Agile Software Project

Final Assessment

Team 70

WorkGoWhere

A website to help users to swiftly look for a space for remote work outside of home.

Name	Student Number	Email
Austin Lian Jia Le	200553249	jlalian001@mymail.sim.edu.sg
Chow Su Yee (Tricia)	200618700	sychow005@mymail.sim.edu.sg
Nicol Chen Hao Quan	210201370	nhqchen001@mymail.sim.edu.sg
Winson Chow Wei Xin	200553607	wwxchow001@mymail.sim.edu.sg

Table of Contents

1.0 Background

1.1 Introduction

The aim of our project is to create a quicker and more efficient process when it comes to finding remote workspaces for working adults, bridging the gap of manually going through various online articles and blogs to a few simple steps. By creating our web app, “WorkGoWhere”, that can more specifically narrow down remote work spaces that meet the needs of users, it can significantly improve the ease and efficiency of finding such places at different places and times.

The team comprises working/studying adults that have experienced the importance of having conducive remote workspaces on multiple occasions, which allowed us to come up with and agree on the idea of the project in the beginning. We conducted surveys targeted at our target audience of working and studying adults to obtain insights and feedback regarding the basis of our idea and to confirm an interest.

It can be expected that a large percentage of working adults would utilise our web app as according to our “3.4 Survey outcomes” in our midterm proposal, there is still a significant interest in remote workspaces regardless of if homes are their preferred working environments or not. **58.1%** of our respondents find their home an optimal environment for productivity while **41.9%** considered otherwise. In a deeper dive, we found out that within the productive group, more than half (**58.3%**) would consider a different working space. On the other hand, for the unproductive group, 4 out of 5 (**80.8%**) respondents prefer an alternate working space from their home or office.

Our web app automatically provides results from our database based on user location and amenity specifications, and redirects users to the web pages or contacts of such remote workspaces so that they can make a reservation if desired. This is a more efficient approach as opposed to relying on general search engines such as Google or word of mouth, which may yield results of less consistent quality and relevance.

We adopted an agile and user-centred design for our web app, which places our users front and centre of the core experience. It is paramount that the whole user experience is highly usable and accessible. A user-centred design helps the team capture an explicit understanding of our users, tasks and environment.

As of this point of research, there does not seem to be dedicated web apps for recommending conducive workspaces. Due to the lack of competitors with similar motivations in the market, we focused on studying articles and blogs based on the top search results from search engines as they will likely be our top competitors.

1.2 Project Scope and Deliverables

The goal of this project is to develop a web application that streamlines the process of finding and booking a suitable work space for the user.

The first feature of our application is a streamlined process to search for a working space. After user input of a location in the search bar, the website will retrieve data from the database and display all the available spaces within the vicinity.

An inappropriate search input will simply yield no results. The results will be in rows of cards users can sift through.

The second feature is an extensive filter menu for users to narrow down their choices. Users can apply filters which will only display workspaces that have filtered tags on the results page. This feature helps users to quickly narrow down choices when finding remote workspaces. Not applying filters shows users all available workspace options.

The third feature is a reservation link that brings the user to their preferred workspace website to reserve their seats. Users can click a button on pop-up cards to go to the official sites of a workspace and make a reservation. This removes the need for users to figure out how to navigate sites or use search engines to make a reservation.

The fourth feature is a separate web page for clients to register or login to manage their existing listing. By clicking “List Now” on the account page, clients can create a new account. Users will need to provide necessary details for account creation. Clients with existing accounts will be able to key in their emails and passwords to login to WorkGoWhere and go to their dashboard page.

The fifth feature is the one which allows clients to list a workspace on WorkGoWhere. They will simply fill up a form with information that will be stored in the database. They will also have the ability to view, manage and delete their listings.

2.0 Planning and Research

2.1 Summary of Group

The team members for the development of the web app are Winson, Tricia, Austin and Nicol. The team regularly participates in on-campus and online discussions to regularly update on progress. Different tasks are assigned to each member at every stage depending on strengths and abilities. The team often highlights and sets up short-term deadlines for our goals in order to maintain consistency, as well as having safety nets to ensure our project succeeds.

Each role is assigned based on each member's comfort level with the responsibilities associated, as well as their past work or project experience. This allows the team to function efficiently.

Member In-charge	**Role**	**Role Description**
Winson	Scrum master	Implementing scrum approach with the team
Tricia	Back end head developer	Spearhead and oversees development of all back end components
Austin	Front end head developer	Spearhead and oversees development of all front end components
Nicol	Developer	Assist development team with assigned tasks

Table 1.0 - Role Distribution

2.2 Planning Process

The team has extensively utilised multiple online platforms as project management tools to better aid the team in organisation and teamwork. The following tools are as follows:

Google Drive

Google Drive is a cloud storage and file-sharing system where we could privately store, share and synchronise our project files online. It is also integrated with collaboration apps such as Docs and Sheets where it has significantly enabled us to collaboratively create important documents such as meeting agendas, reports and more.

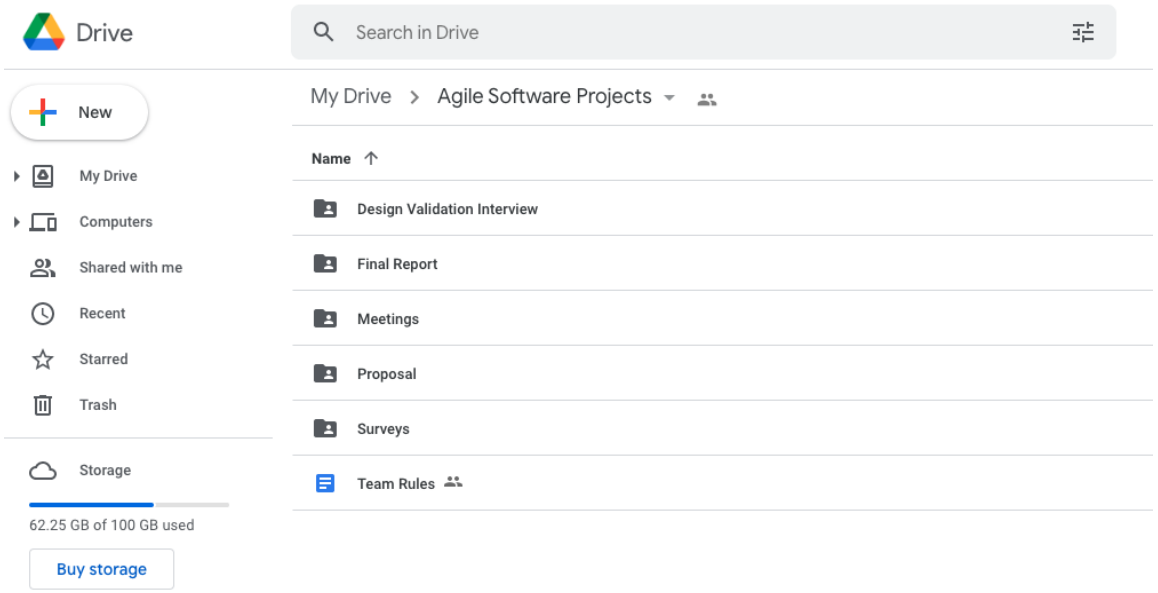


Figure 1.0 - Screenshot of the team's Google Drive

Jira

Jira is a project tracking software. Jira provides scrum and kanban boards which are heavily utilised in the course of the project to manage tasks and achieve our goals.

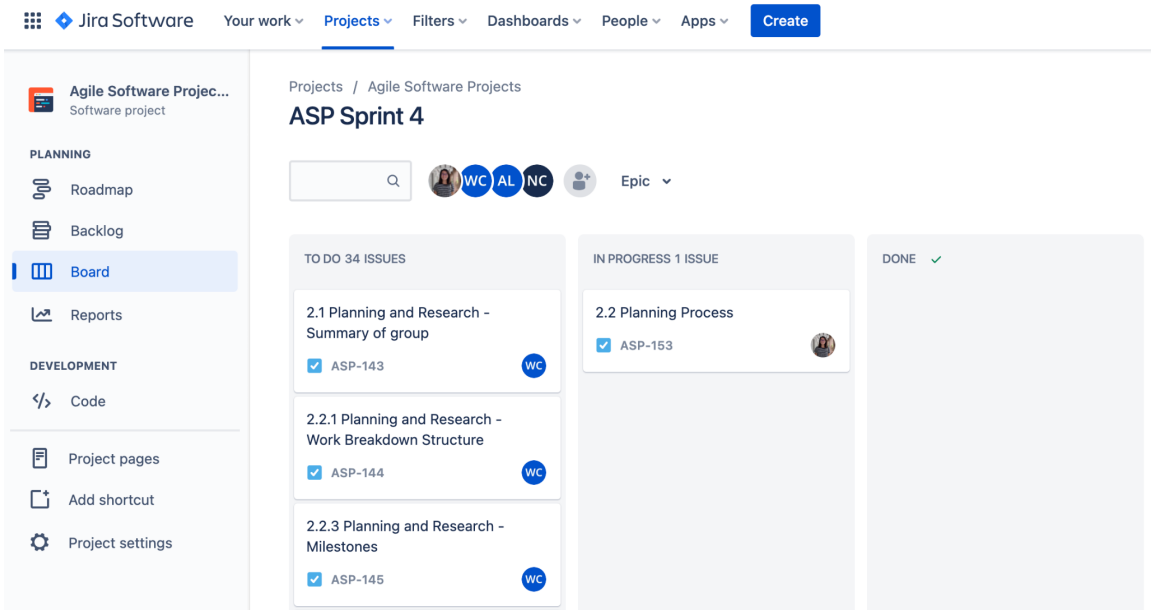


Figure 1.1 - Screenshot of the team's kanban board on Jira

Discord

Discord is a video and audio chat app that we have relied on to conduct our weekly team meetings.

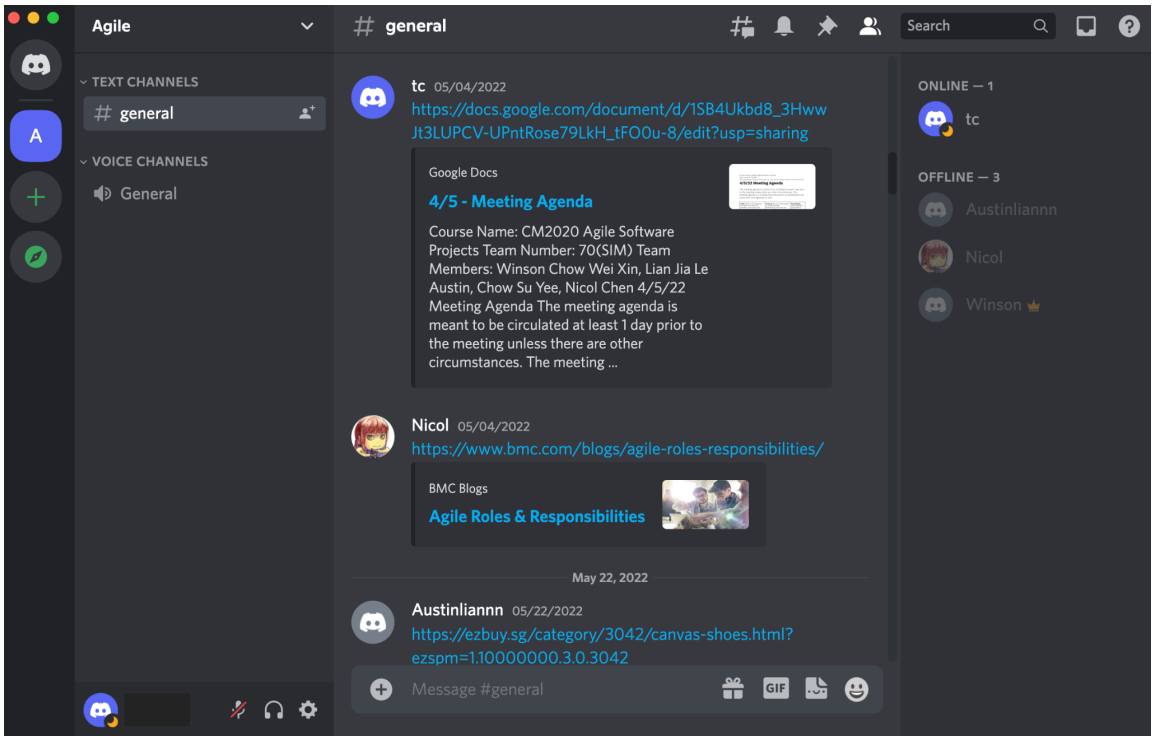


Figure 1.2 - Screenshot of the team's Discord group

GitHub

GitHub is a cloud-based Git repository where we are able to write code in a collaborative manner. It houses two of the team's repositories.

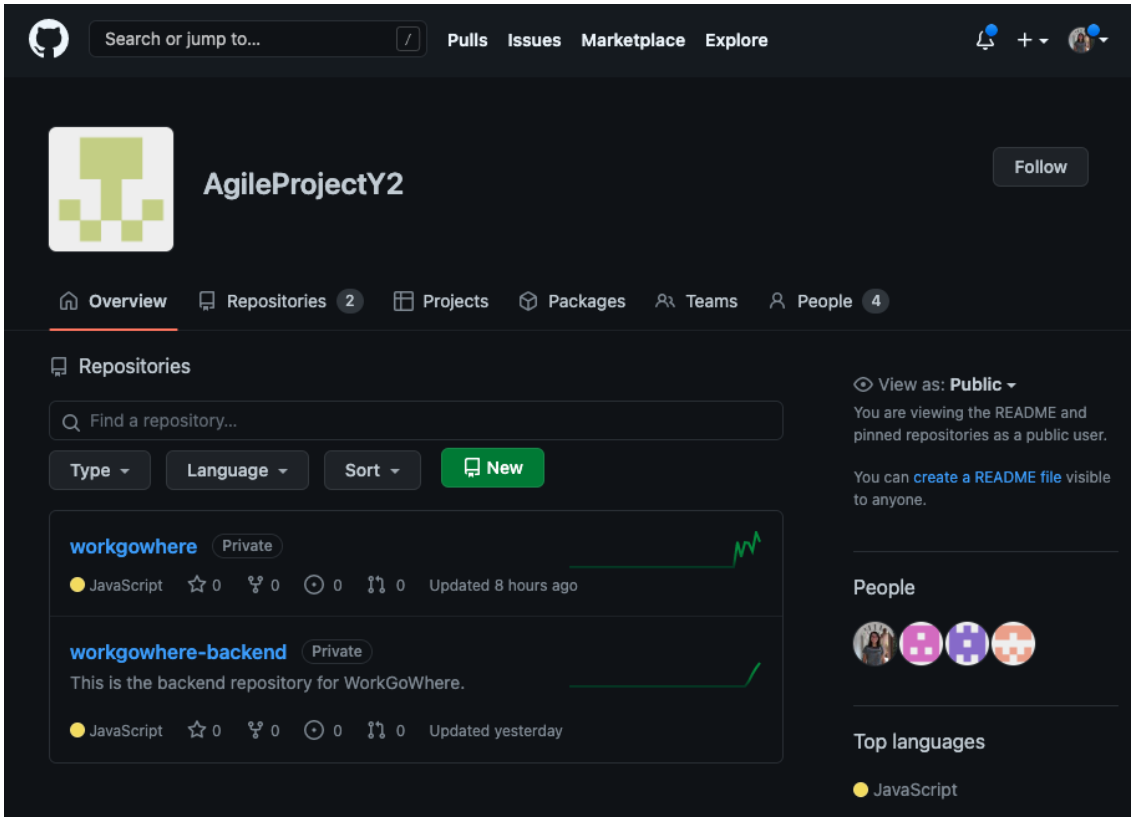


Figure 1.3 - Screenshot of the team's GitHub repository

2.2.1 Work Breakdown Structure (WBS)

The team utilises the scrum framework to break down our work into manageable components. Our development cycle is divided into four sprints, spanning two weeks long. The task to complete in each sprint is planned by the scrum master and development is overseen by our head developers. In the middle of each sprint, the team will conduct a check in meeting and update each other on our progress.

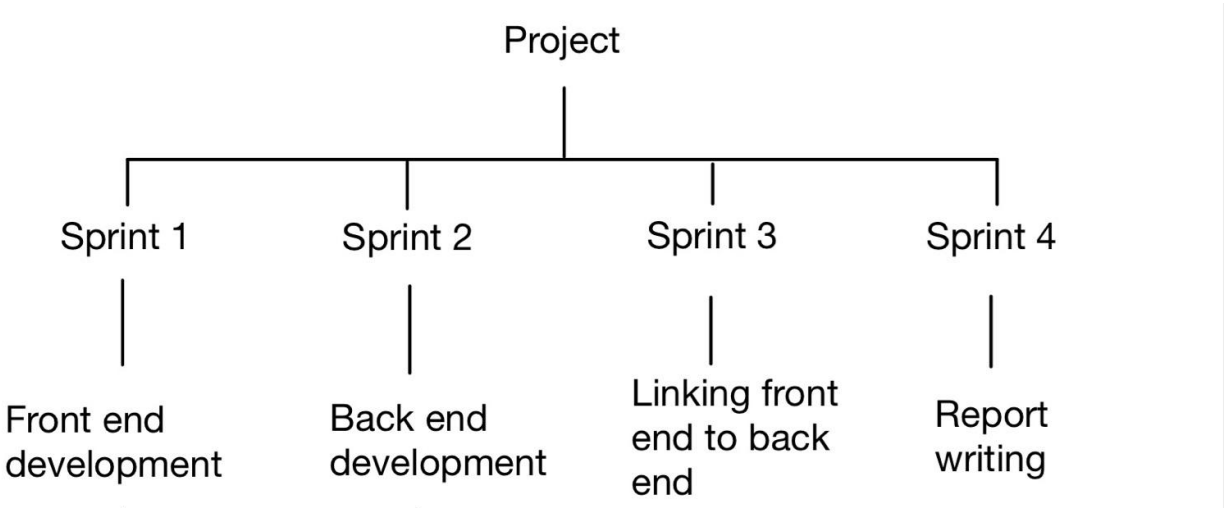
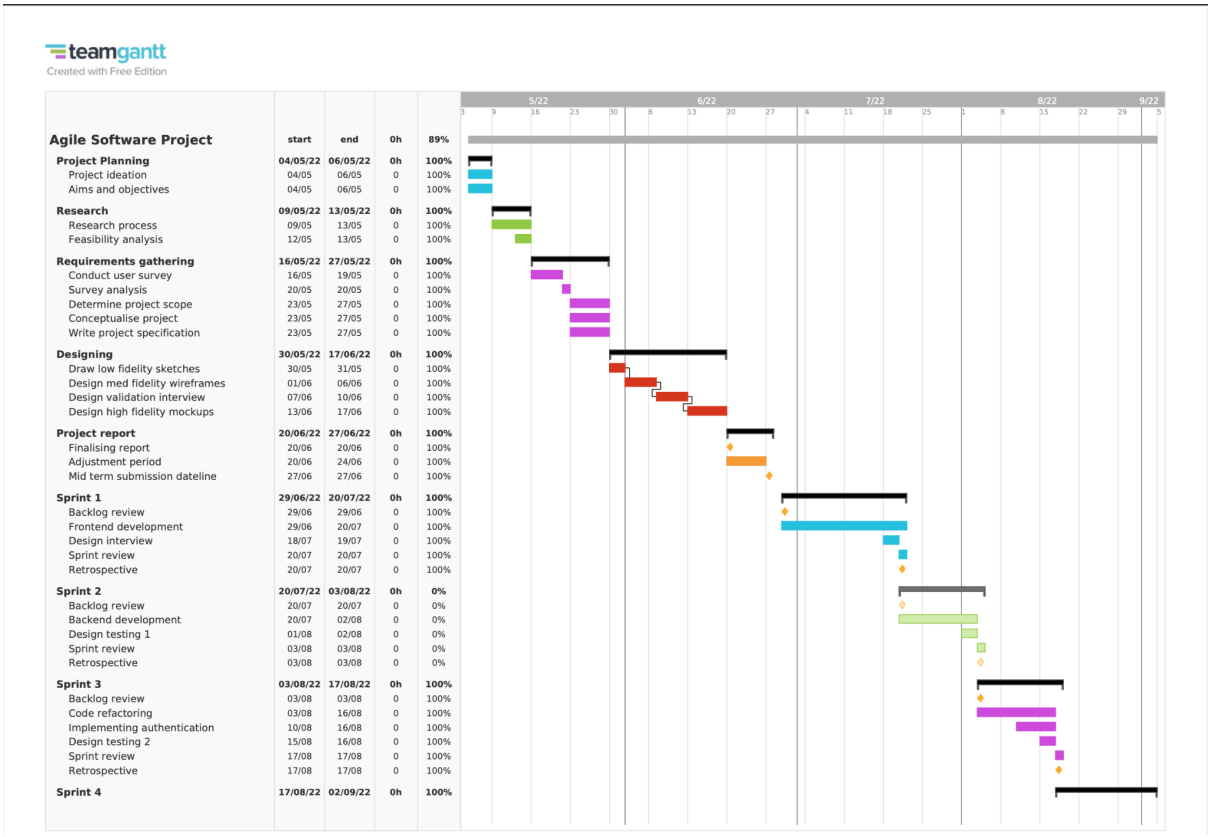


Figure 2.0 - Work breakdown structure mind map

2.2.2 Gantt Chart

The Gantt Charts show the typical number of days required to finish the different phases of our project. It helps us estimate project duration, determine the resources needed, and arrange the order in which our team will complete tasks. They also help with dependency management.



