instituto superior de engenharia de lisboa

Department of Electronical Engineering, Telecommunications and Computers

Remote Lab

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

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

Advisor: Prof. Pedro Miguens Matutino

2 - 06 - 2025

Lisbon School of Engineering

Remote Lab

50565 Ângelo Azevedo

50539 António Alves

Advisor: Pedro 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 a virtual lab with access to remote hardware. This lab consists of a web application running on an embedded system. The web application, accessed via 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 presencial, aumentando o trabalho remoto e autónomo, é necessário criar alternativas ao modelo habitual.

O projeto Remote Lab tem como objetivo fornecer um laboratório virtual 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 dashboard onde os utilizadores podem aderir a um laboratório. É aqui que 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

Chapter 1

Introduction

1.1 Context and Motivation

In the recent years, the need for remote access to laboratory resources has grown significantly, driven by the expansion of online education, research collaboration, and the increasing complexity of experimental setups. Traditional laboratories often require physical presence, which can limit accessibility and flexibility for students, researchers, and professionals. The **Remote Lab** project aims to address these challenges by providing a platform that enables secure, efficient, and user-friendly remote access to laboratory equipment and resources.

1.2 Objectives

The main objectives of the Remote Lab project are:

- To design and implement a scalable platform for remote laboratory access.
- To ensure secure authentication and authorization for different user roles.
- To provide an intuitive user interface for managing and scheduling laboratory sessions.
- To support integration with various types of laboratory hardware.

1.3 Scope

This project focuses on the development of the core platform, including backend services, user management, and basic hardware integration. Advanced features such as real-time data analytics, support for a wide range of laboratory devices, and extensive reporting capabilities are considered out of scope for the current phase.

1.4 Methodology

The project follows a modular and iterative development approach, leveraging modern software engineering practices. The backend is implemented using Kotlin and follows a layered architecture, while the frontend is developed with Next.js to provide a responsive and accessible user experience.

1.5 Structure of the Document

The remainder of this report is organized as follows:

- **Chapter 2:** Related Work – Overview of existing solutions and technologies.
- **Chapter 3:** System Architecture – Description of the overall system design.
- **Chapter 4:** Implementation – Details of the main components and their interactions.
- **Chapter 5:** Evaluation – Assessment of the system's performance and usability.
- **Chapter 6:** Conclusions and Future Work – Summary of achievements and directions for future development.

Chapter 2

Placement

This chapter is organized into two sections, where we describe related work and some systems similar to the one developed in this project.

2.1 Related Work

In recent years, several initiatives have emerged to provide remote access to laboratory resources, especially in the context of higher education and research. Projects such as MIT's iLab and LabShare have demonstrated the feasibility and benefits of remote laboratories, enabling students and researchers to conduct experiments from anywhere in the world. These platforms typically focus on providing secure access, scheduling, and integration with a variety of laboratory equipment. The literature highlights the importance of usability, scalability, and security in the design of such systems, as well as the challenges associated with real-time interaction and hardware integration.

2.2 Similar Systems

There are several systems that offer functionalities similar to those of the Remote Lab project. For example, the iLab Shared Architecture (ISA) provides a framework for sharing laboratory equipment over the internet, supporting both batch and interactive experiments. LabShare is another notable example, offering a collaborative platform for remote experimentation and resource sharing among institutions. Other systems, such as WebLab-Deusto and VISIR, focus on specific domains like electronics and instrumentation, providing specialized interfaces and tools for remote experimentation. These systems serve as valuable references for the development of the Remote Lab platform, informing decisions related to architecture, user experience, and integration with laboratory hardware.

Chapter 3

Proposed Architecture

This chapter presents the proposed architecture for the Remote Lab platform, detailing its main components, their interactions, and the rationale behind the architectural choices.

3.1 System Overview

The Remote Lab platform is designed as a modular and scalable system, enabling secure and efficient remote access to laboratory equipment. The architecture follows a layered approach, separating concerns between the user interface, application logic, and hardware integration. This separation facilitates maintainability, extensibility, and the integration of new features or laboratory devices.

3.2 Main Components

The architecture consists of the following main components:

- **Frontend:** A web-based user interface developed with Next.js, providing users with access to laboratory resources, session scheduling, and experiment monitoring.

- **Backend:** Implemented in Kotlin, the backend exposes RESTful APIs for user management, authentication, authorization, and laboratory session control. It also handles business logic and enforces security policies.

- **Hardware Abstraction Layer:** This layer manages communication with laboratory equipment, abstracting hardware-specific details and providing a unified interface for the backend.

- **Database:** Stores user data, session information, access logs, and configuration settings. The database ensures data consistency and supports auditing requirements.

- **Authentication and Authorization:** Ensures secure access to the platform, supporting multiple user roles (e.g., students, professors, administrators) with different permissions.

3.3 Component Interactions

The components interact as follows:

- Users interact with the frontend to authenticate, schedule sessions, and access laboratory resources.
- The frontend communicates with the backend via secure API calls.
- The backend processes requests, applies business logic, and interacts with the database and hardware abstraction layer as needed.
- The hardware abstraction layer translates backend commands into device-specific instructions, enabling remote control of laboratory equipment.

3.4 Architecture Diagram

Figure 3.1 illustrates the high-level architecture of the Remote Lab platform.

