Bachelor's thesis

Bachelor of Engineering Information and Communications Technology

2025

Uras Ayanoglu

Development of an Embedded Lab Management System

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Bachelor of Engineering Information and Communications Technology

2025 | 75

Uras Ayanoglu

DEVELOPMENT OF AN EMBEDDED LAB MANAGEMENT SYSTEM

The purpose of this thesis was to develop a web-based management system for the embedded laboratory at Turku University of Applied Sciences. The objectives were to implement an integrated functionalities such as inventory management, learning aids for embedded systems, equipment loan tracking, table booking, and enhanced search capabilities powered by a local AI server. This work was undertaken to address challenges faced in manually managing the laboratory's resources, which often resulted in inefficiencies and inaccuracies. By introducing a structured and automated approach, the system aims to support the teaching staff and students in the Embedded Software and IoT track by improving the efficiency of resource utilization and access within the laboratory.

The development of the system followed a constructive research approach. The requirements for the system were based on the needs of teaching staff and students, and relevant theoretical principles. The Django framework was chosen as the development platform due to its maintainability and suitability for rapid prototyping. The implemented system underwent iterative ad hoc testing, incorporating user feedback to refine the functionalities. Key features, such as dynamic inventory categorization, table availability visualization, and AI-assisted search, were tested in the real environment of the lab to ensure usability and effectiveness.

The results demonstrated that the developed system significantly streamlined the management of the laboratory’s resources and improved user experience for both students and staff. In addition to addressing immediate needs, the system

provides a foundation for future use in the university’s curriculum, particularly in the application programming course and Capstone projects. This ensures the system’s continuous improvement and relevance, contributing to innovation in resource management within educational institutions.

Keywords:

HTML, CSS, Django, Python, Docker, AI, AI-Server, framework, inventory tracking, inventory management system.

Contents

Figures

Pictures

Tables

List of abbreviations

AI	Artificial Intelligence
CSS	Cascading Style Sheets
DRY	Don’t Repeat Yourself
HTML	Hypertext Markup Language content.
IoT	Internet of Things
LIMS	Library Information Management System
LMS	Library Management System
MTV	Model Template View
MVC	Model View Controller
NER	Named Entity Recognition
NLP	Natural Language Processing
ORM	Object-Relational Mapping
SQL	Structured Query Language
TUAS	Turku University of Applied Sciences
URL	Uniform Resource Locator
WSGI	Web Server Gateway Interface

1 Introduction

Efficient resource management in academic laboratories plays an essential role in enhancing the quality of teaching and learning. In specialized technical disciplines such as Embedded Software and the Internet of Things (IoT), labs are indispensable environments for hands-on learning and experimentation. Nevertheless, traditional manual methods for managing these resources are often inefficient, prone to errors, and labor-intensive. To tackle these issues, this thesis explores the development of a web-based management system tailored for the embedded laboratory at Turku University of Applied Sciences (TUAS).

The motivation for developing a lab management system arises from the practical challenges encountered in managing the embedded lab's resources. Currently, the embedded laboratory depends on manual processes for inventory tracking and resource booking, which lead to inefficiencies and inaccuracies. With the growing emphasis on digital tools in modern education, developing an automated solution that meets the specific requirements of an embedded lab is both relevant and essential. This thesis not only seeks to resolve these challenges but also supports the university's broader goals for digital transformation.

1.1 Laboratory Management Systems

Laboratory Management Systems (LMS) and Laboratory Information Management Systems (LIMS) have evolved significantly from in-house projects to fully integrated enterprise solutions, primarily serving healthcare, research laboratories, and private sectors to streamline operations (Prasad & Bodhe, 2012). However, these systems primarily focus on managing biological and chemical research workflows, making them unsuitable for technical environments such as embedded laboratories. Similarly, inventory tracking systems, albeit effective for monitoring item usage and costs, lack the specialized features required for academic laboratory environments. Library management systems, which specialize in lending and returning resources such as books, also fall short

in addressing the specific demands of embedded labs, such as equipment lending and workspace booking.

The unique requirements of the embedded lab manager include managing a diverse inventory of technical equipment, facilitating equipment loans, booking laboratory tables, and integrating advanced search capabilities using AI. These needs are not met by existing solutions; hence, a custom-built system is developed.

1.2 Project's Aim and Objectives

This project's main objective is to develop a web-based system that consolidates inventory management, equipment lending, table booking, learning aid providing, and AI-enhanced search functionality into a single cohesive platform. By automating and integrating these processes, the system will address the inefficiencies of manual management while offering enhanced usability for students and faculty. One of the main objectives is to create an easily maintainable and user-friendly solution that meets the current needs of the embedded laboratory while providing a foundation for future improvements.

1.3 Thesis Overview

The thesis is structured into six chapters, each addressing different aspects of the Embedded Lab Management System. Chapter 1 introduces the problem context, project objectives, and the overall structure of the thesis. Chapter 2 examines the technologies utilized in developing the system, including web-based management concepts, the Python programming language, the Django framework, and other supporting tools. Chapter 3 outlines the functional and non-functional requirements, emphasizing maintainability as a key goal, and discusses critical design decisions. Chapter 4 details the development process, focusing on the modular implementation of core applications within the system. Chapter 5 covers the verification phase, demonstrating how the system meets

the specified requirements through various testing methods, including checklists and visual documentation. Finally, chapter 6 evaluates the system's effectiveness, addresses its limitations, and provides recommendations for future improvements.

By bridging the gap between manual processes and digital solutions, this thesis aims to make a practical contribution to academic resource management and set a precedent for ongoing digital innovation at TUAS.

2 Technologies Used for the Embedded Lab Management System

This chapter explores the theoretical foundations and technologies underpinning the development of the Embedded Lab Manager. It provides an overview of the key tools and frameworks used in the project, including Python, Django, virtual environments, Python packages, and Docker. The review discusses why these technologies were chosen and their relevance to the project.

2.1 Overview of the Python Programming Language

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Since its inception in 1991, Python has gained widespread popularity in various domains, including web development, data analysis, artificial intelligence, and scientific computing. The language's design philosophy emphasizes code readability, using indentation to structure code blocks and reducing reliance on complex syntax (Peters, 2004). These characteristics make Python an ideal choice for developers, especially for academic and research projects where clarity and maintainability are crucial.

One of the primary reasons for selecting Python in this project is its extensive ecosystem of libraries and frameworks, which significantly accelerates the development process. For instance, Django, the web framework used in this project, is built on Python and inherits its advantages, such as ease of learning and compatibility with other tools. Additionally, Python’s thriving community ensures that developers have access to robust support and documentation.

Python's relevance extends to its role in integrating advanced functionalities such as artificial intelligence and machine learning. Libraries such as TensorFlow, PyTorch, and Scikit-learn, combined with Python's seamless API capabilities, allow developers to incorporate sophisticated AI models. In this project, Python facilitates communication with a local AI server, enabling features like inventory search and resource recommendations based on user queries.

Moreover, Python’s portability and cross-platform compatibility make it an excellent choice for a web-based management system. The Embedded Lab Manager leverages Python’s ability to run seamlessly across different operating systems, ensuring accessibility for developers and end-users alike. This aligns with the project’s goal of creating a scalable and adaptable solution for resource management.

2.2 Overview of the Django Framework

Django is a high-level Python web framework designed to facilitate the rapid development of secure and maintainable web applications. Released in 2005, Django adheres to the principles of the Model-View-Template (MVT) architectural pattern, enabling developers to create scalable and reusable code. The framework is particularly valued for its "batteries-included" philosophy, providing built-in features such as an authentication system, an ORM (Object-Relational Mapping) tool, and an admin interface (Mele, 2024).

Django’s popularity stems from its comprehensive feature set and robust community support, making it an ideal choice for both small-scale projects and large, enterprise-level applications. Its emphasis on security ensures that developers can implement best practices with minimal effort, including protection against common vulnerabilities like SQL injection and cross-site scripting.

2.2.1 Why Django Was Chosen as a Framework

Several factors influenced the selection of Django for the development of the Embedded Lab Manager:

1. **Ease of Use and Rapid Development**: Django’s built-in tools and clear documentation significantly reduce development time, allowing for quick implementation of core functionalities. This aligns with the project’s requirement for a timely and efficient development process.

2. **Scalability**: Django's modular design and ability to handle high traffic volumes make it suitable for an academic lab management system, where multiple users may access the platform simultaneously.

3. **Security**: Django provides built-in protection against common web vulnerabilities, ensuring that the system remains secure for sensitive operations such as user authentication and resource management.

4. **Community and Ecosystem**: The extensive Django community offers abundant resources, plugins, and third-party applications, which can be leveraged to extend the platform's capabilities without starting from scratch.

5. **Alignment with Project Goals**: As a Python-based framework, Django integrates seamlessly with the other technologies used in the project, including the local AI server. This integration capability simplifies the implementation of advanced features like AI-enhanced search.

2.2.2 Django's Approach to Web Applications

Django employs the Model-View-Template (MVT) architecture (Fig. 1), a variation of the popular Model-View-Controller (MVC) design pattern (Django introduction, n.d.). The framework separates the application logic into three distinct components:

1. **Model** represents the data structure and handles the interaction with the database. In the Embedded Lab Manager, models such as InventoryItem and TableBooking define the structure of key resources and their relationships.

2. **View** contains the business logic and communicates between the model and the template. Views process user requests and render appropriate responses, ensuring a seamless user experience.

3. **Template**: It defines the presentation layer, enabling developers to create dynamic HTML pages with minimal effort. The use of Django templates in this project ensures a consistent and user-friendly interface.

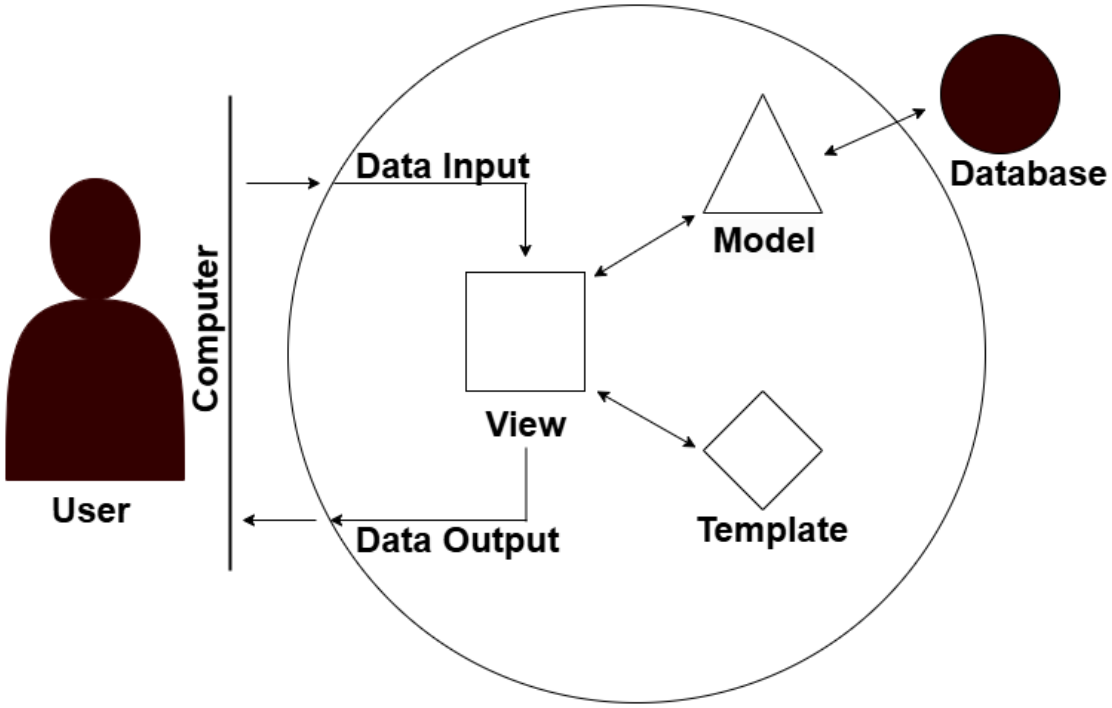


Figure 1. Django architecture as MVT.

Django also emphasizes the concept of "Don’t Repeat Yourself" (DRY), encouraging developers to write reusable and maintainable code. Features such as generic views, middleware, and form handling further streamline development and reduce redundancy (Mele, 2024).

The choice of Django for the Embedded Lab Manager reflects its suitability for building feature-rich, scalable, and secure web applications. By leveraging Django’s strengths, the project achieves its goal of creating an integrated platform for inventory management, equipment lending, table booking, and AI-driven functionalities.

2.3 Importance of Virtual Environments in Development

Virtual environments are isolated Python environments that allow developers to manage dependencies for individual projects without interfering with the system-wide Python installation. They are particularly valuable in projects where multiple Python applications with differing requirements coexist. The Python venv module, included in the standard library since Python 3.3, provides a straightforward method for creating and managing virtual environments.

2.3.1 Benefits of Using Virtual Environments

Virtual environments provide a controlled and isolated workspace for Python projects, ensuring that dependencies remain consistent and do not interfere with system-wide packages or other projects. Benefits of virtual environments include:

1. **Dependency Isolation**

   Virtual environments ensure that dependencies for a specific project do not conflict with those of other projects. This is especially important when working with multiple projects that may require different versions of the same package.

