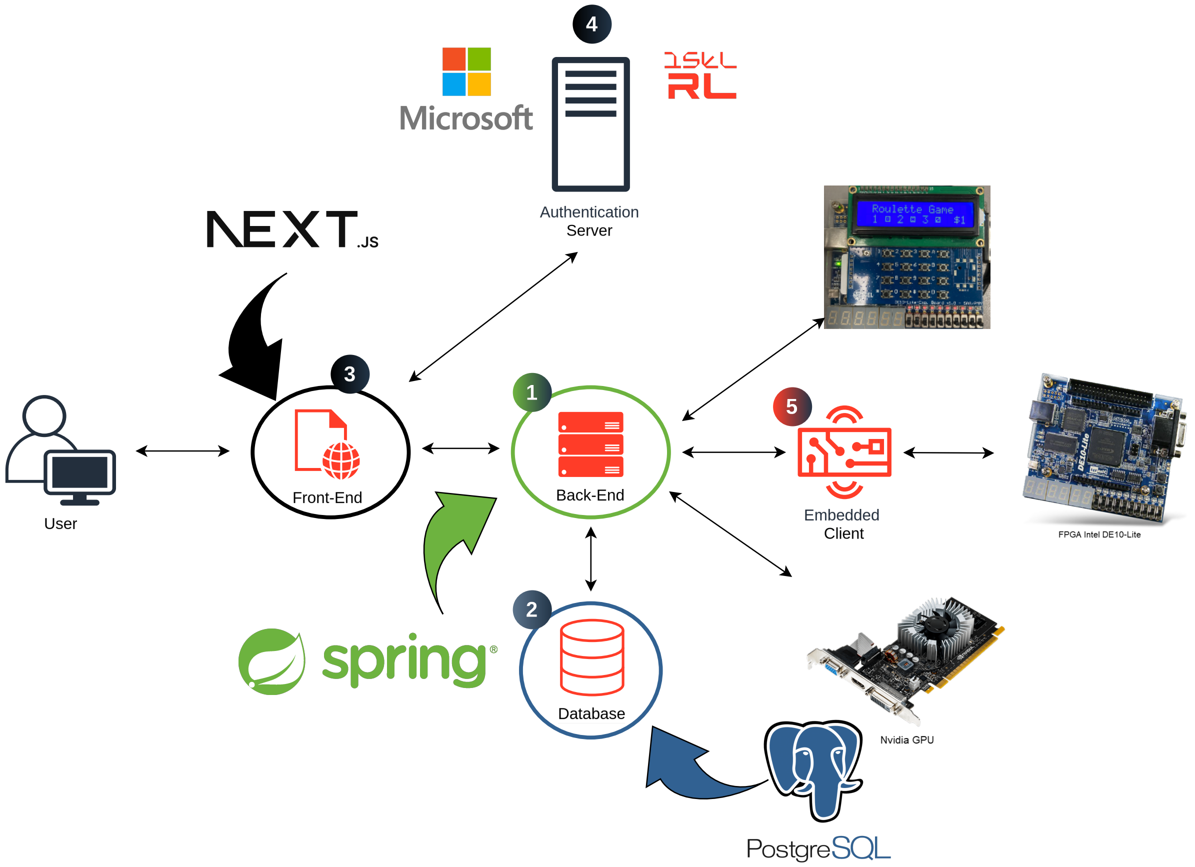

Figure 2.2: Detailed System Architecture

The server in figure 2.1, as a top level entity is zoomed into what is shown in figure 2.2, containing the Back-End (1), Database (2), Front-End (3). It also illustrates the Authentication Server (4) and the Embedded Client (5).

The Back-End (1) contains the Web API and the hardware abstraction layer. The Embedded Client (5) is illustrated and mentioned to show the possibilities of the communication with hardware. In this case the Embedded Client is communicating with a FPGA (COLOCAR REF). Depending on what hardware is associated with a sepecific laboratory, the hardware abstraction layer contained in the Back-End (1) provides different instructions and interactions. The Web API uses HyperText Transfer Protocol (HTTP) as a communication means. Spring (COLOCAR REF) was proposed as the main tecnology for the Web API and Kotlin as the programming language. Chapter 4 takes a deep dive into the Web API explaining the choice of Spring and Kotlin, architecture and implementation details.

The Database (2) was design to hold the necessary information of the system. PostgreSQL was the choice for the proposed database. Chapter 3 describes the database ER model, the reason behind PostgreSQL and implementation details.

The Front-End (3) s developed in Next.js (COLOCAR REF) and is the visual interface for a user. It makes requests to the Back-End (1) for fetching or set data. Chapter 5 takes a deep dive to the Web Application.

The authentication is made through the Web Application, using NextAuth (COLOCAR REF). Using this framework a simple communication with the Authentication Server (4) to authenticate a user. This Authentication Server (4) is provided by Microsoft OAuth (COLOCAR REF). This setup allows users to log in using their university credentials, providing a secure and familiar authentication experience. It also allows to change the OAuth provider or add one if needed.

Chapter 3

Database

The database serves as the foundational component of the system architecture. PostgreSQL was selected as the database management system due to its open-source nature and robust support for relational data models. This choice aligns with previous project implementations and provides the consistency and performance required for the system's operational needs.

This chapter presents an overview of the Entity-Relationship Model (ER Model) and critical implementation details. Complete technical documentation is provided in the accompanying appendix.

