OULUN AMMATTIKORKEAKOULU

INVENTORY MANAGEMENT SYSTEM FOR ROBOTIC LABS

A User-centred Web-Based Solution

Purity Turunen (Purity Turunen)
Bachelor's Thesis
Spring 2025
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree programme name: Information Technology

Author: Purity Turunen
Title of the thesis: Inventory Management for Robotic Labs
Thesis examiner(s): Meija Lohiniva
Term and year of thesis completion: Spring, 2025
Pages: 51

This project describes the development of a web-based inventory management system aimed at increasing efficiency of locating and managing components in a robotics lab. Motivated by the challenges encountered at the Oulu University Robotics Lab, where equipment tracking relied on a manual, Excel-based system; the project enhances accessibility and streamlines administrative tasks.

Principles of database design, inventory control, and user-centred interface development form the foundation of the system. It provides real-time item tracking through a searchable interface and displays accurate component positions on a digital floor layout. Developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js), the application includes features such as role-based access control and an easy-to-use user interface.

The development process also provided practical knowledge of full-stack web development and agile software design in an academic setting. The project improved knowledge of using user-centred design concepts in software solutions for specific environments and promoted practical problem-solving abilities.

The system’s functionality was evaluated using Jest and Postman, demonstrating stable performance and scalability in academic settings. This thesis outlines the design and development process, evaluates the system’s effectiveness, and demonstrates how web-based tools can improve resource management in educational robotics labs.

CONTENTS

GLOSSARY

API	Application Programming Interface
CDN	Content Delivery Network
CI/CD	Continuous Integration / Continuous Deployment
CSS	Cascading Style Sheets
ERD	Entity-Relationship Diagram
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
IMS	Inventory Management System
JWT	JSON Web Tokens
MERN	MongoDB, Express.js, React.js, and Node.js
ODM	Object Data Modelling
ORM	Object-Relational Mapping
RLIMS	Robotic Lab Inventory Management System
SPA	Single-Page Applications
UCD	User-Centred Design
UI	User Interface
UML	Unified Modelling Language
UX	User Experience

1 INTRODUCTION

An inventory management system (IMS) integrates technology and standardized processes to manage the tracking, upkeep, and organization of various lab assets. These systems are essential for managing resources such as educational tools, research equipment, and specialized apparatus, ensuring that the institution remains efficient and organized (1). In the context of a Robotic Lab Inventory Management System (RLIMS) at an educational institution, where the availability and accuracy of resources are important, a well-designed inventory management system becomes indispensable for smooth operations and preventing operational delays.

Moreover, as noted by (1), an effective inventory management system plays a key role in organizing and tracking lab resources to maintain order and ensure efficient operations. It allows lab administrators to maintain a centralized record for each asset, including its name, category, location, usage details, and status (whether used or returned). Moreover, the system tracks the available quantity, ensuring that the required tools are accessible when needed. It also reduces the possibility of equipment misplacement, loss, or delays, which could typically interrupt lab activities.

In dynamic environments like robotic labs, real-time tracking of assets is particularly important. A well-functioning inventory management system allows administrators to instantly access up-to-date information on each asset's status and availability. The feature reduces interruption and improves research and teaching outcomes by simpler allocating in-demand equipment and ensuring its availability whenever required. (1.)

The RLIMS will be designed with key principles to ensure not only functionality but also long-term effectiveness and ease of use. First, operability will be a top priority, ensuring that lab administrators and/or technical staff can easily monitor, maintain, and troubleshoot the system to prevent operational

disruptions. It will focus on ensuring smooth operation and ease of management without overly complex processes as it is highlighted in (2, Chapter 1).

The development of the application will emphasize simplicity, ensuring that both the system architecture and user interface allow administrators to learn and utilize the platform with minimal support. Whether interacting with the inventory database or adjusting settings, users will encounter a clear, intuitive interface that lessens training time. The same applies to the backend; that the system remains to be uncomplicated for developers to navigate and manage, reducing the complexity of upcoming updates and adjustments. (2).

In addition to operability and simplicity, evolvability will be embedded within the system's design. RLIMS is built to meet potential requirements as the robotic lab expands, and new capabilities are added. Its scalable architecture makes it possible to expand the asset's database and add new features without affecting its ability to run smoothly and reliably. This adaptability will ensure the system remains effective over time, even as the lab’s needs evolve and change (2).

1.1 Background and motivation

The need for this web-based application arose from specific challenges in managing inventory at Oulu University’s Robotics Lab. The lab, actively engaged in advanced robotics research, needs to efficiently track a wide variety of devices, components, and project-specific equipment for multiple ongoing projects. According to lab the personnel, the current inventory system lacks essential features for quickly locating items, monitoring their status, and ensuring efficient resource usage.

The complexity of inventory management is made more challenging by the need to maintain accurate, real-time records as multiple projects and experiments are conducted simultaneously. As these projects evolve, it becomes increasingly difficult to ensure that each item's location and current usage status are properly tracked. The tracking of each item's location and current usage status gets more challenging as these programs progress. This

application will enable users to look for products by name or product category and show their precise location on a floor plan to navigate these difficulties. The feature is created especially to cut down on the amount of time spent looking for tools and components, increasing lab capacity.

Additionally, once an item is allocated to a student, it will automatically be removed from the inventory list, reflecting its in-use status and any changes in available quantities. The real-time updating of item statuses is important for maintaining accurate inventory records, particularly in fast-paced, dynamic environments like research labs in learning institutions.

During inventory audits, the application will generate a list of goods that are currently on loan. This feature will simplify the tracking of borrowed items, ensuring they are accounted for and returned in a timely manner.

Implementing inventory monitoring has been demonstrated to boost operational productivity and lower errors, particularly in complicated research environments where effective tracking of a variety of items is essential, according to inventory management study (3).

This web-based application is designed not only to address the unique needs of the Robotics Lab but also to provide a scalable solution that could be applied to other research and industrial settings facing similar inventory challenges.

1.2 Problem statement

The current storage management system at the University's Robotics Laboratory relies primarily on a manual, Excel-based inventory system, which is insufficient for the lab's dynamic and resource-intensive environment. While this system lists the components in the lab, it lacks precise location information, making it inefficient and time-consuming for users to locate items. Users must either search through the storeroom or rely on a general overview of supplies that only provides an approximate idea of where items are stored. This inefficiency reflects the common inventory management challenges identified by

Muller (4), who notes that an item’s location cannot be accurately tracked and controlled, overall inventory accuracy suffers, leading to delays, low or slow productivity.

Effective component retrieval is nevertheless hampered by the lack of real-time tracking, even with efforts to identify storage with titles like "pneumatics," "electronics," and "actuators." The system also fails to track items that are borrowed and are in active use, further complicating resource management. As emphasized by Muller (4), an effective inventory system also tracks the movement of items from storage and usage, a feature notably missing in the lab’s current setup. As a result, a more advanced, dynamic solution that decreases inefficiencies at the lab, improves resource management, and offers real-time updates is required.

Large item collections are difficult to manage in research settings, such as robotics labs. There are significant operational inefficiencies when an item's physical location cannot be tracked from the time of having received until it is used and returned. Such settings frequently experience misplacement, trouble identifying specific components, and difficulties tracking borrowing and deployments, which emphasizes the importance of accurate inventory control to preserve operating efficiency (4).

Lack of a functional inventory management system presents several operational challenges:

- Difficulty in locating components: Users spend significant time searching for items due to the lack of precise location data.
- Limited functionality of the current system: The static, Excel-based system does not support new features, such as deployment tracking or interactive floor plans.
- Inadequate asset tracking for loaned items: Items borrowed for specific projects or by students are not effectively tracked, leading to potential loss or mismanagement.

- Misplacement of items: Components frequently end up in incorrect locations, wasting time and effort in searching for resources.

1.3 Objectives

The primary objective of the RLIMS is to address existing challenges in managing and tracking lab resources by providing a centralized, web-based platform for robotic lab inventory management. The platform is designed to streamline operational efficiency, reduce manual work, and support lab members in quickly locating and managing resources. The following objectives outline specific features and functionalities that address the core needs identified by the lab's supervisory personnel:

User-Friendly Interface: The system will feature an intuitive and accessible interface, ensuring it is easy to use for all lab members regardless of technical expertise. This will minimize training time and reduce errors, which align with the principles highlighted by (5, Chapter 2) which emphasize the importance of user-centred design.

Reliability: Streamlining inventory management with technology improves performance, and minimizes time spent on tasks, while offering real-time inventory visibility. A notification feature can alert administrators to low supplies, ensuring timely restocking and avoiding shortages. GeneMod highlights that such features enhance operational efficiency and prevent disruptions (3).

Floor Plan Integration: Muller (4) emphasizes the need for structured locator systems to ensure accurate item tracking and efficient retrieval. Digital floor plans help formalize item locations, track movement, and optimize space and resources; reducing time spent searching for equipment and enhancing operational efficiency.

2 USER EXPERIENCE IN INVENTORY SYSTEMS

User Experience (UX) design is central to creating effective inventory management systems, especially in specialized environments like robotics labs. UX design is not just about aesthetics or the look and feel of a system. It encompasses the entire experience users have while interacting with a product, including functionality, accessibility, and ease of use. For inventory systems, this means ensuring that users can efficiently perform tasks such as tracking, managing, and retrieving items without unnecessary cognitive or operational burdens. (6.)

Effective UX design prioritizes the why, what, and how of system use. The "why" refers to the purpose behind the system's existence; helping lab personnel organizes and locates critical components efficiently. The "what" addresses the system's features, such as search functions, categorization, and live tracking of assets. The "how" is about creating an intuitive interface that enables users to achieve their goals seamlessly. For example, robotics' lab administrator's benefit from interfaces that present inventory information clearly and minimize steps required to complete frequent tasks. (6.)

Stated by Canziba (6), UX design should be integral from the start of development. When UX is considered as an afterthought, the result is often systems that require extensive training and documentation, increasing costs and reducing efficiency. On the other hand, a well-planned UX design streamlines development and ensures the product meets user needs from the outset. This is particularly essential in environments like robotics labs, where time and accuracy are important requirements.

It is fundamental to concentrate on UX design from beginning of system development. The system may require extensive training and documentation if UX is added later, which raises costs and degrades system performance. A poorly designed user experience (UX) can lead to errors, frustrate users, and complicate the management of important resources. This typically translates

into a decrease in technology trust and an increase in support calls. However, if UX design is done correctly from early on, it can simplify development, lower training costs, increase user satisfaction, and guarantee the product satisfies user needs. (6) This is crucial in settings like robotics labs, where accuracy and speed are critical.

Embedding UX principles into the design of inventory management systems ensures that robotics labs implement solutions that are both functional and intuitive, enhancing user engagement and supporting the lab's operational goals.

2.1 Overview of inventory management systems

User-Centred Design (UCD) is a design concept that focuses the end user throughout the design process. It describes on understanding users' wants, preferences, and challenges to design solutions that improve user happiness and usability. In the context of inventory management systems for the robotic lab, applying UCD principles ensures the inventory management system for robotic lab is both functional and easy to use, supporting efficient inventory management. (5, Chapter 2.)

A key aspect of UCD for inventory management systems in robotic labs is the emphasis on user interface design and workflow integration. The system reduces cognitive strain on users by developing clear, straightforward, and user-friendly interfaces, allowing them to locate, track, and manage, view inventories efficiently. A well-designed UI allows users to complete activities with little effort, decreasing errors and promoting system adoption. This approach is consistent with UCD's main goal of developing systems that satisfy users' genuine needs, hence boosting usability and overall operational efficiency (5, Chapter 2).