2. **Reproducibility**

   By isolating dependencies, virtual environments make it easier to reproduce the development environment on another system. Using a requirements.txt file, developers can recreate the exact environment needed for the project, ensuring consistent behavior across deployments.

3. **Avoiding System Conflicts**

   System-wide installations can lead to compatibility issues, especially when working on projects that require older or newer versions of Python

packages. Virtual environments eliminate this risk by allowing projects to define their dependencies independently.

4. **Portability**

   Virtual environments enable seamless transition between development and production environments. This is crucial for projects like the Embedded Lab Manager, which may require deployment on different servers or systems.

2.3.2 Virtual Environments in the Embedded Lab Manager Project

In this project, the Python venv module was used to create a virtual environment for managing dependencies. This approach ensured that the libraries and tools used during development were consistent and did not interfere with system-wide installations. Key Python packages, including Django, were installed within the virtual environment, simplifying dependency management and minimizing the risk of compatibility issues.

While the setup process for the virtual environment will be detailed in Appendix 1, its importance in the overall development workflow cannot be overstated. The use of a virtual environment facilitated smooth integration of various technologies, including Django, Docker, and the local AI server, by providing a clean and isolated development space.

Virtual environments play a critical role in modern Python development, and their inclusion in this project reflects best practices for managing dependencies and ensuring reproducibility. By leveraging this tool, the Embedded Lab Manager achieves a robust and maintainable development setup.

2.4 Python Packages Used with the Django Framework

The development of the Embedded Lab Manager project relied on several Python packages, each serving a specific purpose in enhancing functionality, maintaining compatibility, or supporting deployment. This section provides an overview of the primary packages listed in the project's requirements.txt file.

2.4.1 Core Packages

The Embedded Lab Manager relies on a set of essential Python packages to ensure smooth functionality, maintainability, and performance. These core packages provide foundational support for web development, database management, and asynchronous processing. Core packages used in the project include:

1. Django (5.1.2):

   Django is the cornerstone of the Embedded Lab Manager, providing the framework for web application development. Its robust features, including the ORM, URL routing, and template engine, make it an ideal choice for building scalable and maintainable web applications.

2. Sqlparse (0.5.1):

   This package is used internally by Django for parsing and formatting SQL queries. It ensures that the database queries generated by Django's ORM are syntactically correct and optimized for execution.

3. Asgiref (3.8.1):

   The ASGI reference implementation is essential for handling asynchronous server interfaces in Django. It enables support for modern

web protocols and real-time features, aligning with Django's push toward asynchronous capabilities.

2.4.2 Image Processing

4. Pillow (10.4.0):

   Pillow is a powerful image processing library used in the Embedded Lab Manager for handling uploaded images. It supports a wide range of image formats and allows resizing, cropping, and other manipulations essential for inventory item images.

2.4.3 HTTP Requests

5. Requests (2.32.3):

   Widely regarded as one of Python's most user-friendly HTTP libraries, Requests is used in this project to facilitate communication with external APIs, such as the local AI server. It provides simple and intuitive methods for sending HTTP requests and handling responses.

6. urllib3 (2.2.3):

   A dependency of the Requests library, urllib3 is a low-level library for handling HTTP connections. It ensures secure and reliable communication by managing connection pooling and SSL verification.

7. certifi (2024.8.30):

   Certifi provides a curated list of root certificates, ensuring secure HTTPS connections when making API requests. This is critical for maintaining data

integrity and privacy during interactions with the AI server or other external services.

8. idna (3.10):

   IDNA, short for Internationalized Domain Names in Applications, ensures that domain names with non-ASCII characters are properly encoded and handled during HTTP requests.

2.4.4 Deployment and Production

9. Gunicorn (>=20.0.0):

   Gunicorn (Green Unicorn) is a WSGI server used for deploying Python web applications in production environments. Its ability to handle multiple requests concurrently ensures that the Embedded Lab Manager can efficiently serve users during peak times.

2.4.5 Relevance to the Embedded Lab Manager

These packages collectively support the development, functionality, and deployment of the Embedded Lab Manager. Django forms the foundation of the system, while complementary libraries like Pillow and Requests extend its capabilities. Deployment packages such as Gunicorn ensure that the application runs smoothly in production, handling user requests efficiently.

By leveraging these carefully chosen packages, the project adheres to best practices for modern web application development and deployment.

2.5 Docker: Containerization for Development and Deployment

Docker is an open-source platform that enables developers to package applications and their dependencies into lightweight, portable containers. These containers can run consistently across various environments, from local development setups to production servers. Docker's ability to create isolated environments ensures compatibility and eliminates issues arising from differences in system configurations.

2.5.1 Overview of Docker

Docker simplifies application development and deployment by encapsulating code, runtime, libraries, and configurations into a single container image. This image can be shared and deployed across multiple systems without the need to install dependencies manually. Docker’s efficiency stems from its use of containerization, which leverages the host operating system's kernel, making it more lightweight than traditional virtual machines.

2.5.2 Benefits of Docker for the Embedded Lab Manager Project

1. **Consistent Development and Deployment Environments**:

   Docker ensures that the application behaves identically across development, testing, and production environments. This consistency minimizes the "it works on my machine" problem often encountered during deployment.

2. **Simplified Dependency Management**:

   By packaging all dependencies within a container, Docker eliminates the need to manually configure environments. This is particularly useful for the

Embedded Lab Manager, which relies on multiple Python packages, database configurations, and external tools.

3. **Scalability**:

   Docker makes scaling applications straightforward. Multiple instances of the Embedded Lab Manager can be deployed using Docker containers, ensuring that the system can handle increased user demand efficiently.

4. **Portability**:

   Docker containers can run on any system with Docker installed, making it easier to deploy the application on different servers or cloud platforms.

5. **Isolation**:

   Docker containers operate in isolated environments, ensuring that the Embedded Lab Manager does not conflict with other applications running on the same server.

2.5.3 Docker in the Embedded Lab Manager Project

In this project, Docker was used to containerize the Embedded Lab Manager for deployment. A Dockerfile was created to define the application's environment, specifying the base image, required dependencies, and setup commands. Additionally, a compose.yaml file was used to configure multi-container setups, including the application server and the database.

The laboratory's infrastructure also includes a Local AI server running on a Docker container. While the Local AI server itself is beyond the scope of this thesis, its use demonstrates the advantages of Docker in maintaining compatibility and ease of integration between various systems. The AI server

provides advanced search functionality to the Embedded Lab Manager, and Docker ensures its reliable operation alongside the management system.

Figure 2 illustrates the Docker-based architecture of the Embedded Lab Manager project.

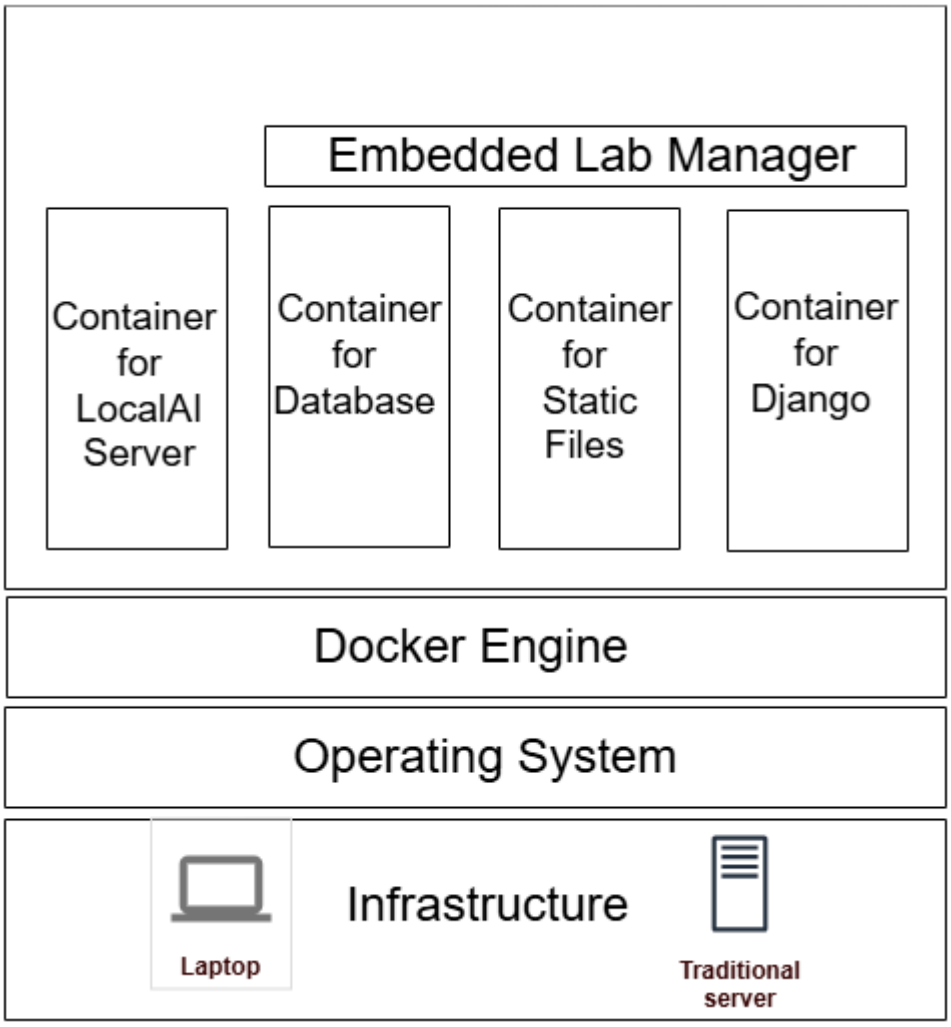


Figure 2. Docker-based architecture of Embedded Lab Manager.

The ability to deploy the Embedded Lab Manager and the AI server as containerized applications within the same environment highlights Docker's role in creating a cohesive and scalable ecosystem for the laboratory. This consistency simplifies maintenance and supports future expansion of the lab's digital tools.

3 Requirements for the Embedded Lab Management System

This chapter outlines the requirements necessary for the development of an effective Embedded Lab Management System. The requirements are divided into functional and non-functional aspects, with a focus on maintainability as a primary goal. Additionally, critical design decisions based on requirement analysis are discussed to justify the chosen architecture and implementation approach.

3.1 Functional Requirements

The functional requirements define the core capabilities that the Embedded Lab Manager should provide to its users. These include features and interactions necessary to achieve the system's objectives.

3.1.1 Inventory Management

- Ability to add, update, and delete inventory items.
- Categorization of items to facilitate easy search and tracking.
- Loan management functionality to monitor borrowed items.
- Ability to display item details including specifications, images, notes and location.

3.1.2 Booking System

- Ability to book lab tables for personal study or assignments.
- Availability status of tables displayed dynamically.
- Ability to approve or reject bookings by admins.
- Ability to edit or cancel bookings by users.

3.1.3 AI Integration

- Ability to query an AI assistant for support.
- Information provided by AI based on available inventory.
- Keyword-based inventory searches.

3.1.4 User Authentication and Authorization

- Secure login and registration system.
- Role-based access control (students, teaching staff, administrators).

3.1.5 Reporting and Feedback

- User feedback through a dedicated form.
- Information availability by filtering Inventory, bookings and loans.
- Information availability about overdue items or pending approvals.

3.1.6 Learning Aids

- Availability of supportive educational resources.
- Gathering all materials together into the same ecosystem.
- Availability of filtering resources by their type.

3.2 Non-Functional Requirements

Non-functional requirements address the operational qualities of the system, ensuring it meets the necessary standards for usability, security, and maintainability.

3.2.1 Maintainability

- Modular architecture to allow easy updates and modifications.
- Clear separation of concerns (inventory, booking, loaning, AI, etc.).
- Comprehensive documentation for code and system operations.
- Version control integration for tracking changes.

3.2.2 Usability

- Intuitive user interface with minimal learning curve.
- Responsive design to support multiple devices (desktop, mobile).
- Consistent user experience across different features.

3.2.3 Performance

- Efficient database queries to ensure fast response times.
- Scalability to support a growing number of users and data.

3.2.4 Security

- Data encryption for sensitive information.
- Secure authentication and authorization mechanisms.
- Protection against common web vulnerabilities (XSS, SQL injection).

3.3 Requirement Analysis – Critical Design Decisions

Critical design decisions were made based on the functional and non-functional requirements to ensure the system meets its objectives effectively.

3.3.1 Modular Design Approach

- Separate Django applications for different functionalities (inventory, booking, AI).
- Independent components to allow parallel development and scalability.

3.3.2 Database Structure

- Use of relational database (PostgreSQL) to manage inventory and bookings.
- Optimization through indexing and foreign key constraints

3.3.3 AI Integration Strategy

- Separation of AI processing logic from the Django web application.
- Use of a dedicated communication module for interfacing with the AI server.

3.3.4 Deployment Considerations

- Docker-based deployment for consistency across environments.

3.3.5 Security Measures

- Deployment in a closed network.
- Role-based access control to prevent unauthorized actions.

