Centre for Industrial Software

sdu.dk

#sdudk

DeskMotion

Semester Project 3 - 2024

Group16 - PET
Szymon Jerzy Nowacki - sznow23@student.sdu.dk
Mathias Lund-Hansen - mlund23@student.sdu.dk
Samuel Osei Adjabeng - saadj23@student.sdu.dk
Ahmad Mohammad Al-Qadiri - ahalq21@student.sdu.dk
Petar Spirovski - pespi23@student.sdu.dk
Patrik Muravski - pamur24@student.sdu.dk
Tim Lukas Adam - tiada23@student.sdu.dk

Krzysztof Sierszecki - krzys@mmmi.sdu.dk

SDU

PET

Personal Electronic Transactor

The Commodore PET is a line of personal computers produced starting in 1977. A single all-in-one case combines a MOS Technology 6502 microprocessor, the dialect of the BASIC programming language in read-only memory, keyboard, monochrome monitor, and, in early models, a cassette deck. The systems were a top seller in the Canadian and United States education markets, as well as for business use in Europe. The PET line was discontinued in 1982 after approximately 219,000 machines were sold.
Website: [1] " Commodore PET " Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Commodore_PET[i]

Abstract

This report is about the design and implementation of a desk management system in the form of a web application aimed at optimizing the usage and health benefits of motorized desks in modern office spaces. The project addresses the rising needs of ergonomic office solutions as working long hours at a desk can cause several health issues like back pain or reduced circulation. A desk management system addresses these challenges by simplifying the management of office spaces and giving more insights on desk usage data to benefit both employees and employers.

The main objectives of the project include the development of a web application that is user-friendly, connects to a database for collecting and storing desk and user data and includes embedded elements to notify users about desk movements through sounds or buzzers at their desk.

The project was developed in the framework Razor Page with C# and C for the backend and bootstrap and customized CSS classes for the styling of the page, ensuring good accessibility and user-experience.

The main objectives were all successfully implemented, delivering a web application including all the outlined features like different user types, desk management and reservation options, monitoring of desk activity and displaying insights on desk usage. This work highlights the importance of innovative office solutions and can be utilized to gather data that can be used to improve the employees' health by realizing the full potential of motorized desks.

Contents

1. Introduction

The increasing usage of motorized desks in office spaces is showing a growing focus on the employees' health and productivity. The desks are offering adjustable heights and an easy switch between different working positions, improving the ergonomics of the workspace and supporting an overall employee well-being. However, the full potential of these new desks can still be improved by adding smart monitoring, collecting desk-usage data and insights and predictive maintenance functionality.

Therefore, this Semester Project aims to design and implement a distributed desk management system in cooperation with Linak and their desks, including the integration of a database for data collection and analysis and embedded system in form of a raspberry pi for safety reasons. The purpose of the project is to enhance desk-monitoring and desk usage, providing practical value to companies and employees. For that the system needs to support different user types like employees and admins, provide an interactive Graphical User Interface and a working database and embedded system, all communicating with each other. The application should be able to manage as many desks as needed and provide valuable insights without errors or performance issues occurring.

1.1. Motivation

Due to the increasing need for smarter office solutions this project is an important part of the transformation to healthier and more productive workspaces. That transformation is essential especially with the significant increase in people spending long hours in front of a computer and sitting at their desks. Those long hours of sitting at a desk have been linked to different health issues and have been proven to be one of the main reasons for back pain, poor posture, and reduced circulation. That underlines the importance of a healthy and flexible workspace, and ergonomic solutions are needed more than ever. Motorized desks address that issue and provide a great solution by enabling the user to easily switch between sitting and standing positions multiple times on a long workday. They integrate more movement and flexibility into the modern office world. Introducing desk usage monitoring and collecting more data to analyze fastens that transformation and enables new opportunities and more tools to improve modern offices' efficiency and comfort, for example a better maintenance process, by monitoring desk behavior, and errors, and being able to predict when a desk will break down or needs to be checked.

1.2. Objective

- Create a Web-application for different user types like employees and admins, that allows the users to register, login, and control and get insights on their desk. Depending on their role and assigned access they can also manage users, control, and configure multiple desks.
- Set up a database with different tables referencing each other to be able to store diverse types of data in the application.

- Ensure reliable communication between Front-end and Back-end including saving data, user input and configurations in the database and handling requests from the desk API.
- Use graph libraries to better visualize desk statistics and trend intuitively and interactively.
- Allow administrators or specific roles in the application to automate or schedule specific desk movements at set times, for example to make the cleaning of an office easier.
- Use a raspberry pi as the embedded system to inform the users when the desk is moving to be careful.

1.3. Delimitation

The inclusion of the embedded system in our project will be limited to simple alerts of buzzing and making noise when the desk is moving without implementing further functionality like the option to take user input or display messages.
The group was not able to effectively test the application, especially for a larger scope including multiple desks in an actual office environment, since for the project only two desks were provided.

2. Methodology

This section details the approach, techniques, tools and frameworks we used in our project. In it we build on knowledge from previous semesters and reflect our transition from Scrum to Kanban and from Jira to GitHub Projects.

