CM2020 Agile Software Projects – Final report

**SpaceWise**

A web-based seat and study room reservation system for libraries and co-working spaces

Authors: Jones Ullagaddi, Zachary Brown, Indu Singh, Roy Mattar

Revision: 02D

University of London

BSc (Computer Science)

2024 October Team11 Tutor Group 5

[Abstract 5](#_Toc190529870)

[Outcome 5](#_Toc190529871)

[Introduction 5](#_Toc190529872)

[Aims and Objectives 6](#_Toc190529873)

[SMART Aims 6](#_Toc190529874)

[Objectives 6](#_Toc190529875)

[Agile Project 7](#_Toc190529876)

[Agile Manifesto 7](#_Toc190529877)

[Data travel – Overview 8](#_Toc190529878)

[Network Connectivity Diagram 8](#_Toc190529879)

[Planning and Research 10](#_Toc190529880)

[Planning overview with Gantt Chart 10](#_Toc190529881)

[Kanban Board 11](#_Toc190529882)

[Resource and time allocation 12](#_Toc190529883)

[Resource Allocation 12](#_Toc190529884)

[Frontend 13](#_Toc190529885)

[Type of Frontend 13](#_Toc190529886)

[Tools used for the Frontend 13](#_Toc190529887)

[Size of every single screen 13](#_Toc190529888)

[Color scheme used for the user interface 13](#_Toc190529889)

[Font used for the Frontend 14](#_Toc190529890)

[Sequence of the screens 14](#_Toc190529891)

[Student screens 14](#_Toc190529892)

[Administrators screen 14](#_Toc190529893)

[Logic behind the screen 14](#_Toc190529894)

[Administrator’s Screens 14](#_Toc190529895)

[Students Screens 15](#_Toc190529896)

[Student Home page 15](#_Toc190529897)

[Student Reservation 15](#_Toc190529898)

[Student Cancel Reservations 15](#_Toc190529899)

[Backend 16](#_Toc190529900)

[Tools used for the Backend 16](#_Toc190529901)

[Size of screen 16](#_Toc190529902)

[Color scheme used for the user interface 16](#_Toc190529903)

[Database Tables 17](#_Toc190529904)

[UML – Table relations 18](#_Toc190529905)

[Interface Diagram of Frontend and Backend modules 19](#_Toc190529906)

[User Experience (UX) 20](#_Toc190529907)

[References 24](#_Toc190529908)

[Appendix A – Meeting logs 25](#_Toc190529909)

[Appendix B 26](#_Toc190529910)

[Rest API Psudocode 27](#_Toc190529911)

[Appendix B 32](#_Toc190529912)

[Appendix C – White box Testing 33](#_Toc190529913)

[Institutions Routes 33](#_Toc190529914)

[Auth Route 40](#_Toc190529915)

[Reservations Route 44](#_Toc190529916)

[Seat Management Route 45](#_Toc190529917)

[Authentication Route 46](#_Toc190529918)

[Appendix D – Black box Testing 47](#_Toc190529919)

[Anatomy of a web page 47](#_Toc190529920)

[Student Screens 48](#_Toc190529921)

[Admin Screens 53](#_Toc190529922)

**Table of Figures**

[Figure 1 Agile principles for Agile Manifesto 7](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529923)

[Figure 2 Network Connectivity diagram 8](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529924)

[Figure 3 Project planning overview 10](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529925)

[Figure 4 Kanban Board in Github with issue tracking 11](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529926)

[Figure 5 UML representation of relationship of Tables 18](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529927)

[Figure 6 Interface Diagram of FrontEnd and BackEnd modles 19](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529928)

[Figure 19 Anatomy of a web page 47](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529929)

[Figure 7 Register Screen 48](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529930)

[Figure 8 Login Screen 48](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529931)

[Figure 9 Student Home Screen 49](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529932)

[Figure 10 Select a Library Screen 49](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529933)

[Figure 11 Manage Space Details Screen 50](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529934)

[Figure 12 Manage Reservations Screen 50](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529935)

[Figure 13 My Reservations Screen 51](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529936)

[Figure 14 Select a Space Screen 51](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529937)

[Figure 15 Select a Seat Screen 52](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529938)

[Figure 16 Register Screen 53](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529939)

[Figure 17 Login Screen 53](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529940)

[Figure 18 Admin Dashboard 54](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529941)

[Figure 19 Manage Your Institution 54](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529942)

[Figure 20 Manage Spaces 55](file:///C:\1_UoL\1_ASP_CM2020\1%20Final%20Report\Report\ASP%20Final%20Report%20-%20Group%205%20Team%2011-%20Spacewise%20R2D.docx#_Toc190529943)

# Abstract

This project is a tool which allows users to reserve their library seat. This tool runs as app for anyone with a Phone / Tablet / Laptop. Users will choose a nick name (instead of real name). Users then book their library seat reservation in their city library branch (any library branch any –available seat in their city). While allowing the user to book (or cancel their booking) the seat, the user also has restriction of 3 bookings per day.

Anyone can use this app. Library administrators can delete the User if they find multiple accounts by the same user.

# Outcome

The goal is to have an app instead of current paper based booking. The app provides freedom to book the seats or rooms. The user can be anywhere and book their seat. The user can use any device and book their seat. Users can also use any other device to cancel their seat. Users can choose any nick name of their choice. Users or Administrators can cancel the nick name. The number of user is not limited. Having said about the freedom and unlimited use; each device can book 6 users.

# Introduction

This app is developed by team of developers. There is Frontend developer; There is also Backend developer with one full stack developer. The Frontend team will focus on the database, and main operational work area for the running the algorithms / database / test tools / profilers / Query managers.

The Frontend team will focus on how the data is presented to the users. User’s perception of the application. User judgment of data. User handling of data, Some users may prefer to use their phone while other users may want to use their laptop.

How users will receive the data. How users will respond to data. Will the users be quiet to themselves (without providing feedback) or users will try to over active.

What kind of color the users presented. The type of font used by the users. User preferences of using the application for themselves or for the group of friends with password sharing.

These are few questions to be monitored by the administrators. Is provides valuable feedback to the administrators as well as the app developers. The behaviors of the individual users very good and valuable information. These are few functionalities the Frontend developers can use.

# Aims and Objectives

The primary aim of this project is to develop an intuitive, scalable, and user-friendly reservation system for library seats and co-working spaces, enhancing accessibility, efficiency, and organization for users and administrators alike.

### SMART Aims

Specific: Develop a web-based application for users to reserve library seats and study rooms in real-time, providing features such as interactive library maps, reservation management, and notifications.

Measurable: Launch the application in one library, achieve a 60% adoption rate among regular users within three months, and reduce the time spent searching for a seat by 50%.

Achievable: Use existing web development skills within the team, focusing on intuitive UI/UX design and integrating with the library’s database for accurate availability tracking.

Relevant: Improve user satisfaction by streamlining the process of finding and reserving library seats and study rooms, addressing a key student pain point.

Time-Bound: Develop a functional prototype within two months and complete the pilot rollout in four months.

### Objectives

1. **Research and Validation**: Conduct thorough market research to identify gaps in existing solutions and understand user needs.
2. **Design and Development**: Build a functional prototype incorporating interactive maps, reservation tools, and a notification system.
3. **User Experience (UX) Optimization**: Conduct user surveys and usability tests to refine the interface for diverse user groups.
4. **Legal and Ethical Considerations**: Ensure compliance with privacy laws and address user concerns about data security.
5. **Scalability**: Develop a robust Backend capable of handling high user demand across multiple locations.
6. **Business Viability**: Analyze the financial model, potential partnerships, and long-term sustainability of the application.

# Agile Project

Agile projects follow 12 principles which are core elements to manage the agile project. Every project which follows agile principles must follow these 12 principles.

## Agile Manifesto

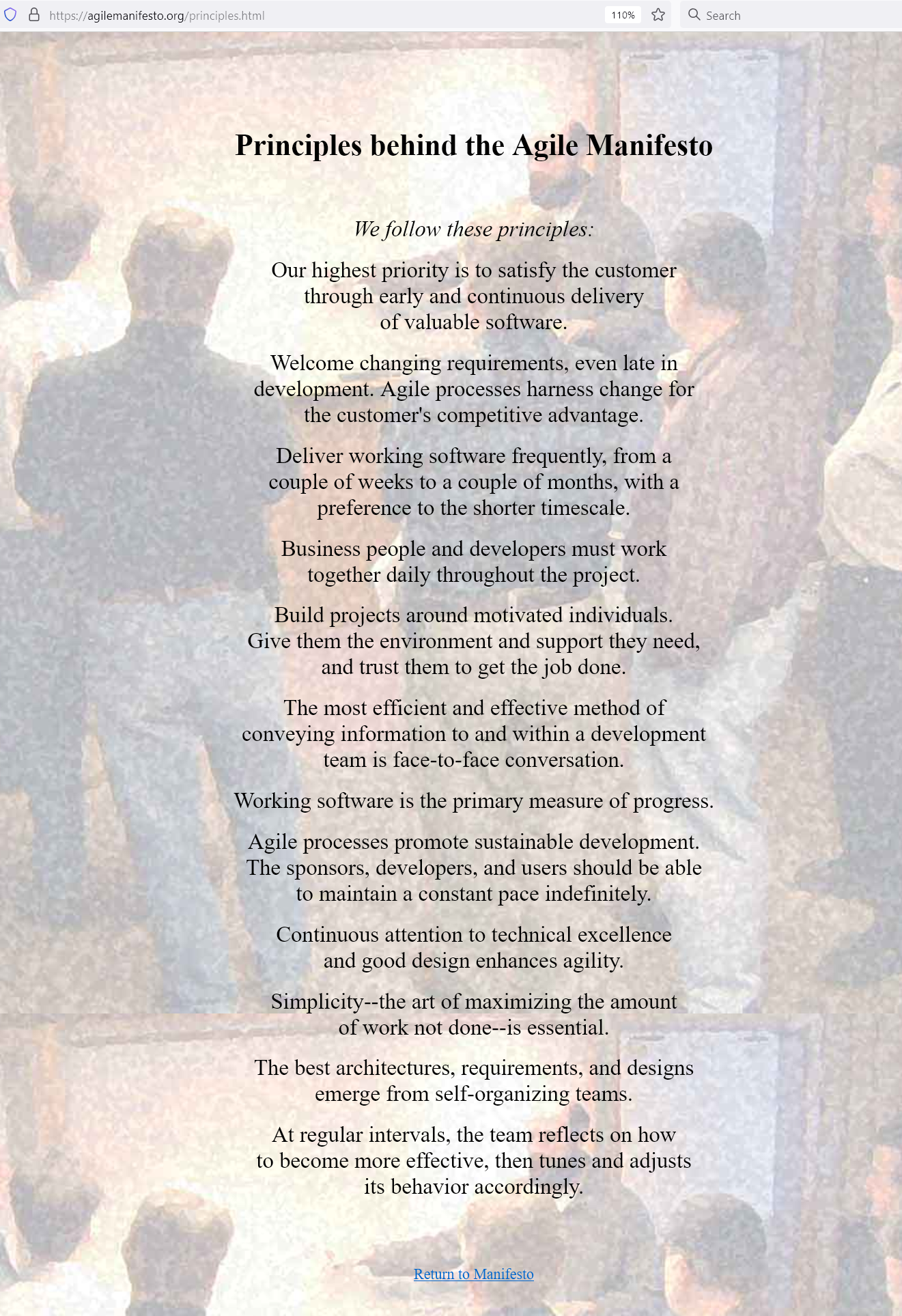


Figure Agile principles for Agile Manifesto

# Data travel – Overview

Data travels through 7 layers of OSI (Open System Interconnectivity) for every Data exchange between Frontend and Backend.

Layer 7: Application layer - High level protocols. Ex. HTTP, HTTPS, Node, React, SQLite.

Layer 6: Presentation layer - Data Transmission between Application and Network

Layer 5: Session layer - Data Managing of communication sessions between Network Nodes.

Layer 4: Transport layer - Reliable transmission of Data segments between points on networks.

Layer 3: Network layer - Management of Addressing / Routing / Traffic control.

Layer 2: Data link layer - Transmission of data frames between Nodes.

Layer 1: Physical layer - Physical medium; Copper cables, Fiber optic networks, RF/Satellites.