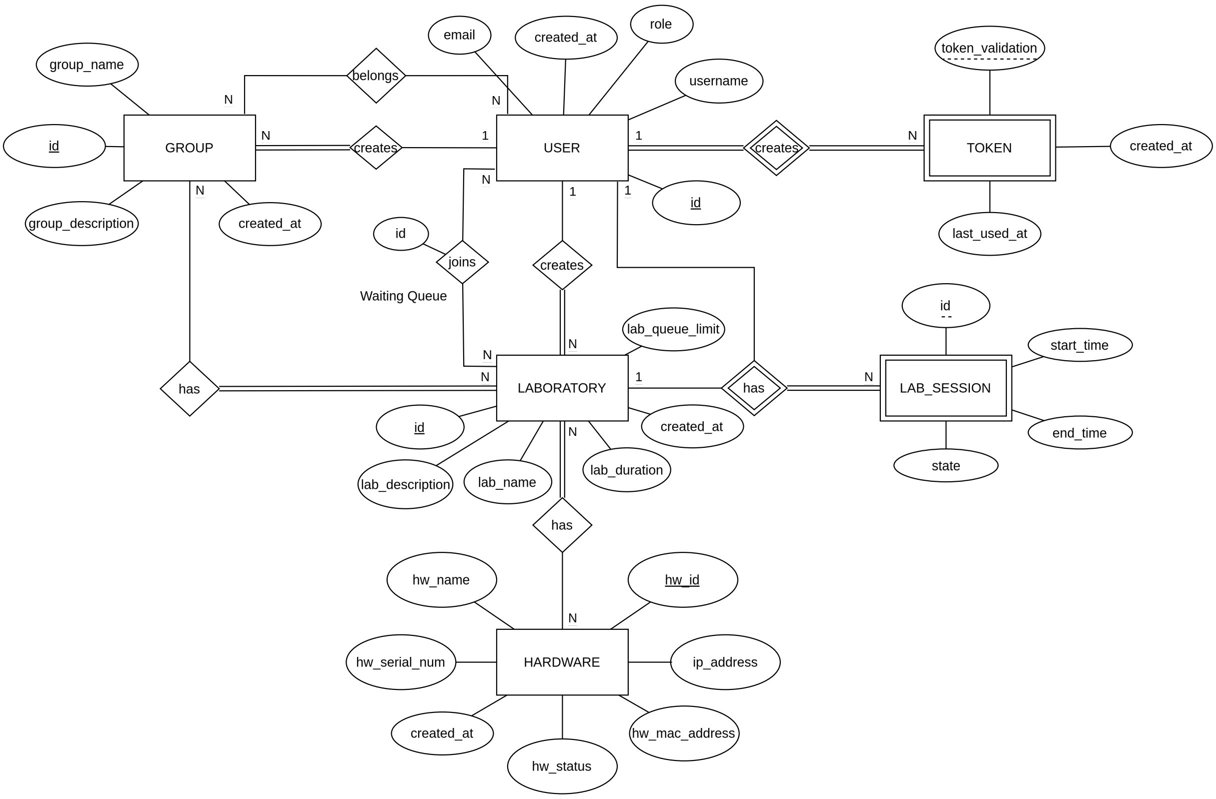

Figure 3.1: Entity-Relationship Model (ER Model)

The database design follows a normalized relational structure that supports user authentication, secure session management, and the remaining system functionalities. The ER model encompasses the core entities required for system functionality while maintaining data integrity and scalability.

Core Entities

User

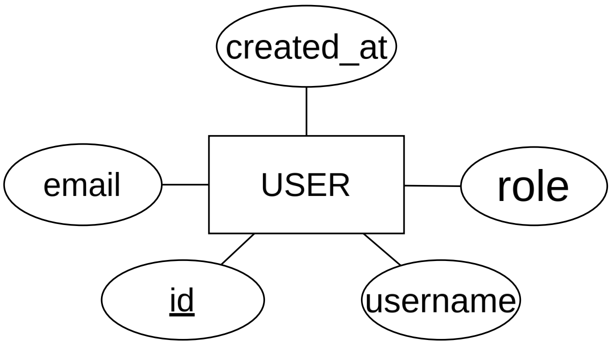

Figure 3.2: User Entity

The **User** entity represents a user in the system. The username and email attributes are provided by the authentication system. The role serves as descriminator attribute to identify whether the user is an administrator, professor or student.

Token

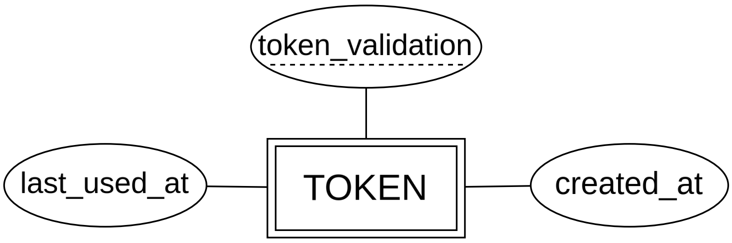

Figure 3.3: Token Entity

A user can create N tokens. The **Token** is a weak entity because it cannot be identified by its attributes alone and therefore requires a user, which is a strong entity, to be identified. Its attributes cannot uniquely identify it. A token is created by only one user.

This is a useful entity for authentication propuses. It was designed to hold a hash value in the *token_validation* attribute.

Authentication workflow:

1. Upon successful user login, a unique token is created with cryptographically secure values and stored in the database.