2.1. Overview of the Methodology

In previous semesters we all used Scrum as our Methodology. However, we found Scrum ceremonies, such as Daily Scrum, frequent retrospectives etc. to be too cumbersome and at times closer to Waterfall than agile especially in an academic setting where meeting daily is difficult. Kanban was chosen for these reasons:

- To minimize the number of scheduled meetings and give the team more time for development and problem solving.
- When done well it can adapt to academic schedules and last-minute requirement changes by allowing tasks to be added, reprioritized or moved anywhere on the board without waiting for the end of a sprint.
- Kanban revolves around the kanban board, while Scrum can and does employ it, it doesn't focus on it and has too many things to do before adding something to the board, such as sprint planning, backlog grooming, and sprint reviews.

Kanban is fundamentally designed to be a base, the essentials, from which teams can add and remove processes according to their needs. And there were some practices we wanted and others that our supervisor recommended we use:

- No estimations required, meaning we limited our tasks to ideally something that could be finished within a sprint and split up tasks that exceeded that time.
- Using Moscow for prioritization.
- A one-week sprint with weekly meetings to ensure tasks are created, assigned and completed.

2.2. Tools and Technologies

GitHub Projects
We switched from Jira (used in previous semester) to GitHub Projects, aiming to centralize our tools with our repository of choice so we would get a simpler solution by decreasing the need to navigate between sites. However, like Jira, GitHub Projects is not a simple solution either requiring a non-trivial amount of setup, needing to use Project Actions to ensure Issues get added to the board and that sprints are organized. GitHub Projects is designed to be a generic solution, meaning unlike Jira which is designed for Scrum or Trello for Kanban, for GitHub Projects you need to choose a template, but that template is too basic and lacking compared to others.

Discord
We used Discord as our primary channel for communication. Discord is great for this as it is a tool all students know and allow for the creation of voice and text channels for different topics. And it allows for communication without the need for formal meetings.

Email

Email is our formal communication channel, from where we communicate with other groups, supervisors and staff. It allows us to document approvals and feedback from our supervisors.

Git and GitHub

We created a public GitHub repository for collaboration. We applied GitHub Workflow where there is a main branch, and a branch is created for each feature or bug fix and said branch is merged into main via a pull request. We did introduce a dev branch in the middle of the project due to the suggestion of a team member, however it didn't change the workflow much and did not receive much use.
We utilized GitHub Actions for automatic testing and planned to use it for publishing the Webserver to Docker Hub. We used issue and pull request templates to ensure Style guidelines were followed, such as test-driven development and that all tests pass.

Technologies

For our web application we chose the Microsoft ecosystem due to our familiarity with the programming language C#. For the webserver we chose to use Asp.NET Razor Pages due to its simplicity since it uses page-based navigation, meaning the URL is determined by the structure of the directory and the controller is placed together with the view, so you don't have to navigate multiple directories to modify one page. To expand upon that simplicity, we chose to use Entity Framework Core (EFCore) as our Object Relational Mapper (ORM), so that we did not need to manually write each migration for our database and for Authentication and Authorization we used Microsoft Identity. We selected PostgreSQL as our database where we deployed both the webserver and database as docker containers using docker compose. And employed xUnit and Moq for unit testing our webserver.

3. Problem Analysis

3.1. Problem Statement

Motorized desks have become common in modern workplaces based on the fact that they generally help improve posture and overall comfort. However, adjusting these desks manually in an optimal way is difficult. Many users struggle to set the desk height correctly, which affects productivity and often leads to discomfort or health issues over time.

The problem does not stop there. Once the workday ends, desks are often left at random heights which becomes a challenge for maintenance staff during cleaning or reorganization since they must manually adjust all desks.

A smarter solution is therefore needed - one that makes desk adjustments automatic and data-driven. By introducing such a system, workplaces could improve user comfort, make maintenance tasks easier, and ensure that desks contribute fully to an efficient and healthy work environment.

3.1.1. Stakeholder Challenges

The challenges faced by different stakeholders often at the cost of the host company show why an automated desk management system is important at the workplace:

- **Office Managers or Admins**: Managing desk usage and ensuring a uniform or orderly appearance across the office can be difficult. Admins are often bombarded by complaints from employees about discomfort or from cleaning staff about inefficiencies in their work. And this ultimately affects the net productivity of the business as a whole.
- **Desk Users**: Adjusting the desk manually in an each-one-for-himself manner without any recommendations or guidance can be frustrating, especially for those unfamiliar with proper ergonomic settings. This often leads to poor posture for desk users, which can cause discomfort or even long-term health issues like back pain, affecting daily productivity.
- **Cleaning Staff**: Desks left at different heights just makes the work of cleaners more difficult, as they must bend, lift, or manually adjust every single desk in order to clean. And apart from the fact that cleaning tasks are more physically demanding, it leads to more overtime work and thus more cost to the company.
- **Technicians**: When desks malfunction, diagnosing problems can be challenging without proper data. Technicians often rely on trial-and-error instead of actual through desk usage data, which is both time-consuming and inefficient.

These challenges make it clear that a system is needed to simplify desk adjustments. A system that will leverage data to improve the workplace for everyone.

3.1.2. Key Challenges

Given the stakeholder challenges at stake, several key issues must be tackled to create a functional and efficient desk management system for host companies.

