Resource Management System

A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Yogesh R
(111801047)

COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

CERTIFICATE

*This is to certify that the work contained in the project entitled "**Resource Management System**" is a bonafide work of **Yogesh R** (**Roll No. 111801047**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my guidance and that it has not been submitted elsewhere for a degree.*

Dr. Albert Sunny

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

Acknowledgements

I would like to express my deepest gratitude to my B.Tech project mentor Dr. Albert Sunny for guiding me throughout the course of this project. I would like to acknowledge the time and effort he has put in to steer me in the right direction for this project.

I would also like to thank my friends and family who constantly supported me and provided the motivation to carry my project forward.

Contents

List of Figures

Chapter 1

Introduction

The Resource management system is an application that facilitates a generalized solution for resource management in an institute or organization. The application provides a web-based portal for administrators and a mobile app for end-users. The resources in the inventory may have an optional pipeline of booking followed by buying or renting. The wide range of applications includes booking slots in labs, borrowing lab equipment, mess management, and buying/selling of other general student commodities such as cycle, books, etc.

1.1 Motivation

Most institutes or organizations still use traditional methods such as record notebooks or excel to track and maintain their inventory. These methods are not effective and do not scale well with frequent updates.

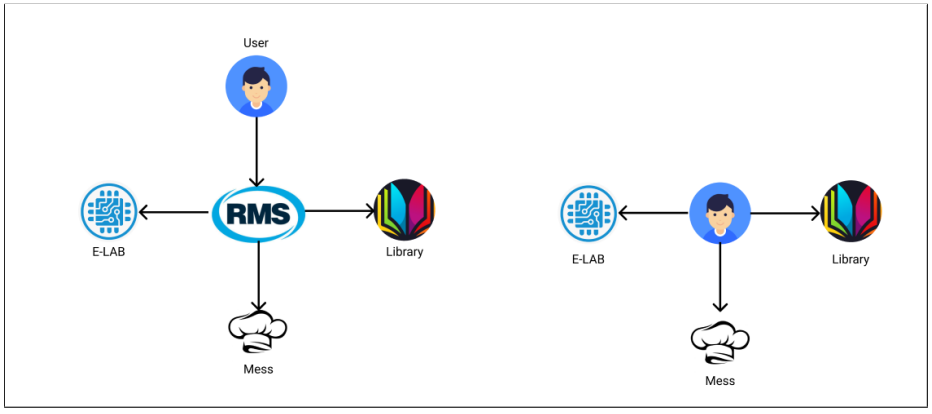

Fig. 1.1: Motivation

As shown in right half of the figure above, even if there exists an automated solution, since each facility has its portal, it makes centralized resource tracking harder and also makes it hard for the end-user to keep track of each of these portals for buying or renting items. For large institutes with many facilities, maintaining each one of them separately becomes cumbersome.

A unified, generalized solution, as shown in the left half of the image, is the need of the hour as it provides high scalability and agility.

1.2 Goals

- Our database design and architecture has ensured a generalized solution for managing various facilities in an institute or organisation.
- A web portal for administrators to manage the inventory.
- Web-scraping to suggest images while creating facilities, categories or items.
- A mobile application for end-users to interact with the inventory.
- Secure authentication and fine-grained authorization up to the level of items for higher flexibility. booking, buying and renting features
- Enhancing the mobile application with admin access which is currently only for the web.
- Users can buy/borrow a **Pack** (a group of items).
- App wide search to allow users to search for products from any part of the application.
- A simple recommendation system for items that were bought or rented together.

1.3 Organization of The Report

Chapter 2 explains the prior works in this domain. Chapter 3 explains the application's technical stack and the reason involved in choosing it. Chapter 4 shows the challenges faced during development. Chapter 5 illustrates the design patterns incorporated during development. Chapter 6 explains the database schema: the entities and their relationships. Chapter 7 explains the workflow of the complete application. Chapter 8 show the user interface of the application. Chapter 9 provides the conclusion and the timeline of the projects development.

Chapter 2

Review of Prior Works

2.1 InvenTree

InvenTree is an open-source Inventory Management System which tracks products from the supplier to the inventory. It provides low-level stock control and part tracking. The main focus of this application is inventory management in which it tracks the product from the manufacturer/supplier to the inventory by using QR codes to tag item locations. The major targets for this application are warehouses, retail stores etc. In contrast to the above approach, our solution aims to monitor the inventory and allow end-users to purchase or rent items from the inventory i.e., we track the item from the inventory to the end-user. The major targets for our application are gated communities, education institutions etc.

Chapter 3

Technical Stack

The resource management system has a client-server-based software architecture. In this architecture, the client application is a web portal used by the administrators and a mobile app for end-users. The client sends requests to the server using GraphQL API. The server validates the client and their request, queries the database, does some computation and sends the response to the client.

3.1 Angular

Angular is a modern front end web development framework built by Google. It is a component based framework for scalable web applications with well-integrated libraries that covers a wide range of features like routing, client-server communication, etc. A complementary package of UI components for Angular is PrimeNG. PrimeNG has over 90 components, with variety of themes, and templates to build a responsive web application. The state management was done using the RxJS package.

3.2 Flutter

Flutter is a modern open-source software development frameworks developed by Google to build cross platform applications in Android, iOS, desktop and web with a single code base. The elegance, and simplicity makes Flutter a prominent choice for mobile app developers. RMS heavily uses the GetX micro framework package in Flutter for state management, route navigation and API services.

3.3 APIs

The application programming interface (API) is a intermediate software that facilitates communication between the client and the server. Resource Management System uses GraphQL for server communication and REST API for third party services.
GraphQL is a data query/manipulation language developed by Facebook. The main advantage of GraphQL over REST API is avoid under-fetching and over-fetching. GraphQL provides exactly the required content to render the with exactly one API end point and one request. The Apollo GraphQL is a software platform that immensely simplifies the underlying complexity, and unifies GraphQL across multiple apps.