2. For subsequent authenticated operations, the system queries the database to verify the client-provided token against stored values.
3. Valid tokens enable secure user identification without transmitting unique identifiers.

The *last_used_at* and the *created_at* are useful for determining token expiration.

Laboratory

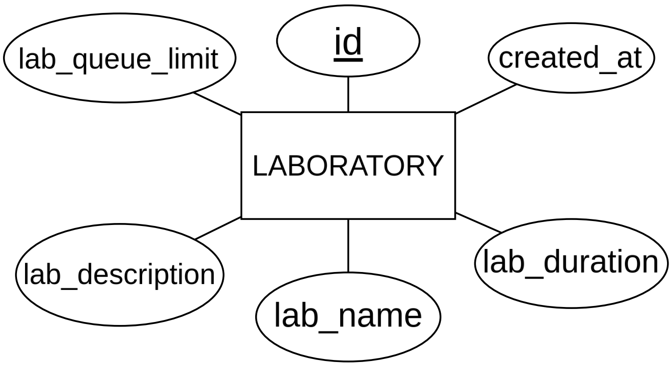

Figure 3.4: Laboratory Entity

A user, as an administrator or professor, can create N laboratories. When creating a **Laboratory**, the user can define the name (*lab_name*) and description (*lab_description*). They can also define the duration of a laboratory session (*lab_duration*) and its queue limit (*lab_queue_limit*).

Hardware

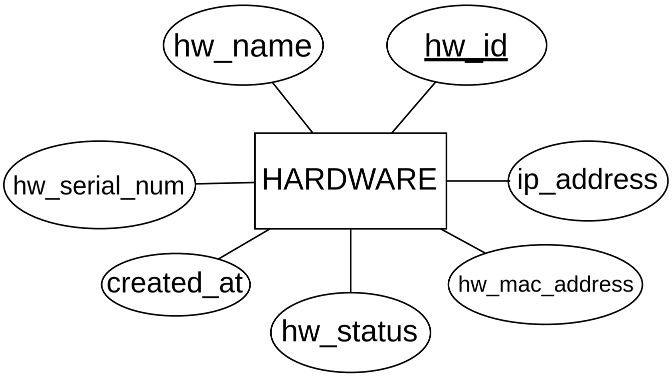

Figure 3.5: Hardware Entity

Upon successful laboratory creation, the user can associate **Hardware** to it, which must be created separately.

For the creation, it requires a name (*hw_name*), IP (*ip_address*) and MAC (*mac_address*) addresses (which can be null depending on the hardware), a status (*hw_status*) to indicate whether the hardware is under maintenance, occupied, or available, and a serial number (*hw_serial_num*) to uniquely identify the hardware. Although it has an ID, the serial number helps physically identify the hardware.

Group

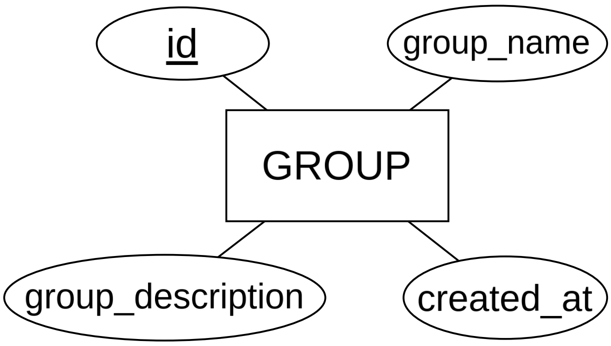

Figure 3.6: Group Entity

For a student to access a laboratory, they must be in a group that is associated with that laboratory. A professor can create a **Group** and associate users to it.

When creating a group, the user needs to name it (*group_name*) and, optionally, add a description (*group_description*) to it.

Lab Session

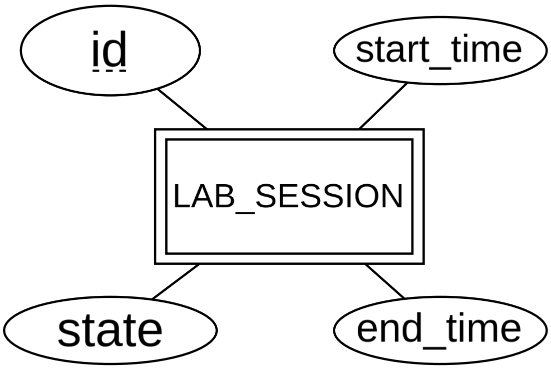

Figure 3.7: Lab Session Entity

Finally, a user can join a laboratory if they are in a group associated with it. If the laboratory is being used, the user enters a waiting queue; otherwise, a **Lab Session** is created.

Lab Session is a weak entity. It requires two strong entities to be identified: the **User** entity and the **Laboratory** entity. This is used to check whether a user is in a lab session or for statistical purposes. The *state* attribute indicates whether the session is over or still running. The *start_time* and *end_time* can be used for statistical details, such as determining how much time a user spent in a laboratory, or for future purposes, such as scheduling sessions.

3.0.1 Implementation Details

After providing an overview of the database entities and their associations, there are important details worth mentioning:

- Although PostgreSQL is being used for its functionalities, it was decided that all logic and verifications are implemented in the Web API, so that no triggers or complex constraints are implemented on the database side.

3.0.2 Conclusion

This section has provided an overview of the database architecture, implementation, and design decisions. It has also presented the ER Model of the database and described a typical user journey, explaining database interactions.

