Toronto Metropolitan University

Department of Electrical, Computer and Biomedical Engineering
Faculty of Engineering and Architectural Science

An Embedded System Design to Manage a Smart Office Environment

by Justin Zandstra, Stanley Tan, Carlo Dumlao

Computer/Electrical Engineering Capstone Design Project

Toronto Metropolitan University, Winter 2025

Acknowledgements

We would like to acknowledge our Faculty Lab Coordinator, Gul Khan, for his guidance and support throughout the 2024-2025 school year in completing this capstone project.

Certification of Authorship

This is to certify that Justin Zandstra, Stanley Tan, and Carlo Dumlao are the sole authors of this report. Any assistance that our group received during the development of the project is fully acknowledged and disclosed in this document. The group has cited all sources from which we obtained data, ideas, or words that are copied directly in the document, and sources are properly credited according to the accepted standards for professional publications.

Student 1 Signature	Student 2 Signature	Student 3 Signature
Justin Zandstra	Stanley Tan	Carlo Dumlao

Table of Contents

Abstract

Efficient office management is crucial in today's workplaces, where productivity and connectivity play a critical role. Current workplaces require integrated solutions to streamline operations, improve communication, and optimize resource utilization. To meet the required standards in modern workplaces, OfficeEase was developed as a Smart Home Office Environment that combines an embedded central hub system with a web and mobile Android application. Collectively, these components form a simple, all-in-one innovative platform tool for effective project management that enhances a workplace's team productivity, efficiency, and collaboration.

The system was built around key design principles addressing the modern office management challenges. The primary features of OfficeEase include the following: Project management, administrator view, calendar event management, and reservation scheduling. The key principles in which OfficeEase is designed to fulfill are to provide a platform that facilitates communication, automates tasks, is easily accessible, and enhances management productivity. As a result, the application is designed for users to take advantage of OfficeEases' functionality to reduce workload and improve task efficiency.

The OfficeEase application's primary feature is the project management component, which performs various automated tasks that are designed to minimize workload and maximize efficiency of project management. In response to potential issues with modern management underperforming due to poor productivity and disorganization, the OfficeEase management component provides a better workflow by utilizing various elements. These elements include task automation, progress tracking and assignment, task management, and resource allocation. The design results in a significant boost in productivity with a more organized, transparent, and efficient system that ensures managers can focus on prioritizing key tasks, designating strong team members, and managing projects more easily.

The proposed Smart Home Office Environment design effectively integrates automated capabilities and user-friendly features to enhance project productivity and collaboration. By utilizing a web and mobile application, the design ensures accessibility and flexibility for many users. Thus, the OfficeEase ecosystem promotes productivity, collaboration, and effective resource management, setting a new standard for smart home office environment systems.

Introduction & Background

In recent years, the rise of hybrid and remote work environments has led to a growing demand for smarter, more efficient office solutions that prioritize connectivity, productivity, and ease of use. As businesses continue to integrate technology into their daily operations, the need for systems that can streamline project management, improve team communication, and simplify resource coordination has become increasingly evident [1]. Traditional methods of managing employees, tasks, and workspaces often fall short in terms of flexibility, automation, and user experience [2]. This capstone design project, an embedded system designed to manage a smart office environment, titled OfficeEase, offers the solution.

This capstone design project, OfficeEase, introduces a smart office management system centred on enhancing project coordination and task oversight. The system was developed with a primary focus on project management features, enabling managers to define project goals, create and assign tasks based on employee skills and workload, and monitor ongoing progress through visual feedback mechanisms. Each project created in OfficeEase includes a title, description, and deadline, and is broken down into specific tasks assigned to one or more employees. The platform tracks both task and deadline progress in real time, giving managers clear insight into which projects are ahead, on track, or falling behind. Our Smart Office Environment system also comes equipped with features for creating calendar bookings, conference room reservations, company announcements and oversight tools for managers to track their employees.

OfficeEase integrates three main components that work together to create an integrated smart office environment. The system includes an embedded server hosted on a Raspberry Pi 5, a responsive web application, and a mobile Android application. Together, these platforms allow users to interact with the system across devices in both centralized office and remote settings. The web and mobile interfaces are role-specific, showing different views and capabilities depending on whether the user is an employee or a manager. Employees can view their assigned tasks, mark them as complete, and track project deadlines. Managers, on the other hand, have access to tools for project creation, auto-assignment of tasks, and in-depth performance analytics for individual team members.

The embedded system design for a Smart Office Environment combines project management, communication and organizational features into a single centralized system. OfficeEase delivers a practical solution for efficient workplace management. The system is designed to be scalable, secure, and easy to use, making it a valuable tool for improving productivity, communication and organization within the modern office workplace.

Objectives

The primary objective of this engineering design project was to investigate, design, and develop a complete infrastructure for managing a smart office environment using embedded

systems and Android mobile applications. The goal was to create a centralized system that could improve communication, organizational efficiency and productivity in the workplace with an emphasis on smart project management features.

A key focus of this system was the development of powerful project management features tailored for modern workplaces [3]. Managers can create projects, define tasks, assign them based on employee skills and availability, and track progress through visual indicators like task and deadline progress bars. By simplifying task assignment and making project status instantly visible, the system promotes better team coordination and accountability.

The project emphasized accessibility from both desktop and mobile environments. Our web application, hosted on an embedded Raspberry Pi server, works alongside a dedicated Android application to ensure that users can interact with the system from anywhere. Whether they are at their desks or working remotely. This cross-platform integration enables real-time collaboration and status updates, a crucial component of effective remote project management.

Acknowledging the need for flexibility for different users in the office, OfficeEase was designed with role-specific views and features. Employees are given a streamlined dashboard to view their assigned tasks, reservations, and announcements. Meanwhile, managers are provided with administrative controls, employee analytics, and performance metrics. This provides company managers with the tools to privately track their employees' performance and oversee multiple projects at once. The system tracks key metrics such as task completion rate and average completion time, as well as employee workload.

In addition to task and project management, the system supports practical features for organizing workplace resources. Managers can schedule meetings, book conference rooms, and publish announcements, while employees can reserve hotdesks or check their calendars. These features improve company organization and reduce scheduling conflicts.

Theory and Design

The development of the Smart Home Office Environment, OfficeEase, requires addressing critical design considerations during the initial planning stage to ensure a successful and functioning smart environment. These considerations include determining suitable hardware components, software architectures, database choices, the main features of the application itself, alternative design choices, and a list of expenses.. Throughout this chapter, these topics will be covered in detail, highlighting the primary characteristics and features of our design.

The Theory and Design consists of four primary components: the Embedded System, the MongoDB database, the Web Application, and the Mobile Android Application. The complete overview of the system is presented in Figure 1. Each of these components provides an essential part in developing our smart home office environment. The Embedded System acts as the central hub of our design, performing the vital functions that make it a smart home environment. The MongoDB database contains all of the information used by OfficeEase. The Web Application provides a user-friendly interface, which allows a convenient method for viewing and interacting

with the system. Likewise, the Mobile Android Application provides users with a mobile interface that makes it easily accessible to view OfficeEase on their mobile devices. Overall, these components form the Smart Home Office Environment, OfficeEase, and are designed to enhance productivity through functional accessibility.

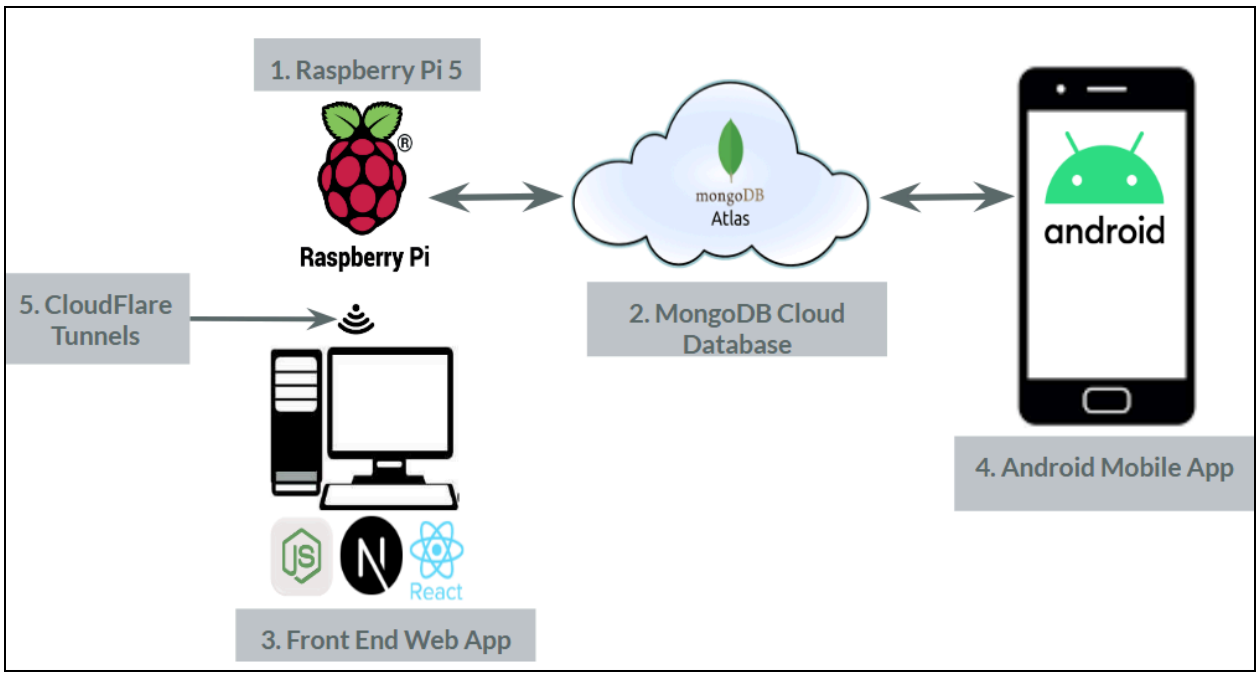

Figure 1. Embedded Central Hub System Overview.

Microprocessor

The embedded system functions as the central processor and server of the web application. It mainly facilitates data flow between the website and the database and handles all backend program operations of the web application. In our design, the embedded central hub must be equipped with a high-performance processor capable of hosting the server. This processor should efficiently handle database management, server-side computations, background tasks, and concurrent user requests.

The criteria for selecting a suitable microprocessor to host the server must fit several specification requirements. For computing, the central processing unit of the microprocessor is required to run at a speed of 2.4 GHz for optimal performance [4]. Processing speeds lower than this standard may be detrimental to the system as operations are executed at too slow a rate for a smart environment to operate smoothly. A simple and small application such as OfficeEase will require a minimum of at least 8 GB of RAM [5]. Similarly, a low amount of memory on the microprocessor will delay and cause potential errors to occur. To ensure a working application, we have selected a microprocessor that contains these standards.

With this in consideration, the chosen microprocessor was the Raspberry Pi 5. This microprocessor contains Broadcom BCM2712, which is a 2.4 GHz quad-core processor with a 64-bit Arm Cortex-A76 CPU [6]. This is a powerful tool that helps with server hosting as it

contains a fast 2.4 GHz processing speed and 4 cores, which allows simultaneous tasks to be handled. Additionally, the Raspberry Pi 5 contains LPDDR4X-4267 SDRAM 4 GB. The specifications of this show a low-power double data rate RAM with a 4267 MT/s data transfer rate. It also contains 4 GB of memory storage, which is a sufficient amount for a server host. Other consideration components include the following: Dual-band 802.11ac WiFi & Bluetooth 5.0 and 5V/5A DC Power via USB-C. These components provide the microprocessor with the ability to access the network and Bluetooth. The power supply is important to monitor as it determines if the board can maintain itself with no issues. These specifications make the Raspberry Pi 5 microprocessor a strong choice, as it is capable of hosting a server with ease [7]. With high processing speed, energy efficiency, cost-effectiveness, and a sufficient amount of memory, it can support the system and satisfy performance requirements.