4 Application Development Process: Modular Approach

The Embedded Lab Manager was developed using a modular approach, ensuring that each application within the project serves a specific purpose while remaining decoupled from other components. This strategy promotes reusability, maintainability, and scalability, aligning with best practices for Django projects. Before delving into the specifics of each application, this section outlines the general methodology employed during the development process.

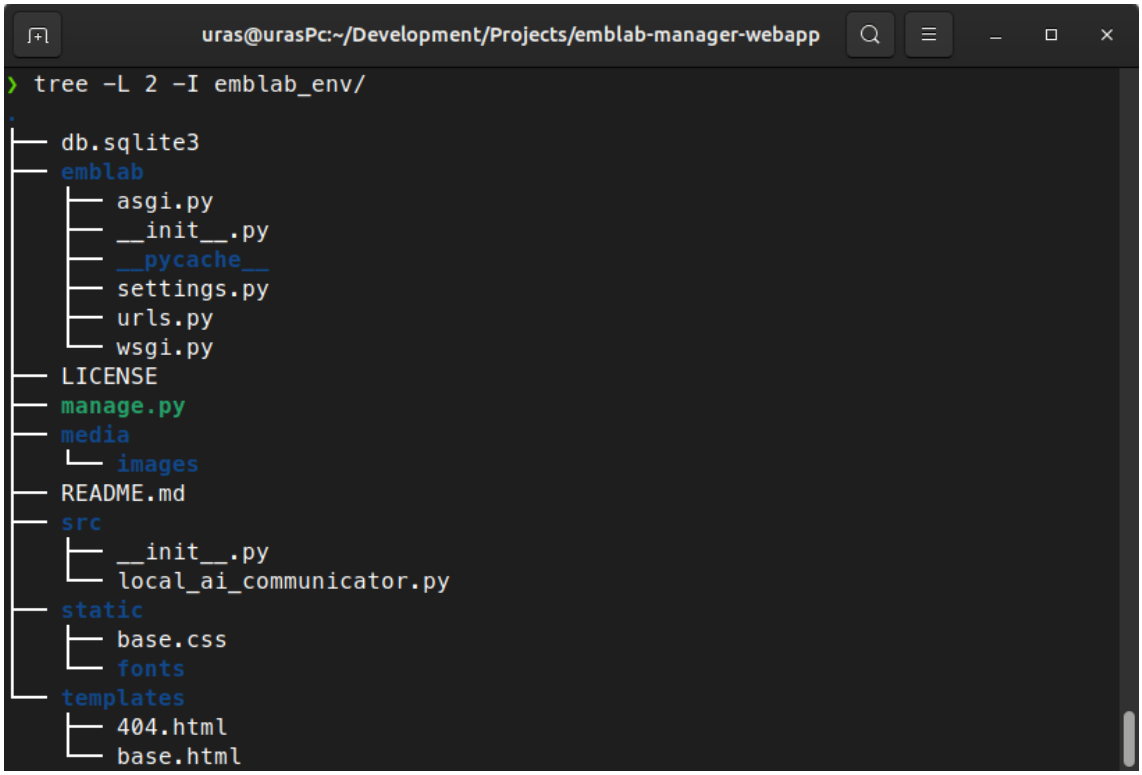
4.1 General Approach to Application Development

4.1.1 Project Setup and Directory Structure

Before creating individual applications, the foundational structure (Picture 1) of the project was established to streamline development:

Shared Directories:

- **Media**: A directory for user-uploaded files, such as images of inventory items.
- **Templates**: A global directory for reusable HTML templates, including a base.html file containing the boilerplate layout for the project.
- **Static**: A directory for shared static files, such as CSS and JavaScript. A base.css file was created here for the global styling of the project.



Picture 1. Project directory structure.

These shared resources provided a unified starting point for all applications, reducing repetitive tasks and maintaining a consistent design.

Settings Configuration:

- Paths for templates, static files, and media files were defined in settings.py.
- Additional settings, such as database configurations, time zone, language, debug options, and security settings, were adjusted as needed during development.

4.1.2 Development Workflow for Applications

1. **Defining Models**:

   The development of each application begins with defining the data models in models.py. This step involves careful consideration of the database schema to ensure it meets both current and anticipated needs of the laboratory staff. Each model is designed to reflect real-world entities, such as inventory items or booking records, with appropriate fields and relationships.

2. **Admin Panel Configuration**:

   After creating the models, the next step is to register them in admin.py. This makes the models accessible via Django's built-in admin interface. The admin panel is customized to provide intuitive options for listing, filtering, and managing data, ensuring ease of use for laboratory administrators. For applications without models, the admin panel is not configured.

3. **Building Views and URLs**:

   The third step is to create views in views.py to define the logic for rendering templates and processing user requests. When working view functions corresponding HTML templates are designed for each view, leveraging reusable components from base.html. For styling the templates CSS files are added to the app-specific static folder. After all these steps, a urls.py file is created for each application to define app-specific URL patterns, enabling modularity and easy integration with other projects.

4. **Registering Applications**:

   Once an application is ready, it is added to the INSTALLED_APPS list in the project's settings.py. This step allows Django to recognize the application. The project's main urls.py file is then updated to include the app's URL patterns, linking the app's views to the broader project.

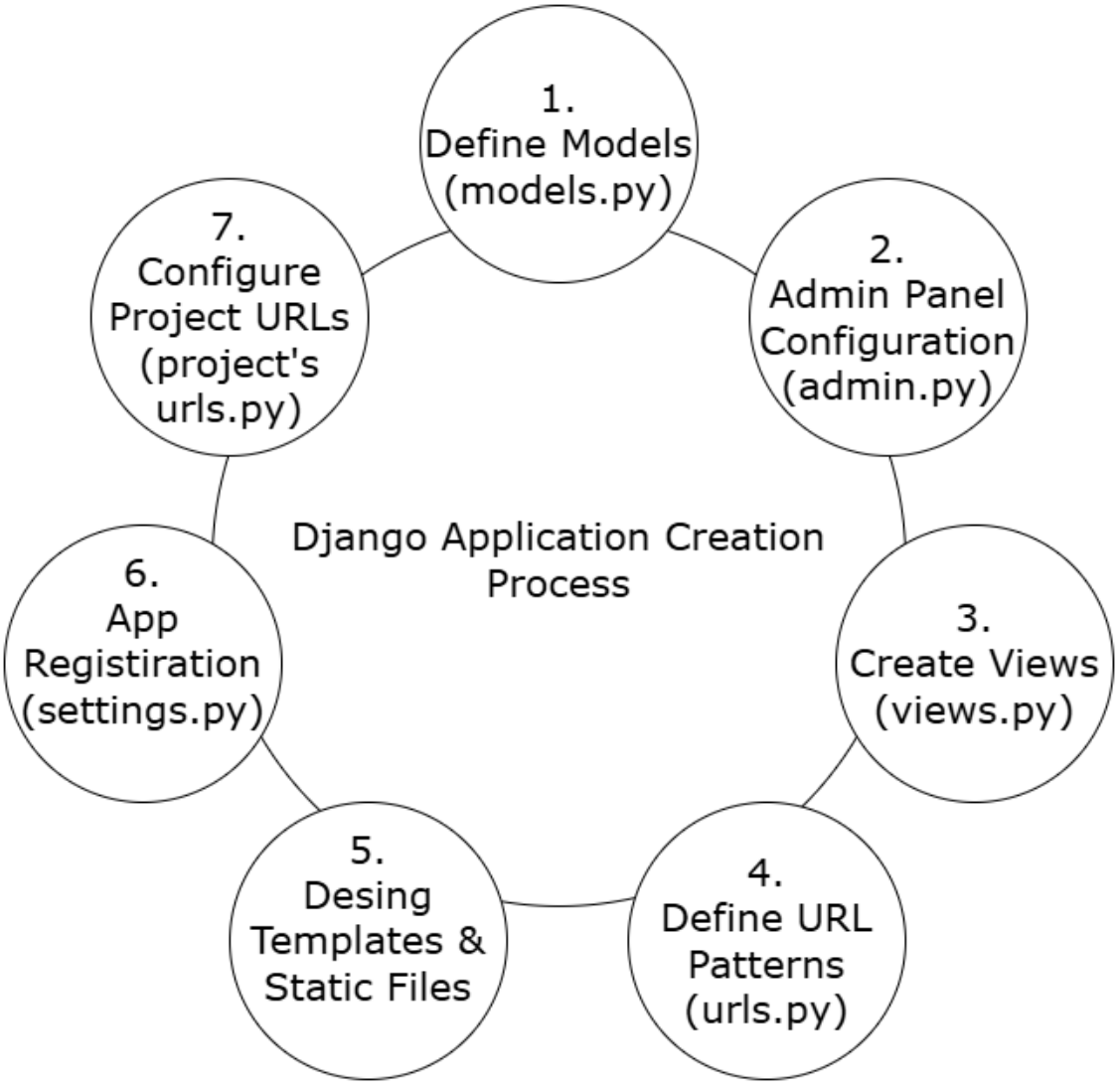


Figure 3. Django application creation process.

4.1.3 Benefits of the Modular Approach

- **Reusability**: Applications and their components can be easily reused in other projects, saving development time.
- **Scalability**: The modular structure allows for seamless addition of new features or applications without disrupting existing functionality.
- **Maintainability**: Each application operates independently, simplifying debugging and updates.

This systematic approach provided a solid foundation for building the Embedded Lab Manager, ensuring that each application was developed efficiently and integrated seamlessly into the overall project. The following sections provide detailed discussions on the development of specific applications.

4.2 Home and Users Applications

4.2.1 Home Application

The Home application was developed to manage the landing page and its associated functionalities. Its primary purpose was to create a centralized space for displaying key information about the web application, including announcements, guidance for users, and a feedback mechanism. This separation into a distinct app provided flexibility and maintainability for managing the project's entry point.

Key Features and Design Choices:

1. Data Models:

   The home application includes three data models:

   - Announcement: Manages announcements that staff can publish on the landing page. It includes attributes such as title, date, description, and

slug. The date attribute allows announcements to be ordered chronologically in the admin panel, and the slug attribute ensures meaningful URLs for better readability and future proofing.

- BackgroundImage: Provides the ability to change the background image of the landing page without modifying the source code. This model supports dynamic customization of the page's visual appearance.

- UserFeedback: Allows users to submit feedback about the web application or inventory. This feature fosters communication between users and the administrative team, enabling continuous improvement.

2. Administrative Convenience:

   The inclusion of filters and ordering options in the admin panel ensures that announcements and other data can be managed efficiently.

3. Future-Proof Design:

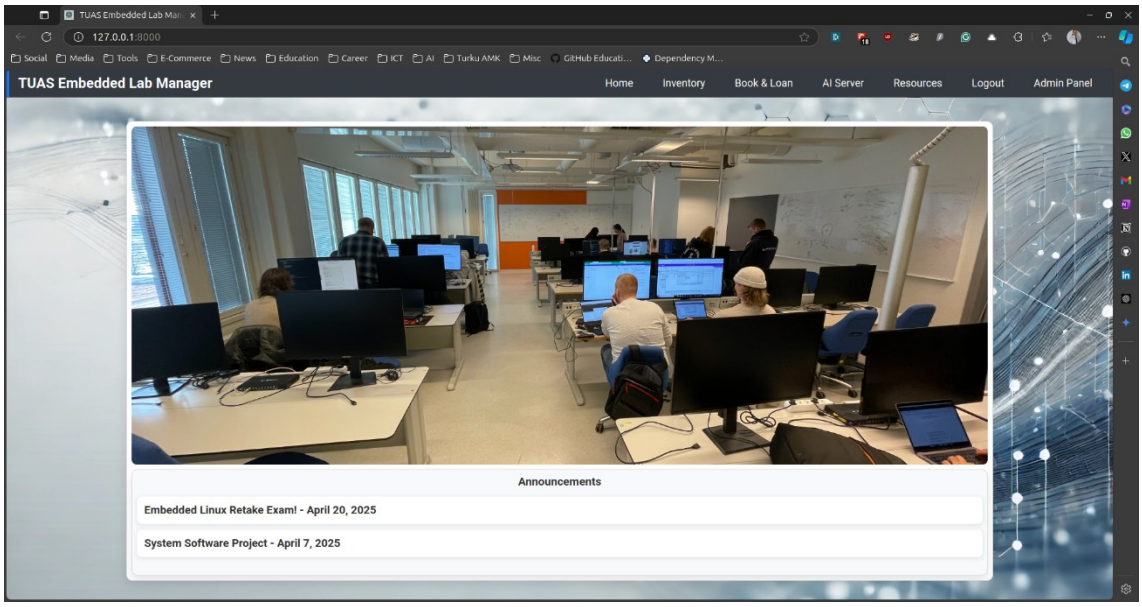
   While the Embedded Lab Manager is deployed locally and does not require SEO, the use of slugs prepares the application for potential future scenarios where external accessibility might be necessary.

4. User Guidance:

   The Home app also provides a brief guide to using the web application, ensuring that users understand the purpose and functionality of each module within the Embedded Lab Manager.

