Research Labs Inventory

Jonathan Moore
Evan Ross
Lizzett Tapia
Sumedha Bhattacharyaa

Research Lab Inventory
Final Report

REVISION – Final
5 December 2024

Table of Contents

Execution Plan

Key
Web App
Data Server
Mobile App
ML Model
Completed
In Progress
Not Started
Behind Schedule

	8/20/24	8/27/24	9/3/24	09/10/24	09/17/24	09/24/24	10/01/24	10/08/24	10/15/24	10/22/24	10/29/24	11/5/24	11/12/24
Basic Page Layouts													
Backend for each Page													
Search Engine Functionality													
Check-out for items													
Web-app Validation													
SQLite Database Created													
Getting a working server													
Python script to populate database													
Relationships Between Data (Queries)													
Database Server Validation													
Create App and Functional Simulation													
Create UI for each page													
Add functionality using SQLite database													
Search Database and Check Out Functionality													
Track Which Users Have Checked Out/In Items													
Mobile App Validation													
Watch Youtube Videos on ML/Nueral Networks													
Choose ML Model and Finalize Code		REDONE											
Finalize Dummy Data Sets													
Create a Working Automation Function													
ML Model Validation													

ConOps Project Report	DUE SEP 15
FSR, ICD, Milestones, Plan	DUE SEPT 26
Project Introduction Presentations	PRESENT ON SEPT 24
Project Update Presentations	PRESENT ON OCT 14 - 23
Final Presentations	PRESENT ON NOV 11-20
Final Report	DUE DEC 5

Validation Plan

Key
Web App
Data Server
Mobile App
ML Model

Test Name	Success Criteria
Input Testing (Happy path)	Accept changes in db for valid input fields for registration, sign-in, etc.
Input Testing (Unhappy path)	Do not make changes in db for invalid input fields for registration, sign-in, etc.
Page route testing	Ensure that all page routing acts as expected (kicking unauthorized users, sign-out upon deletion of account, etc.)
Cart functionality (Happy path)	Cart for check-out should allow users to select as many ites as they want and perform a collective check-out all at once.
Cart functionality (Unhappy path)	Cart shouldn't break when stress tested. Should adapt to the limits of what is possible from user.
Permission level testing	Users of different permission levels are restricted to only their given abilities.
Table Creation Test	Python scripts can successfully create all necessary tables. The tables are created with correct schema.
Random Data Insertion Test	Python scripts successfully generate and insert random data into tables (e.g., 10 random users with valid @tamu.edu emails). No errors occur during the insertion process.
Functional Testing	Can successfully perform Create, Read, Update, and Delete operations on all tables without errors.
Bulk Insertion Performance Test	Python scripts can insert large datasets into hosted database without slowdowns.
Migration to Hosted Server Test	The migration of data and schema from the local SQLite database to the hosted server should be successful. No data should be lost or corrupted during the migration process.
Application UI functionality	App User interface acts as expected, It can open, navigate pages, close, has no bugs
Database Test (Users)	Application can successfully add new users to database
Database Test (Components)	Application can successfully add and remove components from database
User Checkout/Check In	Ensure that Users are able to check out and check in items, and that the checked out items are stored in an active checkout list in a database.
Image Recognition Accuracy	The model correctly identifies items with at least 85% accuracy.
Response Time (Latency)	The time in between inputting a test image and then returning the result shouldn't exceed 15 seconds.
False Positive Rate	The model should have a false positive rate of less than 20% when identifying items.
Robustness to Dataset Variations	Evaluated the model with varying conditions such as rotations.

Validation Plan Continued

Key
Web App
Data Server
Mobile App
ML Model

Test Name	Methodology	Status	Responsible Engineer (s)
Input Testing (Happy path)	Using Playwright to test forms. All inputs are hand picked to stress test bounds of valid inputs.	Tested	Jonathan Moore
Input Testing (Unhappy path)	Using Playwright to test forms. All inputs are hand picked to stress test using invalid inputs.	Tested	Jonathan Moore
Page route testing	Using Playwright for UI regression testing.	Tested	Jonathan Moore
Cart functionality (Happy path)	Playwright for regression testing for the display of the cart.	Tested	Jonathan Moore
Cart functionality (Unhappy path)	Playwright for regression testing for the display of the cart, ensuring it doesn't break.	Tested	Jonathan Moore
Permission level testing	Playwright testing. This requires changing the permission levels of a current user multiple times.	Tested	Jonathan Moore
Table Creation Test	Write and run Python scripts to execute SQL 'create table' statements. After running the scripts, validate the tables using a query like .tables in SQLite to ensure all tables are created.	Tested	Lizzett Tapia
Random Data Insertion Test	Run the Python script that insert random data into the tables. Query the tables using 'select * from' to verify that data was correctly inserted.	Tested	Lizzett Tapia
Functional Testing	Run test cases that insert data, retrieve it using 'Select' queries, update specific fields, and/or delete certain data.	Tested	Lizzett Tapia
Bulk Insertion Performance Test	Measure the time taken to insert data and check for any performance issues.	Tested	Lizzett Tapia
Migration to Hosted Server Test	Import the data into the hosted server and run validation queries to ensure that all data has been correctly migrated.	Tested	Lizzett Tapia
Application UI functionality	Test application within the Android Studio simulated phone	Tested	Evan Ross
Database Test (Users)	Test by adding users through the mobile app, test invalid users as well (such as invalid email) to ensure user has valid credentials.	Tested	Evan Ross
Database Test (Components)	Test by adding cononents to a the database through the mobile app, test both large and small amounts at once.	Tested	Evan Ross
User Checkout/Check In	Test by attempting to check out and check in an item as a logged in user. Make sure to track active checkout table in database to ensure proper functionality	Tested	Evan Ross
Image Recognition Accuracy	Compile test datatset with images not in training or validation sets and run the model through. Record the percentage of correct identifications.	Tested	Sumedha Bhattacharyaa
Response Time (Latency)	Record time in between uploading test images and getting a result. Test across different internets, and image sizes.	Tested	Sumedha Bhattacharyaa
False Positive Rate	Create dataset with items that don't belong in the lab. Test the system's ability to avoid classifying these items. The calculate the percentage of how many times it incorrectly categorized something.	Tested	Sumedha Bhattacharyaa
Robustness to Dataset Variations	Augmented test images to include rotations and measured training accuracy	Tested	Sumedha Bhattacharyaa

Research Labs Inventory

Jonathan Moore
Evan Ross
Lizzett Tapia
Sumedha Bhattacharyaa

Concept of Operations

REVISION – Draft
10 September 2024

Concept of Operations
for
Research Lab Inventory

Team <55>

Approved by:

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	9/10/2024	Moore,Ross, Tapia, Bhattacharyaa		Initial Submission
2.0	9/26/2024	Moore		Format Changes

Table of Contents

List of Tables

List of Figures

1. Executive Summary

An important aspect of a research lab is the ability to track inventory. This is important for many reasons such as the ability to know when an item inventory is low, tracking who is responsible for what items, and knowing what items are available. Our project at its core is to develop a website, along with a smartphone app, that allows users to track and make changes to the Inventory in research labs. Users will need to login using their credentials, and they will either be categorized as a student or a staff member based on initial registration. Students will be able to check out and check in items in the lab, and the inventory will adjust accordingly. Staff members will be able to make larger changes to the inventory, allowing for the addition and deletion of large amounts of components. Our Inventory data will be stored on a database that will be linked to both the app and the website, so that changes can be made from both. Our smart phone application will also have a machine learning component where users will be able to scan an item and have it automatically identified. After scanning an item, the app will open the page where users can either check out or check in that specific item. This will make tracking components in laboratories easy and intuitive.

2. Introduction

The current lab inventory system existing is inefficient and disorganized. Lab members often go to check things out, and are met with the issue of not knowing how many things they checked out, and how much stock the lab has left of items. The current system has a person manually check things out to members, without much of a record. As a consequence, the lab lacks cohesive organization, and its members face challenges in maintaining an efficient workflow. We aim to tackle this simple logistical issue with a highly methodized Lab Inventory Tracking system.

2.1. Background

This Lab Inventory Tracking system aims to replace the manual process of logging items being checked in or out. The system comprises both a website and app that will have a live tracking method that can show what's checked out by whom. The app will also have a Machine Learning component that allows users to use their device's camera to locate items' locations. This will enhance the current system's efficiency as this system can update the log faster than it is being done manually. Replacing the manually updated log, with a live-updated one, allows users to also know the stock and status of items they may need.

2.2. Overview

To check-out an item, a user can direct themselves to either the app or website. Through either, they can add however many items they need to a cart. They will then be prompted to check a box that agrees that they will return items in the condition needed (if it's an item that needs to be returned). The app version will allow users to take a picture of all the items they're checking back in. The machine learning model will then automatically log those items as being checked back in. This process can be shown visually in **Figure 1**.

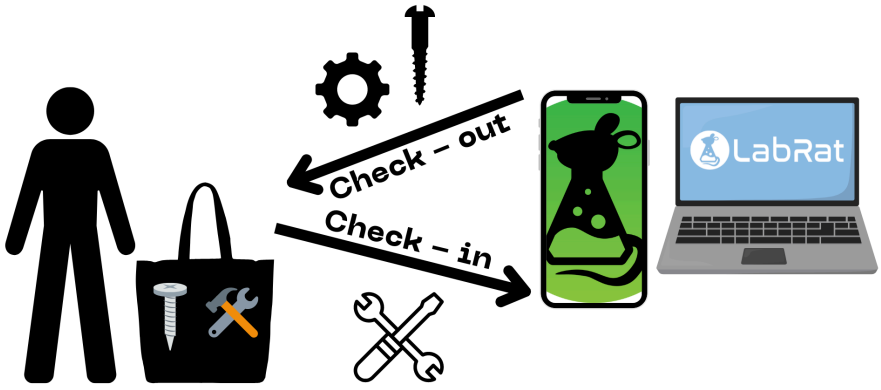


Figure 1. Visual Diagram of how the app and website will work synonymously for checking items in and out.

2.3. Referenced Documents and Standards

Table 1. Referenced Documents

Document Name	Revision/Release Date	Publisher
Google's Machine Learning Crash Course	2023	Google Machine Learning Education
Building a Python Image Recognition System	2024	Cloudinary
OpenCV Library	4.10.0	OpenCV
Flutter document	2017	Flutter

3. Operating Concept

3.1. Scope

The scope of our project is centered around people at Texas A&M. The intent of this project is for research students and faculty to use the application to find items in their associated lab. We will include functionality for other labs at Texas A&M to register their inventory and users, so any Texas A&M labs that aren't included in the initial design can make use of our product. Theoretically we can add functionality for labs outside of Texas A&M to use our services, but it was a deliberate decision to limit our scope to people at this university.

3.2. Operational Description and Constraints

Our project has two interfaces, a native mobile application and a web-based application. From the web-app, users will be able to log in and access all inventory items in the database for their assigned lab. If a user has access to multiple labs on campus, then they will be able to switch access between each for individual access.

Figure 2: Landing Page for Web-app

From the mobile-app, users will be given the same abilities as the web-app in addition to an item-finder. If a user cannot find more of a certain item, they can take an image of what they want to find more of and the machine learning model will help the user. Assuming that there are no barcodes or QR codes on the items, items can be found in the laboratory using this model.

3.3. System Description

The project is divided into four subsystems. These consist of the database, the web-app, the mobile application, and a machine learning model. Below is a visual representation of

how each part interacts:

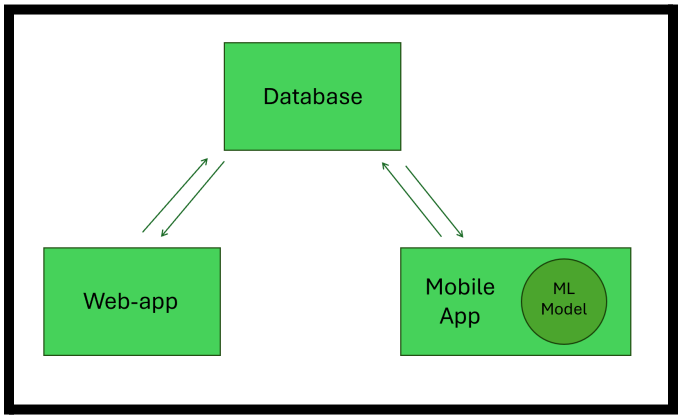


Figure 3: System Layout Description

As you can see, all parts of the project connect back to the database. Whether a user is operating on the web-app side or the mobile-app side, the database will be able to receive requests from both and update all information dynamically. For example, if changes to the database were made from the mobile application, then the database should populate the pages of the web-app automatically. The machine learning model is only accessed from the mobile application, and any time the camera is used to find items or check in/out those items.

3.4. Modes of Operations

The modes of operation for the software will be via android smartphone or computer. The camera functionality will only be able to be accessed through the app on the smartphone. It can be used in any environment where the device being used is functional. It is intended to be used in research labs or similar facilities requiring storage (Libraries, warehouses, ect.).

3.5. Users

As of now, there are only two defined users, but we have the ability to add new permission levels for new types of users later in development. Currently, we have “student” and “staff” roles. Students have the ability to check into their lab and look at all the inventory of their respective lab(s). They can check-out any items they need, see which items are available, and see how much of each the lab has for their own reference. Staff users have slightly more abilities. They are able to see the recent login times of other users and see who the most recent borrower of any item was. This will make tracking students’ usage easier for staff.

3.6. Support

Our website along with the smartphone application will have a FAQ page. This will be the main form of support. Any questions not included in the FAQ can be emailed to the team member whose subsection the question pertains to.

4. Scenarios

4.1. Check-out as Student

As a student, users should be able to login to observe all available tools in the lab, see which ones are available, locate where the item is stored, and check-out the item they need.

4.2. Check-in as Student

As a student, users should be able to login to return any items to the lab. Upon returning, the student should be able to take a picture of the item, use the machine learning model to recognize which item it is, and change the status of the item to “checked-in” so anyone else can use it

4.3. Monitor Students as Staff

As a member of the lab’s staff, users should be able see the recent login times of their students and monitor the most recent borrower of tools from the lab. If something is damaged, then staff members should be able to find the person responsible for damage.

4.4. Add/Manage Inventory as Staff

As a staff member, I should be able to add new items to the inventory of my lab, edit the attributes of any of the items, remove any unnecessary items, and deactivate items that I deem unusable.

5. Analysis

5.1. Summary of Proposed Improvements

This research lab inventory tracker will be able to provide improvement for the current traditional inventory management process. With the use of the website and application, the user will be able to have real-time updates of such inventory. This helps reduce overstocks or stockouts. The automatic inventory updates help minimize manual work that could eventually lead to human error. By having one central database server, there is a reduction in inconsistency, as staff and students are both accessing the same data.

5.2. Disadvantages and Limitations

While the proposed inventory tracker system offers various improvements, it also comes with disadvantages and limitations. There is a possibility of running into server issues that can cause data loss or website downtime. If the server becomes unavailable, the user will not be able to access the inventory data, which can cause delay in tracking or order processing. The machine learning model has the possibility of not being as accurate. As the user uploads a picture of the inventory, the machine learning model could accidentally end up confusing a piece of inventory for another, and therefore updating the wrong data. Another issue could be that item locations are not correct due to human error, such as item misplacement.

5.3. Alternatives

Some alternative solutions for this inventory management include using Excel or Google Sheets. This allows for manual data entry without having to depend on a database server. Using spreadsheets is free and is very simple to use, but not dynamic for a website management system.

5.4. Impact

A website and application inventory tracker has environmental positive impacts. There is a clear reduction in paper consumption, as there is no need for any physical records. By being able to avoid over stockings, laboratories are able to reduce overall waste. Overall, laboratories are able to reduce human error and improve productivity. In return, more reliable information can be provided to users.

Research Labs Inventory

Jonathan Moore
Evan Ross
Lizzett Tapia
Sumedha Bhattacharyaa

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – Draft
26 September 2024

Functional System Requirements
for
Research Lab Inventory

Prepared by: Team <55>

_____
Author Date

Approved by:

_____
Project Leader Date

_____
John Lusher, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	9/26/2024	Moore,Ross, Tapia, Bhattacharyaa		Initial Submission
2.0	12/5/2024	Moore		Final Edit

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Purpose and Scope

The purpose of this Research Lab Inventory management system is to create an efficient system for tracking the inventory of research labs at Texas A&M University. This project will aim to develop a website and smartphone application where users can check in and check out lab items, which will be categorized using photo recognition machine learning. Staff members will be able to make large changes to their inventory, while students can select certain items for personal use. Ultimately, this project is designed to make the lives of researchers at this university easier with an intuitive system of organization.