Remote Access and Secure Web Deployment

To ensure secure, reliable, and low-maintenance remote access to the embedded web application hosted on the Raspberry Pi, a Cloudflare Tunnel was employed as the primary reverse proxy solution. This approach eliminates the need for port forwarding or exposing the local device to the public internet, thereby reducing the system's vulnerability to unauthorized access and simplifying network configuration [8]. Cloudflare Tunnels (also known as Argo Tunnels) securely expose a local web server to the internet by creating an outbound-only connection from the device to Cloudflare’s edge network. When a user sends an HTTP or HTTPS request to the application through a browser, the request is first handled by Cloudflare’s global network, which then forwards it through the Cloudflare tunnel to the local server on the Raspberry Pi.

To provide an easily accessible endpoint for the application, the domain officeease.org was purchased and configured through the Cloudflare dashboard. A CNAME DNS record was created to point the domain to the tunnel’s public endpoint, enabling seamless access to the system via https://officeease.org. All incoming web traffic is routed through Cloudflare’s global content delivery network (CDN), which offers performance optimization, end-to-end TLS encryption, and resilient failover mechanisms, ensuring that users experience a fast, secure, and reliable connection to the smart office system [9].

Database

MongoDB Atlas was used as the cloud-hosted database solution for storing and managing all application data, including user accounts, tasks, projects, and announcements. As a fully managed NoSQL database service, MongoDB Atlas provides a flexible and scalable data model. This fits well with the structure of our smart office environment system, as each entity can vary in complexity and size. Collections such as users, tasks, and projects were designed using document-based schemas, providing an organized data structure and allowing for efficient retrieval and modification of data.

By hosting the database on Atlas, the application benefits from automated backups, encrypted connections and scalability [10]. The database is integrated directly with the web application using Mongoose, a library for MongoDB and Node.js, allowing for smooth communication between the frontend and the backend. Data is queried and modified using CRUD operations in real time through API calls to the Atlas-hosted database [11]. By utilizing MongoDB Atlas, our Embedded System, designed to manage a Smart Office Environment, has the flexibility to scale to any business size and complexity.

Website Application

The web application was developed using a full-stack JavaScript setup made up of Node.js, React.js, and Next.js, each playing a specific role in the system. Node.js runs the backend server and handles all core operations like API requests, user authentication, session tracking, and interactions with the database. On the frontend, React.js was used to build a clean, responsive user interface using components, which helped keep the code organized and easy to manage. Tailwind CSS was used for styling the user interface, allowing for a consistent and modern look across all views while making it easy to implement responsive design principles. Next.js was used as the main framework. It's built on top of React and gives us the ability to manage both the frontend and backend in a single environment, with built-in support for features like server-side rendering and routing. The frontend dynamically renders different dashboards and accessible routes based on the user's role, ensuring that employees and managers only interact with features associated with their permissions. This combination of technologies made development more efficient and helped deliver a fast, reliable, and user-friendly experience [12].

User authentication in the Smart Office Environment system is handled using a combination of cookies and JWT (JSON Web Tokens) to maintain secure access control across both the website and mobile application. When a user logs in, their credentials are verified, and a signed JWT token containing key user information is generated. This token is stored in a cookie on the client side, allowing it to be automatically included in API requests. On the server, authentication middleware is used to intercept incoming requests, extract the token from the cookie, and verify its validity before allowing access to protected routes. If the token is valid, the request is processed, and the user is granted access to features like their dashboard or device settings. Tokens are set to expire after a set duration to enhance security, requiring users to log in again when the session ends. This setup ensures a secure and efficient way to manage user sessions and roles across the platform [13].

Mobile Android Application

The mobile application is designed to run on an Android Nougat 7.0 environment, ensuring that at least 97% of Android devices in the market are going to be supported. Specifically, it is compatible with Google's Nexus and Pixel devices, Sony Xperia, and a portion of Samsung devices such as Galaxy S7, S6, A3, and A8. Conversely, Android devices developed before 2013, or devices with older hardware such as those using the Adreno 330 graphics core, may not be compatible with the mobile [14].

As shown in Figure 2, Java language and the Gradle build system offered by Android Studio lay the foundation of the application. The Gradle build offers a cost-efficient way of constructing the app as it allows the creation of multiple APKs for different device configurations, without the need to modify or tinker with the app's source code [15]. To facilitate connection to the server, the app utilizes the MongoDB Realm Sync module, which provides a local database (or realm) within the device that can be synced independently to a remote MongoDB database on Atlas. In other words, the module allows the app to work offline first, storing data locally and then synchronizing changes when a network connection is available.

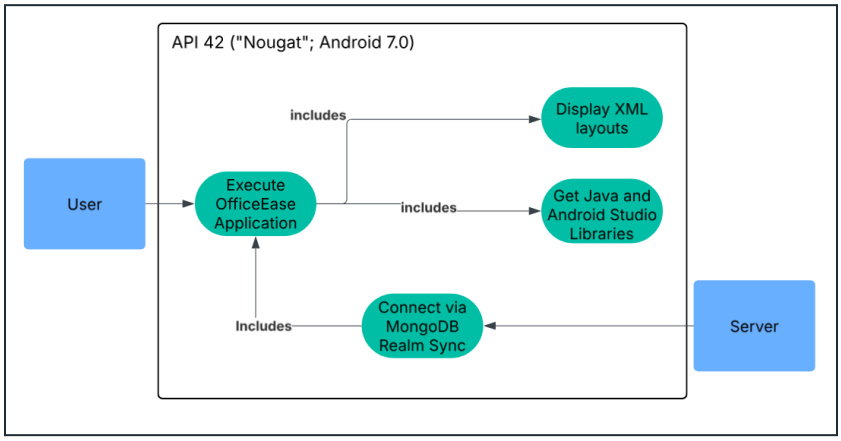

Figure 2. Mobile Android Application Overview

Roles

In web and Android applications, there exist two actors: a manager and an employee. Each actor is provided with certain capabilities that align with their organizational roles and responsibilities. These capabilities are briefly discussed below.

Manager

The manager can fully interact with the system's core functionalities and has a variety of permissions that are necessary for managing a workplace:

- Create Projects and Tasks: able to add, modify, and delete projects and tasks; has the exclusive right to set deadlines.
- Assign Tasks to Employees: Distribute tasks to employees in accordance with their skills and current workload.
- View Analytics of a Project: tracks the progression of the project, work distribution, and the completion of tasks.
- View Status of Employees: able to view the availability of all the employees; the work efficiency of each employee is visible
- Add Events: create events in the dynamic calendar which can be shared among employees
- Add Announcements: create announcements to notify all employees within the workplace
- Reserve a Meeting Room: has permission to book a whole room for meeting purposes

Employee

Employee is restricted to only interacting with the received information and messages shared by the manager:

- View Assigned Tasks: can review and complete the tasks assigned to them; the days remaining to complete the project are visible
- View Invited Events: able to view events shared by a manager via a dynamic calendar.
- View Invited Reservations: able to view his/her reservations, such as invited meetings
- View Announcements: able to view the announcements
- Reserve a Hotdesk: can reserve a hotdesk, or a seat in a working station.

Design Features

Project Management Features

The project management feature serves as the foundation of the smart office environment system. It allows management to create projects, define tasks, assign them to employees, and monitor progress through a centralized, user-friendly interface. These tools are designed to improve task coordination, enhance efficiency, and boost overall productivity in the workplace.

Each project includes a title, description, and deadline, while tasks are linked to both projects and employees, enabling grouped project views and personalized dashboards. Each task contains metadata such as status, deadline, assigned users, and required skills. The system also tracks employee workload and supports both manual and automatic task assignment based on skill matching and current task load. Visual indicators display task and project completion status, helping users quickly assess ongoing progress and upcoming deadlines. These features ensure that both employees and management stay aligned on project timelines and task responsibilities.

The ability to view all projects company-wide, as well as create and delete projects, is limited to those within management. In Figure 3, a manager can create a new project by inputting a title, project description, and a deadline for the project inside the create project form after clicking the create project button.

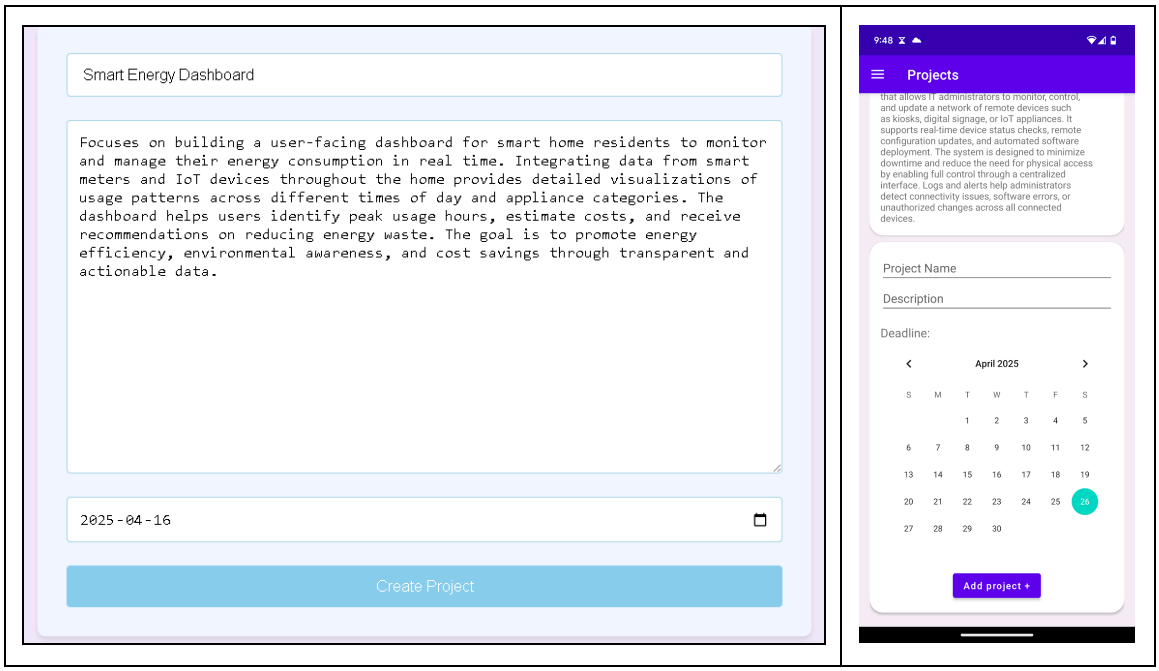

Figure 3. Web and Android Application View for Creating a Project.

The project's view section, shown in Figure 4, displays all of the current ongoing projects within the company in a centralized location to accommodate for multi-project overview and maximize multitasking capabilities. Notice that on the website, there exist progress bars which display the live task and deadline progress of a project. The progress of the project's deadline and task are determined by using Eq. 1 and 2, respectively. Put simply, the deadline bar represents the time elapsed between the project's creation date and the current date, relative to the total time between the creation date and the project deadline. Meanwhile, the task bar shows the total percentage of completed tasks within that specific project. The task bar will be dynamically updated in real time as the manager's employees tick off their completed tasks.

$$Task\ Progress = \frac{Number\ of\ Completed\ Tasks}{Total\ Number\ of\ Tasks} \times 100\% \quad (1)$$

$$Deadline\ Progress = \frac{Current\ Date - Project\ Creation\ Date}{Deadline\ Date - Project\ Creation\ Date} \times 100\% \quad (2)$$

The Android application, on the other hand, has a different approach to visualizing the progress. Rather than a bar, each project is associated with a colored *status* that shows the actual remaining days left to complete the project, as well as the number of tasks that are yet to be completed. Finished projects are indicated by a green status, while overdue and pending projects are labelled as red and grey, respectively. By offering the task and deadline progress, managers can quickly assess a project's overall status and determine whether it is on track, ahead of schedule, or falling behind.

