SOFTWARE REQUIREMENTS SPECIFICATIONS

Occupi - Office Capacity Predictor

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

September 01, 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.

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 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.

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 see visual representations of historical data so that I can understand trends better.

Functional Requirements

Requirements

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

1. Provide a secure authentication process.
   (a) Users must be able to log in using their credentials.
   (b) Users must be able to log in using their Deloitte email address and password.
   (c) Users must be able to log out of the system.
   (d) Users must be able to reset their password.
   (e) Users onboard using an OTP that expires after 60 minutes.
2. Provide a user-friendly interface.
   (a) Users must be able to navigate the app easily.
   (b) Users must be able to search for information quickly.
   (c) Users must be able to use the app in their preferred language.
   (d) Users must be able to choose between different themes: 'Light' and 'Dark'.
3. Provide real-time office capacity updates.
   (a) Users must be able to check in when they arrive at the office.
   (b) Users must be able to view the current office capacity.
   (c) Users must be able to receive notifications about office capacity updates.
4. Provide office capacity predictions.
   (a) Users must be able to view predicted office capacity for the next two days.
   (b) Users must be able to plan their office visits accordingly.
5. Provide historical occupancy data.
   (a) Managers must be able to view historical occupancy data.
   (b) Managers must be able to identify trends and patterns.
6. Generate reports on office occupancy.
   (a) Managers must be able to generate reports on office occupancy for the past week, month, and year.
   (b) Managers must be able to generate reports on team attendance rates, both for the past periods and for specific individuals or dates.
7. Manage user access controls.
   (a) System administrators must be able to manage user access controls.

   (b) System administrators must be able to specify and assign user roles.
   (c) System administrators must be able to revoke user access.
   (d) System administrators must be able to ensure data security.

8. Provide booking functionality.
   (a) Users must be able to book specific rooms for specific time periods.
   (b) Users must be able to view room availability.
   (c) Users must be able to cancel bookings.
   (d) Users must be able to invite other users to join a booking.

9. Provide check-in functionality.
   (a) Users must be able to check in for a specific booking.
   (b) Users must be able to check in using booking details and credentials.
   (c) The system must detect that users are checking in from a specific location.
   (d) The system must implement automatic check-out.

10. Provide notifications.
   (a) Users must be able to receive notifications about maintenance or capacity updates.
   (b) Users must be able to receive notifications about their bookings.

Subsystems

The Occupi system is divided into several subsystems to manage different aspects of functionality and data handling.

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

User Management Subsystem

This subsystem manages all aspects of user accounts, including registration, authentication, and role-based access controls (RBAC). It ensures that users can log in, update their profiles, and access specific features based on their assigned roles.

Profile Management Subsystem

This subsystem handles all user profile-related data, including preferences, language settings, and personal details. It allows users to update their profiles and ensures that profile information is accurate and consistent.

Predictive Model to Graph Subsystem

This subsystem is responsible for generating occupancy predictions based on historical data and real-time inputs. It communicates with the AI model to fetch predictions and then renders the predictions visually in graphs and charts for easier interpretation by the user.

Security Management Subsystem

The security subsystem handles encryption, user authentication, and data protection. It ensures compliance with industry standards like OAuth 2.0 for secure authorization and TLS 1.2+ for data in transit.

Room Management Subsystem

The room management subsystem allows users to book office spaces and meeting rooms. It manages availability, booking times, and room capacity, ensuring users can efficiently manage space resources.

Analytics and Reporting Subsystem

This subsystem enables managers to generate detailed reports on office capacity, team attendance, and historical trends. It provides customizable reporting options for both short-term and long-term analysis.