3.4 NestJS

NestJS is a progressive Node.js framework for developing effective, scalable and reliable server-side applications. It uses the latest JavaScript features, and design patterns. NestJS provides various wrapper libraries for popularly used packages such as apollo-graphql, JWT authentication, etc.

3.5 Database

Resource management system uses MongoDB which is a No-SQL database built for handling big data related tasks more efficiently. Some advantages of MongoDB over SQL databases are:

- It has a very flexible schema which allows us to evolve and store data easily.
- Apart from supporting all features of relational databases, it is built to scale up much faster.
- MongoDB schemas are more intuitive to developers using JS based languages and since our technical stack is completely works by handling JSON data, using MongoDB is a more appropriate choice.

3.6 Development tools

- **Docker**: It allows developers to work and deploy apps without being concerned about the various dependencies needed.
- **GitHub**: It is used for version control and collaboration.
- **Figma**: It is an vector graphics UI designing programs.
- **Postman**: It is a platform to test, build and share APIs. It simplifies API life-cycle and collaboration.

3.7 Image Suggestion

One important feature of our application is suggesting images while the admin creates facilities, categories and items and to achieve this, we use the Google Custom Search Engine (CSE). This service provides a REST API endpoint that helps us scrape the web and suggest relevant images to users.

Since this API suggests images from various domains, one difficulty we faced was the security restrictions that CORS applies. CORS ensures that the server responds to requests only from a particular set of known clients and not any other clients.

As an example, let's say a bank **ABC** serves its clients on **https://abc.com** and it stores user credentials as cookies, now if a malicious website **https://hack.com** tries to make a request to the bank server, then it shouldn't be allowed as this domain is not among the official domains or subdomains of the bank and this is exactly what CORS does. It rejects response to clients if the request does not come from one of it's white-listed domains.

To overcome this issue, we had to set up a CORS proxy server, which proxies (forwards) the request from our client to the server. This works because our server white-lists our client and our server accepts responses from the image (external) server without CORS headers unlike browsers which reject responses without CORS headers.

Although, there are many public proxies available, we chose to run our local one as the major concern among CORS proxies is security. Also, we are white-listing only our own domains, which makes accessing and abusing the proxy from other domains harder.

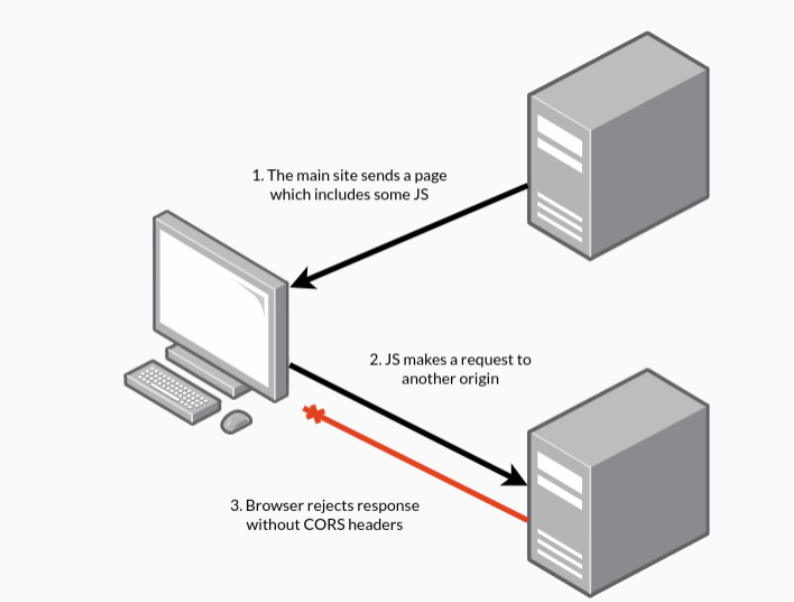

Fig. 3.1: A request failed due to CORS

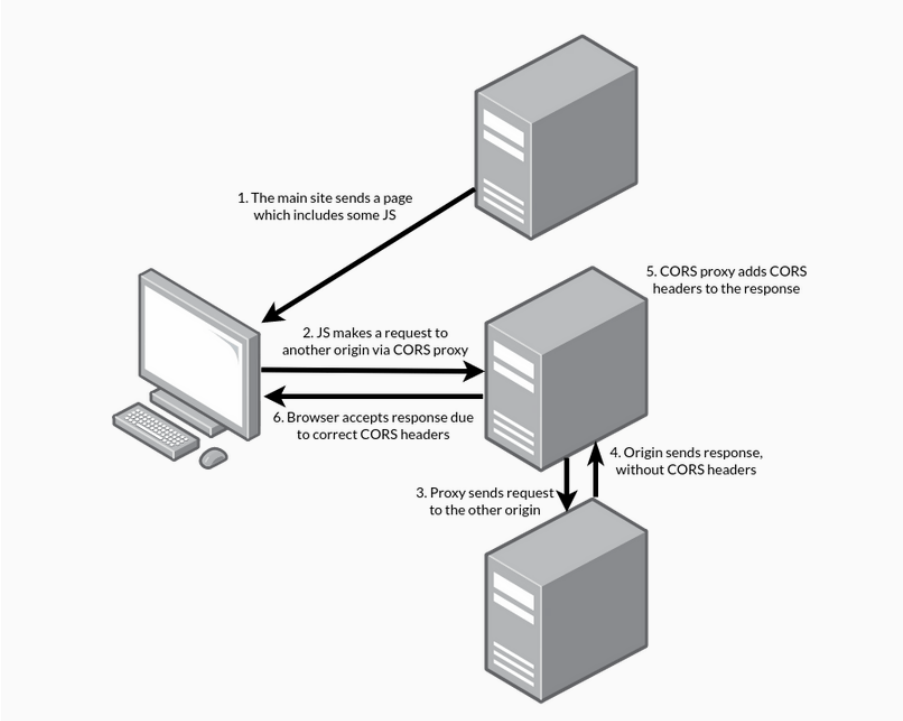

Fig. 3.2: Our proxy server, talks to the required server and sends the result

Source

Chapter 4

Design Patterns

4.1 Angular Architecture

The angular architecture is a union of multiple modules having its own components, and services. A component consists of the UI design (HTML), the UI styling (SCSS) and the logic (Typescript).