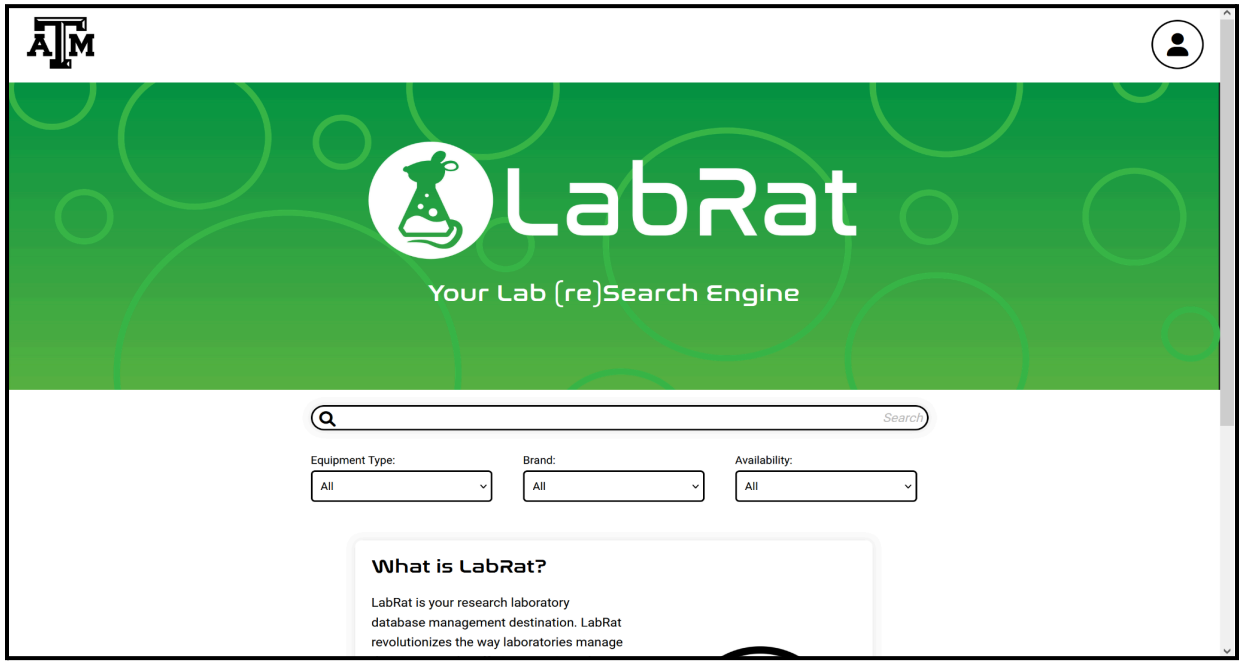


Figure 1. Home Page of Website

1.2. Responsibility and Change Authority

Each team member is responsible for their own subsystem meeting such requirements. Any changes to be done shall be discussed among each other and the project's sponsor, Shima Hasanpour. Since each subsystem acts separately, each team member is solely responsible for their work.

2. Applicable and Reference Documents

2.1. Applicable Documents

There are no applicable documents for this project. All instructions and requirements are given directly from our sponsor by word of mouth during our weekly meetings with her.

2.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Document Name	Revision/Release Date	Publisher
Google's Machine Learning Crash Course	2023	Google Machine Learning Education
Building a Python Image Recognition System	2024	Cloudinary
OpenCV Library	4.10.0 / 4 June 2024	OpenCV
Flutter document	2017	Flutter

Table 1. Reference Documents

2.3. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as "applicable" in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

This section defines the minimum requirements that the development item(s) must meet. The requirements and constraints that apply to performance, design, interoperability, reliability, etc., of the system, are covered.

3.1. System Definition

The Research Lab Inventory Tracker is designed to efficiently streamline the process of tracking and managing inventory at any Texas A&M University laboratory. The objective is to create a website and a smartphone application where students, professors, and administrators can log in, track inventory, and update inventory records. The system integrates a machine learning model into the smartphone application. It will allow users to scan and upload images in order to locate items, as well as help out with the check-in and check-out process. The inventory data is stored in a database server that syncs with both the website and application.

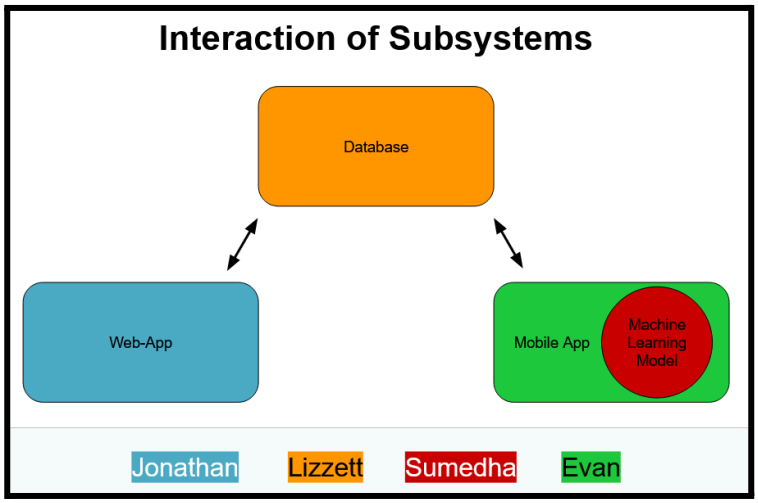


Figure 2. Block Diagram of System

The block diagram provides an overview of the subsystems in the Research Lab Inventory Tracker, as well as their interconnections. The block diagram consists of the database, the website, the mobile application, and the embedded machine learning model. The database is the main repository where all inventory data is stored. It contains information on the users, such as their usernames and email addresses. It also contains item information, such as item location, quantity, and its supplier. The database serves as the backbone of this overall system. It is what keeps track of all the data and ensures there is consistency between these interfaces. The website and mobile application interface allow users to interact with the system. Students will be able to check-out and check-in items, as well as see the location and quantity of the items. Staff members will have more of an administrative control, as they will be the ones to process any new orders. The machine learning component will enable users to scan and immediately identify, and locate, items through image recognition.

3.2. Requirements

3.2.1. Functional Requirements

3.2.1.1. Must Allow User Login

The website, as well as the mobile app, shall allow users to register for accounts and login via the account credentials (email and password). User information will be stored in the created database.

> *Rationale: It is important to know which user is using the software and checking out items in case of a damaged or missing item.*

3.2.1.2. Inventory Management

Both students and staff shall be able to view the current inventory details, such as the item name, item description, location, and availability. When students check-in and check-out items, the inventory data should automatically update. These updates will be able to be seen on both the website and mobile application. The system should generate alerts when stock gets low for any item.

3.2.1.3. Concurrent User Load

The System should be able to support the load of at least 100 users at once without performance quality issues.

> *Rationale: It's likely that every lab member should use this system in the case that it's implemented, so load support is required.*

3.2.2. User Interface Requirements

3.2.2.1. Data Display and Access

The user interface must present all the inventory data stored within the database in a clear and understandable manner. Items in the database will be displayed in a table with important information, such as item name, description, quantity, unit, location, and supplier.

3.2.3. Performance Requirements

3.2.3.1. Machine Learning Item Categorization

The system should accurately categorize items at least 93% of the time by the end of the training period for the machine learning model.

> *Rationale: This is necessary for machine learning to be used as a replacement for manually entering items to check them back in.*

3.2.3.2. Item Scanning and Recognition

The system should accurately recognize items as separate entities.

Rationale: This distinction is necessary to prevent items being grouped together when there's many items in one scan/picture.

4. Support Requirements

For use of the website application the user is required to have a computer with Google Chrome, Mozilla Firefox, or Safari as the browser. For the mobile application the user is required to have a mobile phone with the Android operating system using API 16 (Android 4.1) or above. If the user wishes to use the machine learning component of the mobile application, the user's phone must also have a functional camera.

Appendix A: Acronyms and Abbreviations

GUI	Graphical User-Interface
ORM	Object-relational Mapping
API	Application Programming Interface
IDE	Integrated Development Environment
ML	Machine Learning
API	Application Programming Interface

Appendix B: Definition of Terms

GUI	Visual interface allowing users to interact with electronic devices
ORM	A technique used in programming allowing for the connection between object-oriented programming languages to relational databases
API	A set of rules and protocols that allow software programs to communicate with each other
IDE	A software application that aids programmers in developing code
ML	An ML system allows computers to learn without directly programming anything

Research Labs Inventory

Jonathan Moore
Evan Ross
Lizzett Tapia
Sumedha Bhattacharyaa

Interface Control Document

REVISION – Draft
26 September 2024

Interface Control Document
for
Research Lab Inventory

Prepared by: Team <55>

_____
Author Date

Approved by:

_____
Project Leader Date

_____
John Lusher II, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	9/26/2024	Moore,Ross, Tapia, Bhattacharyaa		Initial Submission
2.0	12/5/2024	Moore		Final Edit

Table of Contents

List of Tables

No table of figures entries found.

List of Figures

1. Overview

This Interface Control Document (ICD) will outline the communication and interaction between the research lab inventory tracking software, its connected systems, and its users. It will detail how our software is designed to facilitate these interactions to guarantee that the system operates effectively and efficiently. This document will also specify the technical and functional requirements for the final integration of the system, providing a clear guide for integration and operational compatibility. The goal is to support the deployment of our solution to Texas A&M University's issues with lab organization.

2. References and Definitions

2.1. References

Google's Machine Learning Crash Course
2023
Google Machine Learning Education

Building a Python Image Recognition System
2024
Cloudinary

OpenCV Library
4.10.0 / 4 June 2024
OpenCV

Flutter Documentation
2017
Flutter

2.2. Definitions

API	Application Programming Interface
SQL	Structured Query Language

3. User Interface (Web-App)

The UI of the web application is designed to streamline the inventory management process in Texas A&M University research labs by offering an intuitive and efficient user experience. The layout will help enable users to quickly search for equipment, filter results based on specific criteria, and access relevant information about lab inventory. The interface is built for ease of navigation, allowing users to perform actions such as signing in, registering, and managing their accounts with minimal effort. Each element of the UI is aimed at simplifying inventory tracking and enhancing the overall efficiency of lab operations. The design will be further elaborated upon in dedicated sections for each functionality.

3.1. Frontend

3.1.1. Home

The home page is designed as the home for users. Most of the functionality will be built into this page, including the search functionality. When users sign in, their initial replaces the person icon on the top right corner to signify a session is active for that user. After signing in, the page will populate with a table of the current inventory of the database so the user can check-out/check-in items. A cart feature is also planned. In this case, a cart appears within the dashboard bar above when items have been added to the user's desired list of items.

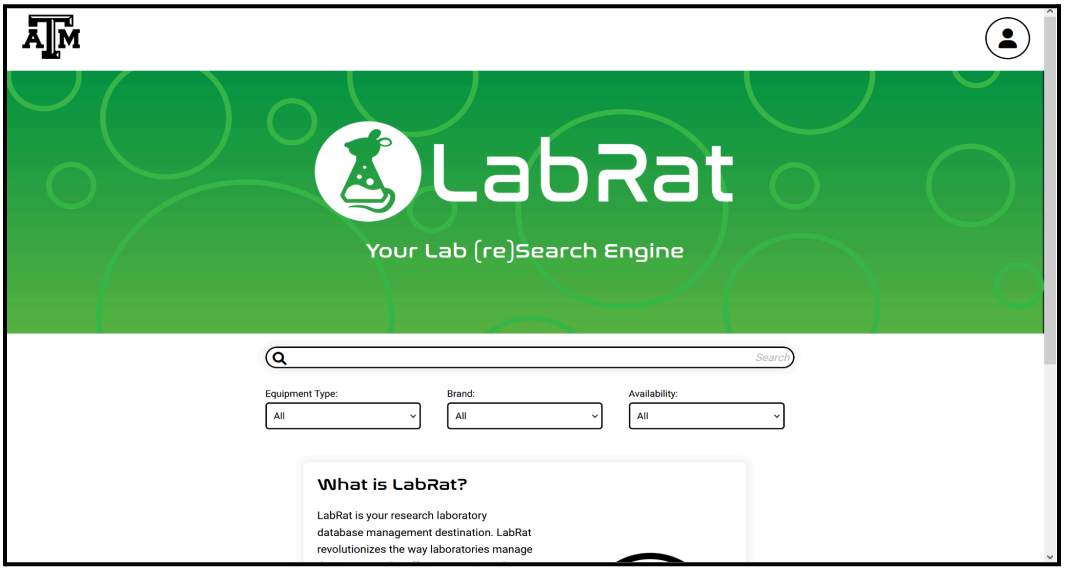


Figure 1. Home Page of Website

3.1.2. Check in/out

The checkout page will be added to finalize any changes to the user's desired items. On this page, users will have to check a box agreeing to our terms of use, so that any items borrowed from the lab will be protected under an agreement.

3.1.3. Registration

The registration page is already fully completed. It provides the user with a form asking for name, email, password, and a password confirmation. All this information is stored in the “users” table of our SQL database. All passwords are encrypted so that no plaintext passwords are stored to protect our users.

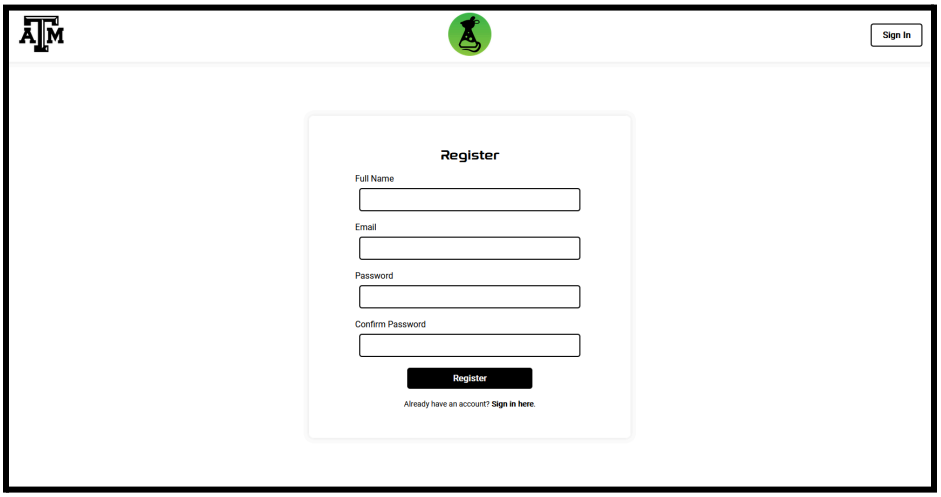


Figure 2. Register Page of Website

3.1.4. Sign-in

The sign-in page is already mostly completed. When a user types their email and password, a session begins under their id and is stored in their browser’s cookies. Upon return to the website, their sign-in will persist and they can return to their previous session. This can be ended by signing out from the home page.

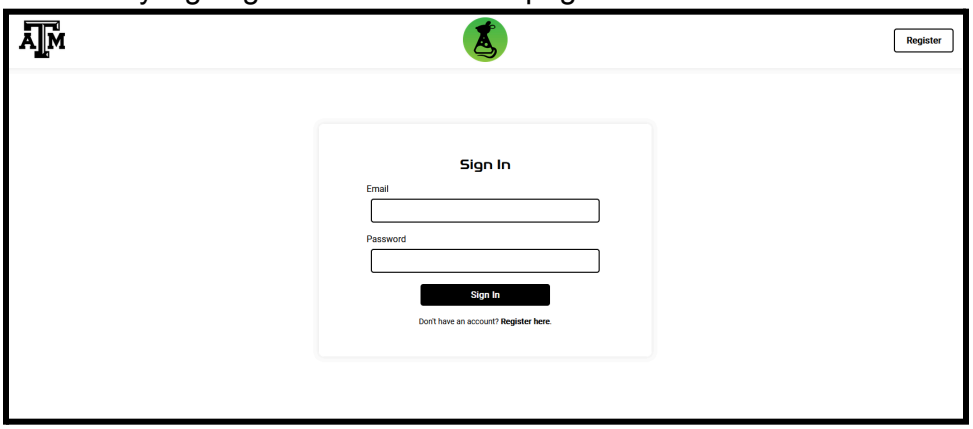


Figure 3. Sign-in Page of Website

3.1.5. Account

The account page gives users the ability to change their information. They have the option to change their name, email and password, but to change their affiliated TAMU labs, it has to be approved by a professor or administrator from that lab.

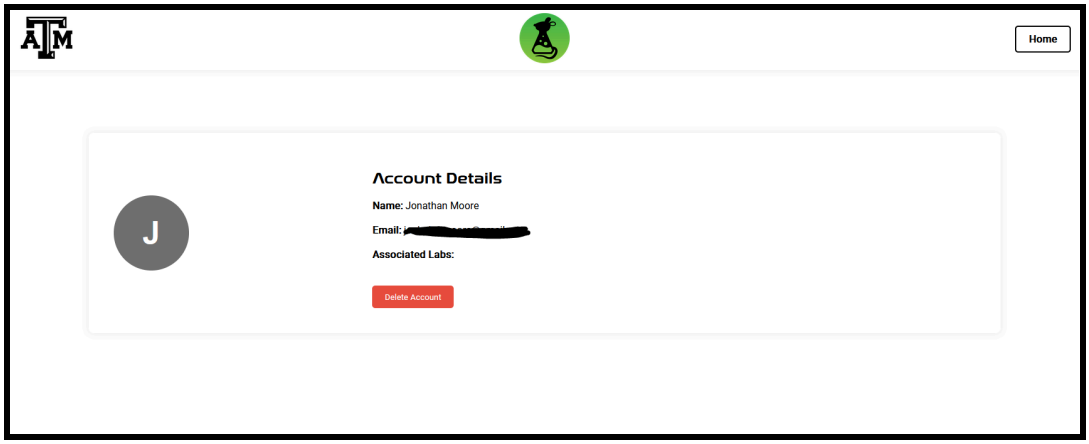


Figure 4. Account Page of Website

3.2. Backend

In order to accomplish the functionality of this website without the use of a framework (such as React, Vue, Angular), we are choosing to use Javascript and Node.js to host and operate the website. This choice was deliberately made by the team to make development move as quick as possible while still providing great results.