Additionally, Modern inventory systems are designed to streamline workflows and simplify user interactions. For a robotics lab, an effective inventory management system should:

- Facilitate quick access to item locations and categories.
- Enable robust search and filtering options.
- Support tailored asset categorization and location tracking.

These features help users locate, manage, and track lab assets efficiently, minimizing time spent on routine tasks. A user-centred approach ensures these functionalities align with lab-specific requirements, boosting system usability and effectiveness.

Kirakowski et al. (7), mention that UCD prioritizes usability by involving users and tailoring systems to their needs. This approach incorporates:

- Understanding User Needs: Gathering insights into workflows ensures the system supports user-specific tasks while minimizing complexity (5, Chapter 2).
- Iterative Design Process: Iteration allows refinement of features through testing. While this project is solo-developed, iterative self-assessment ensures the interface aligns with anticipated user workflows (7).
- Focus on Usability and Satisfaction: A system designed with clear navigation paths, concise labelling, and intuitive workflows reduces user effort and enhances satisfaction (7).
- Function Allocation: Dividing tasks between the user and system appropriately reduces errors and improves efficiency. Automating repetitive tasks allows users to focus on critical activities (7).

Adopting UCD principles ensures the system is easy to use, reduces training time, and increases user satisfaction. This aligns with ISO 9241’s usability guidelines, emphasizing effectiveness, efficiency, and user satisfaction. (7, p. 38–42.)

2.2 Limitations of current solutions

The current inventory management system in the robotics lab relies on manual processes, specifically the Excel spreadsheets, for tracking and managing inventory. While Excel provides basic organizational tools, it is ill-suited for the dynamic and challenging needs of a robotics lab. The limitations presented in Table 1, are based on insights of Wolters (8) which outlines the drawbacks of this approach.

Table 1: Summary of Limitations in Current Inventory Management Solutions. (8.)

Limitations	Description	Implications
User Experience Issues	Excel-based inventory management systems often require complex workflows and lack seamless integration.	Increased cognitive load, overwhelming for users, and poor system adoption.
Inefficient Asset Tracking	Manual data entry and updates are time-consuming and error prone.	Increased risk of misplaced items and reduced operational efficiency.
Static Reporting Mechanisms	Relies on predefined templates, making it difficult to adapt to changing requirements.	Limits the ability to generate real-time or customized insights, slowing decision-making processes.
Limited Collaboration	Excel lacks robust collaboration features, making simultaneous updates and version control difficult.	Increased errors, difficulty in tracking changes, and inefficiencies.
High Error Rates	Prone to typographical and data integration errors as the inventory grows.	Leads to inaccuracies in inventory records,

potentially causing resource shortages or excess stock.

User Experience Issues: Excel-based solutions often require complex workflows that can overwhelm users, especially when managing large datasets. The lack of seamless integration with other systems further exacerbates this challenge, creating a fragmented user experience that increases cognitive load. (8.)

Inefficient Asset Tracking: Excel-based inventory management requires manual data entry, which is both time-consuming and error-prone. In a dynamic environment like a robotics lab, where inventory is frequently updated, this manual process increases the risk of asset misplacement and duplication. Moreover, the lack of real-time tracking features means that inventory data may quickly become outdated, further reducing efficiency. Wolters claims that, excel offers no automation for data updates, requiring planners to manually input changes, which contributes to significant inefficiencies. (8.)

Static Reporting Mechanisms: Excel relies on predefined templates for generating reports, which limits its adaptability to new or changing requirements. When new inventory categories are introduced, the need for manual updates slows down the report generation process. This lack of flexibility makes excel unsuitable for environments where quick, customized reporting is necessary for timely decision-making. (9.)

Limited Collaboration: Excel's lack of robust collaboration features restricts its ability to handle simultaneous updates from multiple users. Without real-time version control or access restrictions, errors are more likely to occur when team members make conflicting changes. Also, excel was not designed for multi-user environments, making it inefficient for teams relying on shared inventory data. (9.).

High Error Rates: As the inventory grows, maintaining accuracy in Excel becomes increasingly challenging. The absence of built-in validation tools makes it prone to typographical and data integration errors. Wolters emphasizes that spreadsheets become fragile as they scale, with small errors potentially leading to significant disruptions. (8.)

To address these issues, the proposed inventory management system applies user-centred design principles to minimize interface complexity while maintaining responsiveness to user requirements. An optimized UI architecture facilitates intuitive navigation, streamlines asset handling workflows, and contributes to improved system usability and user satisfaction.

3 SYSTEM DESIGN

The primary aspect of software development is system design, which centers on developing a software system's architecture, elements, interfaces, and data management strategies. A properly designed system promotes productivity, user satisfaction, assurances, and assists in decreasing down development expenses and time. Dhirendra & Tejas in (10, chapter 1), describes that, designing a software system involves specifying the architecture, components, modules, interfaces, and interactions to meet both functional and non-functional objectives. During this process, specifications are converted into an organized plan or outline that guides the software system's development, maintenance, and expansion.

The goal of software system design is to satisfy performance, scalability, reliability, and security requirements while making the system simple to use, administer, and improve. It should also be flexible enough to consider changes throughout time (10, chapter 1.)

Among the main benefits of well-designed systems, according to Dhirendra and Tejas (2023, Chapter 1), are:

- **Well-defined requirements:** This guarantees that the primary features, performance, and security requirements are precisely identified, resulting in an accurate solution.
- **Maintenance ease:** A well-designed system is less expensive over time and more durable since it is simpler to upgrade and maintain.
- **Performance:** It guarantees that the system runs as smoothly as possible while accounting for aspects like access, response time, and dependability.
- **Savings:** Good design lowers the possibility of errors and rework, which results in financial savings.

The RLIMS system design, based on the MERN stack (MongoDB, Express.js, React.js, and Node.js), benefits from modularity, rapid development cycles, and adaptability; key characteristics in a dynamic laboratory environment where user needs and technical requirements evolve frequently (12).

3.1 Overview of system architecture

Understanding the underlying architecture is essential for developing the RLIMS that will fulfil system requirements and ensure flexibility, maintainability, and performance. The RLIMS is based on the MERN, which provides a unified, efficient, and scalable solution for managing robotic lab assets and equipment.

3.1.1 Architecture classification

The architecture of a system can be classified into two main styles: monolithic and distributed architectures.

Monolithic Architecture

This architecture as shown in figure 1 consists of a single deployment unit where all components and functionalities are linked together. While it makes development and deployment easier, it may lead to challenges in scalability and maintainability as the application grows. For example, if a monolithic system requires an upgrade or a new feature, the entire application may need to be redeployed, increasing the risk of downtime and potential disruptions in service. This can be particularly problematic in environments like robotic labs, where continuous access to inventory data is important. (13.)

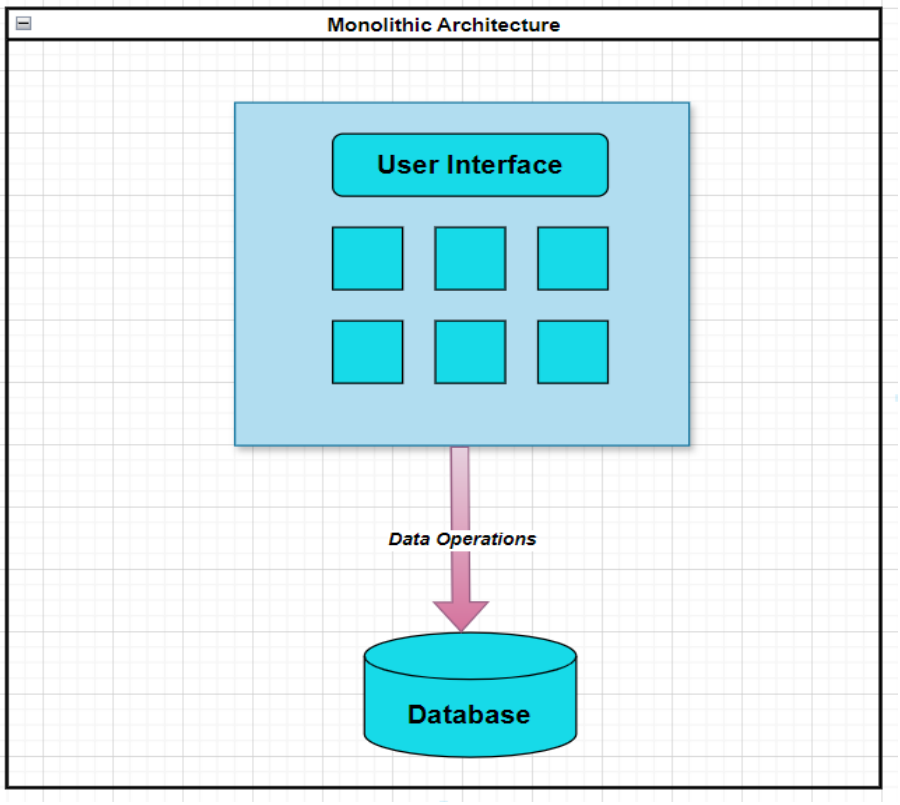

Figure 1: Monolithic architectures are single deployment units (13).

Distributed Architecture

In contrast, distributed architecture**,** as shown in Figure 2 involves multiple deployment units and often organized into micro-services or services that communicate over a network. Each service can be developed, deployed, and scaled independently. This architecture technique is known for supporting key characteristics such as resilience, scalability, and flexibility (13). These features make distributed architecture a preferred choice for modern applications like the RLIMS, which requires flexibility to accommodate growing demands and the ability to scale with ease.

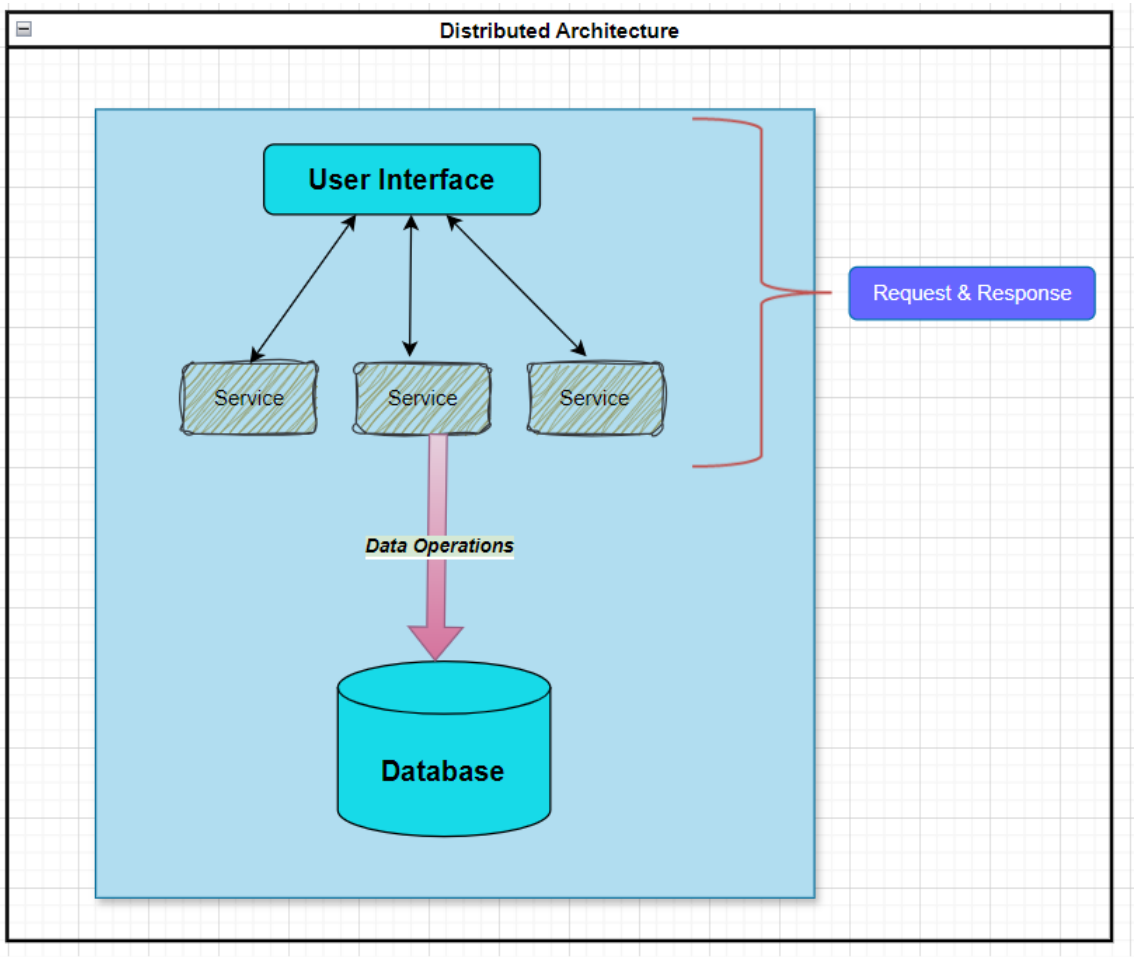

Figure 2: Distributed architectures involve multiple deployment units (13).