## Network Connectivity Diagram

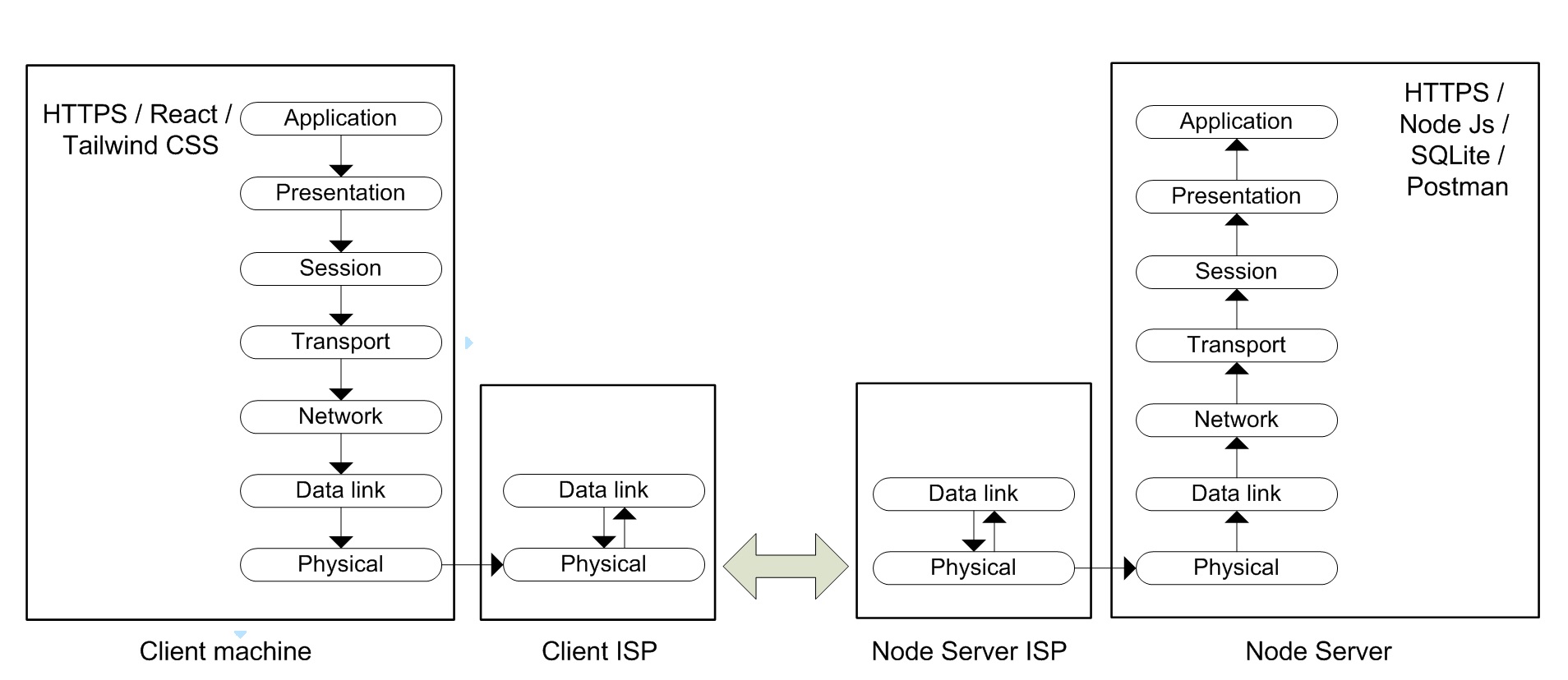


Figure Network Connectivity diagram

The above diagram illustrates a simple communication message flow from React / HTTPS to Node / HTTPS.

User request (Request by React)is picked up by Client machine Application layer. The message then travels through 7 layers (layer 7 to layer1) of Client machine and reaches Client ISP.

Client ISP sends the message to Node Server ISP through routing table.

User request (from React) then travels from Node Server (ISP) to Node Server.

Then finally, the message (Client request from React) travels (from layer 1 to layer 7) to Application layer of Node Server.

Note: All messages from Client to Server (or from Server to Client) follow this above illustrated path.

# Planning and Research

## Planning overview with Gantt Chart

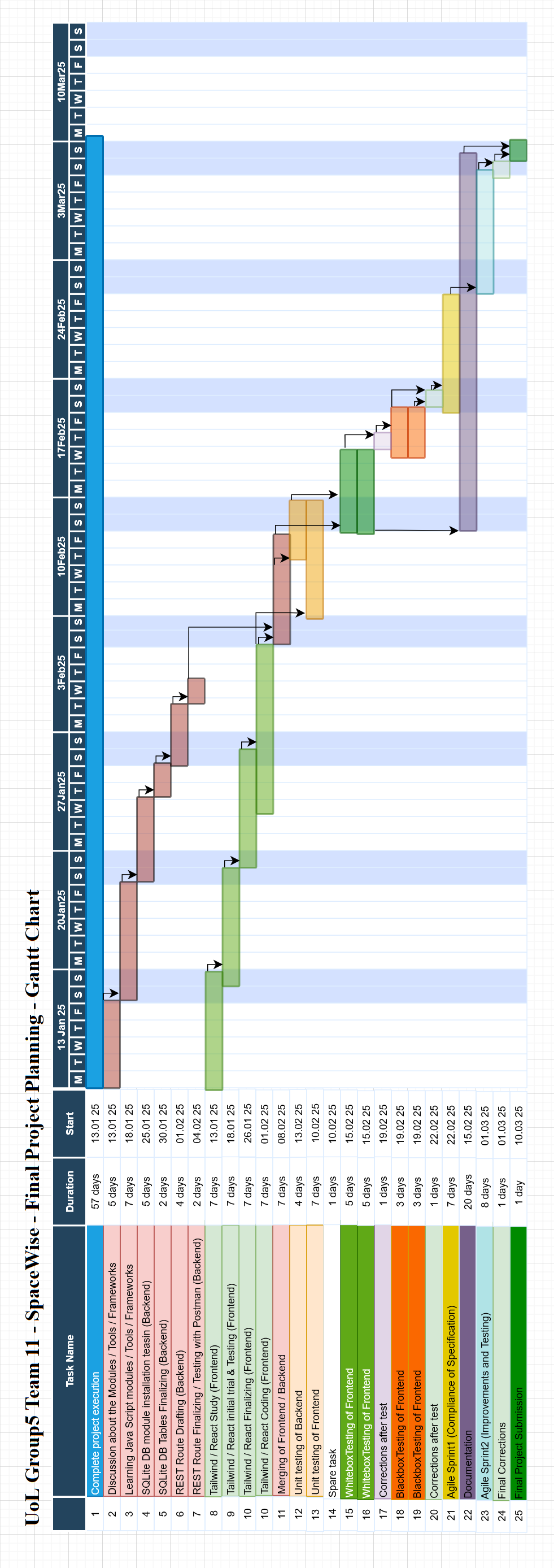


Figure Project planning overview

## Kanban Board

Kanban board in Github is used to do issue tracking. This is a good tool to track issues in pending, Issues in progress and Issue completed.

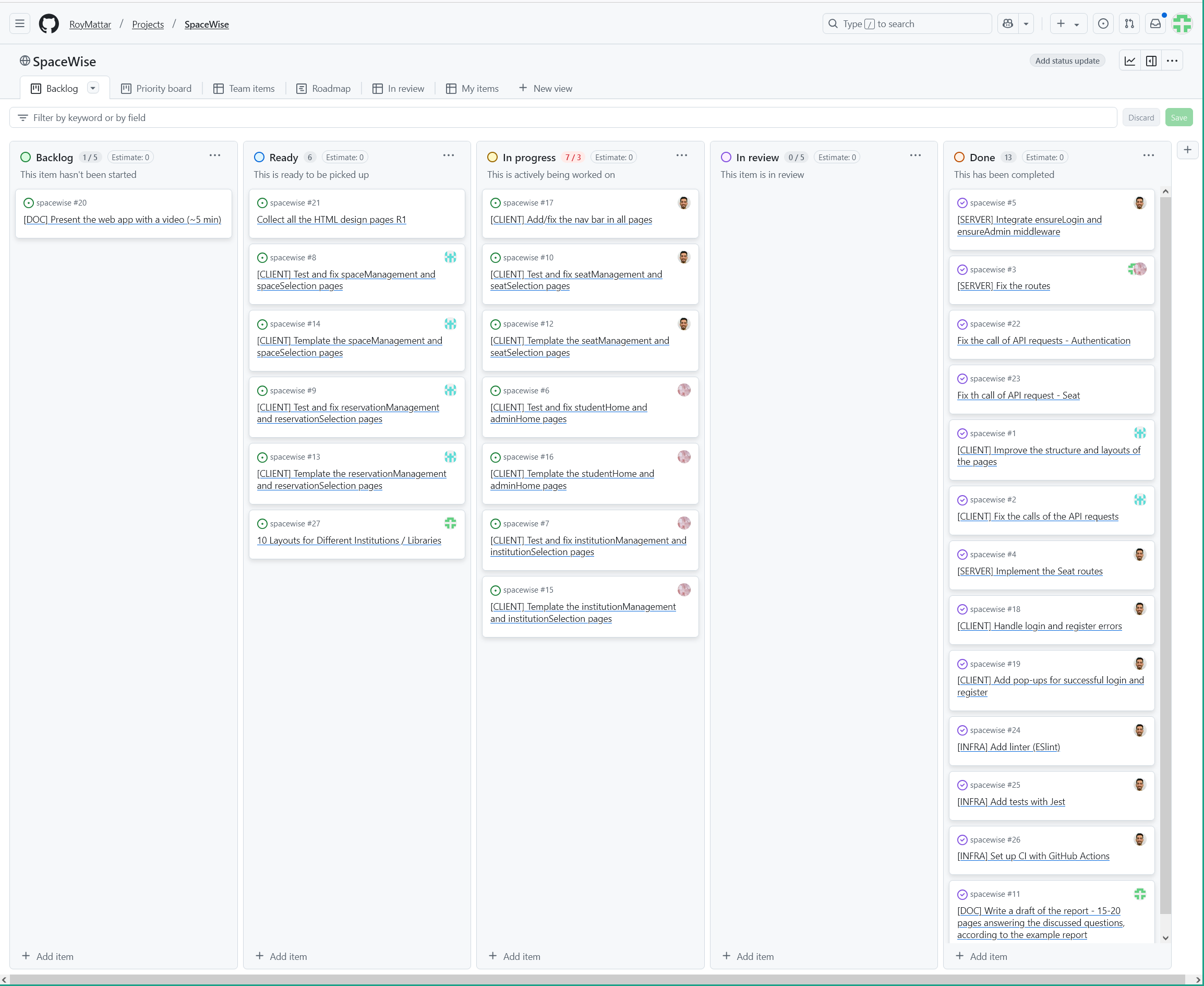


Figure Kanban Board in Github with issue tracking

## Resource and time allocation

Frontend and Frontend teams work concurrently to develop the application. There will be constant information exchange between Frontend team and Frontend team. The university has provided very good communication tool call Slack. Slack is used for every communication between the developers (Frontend as well as Backend).

The first two weeks were used by the team to get familiar with the tools / libraries / protocols / APIs. The team also spent time in knowing the capabilities of the tools. Some users were not familiar with the tools. The first two weeks used to get familiar the tool.

Though the first half of semester was used for the research; The first half of the semester mainly concentrated on market research, project options, project reasoning, project selection etc.

### Resource Allocation

**Roy Matter**: Did most of the work in the project. He did conceptual work and Planning. He programmed frontend as well as backend of the project. He organized the team and arranged the weekly team meeting in slack.

**Zachary Brown**: Did the front end design of the project. He programmed the React, Tailwind CSS.

**Indu Singh**: Did most of the backend design, SQL programming and REST API programming. She also did most of documentation.

**Jones Ullagaddi**: Did some work in Layout of Library and documentation.

## Frontend

Frontend mainly concentrated on the dealing with end users. This involved

* Type of Frontend
* Tool used for the Frontend
* Size of every single screen
* Color scheme used for the user interface
* Font used for the Frontend
* Sequence of the screens
* Logic behind the screen

### Type of Frontend

This part of section deals with what kind of Frontend selected. To be specific Frontend planned to work in windows environment (windows 7, windows 10, windows 11). Frontend planned to work in Linux environment. And Frontend also planned to work with Apple Macintosh.

### Tools used for the Frontend

Tools were carefully selected to keep the system small and efficient.

Frontend selected is React with Tailwind CSS.

React works in conjunction with Tailwind CSS to give the best user experience. Frontend dynamically changes the Seat management and Institution screens based on the number of seats and size of the institution.

### Size of every single screen

While most of the screens are common size, some screens such as Institution screen and seat management screen varies based on the number of type of institution and number of seats available to reserve in the institution.

### Color scheme used for the user interface

All the screens have white background.

Screen where user input is required is with white background and black text. Default texts are displayed in grey color.

Every screen has preferred user action; this type of buttons is with Purple button.

Screen with Registration, Confirm seat are displayed in Black color.

Seats to be Added / add Confirm are with Green button.

Seats to be Deleted are with Red button.

Seats to be Moved / Edited are with Blue button.

### Font used for the Frontend

Default font is managed by “fonts.googleapis.com/css2” this font changes based on the global setting.

### Sequence of the screens

Every client opens with ‘Welcome to SpaceWise’ screen.

Based on the user (Admin or Student) screens will go to respective area of operation.

#### Student screens

* Register (register.html)
* Login (login.html)
* Student Home (studentHome.html)
* Select a Library (institutionSelection.html)
* Manage Space Details (seatManagament.html)
* Manage Reservations (reservationManagement.html)
* My Reservations (reservationSelection.html)
* Select a Space (spaceSelection.html)
* Select a Seat (seatSelection.html)

#### Administrators screen

* Register (register.html)
* Login (login.html)
* Admin Dashboard (adminHome.html)
* Manage your Institution (institutionManagement.html)
* Manage Spaces (spaceManagement.html)

### Logic behind the screen

Depending in the user type (Administrator or Student) screens are divided into two main category (refer to previous section – Sequence of the screens for more details on the screens).

#### Administrator’s Screens

* Administrators will be able to create/delete Institute using “Manage your Institution” screen.
* Administrators will be able to create/delete Space using “Manage Spaces” screen.

#### Students Screens

Upon startup students will be greeted with Login screen.

New student will go to Register screen (student will not get another option until registering).

#### Student Home page

Registered students will be directed to “Student Home” page.

#### Student Reservation

Upon registering students will be able to book either “Co-Working Space” or “Library Space” for their booking. Or Students will go to “Manage Reservations” to Cancel their reservation.

* Student then have to select “Select a Space”.
* After selecting Space Student will then be able to reserve a seat through “My Reservations”.

#### Student Cancel Reservations

* Student will have to select “Manage Reservations” from “Student Home” page.

