GDP report

Anthony Adamson

Joshua Coombe

Shane Ong

Jack Parsons

Aamina Rizvi

Kathy Sui

April 2024

Chapter 1

Project Aims

1.1 Project Aims

The primary aim of this software engineering project was, in partnership with Kuehne+Nagel, to develop a hotdesking system tailored to enhance workplace efficiency, teamwork, and employee satisfaction by automating desk reservations.

After discussion with the team, internal and external supervisor, we developed specification documentation with both the Volere and user story framework. The details can be found in Appendix A, but main project aims are summarised below:

- **Automated booking:** Employees should be able to book desks in specific offices for particular days, ensuring workspace availability. The system should use a calendar UI for date selection and update a database in real-time upon booking.
- **Team cohesion:** An allocation algorithm should enhance team cohesion by positioning team members near each other within the office. The algorithm should process lists of desks and their locations, along with user department data to optimize seating arrangements.
- **Fairness:** Desks are allocated on a first-come, first-served basis to guarantee fairness. The system should automatically reject further bookings once office capacity is reached.
- **Accessibility:** The booking system should be accessible via a webpage

which is designed to be user-friendly and secure, supporting multiple browsers and devices with robust authentication measures.

- **Flexibility:** Features such as single-day bookings, part-day bookings, and easy cancellation options should be incorporated to accommodate flexible work schedules and personal preferences.
- **Adaptability:** The system should adapt to various office layouts.
- **Notifications:** The system should provide timely notifications about bookings through an efficient user interface.
- **Administrative Control:** Admin controls should be included to manage user access and resolve issues, enhancing system integrity and user experience.

Project Timeline: Project milestones were set, including the completion of the MVP, final delivery, and presentation, with specific dates set for each phase to ensure timely progress and delivery. Bi-weekly internal meetings were arranged to ensure the project progressed smoothly and collaboratively.

Chapter 2

Implementation

2.1 Front-end

The frontend of our hotdesking system was designed to be intuitive and user-friendly while maintaining functionality across various devices and platforms. Utilizing React and integrating with a Node.js backend, our system facilitates smooth data flow and responsive interactions.

2.1.1 User Interface

The control flow is as follows:

- **Non-Authenticated Users:** Redirects any unauthorized access to the login page, ensuring security and appropriate user access.
- **Authenticated Users:** Provides access to home, booking, and account pages. Unauthorized URL requests are redirected to the home page.
- **Administrative Access:** Additional administrative functionalities are available for users with admin roles, enhancing system management capabilities.

Each page is responsive and adapts to both PC and mobile devices, ensuring consistent user experience across all platforms.

The images below show the interface, which was developed to agree with mock-ups made in the pre-development phase (of which some are also shown below).