The distributed architecture is chosen for the RLIMS for several strategic reasons, particularly to support scalability, flexibility, resilience, real-time data processing, among other requirements that are central to effective operation in a dynamic lab environment.

Reliability in RLIMS refers to the system's ability to function correctly even when unexpected problems occur. This includes ensuring that the application remains operational despite user errors, system failures, or increased user, thereby ensuring data stays reliable and can be easily accessed. System-wide failures can be avoided, and availability can be maintained by implementing fault tolerance techniques like user permission verification and failure isolation. This idea is critical in distributed systems, given that preventing every fault may be impractical, but assuring system continuity remains essential. (2, p. 6–10.)

Scalability in RLIMS signifies the system's capability to handle rising usage without lowering functionality. For instance, the system can adapt by modifying the inventory tracking service independently or by adding supplementary resources if the quantity of items being checked in and out notably increases. This flexibility preserves RLIMS's efficacy even during periods of heightened activity, which aligns with foundational principles of essential. (2, p. 6–10.)

Maintainability assures that system upgrades, debugging, and enhancements may be implemented easily when lab requirements change. In addition to introducing new tools or equipment, the RLIMS needs to adjust to changes like updating data structures, changing user access roles, improving inventory operations, or integrating with new tools. Designing for maintainability promotes continuous efficiency and adaptability and keeps the system from becoming a long-term burden. In his discussion on resilient systems, Kleppmann (2, p. 18–19) emphasizes this point by pointing out that systems with a clear structure are simpler to maintain and grow over time.

Through a decentralized design, the RLIMS system ensures that even if one part, like the database or user login module, fails, the rest of the infrastructure stays up and running. By dividing the system into independently managed pieces, RLIMS prevents a complete failure that could occur in a single, unified setup. This separation acts as a protective measure, allowing important tasks to continue without interruption—a crucial feature in a laboratory environment where constant inventory access is often necessary (14).

3.1.2 Integration with RLIMS

The MERN stack has been implemented in the distributed architecture of RLIMS, allowing real-time user interaction and data processing. The three MERN stack layers; Frontend, Backend, and Database, as well as their interactions are represented in Figure 3.

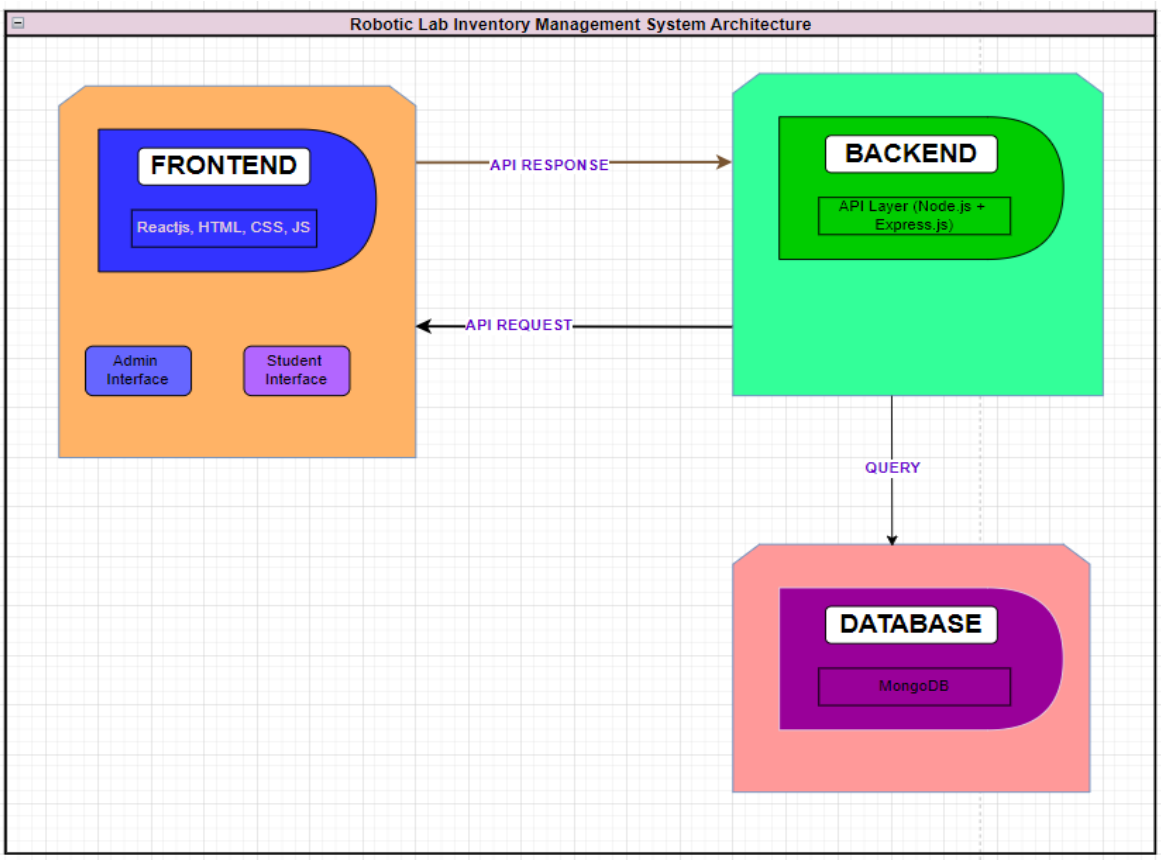

Figure 3: Robotic Lab Inventory Management System Architecture

The breakdown of the three layers is as follows:

Frontend: Developed using React.js, the frontend provides a seamless and user-friendly interface for the users, including administrators and students. React's component-based architecture allows for reusable UI components, enhancing maintainability and scalability (2). This layer is responsible for displaying inventory data, facilitating user actions such as borrowing and returning assets, and providing a seamless experience across devices.

Backend: The backend layer, utilizing Node.js and Express.js, manages the core business logic of the RLIMS. Node.js offers a powerful runtime for executing JavaScript server-side, allowing the system to handle multiple connections simultaneously, which is essential for real-time applications. Express.js simplifies the creation of RESTful APIs, enabling seamless communication between the frontend and backend. Key functionalities of this

layer include processing user requests, managing authentication, and updating asset statuses (4).

Database: The database layer employs MongoDB, a NoSQL database well-suited for handling diverse and evolving data types (6). This layer efficiently stores lab assets, tracks asset usage and borrowing history, and manages user profiles. The flexibility of MongoDB supports future enhancements and scalability as the lab grows, making it a fitting choice for applications with changing data requirements (7).

3.2 Database schema for lab items

The RLIMS database schema is designed using MongoDB, a robust and flexible document-oriented database that effectively solves the dynamic data management requirements of a robotic lab setting. The document-based structure and scalability of MongoDB make it an excellent choice for this project. This approach enables RLIMS to handle varied lab assets, track user interactions, and effortlessly adjust to changing needs, which leads to efficient and effective inventory management in a constantly changing environment (15, chapter 2).

Users Collection:

Information about each user in the platform, including administrators and students.

- **Primary Key:** UserID
- **Key Fields:** Username, Password (hashed), Role (e.g., Admin, Student), Email, CreatedAt, UpdatedAt

Each user can be assigned with:

- One-to-many with **BorrowingTransactions** (a user may borrow multiple times).
- One-to-many with **BorrowingTransactions** (an admin may register multiple assets using the AddItem field).

Assets Collection:

This collection contains information about the all the physical items within the lab, such as robots, tools, components among others.

- **Primary Key:** AssetID
    - **Key Fields:** AssetName, Category, Status, Location, CreatedAt, UpdatedAt

Each asset:

- Can be lent out several times (one-to-many relationship with BorrowingTransactions).
- Can be connected to a user (who borrowed) through the Borrow field.

Borrowing Transactions Collection:

This collection tracks all items borrowing activities, including borrow and return dates.

- **Primary Key:** TransactionID
    - **Key Fields:** UserID, AssetID, BorrowedAt, DueDate, ReturnedAt,Status

Each borrowing transaction:

- Links Users and Assets through foreign keys.
- Records the full borrowing lifecycle, including current status.

The Entity-Relationship Diagram (ERD) for the lab provides a visual overview of how Users, Assets, and BorrowingTransactions are interconnected, emphasizing two important one-to-many links. This setup facilitates effective monitoring, flexible handling of data associations, and future potential growth. The illustration (see Figure 4) supports the RLIMS framework and demonstrates how the layout supports actual laboratory operations.

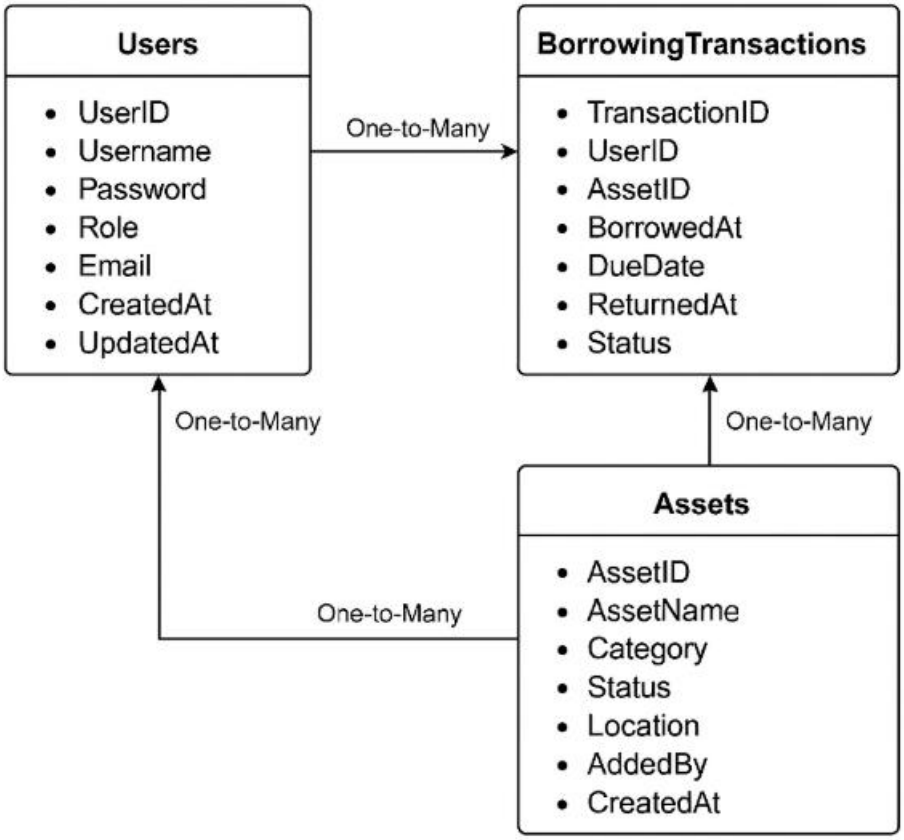

Figure 4: Entity-Relationship Diagram (ERD) for RLIMS.