Use Case Diagrams

User Management Subsystem

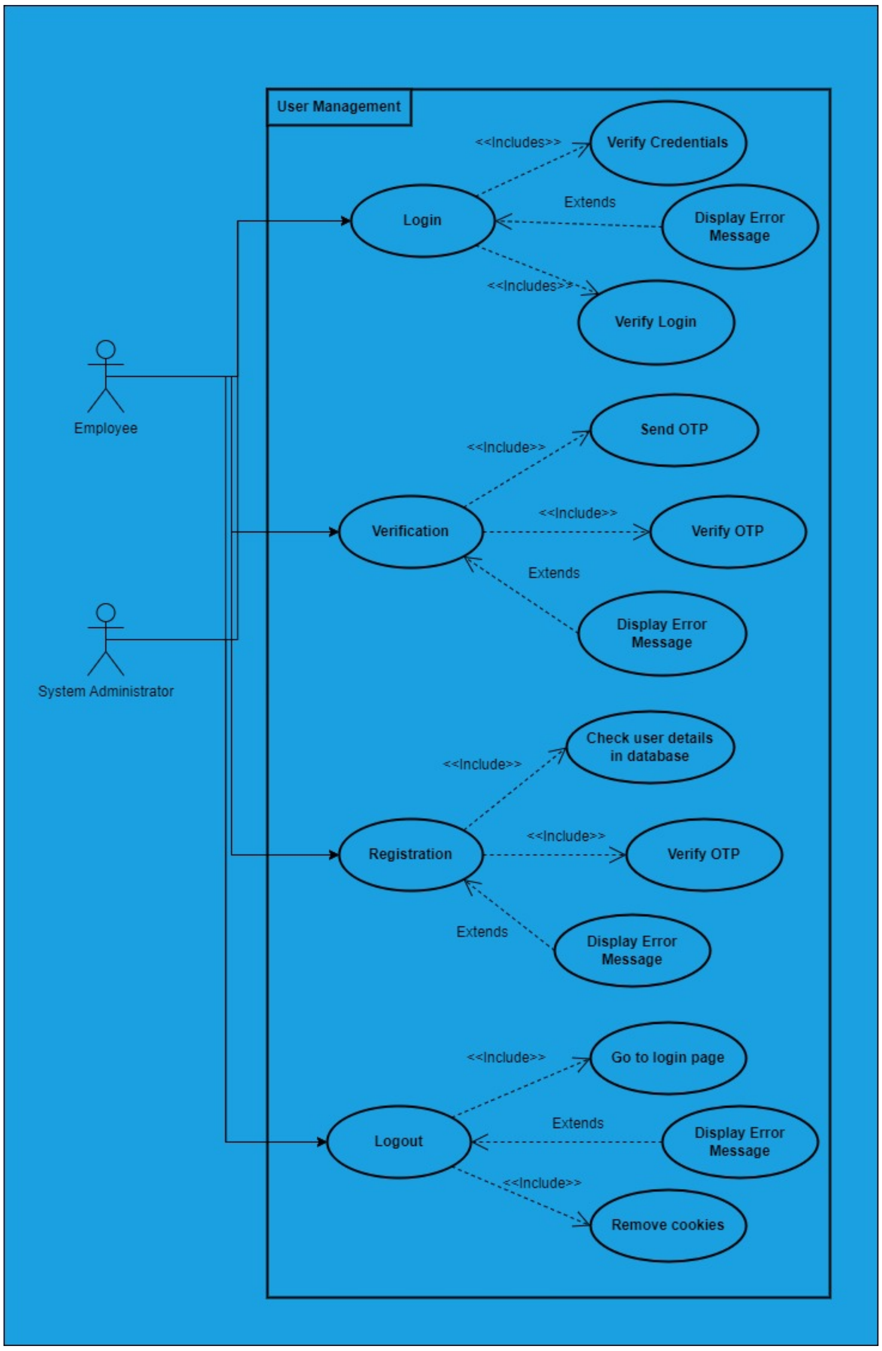


Figure 1: User Management Subsystem

Profile Management Subsystem

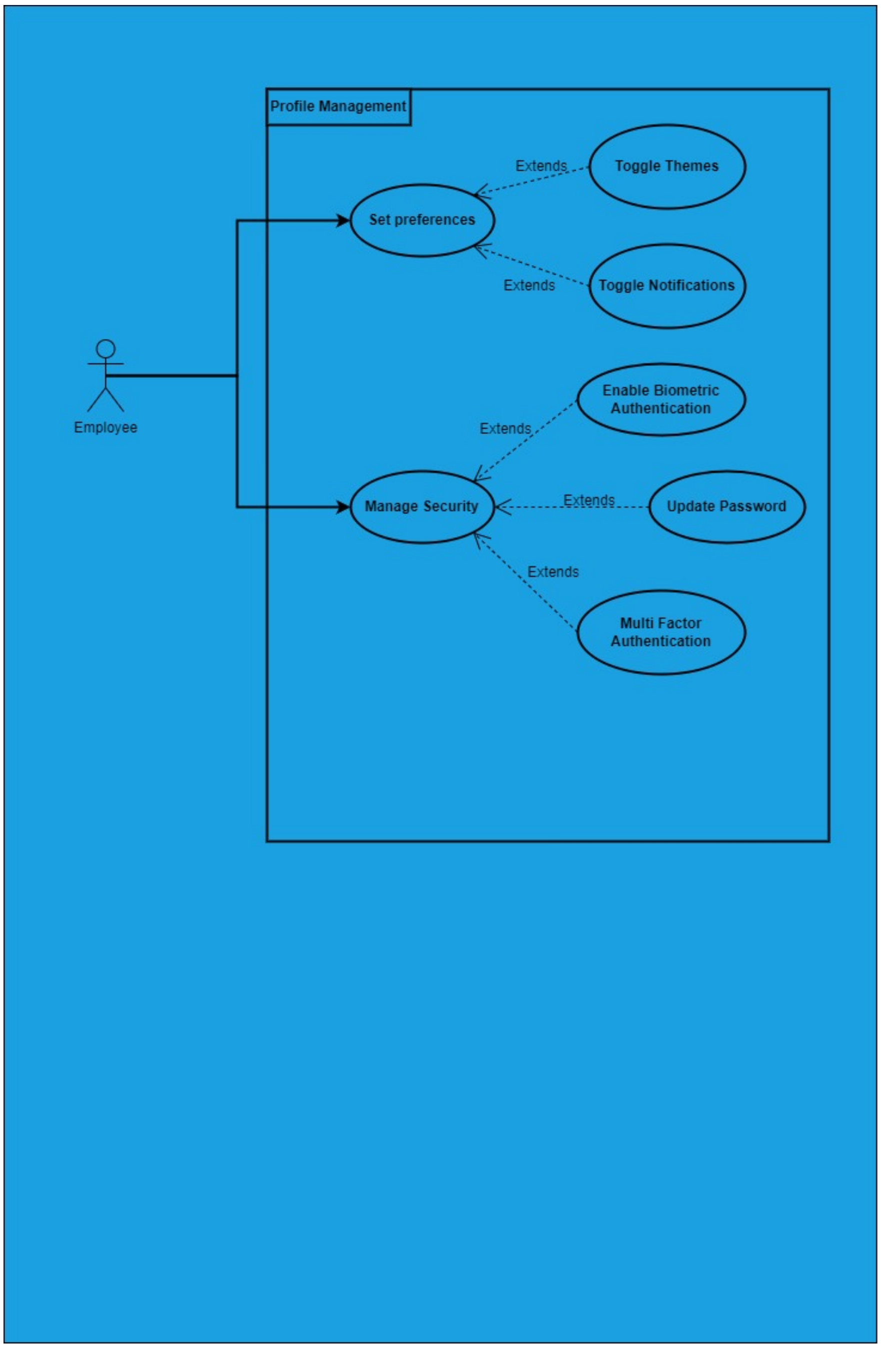


Figure 2: User Management Subsystem

Predictive Model to Graph Subsystem