Agile Software Project	start	end	0h	89%
Project Planning	04/05/22	06/05/22	0h	100%
Project ideation	04/05	06/05	0	100%
Aims and objectives	04/05	06/05	0	100%
Research	09/05/22	13/05/22	0h	100%
Research process	09/05	13/05	0	100%
Feasibility analysis	12/05	13/05	0	100%
Requirements gathering	16/05/22	27/05/22	0h	100%
Conduct user survey	16/05	19/05	0	100%
Survey analysis	20/05	20/05	0	100%
Determine project scope	23/05	27/05	0	100%
Conceptualise project	23/05	27/05	0	100%
Write project specification	23/05	27/05	0	100%
Designing	30/05/22	17/06/22	0h	100%
Draw low fidelity sketches	30/05	31/05	0	100%
Design med fidelity wireframes	01/06	06/06	0	100%
Design validation interview	07/06	10/06	0	100%
Design high fidelity mockups	13/06	17/06	0	100%
Project report	20/06/22	27/06/22	0h	100%
Finalising report	20/06	20/06	0	100%
Adjustment period	20/06	24/06	0	100%
Mid term submission dateline	27/06	27/06	0	100%
Sprint 1	29/06/22	20/07/22	0h	100%
Backlog review	29/06	29/06	0	100%
Frontend development	29/06	20/07	0	100%
Design interview	18/07	19/07	0	100%
Sprint review	20/07	20/07	0	100%
Retrospective	20/07	20/07	0	100%
Sprint 2	20/07/22	03/08/22	0h	0%
Backlog review	20/07	20/07	0	0%
Backend development	20/07	02/08	0	0%
Design testing 1	01/08	02/08	0	0%
Sprint review	03/08	03/08	0	0%
Retrospective	03/08	03/08	0	0%
Sprint 3	03/08/22	17/08/22	0h	100%
Backlog review	03/08	03/08	0	100%
Code refactoring	03/08	16/08	0	100%
Implementing authentication	10/08	16/08	0	100%
Design testing 2	15/08	16/08	0	100%
Sprint review	17/08	17/08	0	100%
Retrospective	17/08	17/08	0	100%
Sprint 4	17/08/22	02/09/22	0h	100%

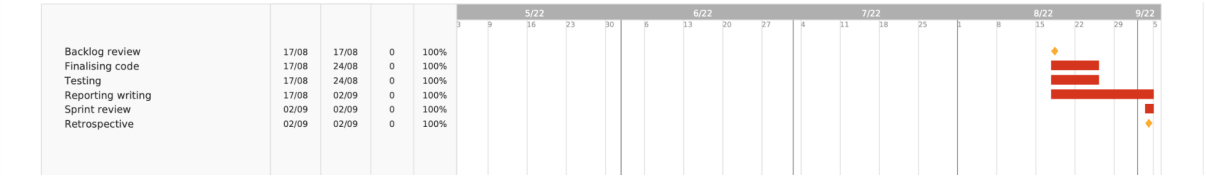


Figure 2.1 - Gantt Chart of The Project

2.2.3 Milestones

Sprint	Milestone	Description	Considered Reached When The Team Has:
1	1	**A static frontend user interface**. In the first milestone, the team's primary focus is to produce a static frontend interface.	1. Completed coding all the web pages. 2. The design of the website closely resembles our mock up. 3. Users are able to navigate around the website. 4. No ground breaking bugs.
2	2	**A working backend.** In the second milestone, the team will be working on setting up a functional backend.	1. Able to post data from postman into the database
3	3	**Link the frontend to the backend.** In the third milestone, the team will be integrating the frontend to the backend.	1. Able to read data from the database to the dashboard 2. Able to post data from the form into the database
4	4	**Create additional features beyond the originally planned scope.** In the fourth milestone, the team aims to add additional features into the website that are beyond the original planned scope. **Report writing**	

Table 2.0 - Milestones

2.2.4 2-Week Sprints

To manage our workflow, the team subscribes to the scrum and kanban framework. By breaking down the project delivery into smaller pieces, we are able to collaborate and respond swiftly to complicated situations.

The team conducts each sprint within a 2-week timeframe to complete a set of work. We arrive with this duration after taking into consideration each team member’s level of commitment inside and outside of school. 2 weeks sprints provide us greater freedom to respond to changes, and enable us to release high-quality work faster and more frequently.

The team begins each sprint by establishing the focus and agenda for the sprint. This creates a setting in which the team is inspired, challenged, and capable of achieving success. Next, we plan the what, the how, the who, the inputs and outputs of the sprint.

After outlining the sprint's aim and the backlog items that contribute to it, the scrum master chooses what can be accomplished and what the team will do to achieve it. To accomplish the sprint goal, the development team plans the work that must be done. The product backlog provides the team an excellent place to plan because it contains a list of prospective items for the current sprint. We usually take into consideration our current capacity as well as the work already completed in the increment.

2.2.5 Retrospectives

The team runs retrospectives at the end of each sprint. This serves as a chance for us to assess our own performance and develop a strategy for future improvements in problematic areas. By reflecting on the past, retrospective helps us embrace the principle of continual progress and guard against the dangers of complacency.

We start each retrospective by making a brief summary of the things that went well and the things that could be done better. Next, we order this list according to importance and brainstorm strategies for improving the list. Towards the end of the meeting, we identified a solution to each problematic area, along with clear owners and deadlines.

2.3 Resource and Time Allocation

After the submission of the Project Proposal in midterms, this marked the start of realising an idea for an actual functional product. This transition was a big step for the team, hence, the first week was dedicated to deciding what tech stack to employ. We were critical in our discussions as we would like to avoid changing the tech stack midway through the project. As a result, a full stack utilising MongoDB, Express, React and Node (MERN) was decided. The following subsections will briefly elaborate on the tech stack and its resources used.

2.3.1 Front End

React was chosen to be the choice of front end framework instead of Angular or Vue as the learning curve would be much more manageable. More time was allocated for the team to learn and familiarise the framework as the team consisted of a mix of new and beginner React developers. Hence, two weeks were allocated to ensure that everyone was comfortable with certain concepts and skills before we started the development process.

The bulk of the learning resources were referenced from free online tutorials such as the official React documentation, YouTube vidoes, and articles. Therefore, during these three weeks, half the team were focused on picking up a new framework, while the other half was learning to improve their React skills.

Topic	**Resource**
General reference	React Documentation, (Facebook, n.d.)
Free tutorials	Free Code Camp, (Free Code Camp, n.d.)
React Crash Course	Traversy Media on YouTube (Traversy Media, 2021)

Table 3.0 - Resources used in learning React

2.3.2 Back End

MongoDB was employed as the document NoSQL database, while Express was used as a middleware for Node, and Node allowed us to run Javascript code outside of a web browser. One week was allocated to familiarise the back end stack as Express and Node have been covered by another module from the course, hence, the team felt more confident to advance into acquiring knowledge on MongoDB.

Similarly to the front end, majority of the learning resources also came from free online tutorials. During that one week of learning, we committed ourselves to ensure the back end stack was set up appropriately.

Topic	**Resource**
General reference	MongoDB Documentation, (MongoDB, n.d.)
Free tutorials	Free Code Camp, (Free Code Camp, n.d.)
Online articles	DEV, (Friday, 2021)

Table 3.1 - Resources used in learning MongoDB, Express and Node

Section 5.0 System Development covers an extensive elaboration of the front and back end stack including our development process and a full breakdown of the tech stack in detail.

3.0 Prototyping and Iteration

3.1 High Fidelity Prototype (First Iteration)

Users are welcomed to a homepage that introduces them to the purpose of the web application. Users can then begin to search for workspaces by keying in the names of locations that they are interested in, which will redirect them to the results page of their searches.

Clicking on the user “List A Space” button at the top of the navigation bar redirects users to the account page where they can login to list a space on our web application for discovery.

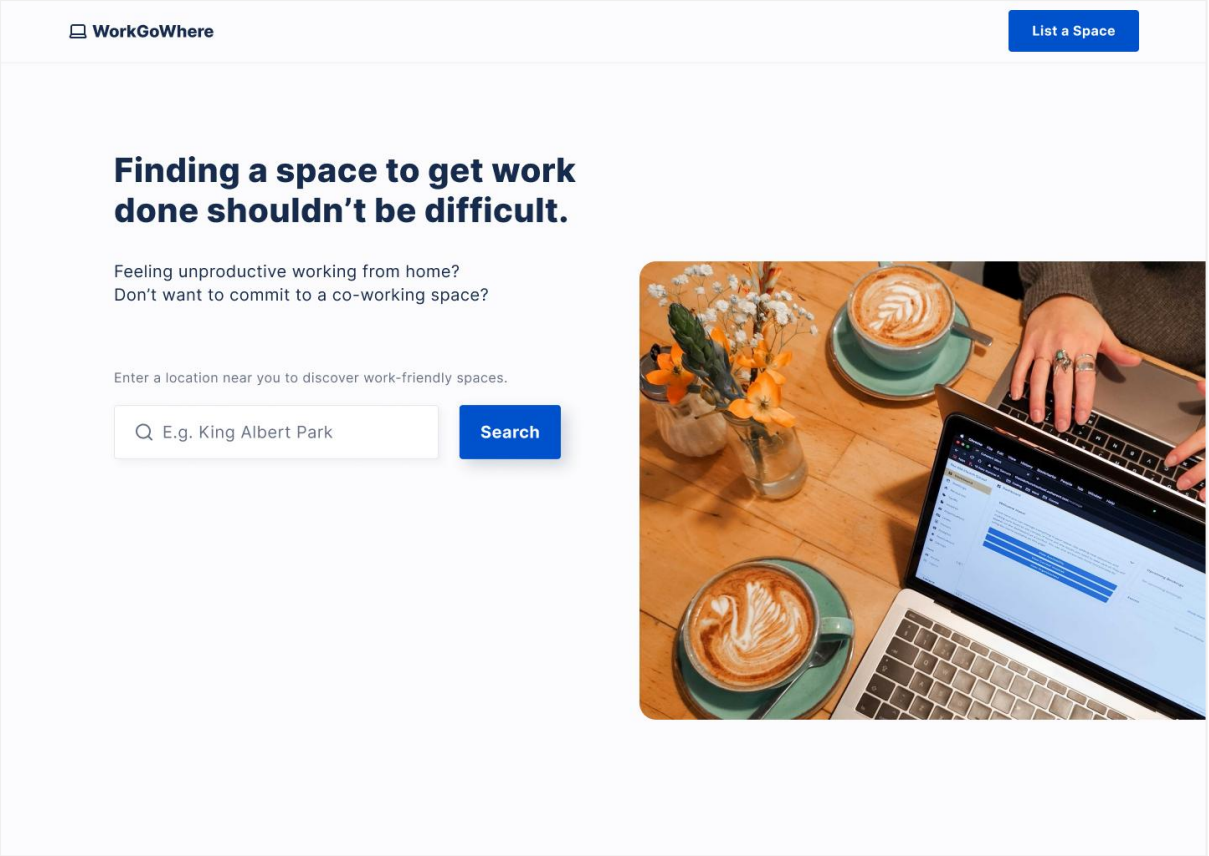


Figure 3.0 - Home page

The result page interface will display a search bar at the top, accompanied by a dropdown list of available filters that users can check to narrow down and improve their accuracy of their desired results.

Should an inappropriate search location be inputted, the result page will simply return default results/no results. (Brief explanation on what happens if an incorrect/unavailable location name is searched, need to decide which one)

The results will be in rows of cards that users can sift through. Each card will showcase a carousel of pictures for each workspace, accompanied by additional information such as the names, addresses, work hours and amenities provided. Hovering over each card causes them to “stand out”, providing a more intuitive visual experience.

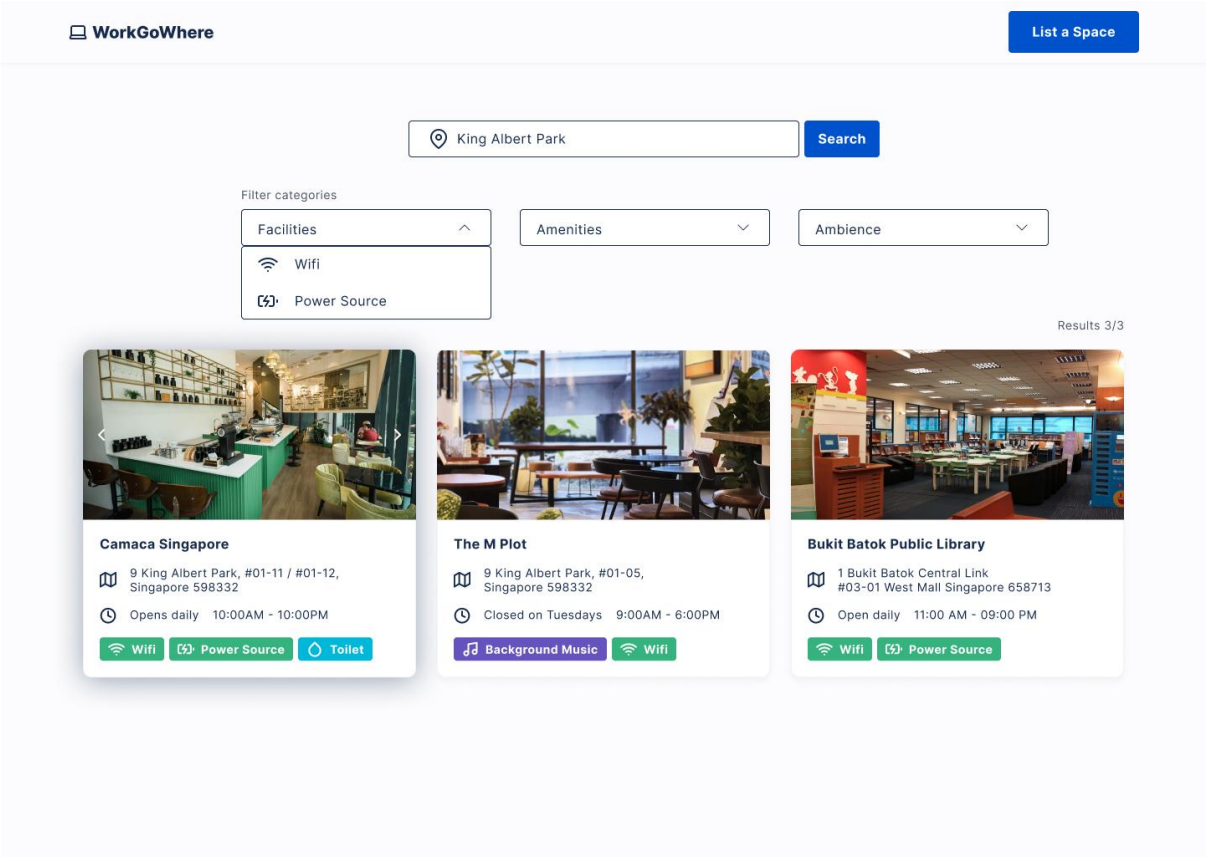


Figure 3.1 - Results page

Upon clicking a card, a pop-up modal will appear displaying even more information and larger pictures. Users can then click the “Secure a Table” button to begin making a reservation.

Doing so will redirect users to the web page of the selected workspace, where they can begin making their reservations.

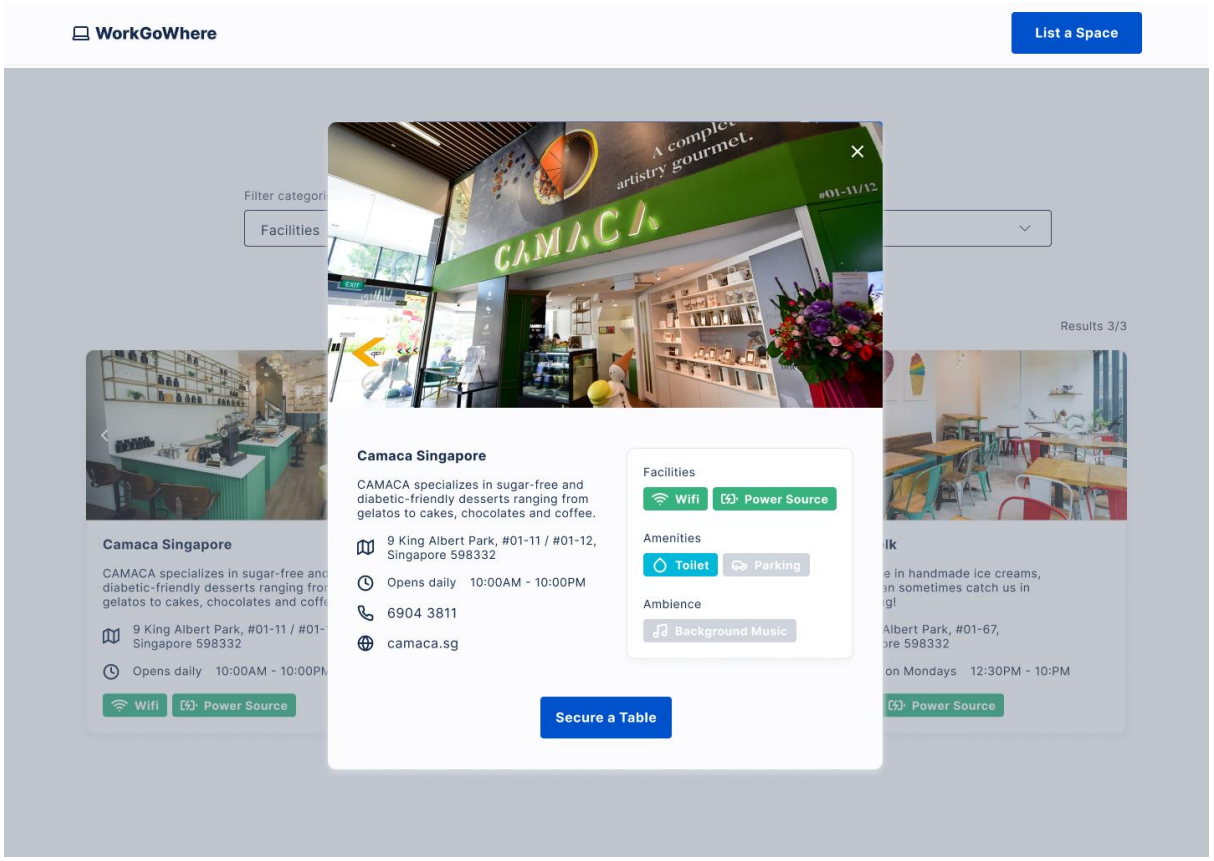


Figure 3.2 - Listing pop up in results page

The “List a Space” button in the navigation bar brings the users to the account page. Users with pre-existing accounts can simply login with their email and password in the login form provided. Successful submission sends users to their own unique dashboard page.

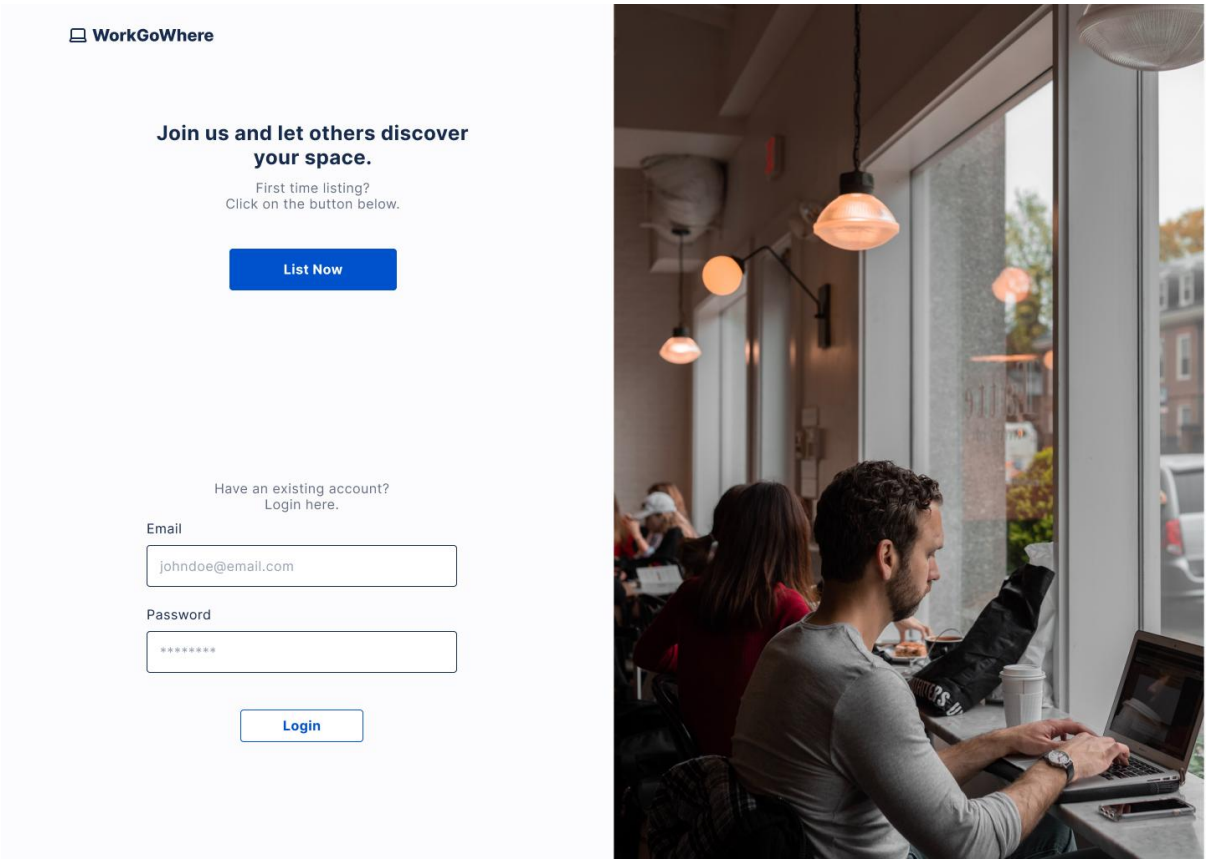


Figure 3.3 - Accounts page

By clicking “List Now” on the account page, a pop-up modal appears with a form, prompting users to create a new account before they are allowed to list a space. Users will need to provide their full name, email and password for account creation.