The issues that immediately come up for solutions include the issues of **ergonomic misuse, desk maintainability** (by cleaners)**, scalability** (to handle the growing or large number of desks)**,** and **data privacy and security** (to enhance system security regarding stored user information)**.** So, the next section deals with the technical constraints in our path as we aim to address these issues.

3.1.3. Technical Constraints

Building a desk management system comes with several technical limitations and considerations. In this section we identify or workout what needs to be considered for a successful system:

- **Hardware Integration**: The system needs to connect IoT components, in our case, the 'Raspberry Pi Pico W' to the mechanical parts of the motorized desks for operation. And since there could very well be different desk models which may use different designs or control methods at the workplace, making sure the system works reliably with all of them is a key challenge.

- **Real-Time Communication**: The system must support real-time communication between the desks, the back-end, and the user interface for it to support any meaningful use. Therefore, achieving low latency and high reliability is crucial for a seamless experience for our desk users.

- **Scalability**: The system needs to handle multiple desks and users simultaneously as it is targeted for normal office space. And because of the logical presumption of limited resources (e.g. hardware limitations), the challenge of ensuring efficient resource management is inevitably one of our constraints – that is if our system is to be able to maintain meaningful performance at any rate as the number of connected desks grows. Another leg to this is to ensure efficient operation without excessive energy consumption as the desks will have embedded systems. This is necessary for sustainability while scaling up.

- **Data Security**: Protecting user data during transmission and storage is essential in today's GDPR world. The system must therefore implement secure protocols to prevent unauthorized access to user information or data breaches.

3.2. Relevant Studies and Gaps

While smart desk solutions and workplace automation have been explored extensively, many existing systems still fall short in some key areas. Firstly, only a few systems offer predefined schedules for automated desk adjustments. This means that tasks like synchronized height changes for cleaning or maintenance are largely still in need of resolution. Additionally, many solutions do not

prioritize user accessibility, and they also often lack intuitive interfaces that cater to both technical and non-technical users.

Our App, ‘DeskMotion’ addresses these gaps with a goal to providing enhanced automation, and predefined scheduling, along with a user-friendly interface, thereby providing a solution that improves functionality and usability across the board.

3.2.1. Conclusion - Impact of Addressing the Problem

An automated desk management system like *‘DeskMotion’* will bring significant improvements to the workplace. First of all, by automating desk adjustments based on user preferences and schedules, employees could maintain better posture, which means less time-offs or hospital visits. And for maintenance staff, the system will save time and effort by eliminating the need to manually adjust multiple desks, thereby reducing physical strain – Simply put, they’ll ‘love’ the desk admins for this.

Additionally, employees would spend less time fidgeting with their desks, allowing them to focus more on their work. A well-managed workspace creates a more efficient and stress-free environment, contributing to higher productivity – i.e. a net positive for the host company. The system’s ability to analyze desk usage data will help office management to identify patterns in desk utilization which could be used to optimize workspace layouts and allocate resources more effectively.

These expected benefits presuppose clear functional and non-functional requirements for building the *‘DeskMotion’* application, such as enabling remote control for desks, ensuring secure data handling, and maintaining real-time responsiveness. In the next chapter, we will outline these requirements in detail to ensure the system effectively addresses the identified challenges.

4. Requirements

We have divided the requirements into Functional and Non-Functional requirements. We use MoSCoW for prioritization and each requirement is presented as a user story which shows who benefits and why:

4.1. Functional Requirements

ID	User Story	MoSCoW
F 1	As a user, I want to remote control my desk through a web app so that I can adjust its height remotely.	Must Have
F 2	As a user, I want visualization of desk usage statistics so that I can understand how they are being used.	Must Have
F 3	As a developer, I want to integrate the Raspberry Pi Pico W (Embedded system) with the web app (distributed system) so that it can communicate with the backend.	Should Have
F 4	As a System-Administrator, I want to automate desks movements at specific times so that maintenance can be done with the least effort.	Should Have
F 5	As a maintainer, I want predictive maintenance alerts so that I can fix issues before they become a problem.	Could Have
F6	As a System-Administrator, I want to see which desks are not yet allocated.	Could Have
F7	As a System-Administrator, I want visualization of desks on the map so we can know where the desks are located.	Could Have

F8	As a Guest, I want reserve a table for temporary use.	Could Have
F9	As a user, I want to see ergonomical messages with recommendations which desk to use depending on my personal characteristics.	Could Have

Table 1 Functional1 requirements

4.2 Non-Functional Requirements

ID	User Story	MoSCoW
NF 1	As a user, I want secure and reliable front-end and back-end communication so that my data is safe and there are no interruptions.	Must Have
NF 2	As a System-administrator, I want a secure database for storing desk and user data so that it is guarded.	Must Have
NF 3	As a user, I want the web app to work on different device types and be accessible so that I can use it from my phone and accommodate my needs.	Could Have
NF 4	As a user, I want to be safe when the desk moves automatically.	Could Have

Table 2 Non2-Functional requirements

5. Design

This chapter describes the structure and key components of the project for management of desks in an office. The purpose of the website is to allow the users to reserve desks, check their availability, and locate them, while admins can view reservations, manage reports, data and more.