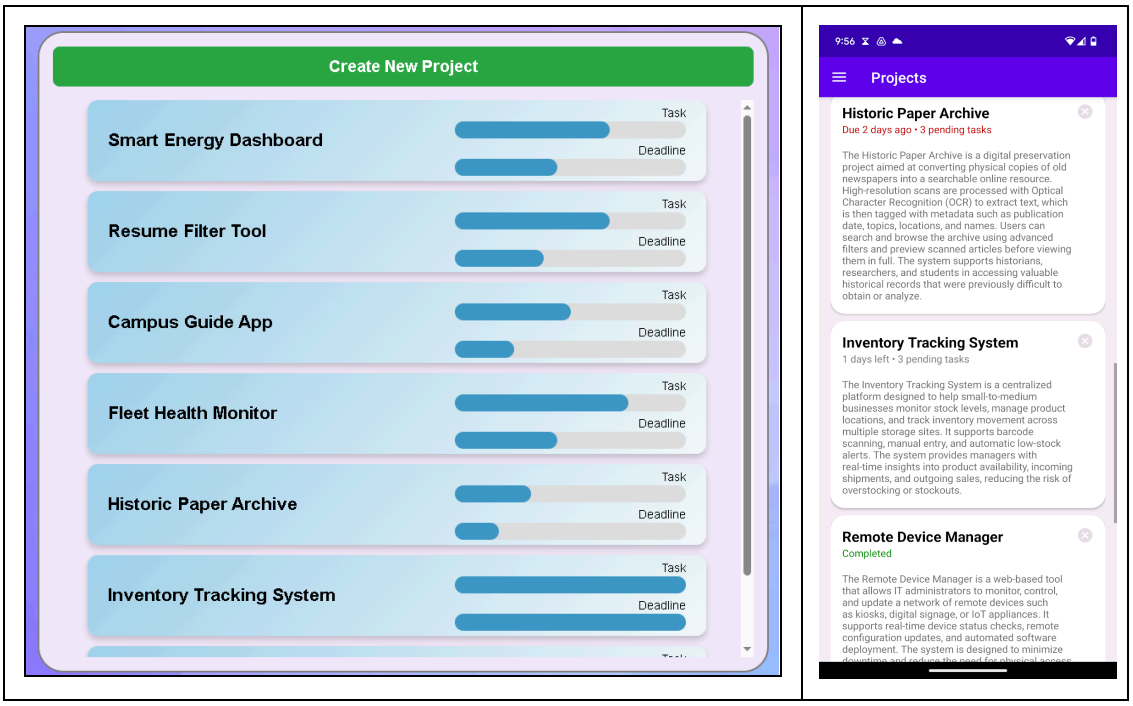

Figure 4. Web and Android Application View for Monitoring Projects.

Within a project, a manager can create new tasks and assign them to a specific number of employees. The creation of a task is presented in Figure 5. Essentially, the manager has two options when it comes to distributing the workload of a project:

1. He/she may manually assign the tasks to employees by marking the checkbox.
2. Alternatively, the auto-assign employees button may be triggered, in which the app will automatically choose the members who have the lowest *workload* and those who are qualified to work on the newly created task based on the predetermined skill set made by the manager.

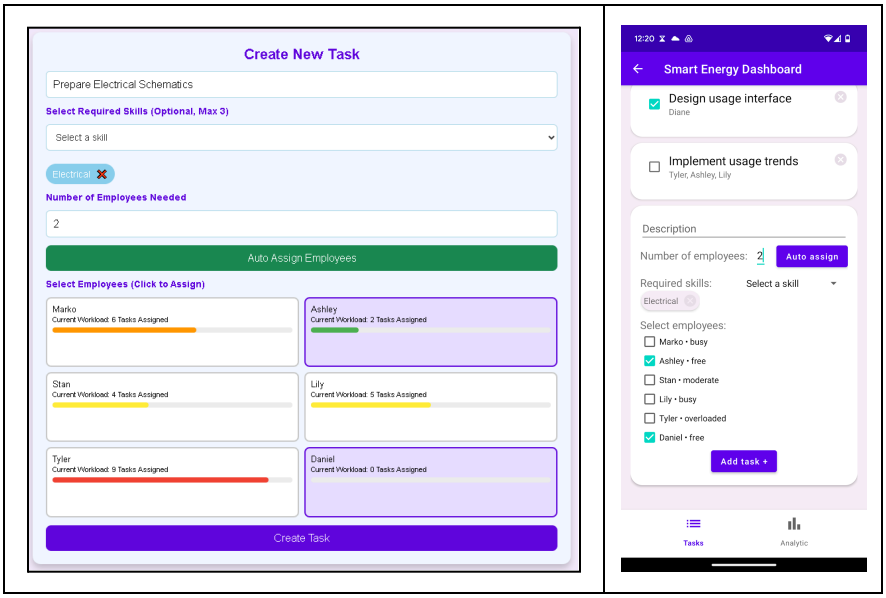

Figure 5. Web and Android Application View for Creating a Task.

The skill(s) of an employee may be of the following: Technical Writer, Electrical, Civil, Software, and Mechanical. Moreover, the *workload* of an employee is associated with the number of pending tasks assigned to him/her – this includes tasks from other projects owned by a different manager. The intensity of this workload is indicated by a colour and a workload status, which is shown in Table 1.

Table 1. Employee's Workload Status by Colour.

Color	Workload Status	Range Of Assigned Tasks
Green	Free	0-2
Yellow	Moderate	3-5
Orange	Busy	6-8
Red	Overloaded	9+

Figures 6 and 7 showcase how the project's tasks are arranged and monitored, from both the website and the Android application. For each task, the manager can view the name of the assigned employees, as well as their profession or skills necessary to complete it. Likewise, the analytics of the tasks (or project) are presented, such as the task and deadline progression, and the work distribution via a pie chart.

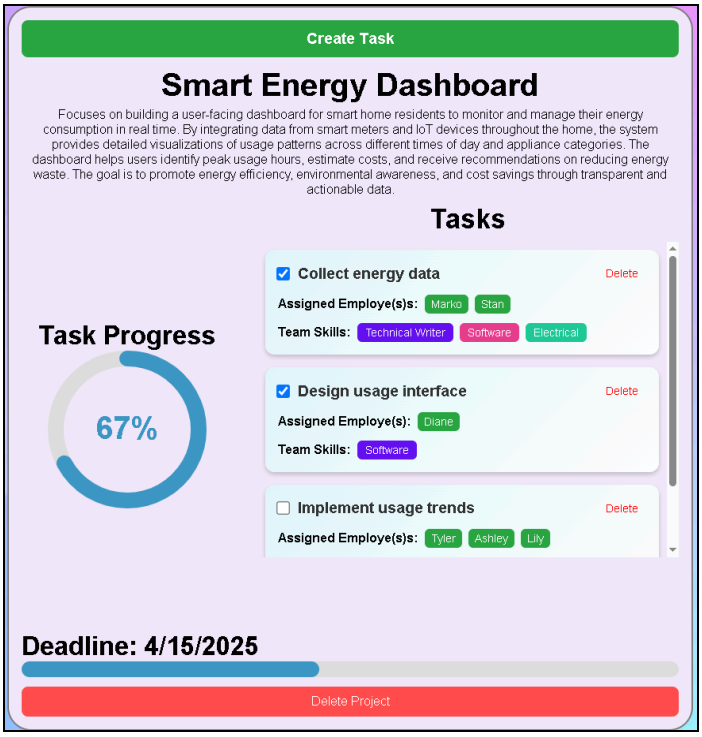

Figure 6. Web Application View for Viewing a Task.

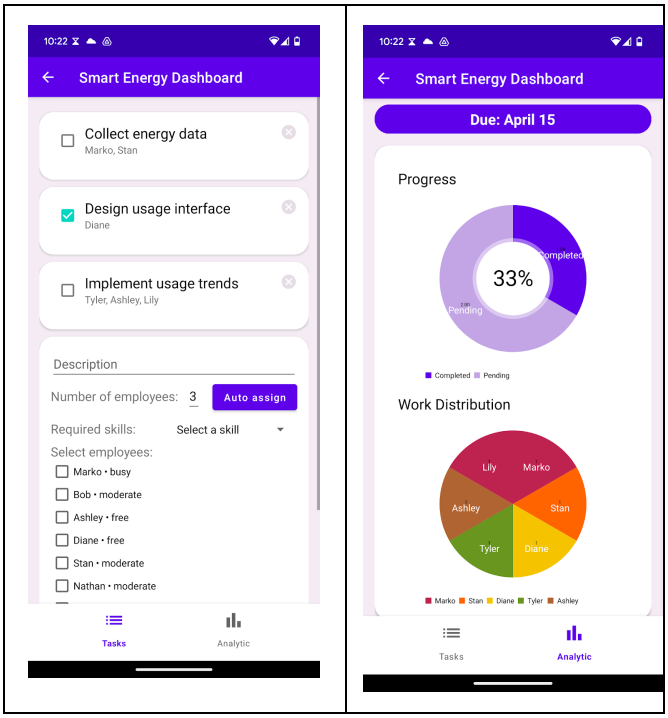

Figure 7. Android Application View for Viewing a Task.

As shown in Figure 8, employees have access only to the projects and tasks that are assigned to them. The user interface is displayed in a minimalistic manner to ensure simplicity and is easy to navigate for employees to quickly locate certain projects and tasks. Within their projects, employees can view the project description, task name, team members, team skill sets, and the deadline associated with the task assignment. When employees complete their tasks, they can click the checkbox next to the task name to mark it as completed. This way, managers can be notified of the task's completion. This implementation is a simple yet effective tool of communication between management and employees, enhancing productivity by minimizing unnecessary information and focusing on more relevant details.

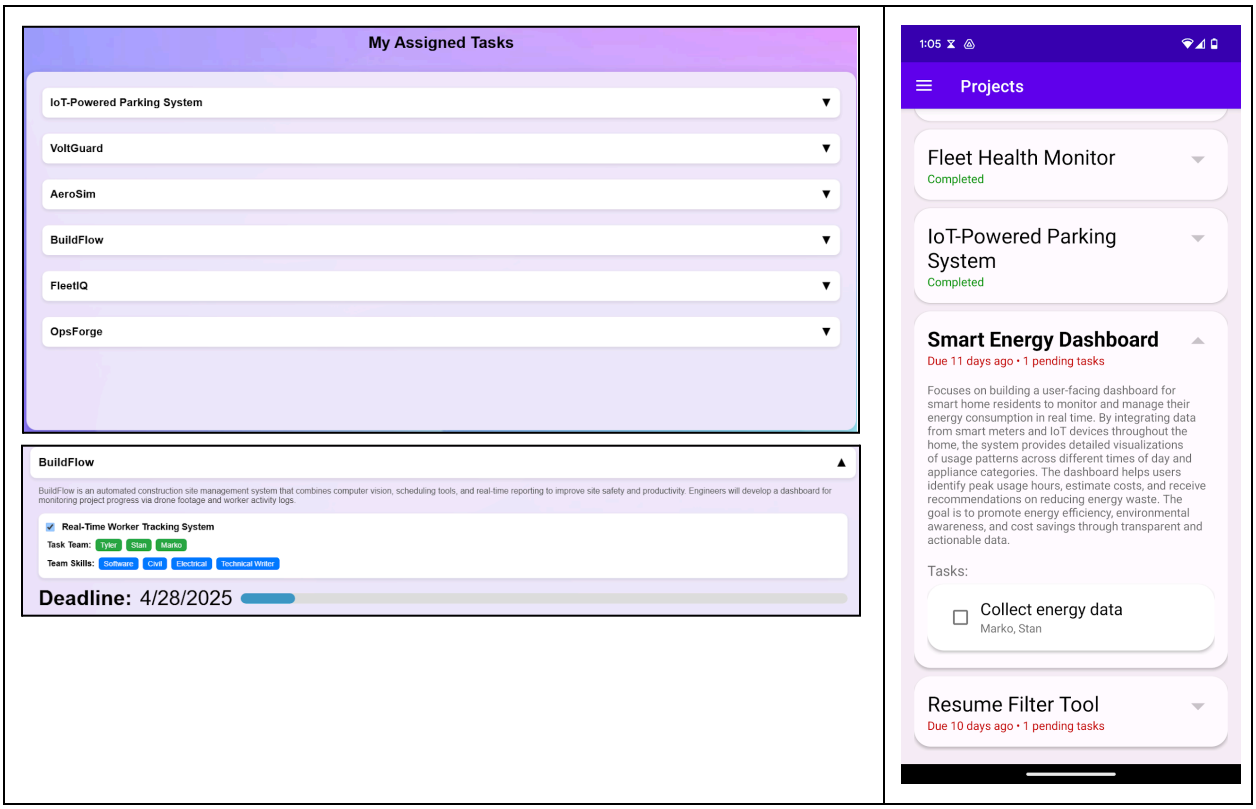

Figure 8. Web and Android Application of Employee View on their Projects and Tasks.