## Backend

Backend holds and manages all the data. It serves Frontend’s request for data and also saves Frontends data.

### Tools used for the Backend

Following are the Software and Tools used for Backend

Node JS – Main operating System for Server

SQLite database – Database to serve front end’s requests

Espress JS – API requests to node

Postman – Testing tool for SQLite database (in lieu of Frontend)

MS Studio Code – Microsoft Studio code is the editor / IDE used for Programing

NPM – package manager

### Size of screen

There is no screen size for Backend tools. Developer is free to use the screen size of choice.

### Color scheme used for the user interface

Backend tools don’t use color schemes. Developer is free to use the screen size of choice.

## Database Tables

institution\_admins

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column** | **Type** | **Comment** | **PK** | **Nullable** | **Default** |
| admin\_id | int |  | YES | NO |  |
| institution\_id | int |  |  | NO |  |

institutions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column** | **Type** | **Comment** | **PK** | **Nullable** | **Default** |
| institution\_id | int |  | YES | NO |  |
| name | varchar(100) |  |  | NO |  |
| bio | text |  |  | YES |  |
| address | varchar(255) |  |  | NO |  |
| opening\_hours | varchar(50) |  |  | YES |  |
| admin\_id | int |  |  | NO |  |

reservations

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column** | **Type** | **Comment** | **PK** | **Nullable** | **Default** |
| reservation\_id | int |  | YES | NO |  |
| space\_id | int |  |  | NO |  |
| seat\_id | int |  |  | NO |  |
| user\_id | int |  |  | NO |  |
| start\_time | datetime |  |  | NO |  |
| end\_time | datetime |  |  | NO |  |
| status | enum('active','expired','cancelled') |  |  | NO |  |

seats

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column** | **Type** | **Comment** | **PK** | **Nullable** | **Default** |
| seat\_id | int |  | YES | NO |  |
| space\_id | int |  |  | NO |  |
| name | varchar(50) |  |  | NO |  |
| type | varchar(50) |  |  | YES |  |
| facilities | json |  |  | YES |  |
| status | enum('available','reserved','unavailable') |  |  | NO |  |

spaces

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column** | **Type** | **Comment** | **PK** | **Nullable** | **Default** |
| space\_id | int |  | YES | NO |  |
| institution\_id | int |  |  | NO |  |
| name | varchar(100) |  |  | NO |  |
| layout | text |  |  | YES |  |

students

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column** | **Type** | **Comment** | **PK** | **Nullable** | **Default** |
| student\_id | int |  | YES | NO |  |

users

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column** | **Type** | **Comment** | **PK** | **Nullable** | **Default** |
| user\_id | int |  | YES | NO |  |
| username | varchar(50) |  |  | NO |  |
| password\_hash | varchar(255) |  |  | NO |  |
| role | enum('admin','student') |  |  | NO |  |

## UML – Table relations

Following is a UML representation of relationship between tables.

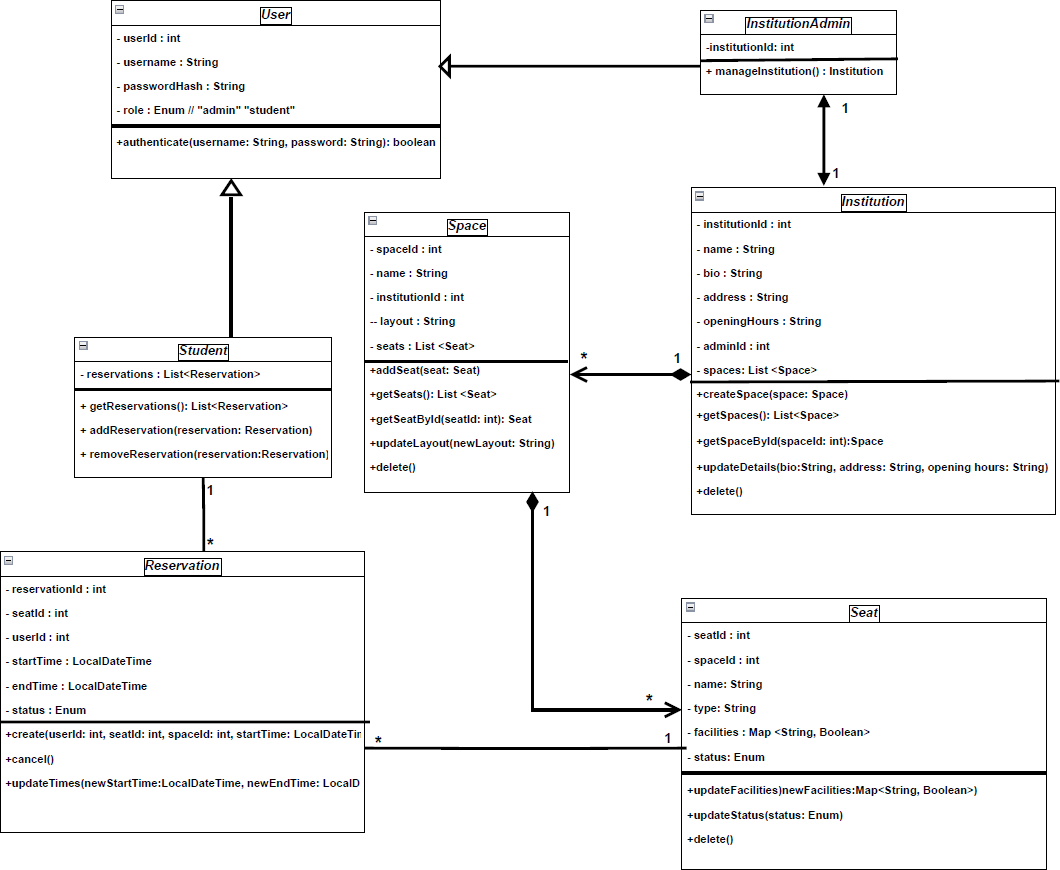


Figure UML representation of relationship of Tables

## Interface Diagram of Frontend and Backend modules

Interface is divided into two major sections.

Frontend which contains React.JS, Tailwind.CSS is loaded onto client’s device as soon as website’s HTML starts to load.

Backend consists of Node.js (loaded on Windows server or Linux server or Unix server) and SQLite Database engine. Express.JS is used to interact between Node and SQLite. The server requires frequent patching and upgrades which is managed by NPM package manager.

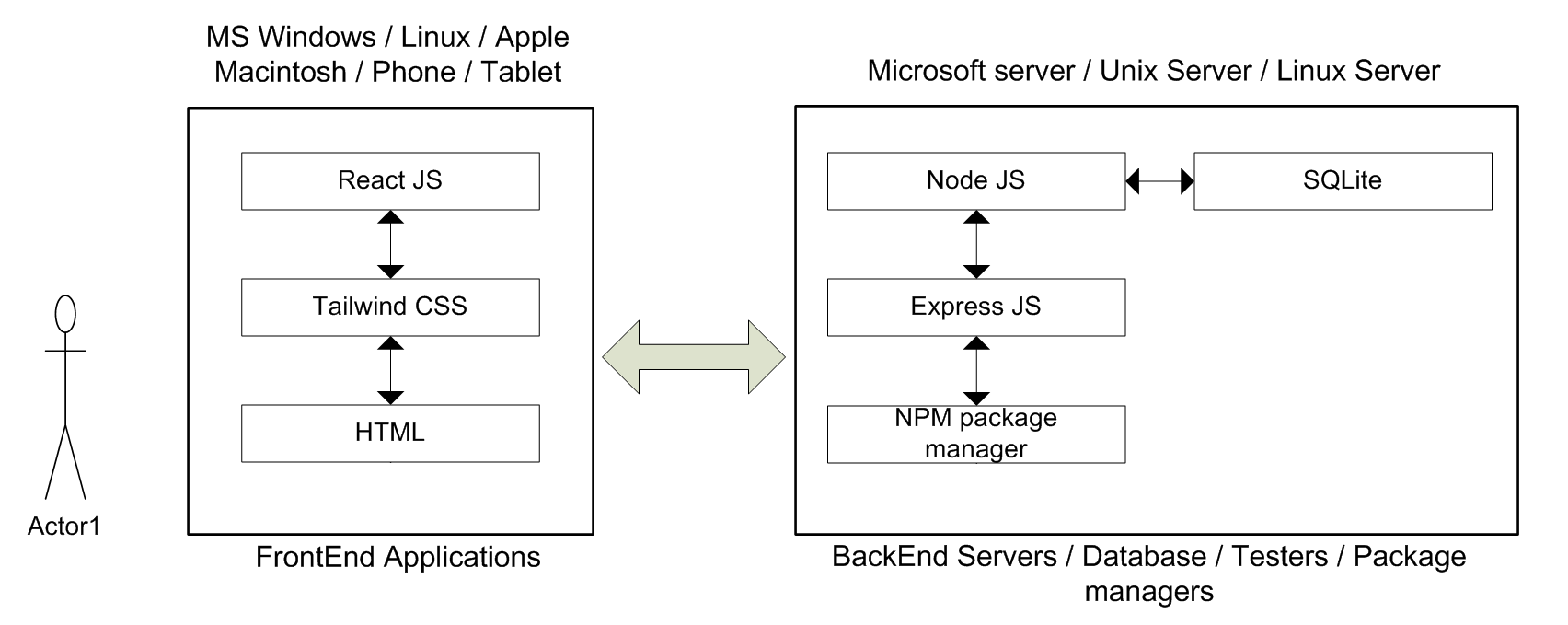


Figure Interface Diagram of FrontEnd and BackEnd modles

# User Experience (UX)

Frontend deals with user experience. Many measurable tasks with Frontend of our application.

* All the screens have responsive behavior. Same application can run in iPhone / Tablet / Android / Desktop / Laptop / Windows / Apple Macintosh / Linux etc. The application resizes to different size screens yet functions remain the same.
* Application designed considering color blind people. Most of color blind have either Bed color blindness or some have blue color blindness. So CSS does not select RED (255,0,0) or BLUE (0,0,255). Some buttons such as Delete must be in red color; this type of button has been selected as bg-red-600 (bg-red-800 while hovering). Some buttons such as Edit must be in blue color; this type of button has been selected as bg-blue-600 (bg-blue-800 while hovering).
* Application has been designed with good fonts from Google. All the fonts have reference to “https://fonts.googleapis.com/css2” and this font has been tested by Google for best performance and readability.
* Rounded edge buttons, forms, screens make more modern appeal.
* Some Elements such as “bio” in the screen “institutionManagement” takes multiple lines as input. This screen starts with minimum height and based on the number of characters entered and the screen size the dialog box automatically changes the height of the text box.
* Since the font and buttons are standard from Google. Adoption arte will be more than 60% (for young and old users).
* Readability is one of the prime goals of this project. All the screens are readable by using “https://fonts.googleapis.com/css2”. In addition enough spacing (minimum padding) has been placed to make sure all the screen elements have been readable.

## References

1. **Nielsen, J. (1GG4).** 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. Retrieved from <https://www.nngroup.com/articles/ten-usability-heuristics/>
2. **Node.js** Java script runtime built on Chrome’s V8 JavaScript engine. Retrieved from <https://nodejs.org>
3. **React.js** Is the library for web and native user interfaces. React is designed to let you seamlessly combine components written by independent Community of users. Retrieved from <https://react.dev>
4. **Expressjs.** Is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Retrieved from <https://expressjs.com>
5. **SQLite.** Is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. It is available for all the operating systems and is lightweight. Retrieved from <https://www.sqlite.org>
6. **Postman.** Is a global software company that offers an API platform for developers to design, build, test, and collaborate on APIs - Faster. Retrieved from <https://www.postman.com>
7. **Tailwind CSS.** Is a utility-first CSS framework for rapidly building modern websites without ever leaving your HTML. Retrieved from <https://tailwindcss.com>
8. **HTML.** Is the standard markup language for creating Web pages. Retrieved from <https://www.w3schools.com/html/default.asp>
9. **CSS.** Is the language we use to style an HTML document. Retrieved from <https://www.w3schools.com/Css/>
10. **Figma. (n.d.).** Collaborative Interface Design Tool.Retrieved from [https://www.figma.com](https://www.figma.com/)
11. **UX Collective. (2023).** Heuristic Evaluation: A Guide to Evaluating Usability. Retrieved from <https://uxdesign.cc/>
12. **OSI Model.** Open Systems Interconnection model is a reference model from the ISO that provides a common basis development for the purpose of systems interconnection. Retrieved from <https://www.iso.org/standard/20269.html>
13. **Git.** Is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Retrieved from <https://git-scm.com/>
14. **GitHub. (n.d.).** Version Control and Collaboration. Retrieved from [https://github.com](https://github.com/)
15. **ISO. (201G).** ISO 8c01:2004 Data elements and interchange formats - Information interchange - Representation of dates and times. Retrieved from <https://www.iso.org/>
16. **Slack. (n.d.).** Collaboration Made Simple.Retrieved from [https://slack.com](https://slack.com/)
17. **Google Forms. (n.d.).** Create and Analyze Surveys for Free.Retrieved from <https://www.google.com/forms>
18. **StackOverflow. (n.d.).** Ǫuestions on API and HTTP Protocols. Retrieved from <https://stackoverflow.com/>
19. **Academia. (n.d.).** The Role of Structured Learning Environments.Retrieved from [https://www.academia.edu](https://www.academia.edu/)

# 

# Appendix A – Meeting logs

18th January: Discussed the Second phase time lines. Discussed the skill gaps of the team. Discussed what skills are need for every individual to reach the team’s final goal of project completion in time.

-Learning of Tailwind CSS, React JS, SQLite, Express JS, Postman

25th January: Discussed overview of the architecture of SpaceWise project.