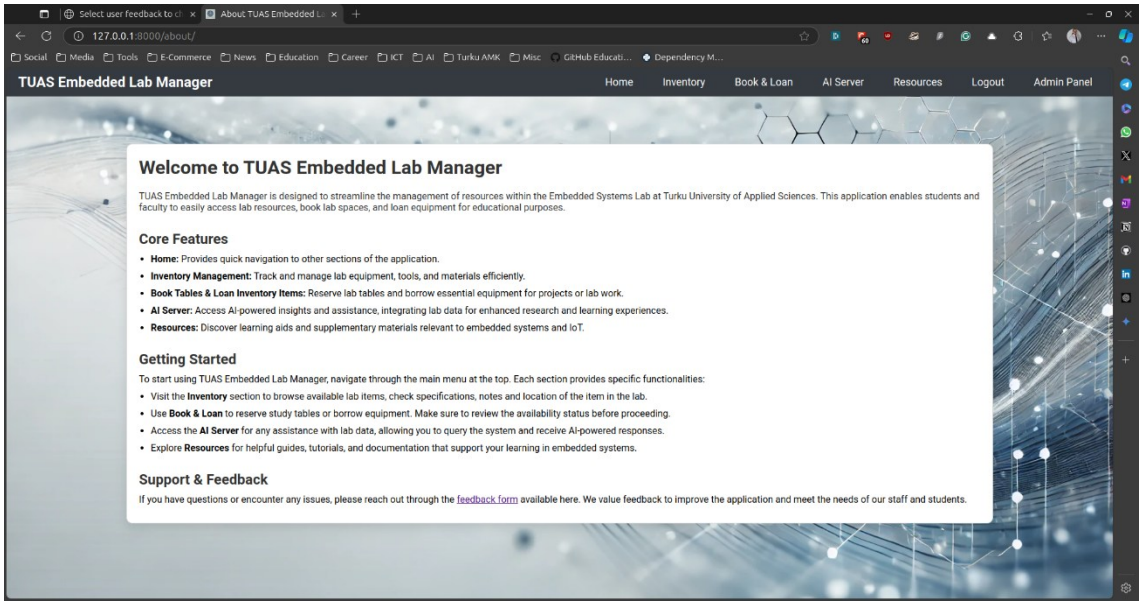
The development workflow for the home application followed the general modular approach outlined earlier, ensuring consistency across the project.

Picture 2 illustrates the landing page of the Embedded Lab Manager. The page serves as the main entry point for users, offering navigation to key features such as inventory management, table booking, AI assistance, and resources. The page design emphasizes a user-friendly and intuitive layout, reducing distractions for both students and teaching staff.



Picture 2. Landing page of the TUAS Embedded Lab Manager.

The about page, shown below (Picture 3), provides an overview of the project's purpose and guides users through its various features. Additionally, a feedback form is linked on this page, allowing users to submit suggestions, issues, or other comments to the teaching staff.



Picture 3. TUAS Embedded Lab Manager's about page.

4.2.2 Users Application

The Users application leverages Django's built-in authentication system to manage user accounts and permissions. It enables administrators to assign specific roles to user groups, controlling access to various features of the web application.

Key Features and Design Choices:

1. **Custom Authentication Views**:

   While Django provides built-in login, signup, and logout functionalities, the built-in system's limited customization options for templates motivated the creation of custom views.

   - Custom views (signup_view, login_view, and logout_view) were developed to provide full control over the design and behavior of authentication pages.

- Corresponding templates and CSS files were created to ensure a seamless and user-friendly interface.

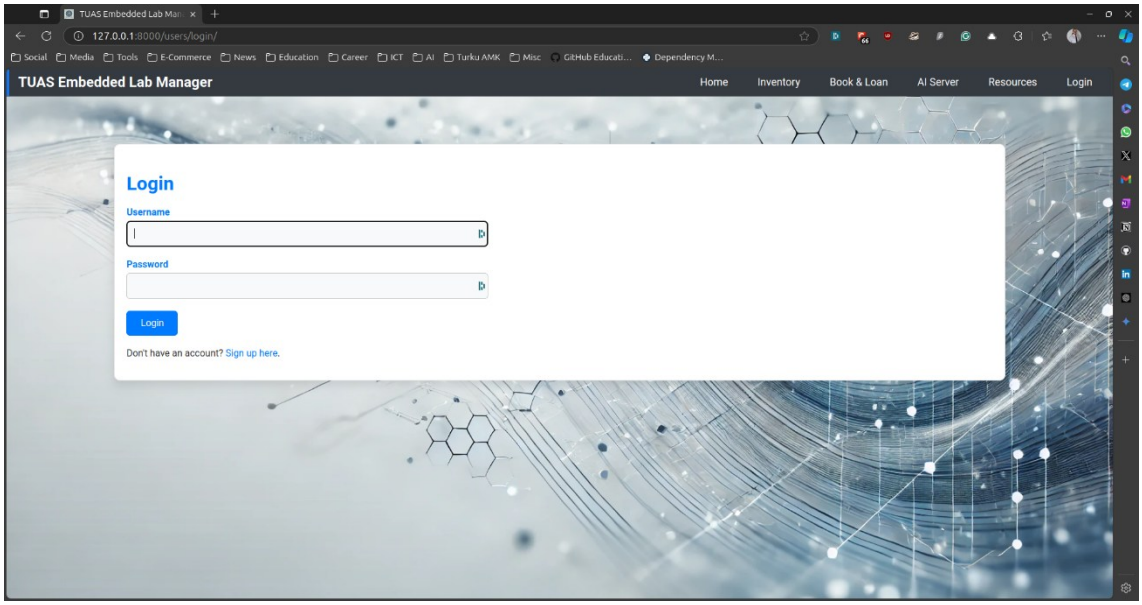
2. **No Additional Data Models**:
   The Users application did not require additional data models since Django's built-in user model provided all the necessary functionality.

3. **Permissions and Groups**:
   User groups and permissions were configured using Django's built-in admin tools, enabling a clear separation of roles and access levels.

By customizing the user authentication process, the Users application improved the usability and aesthetic coherence of the web application while retaining the robust functionality of Django's authentication system.

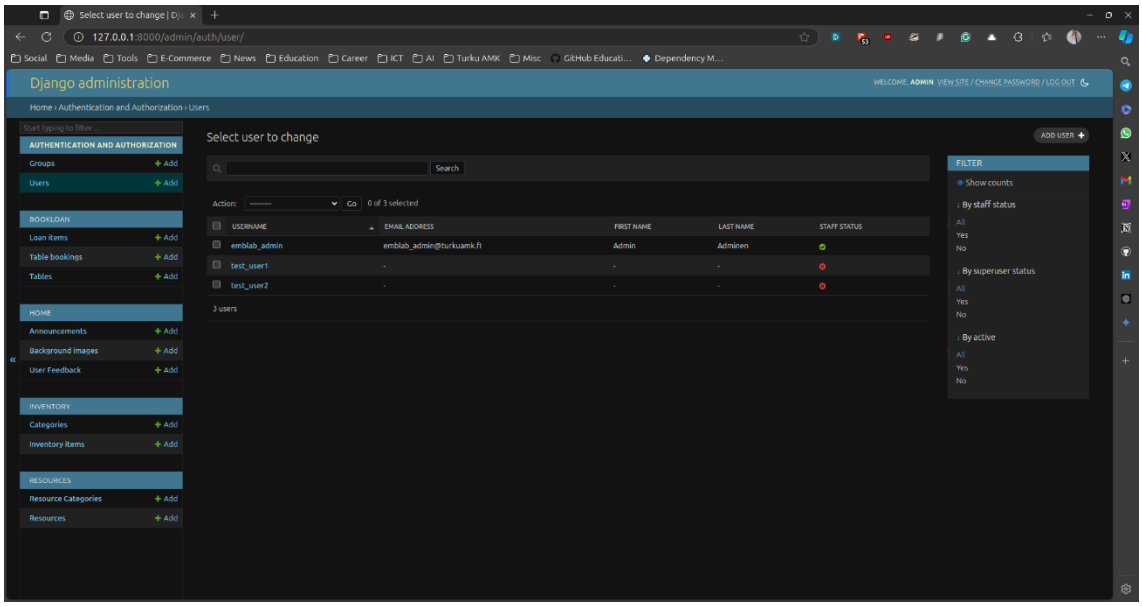
In both applications, repetitive steps such as configuring URL patterns, registering apps in settings.py, and integrating with the project's URL patterns were performed following the same modular approach discussed earlier.

The screenshot below (Picture 4) displays the login page of the application. Users are required to authenticate before accessing the system's features. The page also includes a link to the sign-up page for new users who have not yet created an account. This ensures that both authentication and account creation are streamlined within a single interface.



Picture 4. Embedded Lab Manager login page.

The next screenshot shows the Admin panel's User section under Authentication and Authorization. This interface allows administrators to view, manage, and assign roles to registered user accounts. Through this panel, admins can add new users, edit user information, and control access permissions, ensuring secure and role-based access throughout the application.



Picture 5. Embedded Lab Manager users interface of the admin panel.

4.3 Inventory Application

The Inventory Application serves as the backbone of the Embedded Lab Manager, providing the foundational data models and functionality upon which other applications, such as the Resource, BookLoan, and AI Server applications, depend. Its primary purpose is to organize, manage, and display the inventory items in a user-friendly and efficient manner.

4.3.1 Data Models

The Inventory Application includes two key data models: Category and InventoryItem.

1. Category: The category model is designed to group inventory into logical hierarchies, making navigation and search process intuitive for users.

   Attributes:

   - name: The name of the category (e.g., “Hardware and Components”).
   - parent_category: A self-referential foreign key that allows for nested categories. For example:

     - Top-level: “Hardware and Components”
     - Subcategory: “Passive Components”
     - Sub-subcategory: “Resistors”

   - description: A brief description of the category.
   - slug: Provides meaningful URLs for categories (e.g., /inventory/hardware-and-components/resistors/).

Use Case:

The hierarchical structure ensures that users can easily navigate the inventory. For instance, a 10k carbon film resistor would be categorized as:

Hardware and Components -> Passive Components -> Resistors

2. InventoryItem: The InventoryItem model represents individual items in the inventory, capturing critical details about each item.

   Attributes:

   - name: The name of the item (e.g., "PicoScope 2205A").
   - category: A foreign key linking the item to a Category.
   - can_be_loaned: A boolean indicating if the item is available for loan.
   - location: The item's physical location in the lab.
   - serial_number and mac_address: Unique identifiers to distinguish items, aiding in inventory management and troubleshooting.
   - quantity: Tracks the number of available items.
   - image: Stores an image of the item for visual identification.
   - official_link: Links to the official product page for additional information.
   - specs: Brief technical specifications of the item.
   - notes: Allows teaching staff to add specific hints or lab-related experiences.

   Benefits:

   The attributes collectively provide comprehensive information, helping students and staff locate, understand, and utilize items effectively. For example:

   - A student searching for a Raspberry Pi can identify its MAC address without needing terminal commands.
   - The quantity attribute helps staff plan ahead for lab sessions by checking item availability.

4.3.2 Admin Panel Configuration

By registering the Category and InventoryItem models in admin.py, the Inventory Application enables robust administrative functionality:

- Filtering: Items can be filtered by attributes like category, location, and quantity.
- Search: Administrators can search by name, serial number, MAC address, or location.
- Ordering: Items can be sorted by attributes such as name or quantity.

This configuration simplifies inventory management for teaching staff, reducing the time spent on manual tracking.

4.3.3 User Interface Design

The Inventory Application uses a single template, inventory-index.html, and its associated stylesheet, inventory-index.css. This design ensures that all inventory-related functionality is consolidated onto a single page, minimizing distractions and simplifying navigation.

1. View Functions:

    - inventory_list: Displays the top-level inventory categories, search functionality, and detailed item view.
    - inventory_subcategory_list: Handles navigation through subcategories.
    - Inventory_category_detail: Shows detailed information about items in a selected category.

2. Dynamic Rendering: All categories, subcategories, and items are dynamically loaded from the database. This ensures that any changes made to the inventory (e.g., adding or removing categories or items) are

immediately reflected on the user interface without requiring additional coding.

3. User Interaction:

    - Users can either search for items using the search bar or navigate through categories and subcategories.
    - The dynamic structure ensures ease of use and reduces navigation complexity.