Entity Relationships Overview

- **Users → BorrowingTransactions:** One-to-many: A user can borrow multiple items.
- **Assets → BorrowingTransactions:** One-to-many: An item/s can be lent out or borrowed multiple times depending on availability).
- **Users → Assets:** One-to-many: an admin can add multiple lab assets.

3.3 User Interface and workflow integration

The User Interface (UI) of the Inventory Management System (IMS) is designed with a user-centred approach, ensuring the system is intuitive and efficient for both administrators and students. Focusing on user-centred design principles outlined by Lowdermilk, the system delivers a seamless and responsive experience that adapts to both desktop and mobile platforms. (5, p. 5–11).

Figure 5 illustrates the overall structure of the IMS interface, highlighting the layout and navigation elements designed for both administrators and students.

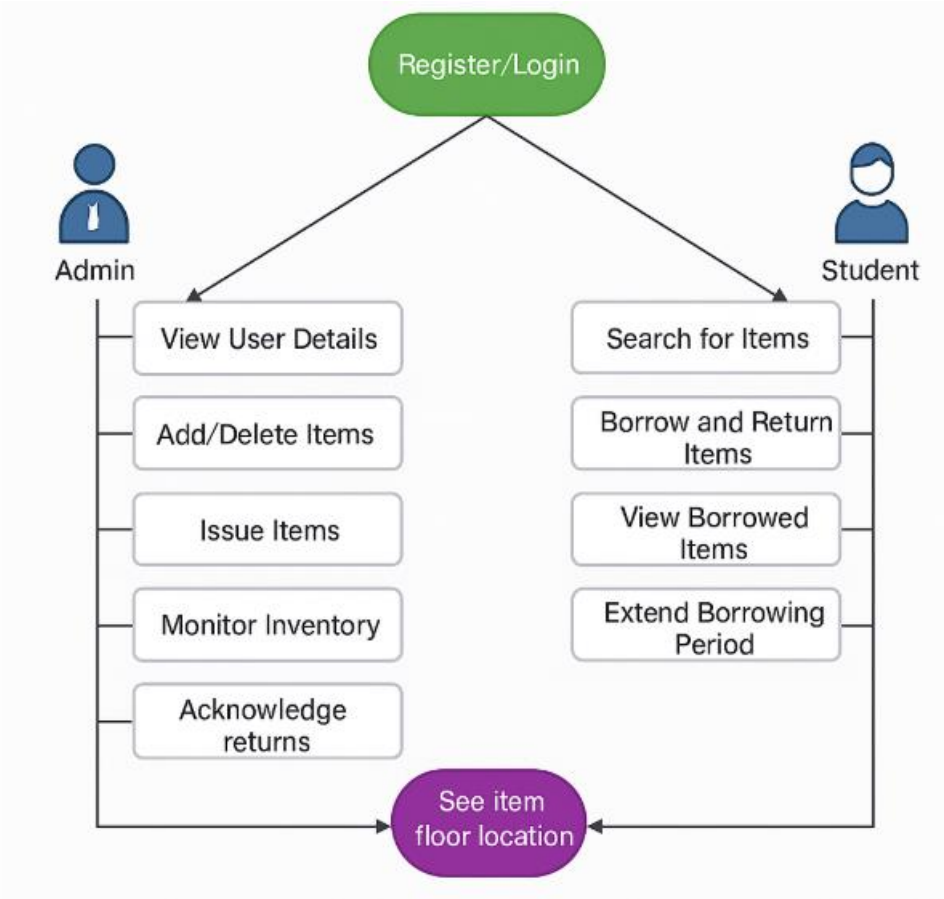

Figure 5: Overall UI Structure Diagram of the IMS

The following user-centered design principles have been applied to the IMS interface to ensure a highly functional and user-friendly experience:

Focus on the User: User-centered design begins with a thorough understanding of the user's requirements, objectives, and tasks. In the IMS environment, the design process guarantees that interfaces are suited to the specific requirements of both administrators and students. This principle is based on theory that design should prioritize making activities simple and user-friendly, allowing for smooth engagement and completion of tasks. (5.)

Usability and simplicity: According to Nielsen (17, 23-43), usability is key to create systems that are easy to learn yet efficient to use. The IMS simplifies

complicated complex activities such as inventory tracking and borrowing, therefore enhancing speed and user confidence through a clear, minimal interface.

Iterative Design: Iterative design is an important part of UCD, involving repeated user testing and adjustments. Norman emphasises that continual feedback loops help the system adapt to better fulfil the user needs. In the IMS, ongoing testing ensures the design remains consistent with expectations. (16.)

Accessibility: Accessibility ensures that the system is usable by everyone, including those with disabilities. Norman continues that (16, p. 1), web applications must be designed to accommodate people with various impairments. The IMS follows this principle by making the system usable by all, including users with disabilities.

Integrating these principles ensures that the IMS provides an efficient and satisfying user experience. It enables users to manage their task with ease, whether interacting with inventory or tracking asset, resulting in improved satisfaction and operational efficiency.

3.3.1 Admin interface

The admin is responsible for managing the overall inventory and user accounts. Their interface is equipped with features that allow them to:

- **Create/register an account/login:** Admins are required to authenticate their credentials before accessing the system, ensuring that only authorized personnel can manage inventory and users (see Figure 3.1). This process ensures that only authorized personnel can manage the inventory and users.

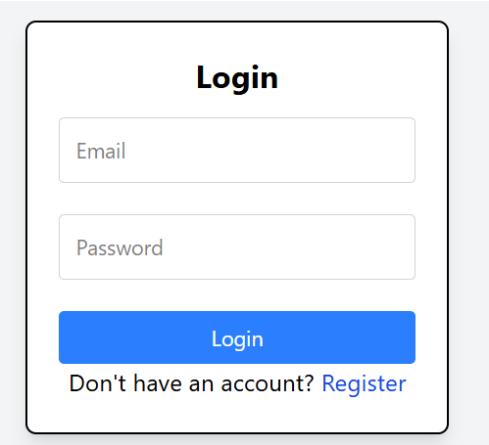

Figure 3.1: Admin Create/Login Page.

- **Issue Items to Students:** Admin can add new items to the system or delete items that are no longer available (see Figure 3.2). This feature includes the ability to input item details such as the name, category, position/location, and quantity.

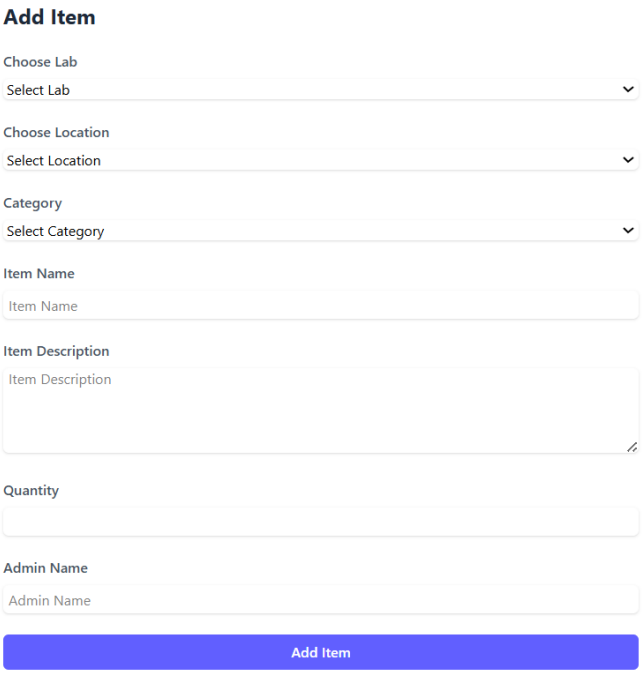

Figure 3.2: Admin functionality to Add items.

- **Issue Items to Students:** Admin can issue items to students and monitor the status of each item, whether it is borrowed or returned (see Figure 3.3). This functionality helps maintain real-time inventory data and monitor usage.

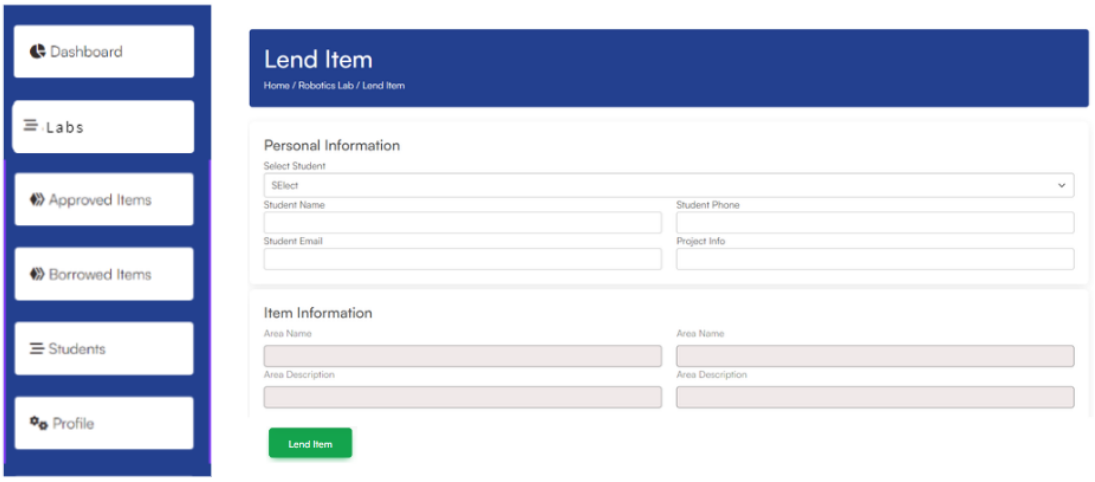

Figure 3.3: Admin Dashboard showing "Issue Items".

- **Acknowledge Items:** Admins can acknowledge when items are borrowed or returned. This ensures the inventory is always current and reflects the actual availability of items as shown in Figure 3.4.

Figure 3.4: "Acknowledge Borrowing/Returning" page

- **Acknowledge Items:** Includes a graphical representation that displays (see Figure 3.5)**.**
    - The total number of items categorized by type, i.e. equipment.
    - The number of items currently lent out to each user on the list.

