instituto superior de engenharia de lisboa

Remote Lab

António Alves
Ângelo Azevedo

Advisor: Pedro Matutino

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

June 2025

Lisbon School of Engineering

Remote Lab

50539 António Alves

50565 Ângelo Azevedo

Advisor: Pedro Matutino

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

June 2025

Abstract

The design, development, implementation, and finally, the validation of digital systems require, in addition to simulators, the use of hardware to verify their implementations in real devices. In the current teaching paradigm, in which face-to-face time is reduced and remote and autonomous work is increased, it is necessary to create alternatives to the current 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.

Contents

List of Figures

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.

4.2 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.

4.3 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.

4.4 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.

4.5 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.

4.6 System Diagram

Figure 4.1: System Architecture Overview

4.7 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.

4.8 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