Discussed about Frontend technology. Tools to be used for the Frontend. Discussed about Backend technology. Tools required for Backend.

1st February: Backend CURD operations table was referred from mid-term submission. Routing tables were built. Backend technology Node-js and SQLite were discussed. Frontend technology React was discussed.

8th February: Routing tables were finalized. Java script for routing table and REST API were finalized. Poastman tool was used to test the Backend. Frontend REACT and TAILWIND CSS were checked for their operation.

-Learning of postman

15th February:

Whitebox testing of Frontend and Backend modules. Blackbox testing of Frontend and Backend modules.

22nd February:

First spring run was done. First iteration was done with first set of tests.

1st March:

Second sprint run was done. Second iteration done with second set of tests. Kan ban board updated.

8th March: Finalizing of the SpaceWise application. Final touch to Frontend / Backend modules. Final integrated testing (prior to submission) done. Documentation checked for errors, formatting and consistency. Application and Document was finalized.

10th March: Final meeting before submission, Checked all the deliverables. Submitted the document.

# Appendix B - Rest API Psudocode

**Authentication**

* **POST /auth/register**
  + **Register a new user or admin.**
  + **Input:**

**{**

**"username": "admin123",**

**"password": "securepassword",**

**"role": "admin" // or "student"**

**}**

* **Output:**

**{**

**"user\_id": 1,**

**"message": "User registered successfully."**

**}**

**Query:**

INSERT INTO users (username, password\_hash, role)

VALUES (

  'admin123', 'securepassword', 'admin'

);

SELECT user\_id FROM users WHERE username = 'admin123';

* **POST /auth/login**
  + **Authenticate student/admin and return a token.**
  + **Input:**

**{**

**"username": "admin123",**

**"password": "securepassword"**

**}**

* **Output:**

**{**

**"token": "jwt\_token",**

**"user\_id": 1,**

**"role": "admin"**

**}**

**Query:**

SELECT user\_id, role

FROM users

WHERE username = 'admin123' AND password\_hash = 'securepassword';

**Institution Management**

* **POST /institution/register**
  + **Creates an institution and links its admin.**
  + **Input:**

**{**

**"institution\_name": "Library Central",**

**"bio": "Main library of the city",**

**"address": "123 Library Street",**

**"opening\_hours": "9 AM - 8 PM",**

**"admin\_id”: 1**

**}**

* **Output:**

**{**

**"institution\_id": 1,**

**"message": "Institution registered successfully."**

**}**

INSERT INTO institutins (institution\_id, bio, address, opening\_hours, admin\_id)

VALUES (1, 'Library Central', 'Main library of the city', '9 AM - 8 PM', 1);

SELECT institution\_id FROM institutions WHERE name = 'Library Central';

* **GET /institutions**
  + **Retrieve a list of all institutions.**
  + **Output:**

**[**

**{**

**"institution\_id": 1,**

**"institution\_name": "Library Central",**

**"bio": "Main library of the city",**

**"address": "123 Library Street",**

**"opening\_hours": "9 AM - 8 PM"**

**},**

**{**

**"institution\_id": 2,**

**"institution\_name": "Co-working Space A",**

**"bio": "Modern space for startups",**

**"address": "456 Startup Lane",**

**"opening\_hours": "24/7"**

**}**

**]**

SELECT institution\_id, name, bio, address, opening\_hours FROM institutions;

* **GET /institutions/:id**
  + **Retrieve details of a specific institution.**
  + **Output:**

**{**

**"institution\_id": 1,**

**"institution\_name": "Library Central",**

**"bio": "Main library of the city",**

**"address": "123 Library Street",**

**"opening\_hours": "9 AM - 8 PM",**

**"admin\_username": "admin123"**

**}**

SELECT i.institution\_id, i.name, i.bio, i.address, i.opening\_hours AS admin\_username

FROM institutions i

JOIN institution\_admins ia ON i.institution\_id = ia.institution\_id

JOIN users u ON ia.admin\_id = u.user\_id

WHERE i.institution\_id = 1;

* **PATCH /institutions/:id**
  + **Update institution details.**
  + **Input:**

**{**

**"bio": "Updated bio for the institution",**

**"address": "New Address Street",**

**"opening\_hours": "8 AM - 6 PM"**

**}**

* **Output:**

**{**

**"message": "Institution details updated successfully."**

**}**

UPDATE institutions

SET bio = 'Updated bio for the institution',

    address = 'New Address Street',

    opening\_hours = '8 AM - 6 PM'

WHERE institution\_id = 2;

* **DELETE /institutions/:id**
  + **Delete the institution and its data.**
  + **Output:**

**{**

**"message": "Institution deleted successfully."**

**}**

DELETE FROM institutions WHERE institution\_id = 1;

**Space Management**

* **POST /institutions/:id/spaces**
  + **Create a space for a specific institution and upload a layout image.**
  + **Input:**

**{**

**"name": "Library Room A",**

**"layout\_image": "base64\_encoded\_image"**

**}**

* **Output:**

**{**

**"space\_id": 5,**

**"message": "Space created successfully."**

**}**

INSERT INTO spaces(name, layout)

VALUES (

    'Library Room A', 'base64\_encoded\_image'

);

SELECT space\_id FROM spaces WHERE name = 'Library Room A' AND institution\_id = 1;

* **GET /institutions/:id/spaces**
  + **Retrieve all spaces for a specific institution.**
  + **Output:**

**[**

**{**

**"space\_id": 5,**

**"name": "Library Room A",**

**"layout\_image": "https://example.com/uploads/room-a.png"**

**},**

**{**

**"space\_id": 6,**

**"name": "Library Room B",**

**"layout\_image": "https://example.com/uploads/room-b.png"**

**}**

**]**

SELECT space\_id, name, layout FROM spaces WHERE institution\_id = 1;

* **GET /institutions/:id/spaces/:spaceId**
  + **Retrieve details of a specific space, including the layout image.**
  + **Output:**

**{**

**"space\_id": 5,**

**"name": "Library Room A",**

**"layout\_image": "https://example.com/uploads/room-a.png"**

**}**

SELECT space\_id, name, layout FROM spaces WHERE institution\_id = 1 AND space\_id = 1;

* **PATCH /institutions/:id/spaces/:spaceId**
  + **Update space details and/or replace the layout image.**
  + **Input:**

**{**

**"name": "Updated Room Name",**

**"layout\_image": "base64\_encoded\_image"**

**}**

* **Output:**

**{**

**"message": "Space details updated successfully."**

**}**

UPDATE spaces

SET name = 'Updated Room Name',

    layout = 'updated.png'

WHERE institution\_id = 2 AND space\_id = 3;

* **DELETE /institutions/:id/spaces/:spaceId**
  + **Delete a specific space.**
  + **Output:**

**{**

**"message": "Space deleted successfully."**

**}**

DELETE FROM spaces WHERE institution\_id = 1 AND space\_id = 2;

**Seat Management**

* **POST /institutions/:id/spaces/:spaceId/seats**
  + **Add seats to a specific space within an institution.**
  + **Input:**

**{**

**"seat\_name": "A1",**

**"type": "Chair",**

**"facilities": ["Outlet", "Lamp"],**

**"status": "Available"**

**}**

* **Output:**

**{**

**"seat\_id": 101,**

**"message": "Seat added successfully."**

**}**

INSERT INTO seats (space\_id, name, type, facilities, status)

VALUES (2, 'C2', 'Chair', '["Outlet", "Lamp"]', 'Available');

SELECT seat\_id FROM seats WHERE name = 'C2' AND space\_id = 2;

* **PATCH /institutions/:id/spaces/:spaceId/seats/:seatId**
  + **Update details of a specific seat (e.g., facilities, status).**
  + **Input:**

**{**

**"type": "Table",**

**"facilities": ["Outlet"],**

**"status": "Unavailable"**

**}**

* **Output:**

**{**

**"message": "Seat details updated successfully."**

**}**

UPDATE seats

SET type = 'Table',

    facilities = '["Outlet"]',

    status = 'Unavailable'

WHERE space\_id = 3 AND seat\_id = 4;

* **DELETE /institutions/:id/spaces/:spaceId/seats/:seatId**
  + **Remove a specific seat.**
  + **Output:**

**{**

**"message": "Seat deleted successfully."**

**}**

DELETE FROM seats WHERE space\_id = 1 AND seat\_id = 1;

**Reservation Management**

* **POST /reservations**
  + **Create a reservation for a seat.**
  + **Input:**

**{**

**"seat\_id": 101,**

**"user\_id": 42,**

**"start\_time": "2024-12-22T09:00:00Z",**

**"end\_time": "2024-12-22T11:00:00Z"**

**}**

* **Output:**

**{**

**"reservation\_id": 201,**

**"message": "Reservation created successfully."**

**}**

INSERT INTO reservations (space\_id, seat\_id, user\_id, start\_time, end\_time, status)

VALUES (2, 2, 1, '2025-02-02 9:00', '2025-02-02 12:30', 'Active');

* **GET /reservations/:student\_id**
  + **Retrieve all reservations for a specific user.**
  + **Output:**

**[**

**{**

**"reservation\_id": 201,**

**"seat\_id": 101,**

**"seat\_name": "A1",**

**"start\_time": "2024-12-22T09:00:00Z",**

**"end\_time": "2024-12-22T11:00:00Z",**

**"status": "Active"**

**},**

**{**

**"reservation\_id": 202,**

**"seat\_id": 102,**

**"seat\_name": "B1",**

**"start\_time": "2024-12-23T14:00:00Z",**

**"end\_time": "2024-12-23T16:00:00Z",**

**"status": "Canceled"**

**}**

**]**

SELECT r.reservation\_id, r.seat\_id, r.start\_time, r.end\_time, r.status

FROM reservations r

JOIN seats s ON r.seat\_id = s.seat\_id

WHERE r.user\_id = 2;

* **PATCH /reservations/:id**
  + **Update the time or status of a reservation.**
  + **Input:**

**{**

**"start\_time": "2024-12-22T10:00:00Z",**

**"end\_time": "2024-12-22T12:00:00Z",**

**"status": "Active"**

**}**

* **Output:**

**{**

**"message": "Reservation updated successfully."**

**}**

UPDATE reservations

SET

    start\_time = COALESCE('2025-01-02 9:30', start\_time),

    end\_time = COALESCE('2025-01-02 12:30', end\_time),

    status = COALESCE('Cancelled', status)

WHERE reservation\_id = 3;

* **DELETE /reservations/:id**
  + **Cancel a reservation.**
  + **Output:**

**{**

**"message": "Reservation canceled successfully."**

**}**

UPDATE reservations

SET status = 'Cancelled'

WHERE reservation\_id = 4;

# 

# Appendix C – White box Testing

For the White box testing all the code has been (as listed below) has been checked (and cross checked by Peers)

## Institutions Routes

const express = require('express');

const router = express.Router();

const fs = require('fs');

const path = require('path');

const multer = require('multer');

const requireAuth = require('../middleware/require\_auth');

const ensureCorrectAdmin = require('../middleware/ensure\_correct\_admin');

// Configure storage location for uploaded files

const publicPath = path.join(\_\_dirname, "..", "public");

const upload = multer({

storage: multer.diskStorage({

destination: (req, file, cb) => cb(null, path.join(publicPath, "space\_layouts")), // Save files in `public/space\_layouts/`

filename: (req, file, cb) => cb(null, `layout\_institution\_${req.params.id}\_space\_${req.params.spaceId}\_${Date.now()}${path.extname(file.originalname)}`) // Rename file to `layout\_space\_<spaceId>\_<date>.png`

})

});

function institutionsRouter(db) {

// ------ Institutions Routes ------

// Route to create a new institution

router.post('/register', requireAuth, function (req, res, next) {

const { institution\_name, bio, address, opening\_hours, admin\_id } = req.body;

// Validate the input (e.g., check if admin\_id exists and belongs to an admin role)

db.get(

'SELECT role FROM users WHERE user\_id = ?',

[admin\_id],

function (err, row) {

if (err) return next(err);

// Check if the user exists and is an admin

if (!row || row.role !== 'admin') {

return res.status(400).json({ message: 'admin\_id must belong to a user with role=admin.' });

}

// Insert the new institution

db.run(

'INSERT INTO institutions (institution\_name, bio, address, opening\_hours, admin\_id) VALUES (?, ?, ?, ?, ?)',

[institution\_name, bio, address, opening\_hours, admin\_id],

function (err) {

if (err) return next(err);

const institution\_id = this.lastID;

console.log(`Created institution in database: id=${institution\_id}, name=${institution\_name}.`);

// Attach the institution\_id to the admin user

db.run(

'UPDATE users SET institution\_id = ? WHERE user\_id = ?',

[institution\_id, admin\_id],

function (err) {

if (err) {

console.error('Database error:', err);

return res.status(500).json({ error: 'Failed to link institution to admin user' });

}

console.log(`Linked institution\_id=${institution\_id} to admin user\_id=${admin\_id}.`);

// Respond with the new institution ID

res.json({

institution\_id: institution\_id,

message: 'Institution registered successfully.',

});

}

);

}

);

}

);

});

//Route to retrieve details of all institutions

router.get('', requireAuth, function (req, res, next) {

db.all(

'SELECT institution\_id, institution\_name, bio, address, opening\_hours FROM institutions',

[],

function (err, rows) {

if (err) return next(err);

res.render('institutionSelection', { institutions: rows });

}

);

});

//Route to retrieve details of a specific institution

router.get('/:id', requireAuth, ensureCorrectAdmin, function (req, res, next) {

const institutionId = req.params.id;

db.get(

`SELECT institution\_id, institution\_name, bio, address, opening\_hours FROM institutions