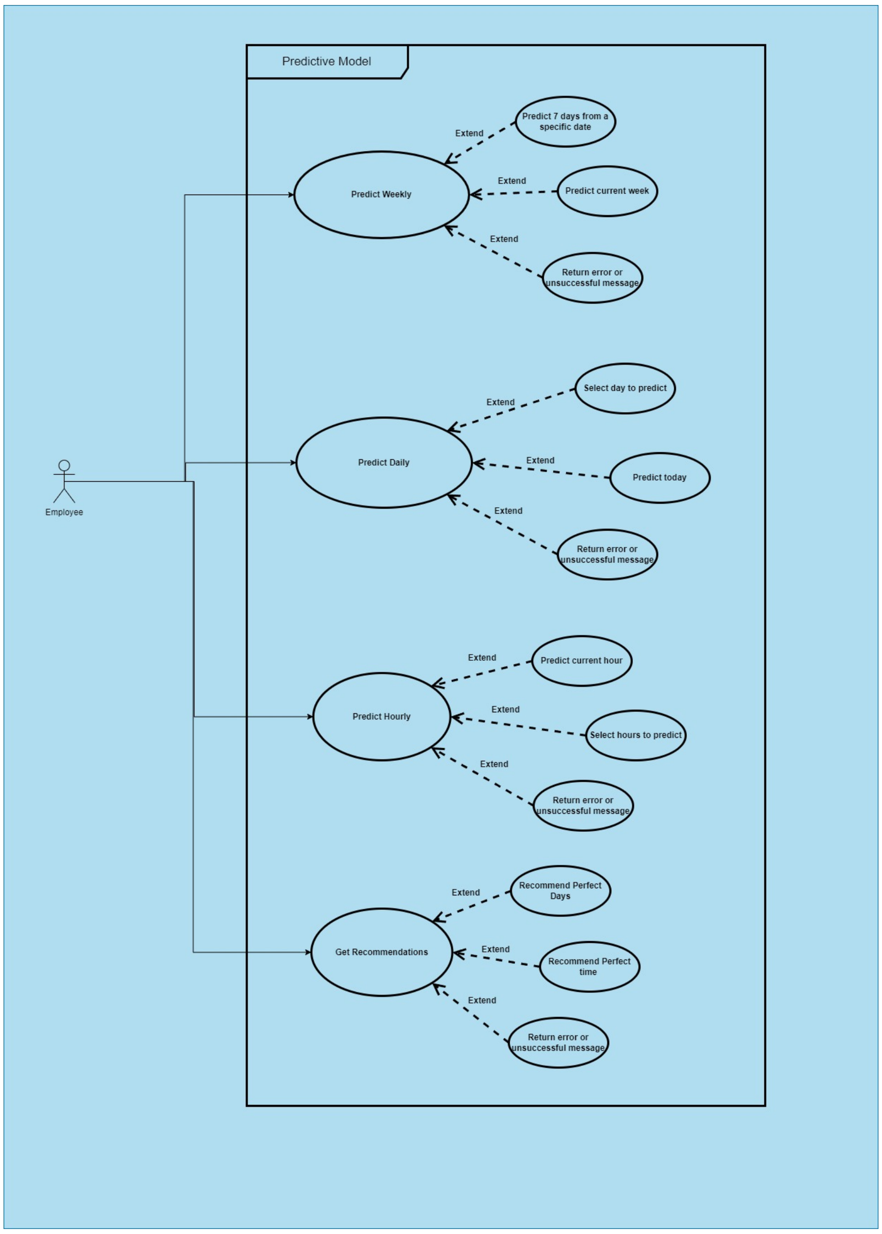


Figure 3: User Management Subsystem

Security Management Subsystem

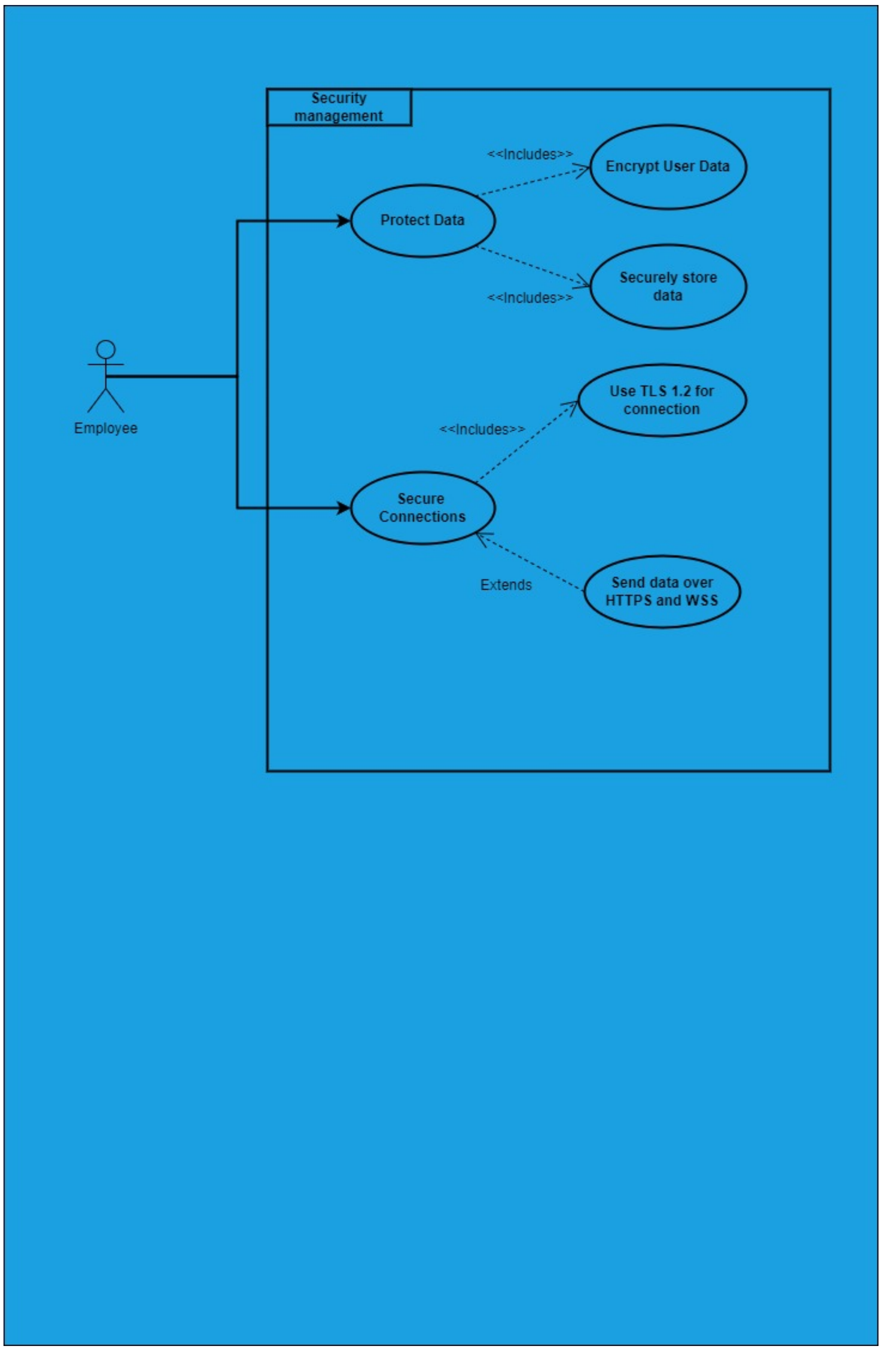


Figure 4: User Management Subsystem

Room Management Subsystem

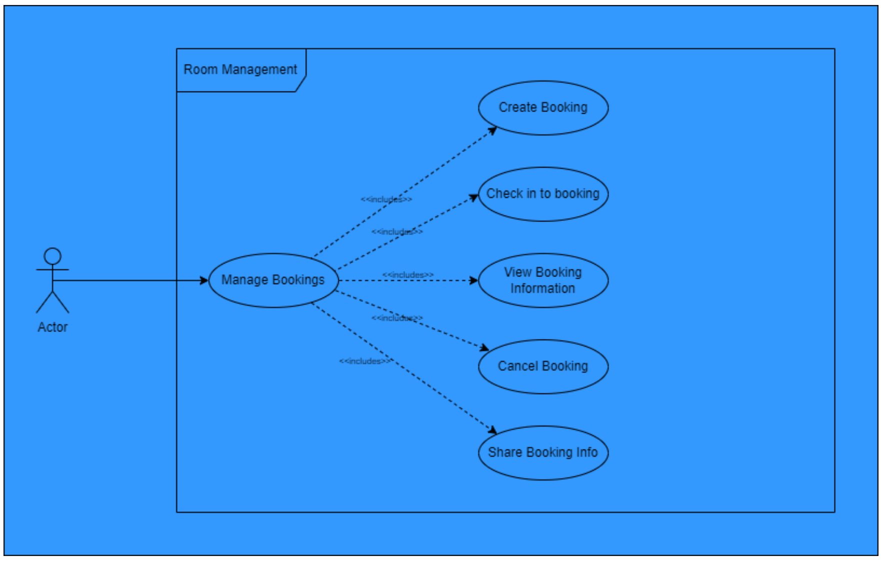


Figure 5: Room Management Subsystem