The service are responsible for providing data, and state management. These are done using the RxJS package. It provides reactive programming approach using observable to easily develop asynchronous and callback functions.

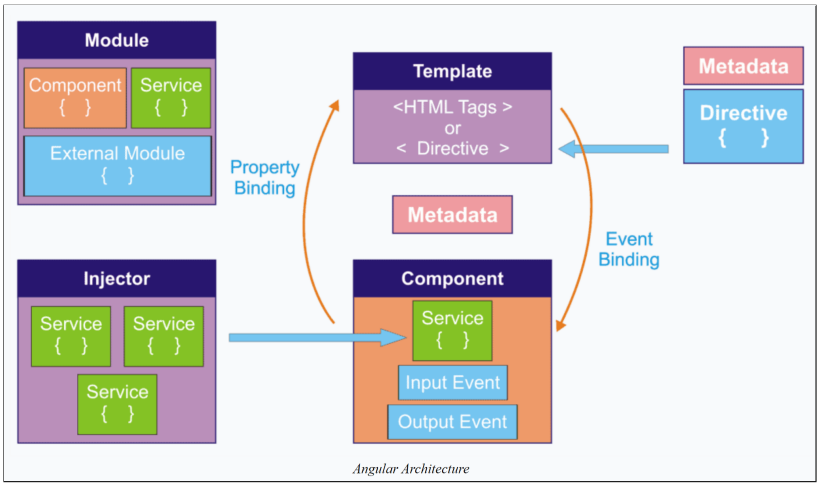

Fig. 4.1: Angular Architecture.

Source

4.2 GetX Architecture (Flutter)

The GetX architecture follows the Model-View-View Model architecture (MVVM). MVVM is one of the most popular UI architecture for mobile app development.

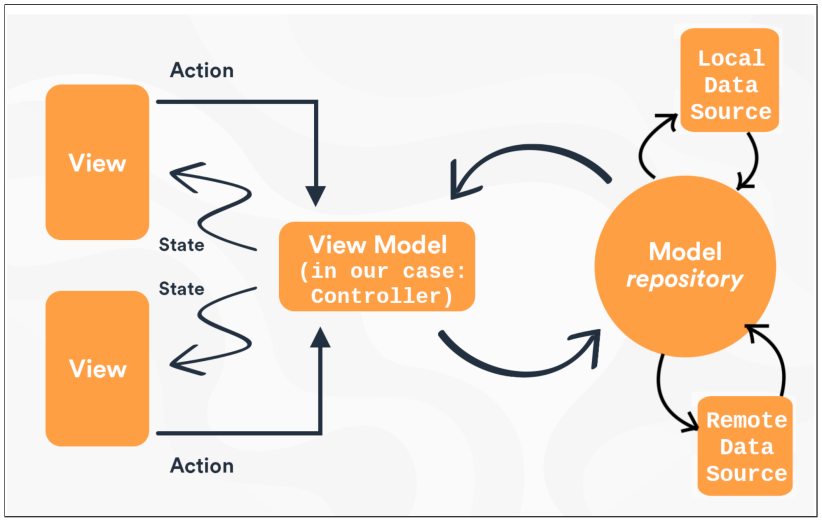

Fig. 4.2: GetX Architecture.

Source

Every module consists of views, controllers, bindings and providers.

- Views : The views holds the UI widget tree that is rendered by flutter to the user.
- Controllers : The controllers consists of the functions that drive the logic of the page.
- Bindings : The bindings will inject all the dependencies for the module.
- Providers : The providers will provide the data required for the module.

Chapter 5

Database

The database model of resource management system consists of 8 entities, and 11 relations.