The documentation should be consulted for a comprehensive deep dive. It explains every entity, its attributes, and provides theoretical insights.

Chapter 4

Web API

The Web API provides endpoints for user management, authentication, authorization, and CRUD operations.

The API is developed with Kotlin and Spring Boot, and follows the Controller-Service-Repository pattern, which is prevalent in many Spring Boot applications. We chose this pattern because of the separation of concerns it provides and the possibilities for unit testing.

To make the codebase even easier to maintain and improve the quality of life during development, Spring Framework's Inversion of Control container (COLOCAR REFERENCIA) and the Strategy pattern principle were also used.

Spring's dependency injection is a well-known technology in Java enterprise programming. It provides an easy way to declare dependencies, since the API was mostly built following Object Oriented Programming (OOP) principles. This framework allows us to declare the necessary dependencies for each module. It also provides a BeanFactory interface for advanced configurations. Using these Spring technologies moves the object management to Spring.

The Strategy pattern allowed us to have more control over the specific implementation since it follows an interface. Every concrete implementation follows an interface, making it possible to change a class dynamically without changing the code. Spring's dependency injection works very well with this strategy design pattern. This makes unit tests much easier when the concrete implementation is not intended to be tested without changing it's code.

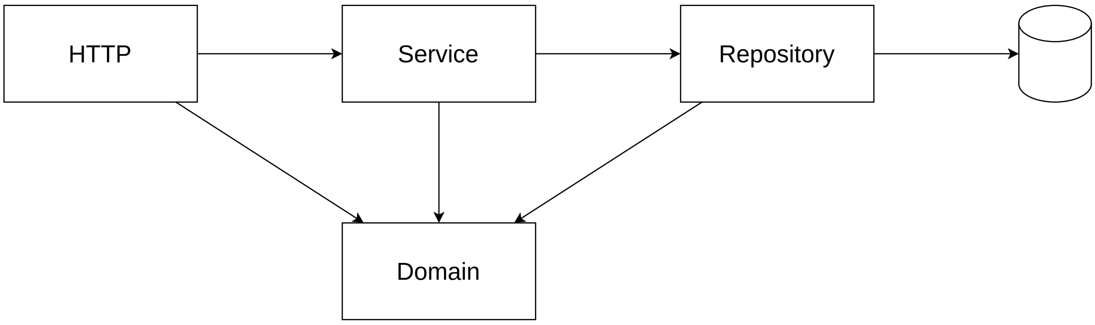

Figure 4.1: API Architecture

Figure 4.1 provides a simple overview of the implemented API. The **HTTP** module (Controller) is responsible for exposing the endpoints and handling the messages. When a request is made, the HTTP module receives the request and hands it to the **Service** module. This is where the logic and verifications are performed. Since it is necessary to fetch and save data, a **Repository** module is needed. The repository module is responsible for communicating with the database.

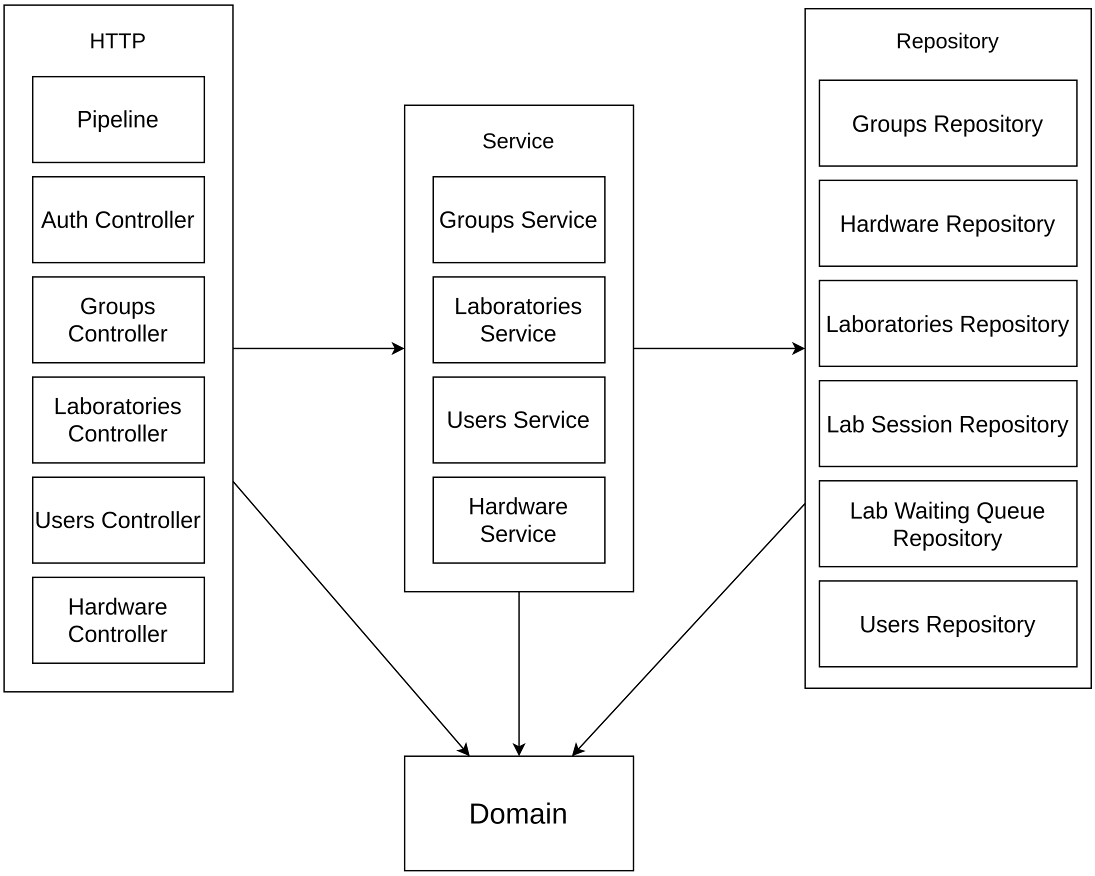

Figure 4.2: API Detailed Architecture

Figure 4.2 provides a more detailed overview of how the architecture is composed.