4.3.4 URL Patterns and Integration

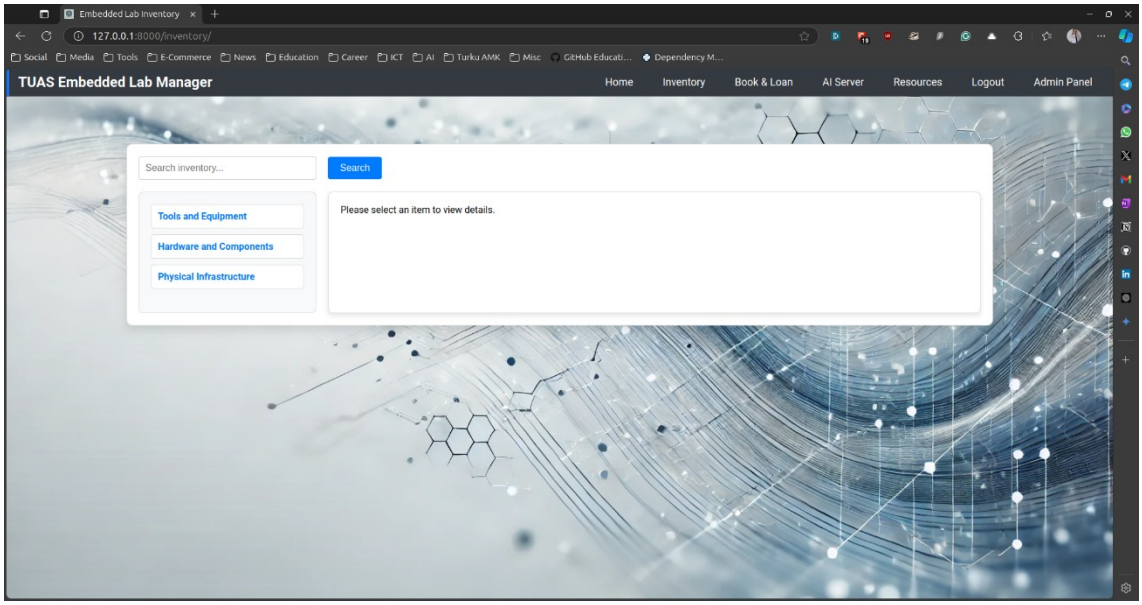
After implementing the models, views, templates, and static files:

- URL patterns were defined in urls.py to map views to specific endpoints.
- The Inventory Application was registered in the project's settings.py, and its URL patterns were included in the main urls.py.

4.3.5 Design Rationale

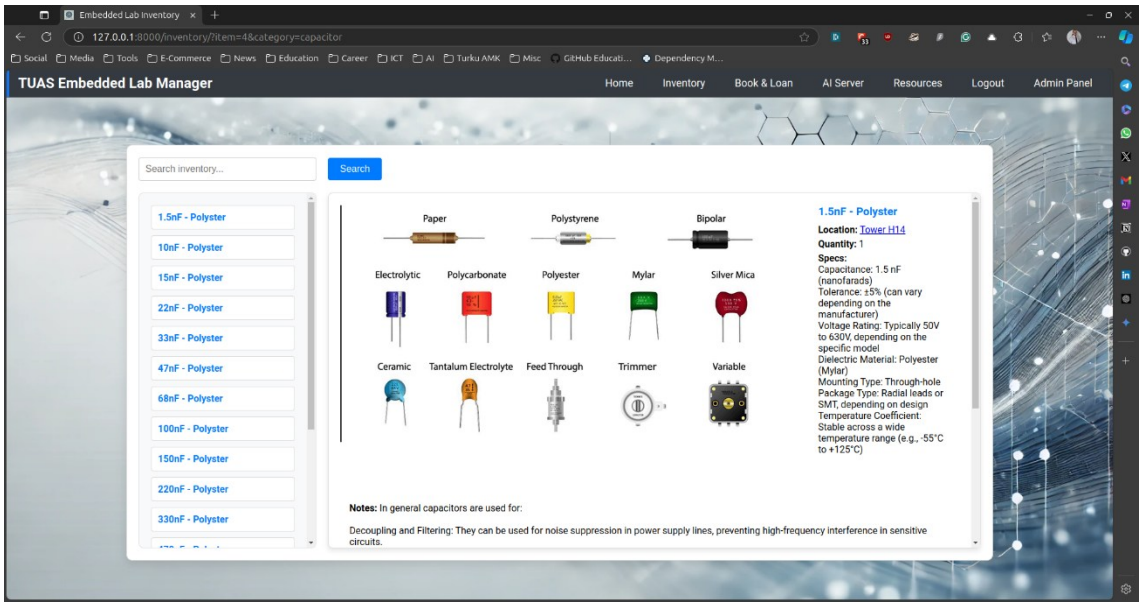
The decision to consolidate inventory functionality onto a single page reflects the goal of minimizing distractions for users. This approach allows students and staff to efficiently locate items without needing to navigate multiple pages. The dynamic, hierarchical structure of categories further enhances usability, making the Inventory Application a critical component of the Embedded Lab Manager.

The following screenshot (Picture 6) demonstrates the inventory index page of the Embedded Lab Manager. This page allows users to browse, search, and filter items by category or subcategory. Students can view detailed information about inventory items, including their specifications, availability, and location within the lab, all from a single unified interface.



Picture 6. Embedded Lab Manager inventory page.

The next screenshot highlights the item detail view, which displays additional information about a selected inventory item. This includes specifications, location details, and an image of the item to help users quickly identify it in the laboratory. This feature enhances usability by providing all relevant information without requiring multiple page navigations.



Picture 7. Inventory item detail page.

4.4 AI Server and Resources Applications

4.4.1 AI Server Application

The AI Server application provides intelligent assistance to students by leveraging an AI-powered system to enhance their learning experience within the embedded lab environment. The application enables students to interact with an AI model to receive guidance on lab assignments, embedded concepts, and electronics-related topics.

4.4.1.1 Design and Architecture

Unlike other applications in the Embedded Lab Manager, the AI Server application does not require a database model. As a result, there are no implementations in models.py or admin.py. Instead, the core functionality is built around handling user interactions via views, templates, and a dedicated communication script.

The AI server operates on LocalAI, an AI framework running in a Docker container within the embedded laboratory's infrastructure at Turku University of Applied Sciences. The LocalAI instance includes a range of pre-installed models, such as:

- Language models: codestral-22b-v0.1, gpt-4, meta-llama-3.1-8b-instruct
- Vision models: gpt-4-vision-preview, llava-v1.6-7b-mmproj-f16.gguf
- Audio Models: tts-1, whisper-1
- Utility Models: jina-reranker-v1-base-en, text-embedding-ada-002, stablediffusion

The technical details of LocalAI are beyond the scope of this thesis; instead, the focus is on the integration of AI functionalities into the Embedded Lab Manager.

4.4.1.2 Separation of Concerns: AI Communication Script

To enhance modularity and flexibility, the AI server communication logic is implemented separately in a script named local_ai_communicator.py, located in the project's src directory at the same level as the core applications. This separation allows:

- Easier maintainability and updates to AI communication without affecting the web application structure.
- The possibility of replacing LocalAI with alternative frameworks (such as OpenAI, Claude, or Gemini) in the future.
- Independent development and improvements to AI interactions without interfering with the web interface.

The script handles key functionalities such as:

1. Receiving user input.
2. Sending queries to the LocalAI server.
3. Processing the AI-generated response.
4. Returning meaningful responses to be displayed in the web application.

4.4.1.3 Integration with Embedded Lab Inventory

One of the major benefits of integrating AI into the system is its ability to provide personalized responses based on inventory data. To achieve this, a connection between the AI system and the project's database was established. This allows the AI to:

- Provide guidance on laboratory assignments while also informing students of available resources in the laboratory.
- Suggest specific components (e.g., microcontrollers, sensors) along with their locations in the inventory.

For example, if a student inquires about a laboratory assignment involving LED control with a microcontroller, the AI response not only provides theoretical guidance but also informs the student of the available microcontrollers and their exact locations in the lab.

4.4.1.4 Keyword Extraction Approach

To enable the AI to retrieve inventory-related data, keyword extraction plays a crucial role in processing user queries. The implemented approach uses basic pattern matching with predefined keywords, where an inventory_related_keywords list is defined. If any keyword in the user's query matches an entry in this list, the AI retrieves relevant inventory information.

This basic approach was chosen for its simplicity and ease of implementation; however, it has some limitations in handling synonyms, plurals, or complex queries. More advanced approaches that could be explored in the future include:

- Natural Language Processing (NLP) Techniques:
  - Tokenization and Lemmatization (NLP preprocessing)
  - Synonym Matching via Word Embeddings (e.g., Word2Vec, GloVe)
  - Named Entity Recognition (NER)
  - Intent Recognition via NLP
- Machine Learning Models:
  - Fine-tuning large models like BERT, GPT, or RoBERTa for keyword extraction.
  - Utilizing NLP libraries such as spaCy or NLTK for simpler keyword identification tasks.

While these advanced approaches are beyond the scope of this thesis, they present opportunities for future improvements, possibly as part of student projects in the Application Programming course.

4.4.1.5 User Interface Design

The AI Server application follows a simple and distraction-free design, displaying the user's query and the AI response on a single page (ai-server.html). This single-page design provides a clean and efficient user experience by:

- Reducing unnecessary navigation between pages.
- Allowing students to focus on their interactions with the AI without distractions.

4.4.1.6 Application Integration

Following the general development workflow outlined in Chapter 4, the AI Server application was integrated into the project by:

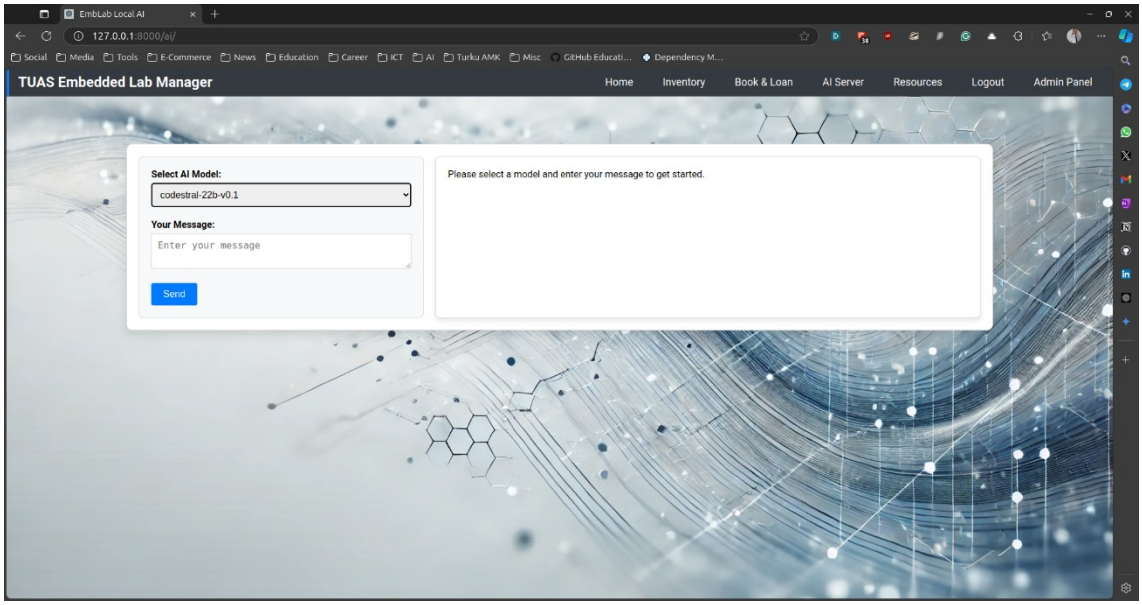
1. URL Routing:

    - A single URL pattern was added to aiserver/urls.py to route requests to the view function.
    - The main project URL configuration (emblab/urls.py) was updated to include the AI Server application.

2. Registration in settings.py:

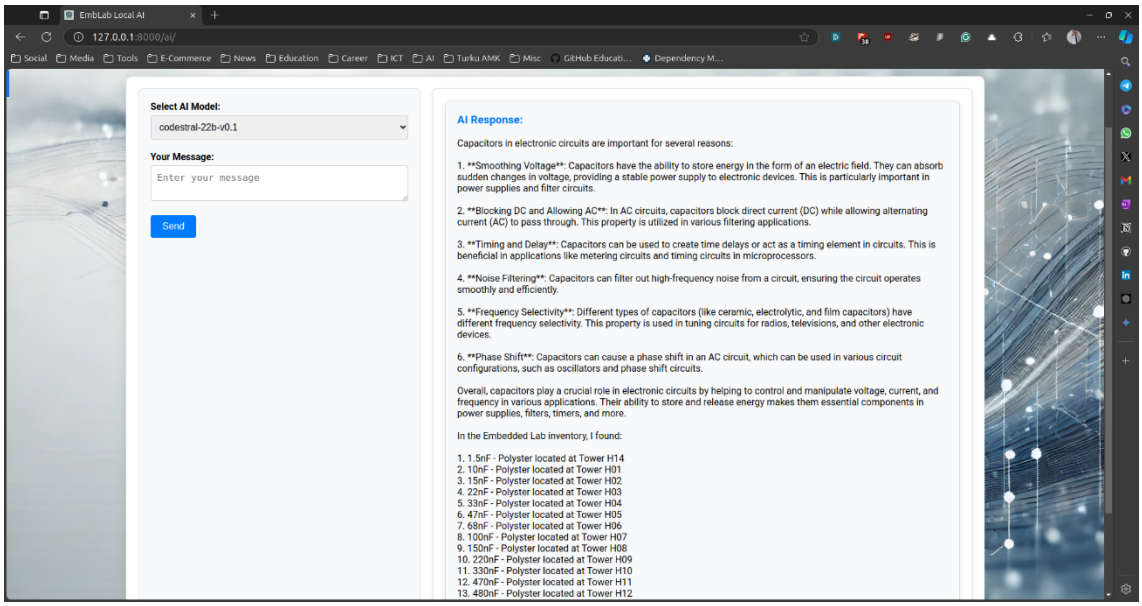
    - The AI Server app was added to the INSTALLED_APPS section, enabling Django to recognize the application.