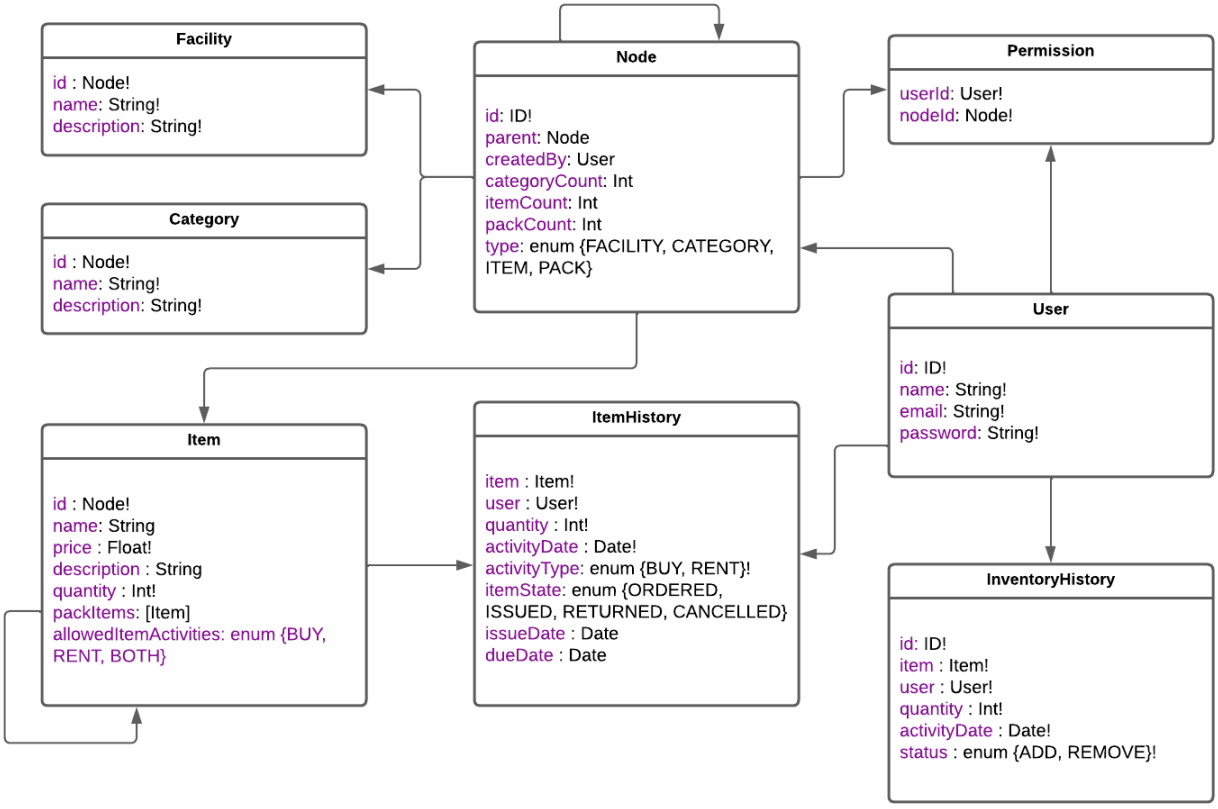

Fig. 5.1: Database diagram

Node is the central entity of the application. A nodes is either a facility, a category or an item. The application follows a forest data structure with each facility being a tree. The root of the tree is a facility with the leave nodes as the item and others category as shown in the figure below.

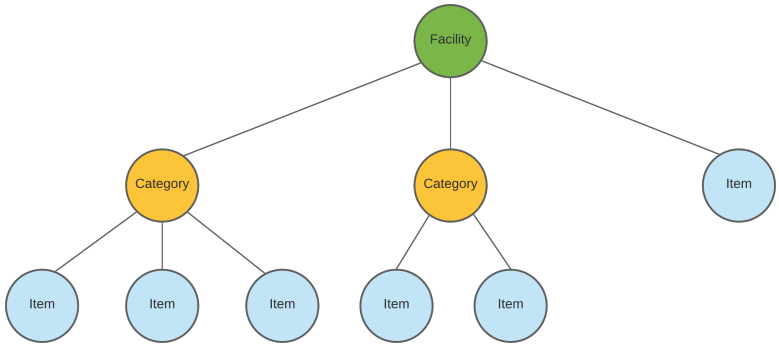

Fig. 5.2: facility tree structure

The permissions entity holds the pair user and node IDs. For details, see section 6.2. The related item and item frequency entities are used to provide the closely brought or rented items. The inventory history keeps track of the added and removed items and the item history keeps track of the brought and rented items.

Chapter 6

Workflow

6.1 Authentication

The authentication uses JSON web tokens (JWT), which is a token-based authentication.

```
mutation createUser ($createUserInput: CreateUserInput!) {
    createUser (createUserInput: $createUserInput) {
        _id
        email
        name
        password
        token
    }
}
```

Fig. 6.1: Sign-Up Mutation

If the user is a first time user, then this user must sign-up with their name, email-id and a password. After the user is successfully registered, he/she is give a token.

```
mutation login ($email: String!, $password: String!) {
    login (email: $email, password: $password)
}
```

Fig. 6.2: Sign-In mutation

If the user is an already existing user, then they must sign-in using their email-id and password, after a successful login, the server returns a token.

The token returned is stored in the local storage and for every request made by that user the token is attached to the header and validated in the server.

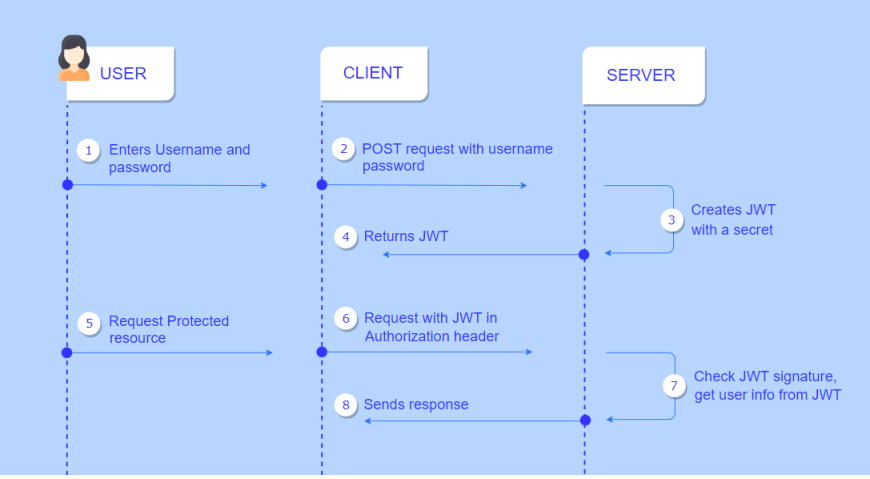

Fig. 6.3: Authentication Flow [1]

6.2 Authorization

Every mutation request goes through the authorization process. The user permission is based on the nodes. Any user with permission to the node has access to all the nodes under it.