WHERE institution\_id = ?;`,

[institutionId],

function (err, row) {

if (err) return next(err);

if (!row) return res.status(404).json({ message: 'Institution not found.' });

if (req.session.user.role === 'admin') {

res.render('institutionManagement', { institutionId, institution: row });

}

}

);

});

//Route to update an institution's details

router.patch('/:id', requireAuth, function (req, res, next) {

const { institution\_name, bio, address, opening\_hours } = req.body;

const institutionId = req.params.id;

db.run(

`UPDATE institutions SET

institution\_name = COALESCE(?, institution\_name),

bio = COALESCE(?, bio),

address = COALESCE(?, address),

opening\_hours = COALESCE(?, opening\_hours)

WHERE institution\_id = ?`,

[institution\_name, bio, address, opening\_hours, institutionId],

function (err) {

if (err) return next(err);

res.json({ message: 'Institution details updated successfully.' });

}

);

});

//Route to delete an institution

router.delete('/:id', requireAuth, function (req, res, next) {

const institutionId = req.params.id;

db.run(

'DELETE FROM institutions WHERE institution\_id = ?',

[institutionId],

function (err) {

if (err) return next(err);

db.run(

'DELETE FROM users where institution\_id = ?',

[institutionId],

function (err) {

if (err) return next(err);

return res.json({ success: true, redirect: '/logout' });

}

)

}

);

});

// ------ Space Management Routes ------

//Route to create a space for a specific institution

router.post('/:id/spaces', requireAuth, function (req, res, next) {

const { name, layoutImage } = req.body;

const institutionId = req.params.id;

db.run(

'INSERT INTO spaces (space\_name, layout\_image, institution\_id) VALUES (?, ?, ?)',

[name, layoutImage, institutionId],

function (err) {

if (err) return next(err);

const spaceId = this.lastID;

const message = `Space ${spaceId} - "${name}" created successfully in institution ${institutionId}.`;

console.log(message);

res.json({ space\_id: spaceId, message: message });

}

);

});

// Route to retrieve all spaces for a specific institution

router.get('/:id/spaces', requireAuth, ensureCorrectAdmin, function (req, res, next) {

const institutionId = req.params.id;

db.all(

'SELECT space\_id, space\_name, layout\_image FROM spaces WHERE institution\_id = ?',

[institutionId],

function (err, rows) {

if (err) return next(err);

if (req.session.user.role === 'admin') {

res.render('spaceManagement', { institutionId, spaces: rows});

} else {

res.render('spaceSelection', { spaces: rows });

}

}

);

});

// Route to retrieve a specific space for a specific institution

router.get('/:id/spaces/:spaceId', requireAuth, ensureCorrectAdmin, function (req, res, next) {

const { id: institutionId, spaceId } = req.params;

db.get(

'SELECT \* FROM spaces WHERE institution\_id = ? AND space\_id = ?;',

[institutionId, spaceId],

function (err, space) {

if (err) return next(err);

if (!space) return res.status(404).json({ error: "Space not found" });

// Check if the image file exists

const imagePath = space.layout\_image ? path.join(publicPath, space.layout\_image) : null;

if (!imagePath || !fs.existsSync(imagePath)) {

console.warn(`Missing image file: ${imagePath}`);

space.layout\_image = null; // Remove broken image reference from result space

// Update DB to remove broken image reference

db.run(

`UPDATE spaces SET layout\_image = NULL WHERE institution\_id = ? AND space\_id = ?`,

[institutionId, spaceId],

(updateErr) => {

if (updateErr) console.error("Failed to update DB:", updateErr);

}

);

}

// Fetch seats of the space

db.all('SELECT \* FROM seats WHERE space\_id = ?', [spaceId], (seatErr, seats) => {

if (seatErr) return next(seatErr);

if (req.session.user.role === 'admin') {

res.render('seatManagement', { institutionId, space, seats });

} else {

res.render('seatSelection', { space, seats });

}

});

}

);

});

//Route to update details of a specific space

router.patch('/:id/spaces/:spaceId', requireAuth, upload.single('layoutImage'), function (req, res, next) {

const { spaceName } = req.body || null;

const { id: institutionId, spaceId } = req.params;

// Store the layout\_image path in DB as a relative path to have it as a web-accessible path (and a path compatible to any OS)

const newLayoutImage = req.file ? `/space\_layouts/${req.file.filename}` : null;

// Get the current image path before updating

db.get(

'SELECT layout\_image FROM spaces WHERE institution\_id = ? AND space\_id = ?',

[institutionId, spaceId],

function (err, row) {

if (err) return next(err);

const oldImagePath = row ? row.layout\_image : null;

// Update the database with the new image path

db.run(

`UPDATE spaces SET

space\_name = COALESCE(?, space\_name),

layout\_image = COALESCE(?, layout\_image)

WHERE institution\_id = ? AND space\_id = ?`,

[spaceName, newLayoutImage, institutionId, spaceId],

function (updateErr) {

if (updateErr) return next(updateErr);

// Delete the old image if the new one was successfully updated in DB

if (oldImagePath && newLayoutImage) {

const oldFilePath = path.join(publicPath, oldImagePath);

if (fs.existsSync(oldFilePath)) {

fs.unlink(oldFilePath, (unlinkErr) => {

if (unlinkErr) console.error("Error deleting old image:", unlinkErr);

else console.log(`Deleted old image: ${oldFilePath}`);

});

}

}

const finalSpaceName = spaceName || row.space\_name;

const finalLayoutImage = newLayoutImage || oldImagePath;

const message = `Space ${spaceId} details updated successfully: space\_name=${finalSpaceName}, layout\_image=${finalLayoutImage}.`;

console.log(message);

res.json({ message: message, imagePath: finalLayoutImage });

}

);

}

);

});

//Route to delete a space

router.delete('/:id/spaces/:spaceId', requireAuth, function (req, res, next) {

const { id: institutionId, spaceId } = req.params;

db.get(

'SELECT layout\_image FROM spaces WHERE institution\_id = ? AND space\_id = ?;',

[institutionId, spaceId],

function (err, row) {

if (err) return next(err);

if (!row) return res.status(404).json({ error: "Space not found" });

// Delete the file from disk if it exists

if (row.layout\_image) {

const filePath = path.join(publicPath, row.layout\_image);

if (fs.existsSync(filePath)) {

fs.unlinkSync(filePath); // Delete the file

console.log(`Deleted file: ${filePath}`);

}

}

// Delete the entry from the database

db.run(

'DELETE FROM spaces WHERE institution\_id = ? AND space\_id = ?',

[institutionId, spaceId],

function (err) {

if (err) return next(err);

res.json({ message: 'Space deleted successfully.' });

}

);

}

);

});

// ------ Seat Management Routes ------

// Route to get all seats within a space

router.get('/:id/spaces/:spaceId/seats', requireAuth, ensureCorrectAdmin, function (req, res, next) {

const { id: institutionId, spaceId } = req.params;

db.all(

'SELECT seat\_id, space\_id, seat\_name, type, facilities, status FROM seats WHERE space\_id = ?',

[spaceId],

function (err, rows) {

if (err) return next(err);

if (req.session.user.role === 'admin') {

res.render('seatManagement', { institutionId, seats: rows });

} else {

res.render('seatSelection', { seats: rows });

}

}

);

});

// Route to get all available seats within a space

router.get('/:id/spaces/:spaceId/available-seats', requireAuth, (req, res) => {

const { spaceId } = req.params;

const { start\_time, end\_time } = req.query;

const query = `

SELECT s.seat\_id, s.seat\_name, s.type, s.facilities

FROM seats s

WHERE s.space\_id = ?

AND s.status = 'available'

AND s.seat\_id NOT IN (

SELECT seat\_id

FROM reservations

WHERE status = 'active'

AND (

(start\_time <= ? AND end\_time >= ?)

OR (start\_time <= ? AND end\_time >= ?)

OR (start\_time <= ? AND end\_time >= ?)

OR (start\_time >= ? AND end\_time <= ?)

)

)

`;

db.all(query, [

spaceId,

start\_time, start\_time, // Overlap at start

end\_time, end\_time, // Overlap at end

start\_time, end\_time, // Contains

start\_time, end\_time // Is contained within

], (err, availableSeats) => {

if (err) return res.status(500).json({ error: 'Database error' });

res.json(availableSeats);

});

});

//Route to add a seat to a space with an institution

router.post('/:id/spaces/:spaceId/seats', requireAuth, function (req, res, next) {

const { seat\_name, type, facilities, status } = req.body;

const spaceId = req.params.spaceId;

if (!seat\_name || !type || !status) {

return res.status(400).json({ error: 'Missing required seat fields.' });

}

db.run(

'INSERT INTO seats (space\_id, seat\_name, type, facilities, status) VALUES (?, ?, ?, ?, ?)',

[spaceId, seat\_name, type, facilities, status],

function (err) {

if (err) return next(err);

const seatId = this.lastID;

const message = `Seat ${seatId} added successfully to space ${spaceId}.`;

console.log(message);

res.json({ seat\_id: seatId, message: message });

}

);

});

//Route to update details of a specific seat

router.patch('/:id/spaces/:spaceId/seats/:seatId', requireAuth, function (req, res, next) {

const { seat\_name, type, facilities, status } = req.body;

const { spaceId, seatId } = req.params;

if (!seatId) {

return res.status(400).json({ error: 'Seat ID is required.' });

}

db.run(

`UPDATE seats SET

seat\_name = COALESCE(?, seat\_name),

type = COALESCE(?, type),

facilities = COALESCE(?, facilities),

status = COALESCE(?, status)

WHERE seat\_id = ? AND space\_id = ?`,

[seat\_name, type, facilities, status, seatId, spaceId],

function (err) {

if (err) return next(err);

res.json({ message:`Seat ${seatId} details updated successfully (space ${spaceId}).` });

}

);

});

//Route to remove a seat

router.delete('/:id/spaces/:spaceId/seats/:seatId', requireAuth, function (req, res, next) {

const { spaceId, seatId } = req.params;

db.run(

'DELETE FROM seats WHERE seat\_id = ? AND space\_id = ?',

[seatId, spaceId],

function (err) {

if (err) return next(err);

res.json({ message: 'Seat deleted successfully.' });

}

);

});

// Route to clear all seats within a space

router.delete('/:id/spaces/:spaceId/seats', requireAuth, function (req, res, next) {

const { spaceId } = req.params;

db.get('SELECT COUNT(\*) as count FROM seats WHERE space\_id = ?', [spaceId], (err, result) => {

if (err) return next(err);

if (result.count === 0) {

const message = 'No seats found to delete.';

console.warn(message);

return res.status(404).json({ error: message });

}

db.run('DELETE FROM seats WHERE space\_id = ?', [spaceId], function (err) {

if (err) return next(err);

const message = `${result.count} seats cleared successfully from space ${spaceId}.`;

console.log(message);

res.json({ message: message });

});

});

});

return router;

};

module.exports = institutionsRouter;

## Auth Route

const express = require('express');

const router = express.Router();

const bcrypt = require('bcrypt');

const requireAuth = require('../middleware/require\_auth');

const saltRounds = 10; // For password hashing

function authRouter(db) {

/\*\*

\* Route to display the login page.

\* Inputs: None

\* Outputs: Renders the 'login' view with an error message if any

\*/

router.get('/login', (req, res) => {

const errorMessage = req.session.errorMessage;

req.session.errorMessage = null; // Clear error message after displaying

res.render('login', { errorMessage });

});

/\*\*

\* Route to handle login form submission.

\* Inputs: req.body.username, req.body.password

\* Outputs: Redirects to the home page if credentials are valid, otherwise re-renders the login page with an error message

\*/

router.post('/login', async (req, res) => {

if (req.session.user) {

return res.status(400).json({ error: "User is already logged in." });

}

const { username, password } = req.body;

try {

// Fetch user from SQLite database using parameterized query

db.get(

`SELECT user\_id, username, role, password\_hash FROM users WHERE username = ?`,

[username],

async (err, user) => {

if (err) {

console.error('Database error:', err);

return res.status(500).json({ error: 'Internal server error' });

}

// If no user found

if (!user) {

return res.status(401).json({ error: 'User does not exist' });

}

// Compare entered password with stored hashed password

const validPassword = await bcrypt.compare(password, user.password\_hash);

if (!validPassword) {

return res.status(401).json({ error: 'Incorrect password' });

}

// Store user data in session (excluding password hash)

req.session.user = {

id: user.user\_id,

username: user.username,

role: user.role

};

// Store institution ID in session if user is admin - assume institution was creating when user registered

if (user.role === 'admin') {

db.get(

'SELECT institution\_id FROM institutions WHERE admin\_id = ?',

[user.user\_id],

(err, institution) => {

if (err) {

console.error('Database error:', err);

return res.status(500).json({ error: 'Internal server error' });

}

req.session.user.institutionId = institution ? institution.institution\_id : null;

// Ensure this is called after the institutionId is set

req.session.successMessage = 'Login successful! Welcome back.';

return res.json({ success: true, redirect: '/home' });

}

);

} else { // For students, respond immediately (no institution needed)

// Set success message in session

req.session.successMessage = 'Login successful! Welcome back.';

return res.json({ success: true, redirect: '/home' });

}

}

);

} catch (error) {

console.error('Login error:', error);

return res.status(500).json({ error: 'Internal server error' });

}

});

/\*\*

\* Route to display the registration page.

\* Inputs: None

\* Outputs: Renders the 'register' view with an error message if any

\*/

router.get('/register', (req, res) => {

res.render('register');

});

/\*\*

\* Route to handle registration form submission.