The following screenshot (Picture 8) displays the AI Server page, where users can interact with the embedded AI assistant. The page allows students to input their queries related to lab assignments, equipment, or embedded system concepts. The simple layout minimizes distractions, focusing on the interaction between the user and the AI.



Picture 8. Embedded Lab Manager AI Server page.

The next screenshot shows an example of the AI Server response. The AI processes the user's query and provides relevant guidance, which may include suggestions for lab work, explanations of embedded concepts, or inventory details. This integration aims to enhance the learning experience by offering timely and context-aware support.



Picture 9. Embedded Lab Manager AI Server response.

4.4.2 Resources Application

The Resources application was developed to centralize all learning and teaching materials in a single, easily accessible location. Previously, these resources were scattered across multiple digital platforms, such as GitLab, Microsoft Teams course channels, and itsLearning. By consolidating these materials within the Embedded Lab Manager, both students and teaching staff can benefit from improved organization, easier updates, and enhanced accessibility.

Having a dedicated application for managing educational resources ensures that updates or modifications can be made efficiently. It also allows for the replacement of outdated materials and the resolution of broken links, creating a streamlined experience for students seeking learning materials and references.

4.4.2.1 Data Models

The Resources application consists of two primary data models: ResourceCategory and Resource, both of which provide a structured way to organize and manage educational content.

1. ResourceCategory

   The ResourceCategory model serves a similar purpose to the Category model in the Inventory application by providing hierarchical categorization for resources.

   - Attributes

     - name: The name of the category (e.g., "Electronics").
     - parent_category: A self-referential foreign key that enables nested categories.
     - slug: A URL-friendly identifier for meaningful resource URLs.

- Use Case:

  This hierarchical structure allows resources to be categorized logically, making it easier for students to locate materials. For example:

  Electronics → Power Electronics → Power Electronics Resources

2. Resouce

   The Resource model stores individual resource details and their classification under specific categories.

   - Attributes:

     - title: The title of the resource
     - category: A foreign key referencing ResourceCategory for classification
     - type: A choice field offering predefined resource types such as: Video, Blog, Book, Article, Course, Tutorial, Documentation
     - link: A web address pointing to the resource.
     - description: A brief overview or summary of the resource, highlighting its content, pros, cons, relevance.
     - slug: A URL-friendly string to generate meaningful links for resources.

4.4.2.2 Admin Panel Configuration

Registering both models (ResourceCategory and Resource) in the Django admin panel provides staff with an efficient way to manage resources. The admin interface enables:

- Listing: Viewing resources categorized by titles, types, and descriptions.
- Filtering: Narrowing down resources based on type or category.

- Searching: Locating specific materials using titles or descriptions.
- Updating: Editing and replacing outdated links or adding new content.

This functionality empowers teaching staff to maintain an up-to-date repository of educational materials efficiently.

4.4.2.3 User Interface Design

The Resources application follows the same single-page approach as the Inventory application to minimize distractions and provide an intuitive browsing experience. By consolidating navigation and content into a single page, the design ensures that students can efficiently find the resources they need without unnecessary complexity.

4.4.2.4 Application Integration

The process of integrating the Resources application into the Embedded Lab Manager followed the standard approach used for other applications:

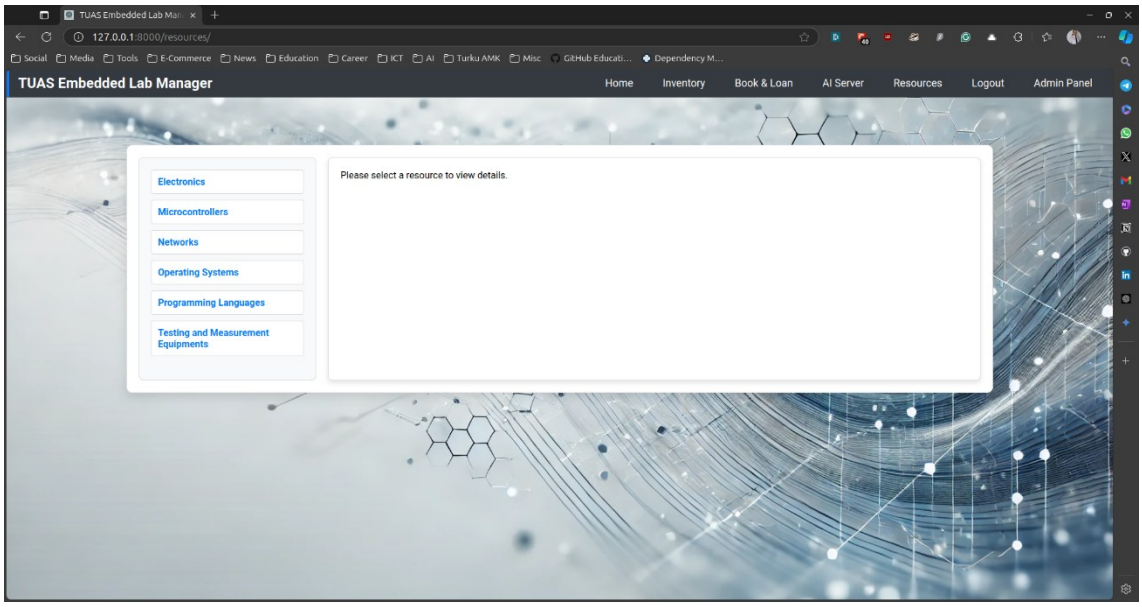
1. URL Routing:

    - Defined a single URL pattern in resources/urls.py to map the resource_list view to the appropriate endpoint.
    - The project's main urls.py file (emblab/urls.py) was updated to include the Resources application routes.

2. Registration in settings.py:

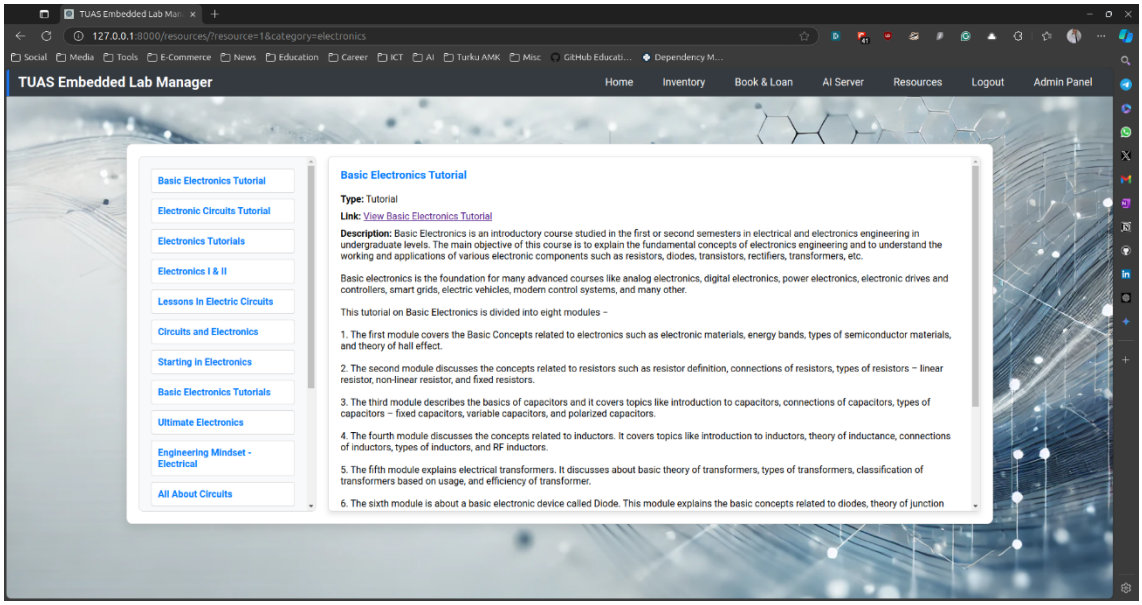
    - The Resources application was added to the INSTALLED_APPS list, ensuring its recognition by Django.

The screenshot below shows the resources page, where users can browse various learning and teaching materials categorized for easy navigation. The page enables both students and staff to quickly access tutorials, documentation, and other helpful resources, centralizing essential materials in one location. The similar structure and design of the resources page, along with the inventory and AI server pages, provides a consistent user experience across the entire application, ensuring familiarity and ease of use.



Picture 10. Embedded Lab Manager resource page.

The following screenshot (Picture 11) illustrates the resource item detail page, which provides further information about a selected resource. This includes the resource title, category, type (e.g., video, tutorial, or article), a description, and a direct link. By organizing detailed resource information in this way, the application enhances the accessibility of learning materials for users.



Picture 11. Embedded Lab Manager resource item detail page.

4.5 BookLoan Application

The BookLoan application was developed to facilitate the borrowing of laboratory inventory items and booking of tables for personal studies and laboratory exercises. This functionality not only provides students with better access to laboratory resources but also enables teaching staff to monitor item usage and table availability effectively. The application helps in planning and optimizing the utilization of laboratory resources, contributing to a well-organized learning environment.

4.5.1 Data Models

The BookLoan application comprises three main data models: Table, TableBooking, and LoanItem. These models collectively support the booking of tables and the loaning of inventory items, ensuring that students can easily reserve laboratory resources as needed.

1. Table

   The Table model stores information about bookable tables available in the laboratory. Different tables provide different functionalities, such as built-in wall sockets or attached monitors.

   - Attributes

     - number: A unique identifier for the table
     - location: The table's physical placement in the laboratory.
     - capacity: Maximum number of students who can use the table simultaneously.
     - description: Additional information about the table's specifications and intended usage.
     - is_active: A boolean indicating whether the table is currently available for booking.
     - bookable_start_time: The earliest time a table can be booked.
     - bookable_end_time: The latest time a table can be booked.

2. TableBooking

   The TableBooking model tracks student bookings of tables and provides administrative oversight.

   - Attributes

     - table: A foreign key linking the booking to a specific Table.
     - user: The user making the booking
     - booking_start: The start date and time of the booking.
     - booking_end: The end date and time of the booking.

- is_confirmed: A boolean indicating whether the booking is approved by the teaching staff. It is set to True by default but can be canceled if necessary.

3. LoanItem

The LoanItem model facilitates tracking of inventory items borrowed by students. Unlike tables, inventory items are defined in the InventoryItem model from the Inventory application, and only their loaning process is handled here.

- Attributes

  - item: A foreign key referencing an InventoryItem.
  - user: The student borrowing the item.
  - loan_start: The date when the loan starts.
  - return_due: The due date by which the item must be returned.
  - Is_confirmed: A boolean flag indicating whether the loan has been approved by an administrator. It is set to False by default.

4.5.2 Admin Panel Configuration

Registering Table, TableBooking, and LoanItem models in the Django admin panel allows teaching staff to efficiently list, filter, and search items based on various criteria, such as table number, location, capacity, loan start, return due date, and confirmation status. This capability provides a streamlined way for staff to oversee resource usage.

4.5.3 User Interface Design and Navigation

The BookLoan application follows a single-page display approach, minimizing distractions and providing an intuitive interface for students. The interface consists of:

- Table Booking Section (Left Side):

  Tables are displayed in a color-coded 3x3 grid, representing the nine bookable tables.

  - Green: Indicates available booking slots.
  - Red: Indicates fully booked tables

- Loanable Items Section (Right Side):
  Items available for loan are displayed in a similar style to the Inventory application.

- Booking and Loaning Form (Bottom Section):

  When a table or an item is selected, additional booking/loaning options appear, including calendar widgets to select start and end dates.

- User Dashboard Links (Top Right Corner):

  Authenticated users can view "My Loans" and "My Bookings" pages to track, edit, or cancel their reservations.

4.5.4 Views and Templates

The BookLoan application contains 11 view functions, handling table availability, loan processing, and user-specific interactions:

1. Table Availability Views

    - get_table_availability_view: Retrieves available time slots for a specific table.
    - get_table_availability: A helper function for calculating availability slots.

2. Booking and Loan Views

    - bookloan_view: Handles booking and loaning of tables and items.
    - booking_list: Displays a list of bookings for the logged-in user.
    - booking_detail: Provides detailed information about a specific booking.
    - loan_detail: Provides detailed information about a specific loan.

3. Booking and Loan Management Views

    - edit_booking: Allows users to modify an existing booking.
    - cancel_booking: Enables users to cancel a booking.
    - edit_loan: Allows users to modify an existing loan.
    - cancel_loan: Enables users to cancel a loan.
    - return_loan: Allows users to mark a loan as returned.

Templates and Stylesheets:

The BookLoan application uses the following templates and CSS files to provide a clean and structured interface:

Templates:

- bookloan.html – Main page for booking tables and loaning items.
- booking-list.html – Displays the user's bookings.
- edit-booking.html – Allows editing of a booking.

- loan-list.html – Displays the user's loaned items.
- edit-loan.html – Allows editing of a loan.

Stylesheets:

- bookloan.css – Styles for the main booking page.
- booking-loan-list.css – Styles for listing bookings and loans.
- booking-loan-edit.css – Styles for the edit pages.