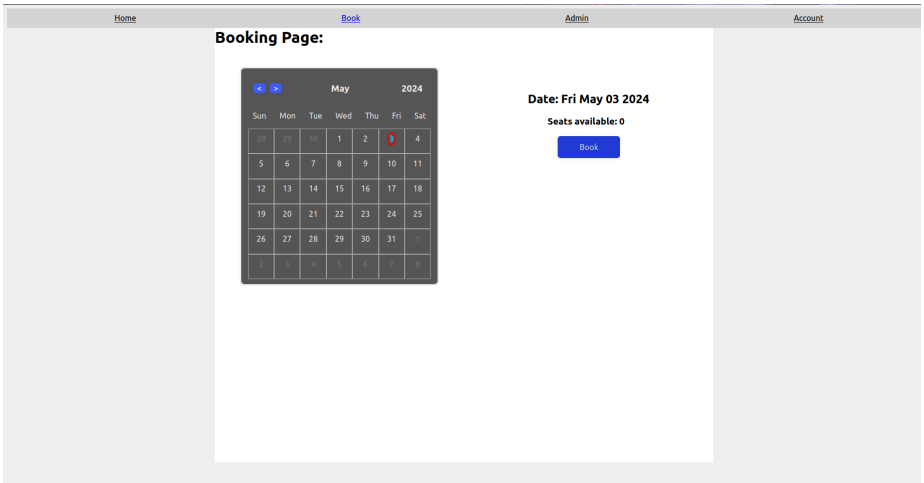

Figure 2.1: Booking page implementation

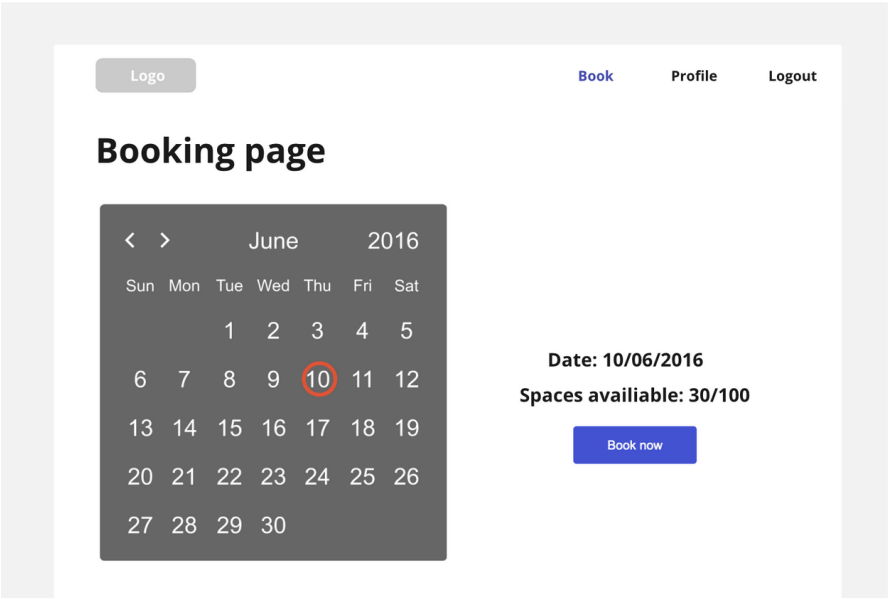

Figure 2.2: Booking page mock-up

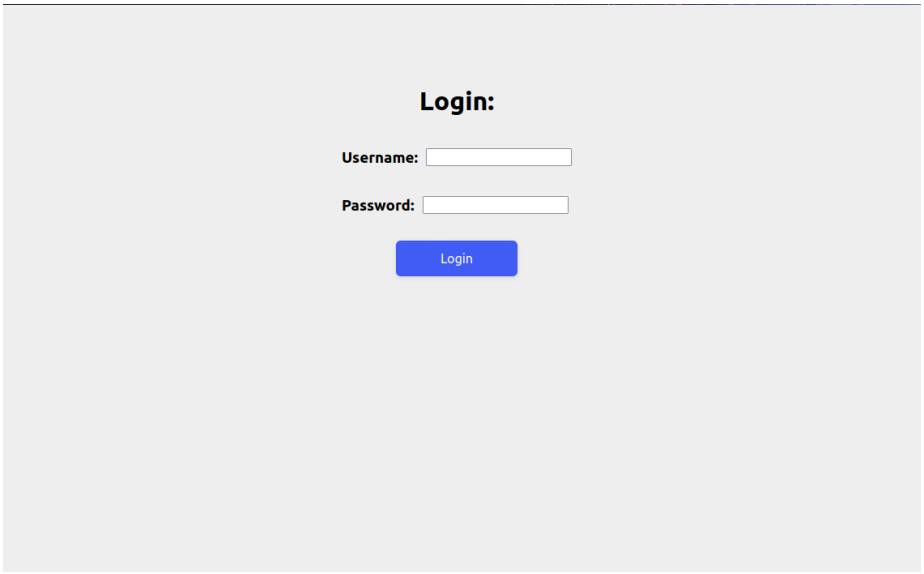

Figure 2.3: Login page implementation

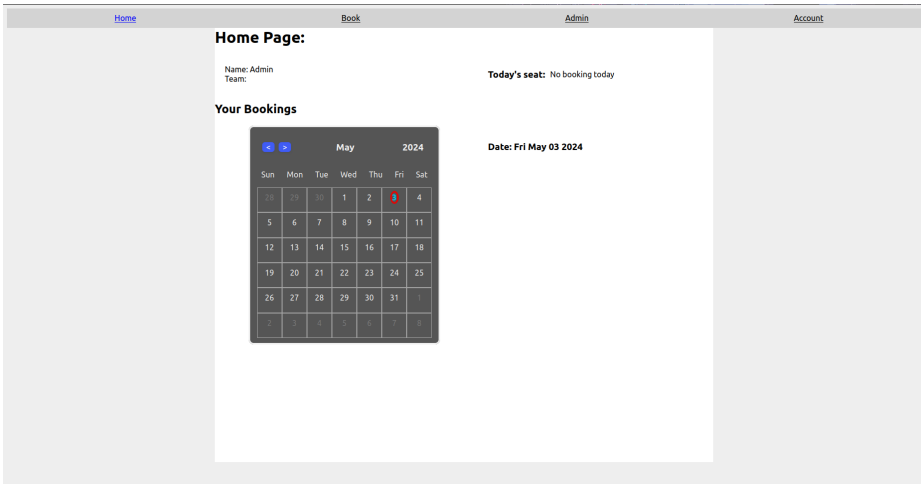

Figure 2.4: Account page implementation

2.1.2 Technical details

The frontend maintains local storage of session data, such as authentication tokens, to ensure persistence across page refreshes. This functionality enhances user experience by maintaining session integrity without repeated logins.

It also communicates with the backend via API calls that manage data securely and efficiently. The frontend updates to reflect changes from these interactions.

2.1.3 Testing with Jest

We have implemented comprehensive testing using the Jest framework to ensure the stability and reliability of the frontend:

- **Screen renders:** Tests verify that the correct elements are displayed when components render.
- **API Interactions:** Tests ensure that API calls are executed correctly and that the frontend processes data as expected.
- **User interactions:** Tests assess the application's response to user inputs like button clicks and form submissions.

Some tests in the testing suite were written before development started so that we had a clear idea of what our end goal was. All tests have passed, confirming robustness and achievement of our initial goals.

2.2 Back-end

Our choice of tools and frameworks in the backend was driven by their robustness, ease of use, and our team's prior familiarity.

2.2.1 Technology Stack

Python

Python was selected for the backend development due to several advantageous features:

- **Ease of use:** Known for its simplicity and readability, Python enables rapid prototyping, which was crucial in the early stages of our project.