\* Inputs: req.body.username, req.body.password, req.body.role

\* Outputs: Redirects to the welcome page after successful registration, otherwise re-renders the registration page with an error message

\*/

router.post('/register', async (req, res) => {

const { username, password, role, institution\_name, bio, address, opening\_hours } = req.body;

try {

if (!username || !password || !role) {

return res.status(400).json({ error: 'All fields are required' });

}

// Check if the username already exists

db.get(`SELECT user\_id FROM users WHERE username = ?`, [username], async (err, user) => {

if (err) {

console.error('Database error:', err);

return res.status(500).json({ error: 'Internal server error' });

}

if (user) {

return res.status(400).json({ error: 'Username already exists' });

}

// Hash the password

const hashedPassword = await bcrypt.hash(password, saltRounds);

// Insert the new user into the database

db.run(

`INSERT INTO users (username, password\_hash, role) VALUES (?, ?, ?)`,

[username, hashedPassword, role],

function (err) {

if (err) {

console.error('Database error:', err);

return res.status(500).json({ error: 'Failed to register user' });

}

console.log(`Created user in database: id=${this.lastID}, username=${username}, role=${role}.`);

// If user is an admin, also create their institution

if (role === 'admin') {

const adminId = this.lastID;

db.run(

'INSERT INTO institutions (institution\_name, bio, address, opening\_hours, admin\_id) VALUES (?, ?, ?, ?, ?)',

[institution\_name, bio, address, opening\_hours, adminId],

function (err) {

if (err) {

console.error('Database error:', err);

return res.status(500).json({ error: 'Failed to create institution' });

}

const institutionId = this.lastID;

console.log(`Created institution in database: id=${institutionId}, name=${institution\_name}.`);

// Attach the institution\_id to the admin user

db.run(

'UPDATE users SET institution\_id = ? WHERE user\_id = ?',

[institutionId, adminId],

function (err) {

if (err) {

console.error('Database error:', err);

return res.status(500).json({ error: 'Failed to link institution to admin user' });

}

console.log(`Linked institution\_id=${institutionId} to admin user\_id=${adminId}.`);

req.session.successMessage = 'Registration successful! You can now log in.';

return res.json({ success: true, redirect: '/'}); // Redirect to the welcome page after successful registration

}

);

}

);

} else {

// Set success message in session

req.session.successMessage = 'Registration successful! You can now log in.';

return res.json({ success: true, redirect: '/'}); // Redirect to the welcome page after successful registration

}

}

);

});

} catch (error) {

console.error('Registration error:', error);

return res.status(500).json({ error: 'Internal server error' });

}

});

/\*\*

\* Route to display the main home page after login.

\* Inputs: None

\* Outputs: Renders the home views

\*/

router.get('/home', requireAuth, (req, res) => {

const successMessage = req.session.successMessage;

req.session.successMessage = null; // Clear message after displaying

// Render the relevant home page with a login success message if exists

if (req.session.user.role === 'admin') {

res.render('adminHome', { successMessage, institutionId: req.session.user.institutionId });

} else {

res.render('studentHome', { successMessage });

}

});

/\*\*

\* Route to handle logout.

\* Inputs: None

\* Outputs: Destroys the session and redirects to the login page

\*/

router.post('/logout', requireAuth, (req, res) => {

req.session.destroy();

res.redirect('/');

});

return router;

}

module.exports = authRouter;

## Reservations Route

const express = require('express');

const router = express.Router();

const requireAuth = require('../middleware/require\_auth');

function reservationsRouter(db) {

// ------ Reservation Management Routes ------

// Route to get all reservations

router.get('/', requireAuth, function (req, res, next) {

db.all(

'SELECT reservation\_id, user\_id, seat\_id, start\_time, end\_time, status FROM reservations',

[],

function (err, rows) {

if (err) return next(err);

res.render('reservationManagement', { institutionId: req.session.user.institutionId, reservations: rows });

}

);

});

//Route to create a reservation for a seat

router.post('/', requireAuth, function (req, res, next) {

const { seat\_id, start\_time, end\_time } = req.body;

const user\_id = req.session.user.id;

db.run(

'INSERT INTO reservations (seat\_id, user\_id, start\_time, end\_time, status) VALUES (?, ?, ?, ?, ?)',

[seat\_id, user\_id, start\_time, end\_time, 'active'],

function (err) {

if (err) return next(err);

res.json({ reservation\_id: this.lastID, message: 'Reservation created successfully.' });

}

);

});

//Route to retrieve all reservations for a specific user

router.get('/:user\_id', requireAuth, function (req, res, next) {

const { user\_id } = req.params;

db.all(

`SELECT reservation\_id, seat\_id, start\_time, end\_time, status

FROM reservations

WHERE user\_id = ?`,

[user\_id],

function (err, rows) {

if (err) return next(err);

res.render('reservationSelection', { reservations: rows });

}

);

});

//Route to update a reservation

router.patch('/:id', requireAuth, function (req, res, next) {

const { start\_time, end\_time, status } = req.body;

const { id } = req.params;

db.run(

'UPDATE reservations SET start\_time = COALESCE(?, start\_time), end\_time = COALESCE(?, end\_time), status = COALESCE(?, status) WHERE reservation\_id = ?',

[start\_time, end\_time, status, id],

function (err) {

if (err) return next(err);

res.json({ message: 'Reservation updated successfully.' });

}

);

});

//Route to cancel a reservation

router.delete('/:id', requireAuth, function (req, res, next) {

const { id } = req.params;

db.run(

'UPDATE reservations SET status = "cancelled" WHERE reservation\_id = ?',

[id],

function (err) {

if (err) return next(err);

res.json({ message: 'Reservation canceled successfully.' });

}

);

});

return router;

};

module.exports = reservationsRouter;

## Seat Management Route

const express = require('express');

const router = express.Router();

//implement routes

// ------ Authentication Routes ------

//Routes to create new Authentication

router.post(' /institutions/', ensureAuthor, function (req, res, next) {

const {space\_id,name,type,facilities,status} = req.body;

global.db.run(

(' INSERT INTO seats (space\_id, name, type, facilities, status) VALUES (?,?,?,?,?) ',

[pace\_id, name, type, facilities, status],

function(err) {

if(err) return next(err);

})

});

//Route to retrieve a specific Authentication

router.get('/institutions/', ensureAuthor, function (req, res, next) {

global.db.all(

' UPDATE seats SET type = '?', facilities = ?, status = '?' WHERE space\_id = ? AND seat\_id = ? '

[type,facilities,status, space\_id,seat\_id ],

function(err,rows){

if(err) return next(err);

res.jason(rows);

}

);

});

## Authentication Route

const express = require('express');

const router = express.Router();

//implement routes

// ------ Authentication Routes ------

//Routes to create new Authentication

router.post(' /auth/register', ensureAuthor, function (req, res, next) {

const {username,password\_hash,role} = req.body;

global.db.run(

(' INSERT INTO users (username, password\_hash, role) VALUES ( ?, ? ,? )',

[username , password\_hash, role],

function(err) {

if(err) return next(err);

})

)

});

//Route to retrieve a specific Authentication

router.get('/auth/login', ensureAuthor, function (req, res, next) {

global.db.all(

' SELECT user\_id, role FROM users',

[user\_id,role],

function(err,rows){

if(err) return next(err);

res.jason(rows);

}

);

});

# Appendix D – Black box Testing

Black box testing is done as a not technical person or a layman using the software.

This usually consists of screen shots of all the screens (admin as well as Student) screens.

## Anatomy of a web page

Title of the screen (dialog)

Canvas Color (R, G, B)

147,244, 246

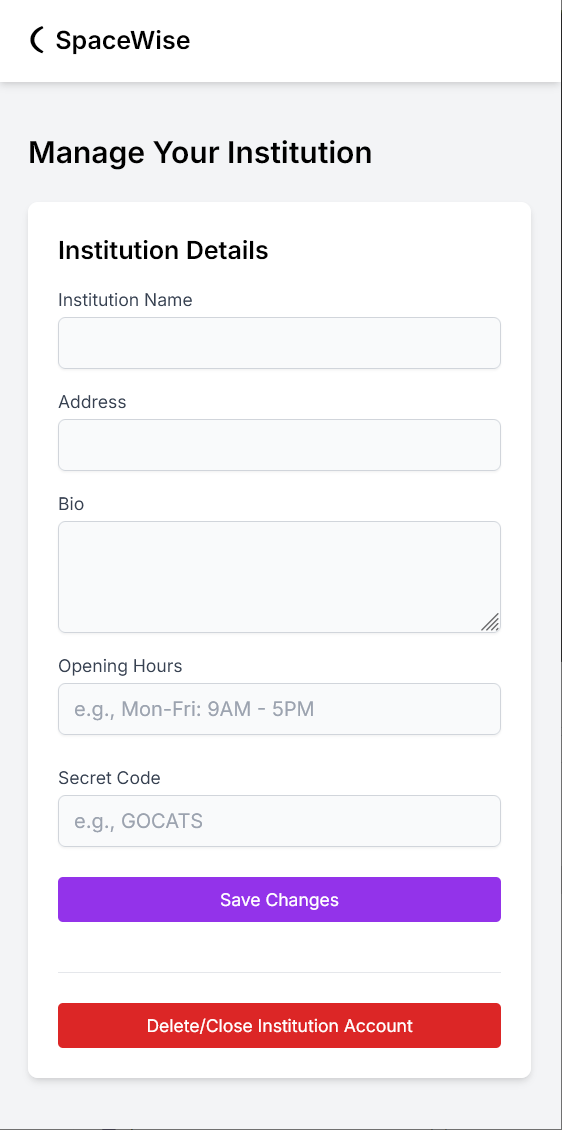


Figure Anatomy of a web page

SpaceWise Logo and Title bar

Cancel button

X axis in pixels

Y axis in pixels

Save button

Single line text input

Multi line text (form) input

Minimum y padding 15 pixels

Minimum X padding 10 pixels

## Student Screens

Student - Register

Register dialog box

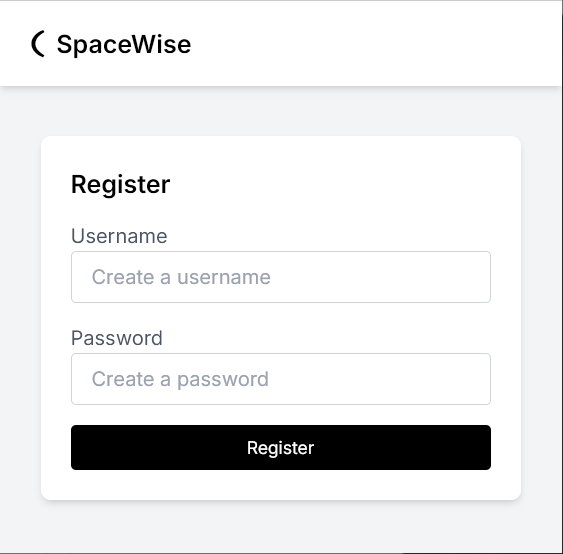


Figure Register Screen

Student – Register:

This screen will be used for student’s registration.

User Name entry

Password entry

Register button

Student - Login

Login dialog box

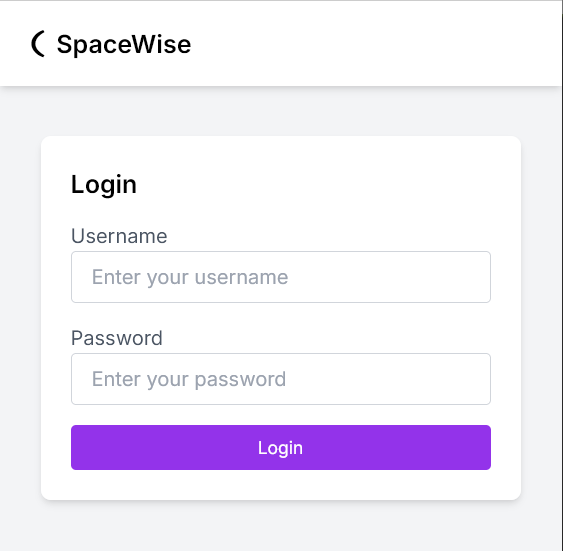


Figure Login Screen

Student – Register:

This screen will be used for student’s Login.

Note: Prior to Login, Student must be Registered.

