UCL Centre for Advanced Spatial Analysis

Web Architecture

CASA0017

UCL Accommodation Finder

Final Report

University College London

Group:	Titan 6
Dankao Chen:	24067882
Ke Bai:	24045300
Tina Samie:	18127300
Zhiyu Cao:	24067861
Github:	https://github.com/Reikimen/casa0017-web-TiTan-6

Contents

1 Introduction

1.1 Background

Commuting satisfaction significantly impacts the campus engagement and academic achievement of students (Taylor and Mitra, 2021). For students from other regions or countries, unfamiliarity with the new environment, coupled with misleading advertisements from specific real estate agents, can lead to poor housing decisions and difficulty in finding suitable walking routes for commuting. Therefore, universities must provide convenient commuting guidance for all new students, assisting them in selecting appropriate accommodations and commuting routes. This proactive approach enhances their overall university experience and academic performance. Developing a functional guidance webpage is the most cost-effective solution to achieve this goal.

1.2 Motivation

The idea for this project stems from the personal experiences of new students at UCL. Some of them encountered significant difficulties in finding accommodation that balanced proximity to campus, affordability, and accurate representations of the housing conditions. The primary target audience for this project was identified as new students at UCL. All data collection and UI design were specifically tailored to students in the London area, aiming to simplify the accommodation search process by integrating accurate housing options and essential commuting information. This approach empowers students to make informed decisions. The webpage design should account for cultural diversity and accessibility, ensuring it is clear, easy to use, and free from potential misunderstandings. Data accuracy and timeliness must be prioritized on the backend to ensure that the information displayed on the webpage is precise and up-to-date. Additionally, the response speed of the web page must be sufficient to allow users to quickly access the information they need without spending excessive time waiting. The scalability of the project must also be taken into account. The final webpage should be replicable and easily adaptable to meet the needs of other universities and regions with minimal modifications.

1.3 Aims and Objectives

The primary aim of this project is to develop a web-based application, **"UCL Apartment Helper,"** that provides new students with a user-friendly platform to locate suitable accommodations near UCL campuses. The objectives include:

1. Integrating Google Maps API to fetch real-time data on UCL buildings and nearby student accommodations.
2. Offering route recommendations and walking distance calculations between teaching buildings and accommodations.
3. Ensuring the platform is accessible to users of diverse linguistic and cultural backgrounds through a simple and intuitive interface.
4. Enhancing user decision-making by dynamically displaying detailed information about each accommodation option, such as distance, duration, and location.

By addressing these objectives, we aim to alleviate the challenges faced by incoming UCL students and promote a more seamless transition into university life.

1.4 Project Overview

The 'UCL Apartment Helper' web application developed by CASA Titan 6 enables UCL students to find suitable accommodation that is close to the UCL campus, and provides recommendations for walking commuting routes. The project integrates with Google Maps and other APIs to obtain dynamic data, calculate routes and display detailed information about accommodation and campus buildings. The front-end adopts a simple and intuitive interface design. The back-end successfully realising the complete functional chain of data and user interaction, which significantly improves the user's decision-making efficiency and experience.

2 Methodology

2.1 back-end Development

2.1.1 Data Collection Process

The data collection process leverages two key services from the Google Maps API (Google Cloud, 2025) : **Places API** and **Routes API**, which are integrated through a unique API key configured via the Google Cloud Console. This section outlines how data is fetched, cleaned, and prepared for further use.

Places API

The Places API enables querying and retrieving location-based information. In this project, two specific queries—"UCL buildings" and "Student halls London UCL"—are used in text search mode. This method returns detailed information for each location, including:

- **Google Maps ID:** A unique identifier for the place.
- **Name and Address:** Textual details to enhance user readability.
- **Geographical Coordinates:** Latitude and longitude for mapping purposes.

A classification system tags each location with a type (*"building"* for UCL teaching facilities and *"accommodation"* for student halls) to facilitate downstream processing. Despite the free-tier API key limiting each query to 20 results, this was sufficient to cover the majority of relevant locations. Data quality was verified manually to ensure that all essential buildings were included.