4. User Interface (Mobile App)

The purpose of the mobile app is to give users the ability to easily and intuitively track, manage, and make changes to the inventory of a Texas A&M University research lab from their Android smartphone. The mobile app is designed to have similar functionality to the website, as well as similar pages and navigation between them. This is so that someone familiar with the website is able to easily navigate the mobile app counterpart, and vice versa. One unique aspect of the Mobile app is that it will allow users to use their camera to scan and identify items in the lab. Once an item is scanned, the user will be sent to the check in/out page for that item. This will be achieved through a machine learning model that is trained to recognize the items in the lab.

4.1. Pages

4.1.1. Login Screen

This is the first screen that users see when they open the mobile app. They will be given the option to enter their email and password to sign in if they already have an account. The password will be obscured for security. If they do not have an account yet there will be a button which brings users to the registration page.

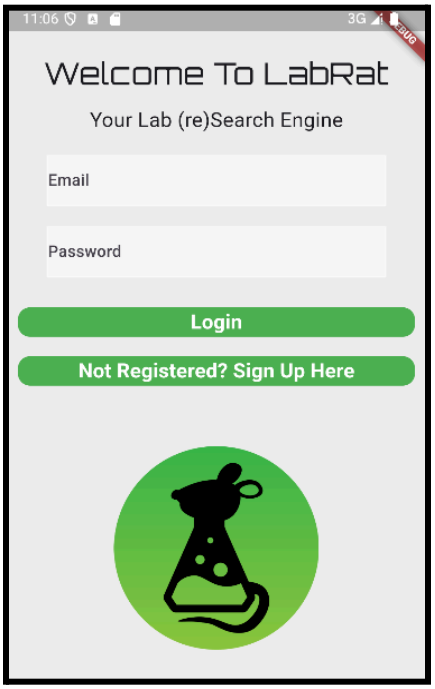


Figure 5. Login Screen of Mobile App

4.1.2. Registration Screen

In this screen users will be able to register for an account. They will be prompted to enter their email and password, as well as password confirmation. User data will be stored in the SQL database “Users” table.

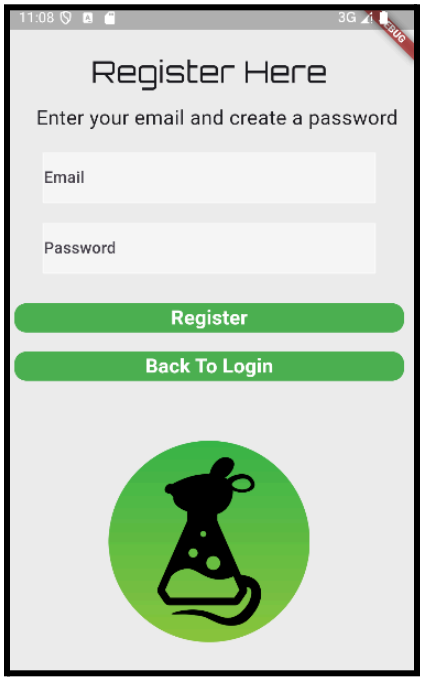


Figure 6. Registration Screen of Mobile App

4.1.3. Home Screen

The home screen will be the main page in the mobile application. It is where users will be able to search for items either through a search bar, or through the drop down menus on the page. It will allow users to see what Items are available to be checked-out/checked-in. When a user selects an Item from the home screen they will be sent to the check out/in page. This is also the page where users will be able to open their camera to scan an item. When a user scans an item they will be sent to the corresponding check in/out page.

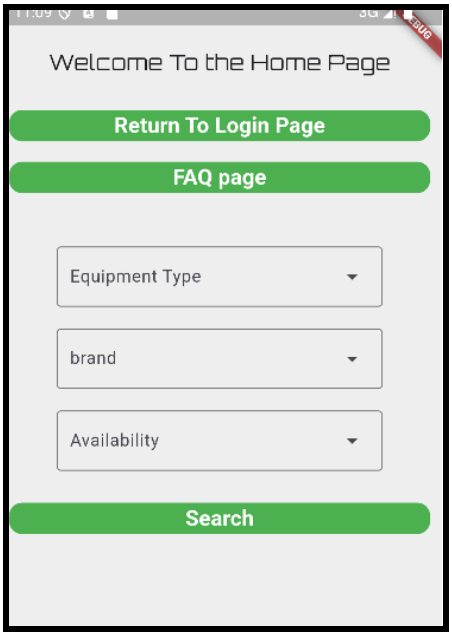


Figure 7. Home Screen of Mobile App

4.1.4. Check Out/In page

This is the page that will allow users to check out/in items. After a user selects an item from the home page they will be sent to this page, which will provide a brief description of the item as well as the ability to agree to the terms of use and check out/in the item.

4.1.5. FAQ page

This page will provide answers to frequently asked questions, such as how to create an account and check in/out items.

4.2. Backend

The Mobile application UI is created using Android Studio along with the plugin Flutter. Flutter uses the dart coding language, which is specifically designed for mobile app development. A Flutter plugin called SQFLITE is used to connect the application to a temporary SQLite database.

5. Database Server

The database is the central point for both data storage and data retrieval. It helps all the subsystems work together and interact with user data and inventory information.

5.1. Database Access

Access to the database will depend on the user's role, whether they are a student or staff. Students will have a more restricted access, as they can only modify check-in and check-out records. On the other hand, staff members will be able to add or remove items.

5.2. Data Formats

Each data entity in the database will have a defined structure. For example, the *Users* data table will have:

- UserID
- FirstName
- LastName
- Email
- Role
- Username
- Hashed-Password

The *Items* data table will have:

- ItemID
- Name
- Description
- CategoryID
- Quantity
- Unit
- Location
- Supplier

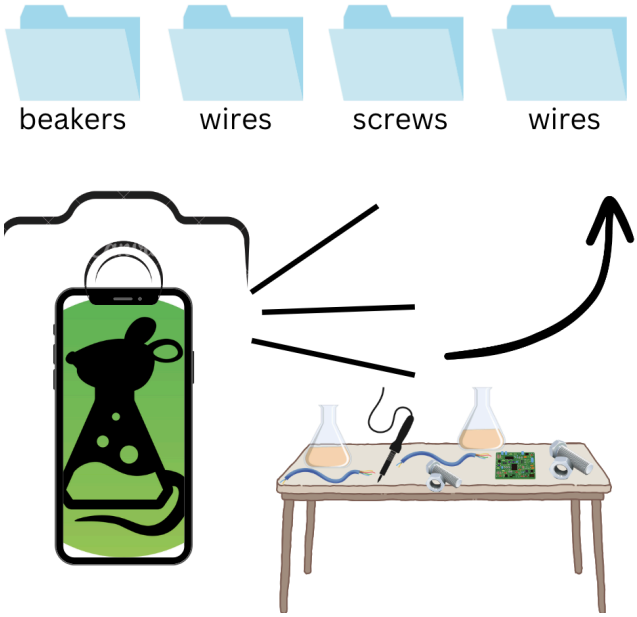
Having a standard data format helps keep consistent communication and proper data retrieval.

5.3. Security and Data Consistency

Data that gets exchanged between the website/mobile app and the database will be encrypted. This helps make sure that confidential data, such as login credentials, cannot be intercepted. Only authorized users will have access to the data. Audit logs will be beneficial, as it allows for traceability in the case that there are any outages or data breaches.

6. Machine Learning Model

The purpose of the ML Model is to allow users to use the app component on their mobile devices to scan all their items, and have them automatically checked-in. This includes identifying how many items they're checking in, as well as the category of each. The overall goal is to increase the efficiency of labs at TAMU, and this is a component of that increase. This also aims to enhance the experience of all app users.



6.1. Pre-trained model

A pretrained model optimized for image classification purposes. The chosen model is called ResNet50 and was made from TensorFlow - an open source machine learning platform.

6.2. Dummy Data Sets

The dummy data sets are the initial data that will be used to train the model specifically with common items in a lab. There's different resolutions, sizes, and formats of images in every sub-category.

6.3. Real Data Sets

Eventually, pictures of actual items from labs will be acquired in order to further train it with the exact items it will be witnessing through user application.

6.4. Automated Function

An automated function will be implemented in the Python code of the ML Model, to enhance the speed of training. This function will allow for the model to run through all the images in a folder, so that the path to each image doesn't have to be manually entered every time.

7. Device Interface Software and Hardware Requirements

7.1. Software Requirements

7.1.1 Software Requirements of Web Application

The website will be compatible with the latest versions of Google Chrome, Mozilla Firefox, and Safari.

7.1.2 Software Requirements of Mobile Application

The device used to run the mobile application must use the Android operating system. As stated in the Flutter Documentation, Android API 16 (Android 4.1) or above is required to run the mobile application.

7.2. Hardware Requirements

7.2.1 Hardware Requirements of Application Server

The server must be equipped with storage to accommodate for large amounts of data for inventory.

7.2.2 Hardware Requirements of Mobile Application

Though not required, the mobile device must have a functional camera if the user wishes to utilize the machine learning aspect of the mobile application.

Research Labs Inventory

Jonathan Moore

WEB APPLICATION SUBSYSTEM FINAL REPORT

REVISION – Final
5 December 2024

Subsystem final report
for
Research Lab Inventory

Prepared by:

_____

Author Date

Approved by:

_____

Project Leader Date

_____

John Lusher, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	12/5/2024	Jonathan Moore		Final Release

Table of Contents

List of Tables

None.

List of Figures

Web Application Subsystem Report

1.Introduction

For this project, it was necessary to create a website that any user at the laboratory could access at any time to manage their items in the lab. This website was designed to be a tool that people can interact with at a simple level without having to know how to use a database. Management of users (such as students, professors, administrators, etc.) is simple and intuitive and allows for many different permission levels for the different types of people that will be interacting with our project. After users check out items, there are certain types of items that need to be returned to the laboratory, and this website tracks that information and will send email reminders to users once full database integration takes place.

2. Development Environment and Tools

The web application was developed without a web framework. This was a deliberate choice by myself so that development could move as quickly as possible since there were a lot of goals set for this subsystem at the beginning of the semester. I used a combination of HTML, CSS, and Javascript to manage the frontend of the website, and Node.js, SQLite3, Python, and additional Javascript to handle all server-side responsibilities. For all validation, I created regression tests and input validation tests using Playwright. With all of these tools, the web application was successfully completed and is ready for integration.

3. Data Preparation

3.1 Data Setup

Using SQLite3, I have a data set of placeholder information that gets re-seeded each time the website boots up on a local server. There were many times I've had to completely wipe the database for testing reasons during development, so I have written a python script to run each time the server notices an empty database at run time. Using this, I was able to move through development very smoothly. This will not appear in the final product, as integration would eliminate the need for a local database, but it is important to mention it since there would be no displayable functionality without it.

3.2 Database Configuration

The database structure is kept very similar to the other subsystem that was made for this purpose, with minor alterations to fit the needs of the website; all of these changes are miniscule and will take minutes to change once real integration begins. Speaking of which, there is no need to worry about the website not connecting to the database correctly next semester, as we have all stuck with using uniform SQL formatting so that all we will need to do is "unplug" our local databases in favor of the new, live database.

3.3 Data Validation

All inputs that the user gives through the website are tested and validated depending on what is entered into each input field of the application. If certain fields are required, then the website will alert the user of incomplete information and block it from reaching the database. Emails are handled differently than other pieces of information on the application. Since emails should be unique to the user, only one account can be registered per email. On top of this, since sending emails is planned with database integration, each email must be valid upon registration. There is a simple validation check for email formatting, so invalid email types are not allowed during registration. This is just one example of protection the website provides to the database and its users.

4. Application Design

4.1 Logo

Below, Figures 1 and 2 are logos that I designed at the beginning of the semester. These were designed using Adobe Illustrator, so we own the right to use these images. The name is a nod to the popular idea of testing rats in laboratories, with the rat image resembling an Erlenmeyer flask. Both of which could be found in labs, so the idea stuck.

Figure 1: LabRat Logo

Figure 2: White LabRat Logo with Title

4.2 Fonts

All fonts used across the app are publicly available, and we have the licenses to use them. The fonts are also shared with the mobile application for parity between systems.

4.3 Intuitive Design

As for the layout and structure of the website's appearance, it is designed to be as user-friendly as possible. With experience in creating accessibility-focused websites, I set a goal to achieve a simple, clean look that can guide users through its pages without having pages of text to read. Using icons that convey functionality (gear icons for settings, person icons for account, shopping cart icons for the cart, etc.) help declutter all of the pages while also providing meaning to the user. Tab ordering for each page was a focal point; people who may not be able to use a mouse (because of restrictions based on physical ability or unavailability of a mouse) are still able to tab through each page, type what they need to, and hit enter to send information. To summarize, intuitive navigation and accessibility was a big focus of the project, and has led to quick understanding from test users.

5. Features

5.1 Registration and Sign In

All users land on the homepage without a login. From here (Figure 3), users can read a small passage clueing them on what our project is about. From there, users can access a Frequently-Asked-Questions (FAQ) page that can offer more guidance.

Figure 3: Home Page

To register for an account, users click the person icon on the right which expands for more options.

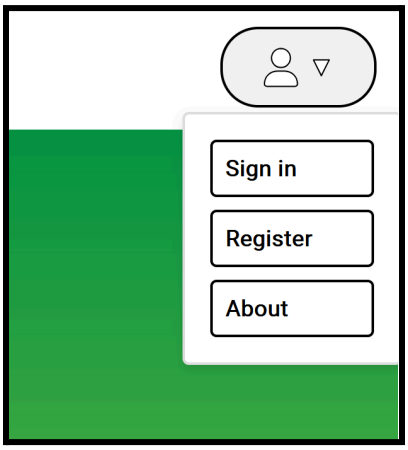


Figure 4: User Dropdown

From here, users have the option to Sign in, Register, and learn more about LabRat. Below are the registration and sign-in pages. Each function exactly how you'd expect. Users must register an account before being able to login, and the website alerts the user if emails or passwords are incorrect upon sign-in.

Figure 5: Registration Page

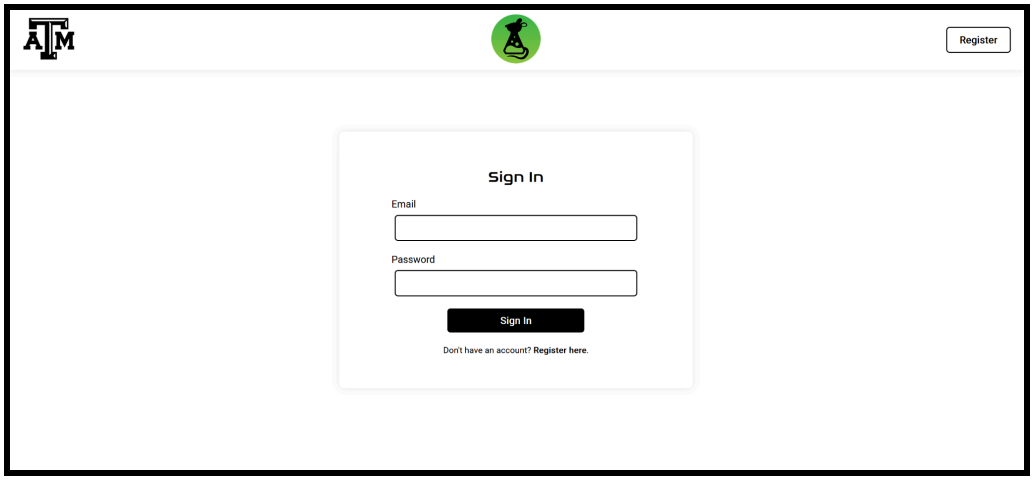


Figure 6: Sign In Page

After signing in, users are automatically routed to the home page, which is now repopulated with new information and features. Every new account starts off as a student account, so if users need permission level changes, they must consult a currently active Administrator. All of this information is available in the FAQ page.

5.2 Search and Filter

After signing in, the home page appears like it does in Figure 7. It consists of a search bar, filters for category, supplier, and availability, and a table filled with items from the lab.