Analytics and Reporting Subsystem

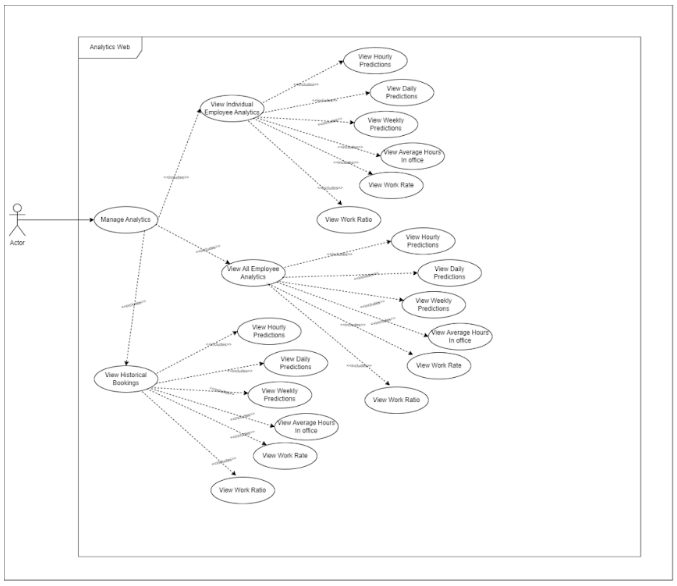


Figure 6: Analytics and Reporting Subsystem

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 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 acceptable downtime is during scheduled maintenance. 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 case of system failure. Data should be backed up every 7 days and be recoverable within 24 hours. Additionally, the system should be monitored for performance and availability, with alerts sent to administrators if issues are detected.