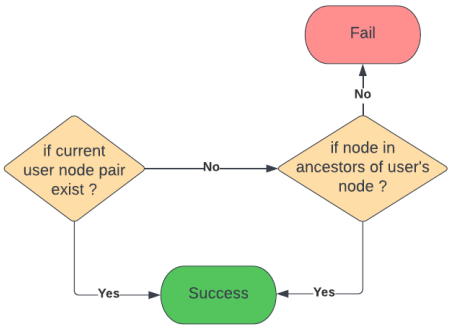

Fig. 6.4: Authorization algorithm - Time Complexity O(log(h))

6.3 Item flow

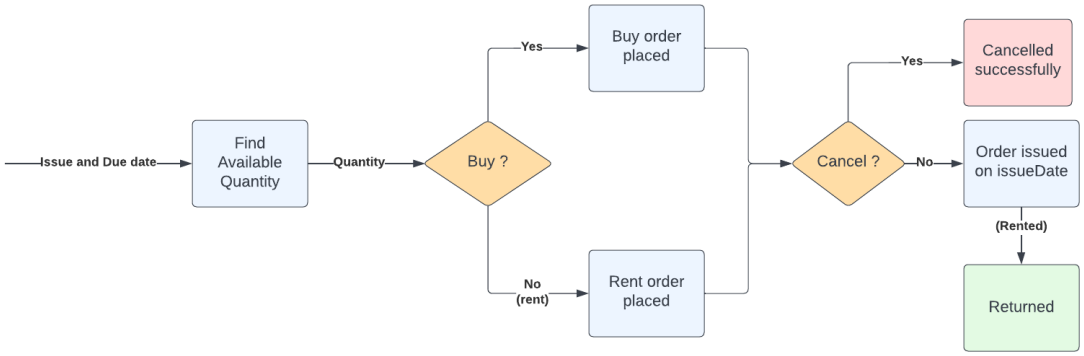

Fig. 6.5: Item buying and renting flow

An item can be brought or rented based on the item configuration. The user is given the privilege to cancel the order before the order has been issued. The rented orders are expected to be return by the stipulated deadline.

6.4 Technical Flow

The server has many GraphQL resolvers. The client can make requests to any of these resolvers. The appropriate resolver parses the request and sends it to the services. The services then executes database operations and return the response back to the resolvers which in-turn return it to the client. The client reads the data and shows appropriate content or messages when required.

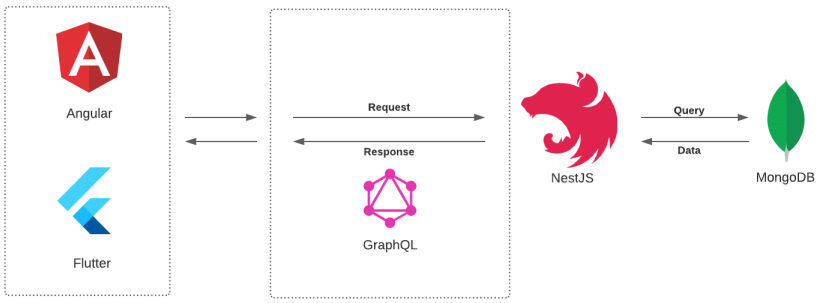

Fig. 6.6: Complete technical flow

6.5 Recommendation system

When a user visits any item or pack page, we help the user by suggesting items that were recently bought along with it. The algorithm to do this is as follows:

Input: $id : String$

1. Find all activities (buy or rent) with associated with this item.
2. For each activity,
   (a) Find all the items bought within an interval of three days from the date of issue of the item.
   (b) Append the list of items obtained above to the results array.
3. Depending on the frequency of items in the results array, we show the top 5 results to the user.

Chapter 7

User Interface

This chapter predominantly shows the mobile version of RMS along with the newly added features in the web. For the complete set of photos including all of web, and the demo kindly refer to the **complete assets** .

The application starts with the login page where the client is prompted to log-in or register into the application. The shared_preference flutter package is used to provide auto-login feature.

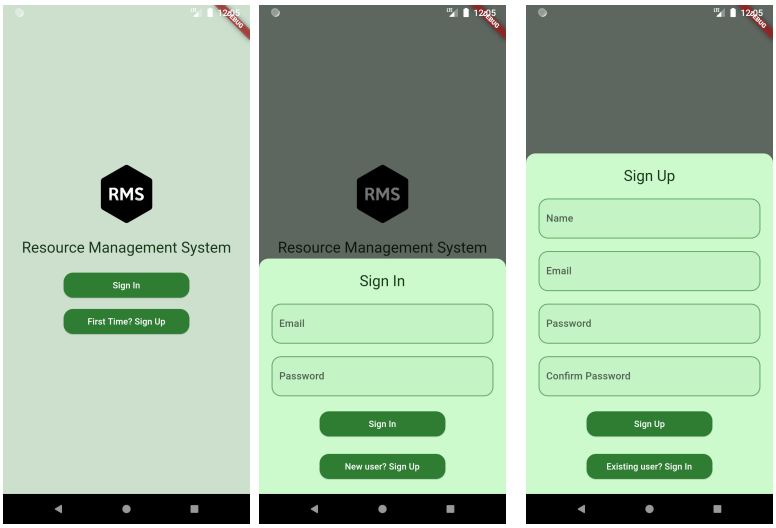

Fig. 7.1: 1. The first page, 2. Sign-in dialog, 3. New user registration (from left).

Once a user has logged in, they will see a grid of all the facilities available in the institute. The super admins or the admins of any category can be seen in the users tab.