- **Integration with data science:** Python's extensive libraries and frameworks for data science made it a natural fit for integrating the algorithm components, which were also likely to be implemented in Python.
- **Team familiarity:** Many team members had prior experience with Python, reducing the learning curve and accelerating development.

FastAPI

FastAPI was utilized for its excellent support for building APIs:

- **Swagger UI:** FastAPI integrates with Swagger UI, providing an automated, rich, and interactive documentation page (refer to the FastAPI documentation for examples). This feature facilitated easy manual testing of API routes.
- **Developer Familiarity:** The lead backend developer had prior experience with FastAPI, which streamlined many development processes.
- **Community Support:** There is a wealth of tutorials and community support available for FastAPI, making it easy for new team members to get up to speed.

Poetry

For package management, Poetry was chosen for its ability to manage dependencies effectively:

- **Virtual environments:** Poetry automatically manages virtual environments, ensuring that dependencies are consistent and isolated across development and production setups.
- **Dependency resolution:** It provides reliable dependency resolution and package management, simplifying the maintenance and update of project libraries.

SQLAlchemy and SQLite

Our system employs SQLAlchemy in conjunction with SQLite for database management:

- **SQLAlchemy:** This ORM framework for Python simplifies database manipulation, making it easier to translate Python classes to database tables and vice versa, enhancing the development process.
- **SQLite:** A lightweight, disk-based database that does not require a separate server process, making it ideal for smaller projects and faster setup during development.