Scalability

The Occupi system should be able to scale to accommodate additional users, data, and features as needed. The system should be designed to support horizontal scaling, allowing for the addition of servers to manage increased traffic and data without affecting performance. Vertical scaling should also be supported, allowing individual components to be upgraded to handle increased loads.

Security

The security of the Occupi system is paramount to protect user data and ensure the integrity of the system. The system should implement strong encryption protocols to protect data in transit and at rest. User authentication should be implemented to ensure that only authorized users access the system, with role-based access controls (RBAC) in place to restrict access to sensitive data and features. Re-authentication should occur every 90 days using OAuth 2.0 for secure authorization, and domain-specific access should be implemented to ensure that only users with the correct domain 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. The system should follow Deloitte's design cues to ensure consistency across all platforms. It should also be responsive, adapting to different screen sizes and devices to provide a seamless user experience. User feedback should be collected regularly to identify areas for improvement and enhance the usability of the 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. Compliance with industry standards and best practices for data security and privacy is essential. Regular audits and reviews should be conducted to ensure ongoing compliance with Deloitte's data protection policies and relevant regulations.

Interoperability

The Occupi system should integrate seamlessly with Deloitte's existing building access control systems. This integration allows the system to automatically update occupancy data based on user check-ins and check-outs. The system should 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.

Compatibility

The Occupi system should be compatible with the following operating systems:

- Windows
- Linux
- MacOS
- Android
- iOS

It should also be accessible from any device with an internet connection, allowing users to access real-time updates on office capacity and occupancy predictions.

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.

Quality Requirements

Performance

Description: 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 manage 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.

Quantification: To quantify the performance of Occupi, several key metrics are essential, and each will be measured using specific methods. The system should support at least 10,000 concurrent users. It must process a minimum of 1,000 requests per second. Horizontal scaling should double system capacity, verified by load tests conducted before and after scaling. Peak load conditions should keep CPU and memory usage below 80

Architectural Strategy: Spread Load, Reduce Load

Reliability

Description: 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.

Quantification: To quantify the reliability of the Occupi system, several key metrics and measures are necessary. The system's uptime should be maintained at 99

Architectural Strategy: Logging, Testing framework, Passive Redundancy

Security

Description: 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.

Quantification: To quantify the security of the Occupi system, several key metrics and methods are essential. All data transmitted over the network must be encrypted using TLS 1.2 or higher, and 100

Architectural Strategy: Limit Exposure, Limit access, Event Logging

Usability

Description: 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.

Quantification: To quantify the usability of the Occupi system, several key metrics are essential. Conduct regular audits to ensure 100

Architectural Strategy: Support User Initiative (Cancel, Undo)

Compliance

Description: 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.

Quantifiers: To quantify the compliance and data protection of the Occupi system, conduct compliance audits at least twice a year, aiming for a 100

Architectural Strategy: Security

Results of Penetration Testing

Results to Penetration Testing

Architectural Design & Patterns

MVC (Model-View-Controller)

We chose the MVC pattern 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:** Consists of user interfaces, such as the mobile application for employees and the management terminal for managers, handling user input and providing a consistent experience.
- **Application Layer:** Encapsulates business logic related to occupancy tracking, prediction algorithms, and other core functionalities, ensuring separation of concerns.