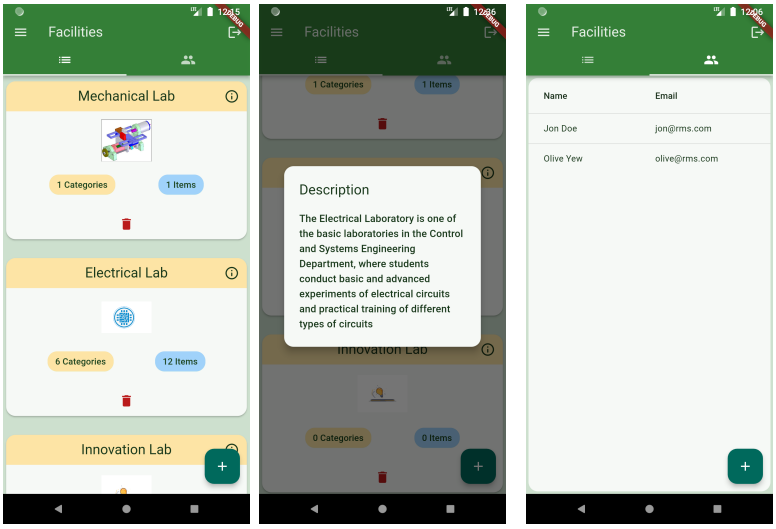

Fig. 7.2: 1. Facilities, 2. Facility Description, 3. Admins panel

When the user clicks on a facility, they will see all the categories or items in that facility. The categories have green headers, and items have blue headers. Clicking on a category recursively shows all the categories or items in it. When the user clicks on an item in the grid, they will be directed to the item's page as shown below.

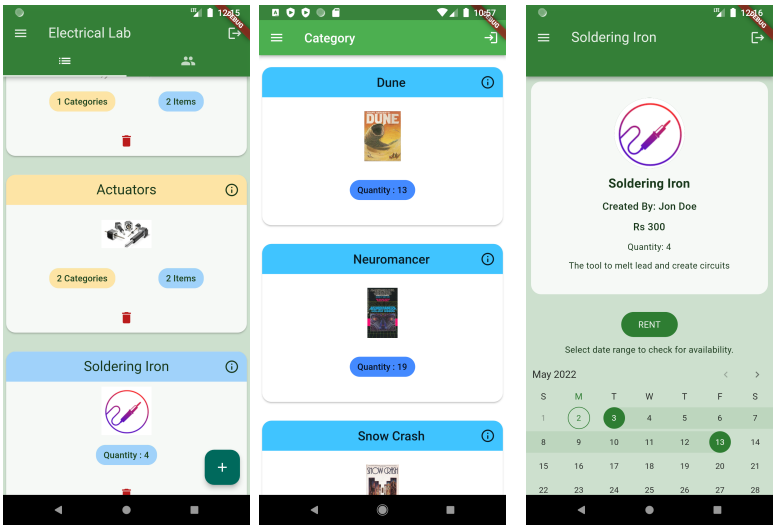

Fig. 7.3: 1. Category: Library, 2. Category: Science Fiction, 3. Item: Dune.

The super admins can create new facilities. The admins of the specific facility or category have the privilege to add a category, an item or a pack. To create a pack, they can use the global search functionality available in RMS.

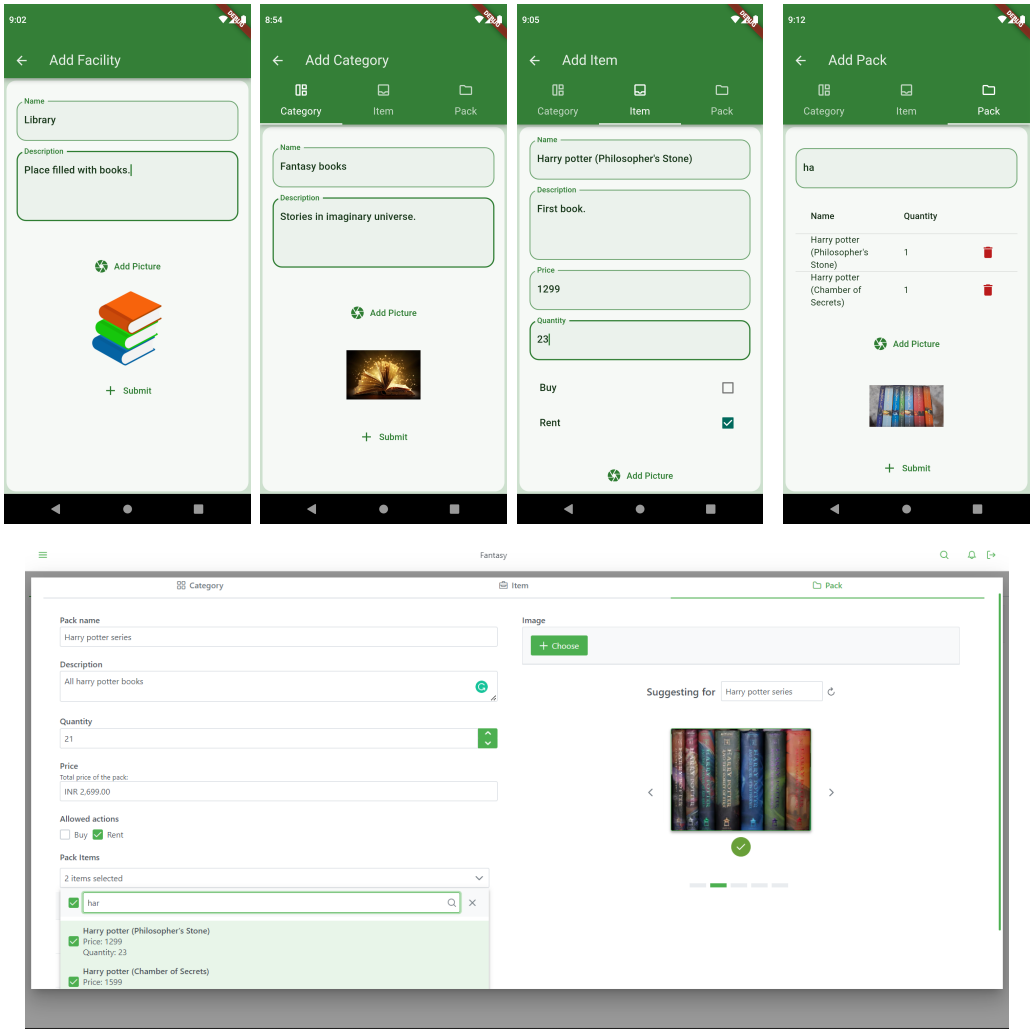

Fig. 7.4: Adding of 1. Facility, 2. Category, 3. Item, 4. Pack (mobile), 5. Add Pack (Web)

The item can be rented or brought based on the availability of that item in the desired date interval. The user has to specific his/her desired date interval and check for availability. If available they can buy/rent it based on the item configuration.