Routes API

The Routes API calculates walking routes between every building and accommodation. The service returns the following key metrics:

- **Origin and Destination IDs:** Corresponding to the places stored in the database.
- **Travel Duration:** Walking time in seconds.
- **Distance:** Route length in meters.
- **Encoded Route String:** A polyline string that can be decoded for mapping.

Walking was chosen as the transportation mode based on the proximity of accommodations to UCL's main campus. The backend systematically computes all pairwise routes using a nested loop to match each building with every accommodation, ensuring comprehensive coverage.

Optimised API Usage

Since the data from Google Maps APIs is relatively static, frequent API calls are unnecessary. To minimize API costs and reduce latency, the project adopts a caching mechanism by storing the fetched data in a database. Updates are scheduled at monthly or yearly intervals, as required, using automated scripts to refresh the database.

2.1.2 Database Setup and Data Management

A Raspberry Pi serves as the backend server for this project, running a Dockerized MySQL database. This lightweight setup is cost-effective, scalable, and ideal for small to medium-scale applications. The backend integrates data collection, cleaning, storage, and management to ensure reliable interaction with the front end.

Database Schema Design

Two core tables, **Places** and **Routes**, were designed to store and retrieve data efficiently.

Places Table

- **Fields:**
  - `place_id` (Primary Key): Unique ID from Google Maps.
  - `name`: Name of the location.
  - `address`: Physical address.
  - `latitude` and `longitude`: Geolocation coordinates.
  - `type`: Classification as *building* or *accommodation*.
- **Purpose:** Serves as the reference table for all UCL locations.

Routes Table

- **Fields:**
  - `route_id` (Primary Key): Unique identifier for each route.
  - `origin_id` and `destination_id`: Foreign keys linked to `place_id` in the **Places** table.
  - `distance`: Route length in meters.
  - `duration`: Walking time in seconds.
  - `polyline`: Encoded route string for visualization.
- **Purpose:** Stores route data between all building and accommodation pairs.

Data Insertion Workflow

1. **Places Table Population:** Data fetched from the Places API is parsed and cleaned to remove duplicates and irrelevant entries. Validated records are inserted into the **Places** table using a dedicated insert function.
2. **Routes Table Population:** Using the entries in the **Places** table, the backend calculates routes between all combinations of buildings and accommodations using the Routes API. The results are stored in the **Routes** table with proper indexing for efficient querying.

Data Cleaning and Error Handling

During data insertion, several checks are performed:

- **Duplicate Removal:** Ensures no redundant entries in the **Places** table.
- **Data Validation:** Verifies that all fields (e.g., coordinates, names) are populated.
- **API Response Handling:** Implements retries for failed API calls and logs errors for debugging.

Security and Scalability

- **Environment Variables:** Sensitive credentials, such as the API key and database password, are stored in a `.env` file. This prevents accidental exposure and simplifies deployment across environments.
- **Docker Integration:** Docker ensures that the MySQL database is portable, isolated, and easily replicated on other machines.

Front-End Data Access

The backend exposes APIs to the front end for dynamically fetching location and route data. These APIs query the MySQL database for relevant information, ensuring a seamless user experience without requiring direct API calls to Google Maps.

Conclusion

By integrating a structured approach to data collection, handling, cleaning, and storage, the backend provides a reliable and efficient system that supports the front-end functionality while minimizing costs and ensuring data integrity.

2.2 front-end Development

2.2.1 User Interface Design

After group discussion, it was decided that the user interface (UI) of this project should adhere to a simple and intuitive design philosophy. The goal is to ensure clarity and usability while maintaining scalability

for future website development(Stone et al., 2005). The theme of the interface adopts a minimalist modern style, using clean text and block-based visuals to convey information effectively without overwhelming the user. This approach minimizes distractions and lowers the barrier to entry, avoiding overly complex interactive features.

The primary function of this project is to assist users in locating suitable student accommodations and recommending commuting routes to and from their designated school buildings. To achieve this, the interface must be designed with intuitiveness and ease of operation. Users are required to select their desired UCL campus building and acceptable commuting time via simple dropdown menus. Based on their input, the system will filter suitable accommodations automatically and display them on the map. Users can click on map markers to view detailed information, including accommodation names, addresses, commuting distances, and estimated travel times.