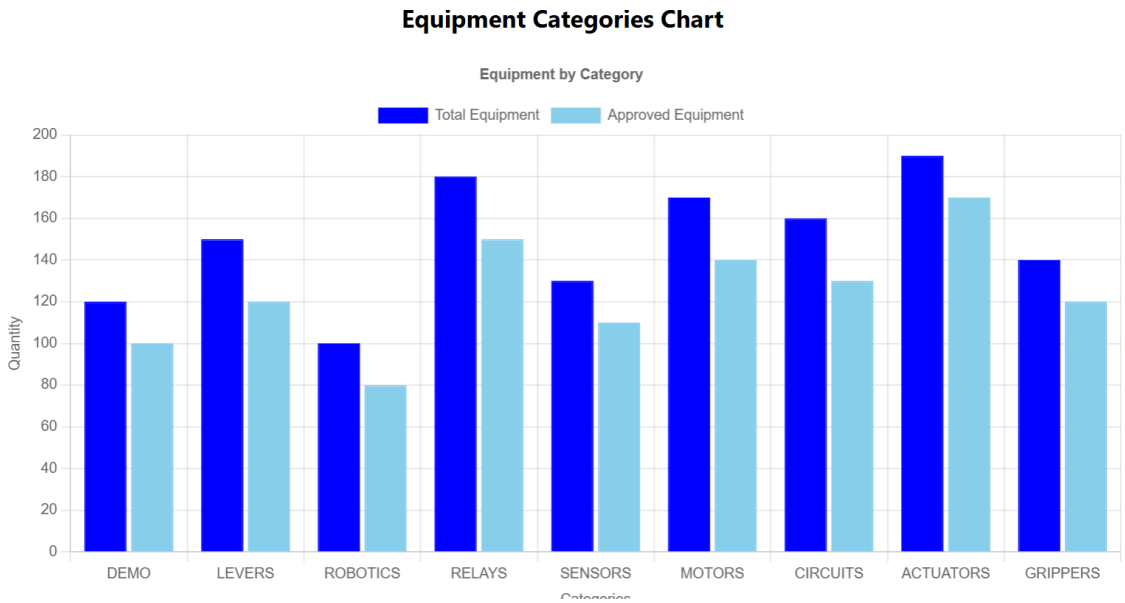

Figure 3.5: "Item Monitoring" with graphical representations.

- **Locate Item/s:** Admins can view the floor map of the lab's storage areas, which help in navigating retrieval of specific items quickly (see Figure 3.6).

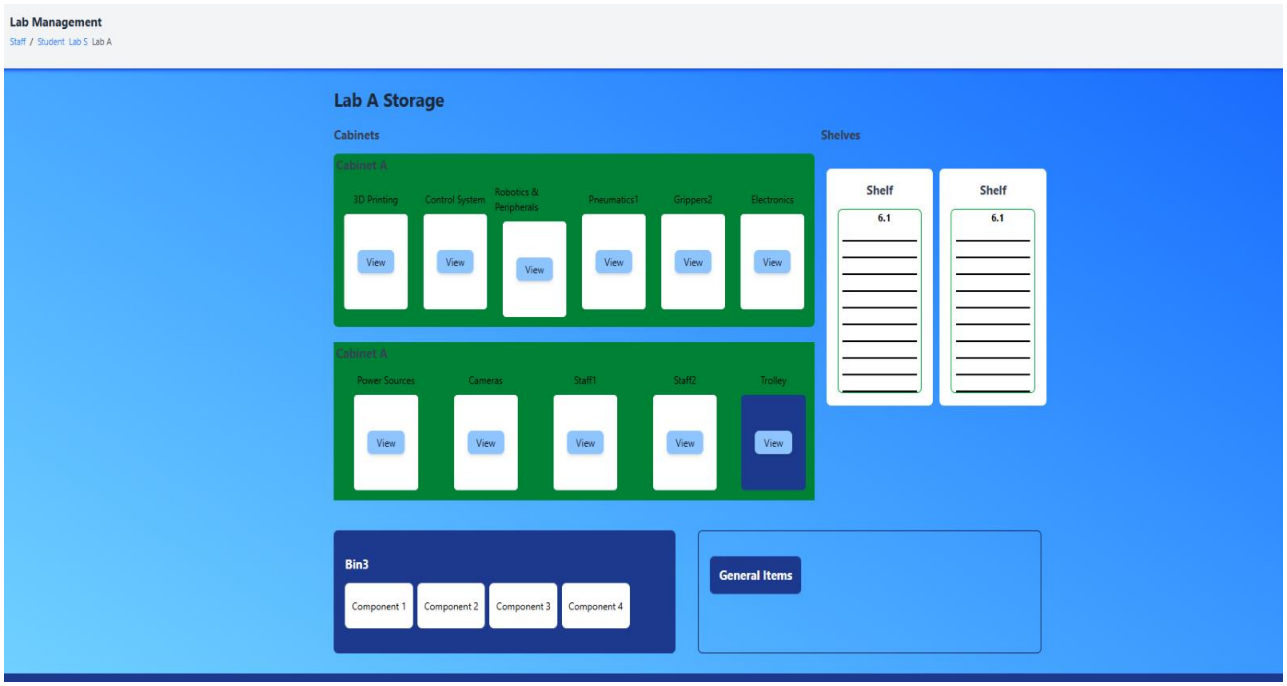

Figure 3.6: Item Floor Location

3.3.2 Student interface

The student interface is designed to be user-friendly, allowing students to easily interact with the inventory system to manage item borrowing and returns. Students can access features such as:

- **Search for Items:** Students can use keywords to look for items, making it possible to find the tools or equipment they need quickly (see Figure 3.7).

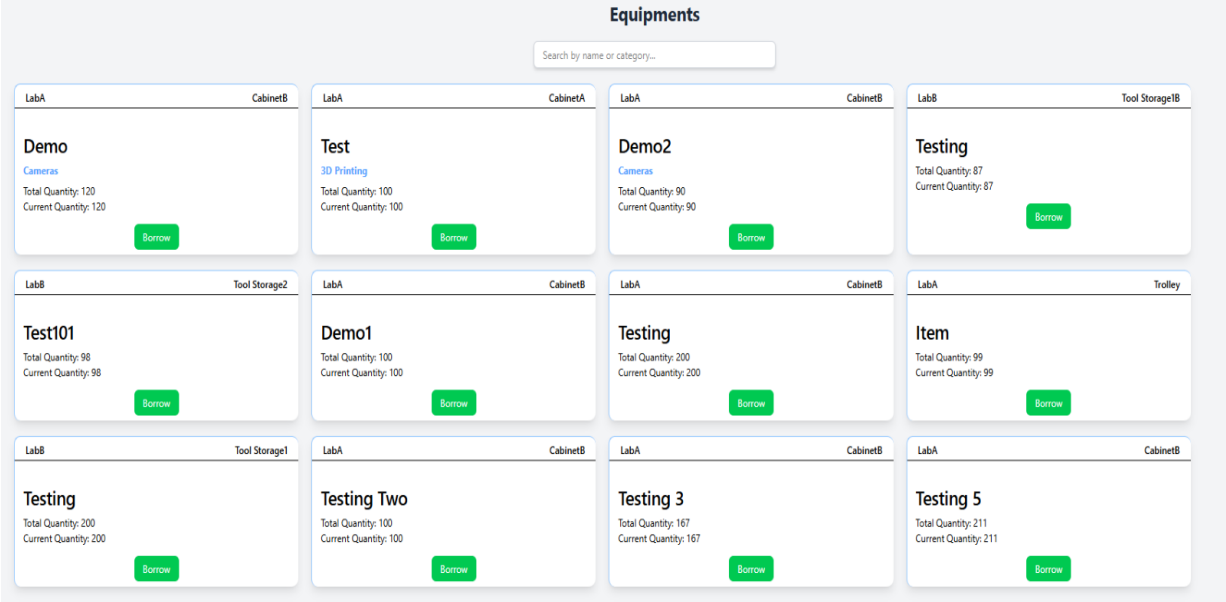

Figure 3.7: Search for Items

- **View borrowed Items:** Students can access a list of all the items they have borrowed, along with the due dates for each as shown in Figure 3.8.

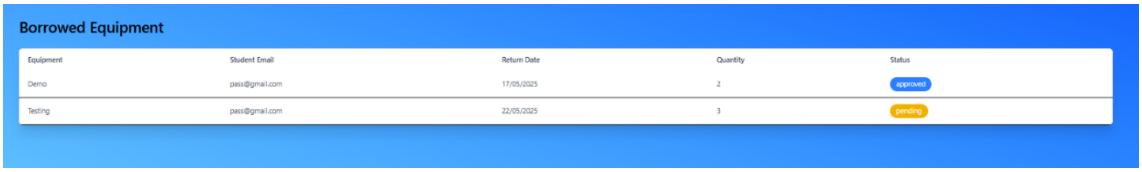

Figure 3.8: Borrowed Items

- **Item Floor Location:** Once an item is confirmed for borrowing, students can view a visual map that shows the physical location of the item in the lab, reducing search time and confusion (see the previous Figure 3.6).

According to fundamental user experience design concepts like clarity, feedback, and consistency, these features put an emphasis on accessibility and simplicity, making it simpler for students to find, borrow, and manage the materials they need for their projects (5, 23-25, 6).

Figure 3.9 represents a typical workflow that students go through within the framework of the system. It starts with students searching for an item and after that, they can submit a borrowing request through the portal. Once submitted, the system notifies laboratory personnel or administrator for approval. When the borrowing operation is granted, a timestamp is added to the record, and the item's availability status is updated immediately.

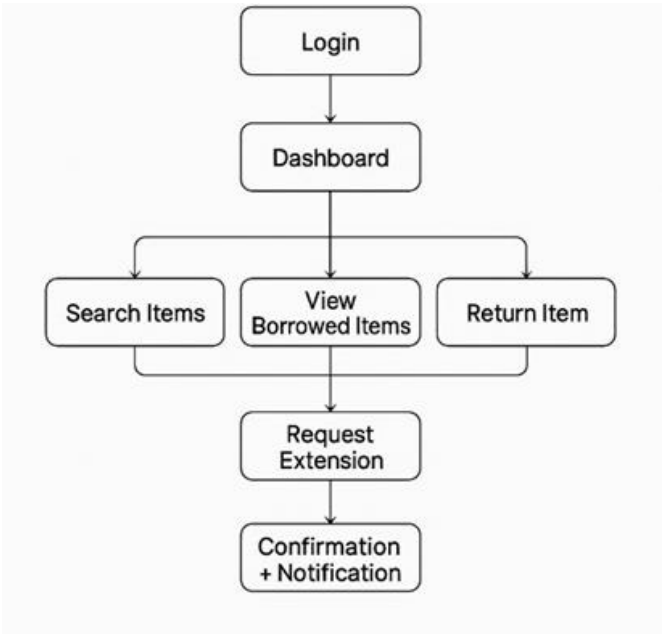

Figure 3.9: Student Workflow Interface

This approach indicates that students have a simple and guided path to accessing the necessary resources while avoiding administrative delays and the requirement for manual tracking.

3.3.3 Workflow integration

The seamless workflow in the frontend and backend depends on RESTful Application Programming Interface (API) interactions, which allow for smooth communication between the user interface and the database. This configuration ensures real-time updates on inventory status, asset locations, and borrowing actions. The uses of React.js for the frontend and Node.js with Express for the backend, user activities are handled quickly and efficiently, which creates fully responsive software. (12.)

The IMS maintains the right balance between functionality and usability by considering user needs and simplifying workflows for both administrators and students. This approach, jointly with the adaptation of the MongoDB database and the MERN stack, ensures that the system can grow and expand in the future. (12). Figure 3.10 represents the overall system workflow, comprising student involvement, administrative the processes, and system updates.