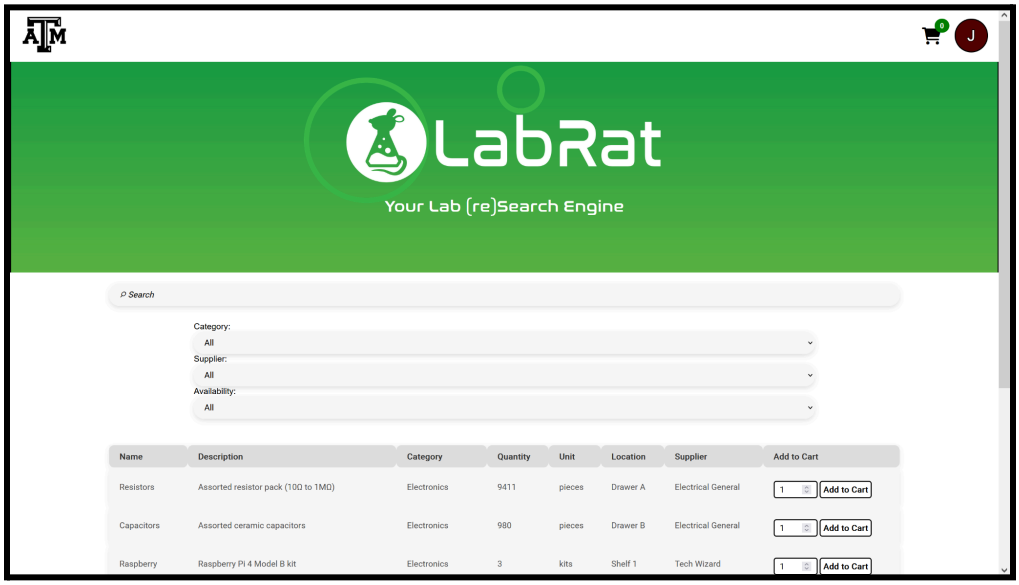


Figure 7: Search Feature

On each entry in the table, users are able to select how many of a chosen item they want to checkout and add them to their cart. Items that are out of stock are greyed out and do not allow people to use them.

5.3 Checkout

Once items are added to the user's cart, the counter bubble on the shopping cart icon next to the user's initial increases with the number of selected items.

Figure 8: Cart Icon

After this, the user can click the cart to view its contents. They are listed in bundled quantities, and users can change the quantity of an item or remove it entirely from the cart as well.

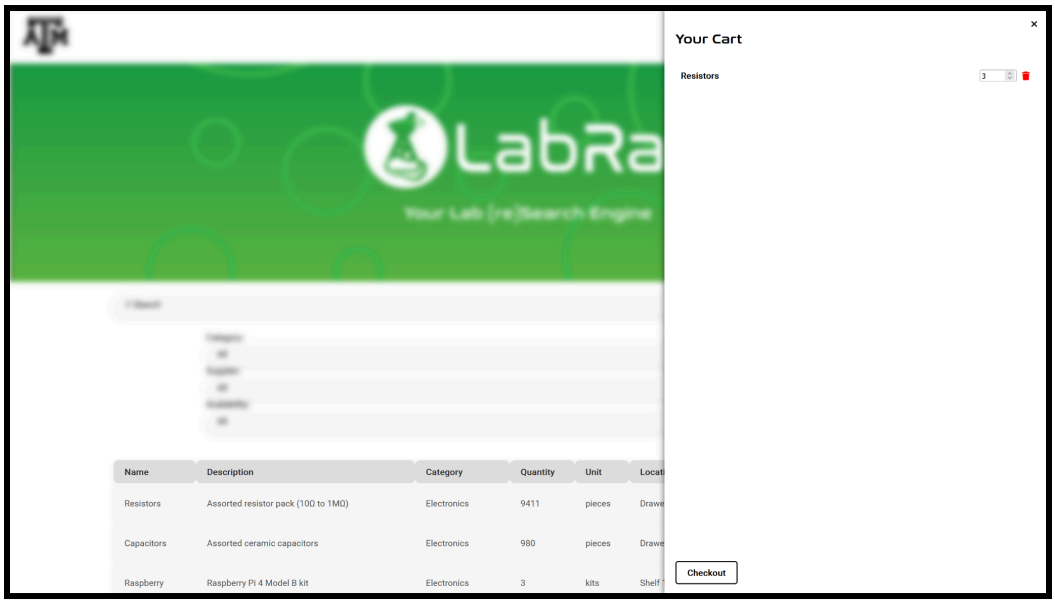


Figure 9: Cart

After clicking checkout, the stock decreases in the database, and the order is added to the orders table for management to observe. However, if an item belongs to the lab and needs to be returned eventually (soldering irons, etc.), then an agreement checkbox appears for the user to sign. This agreement holds users liable for damage of lab property if returned in poor condition or missing. After clicking the agreement, then the checkout button appears for the user. This time, the checkout operates differently. Each time a user checks out a returnable item, it is flagged and added to a list of items that user needs to eventually return. All contents of the list are observable from management's point-of-view, and once database integration is complete, automatic emails can be sent to users reminding them of their agreement and responsibilities.

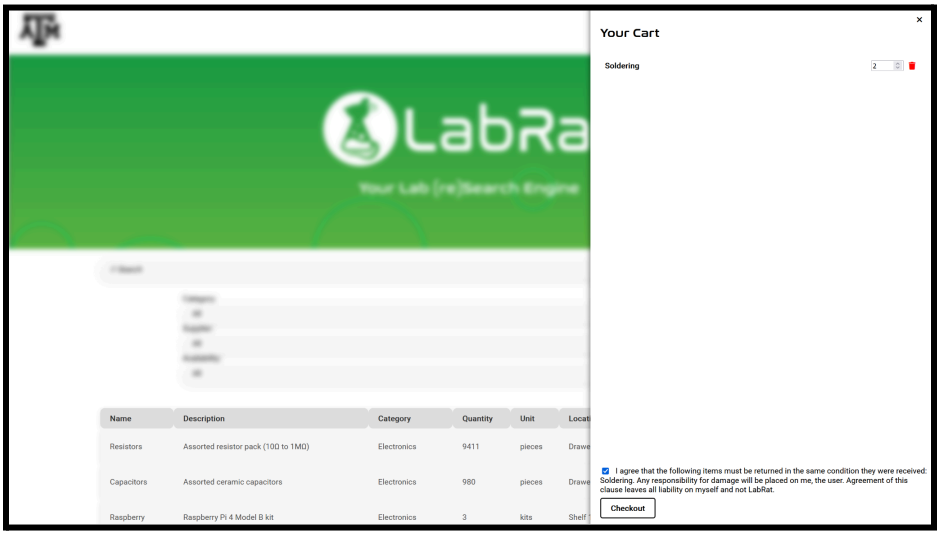


Figure 10: Cart Agreement

5.4 Account Settings

Every user has the ability to change their email, name, and profile color from the account page. Here, you can see information associated with your account. To change user information, users can click the gear icon to change them. The delete account button also works, but it warns the user and asks for confirmation.

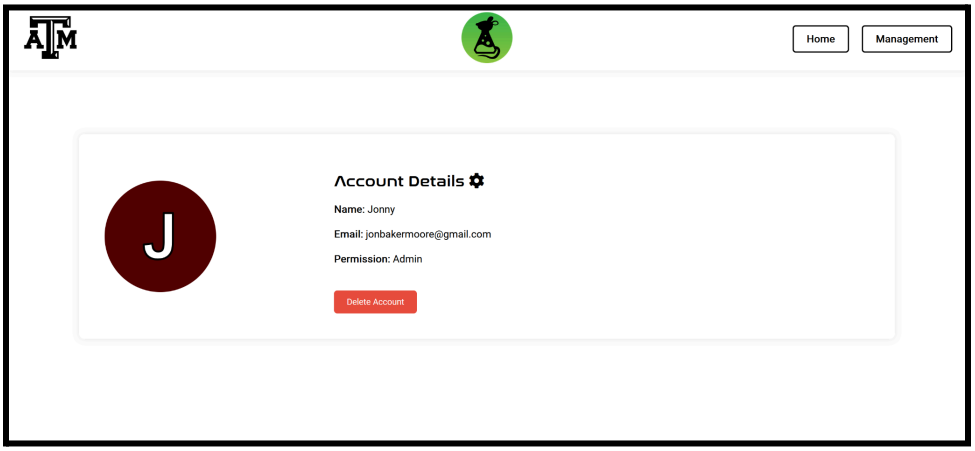


Figure 11: Account Page

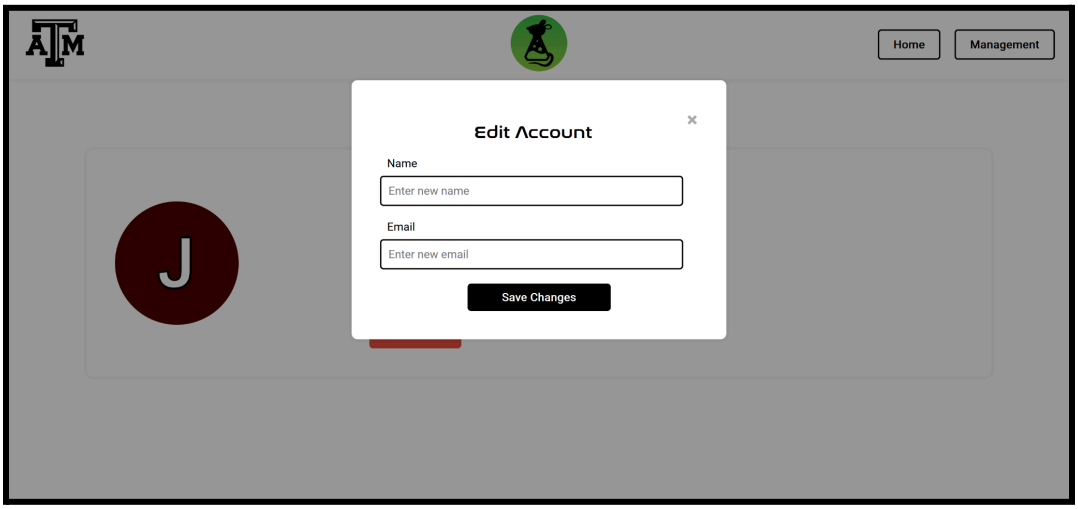


Figure 12: Edit Account Modal

There is also another unnecessary but fun addition of profile color customization. As you can see in Figure 13, a color modal appears after clicking on the user icon. Clicking a color automatically changes this for the user. The only way to know this is a feature is to hover over the user's initial, so it is more of a fun Easter egg for people to discover.

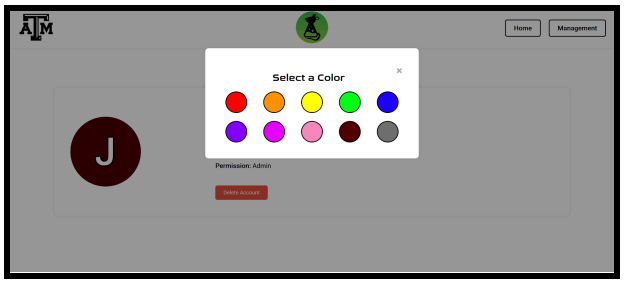


Figure 13: Color Modal

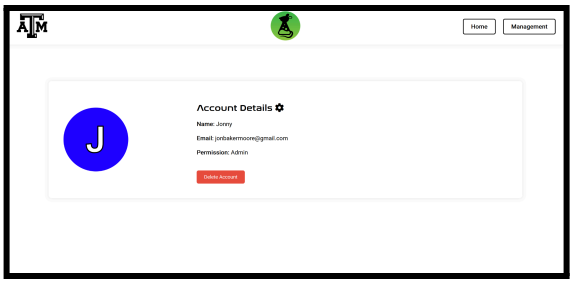


Figure 14: Color After Selection

5.5 Management

There is also the matter of managing users and data behind the scenes. For accounts that have “Professor” or “Admin” status, the management page is available. It is accessed through the account page or through the user icon dropdown menu on the home page. From here, there are three options: Users, Items, and Orders.

Figure 15: Management

Clicking on these options on the left displays a different table of information. From the Users tab, professors and administrators can see all the registered users, their names, emails, and the last time they were active. In Figure 16, there are a lot of fake users mostly for testing, so please ignore it.

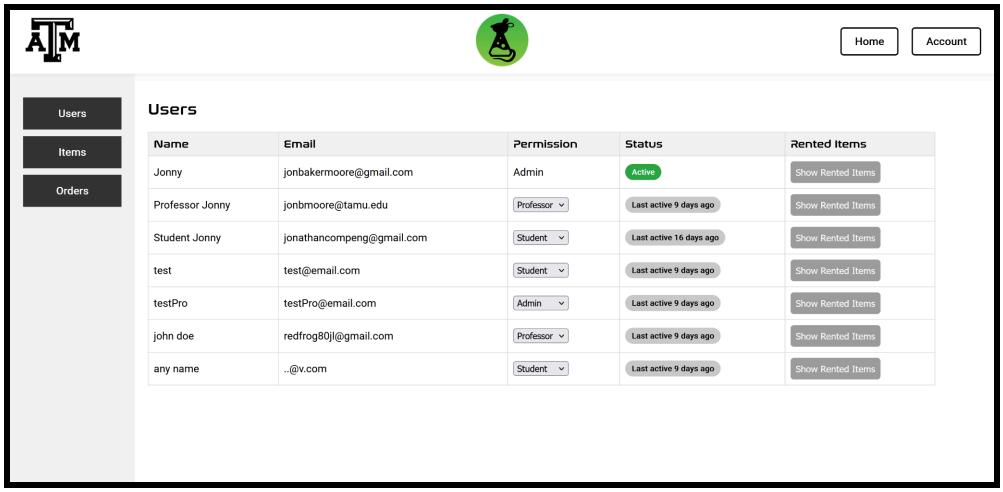


Figure 16: Users Table

From this page, only admins can change the permission level of users, so if someone does not meet the requirements, the column does not appear. Also in the table, the rightmost column has a button that lists all items that user is currently borrowing from the lab.

Below is the Items table. From this table, all items are listed, and on the rightmost column there are icons for editing and deleting items. Above the table there is also a button for adding items.

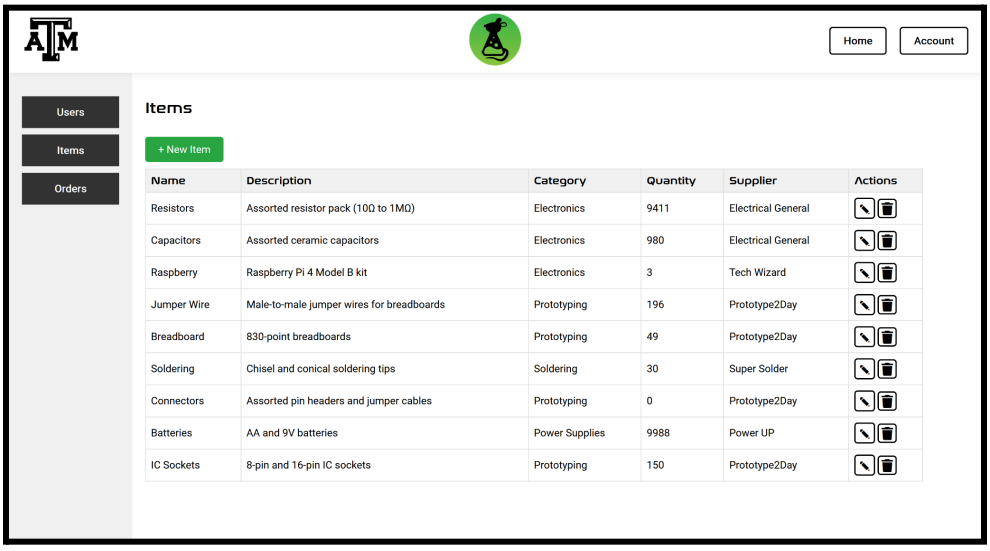


Figure 17: Items Table

The button for new items and the edit (pen) icon both create a modal with a form for the user to fill out. All fields are required for new items, but editing items can be done by only

changing a single attribute and submitting. For simplicity, Figure 19 is for new items, but the format is extremely similar for editing items.

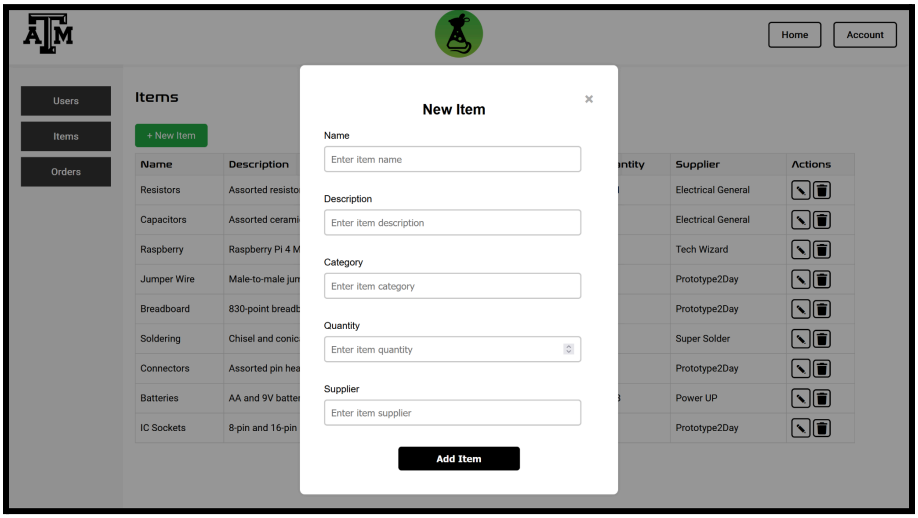


Figure 18: New Item Modal

When the lab receives more of an item, the idea is that professors or administrators go to this page to change the stock of that item. When new items are ordered, Figure 19 is what admins need to fill out to add them.