User Name entry

Password entry

Login button

Student - Home

Student Sign button

Student Sign in

Admin screen button

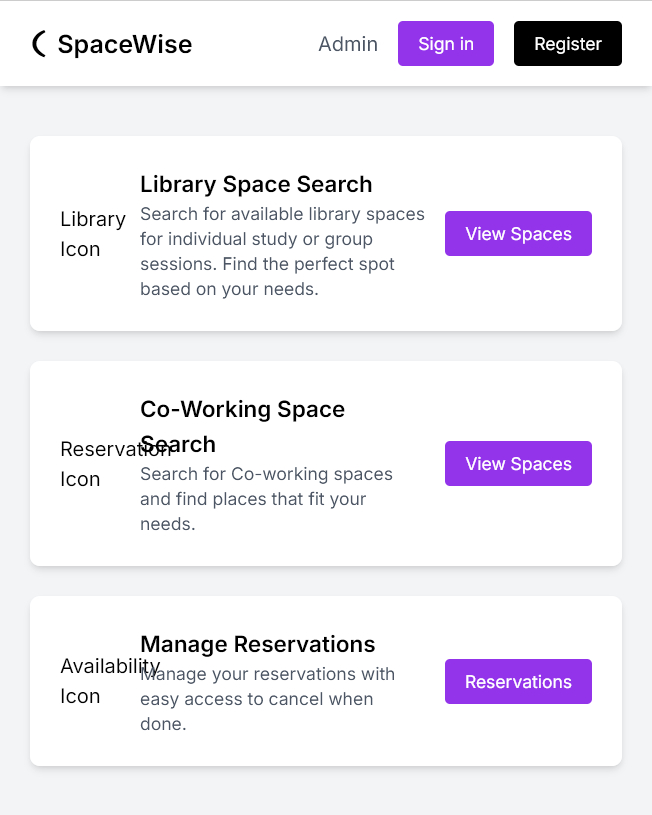


Figure Student Home Screen

Library Space dialog

Library View space button

Co-Working space dialog

Co-working Space View button

Manage reservations dialog

Manage Reservations button

Student – Home:

This screen will be used for

Library Space Search: This button will lead to Different Library Spaces

Co-Working Space : This button will lead to Different Co-Working Space

Manage – Reservations: This button will lead to Manage Reservations screen

Student – Select a Library

Student Register

Student Sign in

Admin screen button

Select a library dialog

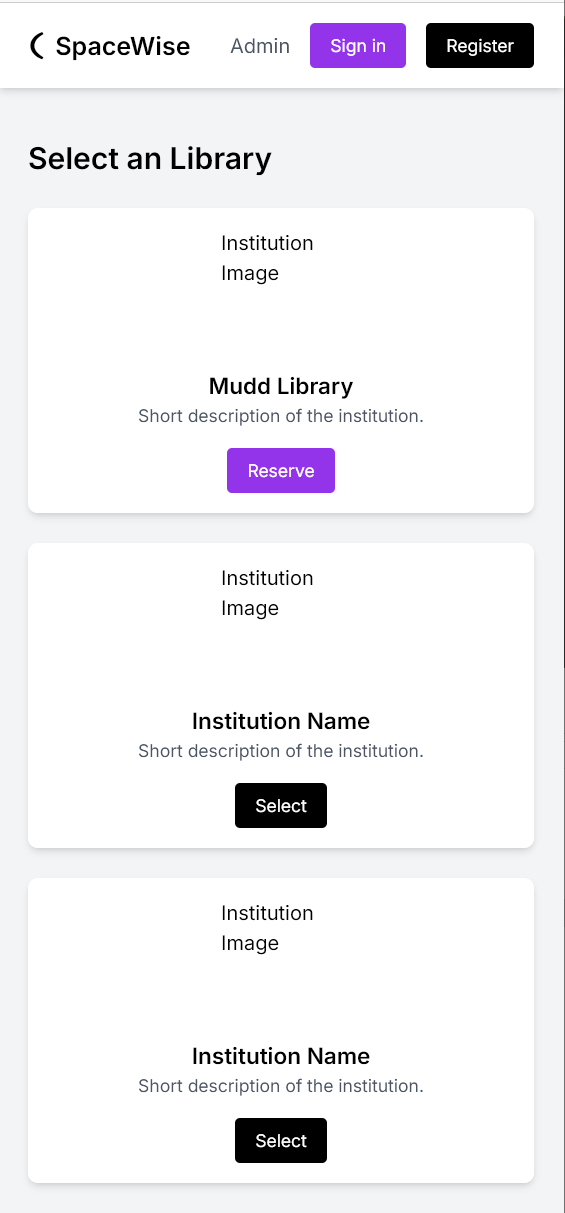


Figure Select a Library Screen

Reserve button

Second library select Dialog

First library select dialog

Select button

Third library select Dialog

select Dialog

Select button

Student – Select Library:

This screen will list all the libraries in the area.

Student will be able to select the preferred library.

This screen also lets user to Sign in as Admin or Register as Admin.

Student – Manage Space Details

Edit Layout Button and Display

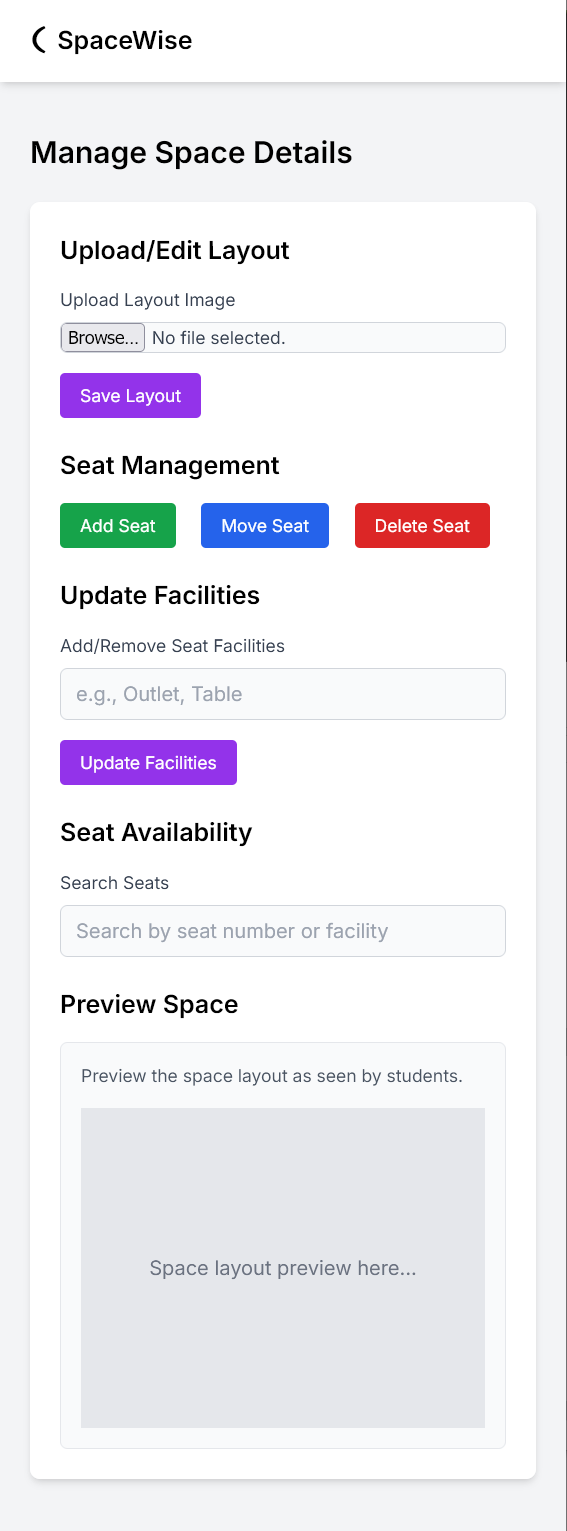


Figure Manage Space Details Screen

Save layout

Add Seat button

Move Seat button

Delete Seat button

Add Facilities Entry

Update Facilities button

Seat Availability display

Preview of space

Student – Manage Space Details:

This screen lets students to Upload or Edit the layout

Seat management section will let students Add Seat, Move Seat or Delete Seat.

Update Facility will let students to update Facility.

Preview will show the current uploaded layout.

Student – Manage Reservations

Manage Reservations dialog

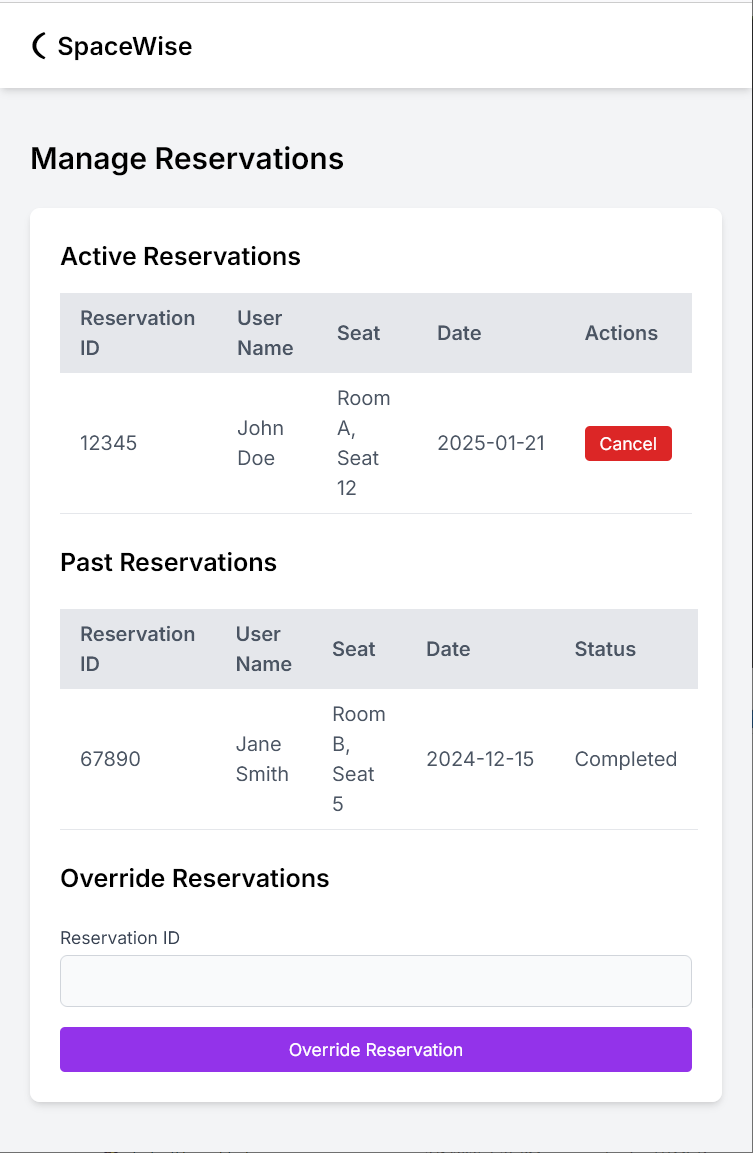


Figure Manage Reservations Screen

Active Reservations dialog

Cancel reservation button

Past Reservations dialog

Reservation ID

Override Reservation button

Student – Manage Reservations:

This screen lets students to manage their reservations.

Students can see their Reservations ID, User name, Room number, Seat, Date, Time.

If students prefer to cancel their Reservation “Cancel” button will cancel the reservation Selected by the dialog box.

Students can also see their past history of Seat Reservation.

Student – My Reservations

Student Sign in

Admin screen button

Student Sign in button

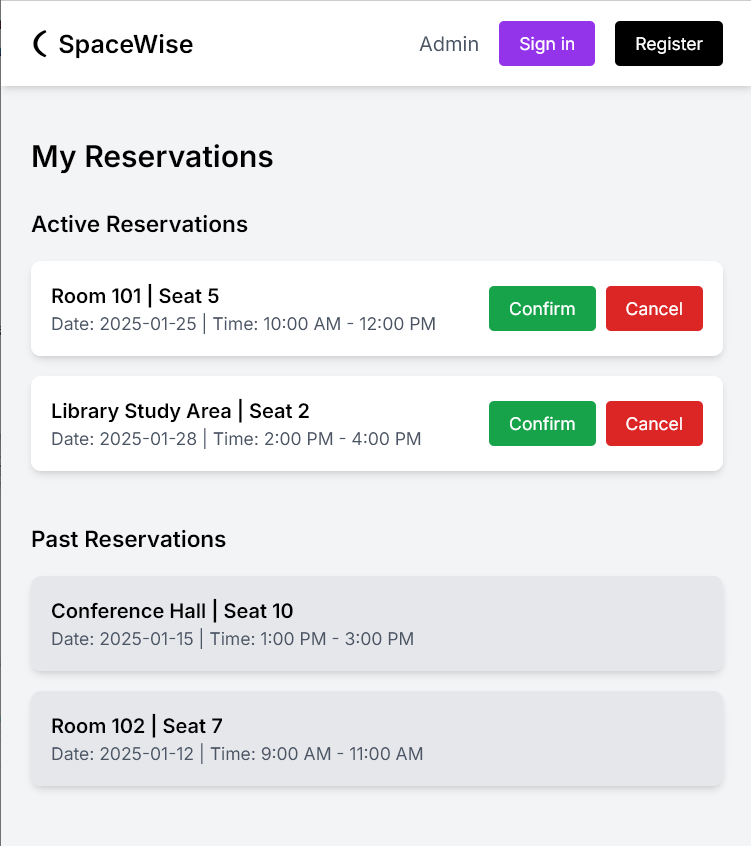


Figure My Reservations Screen

Active Reservations dialog

Confirm Reservation button

Cancel Reservation button

Past Reservations dialog

Student – My Reservations:

This screen lets students to manage their multiple Reservations.

Students can see their Reservations ID, User name, Room number, Seat, Date, Time.

If students prefer to cancel their Reservation “Cancel” button will cancel the reservation Selected by the dialog box.

Students can also see their past history of Seat Reservation.

Student – Select a Space

Student Sign in button

Student Sign in

Admin screen button

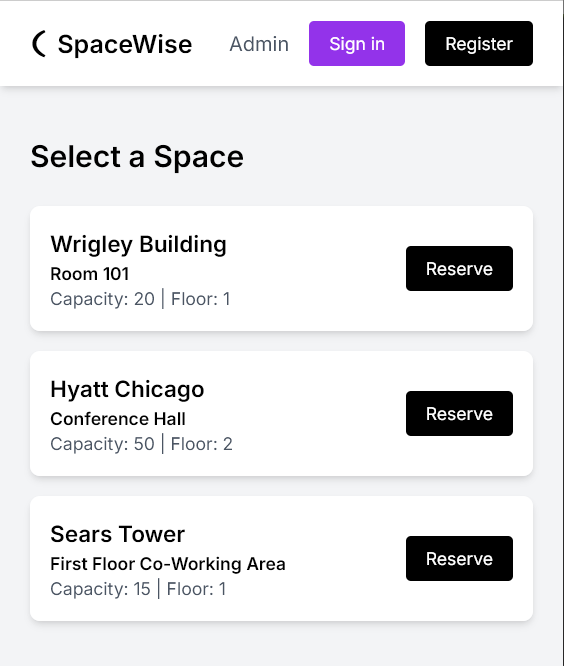


Figure Select a Space Screen

Select a Space dialog

Reserve button

Select Space dialog

Student – Select a Space:

This screen lets students to see multiple places. Students can then go to their respective Spaces by pressing Reserve button on the respective Dialog box.