4.5.5 Application Integration

The BookLoan application was integrated into the project following the same process used for other applications:

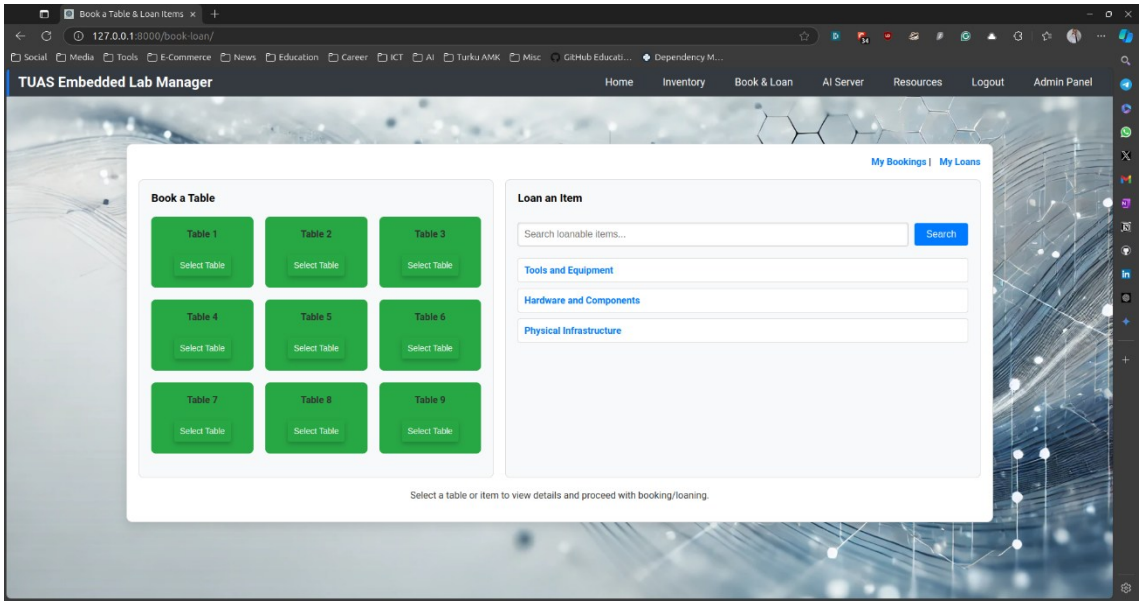
1. URL Routing

    - Multiple URL patterns were defined in bookloan/urls.py to handle various interactions.
    - The project's main urls.py file was updated to include the BookLoan application URLs.

2. Registration in settings.py

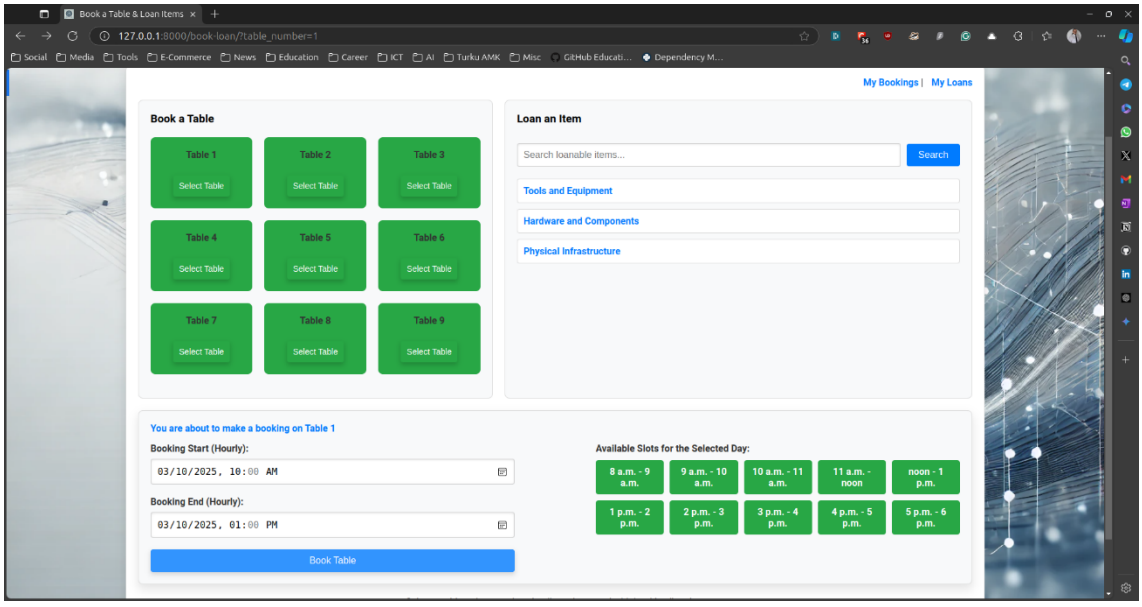
    - The BookLoan application was added to the INSTALLED_APPS list to ensure recognition by Django.

The following screenshot (Picture 12) shows the Book & Loan page, where students can view bookable lab tables and loanable inventory items. The page features a clean interface that allows users to check table availability through color-coded indicators and browse inventory items available for loan. This layout ensures that users can quickly find and reserve resources for their lab activities.



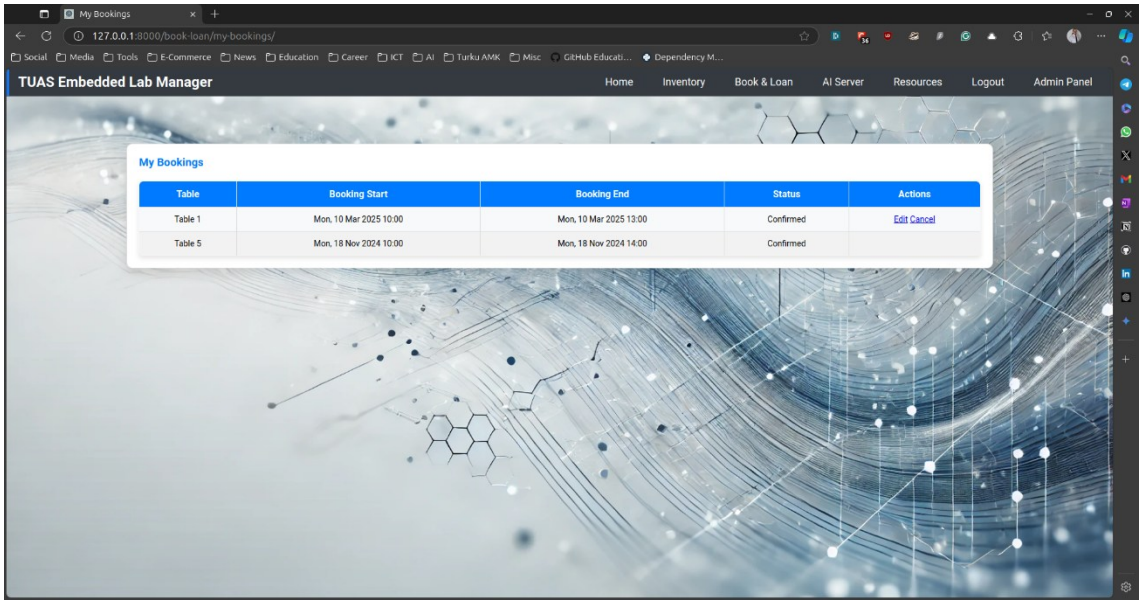
Picture 12. Embedded Lab Manager Book & Loan page.

The screenshot below demonstrates the table booking details interface. This section provides users with additional information about the selected table, including its availability and booking form. Users can specify booking start and end times using the calendar interface, simplifying the reservation process.



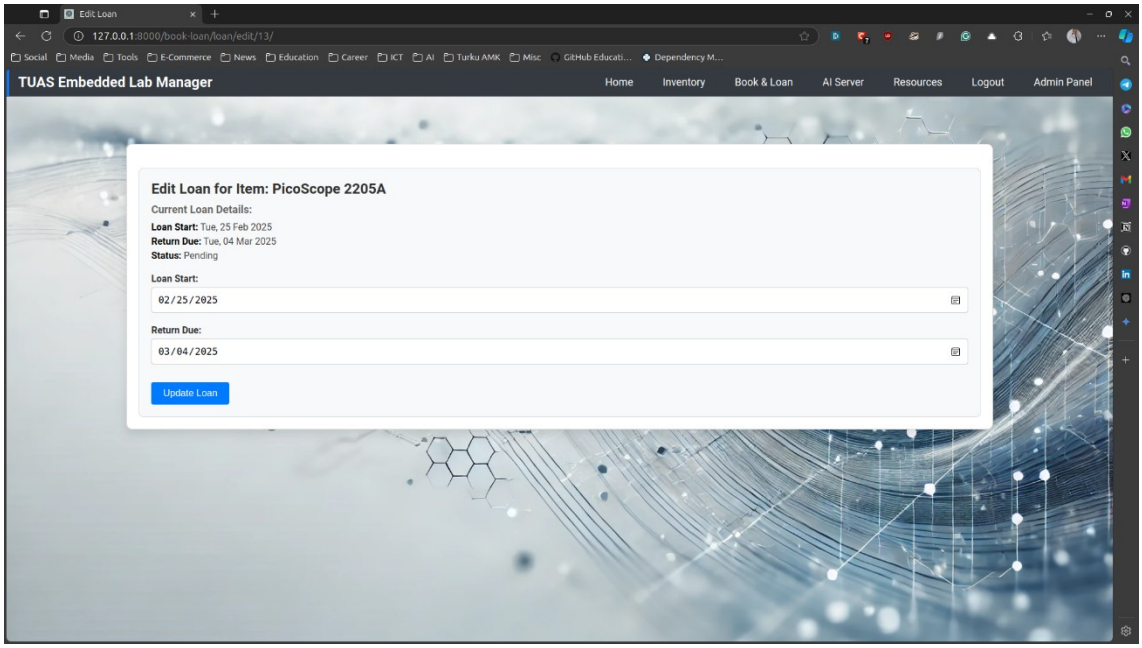
Picture 13. Embedded Lab Manager booking table details.

The following screenshot shows the Booking List page, where users can view a list of all their bookings, both active and past. This feature allows users to easily track their reservations and stay informed about their upcoming or completed bookings.



Picture 14. Embedded Lab Manager My Bookings page.

The final screenshot displays the Edit My Loan page (Picture 15), where users can view and modify loan details for an item they have borrowed. This page allows students to update return dates or cancel their loans. The interface is designed to provide clarity and control over active loans. While this example focuses on loan editing, the Edit My Booking page offers a similar structure, ensuring a consistent experience across both loan and booking functionalities.



Picture 15. Embedded Lab Manager modifying a loan.

5 Verification

Verification ensures that the system meets the defined requirements through testing and validation methods. This chapter presents an overview of how the system was tested and verified.

5.1 Counter Requirements Part

This section presents the achievements and proofs of fulfilling the implemented system requirements (Table 1).

Table 1. Requirement fulfillment status

Requirement	Status	Verification
Inventory item management	Fulfilled	Picture 6 – Picture 7
Loan tracking system	Fulfilled	Picture 12 – Picture 15
Table booking functionality	Fulfilled	Picture 12 – Picture 13
AI Integration	Fulfilled	Picture 8 – Picture 9
User authentication and authorization	Fulfilled	Picture 4 – Picture 5
Learning Aids	Fulfilled	Picture 10 – Picture 11

5.2 Test Results and Documentation

5.2.1 Functional Testing

- Ensuring all features work as intended by performing manual tests.
- Example: Testing item loan and return workflow.

5.2.2 User Interface Testing

- Screenshots of key functionalities demonstrating system behavior.

- Evaluating UI responsiveness across devices.

5.2.3 Ad-hoc Testing

- Randomized testing of different features to check for unexpected behavior.
- Focus on user interactions and performance aspects.

The verification phase demonstrates that the Embedded Lab Manager meets the defined requirements and functions as expected. The system's functionality, performance, and usability have been validated through testing processes and continuous feedback from stakeholders.

6 Conclusion

This chapter provides an overview of the key achievements of the Embedded Lab Manager project, evaluates the effectiveness of the work carried out, identifies the challenges faced, and presents recommendations for further development and improvement.

6.1 Summary of Achievements

The Embedded Lab Manager project successfully achieved main objectives of the thesis by implementing an integrated inventory management system with features that track loaned items. This functionality enables both students and teaching staff to monitor and manage resources efficiently. Additionally, a table booking system was developed to optimize the use of laboratory space, making it easier for students to reserve workstations for their studies and laboratory exercises.

A notable feature of the system is the integration of AI-powered assistance, designed to provide students with relevant information about laboratory resources and support their learning experience. The project also employed a modular architecture, ensuring that the system is maintainable and scalable for long-term use. Finally, the system demonstrated tangible improvements in resource management, providing teaching staff with valuable insights for planning and organizing lab activities.

6.2 Limitations and Challenges

Despite its successes, the project encountered several limitations and challenges. One of the primary areas for improvement lies in the AI search functionality, which currently relies on a basic keyword-matching approach. While functional, this method could be enhanced through more advanced natural language processing (NLP) techniques to improve contextual accuracy.

Additionally, the system does not provide real-time notifications for important events, such as resource updates or upcoming due dates for loans.