Interface Modules

The interface consists of several modules:

1. User Interaction

Two dropdown menus facilitate the interaction. Users can select a UCL campus building and specify their commuting time preferences. This step-by-step process simplifies user actions and reduces cognitive load.

2. Map Display

The map serves as the central information visualization area, intuitively presenting the geographical distribution of campus buildings and recommended accommodations.

- Different colours and marker styles distinguish between buildings and accommodations.
- The map supports dynamic zooming and panning, allowing users to adjust the view based on their commuting preferences.
- Map markers serve as interactive elements; clicking on an accommodation marker displays the walking route between the accommodation and the selected campus building.

3. Information Sidebar

This module provides detailed information about accommodations and commuting routes. When a user selects an accommodation on the map, the sidebar updates to display relevant data dynamically, including:

- Campus building name.
- Accommodation address.
- Commuting distance.
- Estimated travel duration.

This ensures that users have access to comprehensive reference data for decision-making.

4. Navigation Header and Footer

- The **header** prominently displays the title *"UCL Apartment Helper,"* emphasizing the application function.
- The **footer** provides links to additional resources, such as student housing information websites and accommodation booking platforms.

This UI structure prioritises simplicity and practicality. It enables users to access the necessary information and interact with the system seamlessly while laying a solid foundation for future functionality enhancements.

2.2.2 Function and Code

The core functionalities of the project, as reflected in the user interface, include the following:

Core Functionalities

1. **Dynamic Map Loading:** The system loads the Google Maps API script upon page initialization by obtaining the API key from the backend. This enhances security, reduces unnecessary resource usage, and improves page performance.

Figure 1: Dynamic Map Loading

2. **Adding Map Markers:** Two types of markers are displayed on the map: buildings (blue) and accommodations (yellow). Once the user selects a UCL building, the map adjusts its zoom level based on the commuting time set by the user, showing accommodations within the specified range.

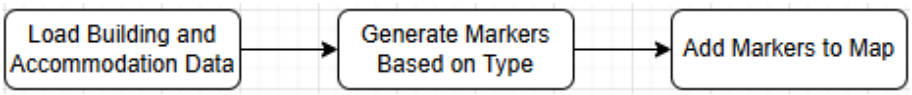

Figure 2: Adding Map Markers

3. **Commuting Route Calculation and Display:** Based on the user-selected building and accommodation, the system calculates walking routes through the backend and displays them as polylines on the map. Commuting distance and time are synchronized and displayed in the sidebar.

Figure 3: Calculation and Display

4. **Sidebar Information Display:** The sidebar provides real-time updates of the selected building, accommodation, and commuting route details, including building name, address, accommodation name, address, commuting distance, and duration.

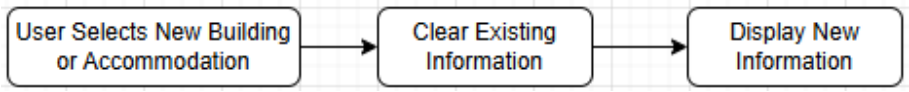

Figure 4: Sidebar Information Display

Initial Front-End Design

In the initial version of the front-end code, several design decisions were made according to (Spolsky, 2008):

- The Google Maps API was used to load the map with a statically set centre.
- `DirectionsRenderer` and `DirectionsService` objects were configured on the front end for path rendering.
- Building data was stored in a local array, and users could select a building to mark its location on the map using its coordinates from the array.
- UCL building names and predefined commuting times were hardcoded into the HTML file.
- A **Search** button was implemented to trigger data loading. The button logic included:
  1. Checking whether the user had selected a building and commuting time.
  2. Prompting the user to complete required fields if no building was selected.
  3. Displaying only the building' s location if commuting time was not specified.
  4. Loading data, updating the map, and refreshing the sidebar if all fields were complete.