Finally, there is the orders page. Orders are listed in order of time and have headers listing who ordered the items, what their email address is, and when they ordered. Orders with multiple items have all of the items listed together and have each quantity listed.

Figure 19: Orders Table

6. Validation and Testing

Validation of this website is fairly straightforward. Using a testing library called Playwright, I am able to record interactions with the website and have those inputs tested across many different browser types all at the same time. I have many tests written, all to help with regression testing for changes between browsers and all fit the validation plan steps that I wrote at the beginning of the project. Below is the terminal output after running the tests followed by a snippet of the HTML file that reads what tests were performed and how long each one took.

```
PS C:\Users\jonba\Documents\GitHub\labrat> npx playwright tes
t

Running 52 tests using 4 workers
  52 passed (51.1s)

To open last HTML report run:

  npx playwright show-report
```

Figure 20: Playwright Terminal Output

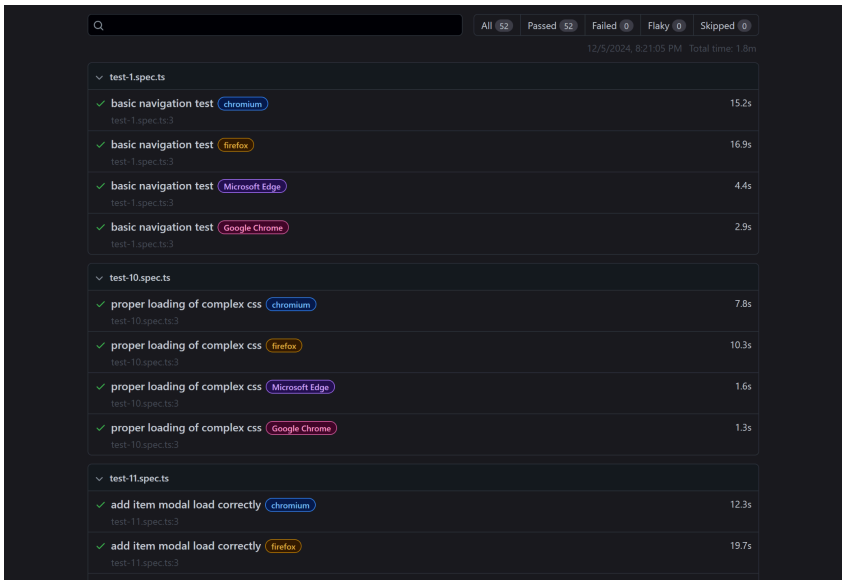


Figure 21: HTML Output

7. Challenges

Project went very smoothly. Hardly ran into issues, as my previous semester had a class over web development. Lucky to say that I did not experience many issues during development. Any issues were resolved very quickly once I consulted Mozilla's Javascript documentation.

8. Integration Plans

The only left to do when integration begins is get the database hooked up properly and troubleshoot any bugs that may appear. It should be a very short process when the time comes in ECEN 404. I also need to host the website, and I have chosen a service called Render; I chose Render for its Node.js server support.

9. Future Improvements

Aesthetically, there are a few things I would like to change about the search bar, filters, and items table on the home page after login. I don’t like the bubbly style I went with so I will likely be changing that soon.

10. Conclusion

The web application is ready for launch. Once I get it hosted online next semester and integrated with our database, most - if not all - requirements will be complete. It serves its purpose of being a tool for students and faculty alike. Hopefully, this can help solve Texas A&M’s laboratory organization issues.

11. References

Mozilla: mdn web docs | Javascript;
https://developer.mozilla.org/en-US/docs/Web/JavaScript

Research Labs Inventory

Lizzett Tapia

DATABASE SUBSYSTEM FINAL REPORT

REVISION – Final
5 December 2024

SUBSYSTEM FINAL REPORT
FOR
Research Lab Inventory

PREPARED BY:

_____

Author Date

APPROVED BY:

_____

Project Leader Date

_____

John Lusher, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	12/05/2024	Lizzett Tapia		Final Release

Table of Contents

List of Tables

No table of figures entries found.

List of Figures

1. Subsystem Introduction

The database subsystem serves as the main repository where all the inventory data is stored. It helps manage important information such as user details, supplier details, inventory stock, and orders. The database can be seen as the foundation of the research lab inventory tracker, as it enables secure data storage and ensures that there is consistency between all interfaces. This subsystem handles all backend data operations, which includes any sort of data update or retrieval.

2. Database Creation and Design

a. 2.1. Relational & Non-Relational Databases

When it comes to creating and designing a database, an important decision that must be determined is whether the data falls under a relational (SQL) or non-relational (NoSQL) model. Relational databases, such as MySQL and SQLite, use structured schemas to help organize the data into tables. This type of model works best when the data needs to be kept strict and consistent, show clear relationships, and overall be able to handle all sorts of data queries. Non-relational databases are simply the opposite. These models are schema-less and work with unstructured data. For this research lab inventory tracker, the data structure falls under a relational model. All the data sets created and used are interconnected within each other. This can be seen when users place item orders, or items belonging to specific categories and suppliers. Overall, working with a relational database helps establish data consistency and maintain system reliability.

b. 2.2. Implementation of Relational Design

To begin implementing this database relational model, five data tables were created: *Users, Suppliers, Category, Items, and Orders.*

- *Users*: Data table to store user information, such as name, email, role, username, and hashed password.
- *Category*: Data table to help group items into categories like *Basic Electronic Component* or *Power Component*.
- *Items*: Data table to help track inventory details for each item, such as the quantity and location.
- *Suppliers*: Data table to keep track of vendor contact information in case a user wants to place a specific order.

- *Orders*: Data table to log transactions, which helps link users to the items that they ordered.

Instead of manually executing SQL commands line by line and one by one, I used Python and SQLite to help with the creation of the database. Using Python is more efficient, as these tables are able to get populated each time the code is run. For example, with the *Users* table, it would have been a hassle to insert user data one by one. It would basically be never ending. Using the *CREATE TABLE* and *INSERT* commands in Python automated the operations quicker and overall helped avoid any potential human errors that could have occurred. The *Faker* library and the *Random* module were used to help generate random realistic data. A similar concept and approach was applied to the other data tables. The screenshot below shows a portion of the Python code used to define and create the *Users* table.

Figure 1: Python code for *Users Table*

```
#Create the Users table if it does not exist with all info of Id, firstname, last name, email, role, username, and password, and stores into columns
cur.execute('''
    CREATE TABLE IF NOT EXISTS Users (
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        FirstName TEXT NOT NULL,
        LastName TEXT NOT NULL,
        Email TEXT NOT NULL UNIQUE,
        Role TEXT NOT NULL,
        Username TEXT NOT NULL UNIQUE,
        PasswordHash TEXT NOT NULL
    )
''')

def generate_email(first_name, last_name): #Function to generate an email based on the first and last name w specific format
    return f"{first_name.lower()}.{last_name.lower()}@tamu.edu" #Use lowercase first and last name separated by a dot and append '@tamu.edu'

def generate_username(first_name, last_name): #Function to generate a username based on the first and last name
    return f"{first_name.lower()}{last_name.lower()}" #Combine lowercase first and last name with no space

def random_role(): #Function to randomly select a role for the user from the list
    roles = ['Admin', 'Student', 'Graduate TA', 'Professor']
    return random.choice(roles)

def insert_random_users(num_users): #Function to insert random user data
    for _ in range(num_users): #loops to create num_users random users
        first_name = fake.first_name() #generating a random first name
        last_name = fake.last_name() #generating a random last name
        email = generate_email(first_name, last_name) #calling the generate email function to create an email based on generated name
        role = random_role() #calling the random_role function to assign a random role to the user
        username = generate_username(first_name, last_name)  #callinf the generate_username function to genearte username based on names
        password_hash = "hashed_" + ''.join(random.choices('abcdefghijklmnopqrstuvwxyz0123456789', k=10)) #creating fake password (hashed_ + random
```

```
173         #Insert the random user data into the Users table
174         cur.execute('''
175             INSERT INTO Users (FirstName, LastName, Email, Role, Username, PasswordHash)
176             VALUES (?, ?, ?, ?, ?, ?)
177         ''', (first_name, last_name, email, role, username, password_hash)) #? represents placeholder for values to prevent SQL injection, and these
178 
179     conn.commit() #save all chnages made to the database (all the inserted rows and columnns)
180     print(f"{num_users} random users with @tamu.edu emails added successfully!") #print confirmation message to show how many users were added
181 
182 #Insert 10 random users with modified data
183 insert_random_users(10) #the number can vary, therefore can change it
184 
185 #Fetch and display the Users table
186 cur.execute("SELECT * FROM Users;") #fetching all rows from the Users table
187 users = cur.fetchall() #retrieve all rows
188 
189 for user in users: #loop through the list if users
190     print(user) #print each row of users
191 
192 conn.close() #Close the connection
193 
194 #summary: code create 10 random users (num can be chnaged) with random first and last name,
195 #   email formatted as frst.last@tamu.edu, a random role assigned to the user, and a random generated hashed password
196 #   all users are added to the Users table in the SQLite database (locally), and data gets printed in order to confirm the insertions
```

■ **Figure 2: Continued Python code for *Users Table***

Once the code gets run, data gets populated into the data table. The number of data added can be changed, as seen on line 183. The screenshot below shows the local *Users* table populated with user information.

SQLite

SQL ▾ 1 / 1 1 - 33 of 33

Id	FirstName	LastName	Email	Role	Username	PasswordHash
1	Lucas	Payne	lucas.payne@tamu.edu	Professor	lucaspayne	hashed_fcic4vyfdp
2	Christy	Wilson	christy.wilson@tamu.edu	Professor	christywilson	hashed_0o3f9dc2ej
3	Christian	Turner	christian.turner@tamu.edu	Student	christianturner	hashed_5cybx5voka
4	Julie	Arias	julie.arias@tamu.edu	Student	juliearias	hashed_ywgybwpgrt
5	Wendy	Smith	wendy.smith@tamu.edu	Student	wendysmith	hashed_2c8ke0rlm2
6	Jennifer	Ballard	jennifer.ballard@tamu.edu	Graduate TA	jenniferballard	hashed_b57dlz0fpt
7	Joseph	Brooks	joseph.brooks@tamu.edu	Student	josephbrooks	hashed_5txer351z6
8	Marilyn	Prince	marilyn.prince@tamu.edu	Graduate TA	marilynprince	hashed_09pa6y5tfx
9	Melinda	Rivera	melinda.rivera@tamu.edu	Student	melindarivera	hashed_i4a1tzeju6
10	Jessica	Jenkins	jessica.jenkins@tamu.edu	Student	jessicajenkins	hashed_4bi3jcbfpz
11	Christopher	Scott	christopher.scott@tamu.edu	Professor	christopherscott	hashed_n5cdajdhsv
12	Rebecca	Livingston	rebecca.livingston@tamu.edu	Student	rebeccalivingston	hashed_nictrg94i1
13	Kenneth	Clark	kenneth.clark@tamu.edu	Student	kennethclark	hashed_6h9ix6c0j7
14	Andrew	Haynes	andrew.haynes@tamu.edu	Professor	andrewhaynes	hashed_ku8eksmdkq
15	Nathan	Alexander	nathan.alexander@tamu.edu	Student	nathanalexander	hashed_xs83le637e
16	Ashley	Clark	ashley.clark@tamu.edu	Graduate TA	ashleyclark	hashed_cif62i299s
17	George	Williams	george.williams@tamu.edu	Graduate TA	georgewilliams	hashed_rd1be9te1a
18	Elizabeth	Clark	elizabeth.clark@tamu.edu	Admin	elizabethclark	hashed_x7kcyxpeua
19	Jerry	Walters	jerry.walters@tamu.edu	Student	jerrywalters	hashed_flnpn1v1yj
20	Jeffrey	Taylor	jeffrey.taylor@tamu.edu	Student	jeffreytaylor	hashed_gsy064r2b6
21	Michelle	Vargas	michelle.vargas@tamu.edu	Graduate TA	michellevargas	hashed_glab6386lm

■ **Figure 3:** ***Users Table***

Like previously mentioned, a similar approach was done for the creation of the other data tables. The screenshots below show the data tables that got created with the Python code within SQLite. It is important to keep in mind that we can keep on populating the data tables with as much information as we'd like, as long as the code gets run. Same thing applies with deleting data or pulling certain information, one simply has to run SQL query commands.

SQLite

SQL ▾ 1 / 1 1 - 5 of 5

CategoryID	CategoryName
1	Basic Electronic Components
2	Integrated Circuits
3	Power Components
4	Measurement and Testing Equipment
5	Prototyping and Development Tools

■ **Figure 4:** ***Category Table***

SQL ▾ 1 / 1 1 - 4 of 4

SupplierID	SupplierName	PhoneNumber	Email
1	Digikey Electronics	1-800-344-4539	support@digikey.com
2	Texas Instruments	1-800-336-5236	support@ti.com
3	Mouser Electronics	1-800-346-6873	info@mouser.com
4	Farnell (Newark)	1-800-463-9275	support@newark.com

■ **Figure 5:** *Supplier Table*

SQL ▾ 1 / 2 1 - 50 of 54

Id	Name	Description	CategoryID	Quantity	Unit	Location	Supplier
1	Resistors (1kΩ)	A 1kΩ for basic electronic components	Basic Electronic Components	114	kits	Bin C	Digikey Electronics
2	Resistors (10kΩ)	A 10kΩ for basic electronic components	Basic Electronic Components	350	pieces	Bin A	Mouser Electronics
3	Resistors (100kΩ)	A 100kΩ for basic electronic components	Basic Electronic Components	296	kits	Bin A	Digikey Electronics
4	Capacitors (1µF)	A 1µF for basic electronic components	Basic Electronic Components	239	kits	Bin A	Texas Instruments
5	Capacitors (10µF)	A 10µF for basic electronic components	Basic Electronic Components	112	units	Bin B	Texas Instruments
6	Capacitors (100µF)	A 100µF for basic electronic components	Basic Electronic Components	457	kits	Bin A	Texas Instruments
7	Inductors (10µH)	A 10µH for basic electronic components	Basic Electronic Components	32	kits	Bin A	Farnell (Newark)
8	Inductors (100µH)	A 100µH for basic electronic components	Basic Electronic Components	299	pieces	Checkout Area	Digikey Electronics
9	Inductors (1mH)	A 1mH for basic electronic components	Basic Electronic Components	404	kits	Bin A	Digikey Electronics
10	Diodes (Standard)	A Standard for basic electronic components	Basic Electronic Components	230	units	Checkout Area	Texas Instruments
11	Diodes (Zener)	A Zener for basic electronic components	Basic Electronic Components	134	kits	Bin A	Digikey Electronics
12	Diodes (LEDs)	A LEDs for basic electronic components	Basic Electronic Components	61	pieces	Bin B	Texas Instruments
13	Diodes (Schottky)	A Schottky for basic electronic components	Basic Electronic Components	419	units	Bin B	Mouser Electronics
14	Transistors (BJTs)	A BJTs for basic electronic components	Basic Electronic Components	289	units	Bin C	Texas Instruments
15	Transistors (MOSFETs)	A MOSFETs for basic electronic components	Basic Electronic Components	132	pieces	Checkout Area	Digikey Electronics
16	Transistors (JFETs)	A JFETs for basic electronic components	Basic Electronic Components	164	units	Checkout Area	Mouser Electronics
17	Wires (22AWG)	A 22AWG for basic electronic components	Basic Electronic Components	265	pieces	Bin A	Texas Instruments
18	Wires (24AWG)	A 24AWG for basic electronic components	Basic Electronic Components	276	kits	Checkout Area	Digikey Electronics
19	Wires (26AWG)	A 26AWG for basic electronic components	Basic Electronic Components	195	kits	Checkout Area	Texas Instruments
20	Op-Amps (LM741)	A LM741 for integrated circuits	Integrated Circuits	499	kits	Bin A	Digikey Electronics
21	Op-Amps (TL081)	A TL081 for integrated circuits	Integrated Circuits	29	units	Bin B	Farnell (Newark)

