SOFTWARE REQUIREMENTS SPECIFICATION

Occupi - Office Capacity Predictor

Name	Student Number
Rethakgetse Manaka	u22491032
Kamogelo Moeketse	u22623478
Michael Chinyama	u21546551
Tinashe Austin	u21564176
Carey Mokou	u21631532

June 24, 2024

Contents

Introduction

The Occupi-Office Capacity Management system is designed to revolutionize the way office space is managed by integrating advanced Machine Learning and predictive models. This innovative system addresses the challenge of efficiently managing office occupancy by providing real-time updates on the current capacity and predicting future occupancy trends.

By leveraging historical data and real-time inputs, the Occupi system enables office managers to make informed decisions about space utilization. It not only provides immediate insights into current occupancy levels but also offers longer-term predictions, facilitating better planning and allocation of resources. This planning capability allows the system to enhance its predictive accuracy over time, creating a dynamic and responsive tool for office capacity management.

The system benefits both daily users and office owners. Employees and visitors can access real-time updates on the current office capacity, ensuring they are aware of occupancy levels before arriving. Office managers and owners receive detailed predictions on future capacity, empowering them to optimize space usage, plan for peak periods, and enhance overall office efficiency.

In summary, the Occupi-Office Capacity Management system introduces a data-driven approach to managing office space, offering both immediate and long-term solutions to enhance occupancy management and improve office operations.

Purpose and Vision

The Occupi-Office Capacity Management system aims to revolutionize office space management by integrating advanced Machine Learning and predictive models. The system addresses the challenge of efficiently managing office occupancy by providing real-time updates on current capacity and predicting future occupancy trends. This enables office managers to make informed decisions about space utilization, ensuring optimal use of office resources.

The vision of the Occupi-Office Capacity Management system is to create a dynamic and responsive tool that benefits both daily users and office owners. By leveraging historical data and real-time inputs, the system offers immediate insights into current occupancy levels and detailed predictions for future capacity. This facilitates better planning and resource allocation, allowing the system to enhance its predictive accuracy over time. Ultimately, the system seeks to introduce a data-driven approach to office capacity management, improving office operations and space efficiency.

Specifications

System Requirements

Occupi is compatible with the following operating systems:

- Windows
- Linux
- MacOS
- Android
- iOS

Internet connection is required for the system to function optimally.

User Stories and Characteristics

Characteristics

Employees Employees are the primary users of the Occupi-Office Capacity Management system. Their main goal is to efficiently manage their work schedules by leveraging real-time occupancy data and predictions.

- **Check-In:** Employees want to use the app to check in when they arrive at the office, providing the system with accurate occupancy data.
- **View Current Office Capacity:** Employees want to use the system to quickly view the current number of people in the office through the app, helping them decide when to come in or find a less crowded time.
- **Plan Visits:** By accessing occupancy predictions, employees want to plan their visits around expected busy times, enhancing their productivity and ensuring a comfortable working environment.
- **Notifications and Alerts:** Employees want to receive notifications about significant changes in occupancy or office conditions, such as reaching maximum capacity or scheduled maintenance.

Managers Managers use the Occupi system to make strategic decisions regarding office space utilization and to ensure a smooth operation of the office environment. Managers want to manage their teams effectively and optimize office resources.

- **Historical Trends:** Managers want to access detailed historical data on office occupancy, helping them identify patterns and trends over time.
- **Space Planning:** Using predictions and historical data, managers want to plan the allocation of office space, schedule shifts, and manage peak times effectively.
- **Reporting:** Managers want to generate reports on office usage, occupancy trends, and space efficiency to inform higher-level strategic decisions.
- **Team Management:** Managers want to generate reports on team attendance rates, productivity, and collaboration patterns, helping them optimize team performance and office layout.

System Administrators System Administrators are responsible for maintaining the overall functionality, data integrity, and performance of the Occupi system.

- **Data Accuracy:** Administrators want to ensure that the data collected by the sensors and other devices is accurate and reliable.

- **System Performance:** Administrators want to monitor system performance, addressing any issues that arise and ensuring that the application runs smoothly.
- **Security:** Administrators want to manage the security of the system, including user access controls, data encryption, and compliance with relevant data protection regulations.
- **Maintenance and Updates:** They perform regular maintenance and updates to the system, ensuring it remains up-to-date with the latest features and security patches.
- **Technical Support:** Provide technical support to other users, troubleshooting issues and providing guidance on using the system effectively.

General Staff General Staff such as facility management or administrative staff use the system to manage the physical aspects of the office environment.