Performance optimization is another area requiring attention, particularly when handling large datasets or serving multiple simultaneous users under high load. Furthermore, the system lacks automated reporting features, which could assist administrators in generating insights on resource utilization without manual data extraction.

6.3 Future Recommendations

Several recommendations are proposed to address the limitations and enhance the system's overall functionality. To improve AI integration, it is suggested to explore machine learning models that can better understand user queries and provide contextually relevant responses. Expanding AI integration to other system features, such as dynamically identifying available table bookings or assisting users in finding learning materials, could further improve the user experience.

Developing a mobile-friendly version of the application is also recommended to provide greater accessibility and convenience for users. Additionally, implementing automated alerts and notifications would help ensure that users are promptly informed of key events related to inventory and bookings. Expanding the system's support to accommodate multiple laboratory locations and cross-department collaboration would increase its scalability and utility across the institution. Lastly, conducting regular user studies is essential for gathering feedback from students and staff, which can guide future refinements and ensure that the system continues to meet evolving needs effectively.

References

Django introduction (n.d.) MDN Web Docs. Available at: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django/Introduction (Accessed: 29 January 2025).

Django Software Foundation (n.d.) Django Web Framework. Available at: https://www.djangoproject.com/ (Accessed: 29 January 2025).

Docker Inc. (n.d.) Docker: Enterprise Container Platform. Available at: https://www.docker.com/ (Accessed: 29 January 2025).

Mele, A. (2024) *Django 5 By Example - Fifth Edition: Build Powerful and Reliable Python Web Applications from Scratch*. 5th edn. Packt Publishing.

Peters, T. (2004) PEP 20 – The Zen of Python, Python Enhancement Proposals (PEPs). Available at: https://peps.python.org/pep-0020/ (Accessed: 29 January 2025).

Prasad, P.J. and Bodhe, G.L. (2012) 'Trends in laboratory information management system', Chemometrics and Intelligent Laboratory Systems, 118, pp. 187–192. doi:10.1016/j.chemolab.2012.07.001.

Python Software Foundation (n.d.a) Python Package Index (PyPI). Available at: https://pypi.org/ (Accessed: 29 January 2025).

Python Software Foundation (n.d.b) Python Programming Language. Available at: https://www.python.org/ (Accessed: 29 January 2025).

Appendices

Appendix 1. Setting Up the Development Environment

Before setting up the Embedded Lab Manager, it is essential to have Python installed on the system. This section provides basic instructions for installing Python on different operating systems and then continues with the steps to set up the development environment.

For Linux (Ubuntu/Debian):

Python can be installed by running the following commands in a terminal window:

- sudo apt update
- sudo apt install python3 python3-pip -y

Installation can be verified by running:

- python3 --version
- pip3 --version

For Windows/MacOS:

Python can be downloaded and installed using the official installer from the Python website.

- For Windows: https://www.python.org/downloads/windows/
- For MacOS: https://www.python.org/downloads/macos/

Installing python on Windows and MacOS platforms are done by simply following:

- Download the latest stable version (Python 3.x).

- Run the installer and ensure checking the "Add Python to PATH" option before proceeding.
- Complete the installation by following the prompts.
- Verify the installation by opening Command Prompt (Windows) or Terminal (MacOS) and running:
    - python –version
    - pip –version

The development environment for the Embedded Lab Manager was set up using Linux, leveraging Python's venv module for virtual environments and Docker for deployment. This chapter provides an overview of the process, including setting up the virtual environment, installing dependencies, and initializing the project.

Detailed, platform-specific setup instructions can be found in the project's repository README files on GitLab and GitHub.

GitLab: https://git.dc.turkuamk.fi/embo/emblab-manager-webapp

GitHub: https://github.com/urasayanoglu/emblab-manager-webapp

Cloning the Repository

Project files can be obtained by cloning the repository from one of the repository link listed below using either SSH or HTTPS .

Clone with SSH:

1. **git@git.dc.turkuamk.fi:embo/emblab-manager-webapp.git**
2. **git@github.com:urasayanoglu/emblab-manager-webapp.git**

Clone with HTTPS:

1. **https://git.dc.turkuamk.fi/embo/emblab-manager-webapp.git**
2. **https://github.com/urasayanoglu/emblab-manager-webapp.git**

Setting Up the Virtual Environment

A virtual environment was created using Python's built-in venv module to isolate the project's dependencies from system-wide installations. This approach ensured that the development process remained consistent and avoided potential conflicts with other Python projects.

The virtual environment is initialized with the following command:

- For Linux/MacOS:
  python3 -m venv emblab_env
- For Windows:
  python -m venv emblab_env

After creating the virtual environment, it is activated using:

- For Linux/MacOS:
  source emblab_env/bin/activate
- For Windows:
  ./emblab_env/Scripts/activate

The virtual environment provided a clean workspace for installing and managing project's dependencies.

Installing Project Dependencies

Once the virtual environment was activated, the required Python packages were installed using the requirements.txt file. This file lists all dependencies necessary for the project, ensuring a reproducible setup.

The following command was used to install the dependencies:

- For Linux/MacOS/Windows:
  pip install -r requirements.txt

Key dependencies include:

- Django: The primary framework for developing the web application.

- Pillow: For handling image uploads and processing.
- Gunicorn: For deploying the application in a production environment.

The requirements.txt file in the project repository serves as a complete reference for the packages and their versions.

Starting the Project

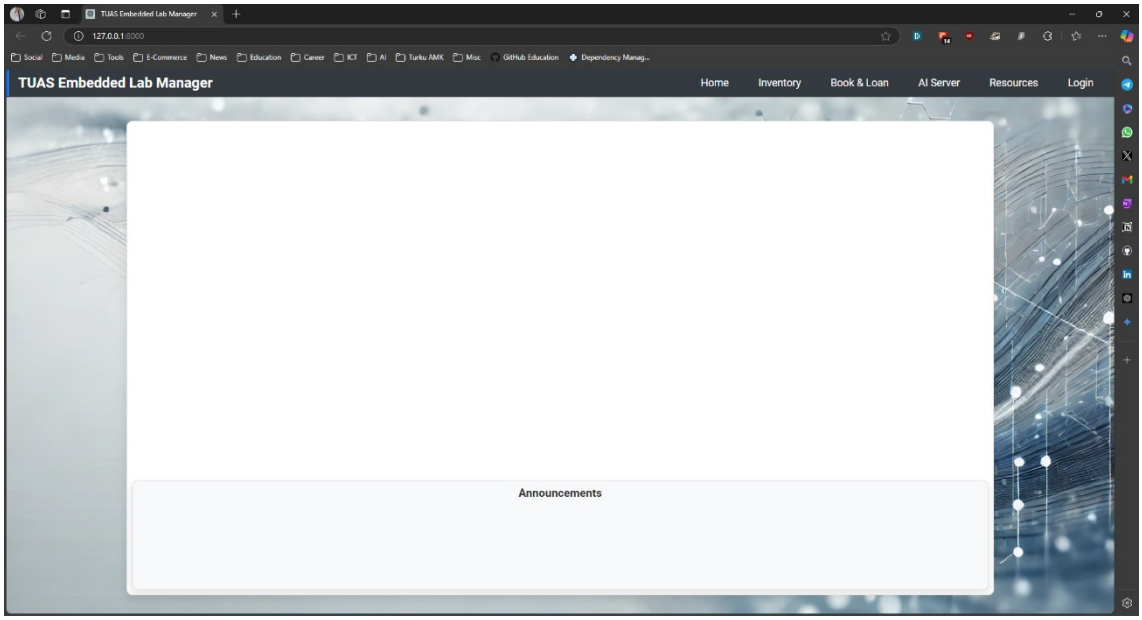
After setting up the virtual environment and installing the requirements, the project was initialized by running the Django development server. To start the development server, following commands were used:

1. Create a superuser for administrative access:
    - python manage.py createsuperuser

    After running the code above, details for the superuser will be asked.

    1. Username: A unique username.
    2. Email address: Optional.
    3. Password: It will be asked two times for confirmation.
2. Apply database migration to set up the required tables:
    - python manage.py makemigrations
    - python manage.py migrate
3. Start the development server:
    - python manage.py runserver

Once the server is running, the application can be accessed in a web browser via http://localhost:8000 Accessing the local server URL (e.g., http://127.0.0.1:8000) allowed the development version of the Embedded Lab Manager to be tested in a browser. The screenshot below (Picture 16) demonstrates the Embedded Lab Manager server running on localhost.



Picture 16. Django development server running.

After running the development server, the Embedded Lab Manager project starts with no inventory items, bookable tables, resources, or background images pre-registered. To fully configure the system, users need to log into the admin panel using the username and password created during the setup process. Admin panel can be reached through localhost:8000/admin/

Once logged in, administrators can access the Site Administration section of the admin panel. From there, inventory items, tables for booking, resources, and background images for the landing page can be added or updated. This administrative interface allows for easy management of the system's data and ensures that users have access to the necessary resources for lab activities. By populating these sections, the system becomes fully functional and ready for use by students and teaching staff.

Appendix 2. Deployment Using Docker

This appendix provides guidance on deploying the Embedded Lab Manager using Docker. It covers the essential concepts, the purpose of different Docker components, and step-by-step instructions to run the application. Detailed implementation of writing Dockerfiles and Compose files is out of the scope of this thesis; however, an overview of their functionality and usage is provided.

Installing Docker

Before deploying the Embedded Lab Manager, Docker must be installed on the system. Below are installation instructions for Linux. For Windows and macOS, Docker Desktop can be downloaded from the Docker website.

- Linux (Ubuntu/Debian):

  Running the below commands on Linux terminal will install Docker.

  1. sudo apt update
  2. sudo apt install docker.io -y
  3. sudo sysyemctl start docker
  4. sudo systemctl enable docker

  To verify that Docker is installed correctly, run the following command:

  docker –version

For Windows and MacOS, Docker can be downloaded and installed by following the instructions provided in Docker website.

Understanding Dockerfile and compose.yaml

Docker provides two primary mechanisms for containerizing applications: the Dockerfile and the Compose file (compose.yaml). Both serve different purposes in the deployment process.

1. Dockerfile

   Dockerfile is a script that contains a set of instructions to build a Docker image. It defines:

   - The base image (e.g., python:3.10-slim in this project).
   - Required dependencies.
   - Environment variables.
   - Commands to prepare and start the application.

   The Dockerfile provided in the repository sets up a Python environment, installs dependencies, configures the database, and prepares static files for deployment.

2. Compose.yaml

   It is used to define and run multi-container applications. It allows specifying multiple services, such as the web application and the database, in a single file. It simplifies container orchestration, enabling easy management of interconnected services. In this project, the compose.yaml file defines:

   - The web application service – Docker file is used to build its image
   - The PostgreSQL database service – Official PostgreSQL Docker image

Why Both Dockerfile and Compose.yaml Are Needed

The Dockerfile provides a step-by-step guide to build a single service container, which ensures that the application is set up consistently across different environments.

The Compose file facilitates the management of multi-container environments, automating the deployment of dependent services (such as the database) alongside the web application.

Using both together offers a flexible approach—allowing for easier testing with standalone containers and full-scale deployment with multiple services.

Running the Docker Containers

Once Docker is installed and the necessary files (Dockerfile and compose.yaml) are set up, the application can be started by following these steps:

1. Navigate to the Project Directory:

   On terminal (Linux, MacOS) or command prompt (Windows) navigate to the project directory containing the Dockerfile and compose.yaml files.

2. Build and Start the Containers:

   Run the following command to build the Docker images and start the containers:
   docker compose up –build
   The --build flag ensures that the image is rebuilt based on the latest changes in the Dockerfile.

3. Running in Detached Mode:

   To run the containers in the background, use:
   docker compose up -d

4. Accessing the Application:

   Once the containers are running, the application will be accessible at:
   http://localhost:8000

5. Stopping the Containers:

   To stop the running containers, use:
   docker compose down

Serving Static Files in Production

In a production environment, it is recommended to use a dedicated service to serve static files separately from the Django application server (Gunicorn). Django's built-in development server is not optimized for serving static content in production.

Why Separate Static File Service?

1. Improves performance by offloading static file handling to a lightweight server like Nginx.
2. Enhances scalability, allowing the web server to focus on processing dynamic content.
3. Optimizes caching mechanisms for frequently accessed static assets.

Running the Static File Service

A separate container needs to be created to serve static files using Nginx or a similar service. This new service should be added to the compose.yaml file, allowing Docker Compose to manage the static file service alongside the other services. Once configured, the static files service can be built and started by running: docker compose up -d

Final Notes on Deployment

By using Docker and Docker Compose, the Embedded Lab Manager can be deployed consistently across various environments. The containerization approach simplifies dependency management, service orchestration, and system scalability. Docker Compose automates the process of managing multiple services, including the web application, database, and static file server, ensuring that each component operates in isolation yet seamlessly communicates with the others.

For further customization and troubleshooting, users may refer to the official documentation for Docker and Docker Compose. These resources provide detailed instructions on container configuration, performance optimization, and advanced deployment strategies.

This deployment guide aims to provide a clear foundation for running the Embedded Lab Manager, enabling developers to focus on enhancing the system’s features without being hindered by complex environment setups.