Limitations of the Initial Version

- Hardcoded UCL building names made maintaining and synchronizing with the backend complex.
- Interaction relied heavily on clicking the **Search** button, lacking real-time updates.
- The map centre did not adjust dynamically to user selections, requiring manual navigation.
- Front-end-based route calculation limited integration with backend data.
- Exposed API key in the front-end code posed significant security risks.

Final Front-End Design

To address the limitations of the initial version, several improvements were implemented in the final design:

1. **Dynamic API Key Retrieval:** The system securely requests the API key from the backend, enhancing security and ensuring proper resource management.

2. **Dynamic Data Handling:** Building and accommodation data are dynamically fetched from the backend using AJAX. Dropdown menus for building selection are populated dynamically, ensuring synchronization with database updates and reducing maintenance efforts.

3. **Improved Map Interaction:**

    - Markers are colour-coded (blue for buildings, yellow for accommodations) for clarity.
    - The map dynamically adjusts its zoom level based on the selected commuting time, ensuring that accommodations within the specified range are visible.
    - Selecting a building re-centres the map on its location and highlights the marker.

4. **Accommodation Marker Logic:** Clicking on an accommodation marker triggers a backend request to fetch walking route data, including distance and time. The map displays the route as a polyline, and the sidebar updates dynamically with relevant details.

5. **Real-Time Updates:** The map and sidebar respond dynamically to user selections, eliminating the need for a **Search** button.

6. **Security and Scalability:** The API key is no longer exposed in the front-end code. Building and accommodation data updates in the backend are automatically reflected in the front-end interface.

Impact of Design Evolution

The final version introduced significant improvements:

- Automatic synchronization between the front-end and backend ensures data consistency.
- Real-time updates for map markers and sidebar information enhance user experience.
- Improved security through retrieval of API key from the back-end API.
- Dynamic data handling and map interaction provide a seamless and scalable system.

Conclusion

This evolution from the initial to the final version highlights the importance of task prioritization and secure integration in front-end development. Although the initial version established a functional foundation, the final design greatly improved usability, scalability, and security, resulting in a robust and reliable system.

2.3 Deployment Optimization

Though in a lab environment, team collaboration can be achieved by deploying the environment on one of the Raspberry Pi's and using it together for their collaborative development. However, such an approach relies too much on off-line, forcing all members to be in the same working space. Moreover, in the unfortunate event that a device breaks, the development team may be faced with the dilemma of redeploying the development environment, despite the support of the archive on GitHub.

2.3.1 Why Docker

Docker, on the other hand, has several significant benefits in front-end and back-end (including database) projects as follows, that's why Docker was adopted for this project(Mouat, 2015):

- **Environmental Consistency**
  - **Problem**: Team members may have different development environments (operating systems, dependency versions, etc.), which can easily lead to the phenomenon of 'it's fine on my computer' .
  - **Solution**: With Docker, everyone can run the same container environment (including the operating system, software version, etc.) to ensure consistency.
- **Simplified Deployment**
  - **Problem**: Manual deployment requires configuring multiple environments (database, back-end services, front-end), which is a tedious and error-prone process.
  - **Solution**: Docker provides an independent environment, and you only need to run `docker-compose up --build` when the project starts.
- **Isolation Solution**
  - **Problem**: There may be other services running locally (e.g. different versions of Node.js, MySQL), which are prone to conflict.
  - **Solution**: Docker containers are independent each other, and different services run in their own isolated environment.
- **Quick Startup and Rebuild Environment**
  - **Problem**: It takes time to configure the environment when new members join.
  - **Solution**: New members can quickly run projects via Docker by simply installing Docker and pulling the project code.