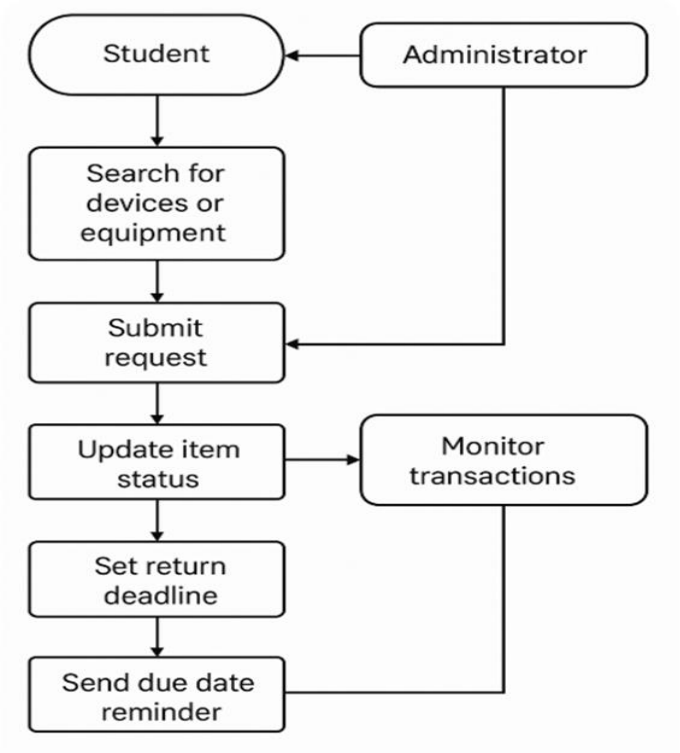

Figure 3.10: Workflow diagram of the RLIMS borrowing process.

4 IMPLEMENTATION

The system was developed using the MERN stack, which includes MongoDB, Express.js, React.js, and Node.js. This stack was chosen for its scalability, flexibility, and high performance, making it an ideal choice for web applications that require fast data processing and dynamic user interfaces. MongoDB serves as the database solution to ensure efficient handling of large datasets, while React.js enables the development of responsive and modular user interfaces. Express.js and Node.js together provide a robust backend capable of managing high-volume requests and seamless RESTful API interactions. (12.)

Subramanian (12) claims that the MERN stack provides a consistent JavaScript environment on both the client and server, making it especially well-suited for full-stack development. In addition to improving performance, this architecture makes maintenance easier and allows for scalability as user demand increases. A simplified architecture flow of the stack is shown in Figure 6.

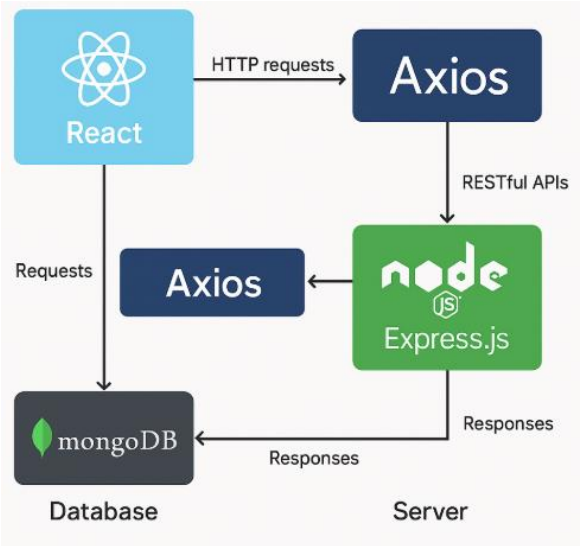

Figure 6: Technology Stack Overview Diagram

These requests (responses and requests) are handled by the backend server, which was built using Node.js and Express.js. The server communicated with MongoDB to retrieve or store data. The client-server cycle was completed when responses returned over the same pipeline.

4.1 Frontend Technologies

The frontend implementation of the RLIMS makes use of React.js, a robust and adaptable JavaScript library for creating user interfaces (UI). React uses a component-based architecture to make it possible to create dynamic and interactive user interfaces. The user interface is divided into separate, reusable parts in this architecture. Modular development and improved maintainability are made possible by each component's ability to control its state and re-render itself as needed. (18.)

React's component-based architecture enables reusable, self-contained UI elements such as buttons, forms, and navigation bars, each managing its own state and rendering logic, which improves both scalability and maintainability (18). Figure 6.1 illustrates the component hierarchy of the frontend in the RLIMS.

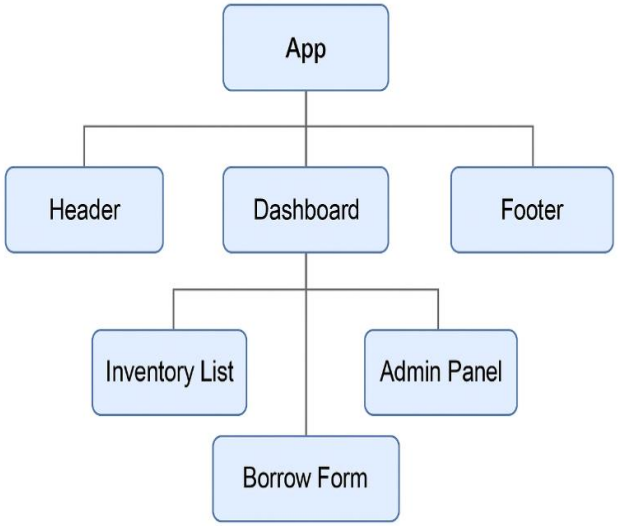

Figure 6.1: React UI Component Structure of the RLIMS

At the root level is the App component, which integrates React Router to manage page navigation between routes such as Login, Dashboard, and Admin Panel. Each of these routes is composed of nested components such as Navbar, Item List, Borrow Form, and Inventory Modal, enabling modular development and clear separation of concerns. The Auth Context provides

global access to authentication states, while Tailwind CSS ensures consistent styling and responsive design throughout the interface.

In addition to UI structure, RLIMS utilizes React's Virtual Document Object Model (DOM) to improve the user interface's responsiveness and performance. React can effectively handle user interface modifications because to the DOM, which reduces the need for direct DOM manipulation. React first updates the DOM, a thin version of the actual DOM, whenever a change is made. The "diffing" mechanism is then used by React to determine the changes between the updated DOM and the prior version. The real DOM is then only updated as needed, minimizing the number of direct manipulations and enhancing rendering efficiency. (18.)

In RLIMS, where the user interface controls the dynamic real-time updates like inventory management, user actions (such borrowing and returning goods), and other interactive components, this procedure is especially helpful. React implies that the RLIMS application stays responsive even when the data changes quick ly by eliminating frequent and costly actual DOM updates, offering a seamless user experience despite the complexity of the system. (18.)

React Router, a popular library for handling navigation in React applications, was used to manage navigation within the application. It enables single-page applications (SPAs), which allow content to be dynamically updated without the need for full-page reloads, resulting in a faster and more responsive user experience. React Router manages navigation between various sections of the system, such as the inventory dashboard, item management, user login, and administrative settings. When users interact with the application (e.g., selecting a category or navigating to an inventory record), React Router updates the view by loading new content and components without refreshing the page. This approach enhances usability and ensures an effortless experience when interacting with large datasets and real-time updates. Additionally, React Router supports conditional routing based on user authentication, allowing different views to be displayed depending on user roles. (18.)

Below is an example of how routes are defined in the App component.

```
// Example of setting up React Router in RLIMS
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Dashboard from './Dashboard';
import Login from './Login';
import AdminPanel from './AdminPanel';

const App = () => {
  return (
    <Router>
      <Switch>
        <Route path="/login" component={Login} />
        <Route path="/dashboard" component={Dashboard} />
        <Route path="/admin" component={AdminPanel} />
        <Route exact path="/" component={Login} />
      </Switch>
    </Router>
  );
};

export default App;
```

React's state management and event handling features were utilized to manage user input and form validation. React's state is used by forms in the RLIMS frontend, like inventory submission forms and user login forms, to record user inputs and verify data prior to submission. The user interface is constantly in alignment with the state through React's continuous data flow, which guarantees a smooth and error-free experience. Additionally, this enables the user to receive real-time feedback in the form of error warnings or input validation signs. (18.)

Axios facilitates the efficient handling of data transfers between the frontend and backend. This HTTP client makes it easier to send queries, manage responses, and communicate with the backend APIs in real time. Axios is a dependable tool for modern online applications given that it offers integrated responses for common issues including request cancelation, error management, and data processing (19). An illustration of using Axios to retrieve data from the inventory API may be found below:

```
axios.get('https://api.rlims.com/inventory')
  .then(response => {
    console.log(response.data); // Logs the data received from the backend
  })
  .catch(error => {
    console.log('Error fetching data:', error); // Handles any error
  });
```

Axios is used in the preceding code to retrieve inventory data by sending a GET call to the backend endpoint /api/items. The inventoryItems variable contains the response, which is handled asynchronously using async/await. Errors that arise during the request are detected and recorded in the console.

Tailwind CSS, a utility-first CSS framework, was used to style the RLIMS frontend due to its flexibility, speed, and consistent design result. Utility classes were applied directly within the HTML markup, allowing for quick creation of responsive, consistent, and visually appealing user interfaces without the need for custom CSS. (20.)
Tailwind CSS was particularly valuable to the RLIMS project in the following ways:

- Rapid Development: Its utility-first nature resulted in fast execution of UI features without writing additional CSS.
- Consistency: The standard utility classes ensured a unified design across all pages and components.
- Customizability: Tailwind's setup file enabled adjustments of colours, spacing, and responsive breakpoints to match RLIMS's specific design needs.

The development workflow and user experience were improved by using Tailwind CSS, which kept the frontend clean, responsive, and extremely maintainable.

4.2 Backend Technologies

The RLIMS's backend is developed using the MERN stack, which consists of Node.js, Express.js, React.js, and MongoDB. This stack makes it possible to create a system that is incredibly adaptable, scalable, and effective. To ensure that essential features like inventory management, user roles, and real-time changes run smoothly, the backend responsible for managing data, business logic, user authentication, and frontend interaction. (12.)

Node.js runs the backend of RLIMS by handling user queries, communicating with the database, and providing API responses. It connects with Express.js and MongoDB to process user requests to borrow items or interact with the inventory system. Along with its event-driven framework, the application is assured to remain responsive without experiencing performance decline, even in the face of heavy traffic or several ongoing processes. (12.)

The RLIMS system's RESTful API development is made easier through Express.js. Express is a Node.js web server framework that makes managing HTTP requests and responses easier. Express.js defines routes for a variety of system operations, including user account management, inventory item retrieval, and login requests. Express Router is used to construct API endpoints that correspond to HTTP methods (GET, POST, PUT, and DELETE), while middleware is used for tasks such as error handling and authentication checks. For instance, Express communicates with MongoDB and returns the data in JSON format when a user checks the inventory via the route /api/items. (12). Shown below is how the API endpoint is set up in Express:

```
// Example Express route to fetch inventory items
app.get('/api/items', (req, res) => {
  Item.find({}, (err, items) => {
    if (err) return res.status(500).json({ error: err });
    res.json(items);
  });
});
```

MongoDB's scalability and adaptability make MongoDB, a NoSQL database, for its backend data storage ideal for this project, especially since it does not require a predefined schema like relational databases. This allows the data model to evolve easily as the system grows. The key structure including Users, Items, borrowing records, among others are stored in MongoDB collections. MongoDB’s flexibility ensures that the data structure can be adjusted without requiring complex database migrations, which is particularly beneficial for systems that undergo frequent updates or feature additions. (12.)

The User, Item, and Borrow Record entities that are essential to RLIMS are designed and managed using Mongoose, (12), an ODM component for MongoDB in Node.js. Mongoose's schema-based approach maintains consistent data limitations within MongoDB's adaptable NoSQL architecture.

Mongoose enforces schema-based validation, which ensures that documents inserted into MongoDB follow pre-set standards such field types, needed attributes, and unique values. Data integrity problems like missing fields or inaccurate data types are prevented due to this verification. This strategy improves data accuracy and facilitates future revisions because RLIMS can change data structures without risking inconsistent or incorrect manual validations.