Administrator View

Another strong feature of OfficeEase is our administrator view window, which is a powerful tool to assist managers in evaluating their employees through several metrics. Similarly, this view window is designed to help monitor employees' performance to determine potential weaknesses to improve. With the support of the view window, managers can maintain a more effective workplace to improve their performance and productivity.

The administrator view displayed in Figure 9 allows managers to single out certain employees and monitor their detailed information. The set of statistics will assist them in evaluating the employee's work performance and their current assigned projects & tasks. The statistics are the following: the number of assigned tasks, the number of completed tasks, pending completion, the number of projects assigned, average completion time, completion ratio, and efficiency rank. The pending value shows the number of incomplete tasks that have been assigned to the employee. This information reflects on the workload of the specific individual. The average completion time is a statistic based on the duration the employee takes to accomplish their task on time, which can be mathematically expressed as Eq. 3.

$$(Avg\ Completion\ Time) = \frac{1}{N}\sum_{i=1}^{N}\left[Completed_i - Created_i\right] \qquad (3)$$

Where $Created_i$ and $Completed_i$ are the time the task was created and completed by the employee, respectively, and N is the number of tasks completed. The completion ratio is the proportion of the number of completed tasks to the total number of assigned tasks. Lastly, the efficiency rank is the rank of how effective the particular employee is compared to the other employees. This rank is determined by comparing each employee's average completion time. A manager may view the summary of the entire rankings, shown in Figure 10, by clicking the *efficiency rankings* button.

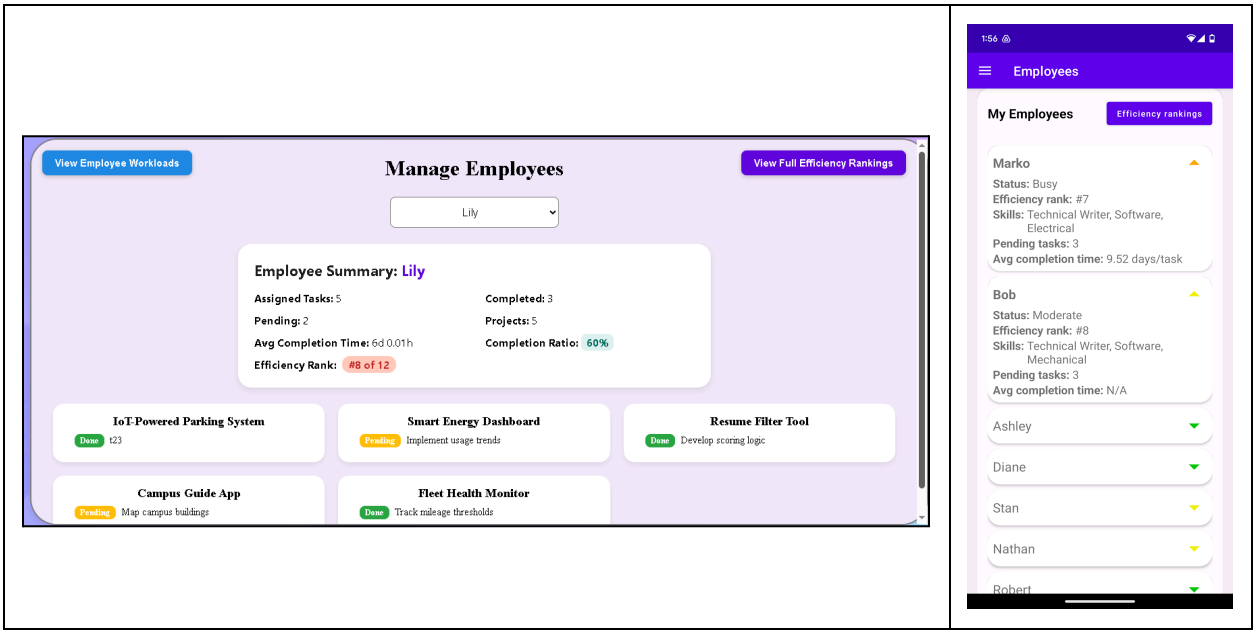

Figure 9. Web and Android Application View for Managing Employees.

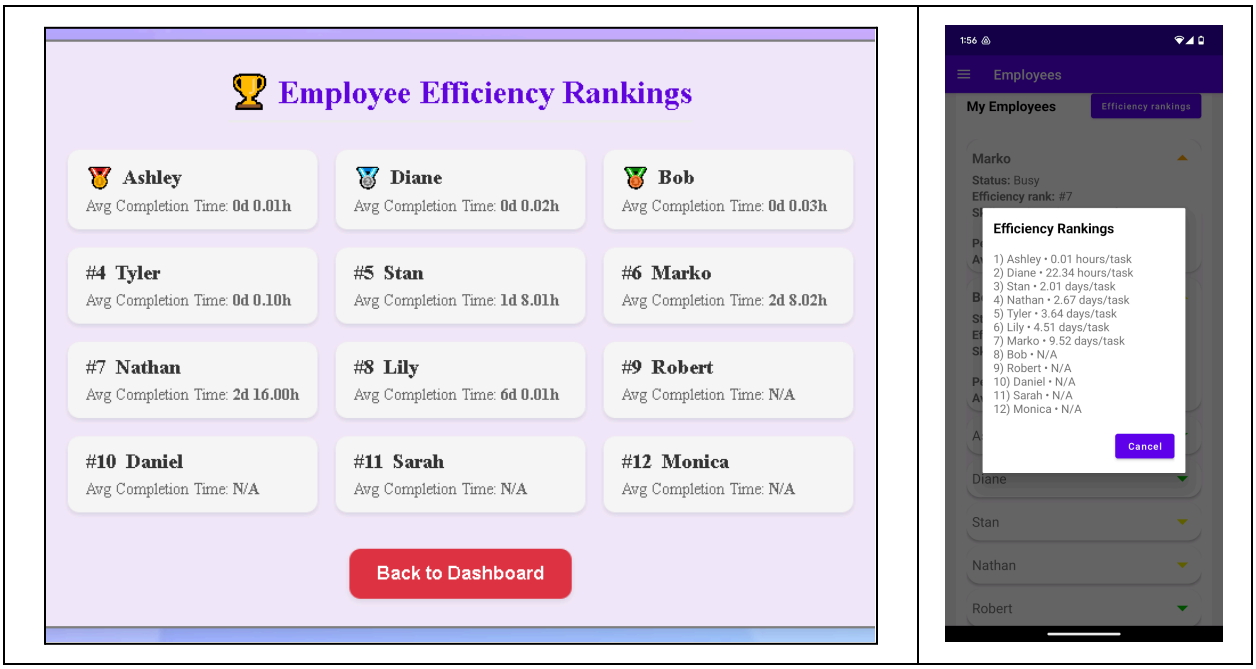

Figure 10. Web and Android Application of Employee Efficiency Rankings.

In addition to the work efficiency and performance, a workload pie chart is displayed to visualize how busy all of the employees are in the workplace, allowing managers to plan their projects and tasks accordingly. This workload overview is presented in Figure 11.

Figure 11. Web and Android Application of Employee Workload Overview.

Calendar

Aside from the project management features, the Web and Android applications also provide a dynamic calendar that aims to inform and organize events in a working environment. To create a new event, the manager can interact with the calendar by selecting the desired date and inputting the name, description, location, and starting time. Furthermore, the manager can share those events with specific employees or participants by manually checking their corresponding checkboxes or by selecting a project in which the app will automatically invite all associated members. The calendar of the invited employees would then be updated accordingly. The primary function of Calendar, creating events, is illustrated in Figures 12 and 13 on both the Web and Android interfaces. OfficeEase enhances the user's experience to view the calendar alongside the event information. This provides users with a simple method to visualize and organize their schedules and project arrangements

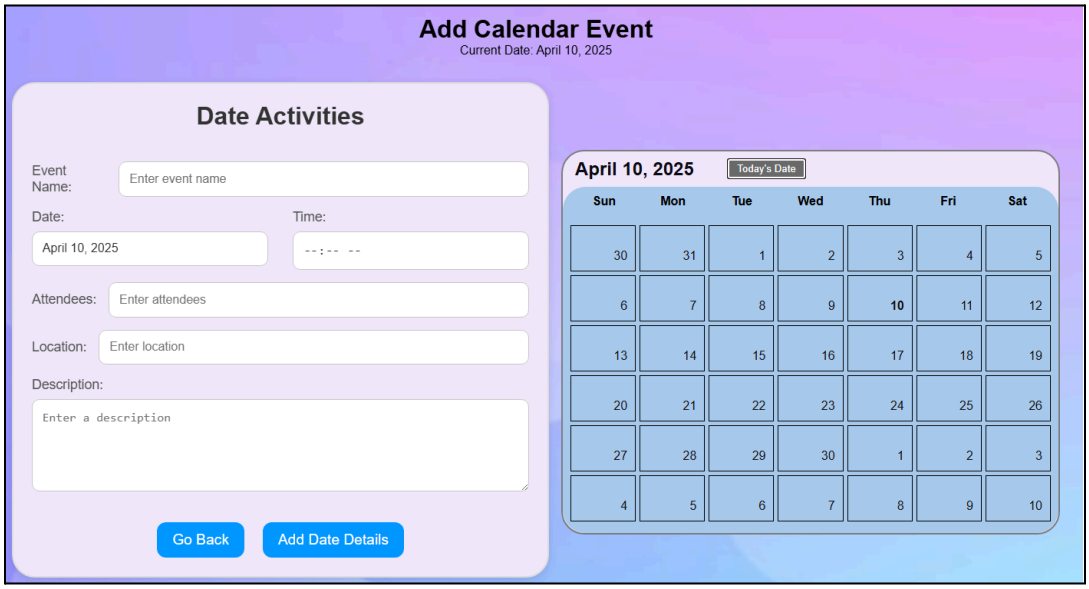

Figure 12. Web Application View for Making a Calendar Event

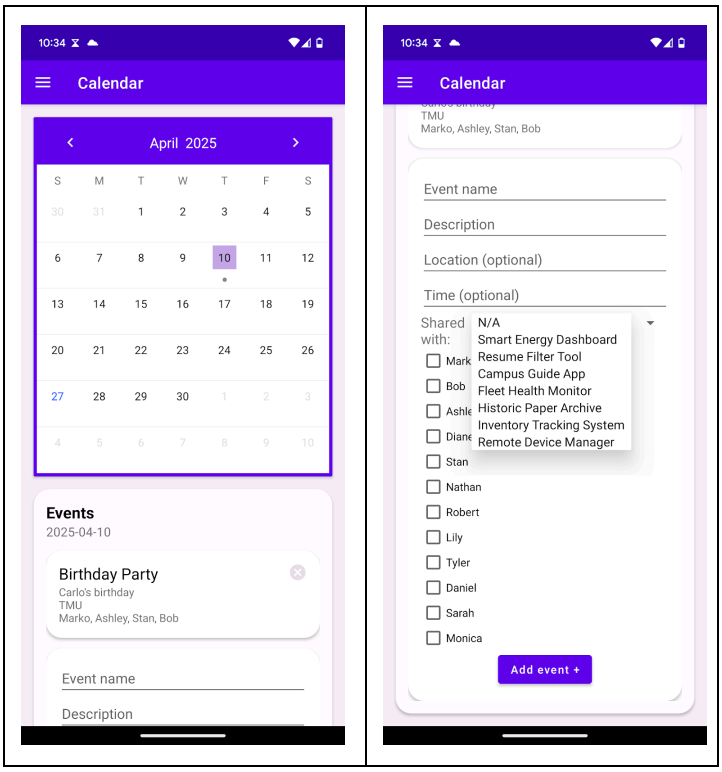

Figure 13. Android Application View for the Dynamic Calendar from a Manager.