■ **Figure 6:** *Items Table*

26	Logic Gates (XOR)	A XOR for integrated circuits	Integrated Circuits	212	kits	Bin B	Texas Instruments
27	Logic Gates (NAND)	A NAND for integrated circuits	Integrated Circuits	76	pieces	Bin B	Mouser Electronics
28	Logic Gates (NOR)	A NOR for integrated circuits	Integrated Circuits	261	units	Checkout Area	Digikey Electronics
29	DACs (8-bit DAC)	A 8-bit DAC for integrated circuits	Integrated Circuits	496	pieces	Bin C	Mouser Electronics
30	DACs (12-bit DAC)	A 12-bit DAC for integrated circuits	Integrated Circuits	322	units	Bin A	Mouser Electronics
31	ADCs (8-bit ADC)	A 8-bit ADC for integrated circuits	Integrated Circuits	407	kits	Bin C	Texas Instruments
32	ADCs (12-bit ADC)	A 12-bit ADC for integrated circuits	Integrated Circuits	372	kits	Bin B	Mouser Electronics
33	Batteries (9V)	A 9V for power components	Power Components	324	units	Bin B	Digikey Electronics
34	Batteries (AA)	A AA for power components	Power Components	33	units	Bin A	Farnell (Newark)
35	Batteries (AAA)	A AAA for power components	Power Components	439	units	Bin C	Mouser Electronics
36	Power Supplies (DC Power Supply)	A DC Power Supply for power components	Power Components	357	pieces	Bin C	Texas Instruments
37	Power Supplies (AC-DC Converter)	A AC-DC Converter for power components	Power Components	367	kits	Checkout Area	Texas Instruments
38	Voltage Regulators (Buck Converter)	A Buck Converter for power components	Power Components	403	pieces	Bin A	Texas Instruments
39	Voltage Regulators (Boost Converter)	A Boost Converter for power components	Power Components	428	pieces	Bin A	Digikey Electronics
40	Transformers (Step-Up)	A Step-Up for power components	Power Components	265	kits	Bin C	Texas Instruments
41	Transformers (Step-Down)	A Step-Down for power components	Power Components	499	kits	Bin C	Mouser Electronics
42	Multimeters (Digital Multimeter)	A Digital Multimeter for measurement and testing equipment	Measurement and Testing Equipment	37	kits	Checkout Area	Texas Instruments
43	Multimeters (Analog Multimeter)	A Analog Multimeter for measurement and testing equipment	Measurement and Testing Equipment	319	kits	Checkout Area	Texas Instruments
44	Oscilloscopes (2-Channel)	A 2-Channel for measurement and testing equipment	Measurement and Testing Equipment	434	pieces	Checkout Area	Mouser Electronics
45	Oscilloscopes (4-Channel)	A 4-Channel for measurement and testing equipment	Measurement and Testing Equipment	135	kits	Bin B	Digikey Electronics
46	Function Generators (10MHz)	A 10MHz for measurement and testing equipment	Measurement and Testing Equipment	117	pieces	Bin C	Farnell (Newark)
47	Function Generators (50MHz)	A 50MHz for measurement and testing equipment	Measurement and Testing Equipment	433	units	Bin A	Digikey Electronics
48	Breadboards (Large)	A Large for prototyping and development tools	Prototyping and Development Tools	92	units	Checkout Area	Mouser Electronics
49	Breadboards (Medium)	A Medium for prototyping and development tools	Prototyping and Development Tools	412	units	Bin C	Digikey Electronics
50	Breadboards (Small)	A Small for prototyping and development tools	Prototyping and Development Tools	29	kits	Bin A	Mouser Electronics

■ **Figure 7:** ***Items Table Continued***

SQL ▾

OrderID	UserID	ItemID	QuantityOrdered	OrderDate
1	8	18	3	2024-04-09
2	20	29	1	2024-04-06
3	13	3	7	2024-08-13
4	5	48	2	2024-03-08
5	1	36	6	2024-08-01
6	18	31	7	2024-09-02
7	6	27	5	2024-07-30
8	16	38	5	2024-10-29
9	9	51	8	2024-02-06
10	7	26	1	2024-01-12
11	13	49	8	2024-05-26
12	8	37	7	2024-06-01
13	16	3	4	2024-08-15
14	16	12	5	2024-08-20
15	16	19	3	2024-09-20
16	10	22	2	2024-07-11
17	2	2	3	2024-10-28
18	13	11	9	2024-03-27
19	14	10	2	2024-01-04
20	3	30	8	2024-01-07
21	6	53	1	2024-03-14

■ **Figure 8:** ***Orders Table***

Overall, this approach helped make sure that the structure for the tables was properly defined, with very important constraints such as unique fields, ID’s, and primary keys. Using Python helped provide a seamless workflow for both debugging and testing. It helped keep the goal in check: accuracy and consistency.

3. Implementation

The database subsystem was initially created and implemented locally using SQLite. However, a hosted server was eventually needed in order for this database to work and be integrated within the other subsystems for ECEN 404. Transitioning to a hosted server environment meant I would be working with MySQL, which is a bit different from SQLite.

a. 3.1. Transitioning to a Hosted Server

Hosting this database on a cloud server would better support the integration within the mobile application and website. MySQL was chosen due to its compatibility with relational schemas. In order to host the MySQL database, an Amazon Web Services (AWS) Relational Database Service (RDS) was configured. This AWS instance was the best option out of all other instances out there due to the fact that it is free for the first twelve months. It was also highly recommended to me by an ECEN 403 TA. This process involved setting up the instance, making sure that the AWS RDS version and MySQL Workbench were both the exact same version, installing appropriate security and database parameters, and establishing a secure working connection between the local SQLite environment and the MySQL cloud server.

b. 3.2. Data Migration with SQLAlchemy

Considering the fact that I had been working with SQLite, and in order to migrate all the data into the MySQL database server, it meant that I would have to switch up the Python code due to slight differences in syntax. Rather than having to go back and retype the code, I worked with SQLAlchemy. SQLALchemy is an Object Relational Mapper (ORM) that allows us to translate python classes and python objects to database tables and database entries. SQLAlchemy made sure that data tables and relationships were accurately recreated in MySQL. This migration process involved using the *create_engine* function in order to help define any necessary connections between SQLite and MySQL databases. I wrote another Python script, *main.py*, that helped transfer this data. Important information from my Amazon Web Services RDS account was needed, such as the endpoint, port, and password. This connection helped enable real-time interaction with the local database and the hosted database. Any changes done locally would be seen in the cloud. The screenshot below shows portions of the *main.py* code.

```python
from sqlalchemy import create_engine, Column, Integer, String, ForeignKey, Date #tools needed in order to help define/manage all the different data
from sqlalchemy.orm import sessionmaker, declarative_base #tools to help handle the database connection from local code to the AWS MySQL workbench
import getpass  #safely ask for AWS hosted server password

#In SQLAlchemy, Base helps serve as a blueprint. it tells SQLAlchemy which classes represent database tables. when creating a table, like users or
Base = declarative_base() #creates the Base class needed for the foundation of all database table definitions. it helps define and organize them, c

#User model: defining a python class named User. it inherits from the Base class previously created, telling SQLAlchemy that this class represents
class User(Base): #Base knows that User is a table and organizes it with others(can use Base later to create all tables or overall help manage the
    __tablename__ = 'users' #setting the name of the database table as users. this is how it'll appear in the database and once hosted in the serve

    #column helps specify that this is a database column, which will end up being filled w information/data
    id = Column("Id", Integer, primary_key=True, autoincrement=True) #unique ID for each user. id: defining a column named id in the users table. "
    first_name = Column("FirstName", String(50), nullable=False) #user first name. first_name: column for storing user first name. "FirstName": col
    last_name = Column("LastName", String(50), nullable=False) #user last name. last_name: column for storing user last name. "LastName": column na
    email = Column("Email", String(100), unique=True, nullable=False) #user email address, which must be unique. email: column for storing user ema
    role = Column("Role", String(50), nullable=False) #the role of the user, such as student, TA, or professor. will later on help classify users i
    username = Column("Username", String(50), unique=True, nullable=False) #user username for login. must be unique(no repeats) and column cannot b
    password_hash = Column("PasswordHash", String(100), nullable=False) #the user encrypted password (for security purposes)

#Category Model: defining a python class named Category. it inherits from the Base class previously created, telling SQLAlchemy that this class rep
class Category(Base): #Base knows that Category is a table and organizes it with others (helps manage the database)
    __tablename__ = 'category' #setting the name of the database table as category. this is how it'll appear in the database and once hosted in the

    id = Column("CategoryID", Integer, primary_key=True, autoincrement=True) #unique ID for each category. id: defining a column named id in the ca
    name = Column("CategoryName", String(100), nullable=False) #creating column for name of each category. String(50): string w max legnth of 50 ch

#Item Model: defining a python class named Item. it inherits from the Base class previously created, telling SQLAlchemy that this class represents
class Item(Base): #Base knows that Item is a table and organizes it with others (helps manage the database)
    __tablename__ = 'items' #setting the name of the database table as items. this is how it'll appear in the database and once hosted in the serve

    id = Column("Id", Integer, primary_key=True, autoincrement=True) #unique ID for each item. id: defining a column named id in the items table. "
    name = Column("Name", String(100), nullable=False) #creating column to store name of the item, string w max 100 characters, nullable=false: col
    description = Column("Description", String(255), nullable=False) #creating column for item description, string w max 255 characters, nullable=f
    category_id = Column("CategoryID", String(100), nullable=False) #creating column for store the category in which the item falls under, string w
    quantity = Column("Quantity", Integer, nullable=False) #column which stores the quanitity of each item, must be an integer, nullable=false: col
```

- **Figure 9:** ***SQLAlchemy Migration Code***

```python
mysql_password = getpass.getpass("Enter your MySQL password: ") #Get MySQL password securely from the user (for security purposes)

# Setup database connections
sqlite_engine = create_engine('sqlite:///inventoryTracker2.db') #create_engine(): helps create a connection to the SQLite database, and it points to
mysql_engine = create_engine(
    f'mysql+mysqlconnector://lizzetttapia:{mysql_password}@database-1.chwuw2260pwi.us-east-1.rds.amazonaws.com:3306/Inventory_Tracker' #connects to
)

#Create sessions, for both databases, for interacting w the SQLite and MySQL databases. Purpose of the session to create a connection to the databas
SQLiteSession = sessionmaker(bind=sqlite_engine) #telling SQLAlchemy that this session is sonnected to the SQLite database(can now produce sessions
sqlite_session = SQLiteSession() #the actual session object, which enables doing all the operations of adding, updating, or querying data in the SQL

MySQLSession = sessionmaker(bind=mysql_engine) #creating a session for the MySQL database, linking it to the mysql_engine
mysql_session = MySQLSession() #creating a new session for interaction w the MySQL database, which allows adding, querying, or updating data in the

#Ensure tables exist in both databases
Base.metadata.create_all(sqlite_engine) #creates all tables in the database if they dont already exists. specifies to create the tables in SQLite
Base.metadata.create_all(mysql_engine) #creates all tables in the database if they dont already exists. specifies to create the tables in MySQL
```

- **Figure 10:** ***SQLAlchemy Migration Code Continued***

4. Database Validation

Validation was an important step in making sure that the database subsystem met the requirements outlined. The validation process involved testing the functionality and security of the database both locally and on the cloud. This involved running different queries in order to test different data operations, verifying connections between data tables, and ensuring password protection.

a. 4.1. Query Validation

Query validation was done by retrieving specific data from the tables, such as counting the total orders placed by each user, finding items ordered along with the quantity and date, and pulling items that have less than a certain quantity. The screenshot below shows a few of the queries done on the cloud, validating functional testing. Many more were done and they successfully worked, such as inserting data and deleting data.

```
1   /*running queries using two or more database tables*/
2
3   /*for item and supplier: counting total quantity of items under each supplier*/
4   SELECT Supplier, SUM(Quantity) AS TotalQuantity
5   FROM items
6   GROUP BY Supplier;
7
8   /*item and supplier: pulling items that belong to a certain supplier */
9   SELECT items.Name AS ItemName, suppliers.SupplierName
10  FROM items
11  JOIN suppliers ON items.Supplier = suppliers.SupplierName
12  WHERE suppliers.SupplierName = 'Texas Instruments'; /*can change su
13
14  /*listing all items from a specific category: can change the category*/
15  SELECT * FROM items WHERE categoryID = 'Basic Electronic Components';
16
17  /*count items per category*/
18  SELECT categoryID, COUNT(*) AS TotalItems
19  FROM items
20  GROUP BY CategoryID;
```

■ **Figure 11:** ***Query Validation***

```sql
/*to select all the items*/
SELECT * FROM Inventory_Tracker.items;

/*selecting certain items where quantity is a certain number*/
SELECT * FROM items
WHERE Quantity < 200; /*can change the quantity*/

/*pulling a certain amount of items and counting their quantities and putting in descending order*/
SELECT Name, Quantity
FROM items
ORDER BY Quantity DESC
LIMIT 7;

/*updating the quantity of a certain item*/
UPDATE items
SET Quantity = 300 /*can change the quantity*/
WHERE Id = 1; /*can change the Id num*/
```

- **Figure 12:** ***Query Validation Continued***

b. 4.2. Connection Validation

Establishing a proper connection between SQLite and MySQL was also very crucial. In order for any changes made locally to also be seen on the cloud, *main.py* needs to properly run. In order for it to properly run, the password to the cloud server must be typed in the terminal. If properly inputted, all changes will be made and be visible. If not, then the code simply will not run. This overall helps establish a proper secure connection.

```
PS C:\Users\lizta\OneDrive\Documents\ecen403> & C:/Users/lizta/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/lizta/OneDrive/Documents/ecen403/main.py
Enter your MySQL password:
Data transferred successfully from SQLite to MySQL.
```

- **Figure 13:** ***Establishing Proper Connection***

5. Conclusion

The database subsystem helps serve as the backbone for this research lab inventory tracker. It is able to provide a proper foundation for all the necessary data. Through the design and implementation of a relational database model, this database maintains data consistency and integrity. Transitioning from a local SQLite environment to a proper AWS cloud environment helps enable proper integration with the other subsystems that will eventually be done. Various validation techniques confirmed the functionality, reliability, and security of the database.

6. References

[1]“Basic Use — SQLAlchemy 1.3 Documentation,” *docs.sqlalchemy.org*. https://docs.sqlalchemy.org/en/13/orm/extensions/declarative/basic_use.html

[2]Prosenjeet Shil, “SQLAlchemy Basics (2/2): The Declarative Mapping Way,” *Medium*, May 30, 2024. https://medium.com/@prosenjeetshil/sqlalchemy-basics-2-2-the-declarative-mapping-way-c9729c102f91 (accessed Dec. 06, 2024).

[3]Tutorialspoint, “SQLite - Python - Tutorialspoint,” *www.tutorialspoint.com*. https://www.tutorialspoint.com/sqlite/sqlite_python.htm

[4]D. to, “Migrate SQLite Database to MySQL with Python,” *YouTube*, Sep. 17, 2023. https://youtu.be/Qv02PxiK938?si=UMBbcPYoTVCGOo-I (accessed Dec. 06, 2024).

Research Labs Inventory

Evan Ross

MOBILE APPLICATION SUBSYSTEM FINAL REPORT

REVISION – Final
5 December 2024

SUBSYSTEM FINAL REPORT
FOR
Research Lab Inventory

PREPARED BY:

_____
Author Date

APPROVED BY:

_____
Project Leader Date

_____
John Lusher, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	12/05/2024	Evan Ross		Final Release

Table of Contents

List of Tables

No table of figures entries found.

List of Figures

1.Introduction

The Mobile Application subsystem can be run on an Android phone and is designed to be the mobile version of the LabRat website/application. The goal of this application as a whole is to modernize the management of research laboratories. The mobile application gives the users the ability to check in and check out items on the go. This allows users to make changes to the inventory easily and intuitively right inside the lab. Though this has yet to be integrated, the app will also have a machine learning aspect which will utilize the built in smartphone camera. From the camera users will be able to scan a specific item to check it in or out. Currently all data including users, items, and the active checkout list is stored in a local SQFlite database. This eventually will be integrated into an externally hosted database which will be connected to the web application as well. The entire mobile application is built with the user in mind and is meant to provide an intuitive and simple solution to research lab inventory management.

2. Tools and Applications Used

2.1 Android and Android Studio

The Mobile application is specifically designed to function on Android phones specifically with the operating system API 16 (Android 4.1) or above. The reason Android was chosen as the operating system is due to its wide variety of development softwares available, as well as its ability to easily publish and access a new app. Android studio was chosen because it is essentially the standard for developing Android applications, as it was made by Android. It also has a wide range of useful plugins such as Flutter.

2.2 Flutter

For development of the application, the plugin Flutter was used along with Android Studio. Flutter is a User Interface toolkit or framework that allows developers to develop an application from a single codebase. It uses the language Dart which makes mobile application development intuitive for people who are not as familiar with app development. Flutter was chosen because of its intuitive language, large community,

and its “Hot reload” feature which allows users to make changes to the app and see the result in real time.

2.3 SQFlite

SQFlite is a plugin for flutter which allows the creation and utilization of a local SQLite database. The database is contained locally in the simulation or on the smartphone. This is strictly for development purposes, next semester we plan to link the externally hosted database to the mobile and web application.

3. Design and UI

3.1 Login Page (Figure 1)

This is the first screen that users see when they open the mobile app. It is meant to greet the users with a visually appealing interface. They will be given the option to enter their email and password to sign in if they already have an account. The password will be obscured for security. If they do not have an account yet there will be a button which brings users to the registration page. There is also a button for the FAQ page located near the bottom of the screen.

3.2 Register Page (Figure 1)

This is the page where users can register if they do not already have a username and password. All they do is enter their email address in the email text box, and create a password by entering it in the password box. The entered email address must be valid, for example something@something.com. This ensures that a user has an email address that can be used to receive emails such as forgotten password emails, which we intend to implement in the future. After they enter their email address and password they can press the register button which will register their account. Alternatively there is a return to login button. Both buttons will send the user back to the login page.