Student - Select a Seat

Student Register button

Student Sign in

Admin screen button

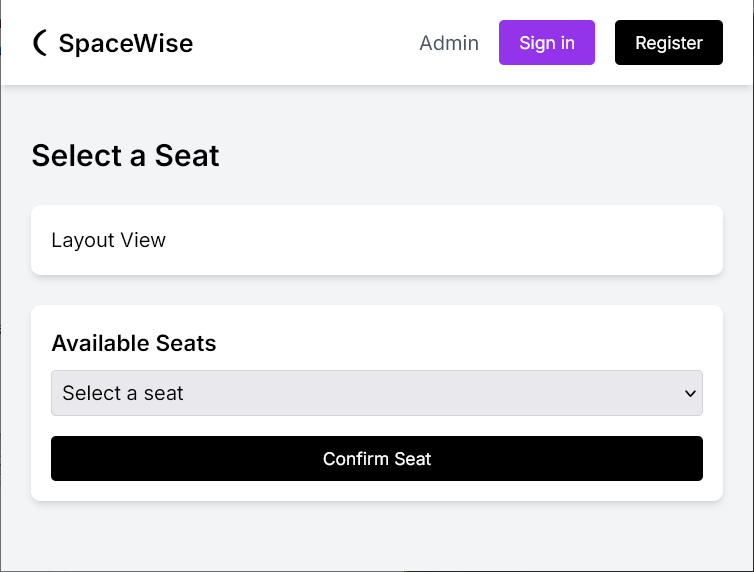


Figure Select a Seat Screen

Layout view

Select Seat pull down list box

Confirm Reservation button

Student – Select a Seat:

When student goes to Select a Seat screen, Student will see the layout of the entire Selected space.

Student will then use the list box which will show the current available seat.

After Selecting the seat Student will press “Conform Seat” button to confirm the Reservation.

## Admin Screens

Admin - Register

Register dialog box

Admin – Register:

This screen will be used for Administrator’s registration.

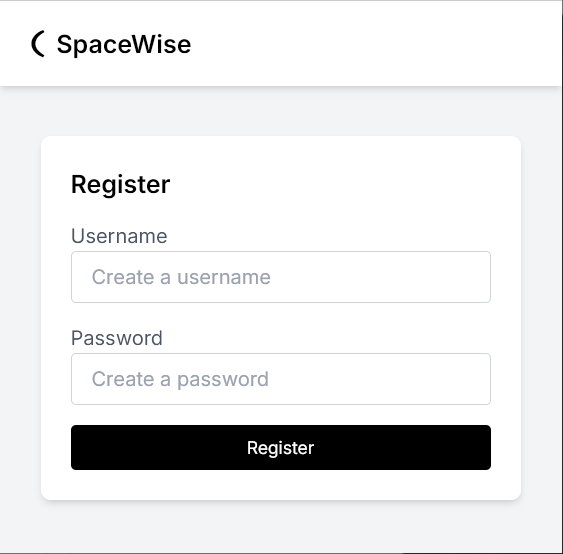


Figure Register Screen

User Name entry

Register button

Password entry

Admin - Login

Admin – Login:

This screen will be used for Administrator’s Login.

Note: Prior to Login, Administrator must be Registered.

Login dialog box

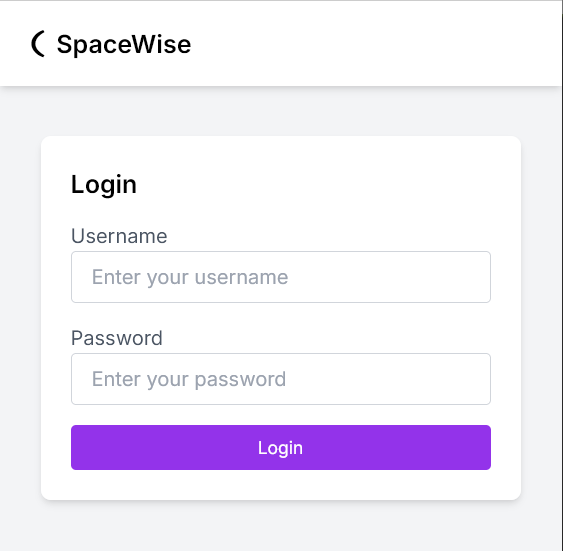


Figure Login Screen

User Name entry

Password entry

Login button

Admin - Dashboard

Admin Dashboard dialog

Student Register button

Student Sign in

Admin screen button

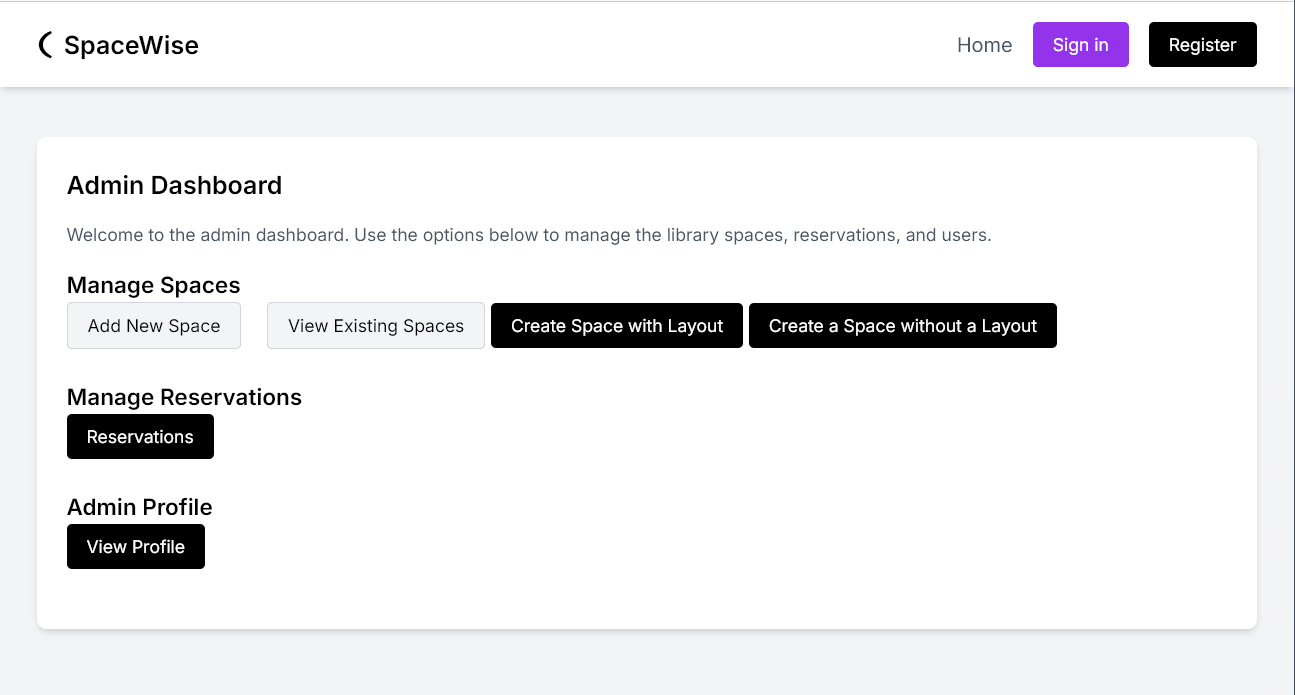


Figure Admin Dashboard

Create a Space without a Layout button

Custom Space with Layout button

View Existing Spaces button

Add New Space button

Admin Profile button

Manage Reservations button

Admin – Dashboard:

In this Dash board Administrator can

- Go to Add New Space screen

- Go to View Existing Spaces screen

- Go to Create Space With Layout

- Go to Create Space Without Layout

- Go to Create Reservations

- Go to View profiles

Admin – Manage Your Institution

Manage Your Institution dialog

Institution Name entry

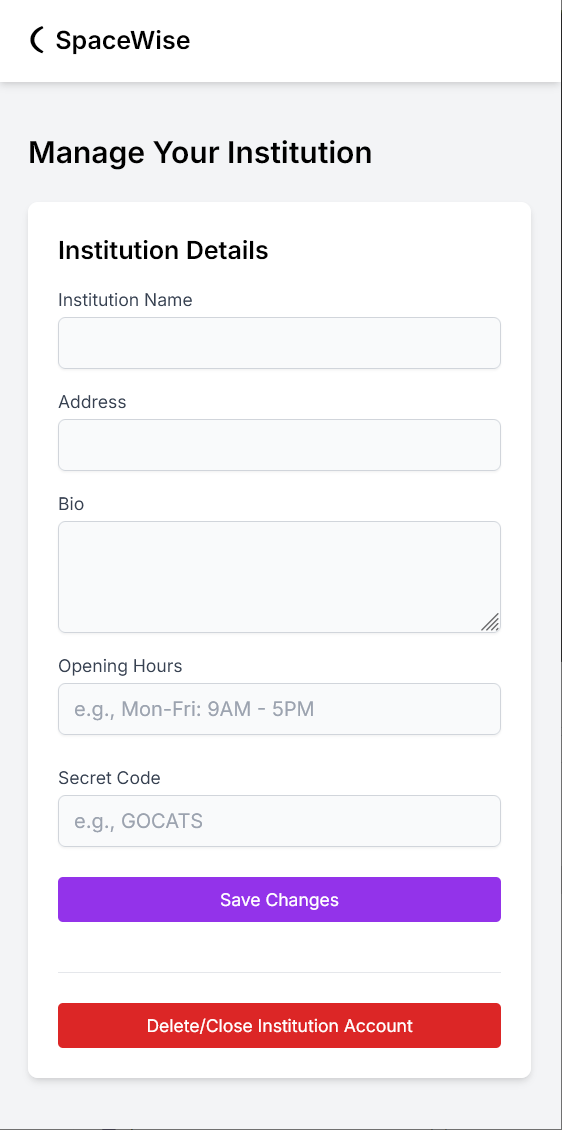


Figure Manage Your Institution

Address entry

Bio details form entry

Opening Hours entry

Secret Code entry

Save Changes button

Delete / Close Institution Account button

Admin –Manage Your Institution:

In this Screen Administrator can

- Enter Institution Name

- Enter Institution Address

- Enter Bio data of the Institution

- Operating Hours

- Secret Code

After Entering the details Administrator can Create the institution by pressing “Save Changes” button.

Or

Administrator can the Delete the Institution by pressing “Delete / Close Institution Account” button.

Admin – Manage Spaces

Manage Spaces dialog

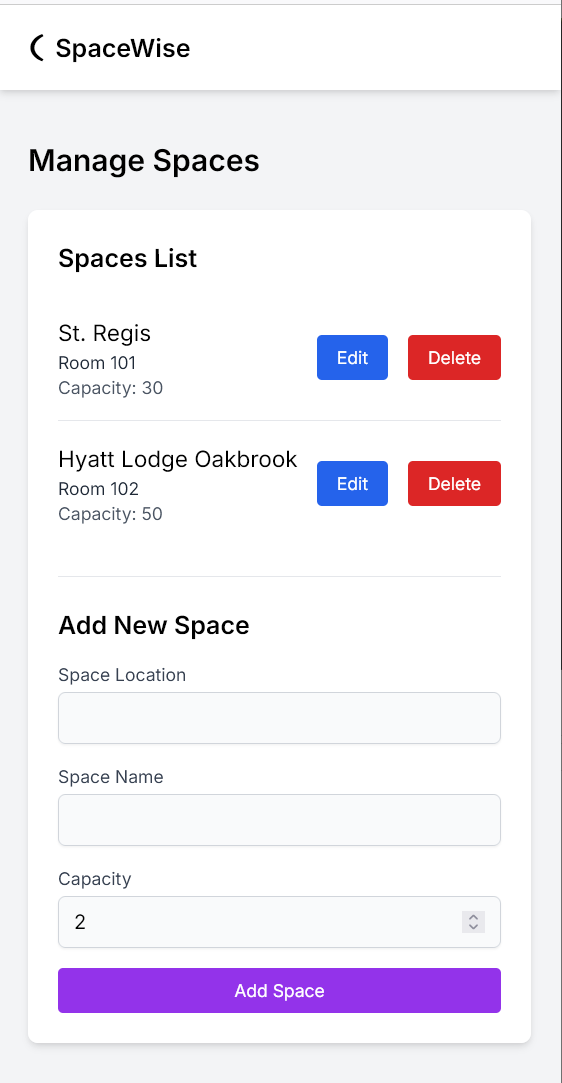


Figure Manage Spaces

Space List Edit button

Space List Delete button

Add New Space dialog

Space Location entry

Space Name entry

Capacity entry

Add Space button

Admin –Manage Spaces:

In this Screen Administrator can Add Space by entering

- Space Location

- Space Name

- Space Capacity

After Entering the details Administrator can Create the Space by pressing “Add Space” button.

Or

Administrator can the Delete the Space by pressing “Delete” button.

Or

Administrator can Edit the Space by pressing “Edit” button.