2.3.2 Doker-based Project Framework

```
Repo-Structure/
├── Group Report/              # Documents or reports for the project
├── Website/                   # Main application folder
│   ├── backend/               # Backend application code
│   │   ├── node_modules/      # Node.js dependencies
│   │   ├── .env               # Environment variables
│   │   ├── Dockerfile         # Dockerfile for backend
│   │   ├── package.json       # Backend dependencies and scripts
│   │   └── server.js          # Main backend server file
│   ├── frontend/              # Frontend application code
│   │   ├── css/               # Stylesheets
│   │   │   └── styles.css     # Main stylesheet
│   │   ├── img/               # Images for the frontend
│   │   ├── js/                # JavaScript files
│   │   │   └── map.js         # Frontend logic for map functionality
│   │   ├── index.html         # Main landing page
│   │   └── login.html         # Login page
│   ├── mysql/                 # Database setup
│   │   ├── data/              # Database data files (if applicable)
│   │   ├── init/              # Initialization scripts for the
database
│   │   │   └── init.sql       # SQL initialization script
│   ├── nginx/                 # Nginx configuration files
│   │   ├── sites-enabled/     # Nginx virtual host configurations
│   │   │   └── frontend.conf  # Configuration for the frontend
│   │   └── nginx.conf         # Main Nginx configuration file
│   ├── docker-compose.yml     # Docker Compose configuration
│   └── Notes-for-developers/  # Developer notes or guidelines
├── README.md                  # Project documentation
├── .gitignore                 # Git ignore file
├── LICENSE                    # Project license file
```

Figure 5: The Repo Structure

The above project structure was designed based on multiple perspectives refer to (Yunrui, 2018) - teamwork, development efficiency, system performance, security and user experience, and has the following characteristics:

Firstly, front-end and back-end are separated. Front-end and back-end separation helps decouple the front-end (UI/UX) and back-end (logic, data processing), each developed and tested independently, reducing interdependence. Technical flexibility, the front and back ends can use different technology stacks. Reusability, the back-end interface can serve multiple clients at the same time (e.g., Web front-end, mobile apps, etc.), improving code reuse.

Second, using Nginx, a high-performance and lightweight reverse proxy server, can handle a large number of concurrent requests while consuming fewer resources. With its strong static resource handling capability, it is particularly suitable for hosting front-end static resources (HTML, CSS, JavaScript), which can improve access speed.

Finally, although not reflected in the structure of the project, phpMyAdmin was used for data visualisation. Developers can better get up-to-date with SQL and being participate. It is worth mentioning that in this project, the . /sql/data folder is used to store both project-generated (e.g., new data per dropdown from the Google API) and running (e.g., cumulative updates to user forms) data, and the use of '.gitignore' ensures that only user forms are uploaded to github.

Note: For information on how to develop with this framework, please visit the project web site provided on the cover of this report.

3 Achievements and Reflections

Default Page

When a users clicks on the website, the main content of the web page is displayed as shown below before any action is taken:

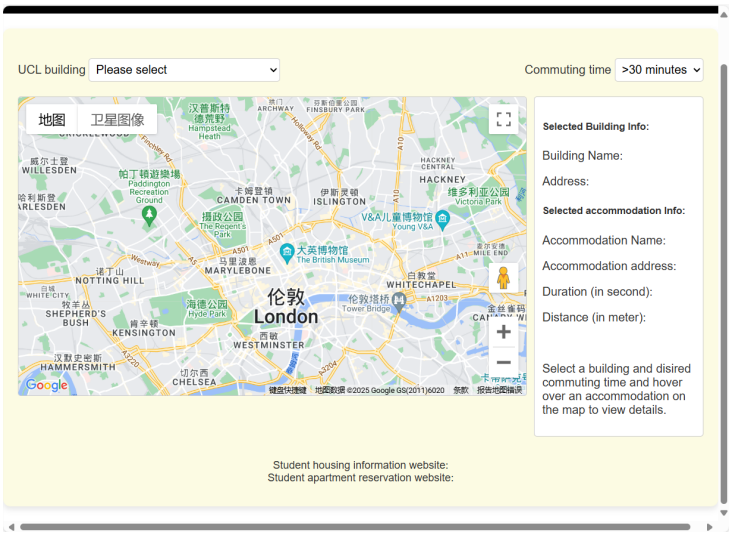

Figure 6: Default Page Status

Recommend student accommodation

After the user selects the UCL building they want to use as a commuting location and limits the commuting time, the webpage prompts you with some recommended UCL student accommodation.