- **Room Booking for Cleaning:** General Staff want to book specific rooms for cleaning through the system, ensuring that all areas are properly maintained without disrupting employees.
- **Maintenance Scheduling:** General Staff want to schedule and manage maintenance activities, disabling rooms or areas as needed to perform repairs or upgrades.
- **Notifications:** General Staff want to receive notifications about required maintenance or cleaning tasks, allowing them to manage their workload effectively.
- **Resource Management:** General Staff want to oversee the allocation and usage of office resources, such as cleaning supplies and maintenance tools, based on occupancy data and predictions.

Stories

Employees

- As an employee, I want to log in using my credentials to access the Occupi system.
- As an employee, I want to check in when I arrive at the office to update the occupancy data.
- As an employee, I want to see the real-time office capacity so that I can decide whether to come to the office.
- As an employee, I want to view predicted office capacity for the next two days so that I can plan my office visits accordingly.
- As an employee, I want to update my profile information so that my contact details and preferences are accurate.
- As an employee, I want to receive notifications about office capacity updates so that I am informed if the office is full.
- As an employee, I want to provide feedback on the app so that the development team can improve it.
- As an employee, I want to navigate the app easily so that I can find information quickly.
- As an employee, I want to use the application in my preferred language so that I can navigate it easily.
- As an employee, I want to search for occupancy data by date so that I can find specific information quickly.
- As an employee, I want to be able to report issues with rooms or facilities so that they can be addressed promptly.

Managers

- As a manager, I want to view historical occupancy data so that I can identify trends and patterns.
- As a manager, I want to generate reports on office occupancy so that I can make informed decisions.
- As a manager, I want to receive alerts about office capacity changes so that I can respond quickly.
- As a manager, I want to view team attendance rates so that I can optimize team performance.
- As a manager, I want to export occupancy data so that I can analyze it externally or share it with stakeholders.

- As a manager, I want to search for occupancy data by date so that I can quickly retrieve specific information for analysis or reporting.
- As a manager, I want to receive feedback on the app from users so that I can understand user needs and improve the application.
- As a manager, I want to receive a daily summary report of office occupancy so that I can review daily trends.
- As a manager, I want to see visual representations of historical data so that I can understand trends better.

System Administrators

- As a system administrator, I want to monitor system performance so that I can identify and address issues.
- As a system administrator, I want to manage user access controls so that I can ensure data security.
- As a system administrator, I want to update the system regularly so that it remains secure and up-to-date.
- As a system administrator, I want to receive alerts about system issues so that I can respond quickly.
- As a system administrator, I want to provide technical support to users so that they can use the system effectively.
- As a system administrator, I want to manage data encryption so that user data is secure.
- As a system administrator, I want to view audit logs of check-ins and check-outs so that I can track user actions for security purposes.
- As a system administrator, I want to integrate the app with building access control systems so that check-in data is automatically updated.
- As a system administrator, I want to set capacity alert thresholds so that I am notified when occupancy reaches critical levels.

General Staff

- As general staff, I want to book rooms for cleaning through the system so that I can manage cleaning schedules effectively.
- As general staff, I want to schedule maintenance activities through the system so that I can plan maintenance tasks efficiently.
- As general staff, I want to receive notifications about required maintenance or cleaning tasks so that I can manage my workload effectively.

Class Diagrams

Functional Requirements

Requirements

Requirements with a ***** next to it represents optional and *nice to have* requirements.

1. Provide a secure authentication process.
2. Users must be able to log in using their credentials.
3. Users must be able to log in using their Deloitte email address and password.
4. Users must be able to log out of the system.
5. Users must be able to reset their password.
6. Users onboard using an OTP that expires after 10 minutes.
7. Provide a user-friendly interface.
8. Users must be able to navigate the app easily.
9. Users must be able to search for information quickly.
10. Users must be able to use the app in their preferred language.
11. Users must be able to choose between different themes: 'Light' and 'Dark'.
12. Provide real-time office capacity updates.
13. Users must be able to check in when they arrive at the office.
14. Users must be able to view the current office capacity.
15. Users must be able to receive notifications about office capacity updates.
16. Provide office capacity predictions.
17. Users must be able to view predicted office capacity for the next two days.
18. Users must be able to plan their office visits accordingly.
19. Provide historical occupancy data.
20. Managers must be able to view historical occupancy data.
21. Managers must be able to identify trends and patterns.
22. Managers must be able to generate reports on office occupancy.
23. Generate reports on office occupancy.
24. Managers must be able to generate reports on office occupancy for the past week.