3.3 Home/Search Page (Figure 1)

This is meant to be the main page for the application. It is the page in which users can search for items that they want to check out or check in. They have the option to search for items by name, which will send them to a list of all items with the name that they have searched. They also have the option to “view all items” which will allow users to view all of the items within the laboratory. There is also a camera button that opens the

built-in phone camera and takes a picture. At the moment this picture is not used in any way, though next semester it will be passed into the machine learning model to have whatever part is scanned identified. When a part is identified by the model it will automatically send the user to the checkout or check in page for that item. There is also a button for the FAQ page located at the bottom as well as a “sign out’ button which will return the user to the login page.

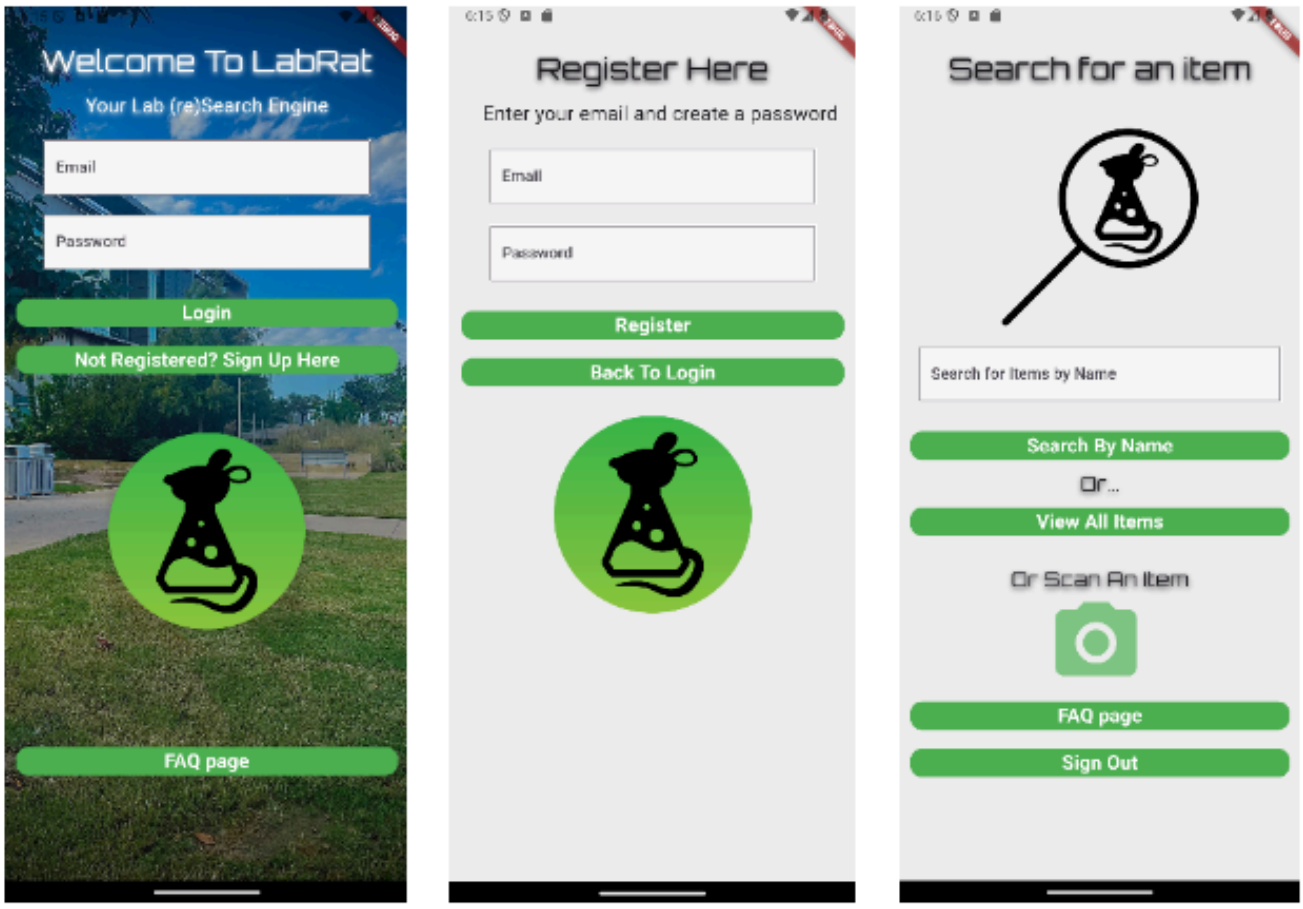


Figure 1. Login Page, Register Page, and Home/Search Page

3.4 View Items Page (Figure 2)

This is the page that is the list of the items in the lab. Depending on how the user accesses this page, whether it is via the search bar, “view all items” button, the page will display a different set of items. As mentioned before the view all items button will display all items in the lab while the using the search bar will filter to show only the searched items.

This page is in a table format with columns for “item”, “quantity”, “Availability”, as well as a column for an image of the Item. The image column will display a pre-loaded image of the item based on the name. If there is no image available for a specific item a “no image found” icon will be displayed instead. The name of the item in the “item” column is selectable and will send the user to the respective check out/In page for that specific Item.

3.5 Check In/Out page (Figure 2)

This page is where users can check in or check out the item that they have selected. Displayed at the top of the page is the item name, item quantity, and whether the item is available for checkout. From there users can enter the quantity that they would like to check out or check in. They then can press the respective button to either check out or check in the item. This page also has a button that can return the users to the home/search page.

It is worth mentioning that users are not able to check out more quantity than is available. Users are also not able to check in items that they have not checked out, or more than they have checked out. Users are also not able to return items that do not require return. In order to check out an item, users must first check the acknowledgement box. If the user fails to do any of these things they will be alerted by the system.

3.6 FAQ page (Figure 2)

This page serves as the FAQ page where users can go to find answers to commonly asked questions. This page will continue to be expanded upon as new possible questions can come up. If a user has a question not on the FAQ page they can email one of the developers and we can assist them.

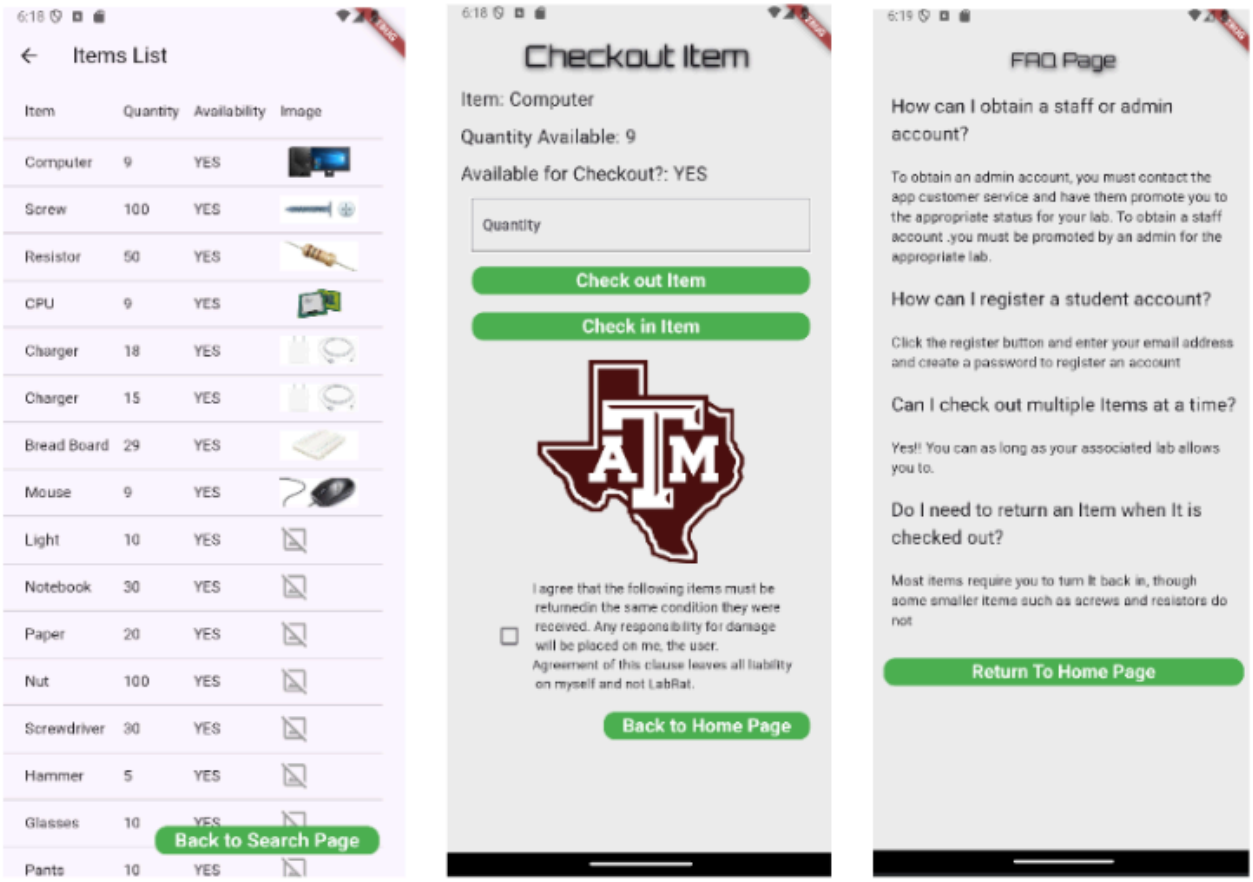


Figure 2. View Items Page, Check In/Out Page, and FAQ Page

3.7 Developer Page (Figure 3)

This page is meant strictly for development and testing purposes, and is not meant to be accessed by the everyday user. The page will not be in the final version of the app. It can be accessed by tapping the image in the middle of the home screen. The page gives the developer the option to add items to the database, view the active checkout list, or delete the database entirely. Deleting the database is useful when wanting to clear all information, or if changes are made to the database that require it to be recreated.

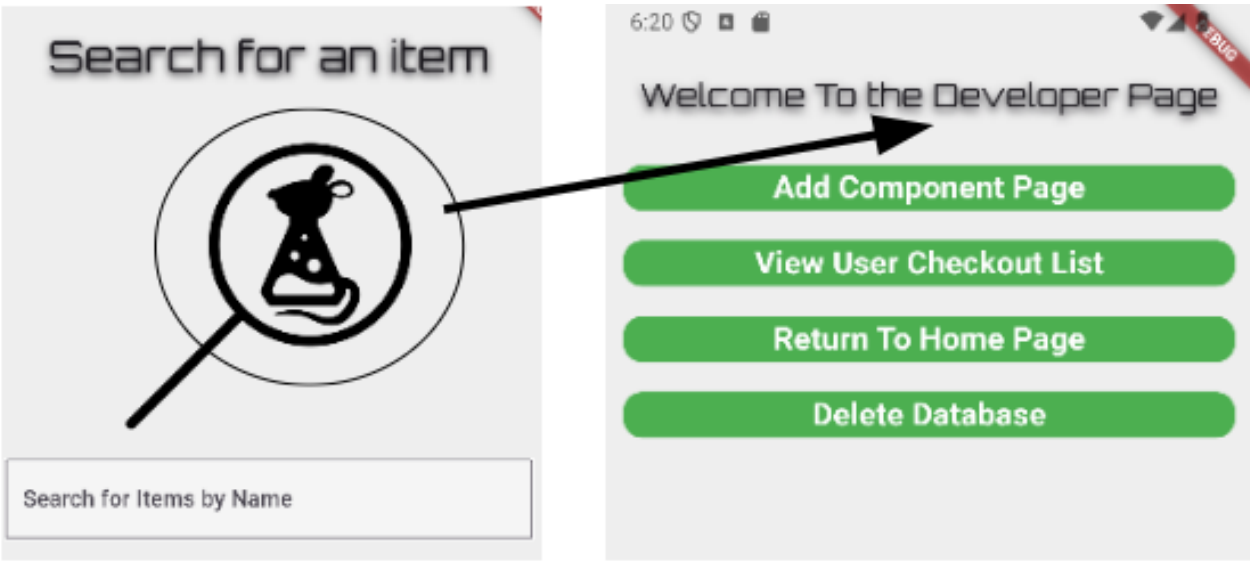


Figure 3. Developer Page

4. Implementation and Operation

4.1 Database

As mentioned before, currently the mobile application temporarily uses a local database for testing and development. Eventually this will disappear as the app is linked to the externally hosted database next semester, but it is important to have a local database to thoroughly test all of the features.

The local database is done using SQFlite plugin which assists with creating a local database in a flutter application. The database is created and maintained in the authentication_db.dart file. This file also contains any functions that pertain to the database including updating, creating new entries, as well as deleting. There is no actual page for the database.

It is difficult to view the contents of the database directly in Flutter through the command window. To view the database I have been loading the database file (which is stored in the simulated phone) into an sqlite database viewer, which gives the contents of the database as a CSV file, which I can view in excel. Viewing the contents of the database is important for validation and making sure that entries act as intended.

4.2 User interface Display

Most code in flutter either operates to display the user interface in a certain way, or change how the user interface acts when interacted with. The root of all pages in flutter is a widget which is the user interface for the page. Within that widget other widgets can be displayed such as columns, text, boxes, or buttons. Widgets are stackable so that one widget can contain others.

Pages can also take in variables which can change the way that the UI is displayed. An example of this is passing the queue to the view items page or passing the selected item to the check out/In page.

4.2 User Interface Functions

Widgets in the user interface can be given a function to perform when interacting with in a certain way. For example many of the button widgets in the app perform the function that navigates to another page when tapped. They also can set variables and text controllers as they do in the email and password boxes in the login screen. They can also be given the functionality to interact with the database through created functions.

5. Validation

5.1 User Interface Functionality

The User interface Functionality was tested by using the app as a user normally would. All of the buttons, text boxes, and the camera Icon works as expected. This was originally tested through the Android Studio simulated phone, but it now has also been tested using a physical android phone and performs as expected for both.

5.2 Database Functionality (Users)

The “Users” table in the database was tested through adding users to the database and then viewing the contents of the database. As more users are registered, these users are added to the database. The Database table is viewed through an SQlite database viewer which exports the tables to a CSV file which can be viewed on excel.

	A	B	C
1	id	email	password
2	1	e@e.com	123
3	2	evanross@gamil.com	gths
4	3	carl@yahoo.com	458
5	4	Carson@gail.com	fosfioohi
6	5	ScottHR@gmail.com	5469
7	6	evanjross@gmail.com	password
8	7	ben@gmail.com	123

Figure 4. Users Table exported from database as CSV file

5.3 Database Functionality (Items)

The “Items” table of the database was checked in multiple ways. The first way is the same as the “Users” Table where the contents of the database can be viewed directly through the database viewer. The second way is looking at the items list screen in the user interface of the application. In both of these ways, the items table was tested by adding and removing items from the database. Changing the quantities by checking in and out items was also tested. Everything worked as expected in the “Items” table.

	A	B	C	D	E
1	id2	item	quantity	availability	returnable
2	1	Computer	9	YES	1
3	2	Screw	100	YES	0
4	3	Resistor	50	YES	0
5	4	CPU	9	YES	1
6	5	Charger	18	YES	1
7	6	Charger	15	YES	1
8	7	Bread Boa	29	YES	1
9	8	Mouse	9	YES	1
10	9	Light	10	YES	1
11	10	Notebook	30	YES	0
12	11	Paper	20	YES	0
13	12	Nut	100	YES	0
14	13	Screwdrive	30	YES	1
15	14	Hammer	5	YES	1
16	15	Glasses	10	YES	0
17	16	Pants	10	YES	0
18	17	Screw	1	YES	0

Figure 5. Items Table exported from database as CSV file

5.4 Check Out/In (Checkout List)

The Check Out/In table of the database was tested in a similar way to the Items table. It can be viewed through the database viewer or it can be viewed through the app interface via the “View User Checkout List” button in the developer page. It was tested by having multiple different users check out multiple different Items to make sure that they are added to the active checkout list. I also tested the ability for users to check back in items to reduce the quantity checked out, or delete the entry all together if the quantity goes to zero. I also tested to make sure that when a user checks out a non returnable item (such as a screw), it is not added to the active checkout list. Everything in this table worked as expected.

	A	B	C	D
1	id3	userEmail	checkoutItem	checkoutQuantity
2	1	evanjross@gmail.com	Computer	5
3	2	evanjross@gmail.com	Charger	6
4	3	evanjross@gmail.com	Bread Board	1
5	4	evanjross@gmail.com	Mouse	1
6	5	ben@gmail.com	Charger	1
7	6	ben@gmail.com	Computer	1
8	7	ben@gmail.com	CPU	1

Figure 6. Active Checkout List Table exported from database as CSV file

6. Challenges

The biggest challenge that I faced throughout this project is being unfamiliar with the coding language. Luckily Android Studio and Flutter has a large community and much documentation available online.

7. Conclusion

The mobile application has been developed as the mobile version of the LabRat inventory management application, which aims to simplify and modernize the way research labs are managed. The mobile application specifically has the user in mind as it will allow users to walk around the lab and look up or scan various items from inside the lab. The user interface as well as the database functionality was tested and validated to ensure that the application is ready to be linked with the other subsystems.

8. Future Plans and Implementation

Currently the mobile application uses a local SQFlite database for testing purposes. In the future we intend to link It to the externally hosted database. This will make the mobile application able to share the tables of information with the web application on one big server.