As explained, the HTTP module contains the controllers, each one with its functions. The pipeline contains the argument resolvers and interceptors. For the implemented system, only one argument resolver and two interceptors were implemented. The argument resolver (COLOCAR REFERENCIA) is used to provide user information to the controllers. Since the authentication method we used was token-based, this argument resolver extracts user information from the request. Every controller that has a parameter with the type *AuthenticatedUser* will be authenticated.

For the request to contain the needed information about the user, an interceptor is required. This is one of the two interceptors implemented. Every request, before reaching the controller, passes through every configured interceptor. This authentication interceptor checks if the handler parameters contain a parameter of the type *AuthenticatedUser*. If it does, the entire process of getting the token from the request, verifying it, and retrieving the user is performed. If not, normal execution continues.

Listing 4.1: Type AuthenticatedUser verification example

```
if (handler is HandlerMethod &&
     handler.methodParameters.any {
          it.parameterType == AuthenticatedUser::class.java
     }
)
```

The other interceptor is for an API key. It checks if the handler contains a custom annotation. If yes, the API key is validated; if not, an unauthorized response is sent. This interceptor is useful for the login endpoint. This login endpoint is to be performed in the Web App and is not meant to be used by end users.

The service module performs the necessary checks, using domain classes defined in the Domain module. These classes provide configurations and methods for validating certain data. Configurations in domain classes are provided by a JSON file containing domain restrictions.

Listing 4.2: Example of the group entry

```
”group”: {
     ”groupName”: {
       ”min”: 3,
       ”max”: 100,
       ”optional”: false
     },
     ”groupDescription”: {
       ”min”: 10,
       ”max”: 1000,
       ”optional”: true
     }
  }
```

This JSON file is converted to a class using Kotlin Serialization. This allows an easy way to change specific values without touching the codebase.

Every response, whether successful or an error, follows a specific format. The API documentation provides an overview of the possible responses. Error messages follow the application/problem+json format (COLOCAR REFERENCIA).

The API is expected to be public, providing full documentation (COLOCAR REFERENCIA POSTMAN) in Postman. It was decided to have the documentation in Postman because of the easy-to-use documentation builder inside the collection containing the tests for the endpoints. The API key is implemented for this reason. In future work, when the API reaches a stable version to be made public, users who want to use it will need to log in to the website and generate a token to use the API.

Chapter 5

Web Application

The web application provides a modern, user-friendly interface that enables users to access, schedule, and manage laboratory resources remotely. Built with Next.js (React), the web app is designed to be responsive and accessible from any device, ensuring a seamless experience for students, professors, and administrators.

5.0.1 Overview

The web application serves as the primary point of interaction for users, integrating with the backend services via RESTful APIs. It supports multiple user roles, each with tailored access and features according to their permissions.

The choice of Next.js as the framework for the web application was motivated by several factors. Next.js is built on top of React, a technology already taught in the course, which allowed us to leverage our existing knowledge. At the same time, Next.js simplifies development by providing built-in solutions for routing, server-side rendering, API integration, and other common tasks. This not only accelerated our development process but also allowed us to work with a modern, widely adopted framework in the industry, expanding our understanding of full-stack web development and exposing us to new possibilities and best practices.

The web application also retrieves the domain configuration in JSON format from the API. By consuming this configuration, the frontend ensures that its domain-related settings are always synchronized and up to date with those defined on the backend. This approach centralizes domain management and reduces the risk of inconsistencies between the client and server.

Figure 5.1: Web Application High-Level Architecture

Figure

5.0.2 Client-Server Logic Separation

The web application leverages Next.js to achieve a clear separation between client-side and server-side logic. All requests to the backend API are made from the server side, ensuring that sensitive operations and data exchanges are handled securely and are not exposed directly to the client. This architecture enhances security, enables better control over data flow, and allows for efficient server-side rendering and data fetching. For more details on using Next.js to perform API requests from the server, see [6].

5.0.3 Main Features

- **Authentication:** Secure login using Microsoft OAuth (NextAuth), supporting university credentials and role assignment.
- **Dashboard:** Personalized dashboard displaying relevant information, upcoming sessions, and quick access to key features.
- **Laboratory Management:** Professors and administrators can create, edit, and manage laboratory sessions, equipment, and participant lists.
- **Calendar and Scheduling:** Interactive calendar for booking and managing laboratory sessions, with real-time availability and notifications.
- **Role-Based Access:** Interface adapts to the user's role, showing only the features and data relevant to their permissions. Users with higher roles can view and interact with the platform as if they had a lower role for testing and support purposes.
- **Responsive Design:** Optimized for desktops, tablets, and mobile devices, ensuring accessibility and usability across platforms.