25. Managers must be able to generate reports on office occupancy for the past month.
26. Managers must be able to generate reports on office occupancy for the past year.
27. Managers must be able to generate reports on team attendance rates.
28. Managers must be able to generate reports on team attendance rates for the past week.
29. Managers must be able to generate reports on team attendance rates for the past month.
30. Managers must be able to generate reports on team attendance rates for the past year.
31. Managers must be able to generate reports on team attendance rates for a specific individual.
32. Managers must be able to generate reports on team attendance rates for a specific date.
33. Manage user access controls.
34. System administrators must be able to manage user access controls.
35. System administrators must be able to specify user roles.
36. System administrators must be able to assign users to specific roles.
37. System administrators must be able to revoke user access.
38. System administrators must be able to set capacity alert thresholds.
39. System administrators must be able to ensure data security.
40. Provide booking functionality.
41. Users must be able to book specific rooms for specific time periods.
42. Users must be able to view room availability.
43. Users must be able to cancel bookings.
44. Users must be able to invite other users to join booking room.
45. Provide Check-in functionality.
46. Users must be able to check in for a specific booking.
47. Users must be able to check in using booking details and credentials.
48. System must detect that users are checking from a specific location.
49. System must implement automatic check out.
50. Provide maintenance scheduling.
51. General staff must be able to schedule maintenance activities through the system.
52. General staff must be able to book rooms for cleaning through the system.

53. Provide notifications.
54. Users must be able to receive notifications about maintenance taking place.
55. Users must be able to receive notifications about office capacity updates.
56. Users must be able to receive notifications about their bookings.

Subsystems

1. User Management
2. Profile Management
3. Predictive Model to graph
4. Security Management
5. Room Management
6. Analytics and Reporting

Use Case Diagrams

Profile Management Subsystem

User Management Subsystem

Room Management Subsystem

Quality Requirements

Performance

The performance of the Occupi system is critical to its success. The system will be accessed by a large number of users and it must be able to handle high traffic loads without slowing down or crashing. The system should respond quickly to user requests, providing real-time updates on office capacity and occupancy predictions. To ensure optimal performance, the system should be able to scale horizontally to accommodate additional users and data. The system should also be able to handle peak loads during busy periods, such as the start and end of the workday. Performance testing should be conducted regularly to identify and address any bottlenecks or issues that may arise.

Reliability

The reliability of the Occupi system is essential to ensure that users can access the system when they need it. The only time that is acceptable for the system to be down is during maintenance. To ensure reliability, the system should be designed with redundancy and failover mechanisms to prevent downtime. Regular backups of data should be performed to prevent data loss in the event of a system failure. Data should be backed up every 7 days and be recoverable within 24 hours. The system should also be monitored for performance and availability, with alerts sent to administrators if any issues are detected.

Scalability

The Occupi system should be able to scale to accommodate additional users, data, and features as needed. The architecture of the system should be designed to support horizontal scaling. This means that additional servers can be added to the system to handle increased traffic and data without affecting performance. The system should also be able to scale vertically, allowing individual components to be upgraded to handle increased load.

Security

The security of the Occupi system is paramount to protect user data and ensure the integrity of the system. Due to the system collecting data on users, it is important that the system is secure. The system should implement strong encryption to protect data in transit and at rest. User authentication should be implemented to ensure that only authorized users can access the system. Access controls should be in place to restrict user access to sensitive data and features. User authentication should take place every 90 days. User authentication systems such as OAuth should be implemented to ensure that user data is secure. Domain-specific access should be implemented to ensure that only users with the correct domain can access the system.

Usability

The Occupi system should be easy to use and navigate, with an intuitive interface that allows users to find information quickly and easily. The system should be accessible and user-friendly, with clear instructions and guidance provided to help users understand how to use the system. The system should follow Deloitte design cues and be consistent across all platforms. The system should also be responsive, adapting to different screen sizes and devices to provide a consistent user experience. User feedback should be collected regularly to identify areas for improvement and enhance the usability of the system.

Interoperability

The Occupi system should integrate seamlessly with Deloitte's building access control system. This allows the system to automatically update occupancy data based on user check-ins and check-outs. The system should also be compatible with a wide range of devices and platforms, including desktops, laptops, tablets, and smartphones. The system should also integrate with the building's network infrastructure to ensure that users are checking in from the correct location. The system should be able to keep track of trends in the Deloitte tuckshop by integrating with the tuckshop system.