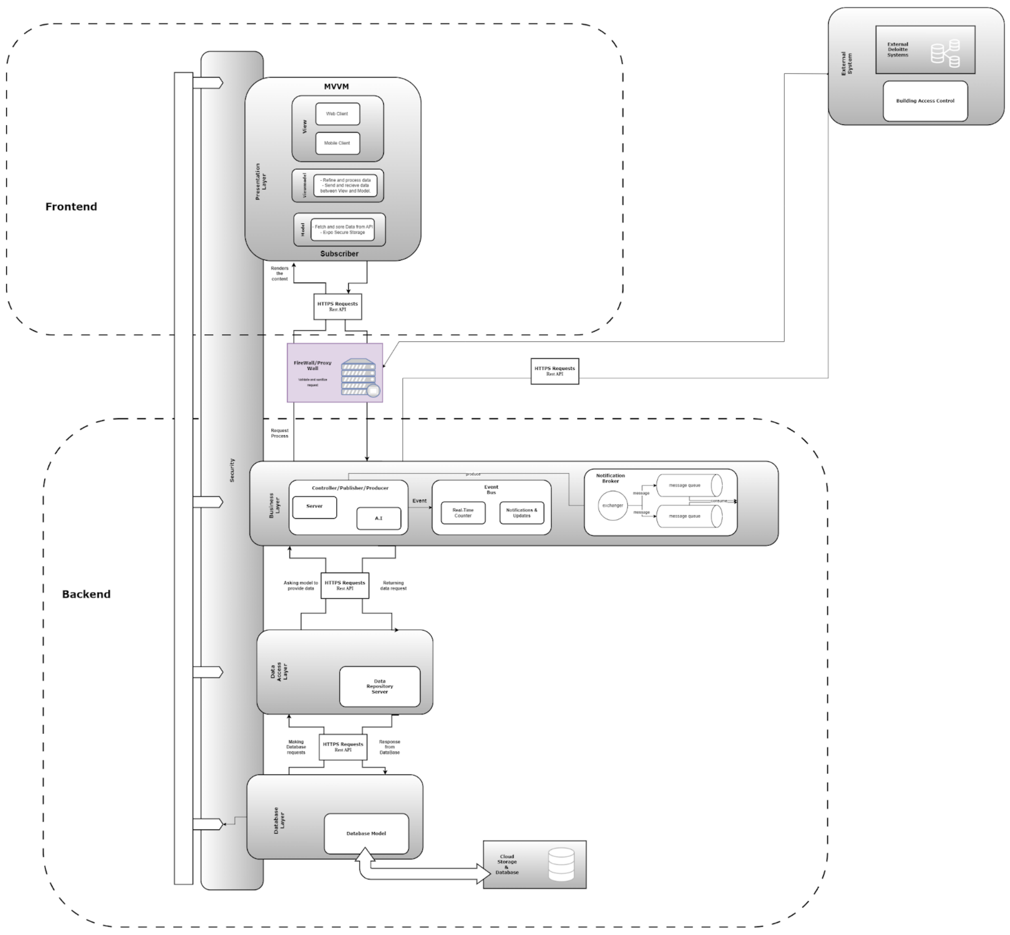
- **Service Layer:** Includes various services or components responsible for specific tasks like user management, occupancy tracking, and real-time updates, implemented as microservices for scalability.
- **Data Access Layer:** Abstracts data storage mechanisms like MongoDB, providing a consistent interface for interacting with data sources.
- **Security Layer:** Handles authentication, authorization, and data protection, implemented as a cross-cutting concern or separate layer based on complexity.

Client-Server Architecture

Client: The client can forward requests to the backend via multiple endpoints and environments (production or development). The domain registrar handles DNS for clients and forwards requests to the server's IP address.

Server: We use a VM instance acting as a server that listens for incoming requests on port 443 (HTTPS) and serves clients with different content, including documentation, landing page, web app, or API backend. A reverse proxy sits between port 443 and our containers.

Architectural Design Diagram



Technology Choices

For Web-based Development

React: A JavaScript library for building user interfaces. It is component-based, which promotes reusability and makes code easier to manage, although 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 may seem verbose for 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 speed up development and ensure consistency, though they might limit customization.

- **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 code sharing between platforms, reducing development time, but may not be as performant as native code for complex apps.

- **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 mobile app development while ensuring a consistent user experience.

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

For Backend Development

Go: A statically typed, compiled language that is efficient for building web services and microservices. Its simplicity may limit it for complex applications.

- **Pros:**
  - Statically typed language 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: Enables rapid development and deployment for web services, microservices, and cloud-native apps. However, MongoDB might not be ideal for applications requiring complex transactions.

- **Pros:**
  - Flexible and scalable NoSQL database.
  - Well-suited for distributed systems and cloud-native applications.

REST API, JSON Web Tokens (JWTs): REST is a standard for designing networked applications, while JWTs offer a secure way of transmitting information between parties.