5.0.4 Integration and Security

The web application communicates securely with the backend via RESTful APIs, using authentication tokens to protect sensitive operations. User sessions and data are managed according to best practices, ensuring privacy and integrity. The frontend is designed to prevent unauthorized access and to provide a robust, extensible foundation for future enhancements.

Chapter 6

Project Organization and Deployment

Chapter 7

Experimental Results

Chapter 8

Conclusions

Chapter 9

Implemented Infrastructure

This chapter details the infrastructure implemented for the Remote Lab platform, covering the main components, technologies, deployment strategy, and integration between system modules.

9.1 Overview

The Remote Lab platform is designed as a modular, containerized system that enables secure and efficient remote access to laboratory equipment. The infrastructure follows a layered architecture, separating the user interface, backend logic, and hardware integration, and is built with scalability and maintainability in mind.

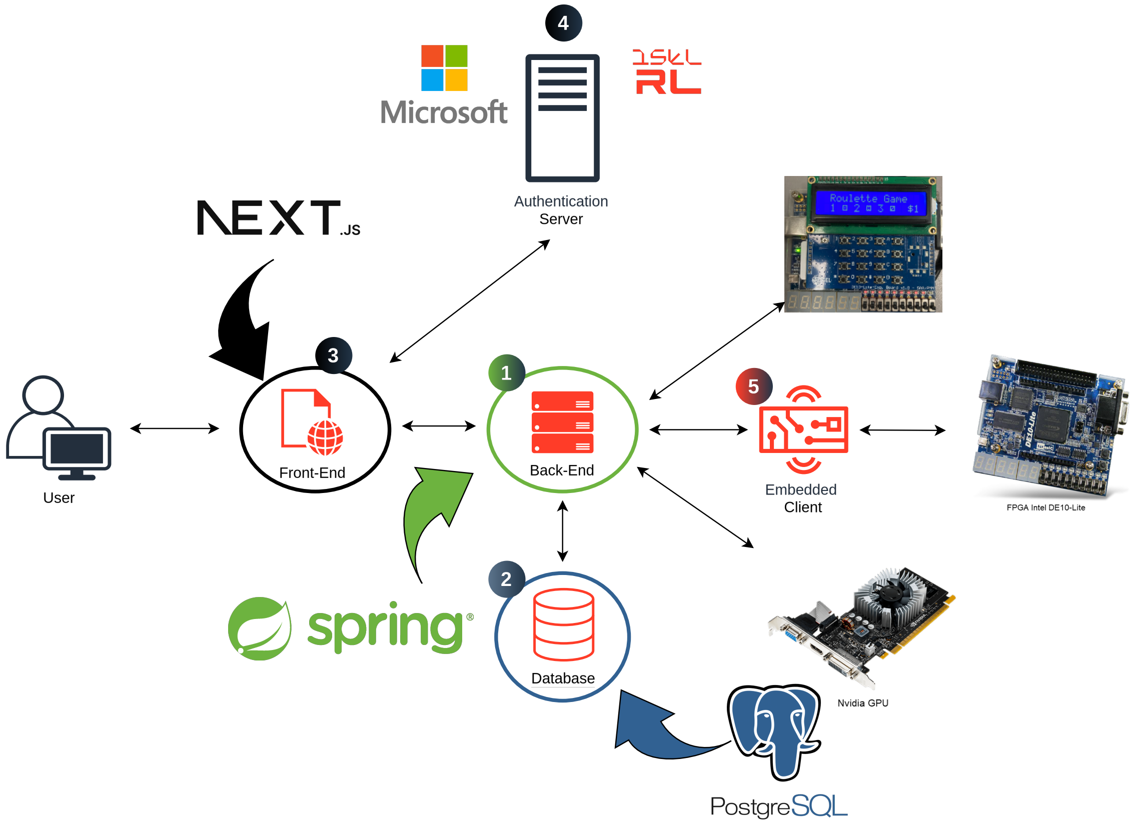

Figure 9.1: System Architecture Overview

9.2 Project Structure

The Remote Lab project is organized into several main directories, each of which is managed as a GitHub submodule. This approach allows for independent development, versioning, and access control of each core component, supporting both modularity and security. The use of submodules also facilitates collaboration among different teams and ensures that sensitive information is handled appropriately.

The main submodules of the project are:

- **api/** – Contains the backend source code, implemented in Kotlin with Spring Boot. This submodule is responsible for business logic, user management and laboratory session control.
- **db/** – Includes database scripts, supporting the persistence layer of the system.
- **docs/** – Stores project documentation, including technical reports, user guides, and architectural diagrams.
- **img/** – Stores project images, including diagrams, screenshots, and other visual representations of the system.
- **nginx/** – This directory is not a GitHub submodule but it provides Nginx configuration files for reverse proxying, load balancing, and secure access to backend services.
- **private/** – Dedicated to sensitive files and configurations, such as environment variables and secrets necessary for the secure operation of the system, and specific to our implementation choices. This submodule contains the information and configuration files required to run the project with our selected authentication (login) and database setup, reflecting the particular options chosen for our use case. It is not included directly in the main repository, ensuring that only authorized members have access to confidential information like API keys, external service credentials, and other private data essential for both production and development environments.
- **website/** – Holds the frontend web application, built with Next.js (React). This submodule provides the user interface for laboratory access, scheduling, and management.
- **wiki/** – Stores the GitHub Wiki pages, including the project documentation, deployment instructions, and other relevant information.