We also plan to implement the machine learning model into the mobile app. This will be accessed via the camera icon on the home screen of the app. When the built in camera takes a picture of an item it will be passed on to the machine learning model to be identified. When it is identified, the user will be sent to the corresponding check out/in screen for that Item.

9. System Requirements

The device used to run the mobile application must use the Android operating system. As stated in the Flutter Documentation, Android API 16 (Android 4.1) or above is required to run the mobile application. Phone must have a functional camera if user wishes to use the machine learning feature.

10. References

[1] “SQLite Viewer,” inloop.github.io. https://inloop.github.io/sqlite-viewer/#

[2] Flutter, “Flutter documentation,” docs.flutter.dev, 2024. https://docs.flutter.dev/

[3] “Dart packages,” Dart packages. https://pub.dev/

Research Labs Inventory

Sumedha Bhattacharyaa

MACHINE LEARNING MODEL SUBSYSTEM FINAL REPORT

REVISION – Final
5 December 2024

Subsystem final report
for
Research Lab Inventory

Prepared by:

_____

Author Date

Approved by:

_____

Project Leader Date

_____

John Lusher, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	12/05/2024	Sumedha Bhattacharyaa		Final Release

Table of Contents

List of Tables

No table entries.

List of Figures

1. Introduction

The machine learning (ML) subsystem is designed to automate the identification and classification of lab items from images to allow mobile users to check in their items back into the lab's inventory. Using a Convolutional Neural Network (CNN), this subsystem processes input images and predicts their corresponding categories, providing a streamlined solution for inventory management. Currently, the system functions as a standalone implementation, but integration with an Android mobile app is planned for next semester. This integration will allow users to upload images directly via the app and receive real-time predictions from the ML model, enabling efficient inventory tracking and an easy way to check in items.

2. Development Environment and Tools

The development of the ML subsystem was carried out using a combination of tools and libraries. **Visual Studio Code (VS Code)** was used as the primary integrated development environment (IDE) for writing and debugging the code, offering an efficient and organized workspace. The implementation was done in **Python** because it's well-suited for machine learning tasks with its many libraries and frameworks. The core model was built using **PyTorch**, a widely used deep learning framework that simplifies the process of defining and training neural networks. Together, these tools provided a robust development environment for building this subsystem.

3. Data Preprocessing

3.1 Data Collection

The datasets used for this project were taken from **Kaggle**, which is a website with several datasets made for deep learning models. Each image was put in a folder with its respective class name and the classes consisted of resistors, screws, and PCB (Printed Circuit Board) components. Each class had around 1200 images in total. These categorized images form the foundation of the training and validation datasets. There was also a data set called 'misclassified' which contained images that intentionally did not belong to the predefined categories. For example, in the 'screws' category, the misclassified version included images

of unrelated objects, such as wires or laptops. The purpose of this dataset was to measure the model's false positive rate, which indicates how often the model incorrectly classifies non-category images as belonging to a specific class.

3.2 Data Transformation

The preprocessing pipeline ensures that all input data is standardized and suitable for the model. Images of varying dimensions were resized to 64x64 pixels, providing uniformity and reducing computational load during training. The pixel values were normalized to a range of [0, 1], which improves numerical stability. To enhance the diversity of the dataset, data augmentation techniques, including random rotations, flipping, and cropping, were applied. This augmentation helped the model generalize better and reduces the likelihood of overfitting.

3.3 Dataset Splitting

The preprocessed dataset was divided into two parts: 75% was used for training the model, 15% was used to validate the model, and 10% was used to test the model. This split allows the model to learn patterns from the training data while using the validation set to evaluate its generalization performance.

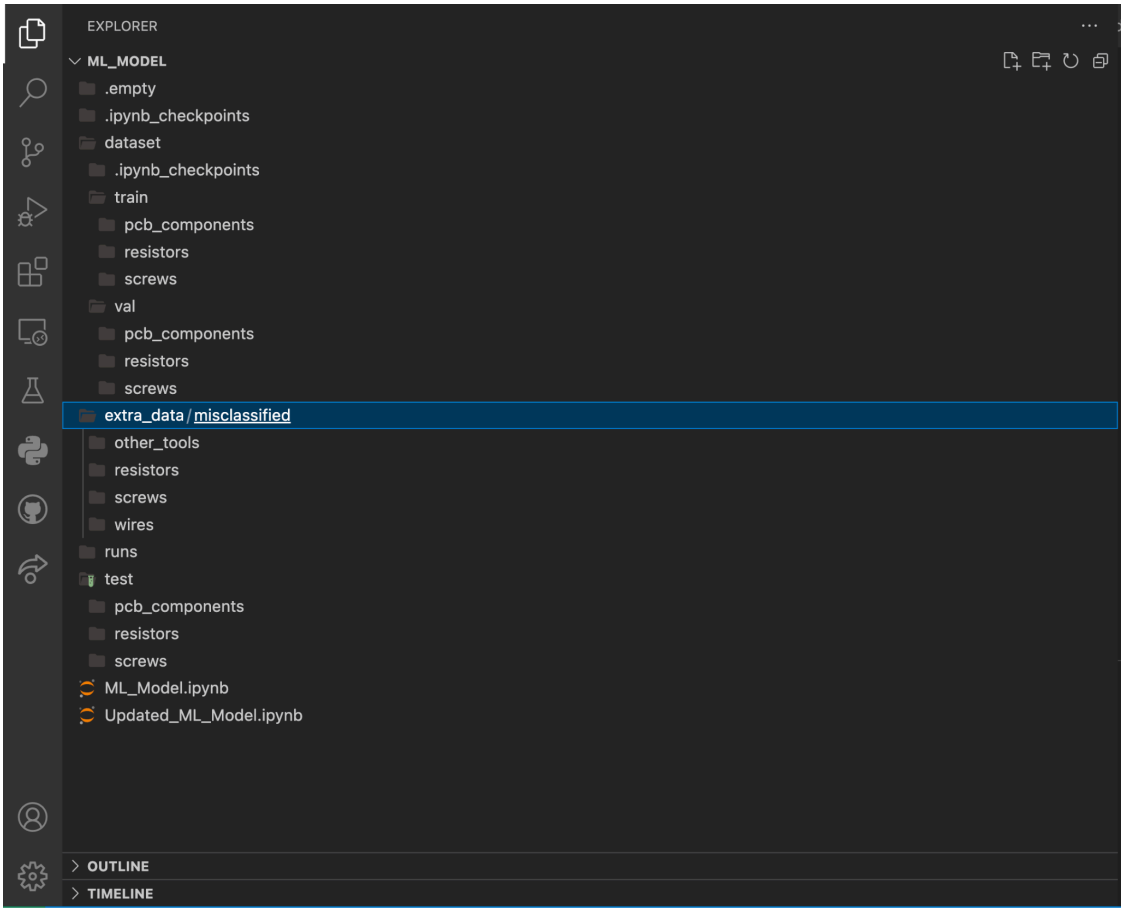


Figure 1. Dataset Organization

4. Model Architecture

4.1 Overview of the CNN Design

The CNN (Convolutional Neural Network) model used in this subsystem is composed of multiple layers designed to extract features and classify images. The first layer is the input layer, which accepts resized 64x64 RGB images. This is followed by three convolutional layers that progressively extract more complex features from the images. Each convolutional layer is followed by a Rectified Linear Unit (ReLU) activation function and a MaxPooling layer. ReLU is a mathematical function which sets all negative values to zero while keeping positive values unchanged. This activation introduces non-linearity into the model, allowing it to learn complex patterns and relationships in the data. MaxPooling layers downsample the feature maps by selecting the most prominent features, reducing spatial dimensions and complexity.

$$f(x) = max(0, x)$$

Equation 1. The Rectified Linear Unit (ReLU) Activation Function.

After the convolutional layers, the output is flattened into a one-dimensional vector to prepare it for the fully connected layers. The first fully connected layer (FC1) contains 256 neurons and applies ReLU activation to further process the features. A dropout layer follows FC1, randomly deactivating 50% of its neurons during training to prevent overfitting and improve generalization. The final fully connected layer (FC2) outputs a vector of probabilities corresponding to the number of classes, enabling the classification of the input image.

4.2 Flow of Data

The input image is first passed through Conv1 with ReLU activation and MaxPooling. This process is repeated for Conv2 and Conv3, extracting progressively higher-level features. The resulting feature maps are flattened into a vector and passed through FC1, where Dropout is applied. Finally, the output from FC2 provides the predicted class label for the image.

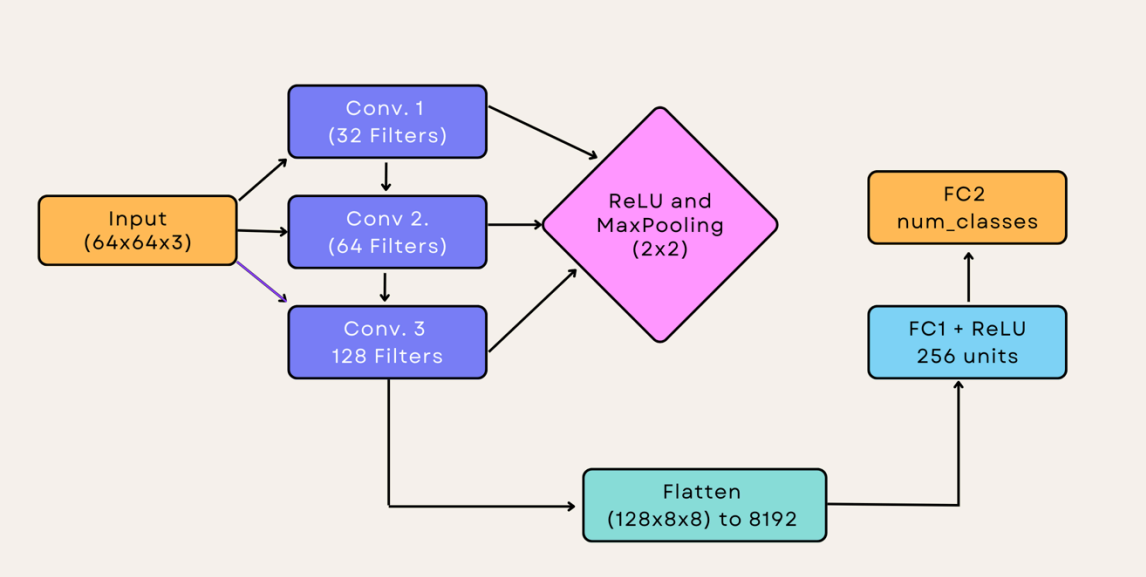


Figure 2. Comprehensive Model Architecture Diagram

5. Training Phase

5.1 Training Data

The training dataset consisted of preprocessed images categorized into their respective class folder. These images were used to teach the model to identify patterns and distinguish between different categories of lab items.

5.2 Training Pipeline

During training, the input images were passed through the network in a forward pass, where predictions were generated at the output layer. The loss function, CrossEntropyLoss, calculates the difference between predicted and true labels. This loss was then backpropagated through the network to compute gradients, which were used to update the model's weights using the Adam optimizer. This iterative process continues for several epochs (10) until the model achieved satisfactory performance on the training dataset.

5.3 Performance Metrics

The model's training performance is evaluated using loss, which quantifies the difference between predicted and true labels. This metric provides insights into how well the model is learning and generalizing to the data. A low loss number indicates that the model is learning the data well. The loss that this model had achieved after 10 epochs, was 0.0049, indicating a low loss.

6. Validation and Testing

6.1 Validation Process

The validation dataset was used to monitor the model's performance on unseen data. The validation metric that was recorded was accuracy. If the validation accuracy is high, while the training loss is also high, there is an indication of overfitting data. However, since this model achieved a standard validation accuracy of 96.35% with the aforementioned low loss number, it's safe to say this model is not overfitting.

6.2 Testing Phase

After training and validation, the final model was tested on a separate test set to evaluate its generalization ability. Additionally, the 'misclassified' dataset was processed through the model to calculate the number of false positives. The Correct Negative Classification Rate

was 91.90% and the False Positive Rate was 8.10% indicating that the model has few false positives.

```
Correct Negative Classification Rate (%): 91.90%
Incorrect Negative Classification Rate (%): 8.10%
Correct Negative Classification Rate: 692/753
```

Figure 3. False Positive (Incorrect Negative Classification) Rate

6.3 Testing Metrics

Key metrics such as **precision**, **recall**, and the **F1 score** were computed during the testing phase to provide deeper insights into the model's performance for each class. For instance, the model achieved a **precision** of **97%**, a **recall** of **96%**, and an **F1-score** of **96%**, averaged across all classes. These results indicate that the model is effective at making correct predictions and consistent in identifying instances of each class. Additionally, a confusion matrix for the testing phase was created. The x-axis variations (numbers outside the highlighted diagonal) are false positives – images that were predicted to be a class when they're not from that class. The y-axis variations indicate false negatives – images that were classified as not being from that class when they are from that class. There were 17 false positives and 0 false negatives.

Classification Report:

	precision	recall	f1-score	support
pcb_components	0.97	1.00	0.99	200
resistors	0.94	1.00	0.97	186
screws	1.00	0.79	0.88	80
accuracy			0.96	466
macro avg	0.97	0.93	0.95	466
weighted avg	0.97	0.96	0.96	466

Figure 4. Classification Report of Measured Metrics from Testing

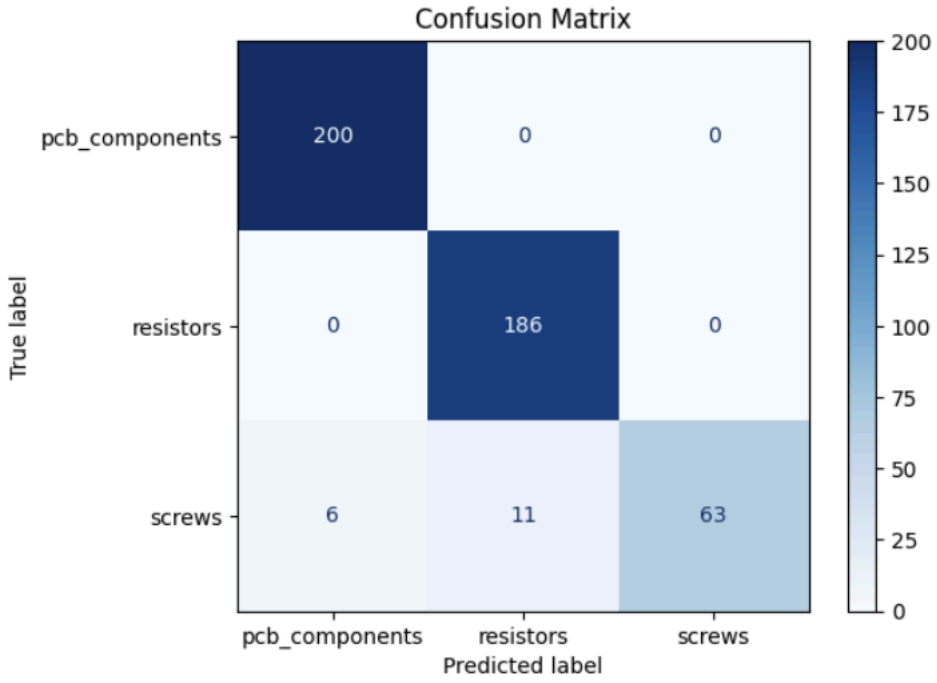


Figure 5. Confusion Matrix

7. Challenges and Solutions

7.1 Issues Encountered

The project faced challenges such as data imbalance, where certain classes (screws) had fewer examples, leading to potential bias.

7.2 Mitigation Strategies

To address these challenges, data augmentation techniques were employed to artificially increase the dataset's diversity. Dropout layers were also used during training to prevent overfitting by randomly deactivating neurons in the fully connected layers.

8. Integration Plans

8.1 Future Integration with the Android App

In the next semester, the machine learning model will be integrated into an Android mobile app. The app, developed by a teammate, will allow users to upload images of lab items. These images will be sent to a backend server where the model will process them and return predictions. The app will display the predicted class label, providing real-time feedback to users.

8.2 Deployment Considerations

To achieve seamless integration, the trained model will be saved in a server-compatible format, such as PyTorch or ONNX. A RESTful API (Representational State Transfer Application Programming Interface) will be developed to facilitate communication between the app and the server. This API will handle tasks such as receiving image uploads, preprocessing them, and passing them to the model for classification. Testing will focus on ensuring minimal latency and accurate predictions, for a smooth user experience.

9. Future Improvements

In the future, the model's performance can be enhanced by experimenting with more advanced architectures, such as ResNet, which may work more effectively. Expanding the dataset to include additional classes and larger datasets will make the system even more versatile and practical for inventory management tasks.

10. Conclusion

The machine learning subsystem is a critical component of this project, enabling efficient classification of lab items once fully integrated. While currently demonstrating intended performance, its planned integration with the Android app will significantly enhance its usability. The system holds great potential for further development, paving the way for a comprehensive and user-friendly inventory management solution.

11. References

[1] Kaggle. "Kaggle Datasets." Available: https://www.kaggle.com/datasets, Accessed: September 2024.
[2] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," PyTorch Documentation. [Online]. Available: https://pytorch.org/docs. Accessed: September, 2024.
[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.