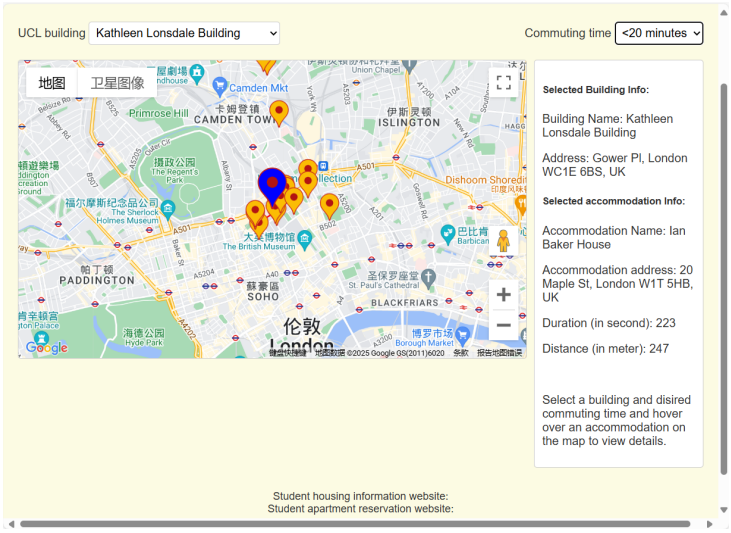

Figure 7: Recommended Accommodation

Recommend walking commute route

After selecting a flat and commuting location (UCL building), the user is given a recommended walking commute to school.

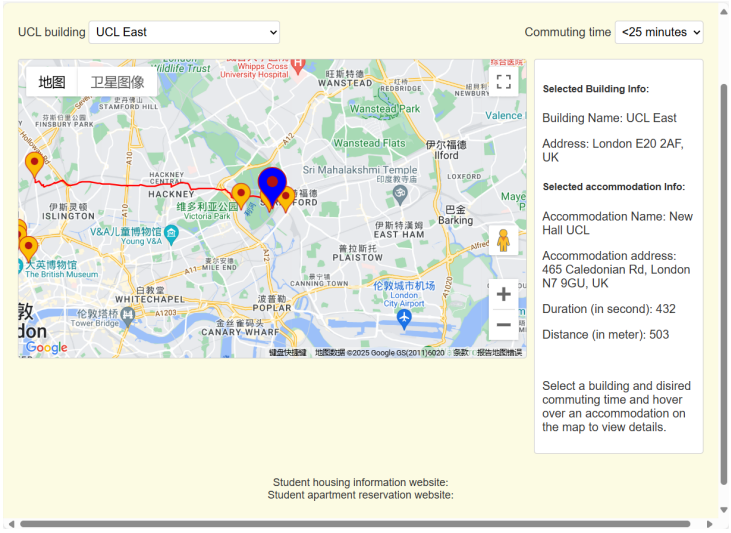

Figure 8: Recommended Routes

Developer View

This is the corresponding developers' perspective:

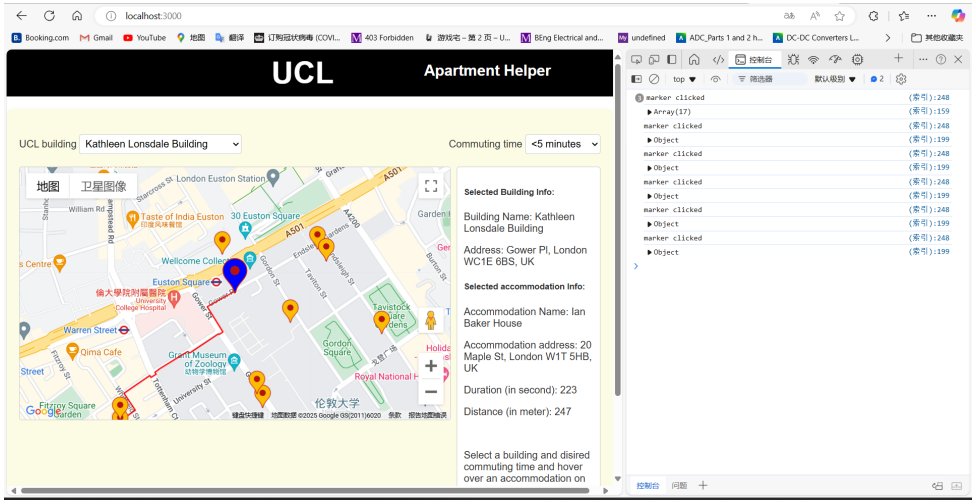

Figure 9: Developer's View

Overall Web Page

Additionally, as the project is explicitly developed for UCL students, a secondary page navigation was included in the header to allow users to input their UCL student account. This feature is designed to facilitate future functionality expansion.

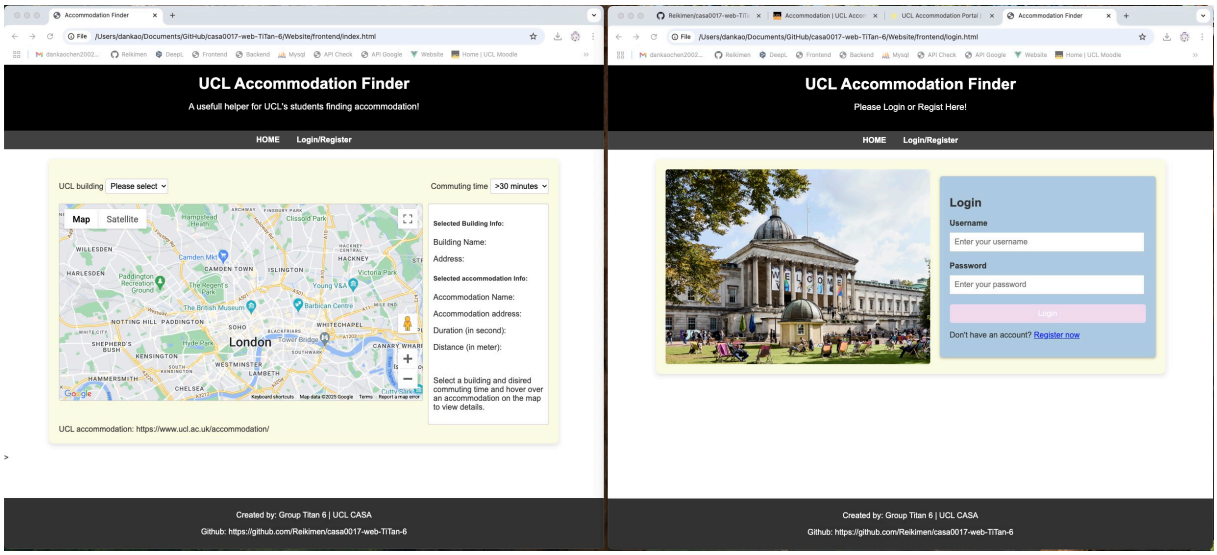

Figure 10: Overall Web Page

Only if this project gains support from UCL administration and access to the student information database is provided, the function of the webpage can be further enhanced to deliver personalised services tailored to the needs of each UCL student.