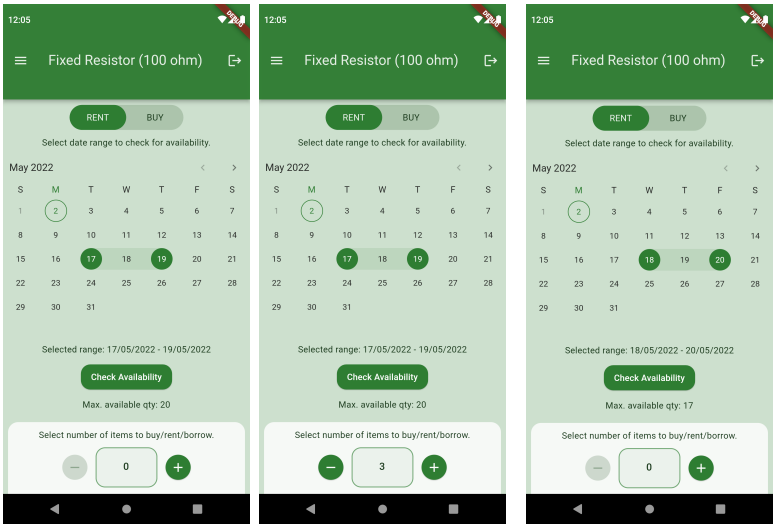

Fig. 7.5: 1. Check availability, 2. rent: three items, 3. checking availability with overlapping intervals (from left).

The user can click on activities in the side bar to view all their activities done in RMS.

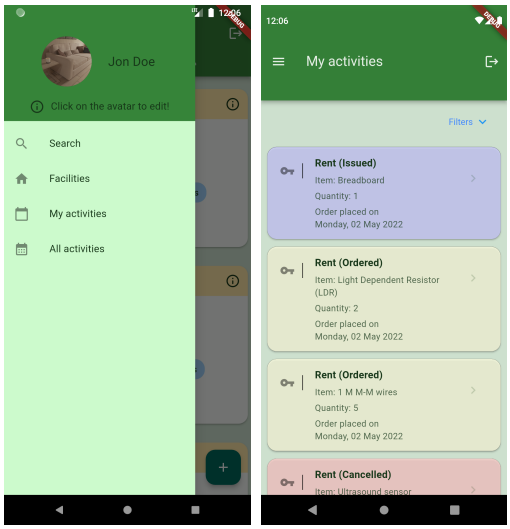

Fig. 7.6: 1. Sidebar, 2. user activities.

Clicking on the filter option, one can add variety of filter as shown below.

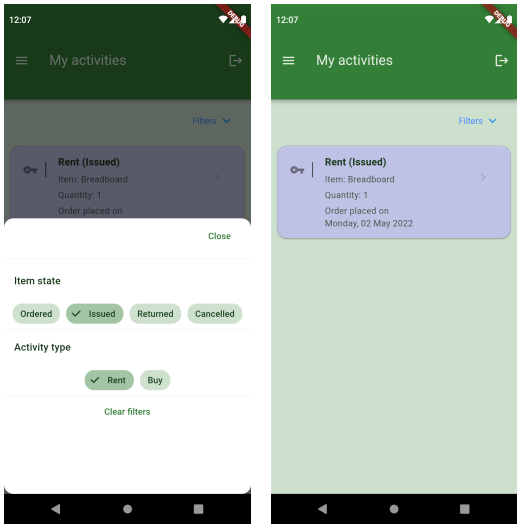

Fig. 7.7: 1. All activities, 2. Dialog for applying activity filters, 3. Filtered activities (from left).

A rented item has three phases, ordered, issued, and returned. Every phase of the activity can be seen by clicking on the activity.

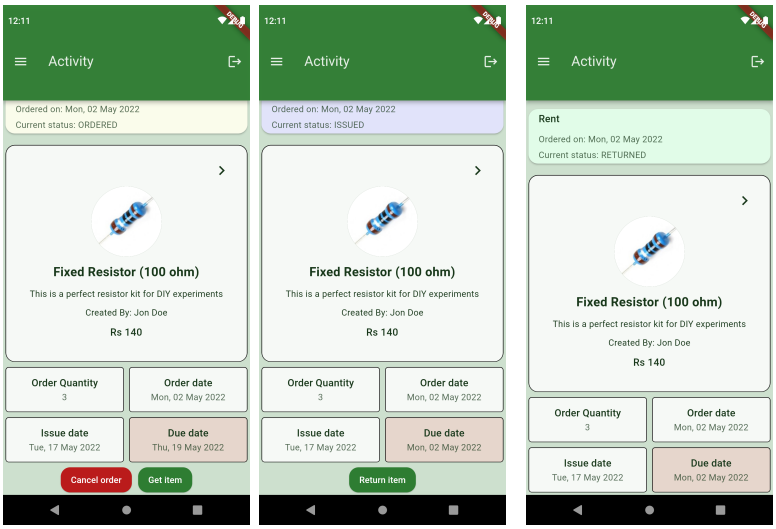

Fig. 7.8: 1. order preview, 2. order issued, 3. item returned.

The RMS provides a global search with which the users can easily navigate to their desired entity.