JWT (JSON Web Tokens) are used by RLIMS for authentication to safeguard system routes and stop unwanted access. After a successful login, a JWT is created and safely saved in an HTTP-only cookie. This token is then included in all subsequent requests to protected API end points. The JWT middleware checks these tokens to guarantee that only authenticated users can conduct restricted activities, such as changing inventory data or borrowing equipment. (12.)

This stateless method eliminates the requirement for server-side session storage, thereby improving both performance and scalability. Furthermore, the token only contains necessary user data (such as email address and username), which improves security and privacy. (12.)

Express.js middleware validates these tokens for protected API endpoints, ensuring that only authenticated users can do sensitive actions like borrowing equipment or updating inventory data. Express middleware authenticates the token before allowing access to routes such as;

- */api/items* - get or change inventory items
- */api/borrow* - send requests to borrow or return

Role-based access control is another feature of the system that restricts authorized users from performing tasks that call for administrative privileges, such managing users or changing inventory records.

In addition to authentication methods, Mongoose reinforces the security of data by setting up schema-level restrictions to primary models such as Borrow Record, Item, and User (as earlier explained), missing fields, invalid types, and inconsistent statuses are avoided by using these schema rules to make sure that data entering the system follows preset structures.

When combined, JWT-based authentication (12), middleware enforcement, role verification, and schema validation create an overall security and data integrity framework that protects the RLIMS backend while safeguarding scalability and performance.

Below is a simplified example of how Express.js uses JWT middleware to authenticate protected routes:

```
const mongoose = require('mongoose');

const itemSchema = new mongoose.Schema({
  name: { type: String, required: true },
  quantity: { type: Number, required: true },
  status: { type: String, enum: ['available', 'borrowed'], default: 'available' }
});

const Item = mongoose.model('Item', itemSchema);
module.exports = Item;
```

4.3 Testing and Debugging

Software development requires testing and troubleshooting to ensure that systems function as expected, potential bugs are identified early, and the software remains stable across different cases (2). These tasks were crucial for verifying key features in the Robo-Lab software, including item handling, user sign-in procedures (login, register, logout), monitoring borrowed items, keeping track of borrower information, and administering the admin interface, among others.

RLIMS adopted a testing approach that combined automated unit tests, manual scenario testing, API evaluations, and real-time code debugging. While Postman and browser Development Tools (DevTools) were used to evaluate frontend behaviour and API responds, Jest and Supertest were used for backend development testing. The integrated Visual Studio Code (VS Code) Debugger was used for real-time debugging (20, 22, 23.), see Figure 6.2.

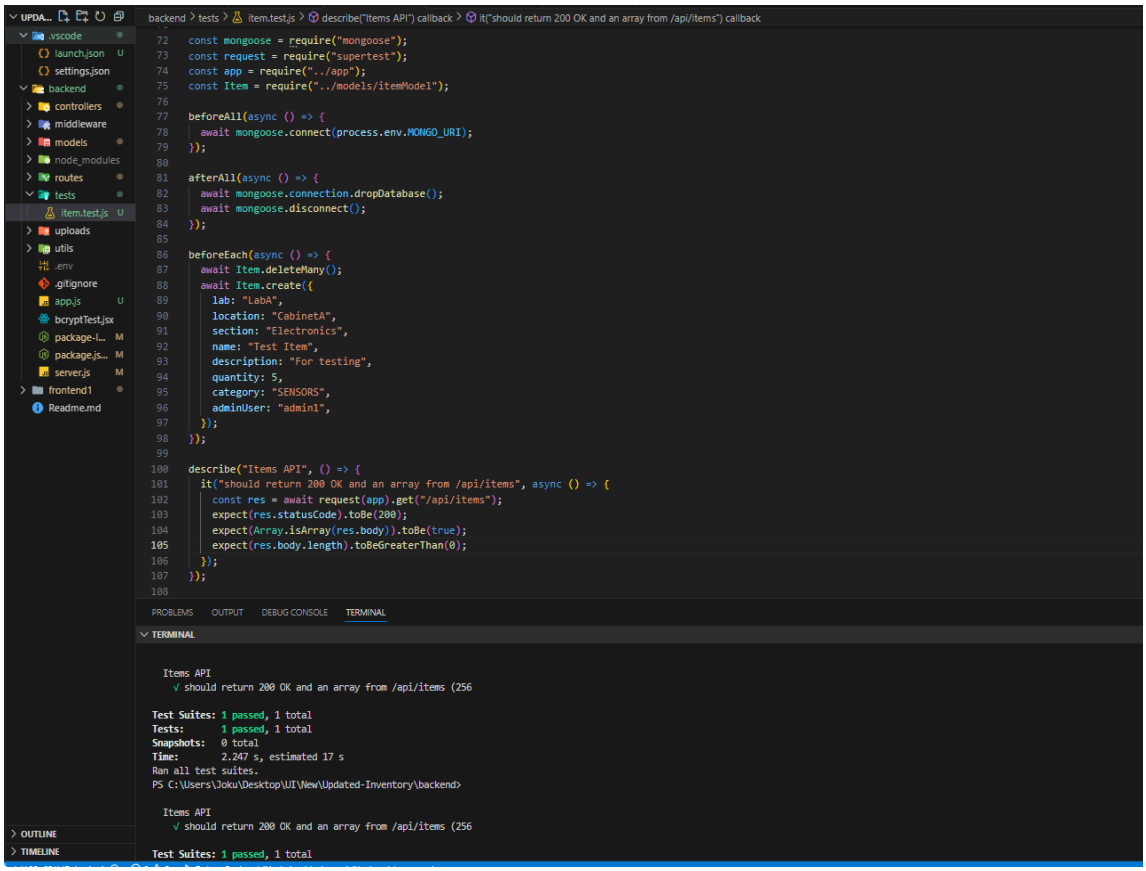

Figure 6.2: Code snippet of the Jest unit test for the GET /api/items endpoint.

A Jest unit test aimed at the GET /api/items endpoint is shown in the figure confirming that inventory data is returned accurately. This was an essential step in the validation process because inventory management is a fundamental component of RLIMS.

The following system layers were the focus of the testing methodology:

• Unit Testing: Used Jest to ensure that individual backend logic and functions performed as expected.

• API Testing: Used Jest + Supertest and Postman to confirm endpoint functionality, response formats, and error handling.

• Frontend Validation: Verified that data from the backend was dynamically rendered by the user interface as intended.

Testing and Debugging Tools used in RLIMS included:

Automated Testing: The backend functionality was validated using Jest and Supertest. Supertest used simulated HTTP queries to test API endpoints such as GET /api/items to ensure that the response and status codes were valid, while Jest used unit tests to verify that the backend logic was correct. (21.)

Manual API testing: Real-time API interactions were simulated using Postman. For example, to confirm that the server returned an array of inventory items and returned a 200 OK response, a GET request was issued to http://localhost:5000/api/items. (22.) This procedure shown in Figure 6.3.

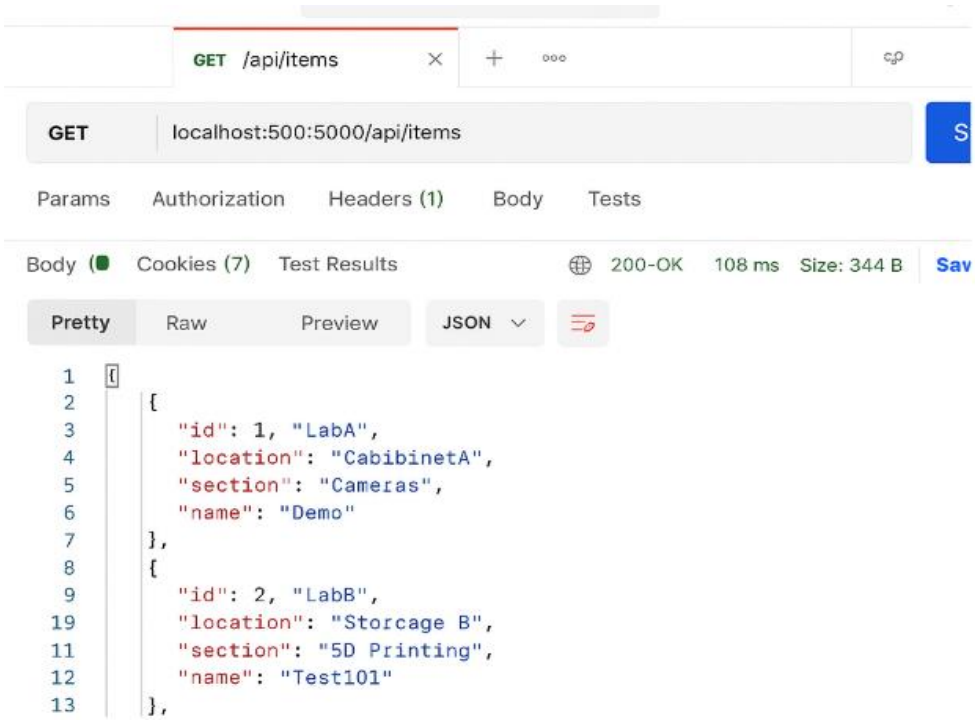

Figure 6.3: Manual API test using Postman to validate GET /api/items endpoint

The /api/items endpoint was manually tested using Postman to further guarantee functionality, as shown above:

1. Launch Postman and select GET as the request method.
2. Enter http://localhost:5000/api/items as the URL.
3. Click "Send."
4. Validate that 200 OK is the response status.
5. Confirm that an array of point details makes up the response content.

Frontend analysis: React Developer Tools enabled the analysis of React component states and structures, ensuring that data was appropriately transmitted from backend APIs and rendered by UI components (24.)

Live Debugging: During the development and testing phases, the VS Code Debugger helped trace backend control flow, check variables, and diagnose test failures (23).

The addition of frontend debugging to backend validation was essential as well. The RLIMS equipment management interface is displayed in operation in Figure 4.5. The interface's left portion shows item cards with the item's location and amount (e.g., Demo, Test101, Cameras). This demonstrates how React components correctly render data provided from the backend.

Concurrently, the browser's DevTools (shown on the right side of Figure 6.4) verify that the backend API was used to retrieve an array of items.

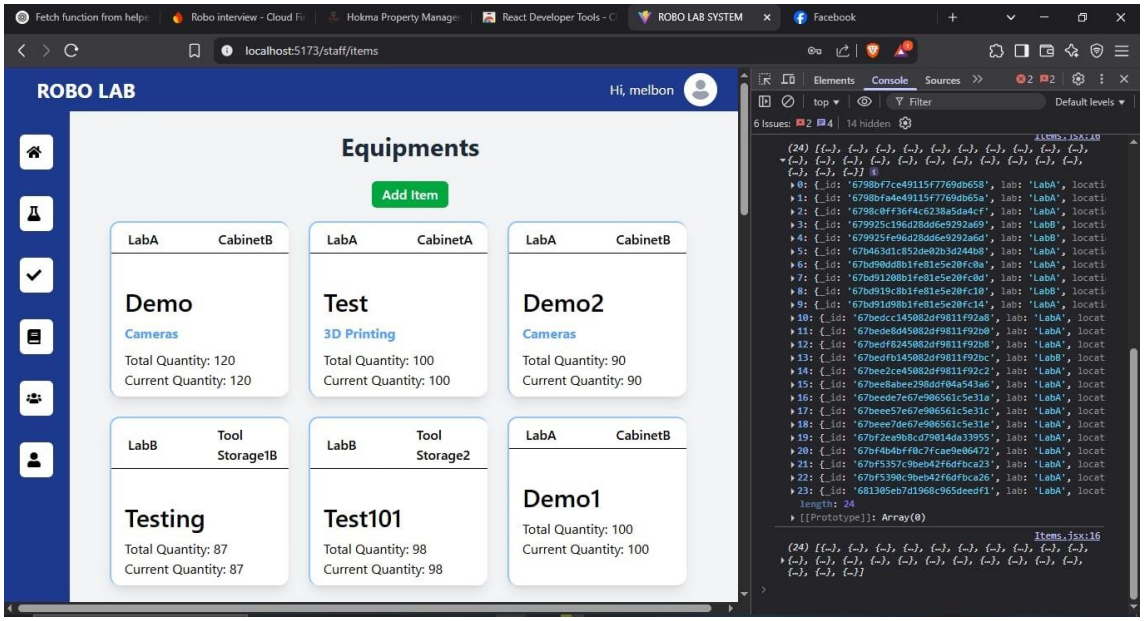

Figure 6.4: Frontend Validation of Item Management with DevTools

The output from the console confirms that:

• The data was successfully retrieved and structured correctly.
• The data was accurately mapped and displayed by the frontend.
• Data integrity and user interface were successfully verified by real-time inspection.