5.1. System Architecture

The project is developed with the ASP.NET Core framework in Razor Pages chosen for the simple and understandable design of the model. The system has a Client-Server Architecture, with the front-end built for user interaction, and the backend built for the logic and database interaction.

5.1.1. Frontend

- Developed using HTML, CSS and C# with Razor Pages for dynamic page rendering
- Provide users with a visually engaging PET styled GUI for easy interaction with the website.
- Implement AJAX for sending asynchronous requests without the need of reloading the page.

5.1.2. Backend

- Built in C# with ASP.NET core to handle the logic in desk reservation, desk status and data analysis.
- Developed REST APIs for seamless communication between front-end and back-end, as well as facilitating smooth data management and handling CRUD operations.

5.1.3. Database

- Created a PostgreSQL database inside Docker Containers.
- Used Containers for the benefit of having easy scalability, isolated environment and better management of the database.
- Stores desk reservations, profile information, and analytics data (shown in the admin dashboard).

5.1.4. Dataflow

- Users send requests like reservation of a desk via the front-end.
- The backend interacts with the database sending a request.
- The database processes the request and send a response back to the front-end displaying it to the client.

5.2. Key Functional Modules

5.2.1. User Features

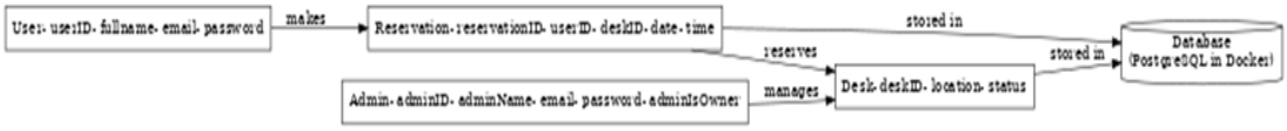
- **Desk Reservation:** User can make a reservation of any desk as long as it's available.
- **Desk Map:** Shows the user which desks are available for reservation, which desk they have reserved, and unavailable desks.
- **Profile management:** In profile management the user can edit information in their account like changing the password or the email they signed up with or setting up a two-factor authentication as an extra security measurement.

5.2.2. Admin Features

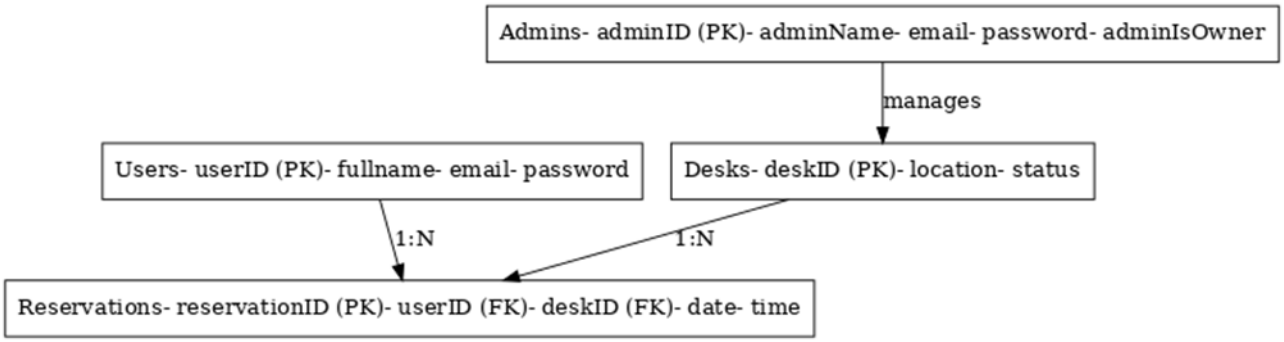
- Reservation management: Admin is available to set desks as unavailable, as well as remove modify desk location.
- Data analysis: The admin has multiple windows for viewing desks, users, reports, as well as charts that the admin can monitor to inspect how users are interacting with the desks.