- **Pros:**

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

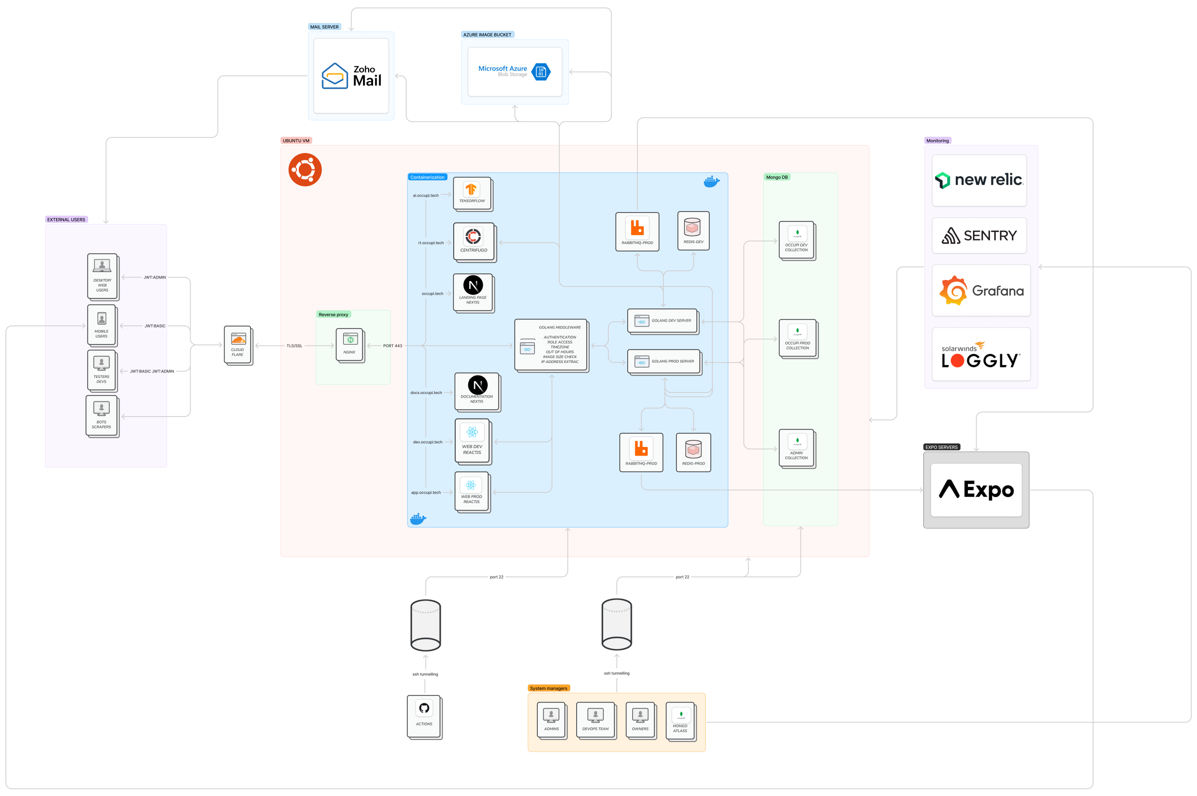
Reasons for Frontend

We chose React for its component-based architecture, which promotes modular and reusable code, alongside Tailwind CSS for rapid styling and customizability. Component libraries such as Schadcn, DaisyUI, and NextUI help accelerate development and maintain a consistent design. This combination ensures a robust and efficient development environment.

Reasons for Backend

Go was selected for its high concurrency support, simplicity, and efficiency, making it ideal for building scalable web services. MongoDB provides a flexible and scalable NoSQL database, suitable for cloud-native apps. REST APIs and JWTs were chosen for their simplicity and security, ensuring scalability and secure data transmission.

Deployment Diagram


MAIL SERVER
Zoho Mail
AZURE IMAGE BUCKET
Microsoft Azure Blob Storage
EXTERNAL USERS
DESKTOP/ WEB USERS
MOBILE USERS
TESTERS DEVS
BOTS SCRAPERS
JWT:ADMIN
JWT:BASIC
JWT:BASIC JWT:ADMIN
CLOUD FLARE
TLS/SSL
UBUNTU VM
Reverse proxy
NGINX
PORT 443
Containerization
ai.occupi.tech
TENSORFLOW
rt.occupi.tech
CENTRIFUGO
occupi.tech
LANDING PAGE NEXTJS
GOLANG MIDDLEWARE
AUTHENTICATION
ROLE ACCESS
TIMEZONE
OUT OF HOURS
IMAGE SIZE CHECK
IP ADDRESS EXTRAC
docs.occupi.tech
DOCUMENTATION NEXTJS
dev.occupi.tech
WEB DEV REACTJS
app.occupi.tech
WEB PROD REACTJS
RABBITMQ-PROD
REDIS-DEV
GOLANG DEV SERVER
GOLANG PROD SERVER
RABBITMQ-PROD
REDIS-PROD
Mongo DB
OCCUPI DEV COLLECTION
OCCUPI PROD COLLECTION
ADMIN COLLECTION
Monitoring
new relic
SENTRY
Grafana
solarwinds LOGGLY
EXPO SERVERS
Expo
port 22
port 22
ssh tunnelling
ssh tunnelling
ACTIONS
System managers
ADMINS
DEVOPS TEAM
OWNERS
MONGO ATLASS