PyTest

PyTest, a powerful testing framework for Python, was utilized to ensure the quality and reliability of the backend:

- **Comprehensive Testing:** PyTest supports a range of sophisticated testing capabilities, which allowed us to implement both simple unit tests and complex functional tests effectively.
- **Ease of Use:** Known for its straightforward syntax and powerful fixtures, PyTest facilitated rapid development of test cases and helped maintain high code quality throughout the project.

2.2.2 Architecture

We organise the API routes by the object they serve (e.g. Users, Bookings, Seats, etc.), and hence it makes sense to arrange the code accordingly as well. This is done by having routes come under FastAPI routers, which can then each be included in the main application.

Each API route is implemented by using the CRUD utilities to retrieve the relevant data from the database, where each basic CRUD operation is a direct SQLAlchemy ORM query.

The login status of a user is maintained by the use of OAuth Bearer Tokens, where JSON Web Tokens are used to maintain session specific data (e.g. user ID and role of the logged in user). This functionality is provided by FastAPI, which we adapt for our needs.

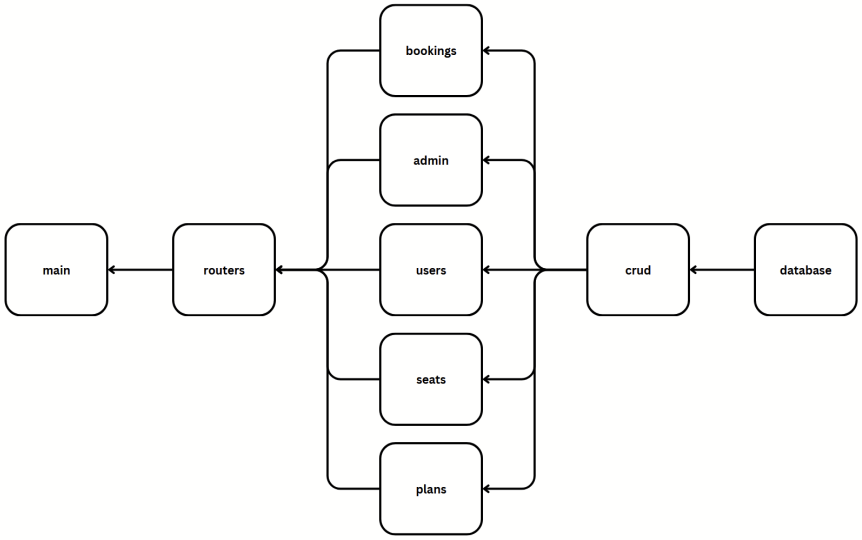

Figure 2.5: Code Layout

2.2.3 Database

The database is responsible for storing and managing all data related to users, bookings, and seats. We utilized SQLAlchemy for database management due to its robust ORM capabilities, which facilitate integration with our FastAPI backend.

CRUD Operations

Our system implements comprehensive CRUD (Create, Read, Update, Delete) functionality to manage data effectively:

- **Create:** Users can register, and administrators can add new seats or booking entries.
- **Read:** Users and administrators can query for booking details, user information, and seat configurations.
- **Update:** Modifications to bookings, user details, and seat configurations are handled through user interactions and admin controls.

- **Delete:** Unneeded data such as outdated bookings or obsolete user accounts can be removed to maintain database integrity.

Integration with FastAPI

SQLAlchemy's integration with FastAPI allows for dynamic data handling:

- **Real-Time updates:** Changes made through the API are immediately reflected in the database, ensuring data consistency across the platform.
- **Transaction safety:** FastAPI and SQLAlchemy together ensure that all transactions are handled safely, preventing data corruption or loss.

2.2.4 Scripts and Validation

Seat creation and management

A dedicated script for seat creation is executed once by the operator of the server, which populates the database with initial seat configurations based on input on a GUI by the operator. This script facilitates the setup process for new office layouts or when expanding to new locations.