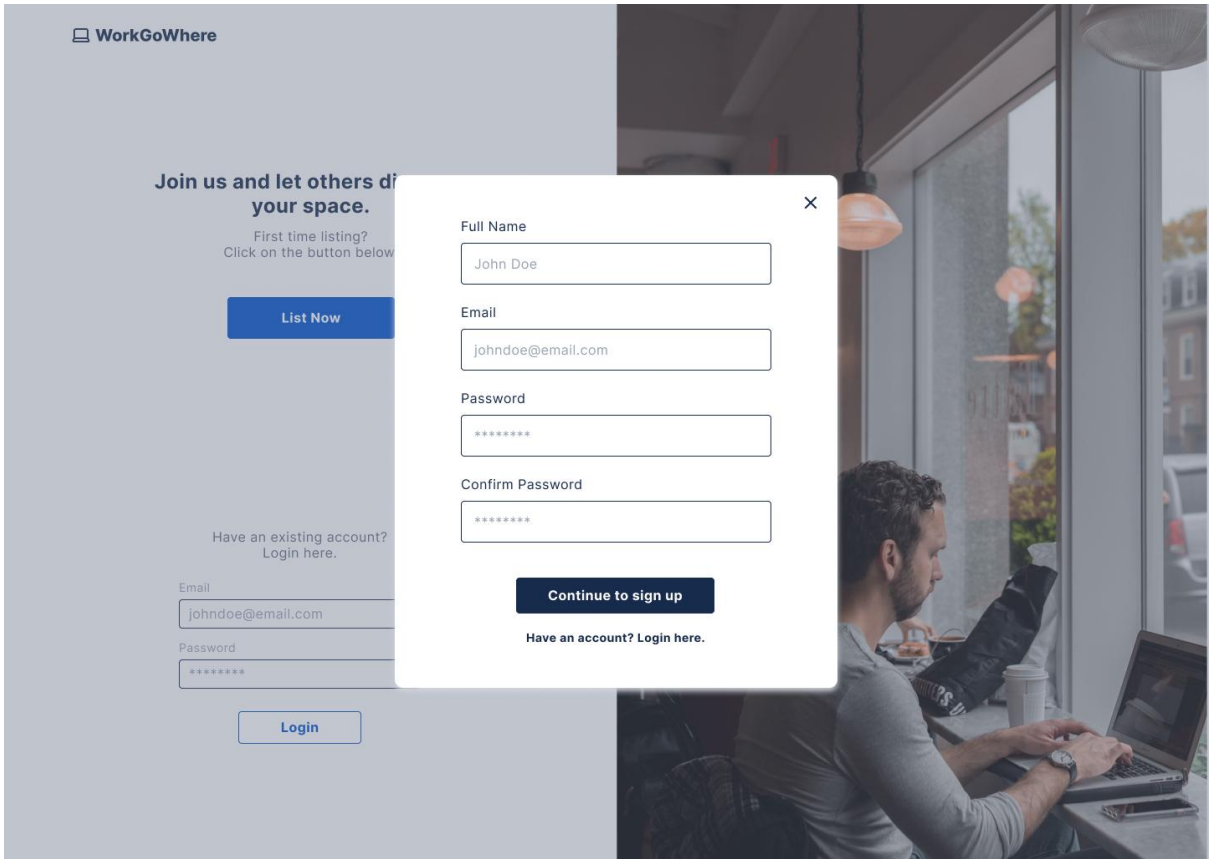


Figure 3.4 - Sign Up Pop-up Modal

The user-form page allows users to fill in information regarding their workspace, including name, description, address, operating hours, contact number, estate, website link and services provided. Users can also upload images of their workspace.

Upon completion, users will click the “Preview your listing” button.

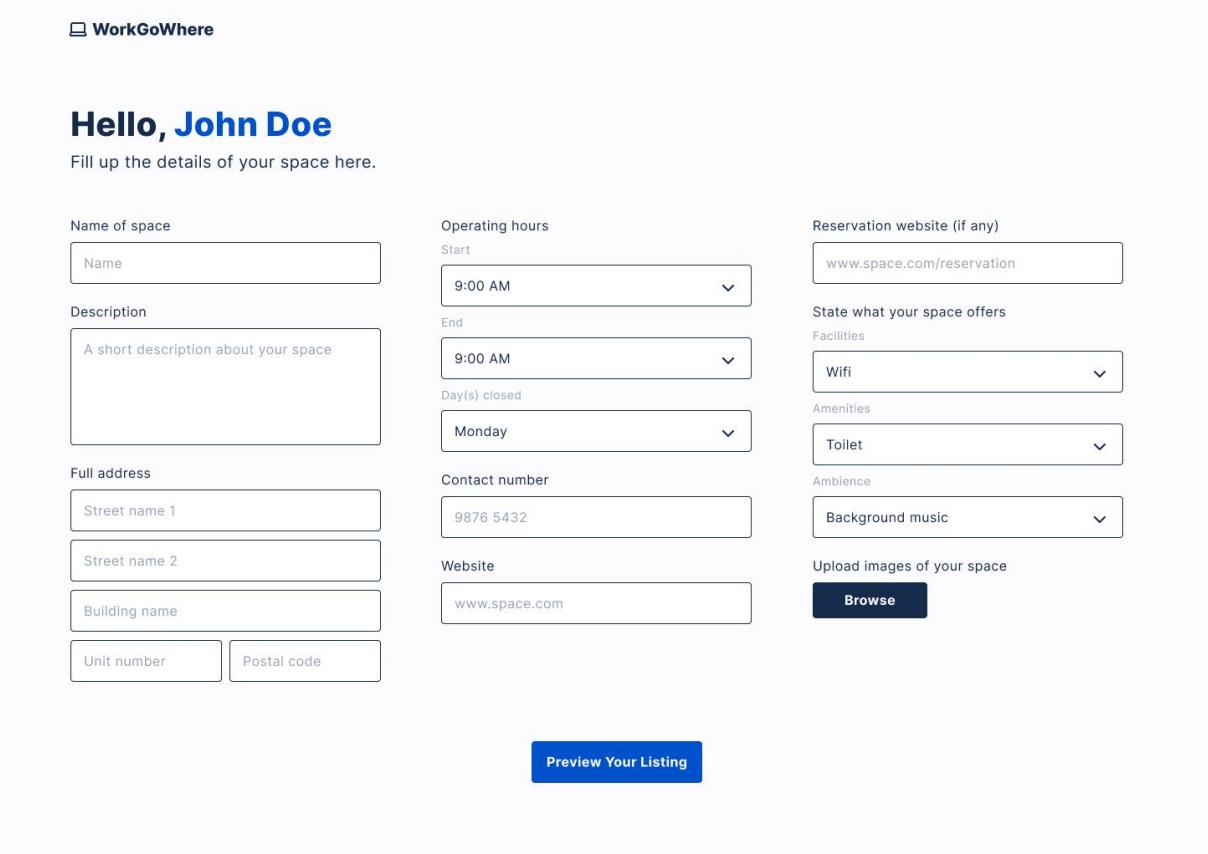


Figure 3.5 - UserForm page

The preview page allows users to preview what their workspace card will look like on the results page and pop-up modals.

Users can click the “Edit” button to return to the user-form page to make more changes. Otherwise, they can simply click the “Submit” button.

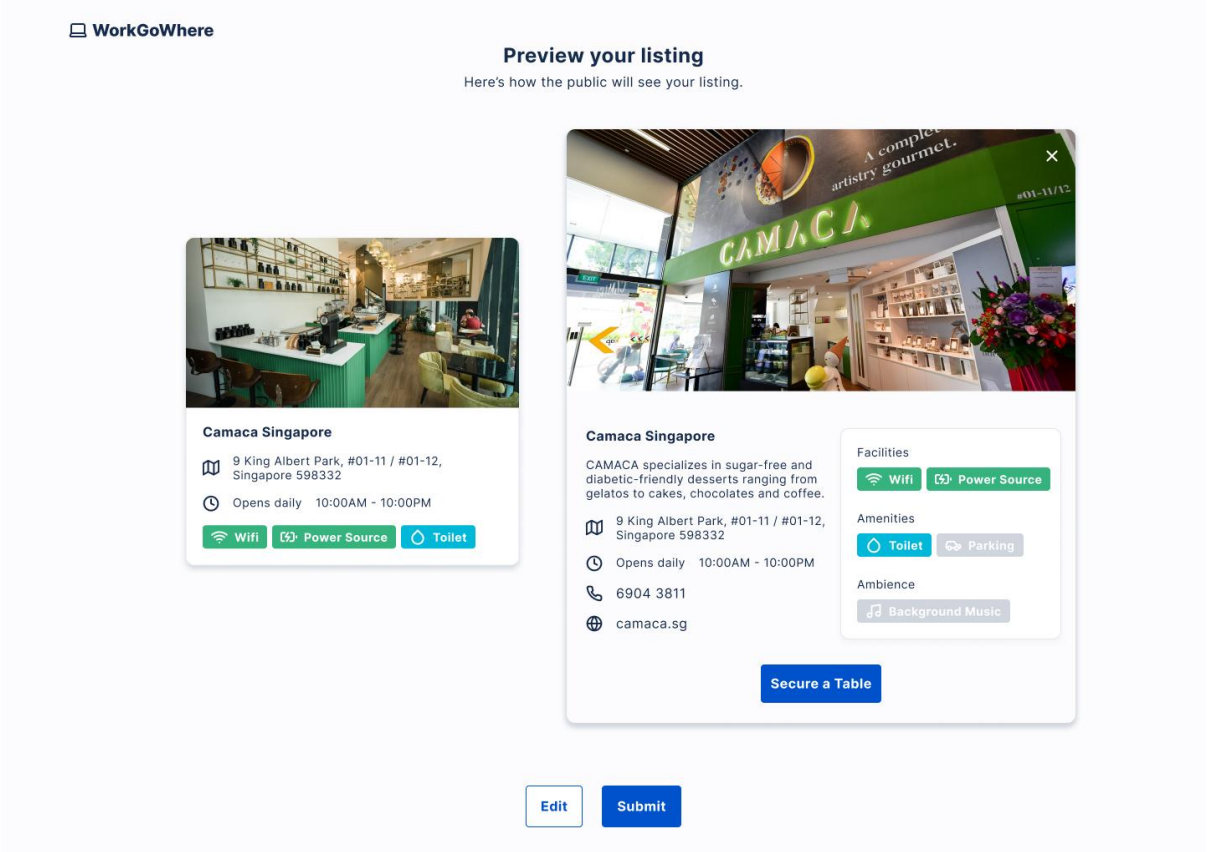


Figure 3.6 - Preview page

Upon successful completion of listing a space, the user can click “List Another Space” to make another listing.

Users can also scroll down or click “Meet the team” to see information about the developers.

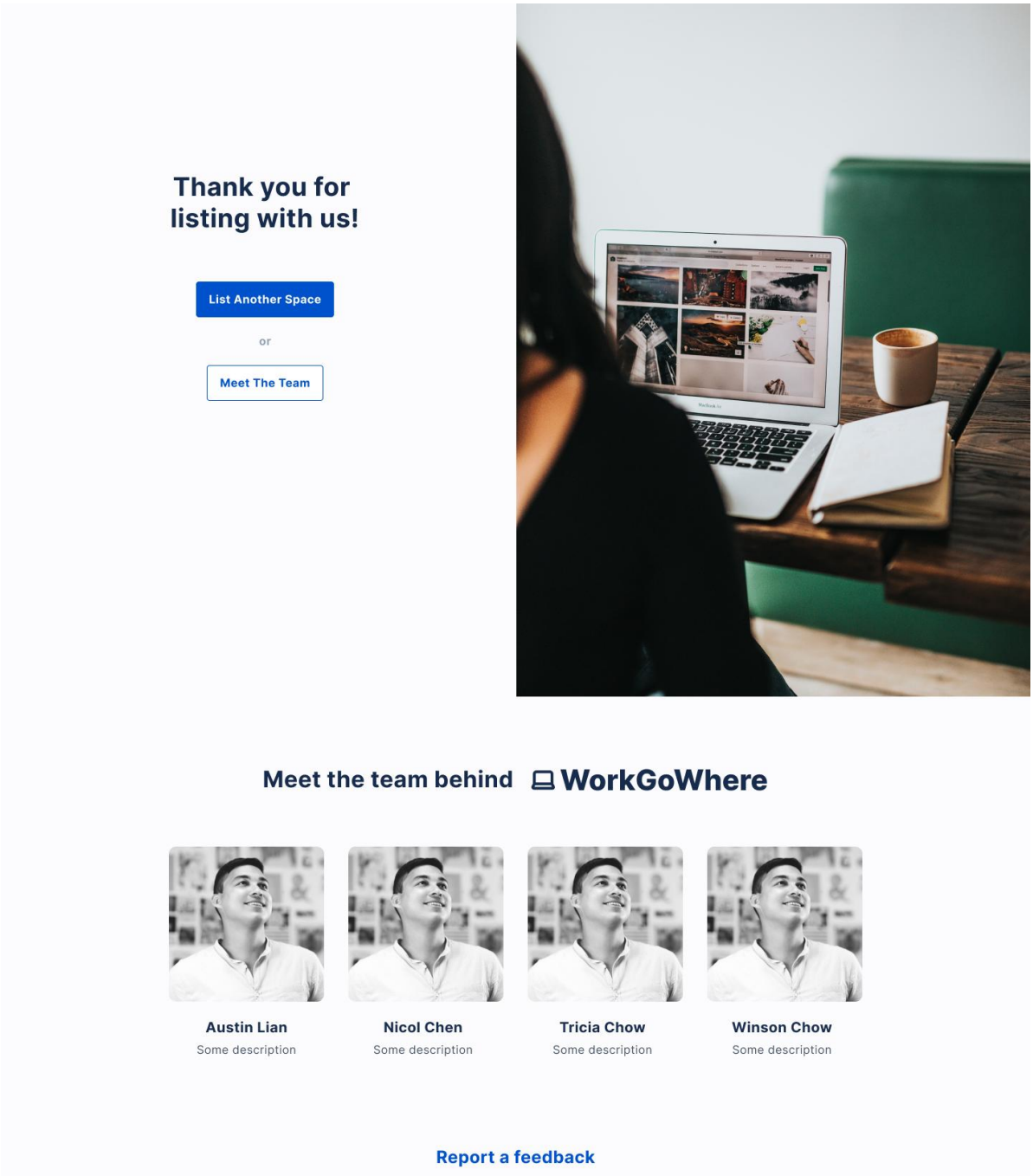


Figure 3.7 - Thank you page

3.2 User Experience Testing

As part of our user centred design efforts, after creating a working version of our website, the team reached out to our stakeholders to perform user experience testing.

User experience testing is conducted on a one to one interview with users. Users are first asked for their initial impression of the website. Their response provides the team insights into whether the intent of our website is conveyed accurately. Next, users are given five minutes to explore the entire website and share their response as they go.

Here are some of the key takeaways from the interviews:

Topic	**Feedback**
Intent of website	Intent is clear. Users are able to understand the purpose of the website from the home page.
Search bar in home page	Users find the search bar in the homepage unintuitive. General consensus is a search bar that is able to search beyond just locations, such as: - Name of working space - Working space with filtered keywords in place
Filter menu in results page	Users find the filter menu unintuitive. The existing dropdown filter bar does not allow selection of multiple keywords. Users have expressed the possibility of having more than one selection.
Modal card layout in results page	Users prefer a modal layout where the carousel images and working space details are side by side.
Lack of account management	The lack of an account management page is brought up among a number of users. The current website only allows users to list their working space.
Working space submission form	Users find some sections of the form unintuitive. The existing form uses a dropdown list for days closed, facilities, amenities and ambience. Users have expressed the possibility of having more than one selection.

Table 4.0 - Summary of User Experience Testing

Please refer to Appendix A for an example of a full interview.

3.3 Outcome and Second Iteration

Instead of using a search bar, we used a button instead. Users can then begin to search for workspaces by clicking on the “Search Now” button, which will redirect them to the results page.

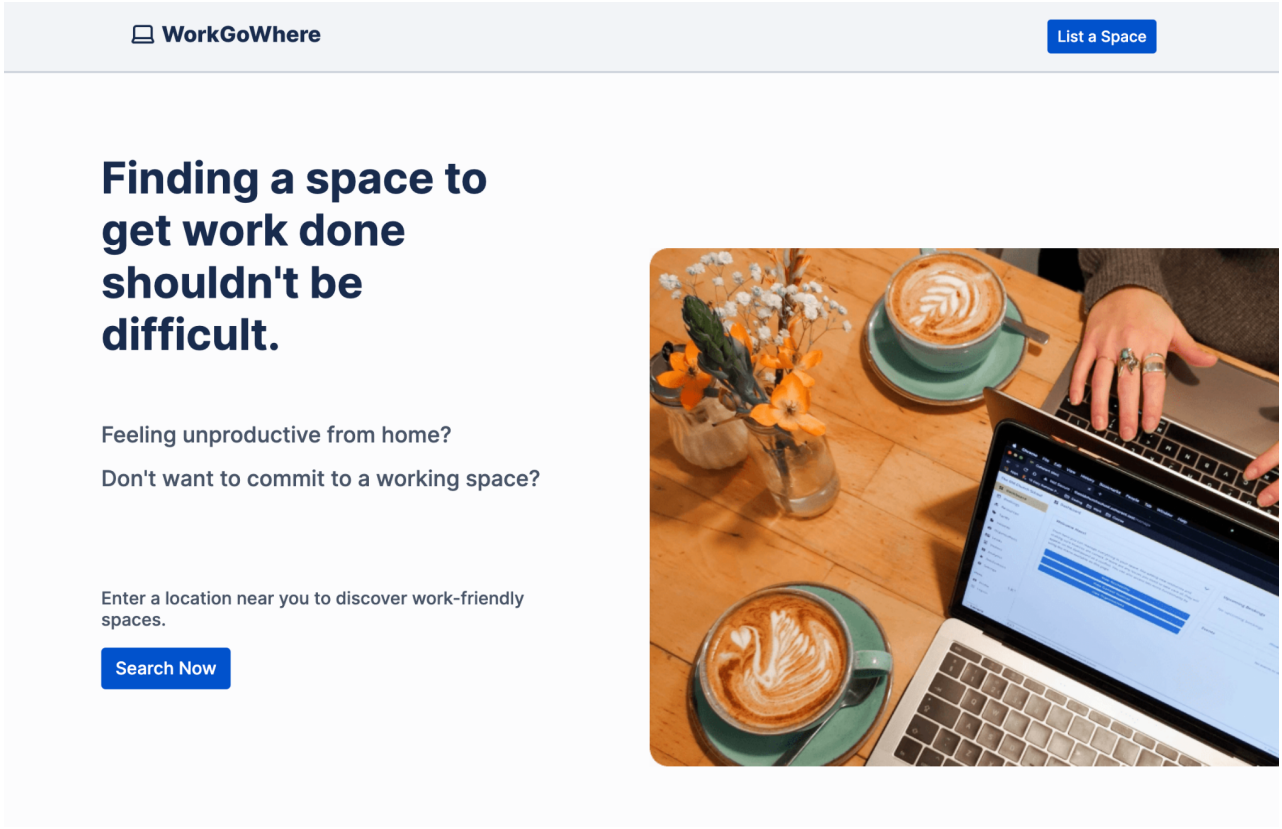


Figure 4.0 - Home page

The filter menu on the results page has also been modified from a dropdown to checkboxes. This alleviates the issue of users being limited to certain options.

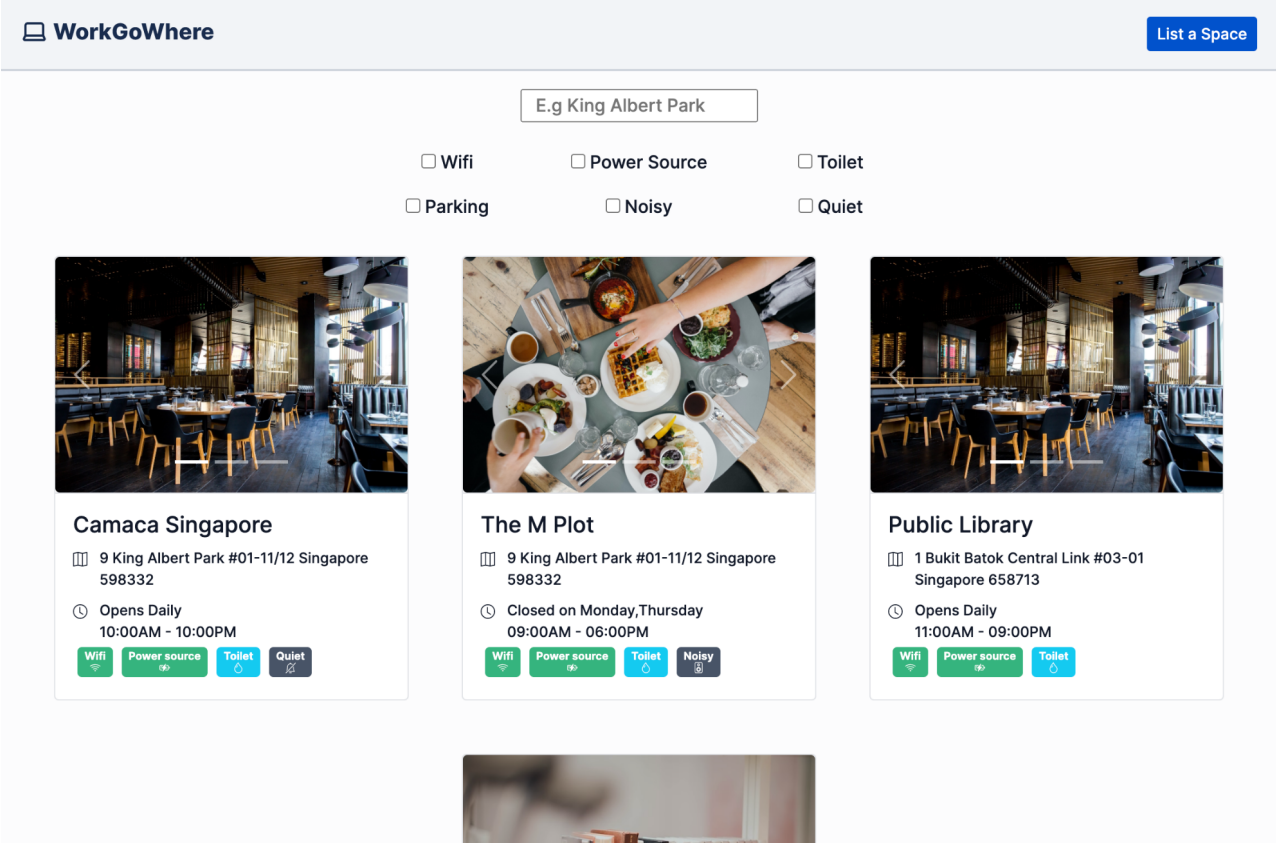


Figure 4.1 - Results page

In the results pop up modal, we rearrange the layout of the card as a quality of life improvement based on users' feedback. Instead of laying the carousel and details from top of each other, they are arranged side-by-side.