Docker-based Deployment

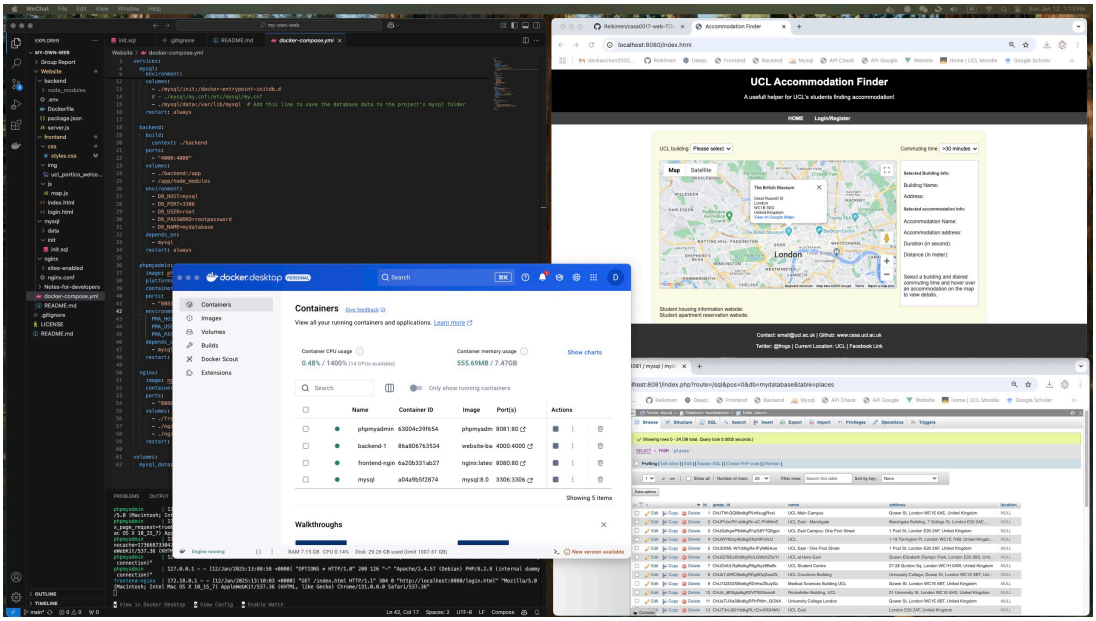

Figure 11: Docker-based Deployment

As you can see from the figure above, this project implements containerised deployment of front-end and back-end, database and phpMyAdmin, which allows you to quickly start all the services with a single command, avoiding cumbersome environment configurations. At the same time, phpMyAdmin provides intuitive database visualisation, which greatly improves development and debugging efficiency.

In fact, there is another surprising feature , that is : development and testing in parallel - front and back-end separation and containerisation so that development and testing can be synchronised , modify the code after a quick refresh to see the results , without additional configuration or complex restart process.

Although the framework was working perfectly, the front-end developers believed that a ‘connection’ to the database made sense, but the back-end was using a ‘Pool’ to access the database. As a result, the development team’s attempts to migrate the deployment were not very successful, resulting in a lot workload of code re-writing.

4 Future Works

Although through the efforts of each member of the team, and even extremely painful integration and collaboration, the project finally succeeded in achieving the front and back-end database linkage, very beautiful UI design, and the implementation of all the functions planned at the beginning of the project.

However, there are still the following areas that can be continuously improved in the future:

1. More user-friendly, more beautiful UI/UX design.
2. Responsive design: Improve the adaptability of the website to different devices, including a better mobile experience. Optimise the image resolution for retina screens to provide higher definition visual effects.
3. Internationalisation support: Add multi-language support (i18n) to facilitate access and use by international users.
4. Access to UCL’s user system, website access through UCL account.
5. Use some very practical gadgets to limit illegal traffic access, to prevent the server from being attacked, for example: ’Are you Robot’.
6. Improve the framework of Docker migration and deployment, so that the front and back end data transmission is more fluent, more efficient deployment.

In the future, we will continue to improve our site as well as conditions allow, allowing more UCL students to move into the halls of preferred residence.

References

Google Cloud (2025). *Google Maps Platform Documentation*. Accessed: 12 January 2025. Available from: https://developers.google.com/maps/documentation.

Mouat, A. (2015). *Using Docker: Developing and deploying software with containers*. " O'Reilly Media, Inc.".

Spolsky, A. J. (2008). *User interface design for programmers*. Apress.

Stone, D., Jarrett, C., Woodroffe, M., and Minocha, S. (2005). *User interface design and evaluation*. Elsevier.

Taylor, R. and Mitra, R. (2021). Commute satisfaction and its relationship to post-secondary students' campus participation and success. *Transportation Research Part D: Transport and Environment* [online]. 96, p. 102890. Available from: https://www-sciencedirect-com.libproxy.ucl.ac.uk/science/article/pii/S1361920921001917.

Yunrui, Q. (2018). "Front-End and Back-End Separation for Warehouse Management System". In: *2018 11th International Conference on Intelligent Computation Technology and Automation (ICICTA)*. IEEE, pp. 204–208.