5.3. Diagrams

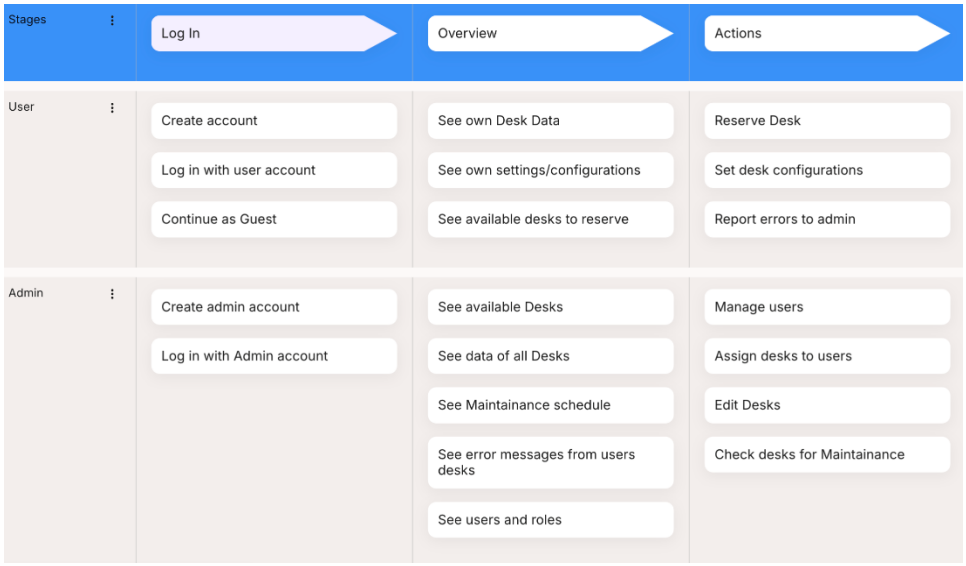
5.3.1. UML Diagram



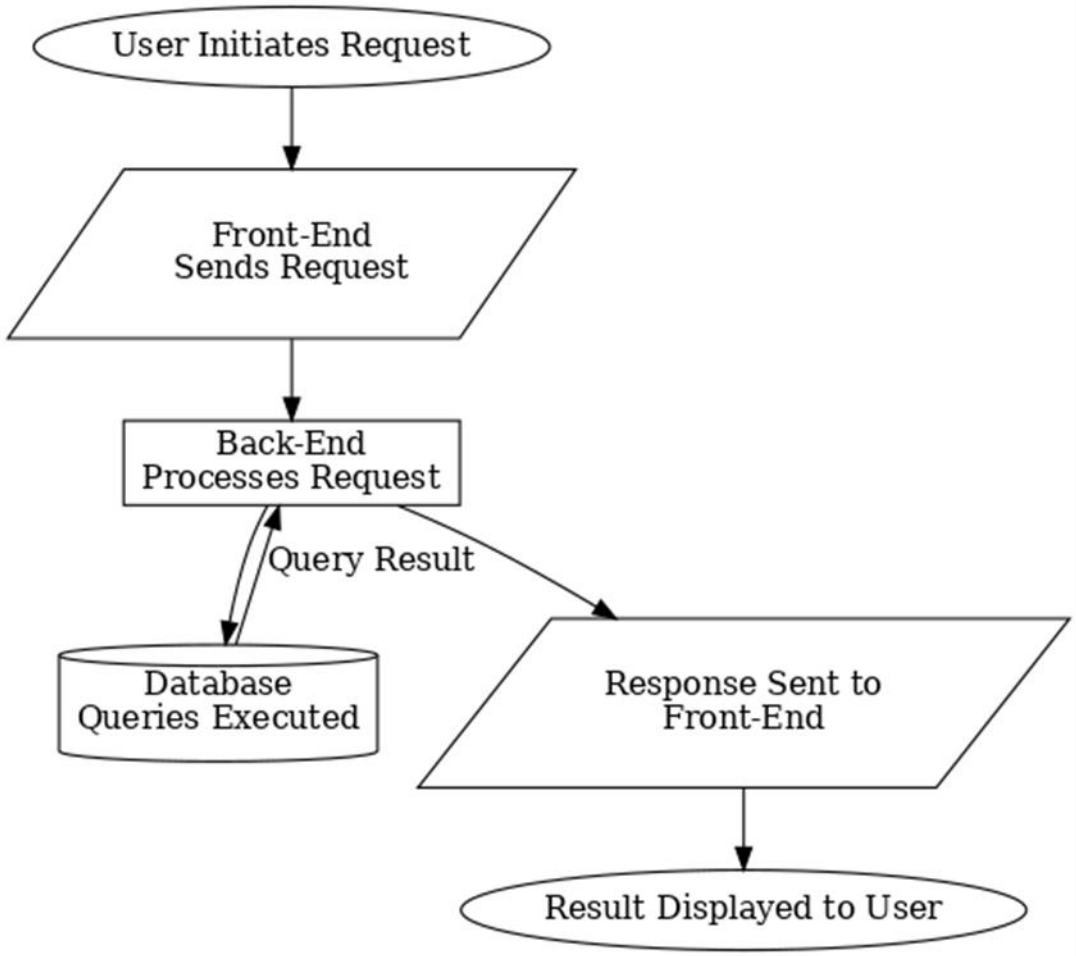
5.3.2. Database Schema



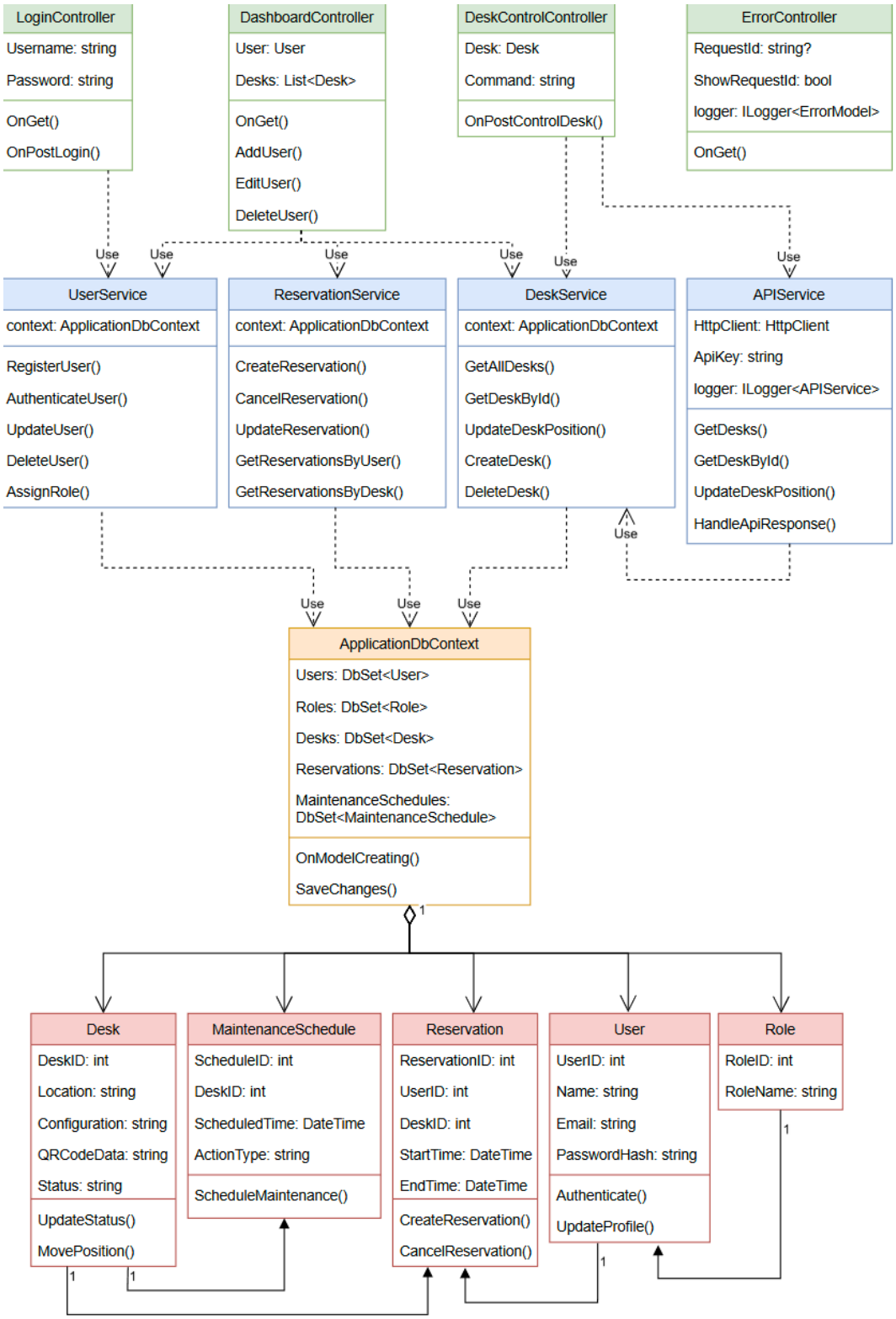
5.3.3. User Journey Map



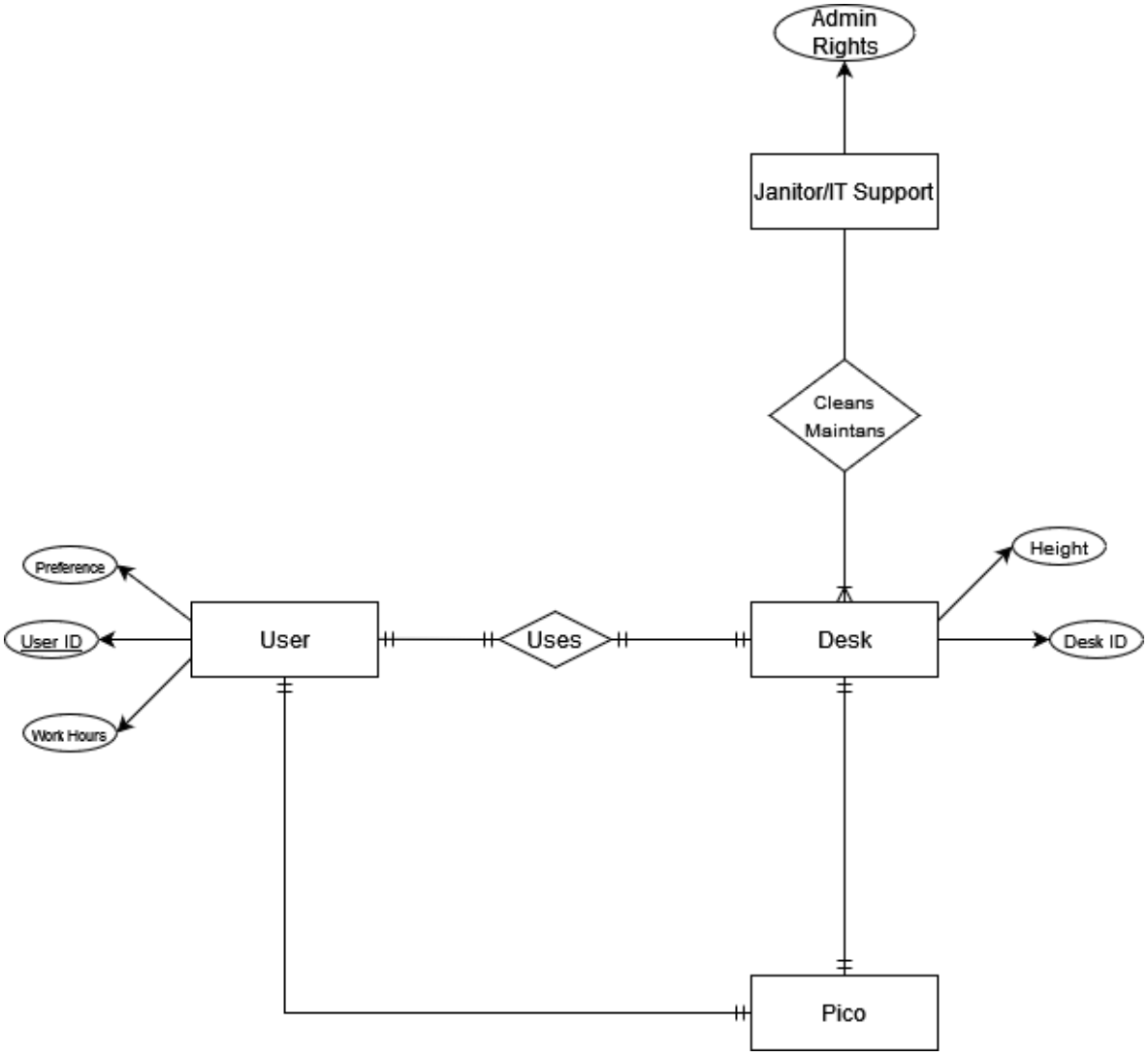
5.3.4. Flowchart



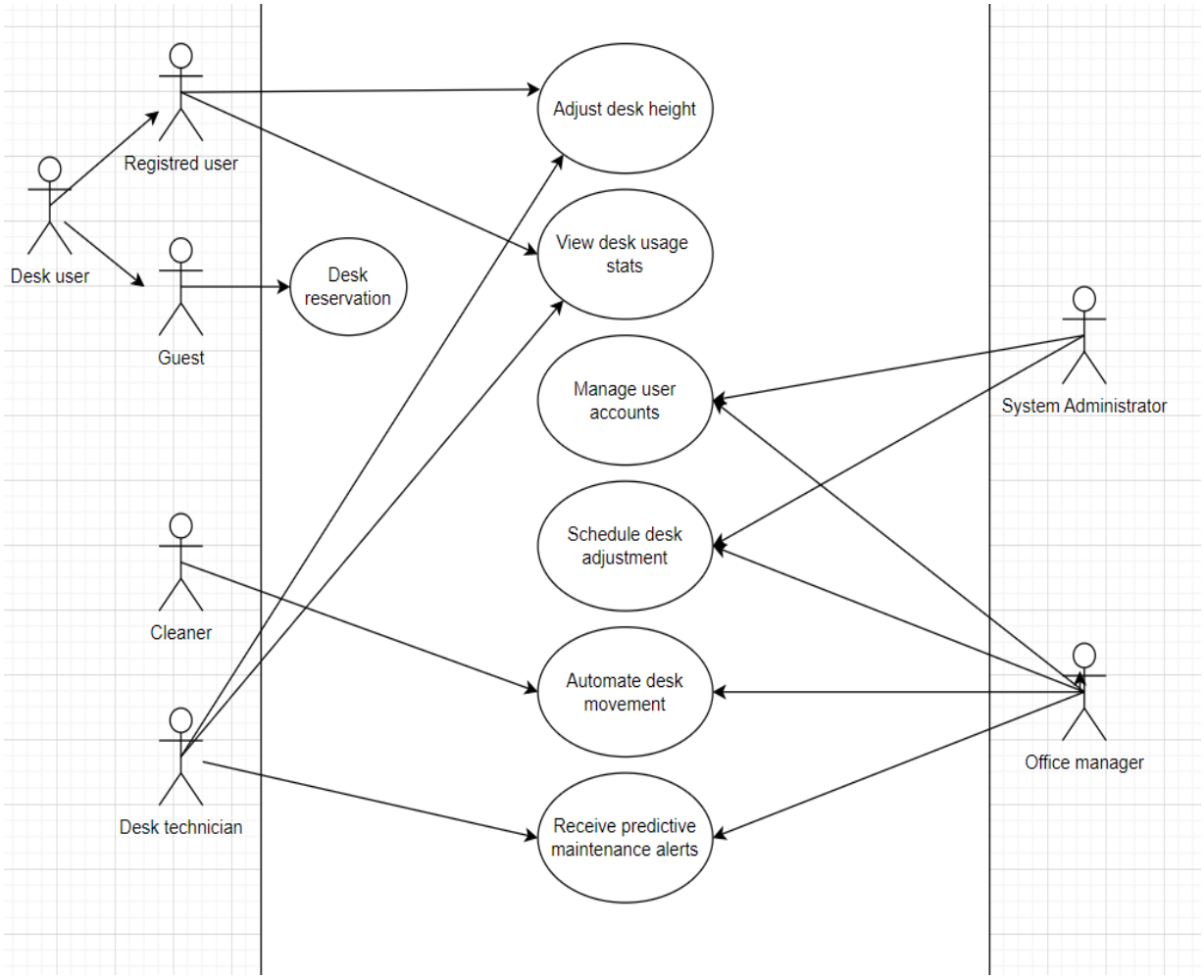
5.3.5. Initial UML Class Diagram



5.3.6 Entity Relationship Diagram



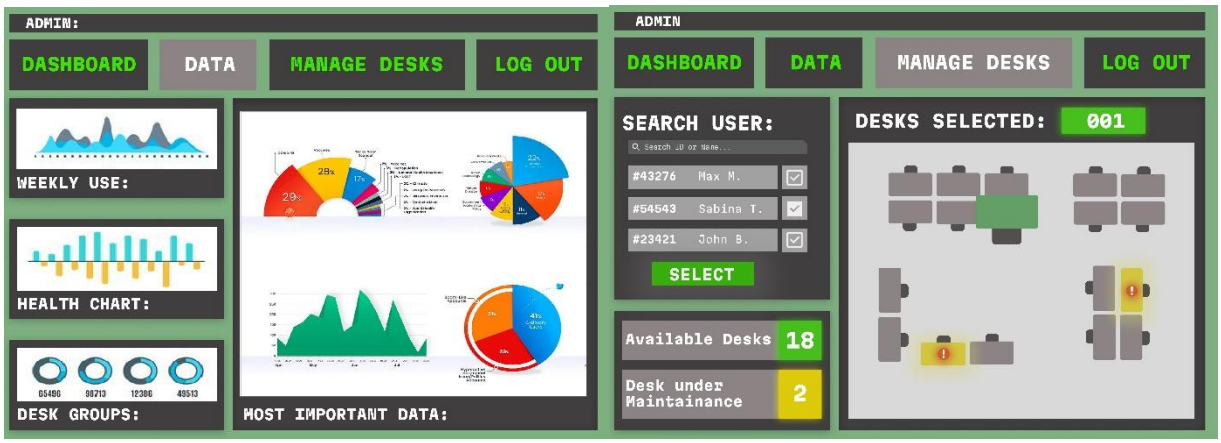
5.3.7 Use Case Diagram



6. Implementation

6.1. Prototype and interface design

Planning and brainstorming were the first steps of deciding the theme for our project. Agreeing on a theme everyone was satisfied with was a challenge. To simplify the process, we created the GUI in Figma first, which allowed for easy modification without editing the project's code. Once we had a clear vision of how the project would look, and features it would include, we began with designing the Graphical User Interface. Figma was chosen as the most reliable software for this task, enabling us to start the development of the first prototype. The prototype was of great use during the early stages of the development, so we could get a visual representation of how the website would look like when we were nearing the end. We designed several