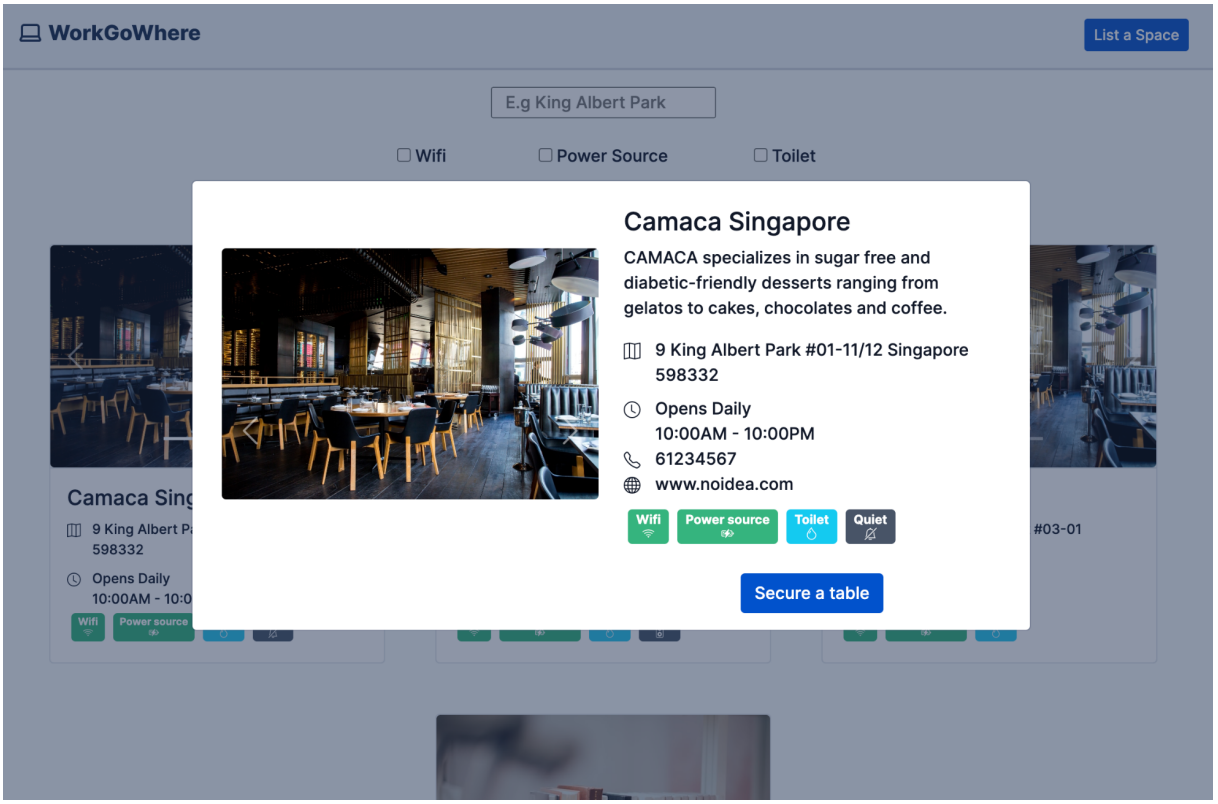


Figure 4.2 - Results page pop-up modal

The accounts page did not undergo any changes from the previous iteration.

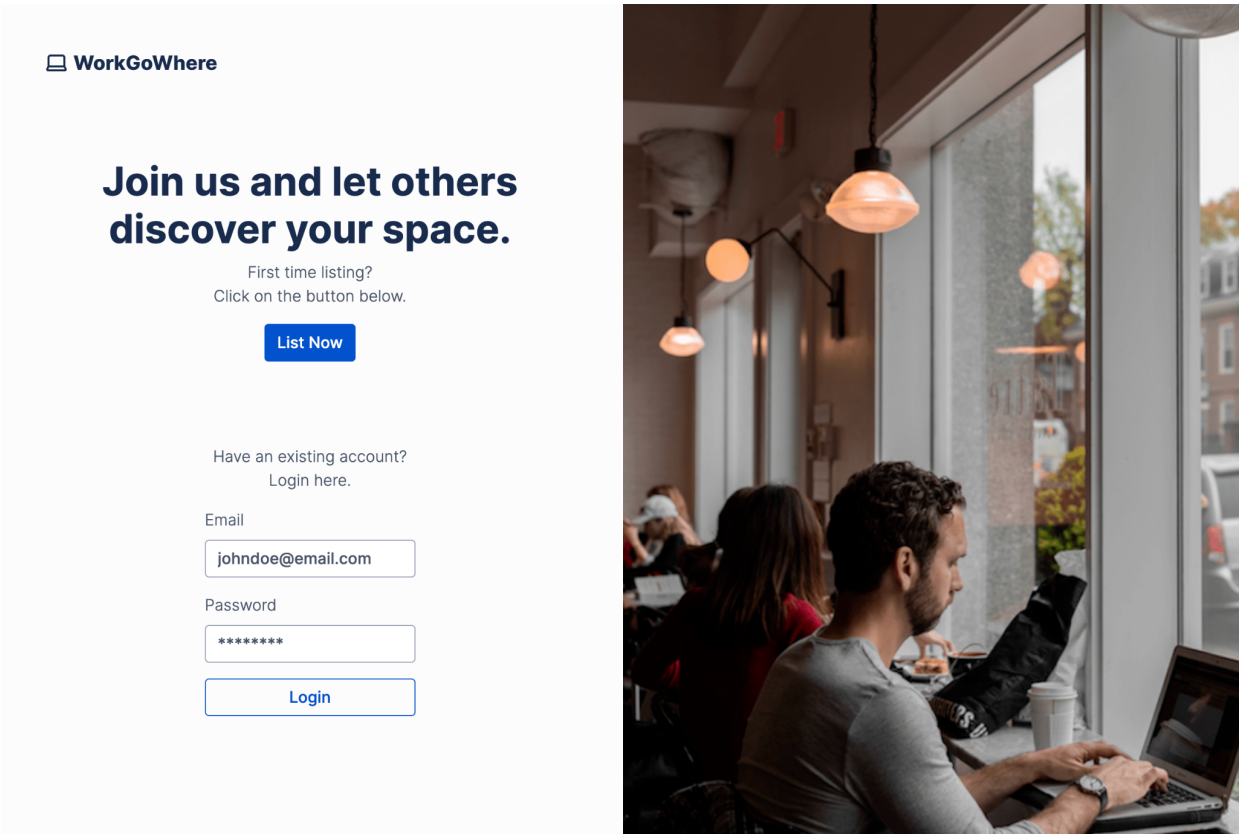


Figure 4.3 - Accounts page

The team added a new addition of a “Dashboard” page because in the previous iteration, users were directed to the user form immediately after signing up or logging in, which did not provide the service and convenience of managing their listings.

The Dashboard Page provides an overview of the user’s listing, a place to manage their existing listings and most importantly, to upload a new listing. New users will not be able to see a dashboard, as there are no listings found in the database tied to new users. They will be prompted to add a new listing by clicking on the “List a Space” button. This will bring them to the user-form page where users will provide information about their listings to be uploaded in the form.

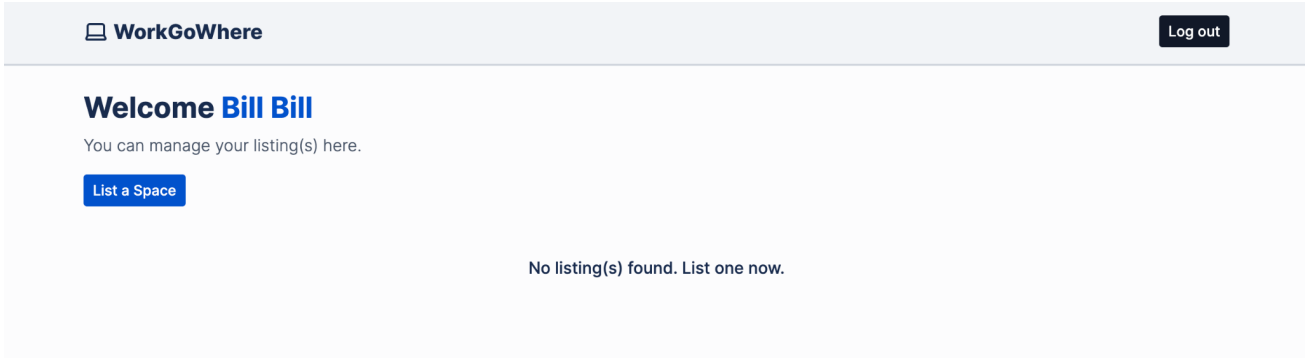


Figure 4.4 - Dashboard page (No existing listing)

For existing users who previously have submitted a listing or listings, they are able to view them in an accordion-styled dashboard, to provide ease of viewing their listing(s) at a glance. When clicked on a row, it will expand and display all the information pertaining to the particular listing. On top of that, users have the option to delete a particular listing by clicking on the “Trash Can” icon should they need to do so. Figure 4.5 shows an example of a dashboard that is populated with a user’s existing listings.

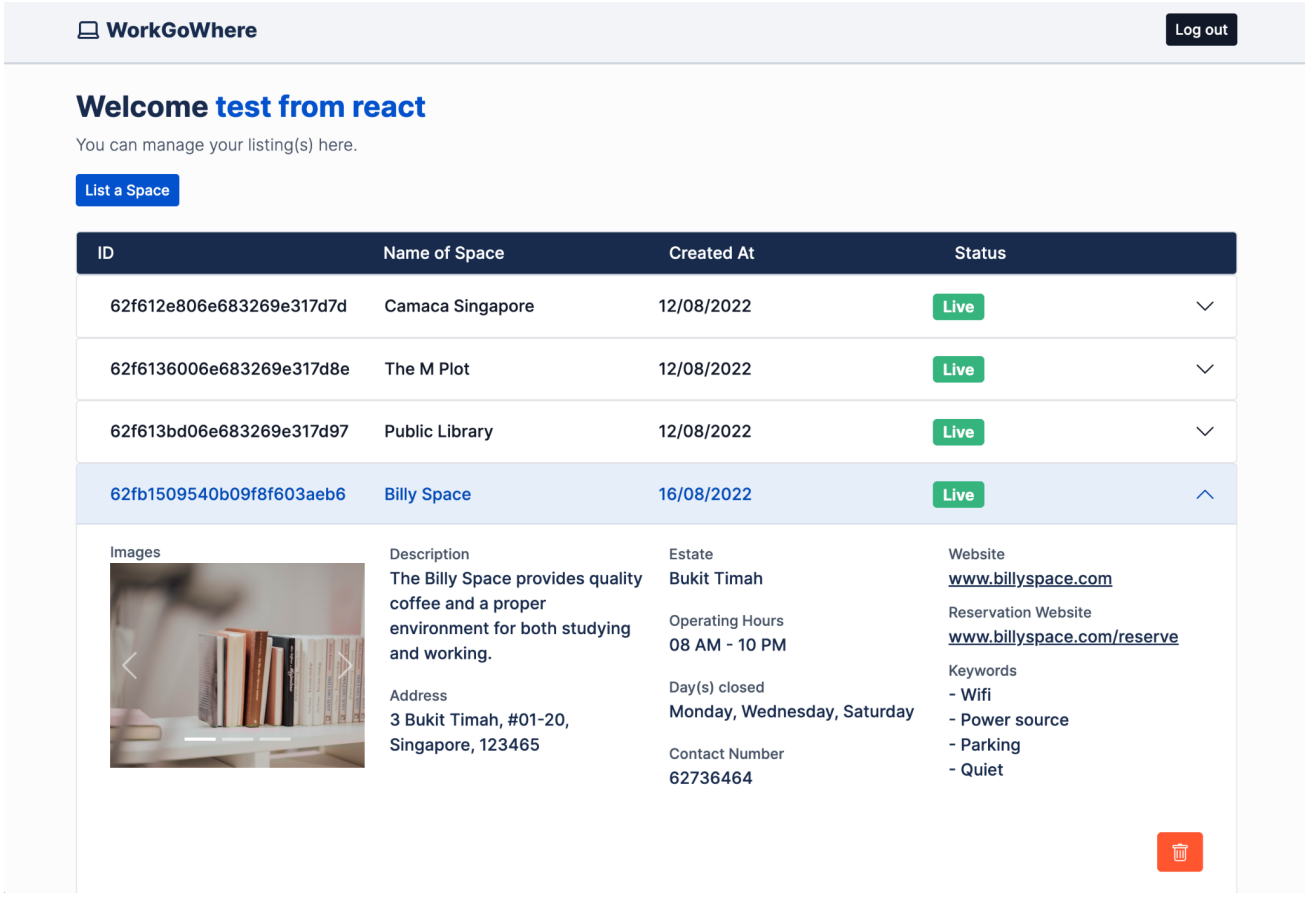


Figure 4.5 - Dashboard page (With existing listing)

On the user form page, all dropdown lists are replaced with checkboxes. This allows users to select more than one option, rather than limiting their choices to one.

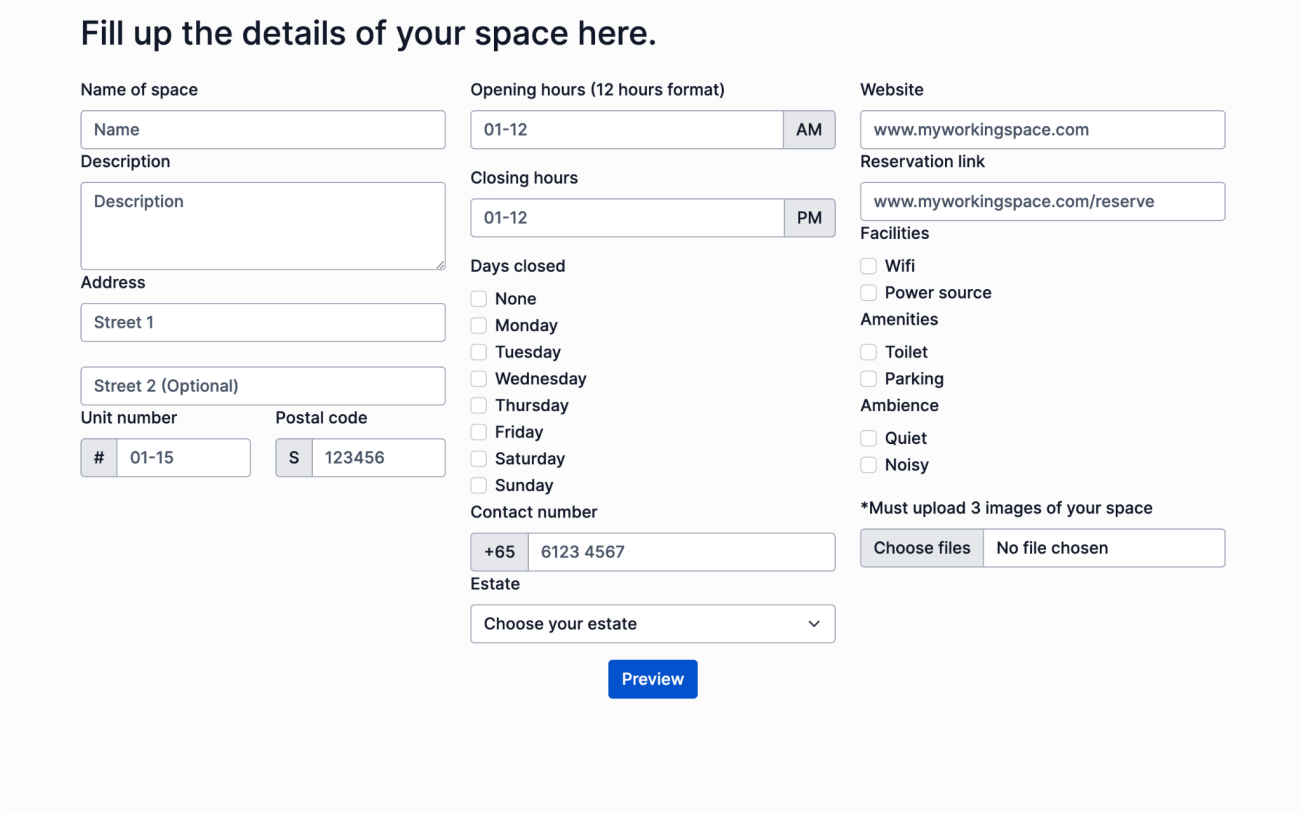


Figure 4.6 - UserForm page

The preview page did not undergo any changes from the previous iteration.

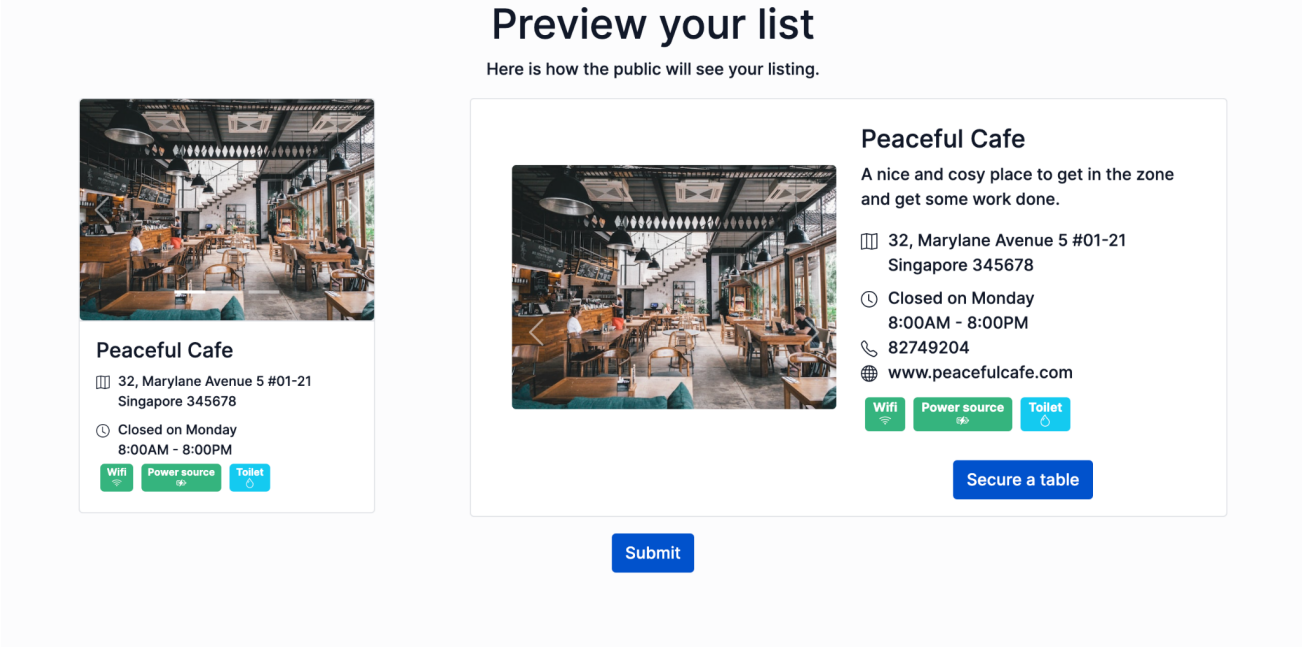


Figure 4.7 - Preview page

The thank you page did not undergo any changes from the previous iteration.

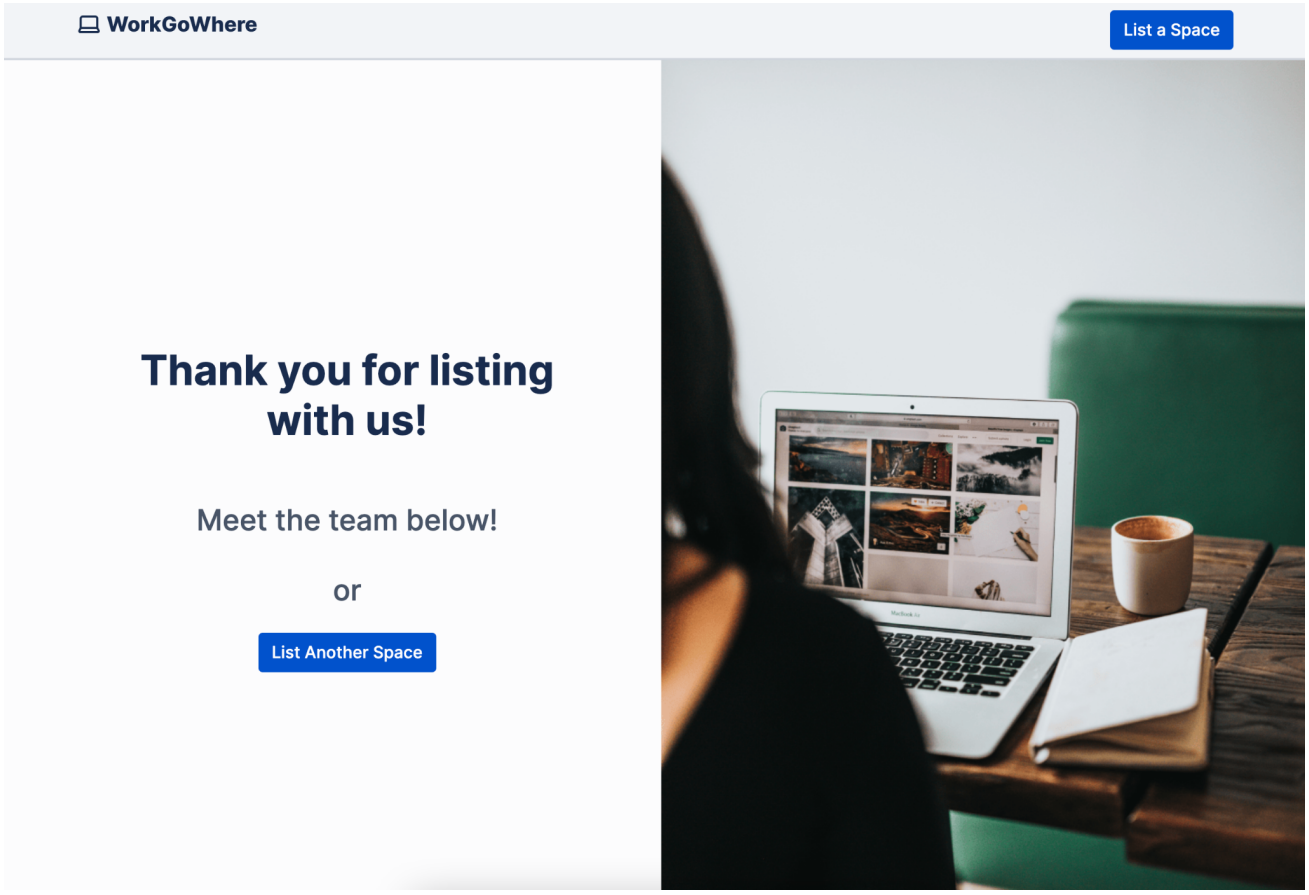


Figure 4.8 - Thank You page

4.0 Design Specifications

4.1 Architecture

4.1.1 Module Design

We will use an activity diagram to describe the behaviour and connections between the modules and components that make up the systems of our application. This visualising tool provides a more thorough explanation of the procedures in a user case diagram. As a result, it determines every scenario that could occur when utilising the program.