Algorithm

A dedicated script to periodically run the seat allocation algorithm separate from the main backend. This helps modularise different components with different purposes, since the backend serves the API primarily.

User management

User details stored in the database include login credentials and future bookings. This information is crucial for both authentication purposes and for maintaining a history of user activity.

Booking Logic and Constraints

The system includes several checks to maintain integrity and prevent common issues:

- **No double booking:** The system checks to ensure that no seat is double-booked for any given time, maintaining a fair and efficient booking process.

- **Capacity constraints:** Total bookings are monitored to ensure they do not exceed the available number of seats, preventing overbooking and ensuring compliance with office capacity regulations.

The described database management strategies ensure that our system remains efficient, scalable, and reliable.

2.3 Algorithm

We researched on various approaches to the algorithm, focusing on encoding the office floor plan and developing a robust seating allocation algorithm.

2.3.1 Encoding the floor plan

The office layout was conceptualized in two primary ways: as a 2D space and as a graph, each with distinct approaches to optimization:

- **2D Space Representation:** Seats were treated as points within a two-dimensional plane, leveraging clustering algorithms like k-nearest neighbours to group seats by department. Challenges included managing physical barriers like walls and ensuring cluster sizes matched department needs.
- **Graph Representation:** The office was modeled as a complete graph with seats as vertices and distances as edge weights. This abstract model required significant preprocessing but facilitated a more sophisticated optimization of seat assignments.

2.3.2 Optimisation Formulation

The optimal algorithm would solve the following optimisation problem:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{a,b \in C_k, a \neq b} \sum_{u,v=1}^{n} d(u,v) \cdot x_{a,u} \cdot x_{b,v} \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{i,j} = 1, \quad \forall i \in \{1, \ldots, m\} \\
& \sum_{i=1}^{m} x_{i,j} \leq 1, \quad \forall j \in \{1, \ldots, n\} \\
& x_{i,j} \in \{0,1\}, \quad \forall i,j
\end{aligned}
$$

Where $x_{i,j}$ indicates whether person i is assigned to seat j. This problem, being a mixed-integer quadratic programming problem, is NP-complete [1], but solvable in practice with tools like GLPK, particularly as real-time performance is not critical.

2.3.3 K-furthest points

One algorithmic approach to this problem is to set k seats as department hubs, corresponding to the number of departments represented in the booking that day. By incrementally assigning seats to departments based on their distance from each hub, taking into consideration the number of seats booked by each department, a full seating assignment is built up.

To ensure the seats in each department are close together, the department hubs are set to be as far away from each other as possible. Once these have been chosen, a seat is chosen that is closest to a department hub but will respect the number of seats needed for each department. This is repeated until all seats have been assigned.

2.3.4 Other Algorithmic Considerations

Alternative strategies were considered, including simpler but approximate methods like vertex cover approaches or clustering algorithms adapted for specific sized clusters. These approaches provided a backup or less computationally intensive option, ensuring robustness and flexibility in our system implementation.

2.3.5 Challenges and Solutions

Implementing these algorithms would pose several challenges, particularly in ensuring scalability and efficiency. Some solutions we came up with include:

- Preprocessing layouts and distances only once per office setup, reducing runtime overhead.
- Utilizing database checks to prevent double bookings and ensure seat allocations stayed within designated limits.

[1] `https://arxiv.org/abs/1407.4798`

2.3.6 Final Implementation Choice

Ultimately, we decided to shy away from the more math heavy optimisation formulation due to its complexity and dependence on outside tools, and instead opted for a more algorithmic approach by implementing the K-furthest points algorithm. We detail how we accomplished that in the following section.

2.3.7 K-furthest points Implementation

To select the set of k points that maximise the distance between themselves, first pick the single furthest point from the center, then repeatedly select the point that has the greatest minimum distance to each of the already selected points. Although this method is not guaranteed to select the optimum set, it will always produce an acceptable result. These seats are now the 'hubs' for each department, although each department isn't actually assigned to a hub yet. Instead, the 'sets' of seats are built up to match the list of numbers of seats booked by each department.