Reservations

OfficeEase contains a user-friendly interface which is easy to navigate and utilize, effectively helping employees reserve certain spaces around their work environment. With administrative accounts, managers have access to all sorts of information and have broader access. They can reserve all types of working spaces, from meeting rooms to hot desks. For reserving a hotdesk, the application will automatically book an available seat based on the criteria inputted by a user, including the duration and preferred time of the reservation.

Figure 14 shows the Web interface for creating a reservation. As mentioned above, managers and employees have different levels of access based on their roles. Managers can view and reserve both meeting rooms and hot desks, meanwhile employees are limited to hot desks only. The Web interface shows the reservation room information and selection on one side and the reservation details on the other side. Users can select their dates, times, and rooms to view availability. By specifying a start time and reservation preferences, OfficeEase will be able to assign an ideal reservation based on their projects, employee availability, and reservation preferences.

Figure 15 shows the Android interface for viewing existing reservations and creating a hot desk reservation. The booked reservations are listed in a simplistic manner that is easy to view. Similarly to the web application, when creating the reservation, the employee is required to select a range of preferences. This allows the application to automatically assign the employee the best available option according to their preferences.

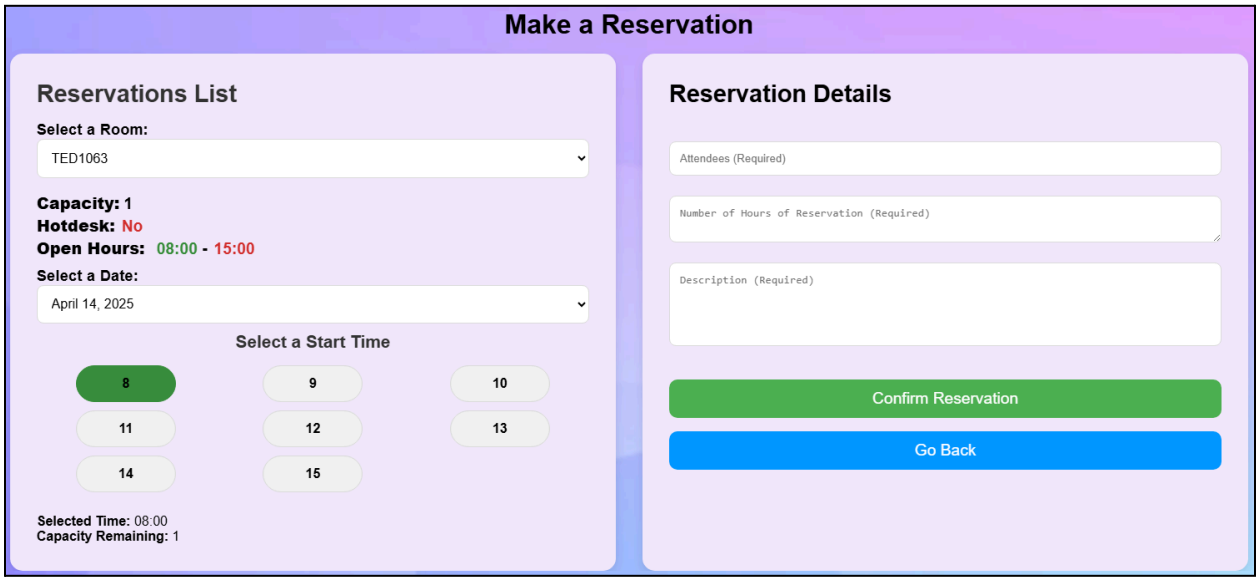

Figure 14. Web Application View for Making Reservations.

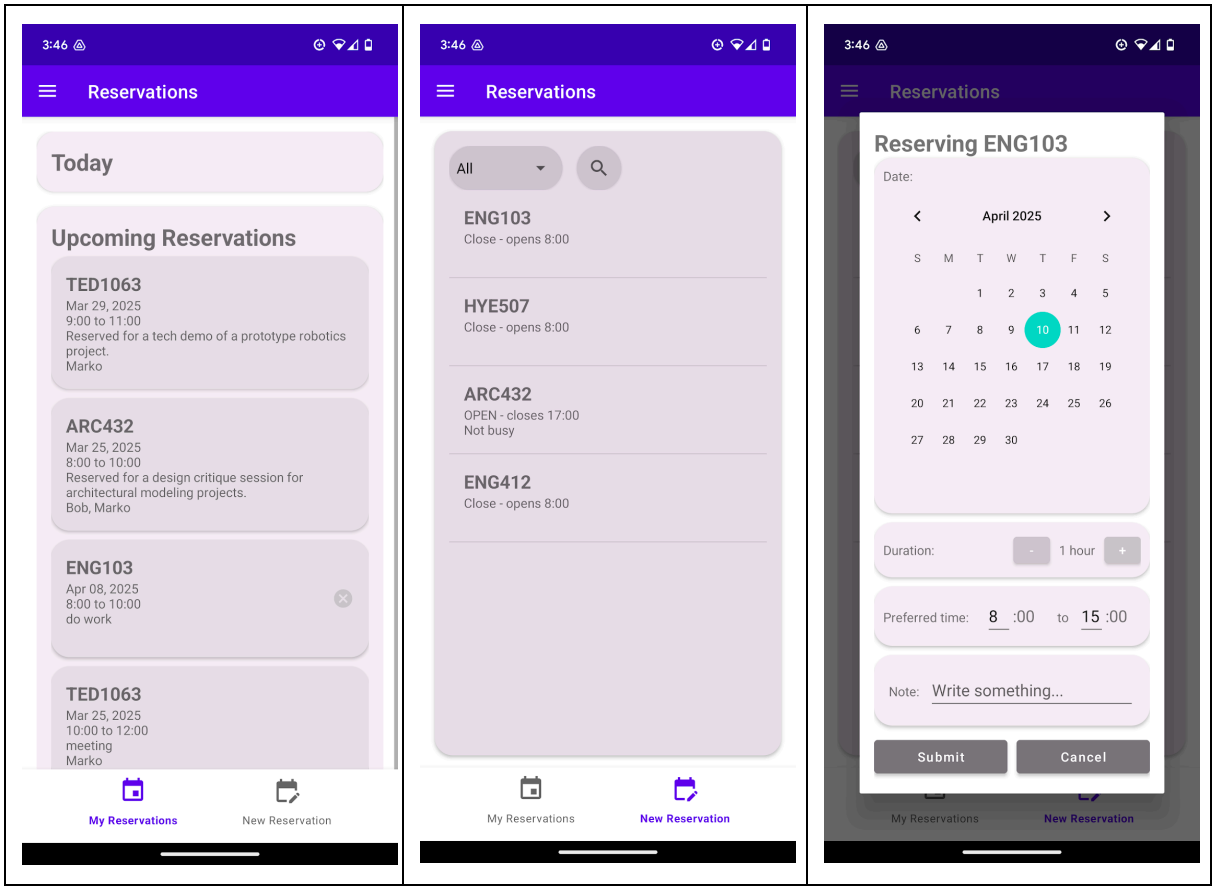

Figure 15. Android Application for Reserving a Hotdesk.

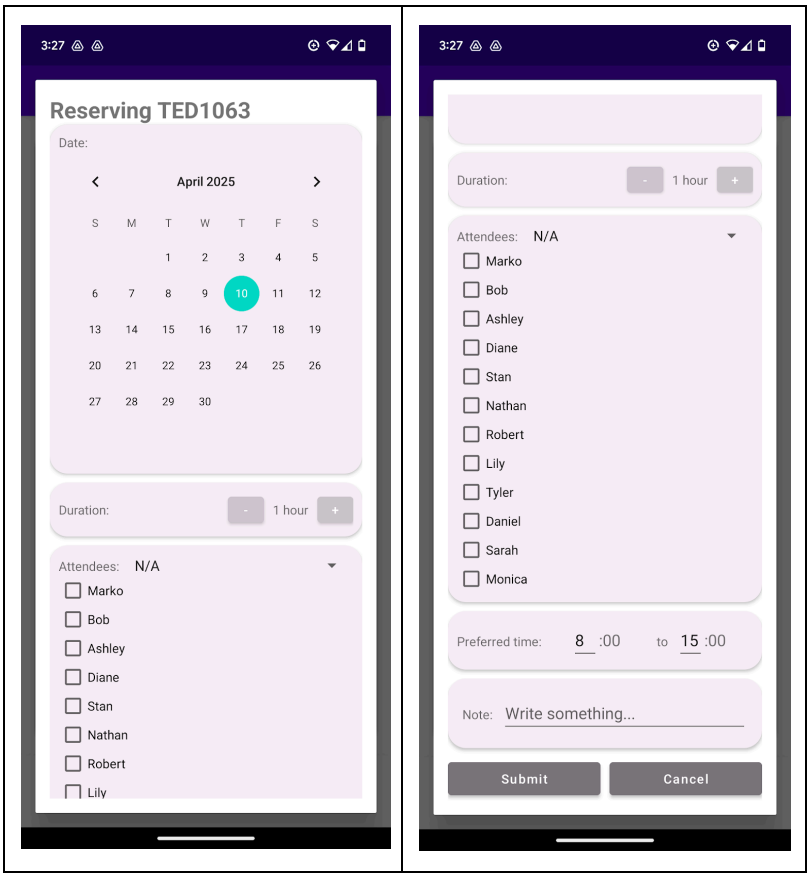

Figure 16. Android Application for Reserving a Meeting Room.

On the other hand, for reserving a meeting room, the manager is required to input the participants, in addition to the desired reservation time. This is shown in Figure 16, which shows the Android interface of reserving a meeting room from a Manager's perspective. The application requires choosing an appropriate time for the meeting, while also ensuring that it does not conflict with the current reservations of the invited employees.

It is important to consider that the opening hours of a room are divided into 1-hour time slots. For instance, if it is open between 8 AM to 11 AM on a particular date, the slots would be 8 AM, 9 AM, 10 AM, and 11 AM. Based on the mentioned three parameters (ie, duration, preferred time, and participants), the reservation time slots for that room are then extracted and forwarded to the actual scheduling algorithm. Three conditions must be met to successfully reserve a meeting room:

1) The room is not at full capacity during the chosen time slot
2) The slot is not in conflict with the current reservation of the manager,
3) The slot is not in conflict with the reservations of the participants.

If any of the above is not met, then the algorithm will go to the next available time slot and iterate again until the conditions are satisfied. When the slots are exhausted, the application will simply notify the user that the reservation is unsuccessful and to choose another date or time.

Alternative Designs

Database Comparisons

For our database, MongoDB Atlas was favoured compared to other databases, such as SQL databases, for several reasons. MongoDB is a NoSQL cloud database. As a NoSQL database, the data is flexible and customizable as it does not require a fixed model scheme, such as other databases like SQL. MongoDB's scalability and flexibility make it simple and suitable for our custom data models, as it can adjust dynamically to most models. Furthermore, for a Smart Home Environment, it is necessary to obtain information at a quick rate, making MongoDB a stronger fit compared to SQL databases, as it allows high-speed inserts and handling large amounts of data efficiently [16]. With a more rapid rate, it reduces the amount of time for data transfer, making information more accessible in real time. Thus, MongoDB Atlas was a much stronger candidate for our database compared to other databases.