4.1.2 Use Case Description

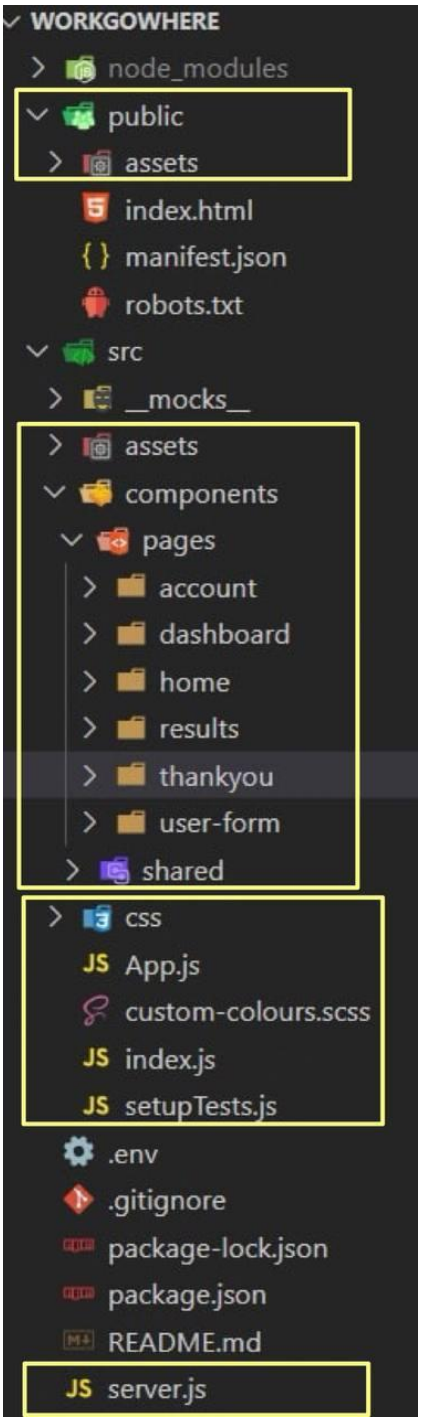
Use Case	- Recommend a space based on user requirements. - Clients can list a space of choice.			
Actors	Users, clients and administrator			
Pre-condition	Client must have an account			
End Result	- Users are able to find a place suitable for them to work in. - Clients can list their spaces for people to work in.			
Main Flow	**User**		**Client**	
		Actions		**Actions**
	1.	User will enter a location.	1.	Client will log into the site with their registered account.
	2.	User will filter spaces based on requirements.	2.	Client will input details of the space.
	3.	User will then click on the space of choice.	3.	Client will be able to preview listing before submitting.
	4.	Pop-up modal will appear to display more information about space.	4.	When details are entered correctly, clients will submit listing.
	5.	Users are able to reselect location and filters.	5.	Space will be listed and can be viewed by users and clients.
Alternate Flow	1.	If location is not found, no spaces will be recommended.	1.	For new clients, they will need to register an account.
	2.	If requirements are not found in the filters, no spaces will be recommended.	2.	Listing will not be listed if the form is not filled up correctly.

Table 5.0 - Use case description

4.2 Technical Design

As mentioned in 2.3.1 Front End and 2.3.2 Back End, the team has resolved to produce a web app that utilises React as a front end framework, and MongoDB, Express, React and Node for the back end set up. The following subsections will go into detail of the technical design as to how we structured our tech stack to achieve our goals.

4.2.1 Front End Stack (React)



The front end files are structured in a way that categorises each JS file into individual folders based on the respective web pages. This structure enables better control over each web page. Thus, providing ease to programmers when debugging or maintaining the web pages. Images that are used in the webpages are separated into the “public” folder which will store the images uploaded by the owners and the “src” folder which stores private images used in the pages.

Apart from individually categorised folders containing respective JS files, we created a separate folder named “shared” which contains certain components that will be used across different web pages. These components created are fixed components which do not require editing or maintenance of code. An example of said component would be the navigation bar where it is used across most web pages and does not require editing or maintenance.

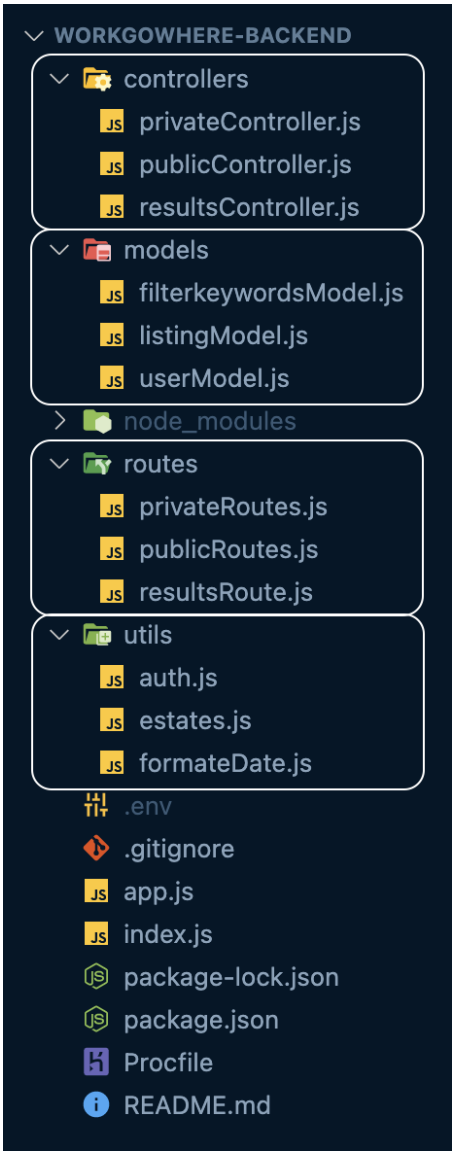
The main “css” styling of all webpages are categorised into another folder where we have a better control over the fonts and the colour scheme used.

Lastly, an uncategorized “server” JS file was created apart from the rest as another port was required to post the uploaded images.

The detailed explanation of the files will be elaborated in Section 5.3 Back End Breakdown.

Figure 5.0 - Front end folder structure

4.2.2 Back End Stack (MongoDB, Express, Node)



The back end files are structured in a Model, View, Controller (MVC) manner. This type of structuring is a common pattern following the software design principle, separation-of-concerns as it provides for a better partition between different functionalities. This also helps with maintaining and improving the code base.

“Model” is typically responsible for managing data and business logic. “Controller” contains functions that manipulate certain data that will interact and update the model. The idea of the controller is to extract those functions into a separate file, and this to make things more modular and easier to manage.

From Figure 5.1, there is no “View” folder found as the results from the back end will be sent to the front end for React to render its user interface.

Furthermore, this back end repository is hosted on Heroku, which is a free platform to deploy applications on the cloud. It is hosted on https://workgowhere.herokuapp.com/. This was done so that developers could directly access the hosted endpoints on the front end, without having to run an additional local server on the back end.

The detailed explanation of the files will be elaborated in Section 5.3 Back End Breakdown.

Figure 5.1 - Back end folder structure

4.2.3 Database Design and Structure

The database has two collections of data: “User” and “Listing”. “User” is in-charge of storing user data upon account creation at our website. On the other hand, “Listing” stores all listings information submitted by a user.

The database is designed to be a one-to-many relationship because a user is valid to own multiple listings, however, a listing will only belong to one user. In Figure 6.0, the properties highlighted in yellow illustrate that `User.listingsPosted` can store an array of `Listing.id`. Whereas the properties highlighted in green illustrate that `Listing.listingOwner` can store a single value of `User.id`.

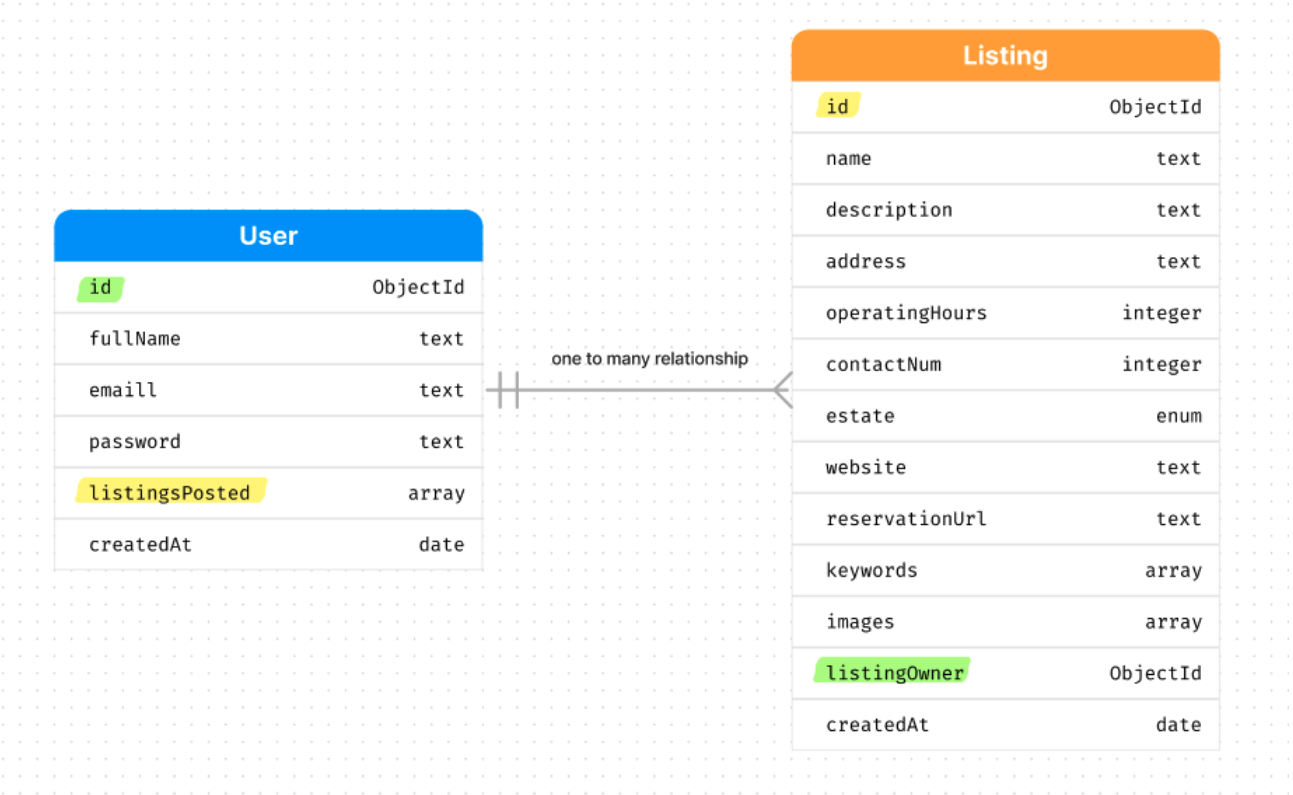


Figure 6.0 - Database entity relationship diagram

4.2.4 Dependencies and Libraries

Stack	Dependencies	Description
Front End	React	React is a free and open-source front-end JavaScript library for building user interfaces based on UI components.
	ReactBootstrap	Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development.
	React Bootstrap Icons	Bootstrap Icon is a free and open-source library for implementing icons of choice when building user interfaces.
	Node Sass	Node Sass is a free and open-source library that provides binding for libSass to Node.js.
	Universal Cookie	Universal Cookie is a free and open-source API which allows loading and saving of cookies in a react application.
	Web Vitals	Web Vitals is a free and open-source library for measuring all metrics on real users.
	Axios	Axios is a free and open-source library for making HTTP requests from the browser.
Back End	Express JS	Express JS is a web application framework that helps to manage servers and routes on top of Node JS.
	Mongoose	Mongoose is an object data modelling library which manages the relationships between data and schema validation. Also, it translates the objects in MongoDB and the objects in the code.
	Cors	Cross Origin Resources Sharing commonly known as “Cors” makes requests from one website to another possible.
	Multer	Multer is a middleware dealing with multipart or form-data. It is commonly used for uploading files such as images.
	Json Web Token	Json Web Token is an open standard that securely transmits information between parties as JSON objects.
	Bcrypt	Bcrypt enables building of a password security website that can improve with hardware technology, guarding against threats.
	Dotenv	Dotenv enables secrets to be separated from the source code which is commonly used when collaborating. An example would be database login credentials.
	Nodemon	Nodemon is a tool that automatically restarts the node application when a change is detected in the file.

Table 6.0 - List of dependencies

5.0 System Development

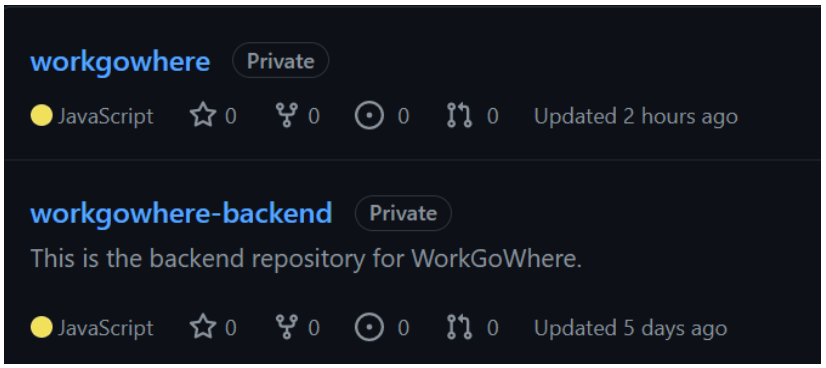


Figure 7.0 - Screenshot of our repositories on GitHub

The team has chosen GitHub as the version control application and has created separate repositories for the front and back end. The separate repositories were created so that the team was able to modify the codes and keep track/view the merge logs of respective repositories. By having separate repositories, the team has better control of the front/back end codes and understanding of the changes made by each team member. The instructions on how to get the application running can be found on the GitHub repository. The link to the GitHub repository can be found here: https://github.com/AgileProjectY2.

To view the application working live, you may click on the following link: https://www.youtube.com/watch?v=w9vIIb_zcD4

5.1 Development Process

The team followed a rigorous process throughout each development cycle. Using the scrum framework, we set each sprint to a duration of two weeks. This sets a soft dateline that is easy for everyone to adhere to. In addition, in the middle of each sprint, the team will have a check-in with each other. Members can use this check-in to update their progress, as well as raising any difficulties they are facing.

Towards the end of each sprint, before everyone's code is merged into the main branch, our code will undergo a code review from the head developer of the sprint. Austin, our front end head developer, reviews all the front end code submissions. Tricia, our back end head developer, reviews all the back end code submissions.

On the last day of the sprint, the team meets again and goes through a sprint review and retrospective. This helps us reflect what went well and highlight any shortcomings. We also take this chance to brainstorm new strategies to increase the team's efficiency in the next sprint.

5.2 Front End Breakdown

5.2.1 Home.js

The Header component is implemented at the top of the page on line 14 (Figure 8.1). Users can click on the “List a space” button to go to the Account.js page if they wish to list a workspace.

The code consists of 2 sections. The first section of the code ranges from lines 18 to 43 (Figure 8.1), where text descriptions of the web app’s purpose are presented to the user. A button “Search Now” is available for users to click on which will redirect them to Results.js where they can begin searching for workspaces.

```
8  export default function Home() {
9
10   //const { renderSearchBar } = SearchBar();
11
12   return (
13     <>
14       <Header />
15
16       <div className="d-md-flex overflow-hidden">
17
18         <Container className="m-0 col-md-mt-5 pt-5 d-flex justify-content-between">
19           <Row className="d-flex flex-column">
20             <Col  className="d-flex mx-md-5 col-lg-10 px-5 flex-column gap-2 mb-4">
21               <h1 className="fw-bold text-black my-4 mx-xl-5 pb-4">
22                 Finding a space to get work done shouldn't be difficult.
23               </h1>
24
25               <h5 className="text-secondary mx-xl-5">Feeling unproductive from home?</h5>
26
27               <h5 className="text-secondary mx-xl-5">
28                 Don't want to commit to a working space?
29               </h5>
30
31               <h6 className="text-muted mx-xl-5 mt-5 pt-4">
32                 Enter a location near you to discover work-friendly spaces.
33               </h6>
34
35
36               <div className="mx-xl-5 ">
37                 <Link to="/results">
38                   <Button>Search Now</Button>
39                 </Link>
40               </div>
41             </Col>
42             </Row>
43           </Container>
```

Figure 8.1 - Code snippet from Home.js

The second section from lines 45 to 51 (Figure 82) is simply for the presentation of a picture on the home page.

```
44
45             <Container className="p-0 pt-5 pb-5 m-0 mt-3 d-none d-lg-block d-xl-block">
46               <Row className="pt-5 mt-5">
47                 <Col className="col-12">
48                   <img src={HomeImg} alt="home page img" className="h-100 w-100"/>
49                 </Col>
50               </Row>
51             </Container>
52
53         </div>
54     </>
55   );
56 }
```

Figure 8.2 - Code snippet from Home.js

5.2.2 Results.js

Results.js consist of three JS files which are “Results.js”, “ResultsModal.js” and “ResultsSetting.js”. “Results.js” consists of the codes to display the results page of the website. “ResultsModal.js” consists of the codes to display the modal when a result card is clicked on. “ResultsSetting.js” consist of HTTP requests to get information from the database.

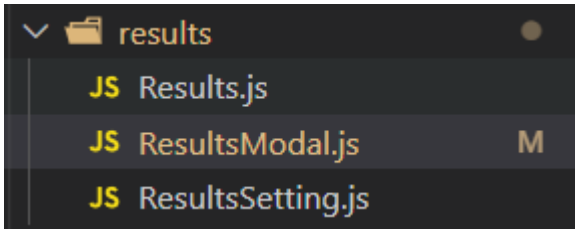


Figure 8.3 - File snippet from results folder

Line 6 and 7 declares “cardsettings” and “filterkeywords” to store response data from the HTTP request with “useState”.

```
6   const [cardsettings, setCardsettings] = useState([]);
7   const [filterkeywords, setFilterkeywords] = useState([]);
```

Figure 8.4 - Code snippet from ResultsSetting.js

Line 9 to 19 uses the “useEffect” hook to make a get HTTP request after rendering once. Line 10 to 13 and line 15 to 18 are the get HTTP requests that respond with the data from the database. Line 12 and 17 are where the responses are stored into the declared variable mentioned above.

```
9   useEffect(() => {
10    axios.get("http://localhost:3001/results/allListings")
11    .then(response => {
12      setCardsettings(response.data.map(listings => listings));
13    })
14
15    axios.get("http://localhost:3001/results/allFilterkeywords")
16    .then(response => {
17      setFilterkeywords(response.data.map(filterkeywords => filterkeywords));
18    })
19  }, []);
```

Figure 8.5 - Code snippet from ResultsSetting.js

Line 21 to 24 returns the variables where the data are stored so that the data can be used in “Results.js” when called.

```
21  return {
22    filterkeywords,
23    cardsettings
24  };
```

Figure 8.6 - Code snippet from ResultsSetting.js

After setting up the HTTP requests, the next step was to create the cards and carousel components. This was done in the shared folder which consists of “Cards.js” and “Carousel.js”.

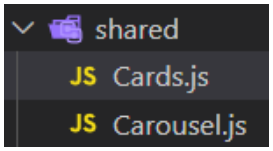


Figure 8.7 - File snippet from shared folder

Line 31 to 38 renders the card component to be displayed with the carousel component which displays the images. The carousel component can be seen in the “Carousel.js” section below.

```
31      <div>
32        <Card style={{maxWidth: "20em"}}>
33          <Card.Header className="p-0">
34            <Carouselcomponent
35              images={props.images}
36              alt="slides images"
37              />
38          </Card.Header>
```

Figure 8.10 - Code snippet from Cards.js

Line 40 to 42 consist of the name as title in the card component. Line 43 to 51 consist of the map icon and address of the space listed. Line 53 to 62 consist of the clock icon and the operation days/hours of the space listed.

```
40      <div onClick={() => setModalshow(true)}>
41        <Card.Body>
42          <Card.Title>{props.name}</Card.Title>
43            <div style={{fontSize: "13px"}}>
44              <div className="d-flex flex-row pb-2">
45                <Col xs={1}>
46                  <Map/>
47                </Col>
48                <Col xs={11}>
49                  {props.address}
50                </Col>
51              </div>
52
53              <div className="d-flex flex-row">
54                <Col xs={1}>
55                  <Clock/>
56                </Col>
57                <Col xs={11}>
58                  {operationDays}
59                  <br/>
60                  {props.operatingHours}
61                </Col>
62              </div>
```

Figure 8.11 - Code snippet from Cards.js

Line 64 to 86 consist of the badges component in the card component. The badges are mapped out based on the category and label to identify the correct icon and colour of the badge. Below is a snippet example of how the badges are being mapped.

```
64                  <div className="d-flex flex-wrap">
65                    {props.keywords.map(({label, category}) =>
66                      {
67                        if(category === "Facilities")
68                        {
69                          if(label === "Wifi")
70                          {
71                            return (
72                              <div key={label.length} className="p-1">
73                                <Badge bg="success">
74                                  <div>{label}</div><Wifi />
75                                </Badge>
76                              </div>
77                            );
78                          }
79                          else if(label === "Power source")
80                          {
81                            return (
82                              <div key={label.length} className="p-1">
83                                <Badge bg="success">
84                                <div>{label}</div><BatteryCharging />
85                                </Badge>
86                              </div>
```

Figure 8.12 - Code snippet from Cards.js

Line 143 to 157 consist of the function called from “ResultsModal.js”.

```
143         <ResultsModal
144           modalshow={modalshow}
145           handleClose={() => setModalshow(false)}
146           name={props.name}
147           description={props.description}
148           address={props.address}
149           operationDays={operationDays}
150           operatingHours={props.operatingHours}
151           contactNum={props.contactNum}
152           estate={props.estate}
153           website={props.website}
154           reservationUrl={props.reservationUrl}
155           keywords={props.keywords}
156           images={props.images}
157           />
```

Figure 8.13 - Code snippet from Cards.js

Line 6 to 27 consist of the carousel items which are the three images uploaded by the owners.

```
6   return(
7       <Carousel interval={null}>
8           <Carousel.Item>
9           <img className="mw-100 mh-100 rounded"
10              src = {"http://localhost:3000/assets/" + props.images.image1}
11              alt="First slide"
12          />
13          </Carousel.Item>
14
15          <Carousel.Item>
16          <img className="mw-100 mh-100 rounded"
17              src={"http://localhost:3000/assets/" + props.images.image2}
18              alt="Second slide"
19          />
20          </Carousel.Item>
21
22          <Carousel.Item>
23          <img className="mw-100 mh-100 rounded"
24              src={"http://localhost:3000/assets/" + props.images.image3}
25              alt="Third slide"
26          />
27          </Carousel.Item>
```

Figure 8.15 - Code snippet from Carousel.js

Line 20 to 34 consist of a declared array (“labels”) and a function to check if the category of the badges are correct before storing the labels.

```
20  let labels = [];
21
22  // badge checker based on category
23  function badge_check(){
24    for(var i=0; i < props.keywords.length; i++){
25      if(props.keywords[i].category === "Facilities"){
26        labels.push(props.keywords[i].label);
27      }
28      else if(props.keywords[i].category === "Amenities"){
29        labels.push(props.keywords[i].label);
30      }
31      else if(props.keywords[i].category === "Ambience"){
32        labels.push(props.keywords[i].label);
33      }
34    }
```

Figure 8.17 - Code snippet from ResultsModal.js

The method used to display the images, name, address, operational days/ hours, description, contact number and the website in “ResultsModal.js” is the same as “Cards.js”. Refer to the snippet provided above in “Cards.js”.