Compliance

Due to personal data being collected, the Occupi system must comply with relevant data protection regulations, such as the Protection of Personal Information Act (POPIA). The system should implement data protection measures to ensure that user data is secure and protected from unauthorized access. The system should also comply with industry standards and best practices for data security and privacy. Regular audits and reviews should be conducted to ensure that the system remains compliant with Deloitte's data protection policies and relevant regulations.

Compatibility

The Occupi system should be compatible with a wide range of devices and platforms, including desktops, laptops, tablets, and smartphones. The system should be accessible from any device with an internet connection, allowing users to access real-time updates on office capacity and occupancy predictions. The system must be compatible with the following operating systems:

- Windows
- Linux
- MacOS
- Android

- iOS

Design Strategy

The architectural design strategy for the development of Occupi involves the following: Agile and Decomposition.

Agile Strategy

This strategy focuses on iterative development, continuous feedback from clients, cross-functional teams, as well as adaptability. Iterative development involves work being divided into small, manageable increments. This aids the development process of the project in that it allows the team to break parts of the projects apart and focus on one part at a time. This makes the project more manageable. Continuous feedback from the client helps to ensure that the product meets their needs. This helps to prevent the team from diverting from the requirements of the product.

Decomposition Strategy

This strategy emphasizes breaking down complex projects into smaller, more manageable components. Decomposition involves dividing the project into discrete, self-contained modules or tasks. This approach enhances the development process by allowing the team to concentrate on individual segments, making the overall project easier to handle. Each module can be developed, tested, and refined independently before integrating it into the larger system. Regular assessments and adjustments at each stage ensure that each part aligns with the overall project goals. This strategy minimizes risks, improves resource allocation, and ensures that the final product meets the desired standards and specifications.

Architectural Design & Patterns

MVC (Model-View-Controller)

We chose the MVC to separate concerns for user interface, business logic, and data management. The MVC pattern promotes a clean separation of concerns between data management (Model), user interface (View), and application logic (Controller), making our codebase more modular and easier to maintain.

Model: Database - The Model represents the core backend of our system, managing the data and the business logic. It serves as the central source of truth for all other views in the system. In our architecture, the Model includes:

- **MongoDB:** The primary database for storing and retrieving application data. MongoDB's NoSQL structure allows for flexible, scalable, and high-performance data management.

View: Frontend - The View is responsible for displaying data to the user and capturing user input. It is the user interface layer of the application. In our architecture, the View includes:

- **Web Frontend (React):** The client-side part of the application for our Management Terminal, built using React. It renders the user interface, allowing users to interact with the system. It displays dashboards and other data visualization elements to assist in understanding office capacity trends.
- **Mobile Frontend (React Native):** The client-side part of the application for mobile users, built using React Native. It provides a seamless and responsive user experience on mobile devices whilst booking, checking in, and viewing office occupancy.

Controller: AI Model Server - The Controller acts as an intermediary between the Model and the View. It processes incoming requests, performs operations on the Model, and updates the View accordingly. In our architecture, the Controller includes:

- **Python AI Model:** Integrates AI-related operations such as data processing, making predictions, and providing intelligent responses. This component works closely with the Go server to handle AI tasks.

Layered Pattern

- **Presentation Layer:** This layer consists of the user interfaces such as the mobile application for employees and the management terminal for managers. It handles user input, displays data, and provides a consistent user experience adhering to Deloitte's design standards and guidelines (Usability and Compliance).

- **Application Layer:** This layer encapsulates the business logic and rules related to occupancy tracking, prediction algorithms, and other core functionalities of the system. It serves as an intermediary between the Presentation Layer and the lower layers, ensuring proper separation of concerns.
- **Service Layer:** This layer consists of various services or components responsible for specific tasks such as user management, occupancy tracking, prediction engine, and real-time updates. These services could be implemented as microservices, promoting scalability and maintainability.
- **Data Access Layer:** This layer abstracts the underlying data storage mechanisms (e.g., MongoDB) and provides a consistent interface for interacting with the data sources. It handles tasks such as querying, updating, and retrieving data related to occupancy, historical trends, and user information.
- **External Systems Layer:** This layer facilitates integration with external systems such as Deloitte's existing building access control systems. It handles communication protocols, data transformations, and any necessary adaptations required for seamless integration (Interoperability).
- **Security Layer:** This layer handles authentication, authorization, and other security-related concerns such as encryption and data protection (Security). It could be implemented as a cross-cutting concern or as a separate layer depending on the complexity of the security requirements.

Client Server

Client: The client in this instance acts as any vector that forwards requests to the backend. Clients can access the server from multiple endpoints and different environments including that of production and development environments. Our domain registrar handles DNS for the clients and forwards requests to the IP address of the server.