The reason we chose to do a cloud database compared to a non-cloud database was to avoid connectivity issues with several devices on the Raspberry Pi 5. To ensure a much smoother operation of the Raspberry Pi 5 with the web application, having a non-cloud database would cause the microprocessor to undergo more stress and data transferring methods. With a cloud database, the microprocessor would not encounter such potential issues. Furthermore, in case of a technical issue where the Raspberry Pi server is down, the cloud database will continue to operate for the Android devices to access. Thus, MongoDB, a cloud database, was a much more practical solution to avoid potential problems that arise with a poorly managed database.

Microprocessor Selection

Similar to the Microprocessor section in Theory and Design, the selection of the microprocessor must fit specific criteria to host and operate the web application with ease. In the early stages of design planning, our initial list of potential hardware components was the following: Arduino, Banana Pi M5, Raspberry Pi 4, and Raspberry Pi 5.

Beginning with Arduinos, as our standard for the central unit was determined, microcontrollers such as Arduinos were deemed to be unfit to be assigned as our central system unit. These are due to the various aspects of the microcontroller, such as low amounts of memory, low processing speeds, and limited networking capabilities [17]. Hence, the microcontroller did not meet the requirements of our project design.

With that in mind, microprocessors were evaluated. By narrowing our scope to meet the project specifications, three potential candidates were considered: Banana Pi M5, Raspberry Pi 4, and Raspberry Pi 5. The Banana Pi M5 and Raspberry Pi 5 contain similar specifications, which made it a difficult decision. However, the Banana Pi M5 has slightly lower capabilities compared to the Raspberry Pi 5. The Raspberry Pi 5 has stronger cores, faster memory, and higher data interface speeds. Lastly, the Raspberry Pi was compared to other Raspberry Pi

models. The specific comparison between Raspberry Pi 4 and Raspberry Pi 5 can be seen in Table 2 below.

Table 2 Comparison of Raspberry Pi 4 vs Raspberry Pi 5 [18].

Comparison: Raspberry Pi 4 vs Raspberry Pi 5			
	Raspberry Pi 4	Raspberry Pi 5	Comparison
CPU	Broadcom BCM2711, CortexA-72 (ARM v8) 64-bit SOC @ 1.8 GHz	Broadcom BCM2712, quad-core CortexA-76 (ARM v8) 64-bit SOC @ 2.4 GHz	2-3x Performance
RAM	1 GB, 2 GB, 4 GB, 8 GB	1 GB, 2 GB, 4 GB, 8 GB	
Connectivity		2.4/5GHz 802.11ac wireless Bluetooth 5.0. BLE 1x PCIe 2.0 interface 2x USB 3.0 2x USB 2.0 GPIO 40-pins	High-speed Peripheral Interface
OS and Data Storage	microSD Card slot	microSD Card slot with support for high-speed SDR104 mode	2x interface speed
Input Power	5 V DC @ 3A (USB-C)	5 V DC @ 5 A DC (PD-enabled)	
Real-time clock (RTC)	N/A	RTC and RTC battery connector	

Table 2 provides a comparative overview of the essential elements of the microprocessor. The results of the Raspberry Pi 4 show many feature implications that conflict with our standard of hardware. The processing speed and interface speeds are below standard compared to the Raspberry Pi 5, which meets the requirements. Thus, the Raspberry Pi 5 was favoured over the Raspberry Pi 4.

As a result, the Raspberry Pi 5 was determined to be the best fit for our project design as it is a powerful microprocessor capable of hosting a web application with ease.

Remote Access and Web Deployment Alternatives

There were several other options available for remote access and web deployment besides the Cloudflare Tunnel approach used in this project. One of the most straightforward alternatives is port forwarding through the router, which makes the Raspberry Pi's local server accessible over the internet. However, this method comes with serious security concerns, including the risk of exposing the device to unauthorized access or DDoS attacks, and often requires a static IP or

dynamic DNS setup [19]. Another common option is hosting the application on a cloud platform like AWS or Azure. These services offer high scalability and global availability, but they also introduce recurring costs and add complexity when it comes to deployment and maintenance [20]. A VPN-based solution could also be used to access the local network remotely, though it typically involves more advanced configuration and setup on both the server and client sides. In comparison, Cloudflare Tunnels offered a more secure, cost-effective, and simplistic solution, making it the ideal fit for an application such as our Embedded System Design to Manage a Smart Office Environment

Material/Component list

The overview of the cost of materials and components used in OfficeEase is shown in Table 3.

Table 3. Total Cost of Materials and Components for the OfficeEase Design

Item	Cost	Quantity
Raspberry Pi 5 + Kit	$113.99	1
Website Domain	$6.99	1
MongoDB	$0	1
Cloudflare Services	$0	1
Totalled Cost	$120.98	

In Table 3, the considerable item in our expenses is the Raspberry Pi 5 Kit, which contains the Raspberry Pi 5 board and protective case. Although the cost of the kit is higher compared to the other materials, the value of the kit makes it a cost-effective purchase compared to other similar products. The other expensive expense is the exclusive website domain, which allows employees to access the web application that is operating on the Raspberry Pi board. Other components include the MongoDB database and Cloudflare services, which have free offers but limited accessibility. However, the scope of these offers falls within the range of our product, making it an ideal component for OfficeEase as it is cost-effective.

Measurement and Testing Procedures

The Logcat provided by Android Studio was used extensively during the development of the Android application. By utilizing the Studio's Log class, initiated logs from the device can be sent to a computer in real time, allowing the group to easily trace the code that raises an exception and verify the functionality of the implemented features. The following statement was used to print to the Logcat:

```
Log.d("myTag", "This is my message")
```

Aside from debugging and error purposes, the Log was also used to dynamically calculate and print the latency between the Android and the server, as shown in Figure 17. The latency is easily determined by finding the lifespan of the thread responsible for server synchronization. In addition to the Logcat, the Profiler of Android Studio (specifically, the Live Telemetry) in Figure 18 was used to analyze and monitor the CPU and memory usage. The app was tested under different scenarios – for instance, adding projects and auto-assigning employees – to determine the worst-case performance.

To assess the performance of the OfficeEase embedded server, a 30-day testing period was conducted using the live web application deployed via Cloudflare Tunnel on a Raspberry Pi 5. The Cloudflare analytics in Figure 19 displays key metrics such as total HTTP requests, bandwidth usage, and cache efficiency. These graphs helped visualize user interaction patterns, data delivery rates, and system responsiveness during critical testing phases and simulated demos. Particular attention was given to how the server handled real-time traffic. This approach ensured the server and overall system could perform with concurrent users under real-world conditions.

MongoDB Atlas's performance monitoring tools were used to track the behaviour of the cloud-hosted database during live testing. As shown in Figure 20, it provides key metrics such as read/write throughput, connection counts, and network traffic across all three shards of the cluster. These values helped measure the backend's ability to handle concurrent operations such as task updates, user logins, and data syncing between the mobile and web applications. These measurements were key in testing our system's readiness for real-world application and future scalability.

Performance Measurement Results

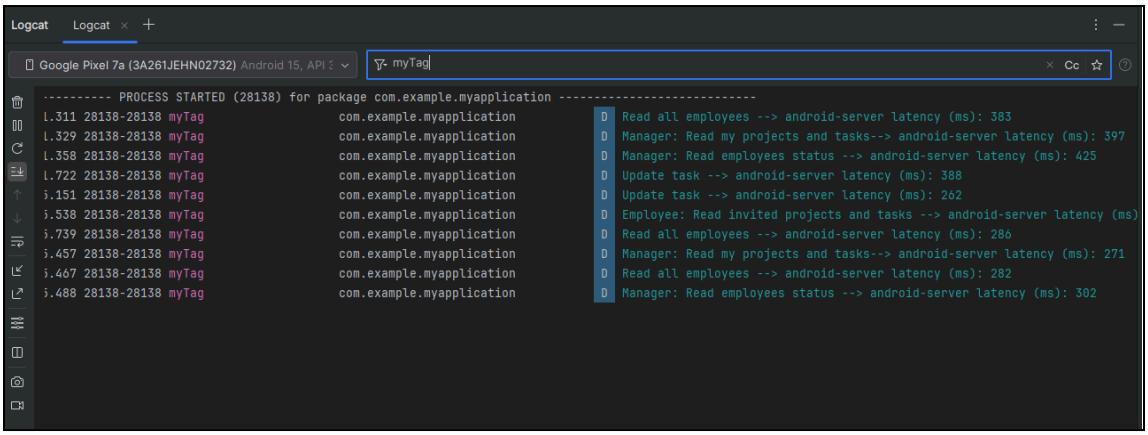

Figure 17. Latency of Android-Server Communication using Logcat.

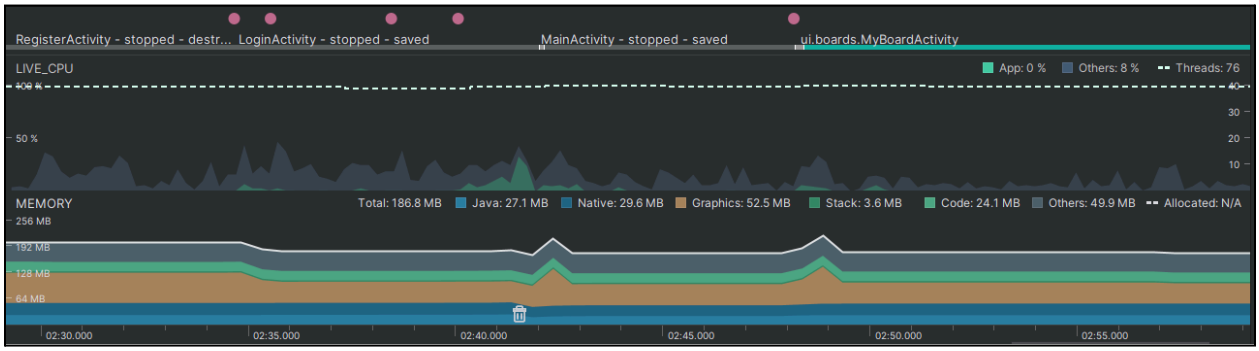

Figure 18. Performance of the Android Application (i.e. CPU and Memory Usage) using Profiler.