Line 14 to 21 consist of the function which handles the change when a checkbox is checked by the user.

```
14  const handleChange = (value, event) => {
15    if (event.target.checked) {
16      setCheckboxQuery(value);
17    } else {
18      setCheckboxQuery("");
19    }
20    setIsChecked(current => !current);
21  };
```

Figure 8.20 - Code snippet from Results.js

Line 25 to 28 consist of the search bar being called. Line 30 to 44 maps the response keywords from the database to be displayed.

```
25      <Header />
26      <Container className="py-3 d-flex justify-content-center align-items-center">
27        <Row>{renderSearchBar}</Row>
28      </Container>
29
30      <Container className="d-flex justify-content-center align-items-center">
31        <Row>
32          {
33            filterkeywords.map(({_id, label, value}) => (
34              <Col key={_id} xs={4} className="d-flex justify-content-center align-items-center p-2">
35                <div>
36                  <input
37                    type="checkbox"
38                    name={label}
39                    value={isChecked}
40                    onChange={(e) => handleChange(value, e)}
41                    />{" "}
42                  {label}
43                </div>
44              </Col>
```

Figure 8.21 - Code snippet from Results.js

Line 53 to 70 filters the response cardsetting from the database and validates the input from the users with the response. This part is to filter the results based on the estate input by the user.

```
53            cardsettings.filter(cardsetting =>{
54              for(var i=0; i<cardsetting.keywords.length; i++){
55                if (query === ""){
56                  if(checkboxQuery === ""){
57                    return cardsetting;
58                  }
59                  else if(cardsetting.keywords[i].label.toLowerCase().includes(checkboxQuery.toLowerCase())){
60                    return cardsetting;
61                  }
62                }
63                else if (cardsetting.estate.toLowerCase().includes(query.toLowerCase()))
64                {
65                  if(checkboxQuery === ""){
66                    return cardsetting;
67                  }
68                  else if(cardsetting.keywords[i].label.toLowerCase().includes(checkboxQuery.toLowerCase())){
69                    return cardsetting;
70                  }
```

Figure 8.22 - Code snippet from Results.js

Line 74 to 103 maps the cards in the “Results.js” based on the filtered value mentioned above in line 53 to 70. Values such as the name and description are passed to the card components here as well.

```
74          }).map((cardsetting) => (
75              <Col
76                key={cardsetting.name}
77                xs={4}
78                className="d-flex flex-wrap justify-content-center align-items-center p-3"
79                >
80                <Cardcomponent
81                  name={cardsetting.name}
82                  description={cardsetting.description}
83                  address={
84                    (cardsetting.address.streetOne) + " " +
85                    (cardsetting.address.streetTwo) + " " + "#" +
86                    (cardsetting.address.unitNum) + " " +
87                    (cardsetting.address.country) + " " +
88                    (cardsetting.address.postalCode)
89                    }
90                  daysClosed={cardsetting.operatingHours.daysClosed}
91                  operatingHours={
92                    (cardsetting.operatingHours.start) + ":00AM"  +
93                    " - " +
94                    (cardsetting.operatingHours.end) + ":00PM"
95                    }
96                  contactNum={cardsetting.contactNum}
97                  estate={cardsetting.estate}
98                  website={cardsetting.website}
99                  reservationUrl={cardsetting.reservationUrl}
100                 keywords={cardsetting.keywords}
101                 images={cardsetting.images}
102               />
103             </Col>
```

Figure 8.23 - Code snippet from Results.js

5.2.3 Account.js

These are the core functions on the account page:

1. A user can sign up for a new account.
2. An existing user can login.

Figure 8.24 shows the code snippet to address item 1. In line 33, axios runs with the configurations defined in line 23, sending the data to the database through its URL endpoint.

```
5   export default function SignUpModal(props) {
6     const [register, setRegister] = useState(false);
7     const [isLoading, setIsLoading] = useState(false);
8     const [error, setError] = useState("");
9
10    const fullNameEl = useRef();
11    const emailEl = useRef();
12    const passwordEl = useRef();
13
14    const handleSubmit = useCallback(
15      () => e => {
16        e.preventDefault();
17        setIsLoading(true);
18
19        const fullName = fullNameEl.current.value;
20        const email = emailEl.current.value;
21        const password = passwordEl.current.value;
22
23        const config = {
24          method: "post",
25          url: "https://workgowhere.herokuapp.com/public-user/register",
26          data: {
27            fullName,
28            email,
29            password,
30          },
31        };
32
33        axios(config)
34          .then(result => {
35            console.log(result);
36            setRegister(true);
37            setIsLoading(false);
38          })
39          .catch(err => {
40            console.log(err);
41            setError(err);
42            setIsLoading(false);
43          });
44      },
45      []
46    );
```

Figure 8.24 - Code snippet from SignUpModal.js

Figure 8.25 shows the code snippet to address item 2. The axios function on line 33 runs very similarly to the Sign Up function mentioned above. However, an additional cookie will be set to the user's browser upon successful login. This is for authentication purposes because if a user tries to bypass the login page to enter pages that are protected, it will fail because there will be no matching cookie that is tied to a logged in user. In addition, the page will also redirect the user to the dashboard page after logging in.

```
8   export default function SignInForm() {
9     const [login, setLogin] = useState(false);
10    const [isLoading, setIsLoading] = useState(false);
11    const [error, setError] = useState("");
12
13    const emailEl = useRef();
14    const passwordEl = useRef();
15
16    const handleSubmit = useCallback(
17      () => e => {
18        e.preventDefault();
19        setIsLoading(true);
20
21        const email = emailEl.current?.value;
22        const password = passwordEl.current?.value;
23
24        const config = {
25          method: "post",
26          url: "https://workgowhere.herokuapp.com/public-user/login",
27          data: {
28            email,
29            password,
30          },
31        };
32
33        axios(config)
34          .then(result => {
35            setLogin(true);
36            setIsLoading(false);
37
38            cookies.set("TOKEN", result.data.token, {
39              path: "/",
40            });
41            // console.log(result);
42            window.location.href = `/user/dashboard/${result.data.user._id}`;
43          })
44          .catch(err => {
45            setIsLoading(false);
46            setError(err.response.data.message);
47            // console.log(err);
48          });
49      },
50      []
51    );
```

Figure 8.25 - Code snippet from SignInModal.js

5.2.4 Dashboard.js

These are the core functions on the dashboard page:

1. Fetch user's name who just signed in.
2. Retrieve all listings data under that user.
3. Render listings data in an accordion-styled dashboard.
4. Delete an existing listing.
5. Button to logout of the user's account.

Figure 8.26 shows the code snippet to address items 1 and 2 in the above list. In line 27 to 34, axios is in-charge of fetching the data with the given configuration, with the right HTTP method, back end API endpoint and authorisation headers. Figure 8.26 also shows that the axios function is wrapped in a useEffect hook because we would want to start fetching data on the first page load.

```
13  export default function Dashboard() {
14    const [name, setName] = useState("");
15    const [listingArr, setListingArr] = useState([]);
16    let { id } = useParams();
17
18    useEffect(() => {
19      const config = {
20        method: "get",
21        url: `https://workgowhere.herokuapp.com/private-user/dashboard/${id}`,
22        headers: {
23          Authorization: `Bearer ${token}`,
24        },
25      };
26
27      axios(config)
28        .then(result => {
29          // console.log(result);
30          setName(result.data.user.fullName);
31          setListingArr(result.data.user.listingsPosted);
32        })
33        .catch(err => {
34          console.log(err);
35        });
36    }, [id]);
```

Figure 8.26 - Code snippet from Dashboard.js

Figures 8.27 and 8.28 show the code snippet to address item 3. Once data is fetched, it is stored in a variable called “listingArr”. With a conditional operator, we check the length of listingArr. If it is more than 0, then we render the header of the accordion and pass the listingArr as a property into ListingAccordion component (the body of the accordion), else, we render a message to the user as feedback.

```
75        {listingArr.length > 0 ? (
76          <>
77            <Container className="d-none d-md-block d-lg-block d-xl-block">
78              <Container>
79                <Row className="border rounded p-2 bg-black text-white">
80                  <Col>ID</Col>
81                  <Col>Name of Space</Col>
82                  <Col>Created At</Col>
83                  <Col>Status</Col>
84                </Row>
85              </Container>
86            </Container>
87            <ListingAccordion listingArr={listingArr} />
88          </>
89        ) : (
90          <p className="text-black text-center m-4">
91            No listing(s) found. List one now.
92          </p>
93        )}
94      </>
95    );
96  }
```

Figure 8.27 - Code snippet from Dashboard.js

With the property received, the ListingAccordion component takes that property and maps out each individual listing data (line 40) into each subcomponent. From Figure 8.28, beyond line 56 is the continuation of the accordion body.

```
39        <Container>
40          {props.listingArr.map((list, index) => {
41            return (
42              <div key={list._id}>
43                <Accordion alwaysOpen>
44                  <Accordion.Item eventKey={index}>
45                    <Accordion.Header>
46                      <Container>
47                        <Row className="align-items-center">
48                          <Col className="d-none d-md-block d-lg-block d-xl-block">
49                            <div>{list._id}</div>
50                          </Col>
51                          <Col>
52                            <div>{list.name}</div>
53                          </Col>
54                          <Col>
55                            <div>{list.createdAt}</div>
56                          </Col>
```

Figure 8.28 - Code snippet from ListingAccordion.js

Figure 8.29 shows the code snippet to address item 4. The handleDelete function from line 15 receives the listingId as a parameter to ensure that once the user clicks on the delete button, it will delete the correct listing. Axios will then run a delete method based on its configuration defined in line 20.

```
12  export default function ListingAccordion(props) {
13    let { id } = useParams();
14
15    function handleDelete(e, listingId) {
16      e.stopPropagation();
17      props.listingArr.filter(listing => listing.id !== listingId);
18      console.log(`Deleted listing ${listingId}`);
19
20      const config = {
21        method: "delete",
22        url: `https://workgowhere.herokuapp.com/private-user/delete/${id}/${listingId}`,
23        headers: {
24          Authorization: `Bearer ${token}`,
25        },
26      };
27      axios(config)
28        .then(result => {
29          console.log("Deleted successfully", result);
30          window.location.reload(true);
31        })
32        .catch(err => {
33          console.log(err);
34        });
35    }
```

Figure 8.29 - Code snippet from ListingAccordion.js

Figure 8.30 shows the code snippet to address item 5. A simple logout function is defined in line 38. This function destroys the existing cookie assigned upon login, and redirects the user back to the account page.

```
38    function logout() {
39      // Destroy existing cookie
40      cookies.remove("TOKEN", { path: "/" });
41      // Redirect user to landing page
42      window.location.href = "/account";
43    }
```

Figure 8.30 - Code snippet from Dashboard.js

5.2.5 UserForm.js

onFromUpdate updates the form when the user inputs new data.

```
38  // Upates form with textfield input
39  const onFormUpdate = (category, value) => {
40    setFormDetails({
41      ...formDetails,
42      [category]: value,
43    });
44  };
```

Figure 8.34 - Code snippet from UserForm.js

handleChange updates the dayClosed variable in formInitialDetails when users interact with the checkbox. The function uses event.target.check to check the status of the checkbox. If it is an empty box that is checked, it will push the item into the daysChecked array. Otherwise, it will remove the item from the array using the splice method.

```
46  // Updates form with daysClosed checkbox input
47  const handleChange = (item, event) => {
48    const isChecked = event.target.checked;
49
50    if (isChecked) {
51      daysChecked.push(item);
52    } else {
53      const index = daysChecked.indexOf(event.target.value);
54      daysChecked.splice(index, 1);
55    }
56
57    onFormUpdate("daysClosed", daysChecked);
58    setIsChecked(current => !current);
59  };
```

Figure 8.35 - Code snippet from UserForm.js

handleKeywordsChange works similar to handleChange. The function triggers when users check the boxes in the keywords section. Rather than pushing a single instance of an object in the keywordsChecked array, it pushes a dictionary.

```
61  // Updates form with keywords checkbox input
62  const handleKeywordsChange = (item, event) => {
63    const isChecked = event.target.checked;
64
65    if (isChecked) {
66      if (item === "Wifi" || item === "Power source") {
67        keywordsChecked.push({ label: item, category: "Facilities" });
68      } else if (
69        item === "Toilet" ||
70        item === "Parking" ||
71        item === "Handicap friendly"
72      ) {
73        keywordsChecked.push({ label: item, category: "Amenities" });
74      } else if (
75        item === "Quiet" ||
76        item === "Noisy" ||
77        item === "Discussion friendly"
78      ) {
79        keywordsChecked.push({ label: item, category: "Ambience" });
80      }
81    } else {
82      setKeywordsChecked(keywordsChecked.filter(items => items.label !== item));
83    }
84
85    onFormUpdate("keywords", keywordsChecked);
86    setIsChecked(current => !current);
87  };
```

Figure 8.36 - Code snippet from UserForm.js

handleSubmit checks the form validity before sending the data into the database.

```
89  // Checks form validity when user submits
90  const handleSubmit = event => {
91    const form = event.currentTarget;
92
93    if (form.checkValidity() === false) {
94      event.preventDefault();
95      event.stopPropagation();
96    }
97    setFormDetails(formInitialDetails);
98    setValidated(true);
99  };
```

Figure 8.37 - Code snippet from UserForm.js

toPreview has two functions. It passes the form data into the preview page and saves the image data into the database using axios.

```
// Passes form data to preview page
const toPreview = () => {
  const imageData1 = new FormData();
  const imageData2 = new FormData();
  const imageData3 = new FormData();

  imageData1.append("file", Object.values(imagefiles)[0]);
  imageData2.append("file", Object.values(imagefiles)[1]);
  imageData3.append("file", Object.values(imagefiles)[2]);

  axios.post("http://localhost:8000/upload", imageData1).then(res => {
    console.log(res.statusText);
  });
  axios.post("http://localhost:8000/upload", imageData2).then(res => {
    console.log(res.statusText);
  });
  axios.post("http://localhost:8000/upload", imageData3).then(res => {
    console.log(res.statusText);
  });

  navigate(`/preview/${id}`, {
    state: {
      name: formDetails.name,
      description: formDetails.description,
      address: {
        streetOne: formDetails.streetOne,
        streetTwo: formDetails.streetTwo,
        unitNum: formDetails.unitNum,
        country: "Singapore",
        postalCode: formDetails.postalCode,
      },
      operatingHours: {
        start: formDetails.start,
        end: formDetails.end,
        daysClosed: formDetails.daysClosed,
      },
      contactNum: formDetails.contactNum,
      estate: formDetails.estate,
      website: formDetails.website,
      reservationUrl: formDetails.reservationUrl,
      keywords: formDetails.keywords,
      imageUrl: imageUrl,
      images: formDetails.images,
      imagefiles: imagefiles,
    },
  });
};
```

Figure 8.38 - Code snippet from UserForm.js

```
131   const imagePrev = event => {
132     var Url1 = URL.createObjectURL(event.target.files[0]);
133     var Url2 = URL.createObjectURL(event.target.files[1]);
134     var Url3 = URL.createObjectURL(event.target.files[2]);
135
136     var image1 = event.target.files[0].name;
137     var image2 = event.target.files[1].name;
138     var image3 = event.target.files[2].name;
139
140     var file1 = event.target.files[0];
141     var file2 = event.target.files[1];
142     var file3 = event.target.files[2];
143
144     var allImageURL = { Url1, Url2, Url3 };
145     var allImageName = { image1, image2, image3 };
146     var allImageFiles = { file1, file2, file3 };
147
148     setImageUrl(allImageURL);
149     setImagefiles(allImageFiles);
150     onFormUpdate("images", allImageName);
151   };
```

Figure 8.39 - Code snippet from UserForm.js

After line 197 contains all the code pertaining to the form.

```
197   return (
198     <>
199       <section className="listing">
200         <Container>
201           <h2 className="py-3">Fill up the details of your space here.</h2>
202
203           <Form noValidate validated={validated} onSubmit={handleSubmit}>
204             <Row className="mb-3">
205               <Col md={4}>
206                 <Form.Group controlId="validationCustom01">
207                   <Form.Label>Name of space</Form.Label>
208                   <Form.Control
209                     required
210                     type="text"
211                     placeholder="Name"
212                     onChange={e => onFormUpdate("name", e.target.value)}
213                     value={formDetails.name}
214                   />
215                   <Form.Control.Feedback>Looks good!</Form.Control.Feedback>
216                 </Form.Group>
217
218                 <Form.Group controlId="validationCustom02">
219                   <Form.Label>Description</Form.Label>
220                   <Form.Control
221                     required
222                     as="textarea"
223                     rows={3}
224                     placeholder="Description"
225                     onChange={e => onFormUpdate("description", e.target.value)}
226                     value={formDetails.description}
227                   />
228                   <Form.Control.Feedback>Looks good!</Form.Control.Feedback>
229                 </Form.Group>
230
231                 <Form.Group controlId="validationCustom01">
232                   <Form.Label>Address</Form.Label>
233                   <Form.Control
234                     required
```

Figure 8.41 - Code snippet from UserForm.js

5.2.6 Preview.js

userFormDetails retrieves the data for each variable using the useLocation function from UserForm.js.

```
export default function Preview() {
    const location = useLocation();
    const cookies = new Cookies();
    const token = cookies.get("TOKEN");

    let { id } = useParams();

    const userFormDetails = {
        name: location.state.name,
        description: location.state.description,
        address: {
            streetOne: location.state.address.streetOne,
            streetTwo: location.state.address.streetTwo,
            unitNum: location.state.address.unitNum,
            country: "Singapore",
            postalCode: location.state.address.postalCode,
        },
        operatingHours: {
            start: location.state.operatingHours.start,
            end: location.state.operatingHours.end,
            daysClosed: location.state.operatingHours.daysClosed
        },
        contactNum: location.state.contactNum,
        estate: location.state.estate,
        website: location.state.website,
        reservationUrl: location.state.reservationUrl,
        keywords: location.state.keywords,
        images: location.state.images
    }
```

Figure 8.43 - Code snippet from Preview.js

The SubmitForm function uploads all the data from the form to the database. We configure the method, url, headers and data into axios to post form information.

```
const imageUrl = { imageUrl: location.state.imageUrl };

const SubmitForm = e => {
  e.preventDefault();
  const config = {
    method: "post",
    url: `https://workgowhere.herokuapp.com/private-user/new-listing/${id}`,
    headers: {
      Authorization: `Bearer ${token}`,
    },
    data: userFormDetails,
  };

  axios(config).then(res => {
    window.location.href = "/thankyou";
    console.log(res.data);
  });
};
```

Figure 8.44 - Code snippet from Preview.js

Line 67 to 94 uses the SmallListingCard component. Data is retrieved from userFormDetails and passed into the component to render the card.

```
67          <Row>
68            <Col className="col-4">
69              <SmallListingCard
70                name={userFormDetails.name}
71                address={
72                  userFormDetails.address.streetOne +
73                  " " +
74                  userFormDetails.address.streetTwo +
75                  " " +
76                  "#" +
77                  userFormDetails.address.unitNum +
78                  " " +
79                  userFormDetails.address.country +
80                  " " +
81                  userFormDetails.address.postalCode
82                }
83                daysClosed={userFormDetails.operatingHours.daysClosed}
84                operatingHours={
85                  userFormDetails.operatingHours.start +
86                  ":00AM" +
87                  " - " +
88                  userFormDetails.operatingHours.end +
89                  ":00PM"
90                }
91                keywords={userFormDetails.keywords}
92                images={imageUrl.imageUrl}
93              />
94            </Col>
```

Figure 8.46 - Code snippet from Preview.js

Likewise for LargeListingCard.

```
95          <Col className="col-8">
96            <LargeListingCard
97              name={userFormDetails.name}
98              description={userFormDetails.description}
99              address={
100               userFormDetails.address.streetOne +
101               " " +
102               userFormDetails.address.streetTwo +
103               " " +
104               "#" +
105               userFormDetails.address.unitNum +
106               " " +
107               userFormDetails.address.country +
108               " " +
109               userFormDetails.address.postalCode
110             }
111             daysClosed={userFormDetails.operatingHours.daysClosed}
112             operatingHours={
113               userFormDetails.operatingHours.start +
114               ":00AM" +
115               " - " +
116               userFormDetails.operatingHours.end +
117               ":00PM"
118             }
119             contactNum={userFormDetails.contactNum}
120             estate={userFormDetails.estate}
121             website={userFormDetails.website}
122             reservationUrl={userFormDetails.reservationUrl}
123             keywords={userFormDetails.keywords}
124             images={imageUrl.imageUrl}
125           />
126         </Col>
```

Figure 8.47 - Code snippet from Preview.js

5.2.7 ThankYou.js

Lines 1 to 7 of Thankyou.js imports the necessary components and resources. Lines 1 to 3 import libraries and components such as “Button” and “Container”. Lines 3 to 7 import dependencies for stylising our page.

```
1  import React from "react";
2  import { Link } from "react-router-dom";
3  import { Button, Container, Row, Col } from "react-bootstrap";
4  import ThankYouImg from "../../../assets/thankyou.png";
5  import Logo from "../../../assets/logo.svg";
6  import TeamPic from "../../../assets/640x360.png";
7  import Header from "../../../components/shared/Header";
```

Figure 8.48 - Code snippet from Thankyou.js

The Thankyou.js code consists of 2 main sections.