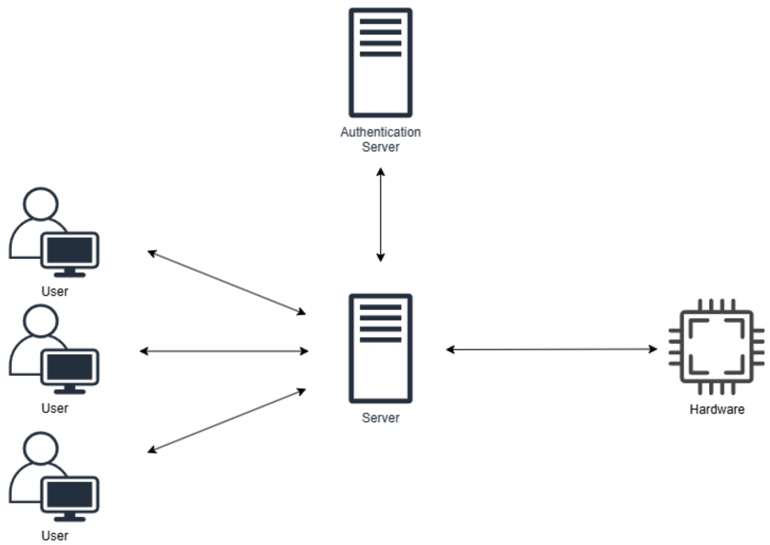

Figure 3.1: High-level architecture of the Remote Lab platform.

3.5 Design Rationale

The architectural choices were guided by the need for scalability, security, and ease of integration with diverse laboratory equipment. The use of a layered architecture and standardized interfaces ensures that the platform can evolve to meet future requirements and support additional functionalities.

Chapter 4

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.

4.1 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.

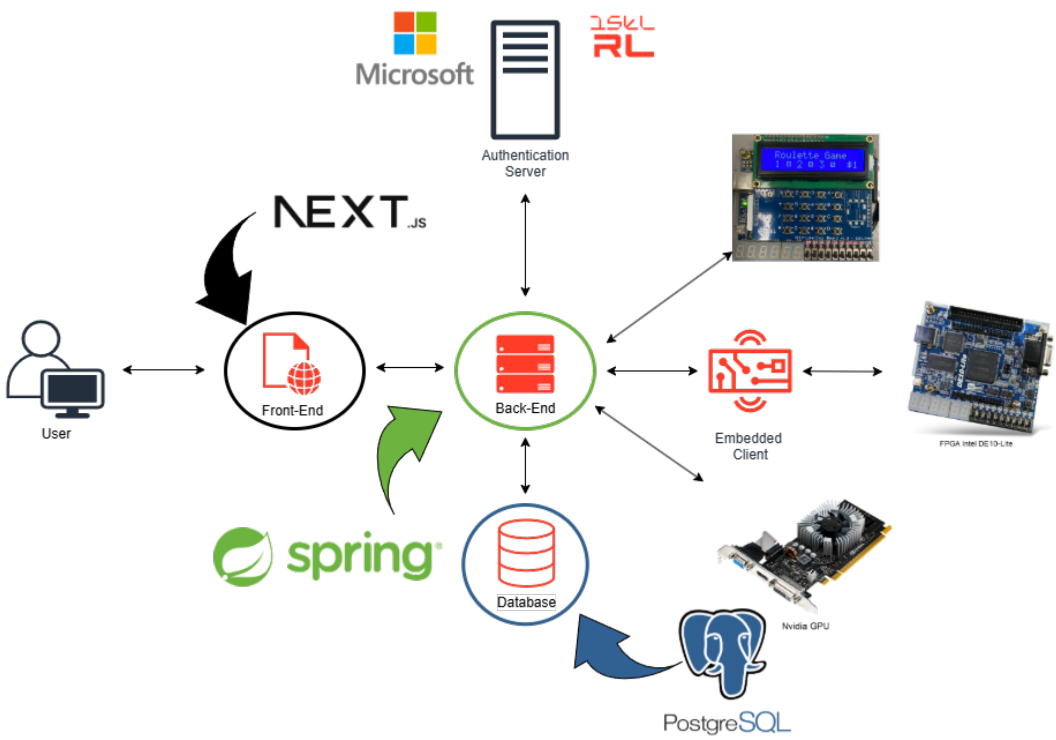

Figure 4.1: System Architecture Overview

4.2 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, migrations, and documentation, supporting the persistence layer of the system.
- **docs/** – Stores project documentation, including technical reports, user guides, and architectural diagrams.
- **img/** – Stores project images, including technical reports, user guides, and architectural diagrams.
- **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.

4.3 Implementation Details

Key implementation decisions and details include:

- **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.
- **Backend:** The backend uses JDBI for type-safe database access and is configured via environment variables for flexibility and security.
- **Frontend:** The frontend is built with Next.js, providing a modern, responsive interface and integrating with the backend via RESTful APIs.
- **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.
- **Hardware Abstraction:** The backend abstracts hardware-specific details, allowing for easy extension to new laboratory equipment.
- **Automation:** The `start.sh` script automates the bootstrap process, starting all necessary services with a single command.

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.

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.

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

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.

4.4 Database

Database

4.5 Web API

Web API

4.6 Web Application

Web App

4.7 Deployment

4.8 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.

4.9 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.

4.10 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.

4.11 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.

4.12 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.

4.13 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