This is implemented in the form of a loop that assigns one seat to a set on each run through. On each loop it must be maintained that the assignment allows for the size of each set to match the department sizes by the time the loop terminates. Each set has a variable maximum size that decreases based on the sizes of the other sets. Once this maximum is reached the set becomes unavailable for assignment.

The maximum size of each set is initially the maximum number of seats that any department has booked. Considering a sorted list of the number of seats each department has booked, for the i-th entry at least k+1- i sets must have that many seats or fewer at all times. So, when i-1 sets are larger than the value of the i-th entry, the remaining k+1-i sets have their maximum sizes set to that value. This method of updating maximum sizes allows for flexibility, so that the departments aren't fixed to a hub initially and are moved to suit the seat layout.

Chapter 3

ERI

The ethical implications of our project are quite forthcoming. It is clear that an algorithm that determines where an individual employee may be working on any given day must be subject to scrutiny, both in and throughout the development process. During development, our primary concerns with regards to the ERI were that of employees with protected characteristics, more specifically neuro-divergence and accessibility requirements. To further this, we had constructed a survey to learn the user experiences of previous Hot-Desking systems - with a focus on the the experiences of those with the aforementioned characteristics. The following is an itemised list of the most significant development choices we had considered, and how their associated ethical implications influenced said choices.

- **Accessibility Room-Selection:** Initially, we had considered that during the creation of a particular office's floor plan through our platform that those with Admin positions - due to their familiarity with their office - could select the region of seats most appropriate for those with accessibility requirements. However, we had noticed that this may create an undesirable dynamic between those who require accessibility and higher management. Further, that this approach may effectively remove the agency of those who require accessibility.
- **Accessible Seat-Selection:** Analogous to above, we had considered that individual seats may be appropriate to be listed as 'accessible'. We had similar reservations as with the Room-selection idea, but further we had realised that accessibility requirements come in many different types. We

had decided that, given the general low-spec information we would have access to for any particular floor plan, that it would be reductive to have our algorithm always allocate individuals with accessibility required in previously assigned seats. As shown through our survey, the relationship these individuals have with their workspace are more appropriately handled through a more personal lens, and we would rather - in these instances - that our platform had more of an advisory role, rather than an authoritative one.

- **Future Development/Considerations:** We believe, as suggested by our survey, that employees who are neurodivergent/require accessibility would benefit from some block booking system, where they could be allocated the same seat over a number of different days. If development went further, we believe that this feature would certainly improve the user experiences of the aforementioned employees.

Chapter 4

Reflections

As a team, we embraced the diversity of our skills, which enhanced our project's scope and depth. Each member specialized in different aspects of the project, from front-end design and backend implementation to algorithm research and database management. This specialization allowed us to focus deeply on individual components.

Our bi-weekly meetings were instrumental in keeping the project on track and ensuring continuous feedback and iterative improvements. Through these meetings, we not only shared progress but also learned a lot, which proved essential for overcoming technical challenges and ensuring everyone was aligned with the project's goals.

We faced several technical challenges throughout the project. Through a combination of intensive research, creative problem-solving, and efficient implementation, we managed to overcome these and create a deliverable product.

Throughout the project, team members ventured out of their comfort zones, learning new technologies and frameworks. For instance, some of us who had never worked with React or FastAPI got the opportunity to learn and implement them hands-on. Additionally, the project improved our skills in areas such as API testing, database management, and front-end accessibility design.

Reflecting on the project, we all agree that it was a significant learning experience, teaching us the value of teamwork, effective communication, and agile project management in software development. We are proud of what we achieved together and are grateful for the supportive and enthusiastic environment that allowed each of us to contribute meaningfully whilst learning a lot. We believe the skills and experiences gained through this project will be invaluable in our future endeavors in technology and beyond.