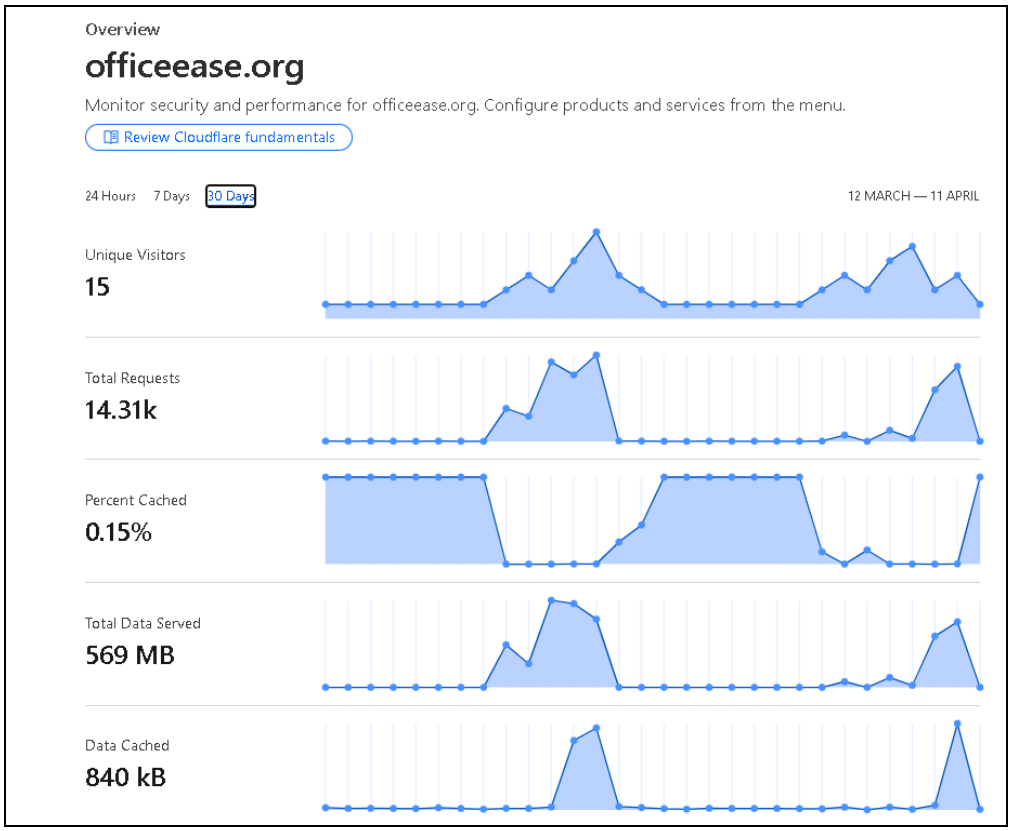

Figure 19. Cloudflare Analytics Overview for the officeease.org Web Application

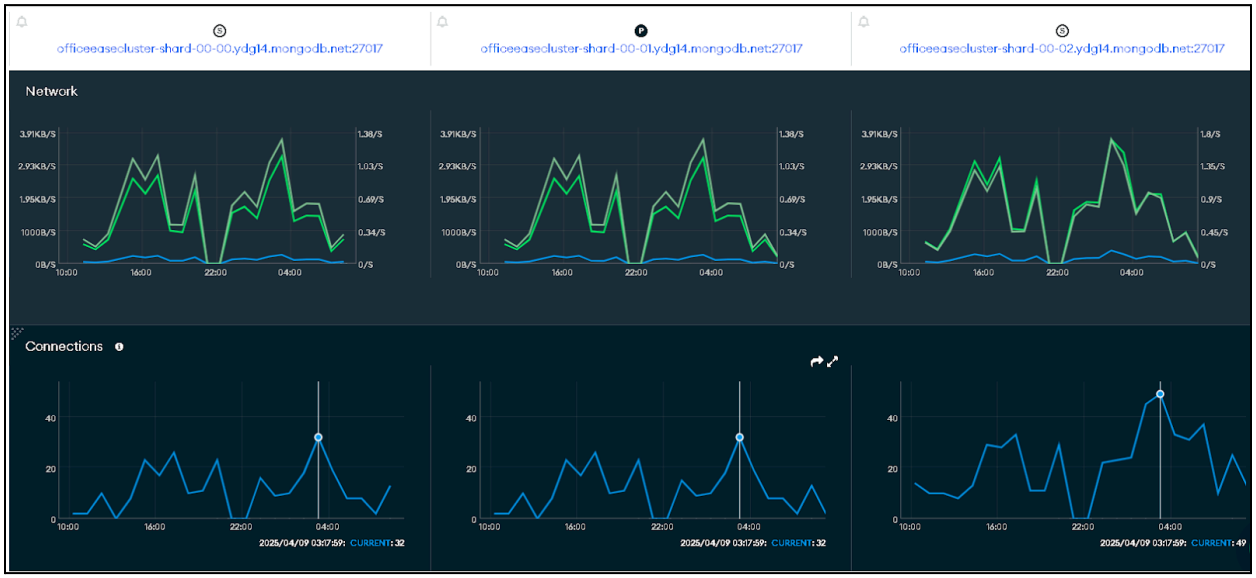

Figure 20. MongoDB Atlas Cluster Metrics for Backend Data Transfer
(Network Activity and Connection Count across Database Shards during System Testing).

Analysis of Performance

It is apparent that the CPU and memory usage of the Android app tend to reach their peak whenever the screen transitions to a new activity (or when the user navigates to a different feature). The maximum CPU and memory utilization are 40% and 192MB, respectively. These parameters are determining factors if our app is well optimized, as they correlate to how much it consumes the battery of the device and how frequently the user will encounter slow and choppy animations [21] .

Over a 30-day testing period, the OfficeEase web application, deployed via Cloudflare Tunnel on a Raspberry Pi 5, recorded 15 unique visitors and 14,310 total HTTP requests, serving 569 MB of data. Despite a low cache hit rate of 0.15%, due to the real-time, user-specific nature of the content, the system consistently delivered dynamic content without noticeable performance degradation. Usage spikes aligned with key testing and demonstration phases, and no service interruptions or data loss were observed. These results confirm the stability, reliability, and responsiveness of the embedded system infrastructure under realistic user interaction conditions.

During system testing, MongoDB Atlas metrics showed consistent network activity across all three shards of the OfficeEase database cluster, with peak data transfer rates reaching approximately 3.9 KB/s. The graphs also indicate regular bursts of activity around key testing windows, suggesting healthy read/write operations and effective backend communication with the embedded server. Connection counts steadily rose to peaks of 40–50 concurrent connections, demonstrating stable and scalable access patterns under simulated multi-user conditions. These performance indicators confirm that the cloud-hosted database was able to handle real-time project management operations and sync data between web and mobile clients without bottlenecks or connectivity failures.

Conclusions

Our Embedded System Design to Manage a Smart Office Environment capstone project was completed successfully. Our system integrates project management, resource scheduling, employee analytics, and mobile connectivity into a unified web and Android platform. Hosted on a Raspberry Pi 5 and supported by MongoDB Atlas, the system was designed to streamline workplace operations by improving communication, organization, and productivity. The completed application features dynamic project and task assignment, calendar and reservation tools, role-specific dashboards, and performance tracking. All of which were validated through testing under real-world concurrent user scenarios.

Overall, our system met all of the set objectives for this capstone project by developing a working prototype infrastructure, including smartphones to serve as a management system for the employees of a smart office or working environment. Our system is capable of generating reminders, managing meetings, project progress and much more.

Despite the overwhelming success of our capstone project, we encountered our fair share of difficulties, specifically while implementing our user authentication system and endpoint API's. Particularly for distinguishing between manager and employee accounts. Despite early success with database connectivity and session management, handling multiple user roles with proper access control required several iterations and led to significant delays in development. We also had some difficulties connecting the Android mobile app to the database through MongoDB Realm due to its complexity and lack of documentation. All errors and faults were eventually resolved.

Future improvements for OfficeEase include refining the authentication system for greater security and maintainability, adding support for iOS devices, and implementing advanced scheduling tools with real-time conflict detection. Expanding the analytics features with AI-driven insights and integrating push notifications would also enhance user experience and responsiveness. Overall, our implementation of our Embedded System Designed to Manage a Smart Office Environment was a tremendous success, and we are looking forward to expanding and improving the capabilities of OfficeEase in the future.

References

[1] D. Marikyan, S. Papagiannidis, R. Ranjan, and O. Rana, “Working in a Smart Home-office: Exploring the Impacts on Productivity and Wellbeing,” pp. 275–282, 2021, doi: 10.5220/0010652200003058

[2] A. Shashank and R. Vincent, “An Innovative Approach for Secured Smart Office and Home System using IoT,” Journal of Physics. Conference series, vol. 1716, no. 1, pp. 12056-, 2020, doi: 10.1088/1742-6596/1716/1/012056

[3] A. Salosin, O. Gamayunova, and A. Mottaeva, “The effectiveness of the Smart Office system,” J. Phys.: Conf. Ser., vol. 1614, no. 1, p. 012028, Aug. 2020, doi: 10.1088/1742-6596/1614/1/012028.

[4] “How to Build a Home Server,” Acer Corner, Dec. 25, 2023. https://blog.acer.com/en/discussion/1163/how-to-build-a-home-server (accessed Nov. 30, 2024).

[5] Anushree Burad, “How Much RAM for a Dedicated Server: Find Your Perfect Fit,” Bluehost Blog, Nov. 14, 2024. https://www.bluehost.com/blog/how-much-ram-for-dedicated-server/

[6] “Raspberry Pi 5”,

[7] Kyuchukova, D., et al, "A study on the possibility to use Raspberry Pi as a console server for remote access to devices in virtual learning environments," in 2015 International Conference on Information Technology Based Higher Education and Training (ITHET), 2015, pp. 1–4.

[8] H. Mohammady. "Enhancing the security and privacy of home storage servers in private clouds using zero trust principles", 2024.

[9] J. Waguia, A. Menshchikov, "Threats and security issues in cloud storage and content delivery networks: Analysis," in 2021 28th Conference of Open Innovations Association (FRUCT), 2021, pp. 194–199.

[10] M. Ward. "NoSQL database in the cloud: MongoDB on AWS," in Amazon Web Services, 2013.

[11] M. Nguyen. "Full-stack crud application: User management system," 2023.

[12] Tulsyan, R., et al, "The Impact of JavaScript Frameworks on Website Performance and User Experience," in 2024 IEEE International Conference on Big Data & Machine Learning (ICBDML), 2024, pp. 299–305.

[13] S. Dalimunthe, J. Reza, A. Marzuki. "The model for storing tokens in local storage (Cookies) using JSON Web Token (JWT) with HMAC (Hash-based Message Authentication Code) in e-learning systems," in Journal of Applied Engineering and Technological Science, vol. 3, no. 2, pp. 149–155, 2022.

[14] N. Dongre, T. Agrawal. "A Research On Android Technology With New Version Nougat (7.0, 7.1)," in IOSR Journal of Computer Engineering, vol. 19, no. 02, pp. 65–77, 2017.

[15] Liu, P., et al. "Understanding the quality and evolution of Android app build systems," in Journal of Software: Evolution and Process, vol. 36, no. 5, pp. e2602, 2024.

[16] “Choose a database: Cloud Firestore or Realtime database | Firebase,” Google, https://firebase.google.com/docs/firestore/rtdb-vs-firestore?_gl=1%2A19drryw%2A_up%2AMQ..%2A_ga%

[17] D. Wheat, Arduino internals. Apress, 2012.

[18] “RPi 5 vs RPi 4: A Comparison of Raspberry Pi Benchmarks and More,” Elektor, Nov. 01, 2023. https://www.elektormagazine.com/news/raspberry-pi-5-vs-raspberry-pi-4-a-comparison

[19] Wang, H., et al. "Port Forwarding Services Are Forwarding Security Risks," in arXiv preprint arXiv:2403.16060, 2024.

[20] N. Polani, U. Haider, "Comparing the Performance and Cost of Different Cloud Service Providers for Small Applications: A Comparison between AWS and Azure," in 2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC), 2024, pp. 182–184.

[21] [13] Mahendra, Mohammad, and Bayu Anggorojati. “Evaluating the performance of Android based cross-platform App Development Frameworks.” Proceedings of the 6th International Conference on Communication and Information Processing, 27 Nov. 2020, pp. 32–37, https://doi.org/10.1145/3442555.3442561.