This modular structure, based on GitHub submodules, allows for independent development, testing, and deployment of each component, supporting both scalability and maintainability. By clearly separating concerns and leveraging best practices such as containerization and secure secret management, the project is well-positioned for collaborative development and future expansion.

9.3 Implementation Details

Key implementation decisions and details include:

- **Authentication:** Microsoft OAuth (via NextAuth) is used for secure authentication, enabling university login. The system is designed to be flexible, so other OAuth providers or even an internal login mechanism could be used if required.
- **Role-based Access Control:** The system enforces permissions based on user roles, ensuring secure and appropriate access to resources.
- **Frontend:** The frontend is built with Next.js, providing a modern, responsive interface and integrating with the backend via RESTful APIs.
- **Backend:** The backend uses JDBI for type-safe database access and is configured via environment variables for flexibility and security.
- **Hardware Abstraction:** The backend abstracts hardware-specific details, allowing for easy extension to new laboratory equipment.
- **Containerization:** All major components (frontend, backend, database) are containerized using Docker, ensuring consistent environments across development and production.
- **Orchestration:** Docker Compose is used to manage multi-container deployments, networking, and environment configuration.
- **Automation:** The `start.sh` script automates the bootstrap process, starting all necessary services with a single command.
- **Cloudflare Tunneling:** Cloudflare tunneling is used to securely expose local development environments to the internet, enabling remote access and testing without requiring direct network configuration or public IP addresses.

These choices ensure the platform is robust, extensible, and easy to deploy or develop locally.

9.3.1 Authentication

Authentication in the Remote Lab platform is implemented using Microsoft OAuth, integrated via the NextAuth library on the frontend. This setup allows users to log in using their university credentials, providing a secure and familiar authentication experience. Upon successful login, users are assigned roles (such as student, professor, or administrator) that determine their permissions within the system.

The authentication process is enforced on both the frontend and backend. The frontend uses NextAuth to handle the OAuth flow, manage user sessions, and securely store authentication tokens. The backend validates these tokens on each request to ensure that only authenticated users can access protected resources and perform actions according to their assigned roles.

The system is designed to be flexible and extensible. While Microsoft OAuth is used for university login, the authentication layer can be adapted to support other OAuth providers (such as Google or GitHub) or even an internal login mechanism if required. User information and session data are managed securely, following best practices to protect sensitive data and maintain user privacy.

9.3.2 Role-Based Access Control (RBAC)

The system implements Role-Based Access Control (RBAC) to ensure that users have access only to the resources and actions appropriate for their role. Each authenticated user is assigned a role, such as student, professor, or administrator, which determines their permissions within the platform.

Roles are enforced both on the backend and frontend. On the backend, endpoints and business logic check the user's role before allowing access to sensitive operations, such as managing laboratory sessions, accessing administrative features, or modifying user data. On the frontend, the user interface dynamically adapts to the user's role, displaying only the features and options relevant to their permissions.

The main roles in the system are:

- **Student:** Can view and book laboratory sessions, access their own session history, and interact with laboratory equipment during their reserved times. Students have access only to features relevant to their participation in laboratory activities.

- **Professor:** In addition to all student permissions, professors can create and manage laboratory sessions, view and manage student participation, and access additional data and reports related to their classes or laboratories.

- **Administrator:** Has full access to all system features, including user management, system configuration, and oversight of all laboratory sessions and resources. Administrators can manage roles, permissions, and perform maintenance or troubleshooting tasks across the platform.

In addition, the web application allows users to view and interact with the platform as if they had a lower role than their own. This feature is particularly useful for testing, support, and understanding the user experience from different perspectives. For example, an administrator or professor can switch to a student view to verify permissions, troubleshoot issues, or provide guidance, without needing to log in as a different user.

9.3.3 Cloudflare Tunneling

Cloudflare tunneling is used in the project to securely expose local development environments to the internet. This is achieved by creating a secure tunnel between the local machine and a public Cloudflare endpoint, allowing remote access to the platform for testing, demonstrations, or collaboration without requiring firewall changes or public IP addresses.

In our implementation, Cloudflare tunneling is integrated with the deployment workflow through the `start.sh` script and Docker Compose. By passing the appropriate flag (`cloudflare` or `c`) to `start.sh`, the system automatically starts a Cloudflare tunnel alongside the other services. The Docker Compose configuration includes a service for Cloudflare, which establishes the tunnel and routes external traffic securely to the local environment. This setup enables developers and stakeholders to access the development instance remotely in a secure and convenient manner, facilitating real-time testing and collaboration.

This approach provides a secure and flexible way to manage access, making it easy to introduce new roles or adjust permissions as the platform evolves. The RBAC system is central to maintaining the integrity and security of the Remote Lab environment.

9.4 Web Application

9.5 Deployment

The deployment process for the Remote Lab platform is designed to be straightforward, secure, and reproducible, leveraging modern DevOps practices and containerization technologies.

9.5.1 Containerization and Orchestration

All major components of the platform—including the backend (api), frontend (website), and database—are containerized using Docker. This ensures consistency across development, testing, and production environments. Docker Compose is used to orchestrate multi-container deployments, manage networking between services, and handle environment-specific configurations.

9.5.2 Environment Configuration and Secrets

Sensitive configuration files and environment variables required for deployment are managed in the **private/** submodule. This submodule contains the necessary secrets, such as API keys, database credentials, and authentication settings, tailored to the specific requirements of the platform. Access to this submodule is restricted to authorized team members, ensuring the security of confidential information.