Chapter 5

Next steps

Potential future enhancements include:

- **Cancellation feature:** Adding a 'cancel' button for bookings to provide users with more control over their reservations.
- **Office layout display:** Integrating a visual representation of the office layout to assist users in selecting preferred desks.
- **Granularity in booking times:** Allowing users to book based on timeslots in a day may add more flexibility.
- **Greater departmental synergy:** The algorithm could be further developed to take into account relationships between departments as well.

Appendix A

Volere Requirements

Below are the specifications we created that the project should achieve:

1.
   - **Description:** Employees can book a desk in a specific office for a particular day.
   - **Rationale:** To guarantee workspace availability.
   - **Fit Criterion:** Employee can select a date and office location and receive a confirmation of the booking.
   - **Technical Details:** Integration with calendar API for date selection. Real-time database update on booking.
   - **Priority:** High

2.
   - **Description:** The system allocates seats to enhance team cohesion.
   - **Rationale:** To improve teamwork and efficiency.
   - **Fit Criterion:** Allocation algorithm prioritises placing team members in proximity.
   - **Technical Details:** The algorithm is given a list of desks and their positions, as well as a list of users and their departments. It assigns each seat a department, so that all users are represented, then randomly assigns each seat within the department, paying no attention to the user themself.
   - **Priority:** High

3.
- **Description:** Desks are allocated on a first-come, first-served basis.
- **Rationale:** To ensure fairness and transparency.
- **Fit Criterion:** Desks are assigned based on the order of booking requests received.
- **Technical Details:** Booking system rejects further bookings when the office capacity has been reached.
- **Priority:** High

4.
- **Description:** Employees book desks for single-day use.
- **Rationale:** To maintain flexibility and availability.
- **Fit Criterion:** System restricts bookings to one day at a time.
- **Technical Details:** Restriction logic in booking function to limit reservation period.
- **Priority:** High

5.
- **Description:** The booking system is easy and efficient for reservations.
- **Rationale:** To facilitate ease of use.
- **Fit Criterion:** User satisfaction testing.
- **Technical Details:** Intuitive UI/UX design, possibly using a framework like React or Angular for frontend.
- **Priority:** Medium

6.
- **Description:** Booking system accessible via a webpage.
- **Rationale:** To ensure accessibility from any device.
- **Fit Criterion:** System is accessible and functional on multiple web browsers.
- **Technical Details:** Compatible with various screen sizes and browsers.
- **Priority:** Medium

7.
- **Description:** Users can securely sign up and log into the system.
- **Rationale:** To personalise and secure bookings.
- **Fit Criterion:** System supports secure authentication and user data privacy.
- **Technical Details:** Perhaps this could be implemented OAuth2
- **Priority:** Medium

8.
- **Description:** System adaptable to different office layouts.
- **Rationale:** To be usable across various company locations.
- **Fit Criterion:** System supports configuration for different office floor plans.
- **Technical Details:** Seats are implemented as coordinates on a 2d plane, hence can be easily changed.
- **Priority:** Medium

9.
- **Description:** Notifications provided through a main page or individually.
- **Rationale:** To keep employees well-informed.
- **Fit Criterion:** Users receive timely and clear notifications regarding their reservations.
- **Technical Details:** Could be shown on the main webpage, or perhaps sent individually through email.
- **Priority:** Medium

10.
- **Description:** Admin controls for user access and issue resolution.
- **Rationale:** To manage system integrity and user issues.
- **Fit Criterion:** Administrators can adjust user permissions and resolve common system issues.
- **Technical Details:** Role-based access control for different user levels.

- **Priority:** Low

11.
- **Description:** Option to book a desk for part of a day.
- **Rationale:** To provide flexibility.
- **Fit Criterion:** System allows booking for specific time slots within a day.
- **Technical Details:** Time slot management integrated into the booking functionality.
- **Priority:** Low

12.
- **Description:** Easy cancellation of desk bookings.
- **Rationale:** To free up desks for others.
- **Fit Criterion:** Users can cancel reservations with immediate system update.
- **Technical Details:** Cancellation feature linked with real-time database updates to free up slots immediately.
- **Priority:** Low