Lines 17 to 66 in Figure 8.49 and Figure 8.50 is the first section, the “Thank you” section. Lines 68 to 125 in Figure 8.51 and Figure 8.52 contain the second section, the “Meet the Team” section.

```
10 export default function Thankyou() {
11
12   return (
13     <>
14
15     <Header />
16
17     <div className="d-md-flex overflow-hidden d-none d-md-block">
18       <Container className="m-0 d-flex justify-content-between" lg={6} xl={6}>
19         <Row className="d-flex flex-column">
20           <Col className="d-flex px-5 mx-xl-3 flex-column align-items-center justify-content-center mb-4">
21             <h1 className="fw-bold text-black text-center mx-md-5 pl-md-5 pb-5">
22               Thank you for listing with us!
23             </h1>
24             <h3 className="text-muted text-center pb-3">Meet the team below! <br/><br/> or </h3>
25             <Link to={`/account`}>
26             <Button>List Another Space</Button>
27             </Link>
28             {/* <Button
29               variant="outline-primary"
30             >
31               Meet The Team
32             </Button> */}
33           </Col>
34
35         </Row>
36       </Container>
37
38       <Container className="p-0 m-0 d-none d-md-block">
39         <Row>
40           <Col>
41             <img src={ThankYouImg} alt="thank you page img" className="w-100" />
42           </Col>
43         </Row>
44       </Container>
45     </div>
```

Figure 8.49 - Code snippet from Thankyou.js

In the first section, lines 17 to 33 (Figure 8.49) contain code that displays the thank you message and another message informing users they can learn more about the development team below. There is also a button to allow users to list another space by redirecting them to Account.js (as seen on line 25). A picture will be on the right side on screens of size lg and greater (lg and xl).

Lines 47 to 66 (Figure 8.50) contain code that better optimises the user's visual experience for smaller, mobile screens.

```
37
38      <Container className="p-0 m-0 d-none d-md-block">
39        <Row>
40          <Col>
41            <img src={ThankYouImg} alt="thank you page img" className="w-100" />
42          </Col>
43        </Row>
44      </Container>
45    </div>
46
47      <Container className="d-md-none">
48        <Row className="d-flex flex-column">
49          <Col className="d-flex px-5 flex-column align-items-center justify-content-center gap-2 mb-4">
50
51            <h1 className="fw-bold text-black text-center">
52              Thank you for listing with us!
53            </h1>
54            <Button>List Another Space</Button>
55            <h6 className="text-muted text-center">or</h6>
56            <Link to={`/account`}>
57            <Button>List Another Space</Button>
58            </Link>
59            {/* <Button
60              variant="outline-primary"
61            >
62              Meet The Team
63            </Button> */}
64          </Col>
65        </Row>
66      </Container>
```

Figure 8.50 - Code snippet from Thankyou.js

In the second section, lines 68 to 89 (Figure 8.51) contain code that displays a message stating “Meet the team behind”, followed by a picture of WorkGoWhere’s logo. Lines 91 to 116 (Figure 8.52) contains code that displays images, names and descriptions of each team member in a row, allowing users to learn about the developers. Lines 118 to 125 (Figure 8.52) is code for the “Report a feedback” button, in which features have yet to be implemented at the current stage.

```
68      <Container>
69
70        <Row className="row justify-content-center p-3">
71          <Col className="col-auto">
72            <h3 className="fw-bold text-black text-center">
73              Meet the team behind
74            </h3>
75          </Col>
76
77          <Col className="col-auto d-none d-md-block">
78            <img src={Logo} alt="logo" />
79          </Col>
80
81        </Row>
82
83
84        <Row className="row justify-content-center p-2 my-1 d-md-none">
85
86          <Col className="col-auto">
87            <img src={Logo} alt="logo" />
88          </Col>
89        </Row>
```

Figure 8.51 - Code snippet from Thankyou.js

```
91        <Row className="row justify-content-center p-5">
92          <Col className="d-flex justify-content-center flex-column align-items-center">
93            <img src={TeamPic} alt="team" width="200px" className="my-1" />
94
95            <h5 className="fw-bold text-black text-center">Austin Lian</h5>
96            <h6 className="text-muted text-center">Some description</h6>
97          </Col>
98          <Col className="d-flex justify-content-center flex-column align-items-center">
99            <img src={TeamPic} alt="team" width="200px" className="my-1" />
100
101           <h5 className="fw-bold text-black text-center">Nicol Chen</h5>
102           <h6 className="text-muted text-center">Some description</h6>
103         </Col>
104         <Col className="d-flex justify-content-center flex-column align-items-center">
105           <img src={TeamPic} alt="team" width="200px" className="my-1" />
106
107           <h5 className="fw-bold text-black text-center">Tricia Chow</h5>
108           <h6 className="text-muted text-center">Some description</h6>
109         </Col>
110         <Col className="d-flex justify-content-center flex-column align-items-center">
111           <img src={TeamPic} alt="team" width="200px" className="my-1" />
112
113           <h5 className="fw-bold text-black text-center">Winson Chow</h5>
114           <h6 className="text-muted text-center">Some description</h6>
115         </Col>
116       </Row>
117
118       <Row>
119         <Col className="d-flex px-5 mt-3 flex-column align-items-center justify-content-center gap-2 mb-4">
120           <button className="btn btn-link btn-lg" type="submit">
121             <h5 className="fw-bold text-primary">Report a feedback</h5>
122           </button>
123         </Col>
124       </Row>
125     </Container>
126   </>
127 );
128 }
```

Figure 8.52 - Code snippet from Thankyou.js

5.3 Back End Breakdown

5.3.1 app.js

Figure 9.0 shows a code snippet from app.js where the server connection is configured to be live at http://localhost:3001/. This is an essential first step to ensure the back end is live.

```
17  app.listen(process.env.PORT || 3001, () => {
18    console.log("Server is running...");
19  });
```

Figure 9.0 - Server setup

Figure 9.1 shows a snippet that is responsible for the database connection to MongoDB. In line 22, mongoString represents the MongoDB URI that is kept private in a .env file.

```
7   const mongoString = process.env.MONGODB_URI;
```

```
21  // Connection to mongoDB
22  mongoose.connect(mongoString, {
23    useNewUrlParser: true,
24    useUnifiedTopology: true,
25  });
26
27  // Connect to DB and throw any error if connection fails
28  database.on("error", err => {
29    console.log(`Unable to connect to database. ${err}`);
30  });
31  // .once will only run one time
32  database.once("connected", () => {
33    console.log("Database connected!");
34  });
```

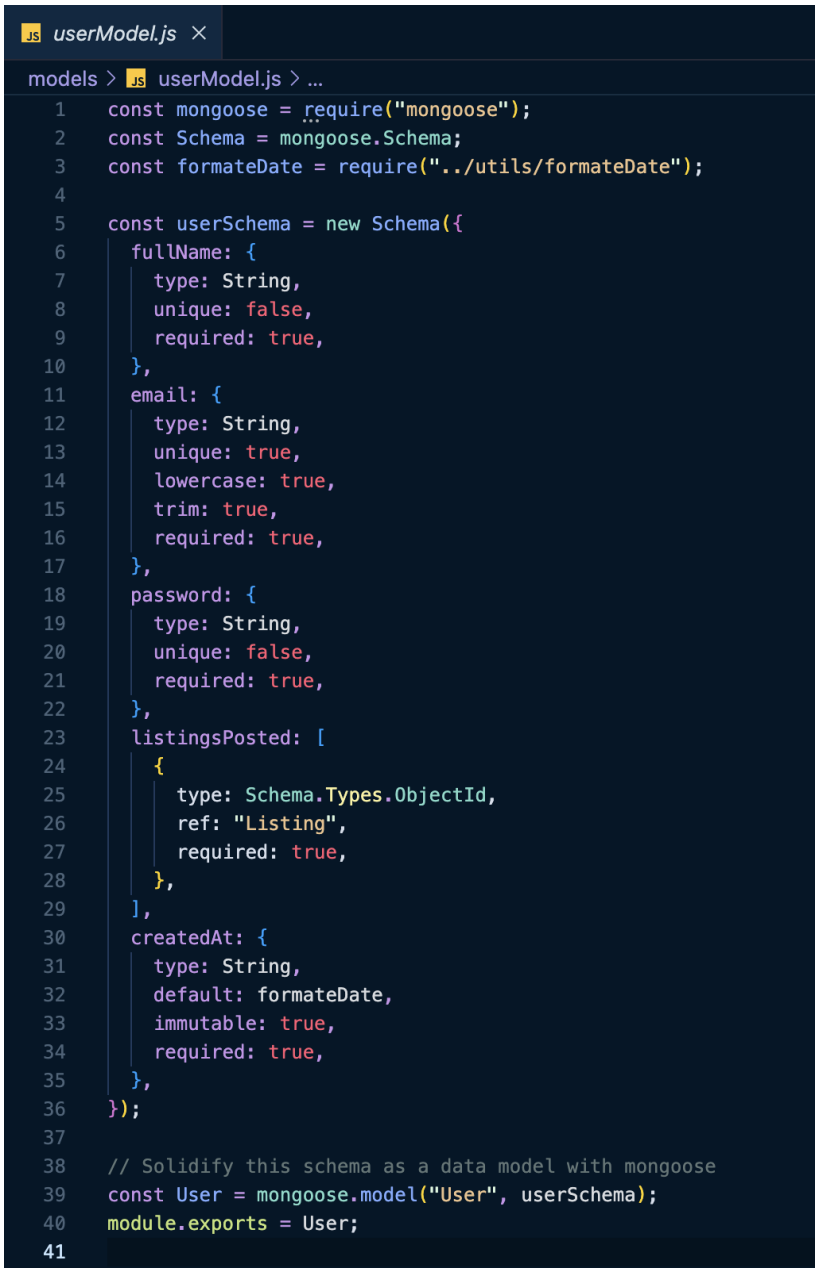
Figure 9.1 - Database connection

Figure 9.2 shows the section where the server connects all the routes created that handles every type of request such as get, post and delete requests.

```
36  // Connect all the routes
37  app.get("/", (req, res) => {
38    res.json({ message: "Your server is live!" });
39  });
40  app.use("/private-user", auth, privateRoutes);
41  app.use("/public-user", publicRoutes);
42  app.use("/results", resultsRoute);
```

Figure 9.2 - Routes connection

5.3.2 /models Folder



This folder stores the files that define database schema, namely: userModel.js, listingModel.js.

Figure 9.3 shows the User schema and its attributes that define a valid user.

The Listing schema will not be demonstrated as the structure is similar, but with different attributes.

Figure 9.3 - Schema for a user

5.3.3 /routes Folder

This folder contains all the endpoints of the web app that utilises HTTP method routes like get, post, delete, put and so on. These methods are accessed through the router object from Express and they are structured to be isolated from the rest of the application, precisely running the middleware and routing functions only.

Figure 9.4 shows one of the three files contained in the /routes folder. Here, we can see that the get, post and delete methods are being employed with their specific endpoints.

Figure 9.4 - Code snippet from privateRoute.js

5.3.4 /controllers Folder

The files in the controllers folder accept the user requests and interact with the model. The controller files contain code that communicates to the model what actions should be taken from the database.

Figure 9.5 shows one of the three files contained in the /controllers folder. It shows a snippet of the publicController.js file where the code is written to create a new instance of a user in the database.

Figure 9.5 - Code snippet of publicController.js

5.3.5 /utils Folder

“Utils” stands for “utility” and this folder contains files that create special functions that provide an addition to another part of the program. For instance, Figure 9.6 is an example of a small helper function that helps to reformat a date to dd/mm/yyyy, and it can be imported into any files that need this particular function. The /utils folder also contains files like auth.js and estates.js.

Figure 9.6 - Code snippet of formateDate.js

5.4 Test Driven Development

The following two subsections are some examples of the application's unit testing, mainly on the Account.js and Dashboard.js files.

5.4.3 Account.test.js

To ensure the account page is working as expected, we conduct a few tests for validation. From Figure 10.0, the first test block from line 19 to 26 ensures that the "List Now" and "Login" buttons are not disabled upon rendering on the browser. The rationale was to ensure these essential buttons behaved normally, otherwise, the account page would be obsolete. The second and third test block from line 28 to 45 ensures that axios would be able to execute a post method by defining a mock axios connection in line 8.

```
8   jest.mock("axios");
9
10  const MockedAccount = () => {
11    return (
12      <BrowserRouter>
13        <Account />
14      </BrowserRouter>
15    );
16  };
17
18  describe("Account", () => {
19    it("should render Account page correctly", () => {
20      render(<MockedAccount />);
21      const listNowBtn = screen.getByRole("button", { name: /list now/i });
22      const loginBtn = screen.getByRole("button", { name: /login/i });
23
24      expect(listNowBtn).not.toBeDisabled();
25      expect(loginBtn).not.toBeDisabled();
26    });
27
28    it("should sign up a new user", async () => {
29      axios.post.mockImplementationOnce(() => Promise.resolve(mockNewUser));
30      await expect(postSignUp()).resolves.toEqual(mockNewUser);
31      expect(axios.post).toHaveBeenCalledWith(
32        "https://workgowhere.herokuapp.com/public-user/register",
33        mockNewUser
34      );
35    });
36
37    it("should sign in a user", async () => {
38      axios.post.mockImplementationOnce(() => Promise.resolve(mockExistingUser));
39      await expect(postSignIn()).resolves.toEqual(mockExistingUser);
40      expect(axios.post).toHaveBeenCalledWith(
41        "https://workgowhere.herokuapp.com/public-user/login",
42        mockExistingUser
43      );
44    });
45  });
```

Figure 10.0 - Code snippet of Account.test.js

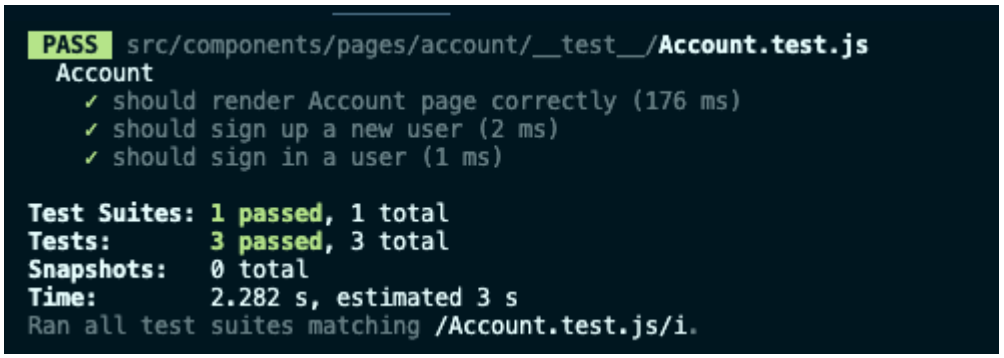


Figure 10.1 - Code snippet of Account.test.js's test result in console

5.4.4 Dashboard.test.js

Since the main function of the dashboard page is to be able to fetch data from the database, we test the axios get method to ensure it is able to do so. Similar to the test block in the above section, we utilised a mock axios connection to fetch data.

```
7   jest.mock("axios");
8
9   describe("Dashboard", () => {
10    it("should fetch user listing", async () => {
11      await axios.get.mockResolvedValue(mockFetchUserListing);
12      const data = await getUserListing();
13      expect(data).toEqual(mockFetchUserListing);
14      axios.get.mockImplementationOnce(() =>
15        Promise.resolve(mockFetchUserListing)
16      );
17      await expect(getUserListing()).resolves.toEqual(mockFetchUserListing);
18      expect(axios.get).toHaveBeenCalledWith(
19        "https://workgowhere.herokuapp.com/private-user/dashboard/:id",
20        mockFetchUserListing
21      );
22    });
23  });
```

Figure 10.2 - Code snippet of Dashboard.test.js

```
PASS src/components/pages/dashboard/__test__/Dashboard.test.js
API (variable)
    ✓ should fetch user listing (5 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.148 s
Ran all test suites matching /Dashboard.test.js/i.
```

Figure 10.3 - Code snippet of Dashboard.test.js's test result in console

5.5 Usability Testing

For the usability test, we will be using the system usability scale (SUS) (Brooke, n.d.) where testers are given a list of 10 questions which they rate using a 5 point scale after using our website. We chose to use SUS as it has been cited in more than 1300 journals and publications, making it an industry standard. The use of SUS has numerous advantages, including that it:

- Is a fairly simple scale to use when giving participants
- Has dependable outcomes even with tiny sample sizes
- Is reliable because it can distinguish between systems that are useable and those that are not

#	Question
1	I think that I would like to use the system frequently
2	I found the system unnecessary complex
3	I thought the system was easy to use
4	I think that I would need the support of a technical person to use this system
5	I found the various functions of this system were well integrated
6	I thought there were too much inconsistency in this system
7	I would imagine that most people would learn to use this system very quickly
8	I found the system very cumbersome to use
9	I felt confident using this system
10	I needed to learn a lot of things before I could get going with this system

Table 8.0 - SUS table of questions

SUS generates a single number that serves as an indicator of the system overall usability.

Please take note that individual item scores are not significant on their own. We first add the score contributions from each item before calculating the SUS score. The scoring contribution for each item will vary from 0 to 4. To determine the overall value of SU, multiply the total score by 2.5. SUS ratings range from 0 to 100.

Tester	Score	SUS score
1	35	35 * 2.5 = 87.5
2	29	29 * 2.5 = 72.5
3	31	31 * 2.5 = 77.5

Table 8.1 - SUS score tabulated

6.0 Analysis

Earlier in the project, our group had performed a competitor analysis which was detailed in our mid-term proposal. Our closest competitors would turn out to be blogs and articles which recommended remote workspaces for users.

The blogs and articles we had analysed tended to be in formats of a list, with descriptions of the places and additional contact information. We would then design our web app, WorkGoWhere, with the core strengths and weaknesses of these competitors in mind.

6.1 Common flaws and Improvements

Most of these blogs/articles are presented on static websites with minimal interaction.

They tend to lack any form of user interaction such as search options or filters for their lists as seen in Figures 11.0 and 11.1. Tending to only provide information, the user may need to manually search through them one by one to find options that they may be interested in. This can be another inconvenience when the user only desires options that fulfil their needs; for instance, all of the workspaces recommended may not be at locations that are convenient or accessible to the user, thus the blog/article may not have helped the user at all.


COUNSELLING

23 Best Free Places to Study in Singapore With Wifi and Power Sockets

By Shahzaad Kamboh

Last Updated: April 22, 2022

TWEET 11 COMMENTS

If you are looking for the best places to study in Singapore,

..then you'll love to check this post.

These are some good spots to head to in Singapore when you want to get things done.

But every time either you run out of motivation or things start pissing you off.

So, worry not.

It includes conducive free places of Singapore like libraries. Some quiet coworking spaces and also 24 hours open cafes to study in Singapore.

Just filter these study spots that describe your mood.

The Best Study Spots in Singapore

Best Student Friendly Cafes of Singapore
Best Study Places in Singapore Libraries
Best 24 Hour Places to Study in Singapore
Best Studying Places in Singapore Community Centres
Best Coworking Places in Singapore to Study and Work

Figure 11.0 - Snippet of Frostfairs (Kamboh, 2022)

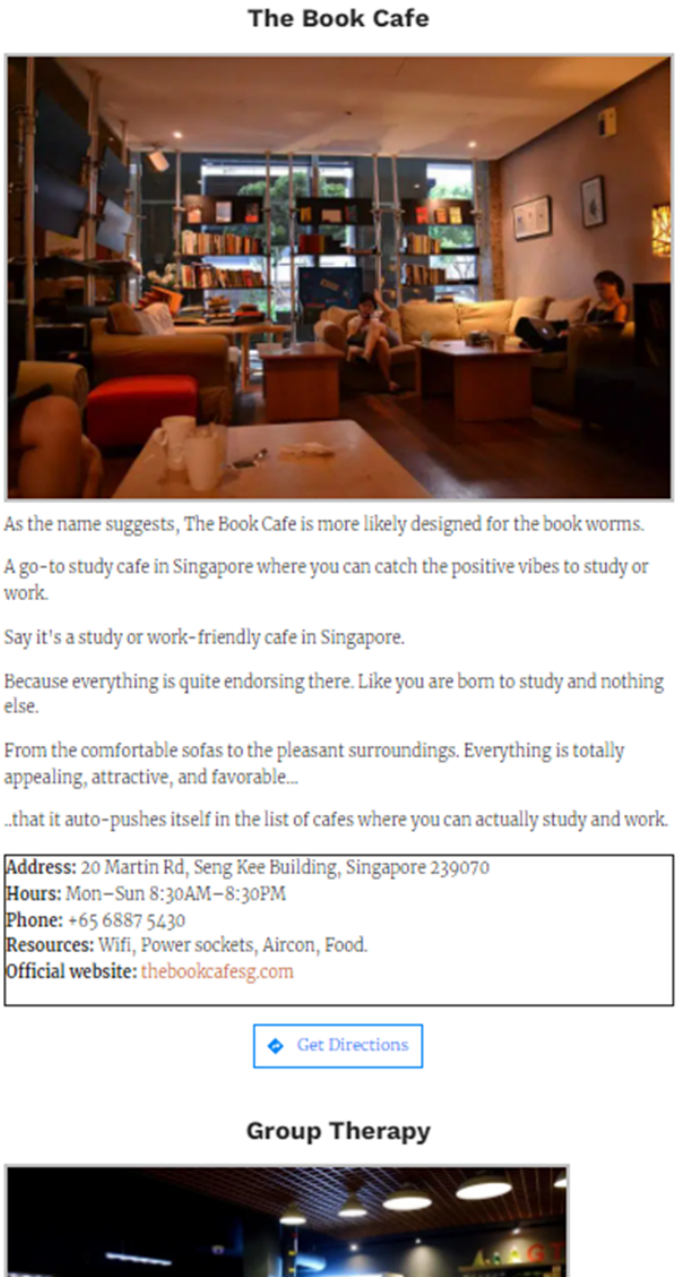


Figure 11.1 - Snippet of Frostfairs (Kamboh, 2022)

Additionally, some of the blogs/articles found were a few years old and likely not updated (Figure 11.1). The establishments recommended by them may not be available anymore or have changed.

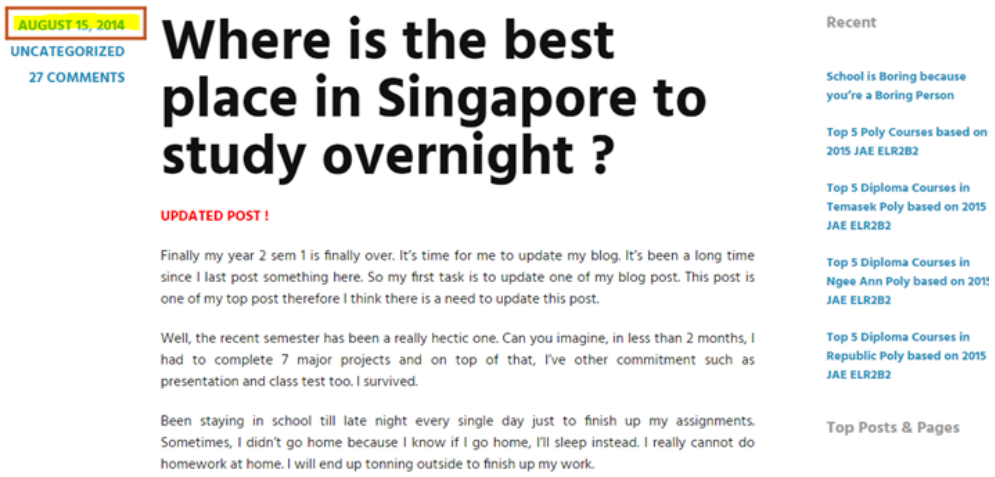


Figure 11.2 - Snippet of Blogwithlutefield (Field, n.d.)