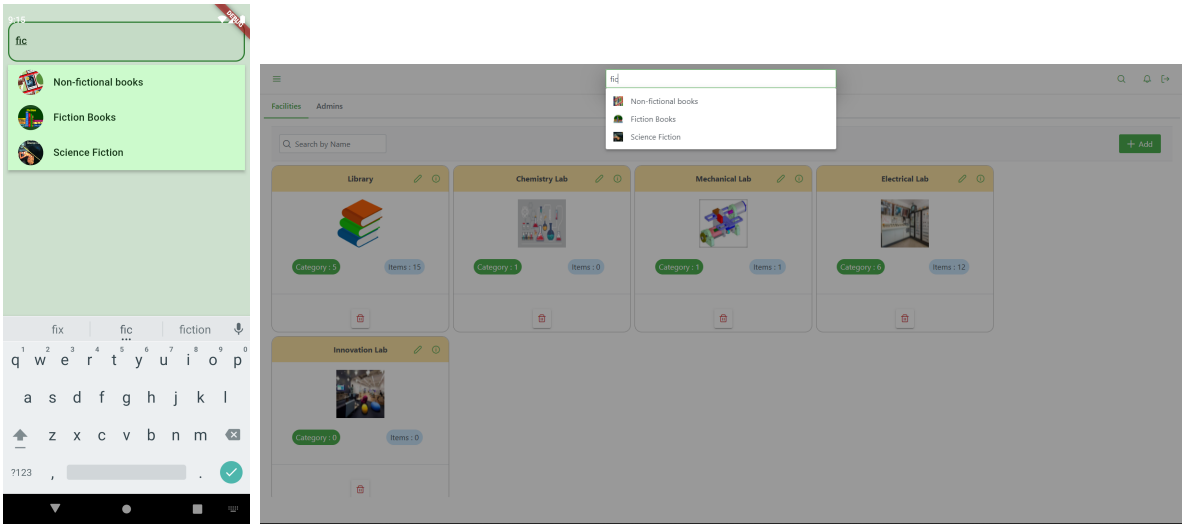

Fig. 7.9: 1. mobile, 2. web.

The closely related items can be seen in the also got with panel of the item details.

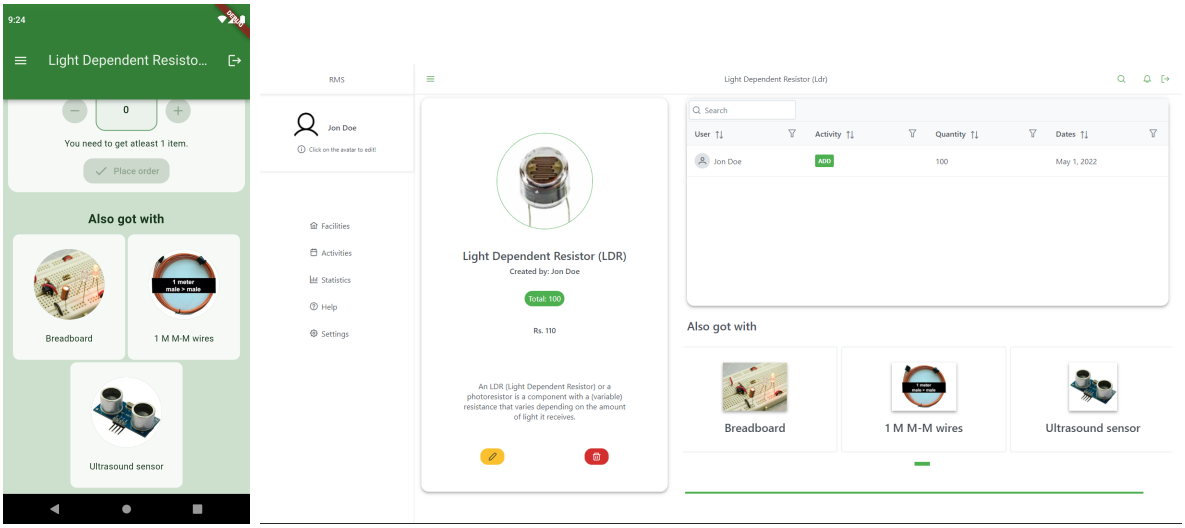

Fig. 7.10: 1. mobile, 2. web.

Chapter 8

Conclusion and Timeline

The resource management system provides a generalized approach for the management of facilities and the resources of any organization. The software support cross-platform interaction as it was build for web, android and iOS devices. It has a simple, intuitive user interface powered by a durable, scalable and robust back-end. It uses the latest cutting-edge frameworks and technologies to achieve many modern features and requirements. The RMS software will be a great asset to institutions and organizations.

8.1 Timeline

8.1.1 7^{th} semester mid-term

- Initiate the project. Created the repository, and decided the technical stack.
- Designed the database schema
- Designed the admin web portal using Figma.

8.1.2 7^{th} semester end-term

- Developed the server using NestJS and MongoDB.
- Developed the admin web portal using Angular.
- Implemented features such as adding, updating, and removing all entities, along with image suggestion.

8.1.3 8^{th} semester mid-term

- Developed the mobile app for users using Flutter.
- Implemented the borrowing, buying and renting logics.

8.1.4 8^{th} semester end-term

- Added admin privileges in the mobile app.
- Migrated mobile app to GetX Architecture (MVVM).
- Incorporated packs into RMS.
- Implemented the global search and recommendation system.
- Made the UI aesthetically better.

For the source, visit our GitHub repository.

References

1. https://metamug.com/article/security/jwt-java-tutorial-create-verify.htm
2. https://docs.mongodb.com/manual/applications/data-models-tree-structures/
3. https://www.mongodb.com/developer/article/mongodb-schema-design-best-practices/
4. https://www.figma.com/resources/learn-design/
5. https://graphql.org/
6. https://www.apollographql.com/
7. https://angular.io/
8. https://nestjs.com/
9. https://www.docker.com/
10. https://www.postman.com/
11. https://httptoolkit.tech/blog/cors-proxies/
12. https://pub.dev/packages/get
13. https://flutter.dev/