Appendices

Main Methods For Web Application

Home Screen

```
326
327       // Project Management State
328       const [projects, setProjects] = useState([]);
329       const [selectedProject, setSelectedProject] = useState(null);
330
331       const [isCreatingProject, setIsCreatingProject] = useState(false);
332       const [isCreatingTask, setIsCreatingTask] = useState(false); // Toggle create task form
333
334       // Fetch all projects for the authenticated user
335       useEffect(() => {
336         const fetchProjects = async () => {
337           try {
338             const res = await axios.get("/api/projects");
339             setProjects(res.data);
340           } catch (error) {
341             console.error("Error fetching projects:", error);
342           }
343         };
344         fetchProjects();
345       }, []);
346
347       // Handle project selection
348       const handleProjectSelect = async (project) => {
349         try {
350           // Fetch the latest tasks for the selected project
351           const res = await axios.get(`/api/tasks?projectId=${project._id}`);
352           setSelectedProject({
353             ...project,
354             tasks: res.data, // Update the tasks with the latest data from the database
355           });
356         } catch (error) {
357           console.error("Error fetching tasks for project:", error);
358         }
359       };
360
```

Figure A.1. Initialization Code of the Home Screen

```
363       // Handle new project creation
364       const handleProjectCreated = (newProject) => {
365         setProjects([...projects, newProject]);
366       };
367
368       // Handle project deletion
369       const handleDeleteProject = async (projectId) => {
370         try {
371           await axios.delete(`/api/projects`, { data: { projectId } });
372           setProjects(projects.filter(project => project._id !== projectId));
373           if (selectedProject && selectedProject._id === projectId) {
374             setSelectedProject(null); // Clear the selected project if it was deleted
375           }
376         } catch (error) {
377           console.error("Error deleting project:", error);
378         }
379       };
380
381
382       // Handle new task creation
383       const handleTaskCreated = async (newTask) => {
384         // Re-fetch the project to get fully populated task data
385         await handleProjectSelect(selectedProject);
386       };
387
```

Figure A.2. Project and Task Handle Functions

```
// Handle task completion
const handleTaskCompletion = async (taskId, isCompleted) => {
  try {
    // Update backend
    await axios.patch("/api/tasks", { taskId, isCompleted });

    // Fetch fresh task data from server
    const res = await axios.get(`/api/tasks?projectId=${selectedProject._id}`);
    const updatedTasks = res.data;

    // Update selectedProject with fresh tasks
    const updatedSelectedProject = {
      ...selectedProject,
      tasks: updatedTasks,
    };
    setSelectedProject(updatedSelectedProject);

    // Update project in full projects list
    setProjects((prevProjects) =>
      prevProjects.map((proj) =>
        proj._id === updatedSelectedProject._id ? updatedSelectedProject : proj
      )
    );
  } catch (error) {
    console.error("Error updating task or fetching updated list:", error);
  }
};


// Calculate task progress
const completedTasks = selectedProject?.tasks?.filter((task) => task.isCompleted).length || 0;
const totalTasks = selectedProject?.tasks?.length || 0;
const taskProgress = totalTasks > 0 ? Math.round((completedTasks / totalTasks) * 100) : 0;
```

Figure A.3. Task Completion Handle Function

```
          {sortedEmployees.map((emp, index) => {
            const employee = employees.find(e => e._id === emp.id);
            const avgMs = emp.avg;
            const avgDays = Math.floor(avgMs / (1000 * 60 * 60 * 24));
            const avgHours = ((avgMs % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60)).toFixed(2);
            const avgDisplay = isFinite(avgMs) ? `${avgDays}d ${avgHours}h` : "N/A";

            const getRankBadge = (idx) => {
              switch (idx) {
                case 0: return <span role="img" aria-label="gold">🥇</span>;
                case 1: return <span role="img" aria-label="silver">🥈</span>;
                case 2: return <span role="img" aria-label="bronze">🥉</span>;
                default: return <span style={{ fontWeight: "bold" }}>#{idx + 1}</span>;
              }
            };
```

Figure A.4. Employee Rank Sort Function

```
{/* Workload Pie Chart View */}
{showWorkloadChart && (
  <>
    <h3 style={{ fontSize: "2rem", marginBottom: "1.5rem" }}>Employee Workload Overview</h3>
    <div style={{ marginTop: "1rem", width: "90%", maxWidth: "340px" }}>
      <Pie
        data={{
          labels: ["Free (0-2)", "Moderate (3-5)", "Busy (6-8)", "Overloaded (9+)"],
          datasets: [
            {
              label: "# of Employees",
              data: (() => {
                const counts = { free: 0, moderate: 0, busy: 0, overloaded: 0 };
                Object.values(employeeWorkload).forEach(count => {
                  if (count <= 2) counts.free++;
                  else if (count <= 5) counts.moderate++;
                  else if (count <= 8) counts.busy++;
                  else counts.overloaded++;
                });
                return [counts.free, counts.moderate, counts.busy, counts.overloaded];
              })(),
```

Figure A.5. Visual Workload Pie Chart Function

```
{selectedEmployee && employeeTasks.length > 0 && (() => {
  const completedTasks = employeeTasks.filter(task => task.isCompleted && task.completedAt);
  const totalCompleted = completedTasks.length;
  const totalMs = completedTasks.reduce((sum, task) => sum + (new Date(task.completedAt) - new Date(task.createdAt)), 0);
  const avgMs = totalCompleted > 0 ? totalMs / totalCompleted : null;
  const avgDays = avgMs !== null ? Math.floor(avgMs / (1000 * 60 * 60 * 24)) : null;
  const avgHours = avgMs !== null ? ((avgMs % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60)).toFixed(2) : null;
  const avgDisplay = avgMs !== null ? `${avgDays}d ${avgHours}h` : "N/A";
  const rank = sortedEmployees.findIndex(e => e.id === selectedEmployee._id) + 1;

  return (
```

Figure A.6. Restriction Conditions for Task Display

CreateTask.js

```
useEffect(() => {
  const fetchEmployees = async () => {
    try {
      const res = await axios.get("/api/employees");
      setEmployees(res.data);
    } catch (error) {
      console.error("Error fetching employees:", error);
    }
  };

  fetchEmployees();
}, []);

useEffect(() => {
  const fetchWorkload = async () => {
    try {
      const res = await axios.get("/api/employee-workload");
      setWorkload(res.data);
    } catch (error) {
      console.error("Error fetching workload:", error);
    }
  };

  fetchWorkload();
}, []);

useEffect(() => {
  if (skillsRequired.length === 0) {
    setFilteredEmployees(employees);
    return;
  }
```

Figure A.7. Initialization Code of CreateTask.js

```
    const matchedEmployees = employees.filter(employee =>
      skillsRequired.every(skill => employee.skills.includes(skill))
    );

    setFilteredEmployees(matchedEmployees);
  }, [skillsRequired, employees]);

  const handleSkillChange = (e) => {
    const selectedSkill = e.target.value;

    if (skillsRequired.includes(selectedSkill)) {
      setSkillsRequired(skillsRequired.filter(skill => skill !== selectedSkill));
    } else if (skillsRequired.length < 3) {
      setSkillsRequired([...skillsRequired, selectedSkill]);
    }
  };

  const handleEmployeeToggle = (empId) => {
    if (assignedEmployees.includes(empId)) {
      setAssignedEmployees(assignedEmployees.filter(id => id !== empId));
    } else if (assignedEmployees.length < numEmployees) {
      setAssignedEmployees([...assignedEmployees, empId]);
    }
  };

  const handleNumEmployeesChange = (e) => {
    const value = parseInt(e.target.value, 10);
    setNumEmployees(isNaN(value) || value < 1 ? 1 : value);
  };
```

Figure A.8. Handle Functions of CreateTask.js Part 1

```
  const handleAutoAssign = () => {
    const eligible = filteredEmployees.map(emp => {
      const count = workload[emp._id] || 0;
      return { ...emp, taskCount: count };
    });

    const sorted = eligible.sort((a, b) => a.taskCount - b.taskCount);
    const selected = sorted.slice(0, numEmployees).map(emp => emp._id);
    setAssignedEmployees(selected);
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await axios.post("/api/tasks", {
        name,
        projectId,
        skillsRequired,
        assignedEmployees,
      });

      onTaskCreated(res.data);
      setName("");
      setSkillsRequired([]);
      setAssignedEmployees([]);
      setNumEmployees(1);
    } catch (error) {
      console.error("Error creating task:", error);
    }
  };
```

Figure A.9. Handle Functions of CreateTask.js Part 2

Main Methods For Android Application

```
public static void readInvitedProjects(){
    MongoCollection<Document> col_project = getCol( collection: "projects");
    MongoCollection<Document> col_task = getCol( collection: "tasks");
    long t1  = Calendar.getInstance().getTimeInMillis();
    col_project.find().iterator().getAsync(new App.Callback<MongoCursor<Document>>() {
        @Override
        public void onResult(App.Result<MongoCursor<Document>> result) {
            projects.clear();
            while(result.get().hasNext()){
                Document doc = result.get().next();
                Project p = new Project(doc);
                col_task.find(new Document("projectId", p._id)).iterator().getAsync(new App.Callback<MongoCursor<Document>>() {
                    @Override
                    public void onResult(App.Result<MongoCursor<Document>> result) {
                        boolean has_project = false;
                        while(result.get().hasNext()){
                            Document doc = result.get().next();
                            if(doc.get("assignedEmployees") != null){
                                Task t = new Task(doc);
                                has_project |= t.assignedEmployees.contains(user_id);
                                p.tasks.add(t);
                            }
                        }
                        if(has_project) projects.add(p);
                    }
                });
            }
            long t2  = Calendar.getInstance().getTimeInMillis();
            Log.d( tag: "myTag",  msg: "Employee: Read invited projects and tasks --> android-server latency (ms): "+(t2-t1));
```

Figure A.10. Thread used for reading the projects (and tasks) of the logged-in user from the database.

```
public static void updateTask(Task task){
    MongoCollection<Document> col = getCol( collection: "tasks");
    long t1  = Calendar.getInstance().getTimeInMillis();
    col.updateOne(new Document("_id", task._id), task.get()).getAsync(new App.Callback<UpdateResult>() {
        @Override
        public void onResult(App.Result<UpdateResult> result) {
            long t2  = Calendar.getInstance().getTimeInMillis();
            Log.d( tag: "myTag",  msg: "Update task --> android-server latency (ms): "+(t2-t1));
        }
    });
}
```

Figure A.11. Thread used for updating/modifying a task in the database.

```
col.find().iterator().getAsync(new App.Callback<MongoCursor<Document>>() {
    @Override
    public void onResult(App.Result<MongoCursor<Document>> result) {
        employees_assigned.clear();
        employees_pending.clear();
        employees_completionN.clear();
        employees_completionTime.clear();
        while(result.get().hasNext()){
            Document doc = result.get().next();
            try{
                ArrayList<ObjectId> members = (ArrayList<ObjectId>) doc.get("assignedEmployees");
                boolean isCompleted = doc.getBoolean( key: "isCompleted");
                for(ObjectId m:members){
                    if(employees_assigned.containsKey(m)){
                        employees_assigned.put(m, employees_assigned.get(m)+1);
                        if(!isCompleted){
                            employees_pending.put(m, employees_pending.get(m)+1);
                        }
                        else{
                            employees_completionN.put(m, employees_completionN.get(m)+1);
                            employees_completionTime.put(m, employees_completionTime.get(m)+doc.getDate( key: "completedAt").getTime()-doc.getDate( key: "createdAt").getTime());
                        }
                    }
                    else{
                        employees_assigned.put(m, 1);
                        if(!isCompleted){
                            employees_pending.put(m, 1);
                            employees_completionN.put(m, 0);
                            employees_completionTime.put(m, (long) 0);
                        }
                        else{
                            employees_completionN.put(m, 1);
                            employees_completionTime.put(m, doc.getDate( key: "completedAt").getTime()-doc.getDate( key: "createdAt").getTime());
                            employees_pending.put(m, 0);
                        }
                    }
```

Figure A.12. Thread used for obtaining the *work efficiency* of all employees from the database.

```
binding.buttonAutoselect.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        int val = Integer.valueOf(binding.editTextMembersNum.getText().toString());
        ArrayList<CheckBox> boxes = new ArrayList<>(checkboxes);

        for(CheckBox cb:boxes){
            cb.setChecked(false);
        }

        ArrayList<CheckBox> candidate = new ArrayList<>();

        CheckBox min_box = null;
        int min_val;
        while(!boxes.isEmpty() && candidate.size() < val) {
            min_val = 999999;
            for (int i = 0; i < boxes.size(); i++) {
                ObjectId memberID = Server.getEmployeeId(boxes.get(i).getText().toString().split( regex: " - ")[0]);
                if(!Server.employees_assigned.containsKey(memberID)){
                    min_val = 0;
                    min_box = boxes.get(i);
                    break;
                }
                else if(Server.employees_assigned.get(memberID) < min_val) {
                    min_val = Server.employees_assigned.get(memberID);
                    min_box = boxes.get(i);
                }
            }
            candidate.add(min_box);
            boxes.remove(min_box);
        }

        for(CheckBox cb:candidate){
            cb.setChecked(true);
        }
```

Figure A.13. Algorithm used to auto-assign employees for a task.