WorkGoWhere has overcome these problems that our competitors face by dynamically referencing content from an external database that can be retrieved from and updated.

The web app content is generated in real-time and displays content relevant to the user based on their inputs. This can range from recommended locations based on user search and filter settings to how the data appears on devices of different resolutions. Clients can also use the feature in our web app that allows them to list workspaces, which will regularly add more content for consumers and circumvents outdated information.

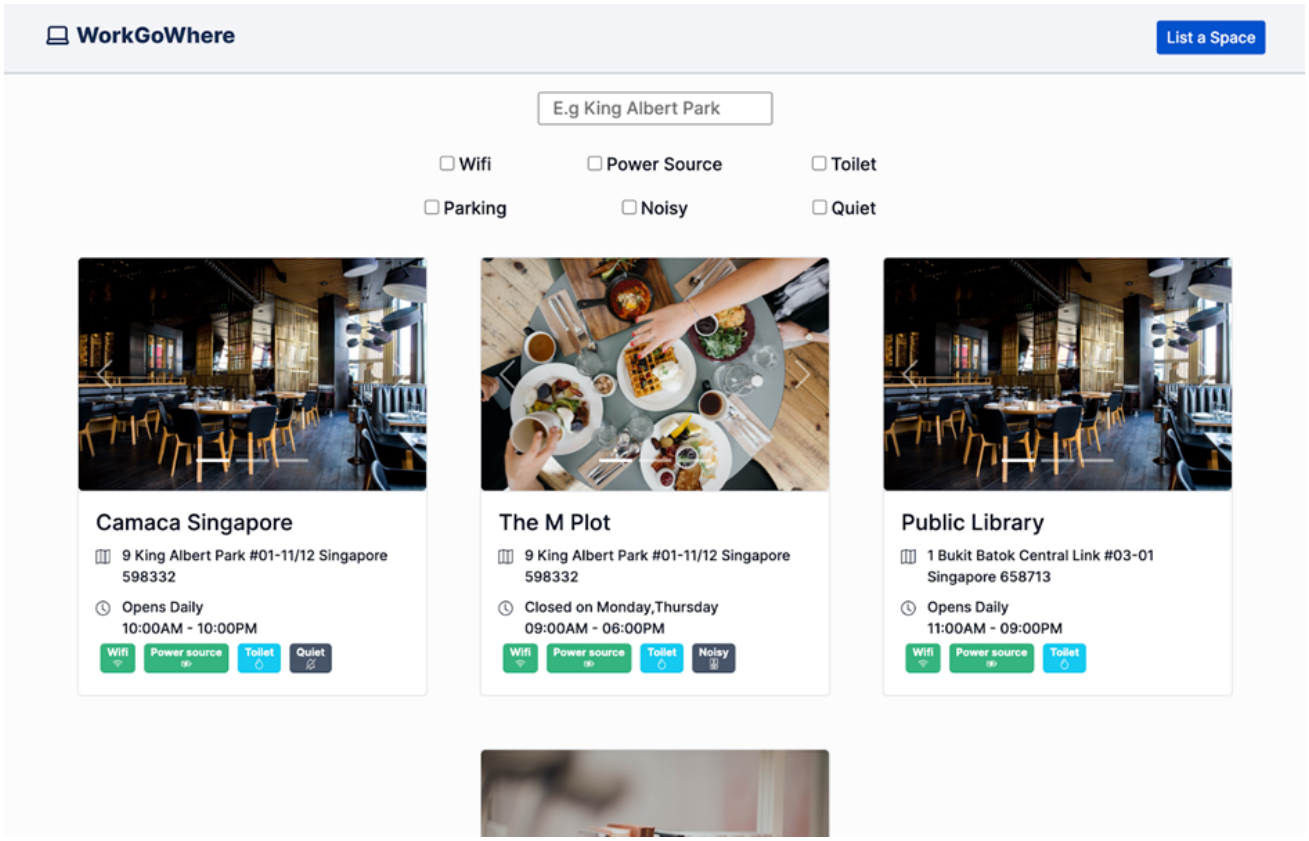


Figure 11.3 - The search features of our dynamic web app

On another note, the descriptions of these blogs/articles would occasionally lack detail (e.g stating that a cafe is “beautiful” without a form of comparison or omitting price ranges). Thus, the effectiveness of the information provided can be inconsistent due to it being dependent on the author’s writing abilities. We will address this a bit further in the next section.

6.2 Notable strengths

When visiting the listicles, we thought of the most important details that a user would want to look out for when searching for a remote workspace. Each of these blogs/articles at the very least displayed key information: Pictures of each location, the services provided, the contacts and addresses of each location and opening hours.

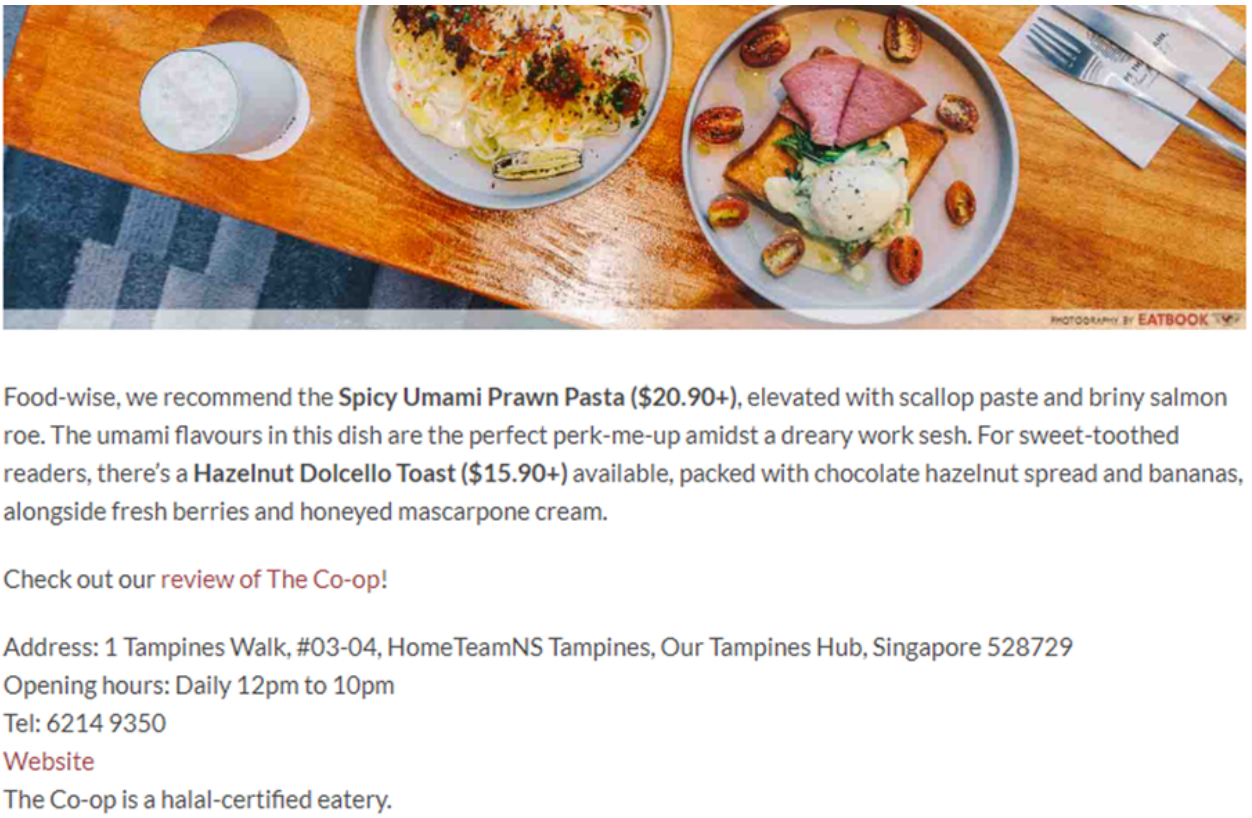


Figure 11.4 - Snippet of Eatbook.sg (Tay, 2021)

With this in mind, we did our best to enhance each of the features that would showcase such information in WorkGoWhere. For example, we have implemented a carousel of multiple pictures for each of the workspaces recommended by our web app instead of just one, so users have a better idea of what to expect.

We took the key services and facilities provided by each workspace and implemented them as categories/tags that we could use as filters; that way, users can look out for places offering specific services. Additionally, we made such information more easily visible; We made them appear on each pop-up card on the results page with more visibly distinct and colourful tags, instead of having to read through the descriptions of each workspace to find such details. We felt that this change would also be more accessible to users with dyslexia or any related visual weaknesses, as they can

better identify such information by the colours of the tag boxes, or simply by the accompanying logos in them.

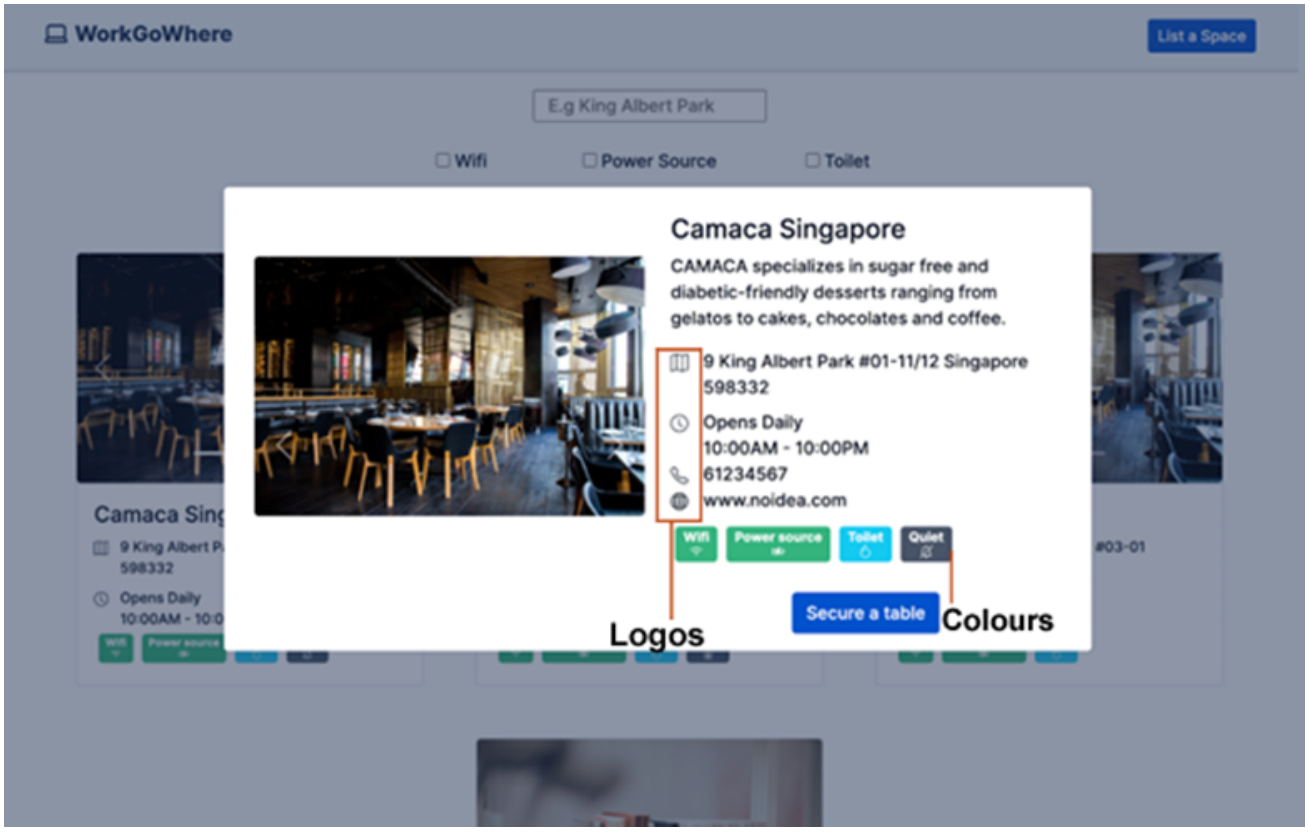


Figure 11.5 - Important information is more visually outstanding on our web app

The display of critical information is clearer and more consistent this way as well, which can help better mitigate the aforementioned issue of less helpful information due to a disparity of writing abilities.

6.2 Overall comparison

Regardless of these improvements, we will still have to keep in mind that not everyone may wish to use WorkGoWhere when searching for remote workspaces. The reasons for this may be due to personal preferences and habits of utilising a more commonly used general search engine or less tech savvy individuals being uncomfortable with using a new web app. While this problem is not entirely avoidable, further development of the web app to make it more accessible and easier to use may potentially help mitigate such issues over time.

Ultimately, the blogs/articles may not always be clear and concise when recommending workspaces that cater to different needs. The process of finding suitable recommendations can be tedious, which may turn people off from trying different workspaces.

Thus, we combined the strengths of our competitors and further enhanced them on our web app. We also overcame the weaknesses of these blogs/articles by making our web app dynamic, providing a greater degree of convenience.

7.0 Evaluation

7.1 SWOT Analysis

WorkGoWhere	
Strengths	● Users can search based on their specific needs, removing the need to manually check every workspace listed. ● Key information is provided, such as address, contacts and opening hours. ● Visual aid such as coloured tags and logos make information easier to identify, especially for users with visual weaknesses. ● Link to each workspace's official website/contact is available for users to immediately access. ● Space owners can list their workspaces and update information. ● Space owners can create and login with an account that manages the workspace data they have submitted to the external database.
Weaknesses	● Updating the information for workspaces will still require the clients or developers to manually make edits.
Opportunities	● Potentially attractive to clients who want to promote their workspaces as listing their workspace on WorkGoWhere can act as a form of free advertising. ● Potential business partnerships with clients who want WorkGoWhere to further promote their workspace. ● May be especially useful to users who often work remotely or enjoy trying new workspaces.
Threats	● Blogs/Articles that have more detailed recommendations. ● Blogs/Articles that contain information regarding workspaces that WorkGoWhere lacks knowledge of. ● Improvements in the search algorithms of general search engines that can bring users better results.

Table 9.0 - SWOT analysis

7.2 Current Limitations

After a critical assessment of our project, there are several limitations and shortcomings that we would like to address.

Limitations	**Description**
Image upload function	Located in the /user-form page, users are able to upload images successfully, however, the current code and setup we have can only run under development mode, and it is not suitable for production. This is because the images are saved directly to the local disk storage, instead of to the database.
Lack of edit function	Located in the /dashboard page, if users have an existing listing, there is no option for them to edit their listing information.
Quality of code is average	Due to the lack of experience creating a MERN full stack web application among the team members, there is ample space for improvement in terms of quality of code and application of best practices.

Table 10.0 - Current limitations

7.3 Future Work and Areas for Potential Improvements

Based on the limitations discussed in 7.2 Current Limitations, Table 11.0 will elaborate on the team's suggestions for future work and potential improvements.

Limitations	**Potential Improvements**	**Description**
Image upload function	Store images in a cloud storage	An efficient way of storing users' images is by saving them into a cloud storage like Google Cloud Storage (GCS). GCS stores its content in "buckets" and each file has a unique public URL. With this in mind, the team could take advantage of this service and store users' images in GCS, and get the file's public URL to store the URL in the database. The benefits of this method would solve our current limitation and it also would not take up unnecessary memory in the database because only the image URL are saved in the database instead of the raw images.
Lack of edit function	Implement a put HTTP method	On the front end, create an "Edit" button in the /dashboard page. On the backend, create a new endpoint that uses a put method so that existing listing data can be mutated and updated to the database.
Quality of code is average	Refactor code and do more extensive code reviews	The team could afford to spend more time refactoring code during the time spent on the project. In addition, we could potentially engage with a more seasoned full stack developer to help give constructive feedback and tips for better code quality.

Table 11.0 - Potential improvements for the project

7.4 Methodologies Employed

7.4.1 Agile

The team chose to use agile, a project management methodology that emphasises incremental and iterative project completion phases. This is suitable as the components of our project are built in phases. Since agile features continuous feedback, we are able to communicate with our stakeholders and implement their feedback in the development iteration. This allows us to adapt to problems as they arise.

7.4.2 Scrum

Prior to our mid term submission, it is common to have at least two meetings per week due to lack of planning. Since this is the first time the team is building a project of such scope, we decided to subscribe to the scrum framework to help us deal with any unpredictability and solve complex problems. Scrum helps us minimise the need for meetings as our goals and objectives are clearly defined during the sprint planning phase. Finally, sprint reviews and sprint retrospectives helps the team refine our teamwork and develop new strategies to solve new problems.

7.4.3 Kanban board

Although the team is adhering to the scrum framework, we prefer to use a kanban board to visualise our workflow for the current sprint. After selecting the tasks from the product backlog into the new sprint, Jira automatically splits all tasks according to the assignee and populates each member’s board. This helps the team easily visualise and track personal progress, alongside each other.

7.4.4 GitHub

The team used GitHub as our version control system to track our changes we make to files. It provides us with a record of what has been done. If we accidentally break any code, we can easily revert back to previous versions if necessary. GitHub also facilitates cooperation by enabling the merging of several contributors' edits into a single repository.

8.0 Conclusion

8.1 Achievements and Outcomes

All in all, the website was successful as the site was created as intended and made mobile friendly. Apart from the site being successfully created, the team made use of the materials made available online as well as from the course such as SWOT analysis and methodologies such as agile, scrum and kanban to complete this project.

The goal from the early stages of this project is to provide convenience and a more efficient process when it comes to finding remote workspaces for working adults. WorkGoWhere, the web application that the team has produced, does exactly that. In a hypothetical way, if this project went to production and made live for virtually anyone who has a device and internet connection, this web application that acts more like a tool could potentially impact the lives of many.

It saves time. You no longer have to manually go through multiple articles to search for a suggestion or answer, because you could easily filter out your options on WorkGoWhere. This is particularly useful for a busy working adult who would appreciate efficiency.

It helps promote businesses. Business owners who listed their space on WorkGoWhere can view this as an advertising opportunity to attract potential patrons to their premises, especially during downtime.

Final word count: **9,999**

9.0 Individual Reflection

I would like to say that my teammates are the best people to work with. Since the start of this module till the end, they have been supportive and would take initiatives to make up for each other's weaknesses. At the start of the module, we made sure that each member of the team would share his/her strengths and weaknesses so that we are aware and would be able to help each other. Apart from helping each other, we were able to learn new libraries and methods to build the front and backend of the website. During discussions, everyone was able to share their thoughts and difficulties which allowed us to communicate with each other effectively. Personally, I feel that each member of the team was able to contribute equally in this project. By adopting the methodologies learned in this module, the team and I were able to work effectively through Jira and other tools which were mentioned in the report. All in all, I think that the team and I were able to research in-depth the topic on hand, code efficiently for both front and backend, work and communicate effectively by using the methodologies. The project was successful and it was a fruitful journey with my teammates.

Appendix A - User Experience Interviews

Name of interviewer:	Winson
Name of interviewee:	Tyler Tan
Date of interview:	16 Aug 2022
Interview duration:	10 minutes

As Users

PRODUCT REACTION QUESTIONS These questions are meant to help identify suggestions or ideas that the person has. Ideally these are asked after the person has used the product or you've walked them through doing a few tasks.	
Question	**Response**
What do you think of this service? (meant to be asked at the homepage to gauge initial reaction)	A website for booking working spaces, similar to cafe searching websites.
What's most appealing about this service?	Easy for students and working adults to navigate.
What's the hardest part about using this service?	Search bar does not feel intuitive.
Was there anything missing from this service that you expected?	Some filters seem to overlap (Ex: noisy vs quiet). Users should not be able to select both options together.
Would you keep using this service after what you saw today?	Yes. I would like to explore places I have not visited before. Filter menu is also comprehensive enough to refine what I am search for
Does this remind you of any other products?	Similar to HungryGoWhere website

As Working Space Owners

PRODUCT REACTION QUESTIONS These questions are meant to help identify suggestions or ideas that the person has. Ideally these are asked after the person has used the product or you've walked them through doing a few tasks.	
Question	**Response**
What's most appealing about this service?	Sign up process and form is easy to fill up. Does not take much time to list a space. Simple registration process

What's the hardest part about using this service?	No hard part.
Was there anything missing from this service that you expected?	It would be better to include some way to verify the user signing up.
Would you keep using this service after what you saw today?	Yes
Does this remind you of any other products?	No

References

Brooke, J. (n.d.). *System Usability Scale (SUS)*. Usability.gov. Retrieved August 31, 2022, from https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

Facebook. (n.d.). *React*. A JavaScript library for building user interfaces. https://reactjs.org/

Field, L. (n.d.). *Where is the best place in Singapore to study overnight ?* https://blogwithlutefield.wordpress.com/2014/08/15/where-is-the-best-place-in-singapore-to-study-overnight/#respond

Free Code Camp. (n.d.). *Learn to code — for free. Build projects. Earn certifications.* https://www.freecodecamp.org/

Friday, H. (2021, January 25). *Part 1: Setting Up Your Backend with Mongoose, Express & MongoDB*. Part 1: Setting Up Your Backend with Mongoose, Express & MongoDB. Retrieved August 18, 2022, from https://dev.to/halented/part-1-setting-up-your-backend-with-mongoose-express-mongodb-2f2p

Kamboh, S. (2022, April 22). *23 Best Free Places to Study in Singapore With Wifi and Power Sockets*. Frostfairs. Retrieved August 31, 2022, from https://frostfairs.com/places-to-study-in-singapore/

MongoDB. (n.d.). Welcome to the MongoDB Documentation. https://www.mongodb.com/docs/

Tay, P. (2021, April 7). *12 Quiet Cafes In Singapore For All Your Studying And Remote Working Needs*. Eatbook. Retrieved August 31, 2022, from https://eatbook.sg/quiet-cafes/

Traversy Media. (2021, January 19). *React JS Crash Course*. https://www.youtube.com/watch?v=w7ejDZ8SWv8&t=1664s