4.4 Deployment and hosting

The frontend, backend, and database services are all separated in the modular architecture used to deploy RLIMS. High scalability, maintainability, and performance in many contexts are guaranteed by this configuration. The modular deployment pipeline is displayed in Figure 6.5, emphasizing the connection between database provisioning, backend deployment, and frontend hosting.

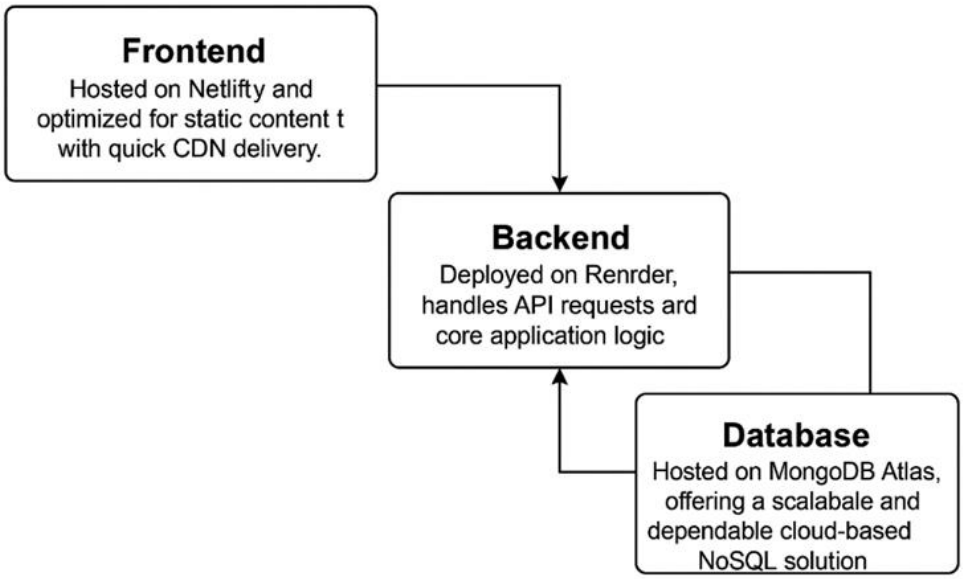

Figure 6.5: Overview of the Modular Deployment Workflow

Frontend Deployment:

Netlify, a platform designed for single-page applications, hosts the user interface built with React.js. When code is pushed to the linked GitHub repository, npm run build initiates automatic builds, which are then deployed throughout the worldwide Content Delivery Network (CDN) of Netlify. This facilitates continuous integration and delivery (CI/CD) tasks like preview deployment, environment configuration, version control, and quick load times. (25.)

Backend Hosting:

The backend server, developed with Node.js and Express.js, is deployed on Render and handles server functions including API routing, authentication (via JWT), and data logic. Render enables safe environment variables for storing sensitive data, such as database credentials and API keys, and rebuilds the backend service whenever updates are committed (23, 26.)

Database Layer:

The system's data is hosted in MongoDB Atlas, a cloud-based NoSQL server that supports collections such as Users, Items, and Borrow Records. MongoDB Atlas provides availability, scalability, and automatic backups. Mongoose, which offers object modelling and schema enforcement for MongoDB, is used by the backend to connect to the database (11).

Security and Configuration Management:

Deployment of specific environment variables and .env files are used to manage sensitive credentials (such as database URIs and JWT secrets). This permits various settings for the stages of development, testing, and production and keeps secrets out of version control. (25, 26.)

Development and Deployment Toolchain:

- The main IDE for front-end and back-end development was Visual Studio Code (23).
- Throughout the deployment process, Postman assisted with manual API testing (22).
- Source code modifications were directly linked to automated deployments through the usage of GitHub for version control and CI/CD integration (27).

This modular and automated deployment approach allows RLIMS to effortlessly evolve with future enhancements while maintaining a safe and dependable infrastructure.

5 CONCLUSION

In academic lab settings, the application effectively modernizes traditional inventory management into a digital, efficient platform. The solution, which was created with the MERN stack (MongoDB, Express, React, and Node.js), offers a cross-platform capabilities, real-time updates, and full-stack performance.

RLIMS is designed to meet laboratory environments' needs for operation. It has role-based access, safe user authentication, and a simple interface built with Tailwind CSS and React.js. These decisions support responsive design, which makes it possible to use the site on desktops, laptops, tablets, and smartphones.

The performance and reliable functionality are assured by the backend APIs, which were created using Express.js and tested using Jest for unit testing. Postman API testing made it possible to validate endpoint behaviour and simulate a variety of user situations. Throughout the project, additional tools like VS Code Debugger and DevTools improved the debugging and development process.

RLIMS promotes inventory management's reliability, traceability, and effectiveness by replacing the spreadsheet-based systems. Students benefit from a user-friendly interface for borrowing and returning lab items, while administrators gain real-time visibility and greater control over asset allocation.

Finally, RLIMS demonstrates the efficacy of modern web development frameworks in addressing practical issues in educational and research settings. It is a beneficial and sustainable solution for long-term inventory management needs because of its scalable architecture, which permits future expansion in the form of analytics, automatic notifications, or interaction with IoT-based asset tracking.

REFERENCES

1. Deshmukh, R. R., Pawar, A., Katore, V., Dabhade, N., Kasture, V. & Bagul, S. 2022. Inventory Information System. International Journal of Scientific Research & Engineering Trends, Vol. 8, Issue 1, p. 371–373. Search date: 24.02.2025. Available: https://ijsret.com/wp-content/uploads/2022/01/IJSRET_V8_issue1_150.pdf

2. Kleppmann, M. 2021. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media. E-book. Read: 25.2.2025. Available: https://learning.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/ Access required.

3. GeneMod. 2025. Improving Lab Inventory Management: Automation, Customization, and Efficiency. Search date: 08.02.2025. Available: https://genemod.net/blog/improving-lab-inventory-management-automation-customization-efficiency.

4. Muller, M. 2003. Essentials of Inventory Management. 2nd ed. AMACOM. E-book. Read: 8.5.2025. Available: https://www.iibms.org/pdf/e-library/essentials_of_inventory_management.pdf.

5. Lowdermilk, T. 2013. User-Centered Design: A Developer's Guide to Building User-Friendly Applications. O'Reilly Media. E-book. Read: 25.2.2025. Available: https://learning.oreilly.com/library/view/user-centered-design/9781449359812/ Access required.

6. Canziba, E. 2018. Hands-On UX Design for Developers. O'Reilly Media. E-book. Read: 18.2.2025. Available: https://learning.oreilly.com/library/view/hands-on-ux-design/9781788626699/ Access required.

7. Kirakowski, J., Bevan, N., & HFRG (Eds.). 1998. Handbook of User-Centred Design: WP 6, Deliverable D6.2 (Final Version). E-book. Read: 18.2.2025. Available: https://uxp.ie/INUSE_Handbook_of_UCD.pdf

8. Wolters Kluwer. 2025. Modern Solutions vs. Excel for Inventory Management. Search date: 22.02.2025. Available: https://www.wolterskluwer.com/en/expert-insights/modern-solutions-vs-excel-for-inventory-management

9. Kes Systems. 2023. Inventory Management Made Easy: How to Avoid Manual QuickBooks Imports. Search date: 15.02.2025. Available: https://kes-systems.com/wp-content/uploads/2023/07/QuickBooks-Inventory-Made-Easy.pdf

10. Dhirendra, S., & Tejas, P. 2024. System Design Guide for Software Professionals. O'Reilly Media. E-book. Read: 25.2.2025. Available: https://learning.oreilly.com/library/view/system-design-guide/9781805124993/ Access required.

11. MongoDB. 2023. MongoDB Atlas documentation. Search date: 28.02.2025. Available: https://www.mongodb.com/cloud/atlas

12. Subramanian, V. 2019. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node. Apress. O'Reilly Media. E-book. Read: 25.2.2025. Available: https://learning.oreilly.com/library/view/pro-mern-stack/9781484243916/ Access required.

13. Richards, M. and Ford, N. 2025. Fundamentals of Software Architecture. 2nd ed. Sebastopol, CA: O’Reilly Media. E-book. Read: 25.2.2025. Available: https://learning.oreilly.com/library/view/fundamentals-of-software/9781098175504/ Access required.

14. Newman, S. 2021. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media. E-book. Read: 25.2.2025. Available:

https://learning.oreilly.com/library/view/building-microservices-2nd/9781492034018/ Access required.

15. Bradshaw, S., Brazil, E., & Chodorow, K. 2019. MongoDB: The Definitive Guide. (3rd ed.). O'Reilly Media. E-book. Read: 25.2.2025. Available: https://learning.oreilly.com/library/view/mongodb-the-definitive/9781491954454/ Access required.

16. Norman, D. A. 2013. The Design of Everyday Things. Tantor Media, Inc. Audiobook. Search date: 31.3.2025. Available: https://learning.oreilly.com/videos/the-design-of/9781452624129/ Access required.

17. Nielsen, J. 1994. Usability Engineering. O'Reilly Media. E-book. Read: 25.2.2025. Available: https://learning.oreilly.com/library/view/usability-engineering/9780125184069/ Access required.

18. Accomazzo, A., Lerner, A., Murray, N., Allsopp, C., Gutman, D. & McGinnis, T. 2017. Fullstack React: The Complete Guide to ReactJS and Friends. Fullstack.io. E-book. Read: 10.3.2025. Available: https://demo.smarttrainerlms.com/uploads/0003/trainings/course/45/modules/fullstack-react-book-r30_1510302324482009603.pdf

19. Axios Documentation. 2020. Axios: Promise-based HTTP client. Search date: 18.2.2025. Available: https://axios-http.com

20. Tailwind CSS. 2025. Get started with Tailwind CSS. Search date: 28.03.2025. Available: https://tailwindcss.com/docs

21. Orie, C. 2019.Testing Node.js + Express API with Jest and SuperTest. DEV Community. Search date: 15.04.2025. Available: https://dev.to/nedsoft/testing-nodejs-express-api-with-jest-and-supertest-1km6

22. Postman. 2025. Your Complete API Platform, From Design to Delivery. Search date: 18.03.2025. Available: https://www.postman.com/

23. Visual Studio Code Docs. (n.d.). Search date: 8.02.2025. Available: https://code.visualstudio.com/docs

24. React. (n.d.). The library for web and native user interfaces. Search date: 28.03.2025. Available: https://reactjs.org/

25. Netlify. 2025. Netlify documentation. Search date: 10.03.2025. Available: https://docs.netlify.com/

26. Render. 2025. Render documentation. Search date: 5.03.2025. Available: https://render.com/docs

27. GitHub Docs. 2025. Get started with GitHub documentation. Search date: 8.03.2025. Available: https://docs.github.com