9.5.3 Automation with start.sh

To further streamline the deployment process, the platform provides a `start.sh` script located at the root of the repository. This script automates the bootstrap process by orchestrating the initialization of all required services and dependencies with a single command. It handles tasks such as building Docker images, starting containers using Docker Compose, and ensuring that environment variables and configuration files are correctly loaded from the `private/` submodule.

The `start.sh` script also supports several flags to customize the deployment process, such as selecting the environment (development or production), starting only the API, enabling Cloudflare tunneling, or switching branches. These options make it easy to adapt the deployment to different scenarios with simple command-line arguments.

9.5.4 Deployment Steps

1. **Clone the Repository and Submodules:** Clone the main repository and initialize all submodules, including `private/`, to ensure all components and configurations are available.

2. **Configure Environment Variables:** Ensure that all required environment variables and secret files are present in the appropriate locations, as provided by the `private/` submodule.

3. **Build and Start Services:** Use the provided `docker-compose.yml` file to build and start all services with a single command (e.g., `docker compose up --build`).

4. **Access the Platform:** Once all containers are running, the platform can be accessed via the configured web address. Nginx is used as a reverse proxy to route traffic securely to the appropriate services.

9.5.5 Local and Production Deployment

The deployment process is designed to be nearly identical for both local development and production environments. Developers can run the entire stack locally using Docker Compose, mirroring the production setup. For production, additional considerations such as SSL certificates, domain configuration, and scaling may be applied, but the core process remains the same.

This approach ensures that deployments are reliable, repeatable, and secure, minimizing the risk of configuration drift and simplifying both initial setup and ongoing maintenance.

9.6 Technologies Used

- **Frontend:** Implemented with Next.js (React framework), providing a modern, responsive web interface for users to interact with laboratories, schedule sessions, and control hardware.

- **Backend:** Developed in Kotlin using Spring Boot, exposing RESTful APIs for user management, authentication, laboratory session control, and business logic enforcement.
- **Database:** PostgreSQL is used to persist user data, session information, access logs, and configuration settings.
- **ORM/Database Access:** JDBI is used for type-safe, modular database access in the backend.
- **Authentication:** Microsoft OAuth via NextAuth is used for user authentication, supporting multiple roles (student, professor, administrator).
- **Containerization:** Docker is used to containerize all major components (frontend, backend, database), ensuring consistent deployment across environments.
- **Orchestration:** Docker Compose manages multi-container deployment, networking, and environment configuration.

9.7 System Components

- **Web Application (Frontend):** Provides dashboards, laboratory access, real-time hardware monitoring, and session management. Built with Next.js and deployed as a Docker container.
- **API Server (Backend):** Handles authentication, authorization, laboratory and user management, and hardware abstraction. Built with Kotlin and Spring Boot, also containerized.
- **Database:** PostgreSQL instance running in a Docker container, with persistent storage volumes.
- **Hardware Abstraction Layer:** Backend modules abstract hardware-specific details, exposing unified interfaces for laboratory equipment control.

9.8 Deployment Architecture

The system is deployed using Docker Compose, which defines and manages the following services:

- **db:** PostgreSQL database container, with health checks and persistent volumes.
- **api:** Backend API container, built from the Kotlin/Spring Boot project, depending on the database service.
- **website:** Frontend container, built from the Next.js project, depending on the API service.

All services are connected via Docker networks to ensure secure and efficient communication. Environment variables and secrets are managed via `.env` files.

9.9 Build and CI/CD

- **Gradle:** Used for building and managing backend dependencies.
- **NPM:** Used for frontend dependency management and builds.
- **Dockerfiles:** Multi-stage builds are used for both backend and frontend to optimize image size and security.
- **GitHub Actions:** (If applicable) Used for continuous integration and automated builds.

9.10 Notable Implementation Details

- The backend uses JDBI for database access, configured with application-specific requirements.
- Environment variables are used to configure database connections and secrets, improving security and flexibility.
- The system supports role-based access control, with different permissions for students, professors, and administrators.
- The hardware abstraction layer allows for future extension to new types of laboratory equipment.

9.11 Summary

The implemented infrastructure leverages modern web technologies, containerization, and modular design to provide a robust, scalable, and maintainable platform for remote laboratory access.

References

[1] MIT iLab Project. ilab shared architecture (isa). https://icampus.mit.edu/projects/ilabs/index.html, 2025. Accessed: 2025-05-29.

[2] LabShare Project. Labshare: Collaborative remote laboratories. https://dbpedia.org/page/Labshare, 2025. Accessed: 2025-05-29.

[3] WebLab-Deusto. Weblab-deusto: Remote laboratory for electronics. https://www.weblab.deusto.es/, 2025. Accessed: 2025-05-29.

[4] L. F. Zapata Rivera and Larrondo Petrie. The remote laboratory management system (rlms) pattern. https://hillside.net/sugarloafplop/2018/program/papers/GroupA/12.3.pdf, 2018. Accessed: 2025-05-30.

[5] D. Lowe, T. Machet, et al. Sahara labs: Remote laboratory framework. https://github.com/sahara-labs, 2013. University of Technology Sydney. Accessed: 2025-05-30.

[6] Juan Cruz Martinez. Using next.js server actions to call external apis. https://auth0.com/blog/using-nextjs-server-actions-to-call-external-apis/, 2023. Accessed: 2025-05-29.