Server: We have a singular VM instance that acts as a server which listens for incoming requests on port 443 (HTTPs) and serves clients multiple different pages depending on what URL they sent in. A client may be served our documentation site, the landing page, the actual web app, or the API backend (which may interact with the database) which are all hosted in containers. A reverse proxy sits between port 443 and our containers.

Technology Choices

For Web-based Development

React: A JavaScript library for building user interfaces. It's component-based which promotes reusability and makes code easier to manage. However, the learning curve can be steep for beginners.

- **Pros:**
  - Component-based architecture allows for modular and reusable code.
  - Large ecosystem with a vast array of libraries and tools.
  - Virtual DOM for efficient rendering and updates.
  - Supported by a large community and backed by Facebook.

Tailwind CSS: A utility-first CSS framework that allows for highly customizable designs. It can speed up development, but the class-based approach might seem verbose to some.

- **Pros:**
  - Utility-first approach allows for rapid styling and prototyping.
  - Highly customizable and flexible.
  - Promotes consistency and maintainability across projects.

Component libraries (Schadcn, DaisyUI, NextUI, Framer Motion): These libraries provide pre-built components that can speed up development and ensure consistency across the app. However, they might limit customization to some extent.

- **Pros:**
  - Accelerates development by providing pre-built UI components.
  - Ensures consistent design and styling across the application.
  - Well-documented and actively maintained by communities.

For Mobile Development

React Native: A framework for building native apps using React. It allows for code sharing between platforms which can reduce development time. However, it might not be as performant as native code for complex applications.

- **Pros:**
  - Write once, run anywhere (iOS and Android) with native performance.
  - Large community and extensive ecosystem of libraries and tools.
  - Easier learning curve for developers familiar with React.

Component libraries (UI Kitten, GlueStackUI): Similar to web development, these libraries can speed up development and ensure consistency. The same potential limitation in customization applies.

- **Pros:**
  - Pre-built components specifically designed for React Native.
  - Consistent look and feel across platforms.
  - Actively maintained and well-documented.

Reasons for Frontend

We chose React for our project due to its component-based architecture which promotes modular and reusable code and its efficient rendering capabilities via the Virtual DOM. Tailwind CSS was selected for its utility-first approach, enabling rapid styling and prototyping with high customizability and consistency. Additionally, component libraries like Schadcn, DaisyUI, NextUI, and Framer Motion offer pre-built components that accelerate development and maintain a consistent design across the app, despite some limitations in customization. This combination of tools provides a robust and efficient development environment backed by a large community and extensive ecosystem.

For Backend Development

Go: A statically typed, compiled language that is efficient and excellent for building web services and microservices. However, its simplicity can be a limitation for some complex applications.

- **Pros:**
  - Statically typed language which promotes code safety and reliability.
  - Excellent performance and high concurrency support.
  - Simple and clean syntax, easy to learn and maintain.
  - Built-in support for concurrency and parallelism.

MongoDB driver support: This allows for rapid development and deployment, especially for web services, microservices, and cloud-native apps. However, MongoDB might not be the best choice for applications that require complex transactions with multiple operations.

- **Pros:**
  - Supports a flexible and scalable NoSQL database.
  - Enables rapid development and deployment of web services and microservices.
  - Well-suited for cloud-native applications and distributed systems.
  - Requires a different mindset and approach compared to traditional relational databases.

REST API, JSON Web Tokens: REST is a standard for designing networked applications. It can be simpler and more scalable than other approaches, but it might not be as real-time as some applications require. JSON Web Tokens provide a secure way of transmitting information between parties. However, they can be vulnerable if not handled properly.

- **Pros:**
  - REST APIs provide a standard and widely adopted architectural style for building web services.
  - JWTs offer a compact and secure way to transmit claims between parties.
  - Stateless authentication mechanism reducing server-side overhead.

Reasons for Backend

For backend development, we selected Go due to its statically typed nature which promotes code safety and reliability. Go's excellent performance, high concurrency support, and simple clean syntax make it ideal for building web services and microservices. Its built-in support for concurrency and parallelism further enhances its efficiency. We also chose MongoDB for its flexible and scalable NoSQL database capabilities, enabling rapid development and deployment, especially for web services, microservices, and cloud-native applications. Additionally, we opted for REST API and JSON Web Tokens (JWTs) for their simplicity, scalability, and secure way of transmitting information. REST APIs provide a standard architectural style for building web services, while JWTs offer a compact, secure, and stateless authentication mechanism reducing server-side overhead.

Architectural Design