themes for the project which we afterwards narrowed down to one theme everyone was satisfied with. To be able to code the backend and have somewhat of a Graphical User Interface we used Bootstrap. We then modified the Bootstrap files to align more with theme similar with the theme in we created in Figma with slight modifications.



6.2. Styling

For the styling of the web application the group decided to go with a different approach then usually and inspire from the group name "PET" which refers to the Commodore PET computers. Therefore, the design concept was created around the visual appeal of the commodore PET in form of a minimalistic and old school style that got recreated by using the famous neon-green colour and neutral grey backgrounds in the application. That style was combined with a modern layout and adjustments to ensure a good readability to enable the best possible user-experience. The retro design and color palette were blended with modern styling to achieve the appearance of a PET graphical interface redesigned with improved readability and a clean layout, suitable for a modern web application. While implementing that design the accessibility and readability were prioritized to ensure the user can easily navigate through the application.

For the styling process the Figma prototype created in the beginning of the project served as both inspiration and guideline for the final styling of the application, since it contained the desired look, we decided on in the beginning. By using that prototype as a blueprint different group members could work on styling different aspects of the website while keeping a cohesive design, since everyone could rely on the design foundation of the prototype.

For the styling we took use of bootstrap preconfigured styles and grid system to quickly have a clear front-end of the application as a foundation, that can be tested without putting too much work into the design early in the process. Those bootstrap components could later be extended by creating and customizing different classes for different elements to achieve the final PET styled look of the application. Most of the foundational styling was done in CSS on the site.css file in the project, that injects the style configurations and styling of different html elements into every page of the website, to avoid redundancy and enable a coherent design all over the website. When there was the need to deviate from that default styling for specific elements on certain pages, inline style sections were utilized to overwrite or add extra styling attributes that only apply to the page that includes the style section. With that, a global and flexible styling was implemented to cover every page of the application that could still be overwritten in certain pages that needed extra customization or adjustment. The bootstrap system and certain style attributes were used to ensure an adaptable website that retains the layout of the design on different devices with different sizes and resolutions.

6.3. Raspberry Pi Pico W integration

Our task was to implement the lighting of the light bulb and the production of sounds when the table moved.

6.3.1. Tools and Libraries

Tools

We used the Raspberry Pi Pico Visual Studio Code extension. This is the official Visual Studio Code extension for Raspberry Pi Pico development. This extension equips you with a suite of tools designed to streamline your Pico projects using Visual Studio Code and the official Pico SDK.

Libraries

```
#include <stdio.h>           // standart for c
#include <string.h>          // string manipulation
#include "pico/stdlib.h"     // pico standart
#include "pico/cyw43_arch.h" // wifi
#include "pico/multicore.h"  //for running 2 cores
#include "lwip/tcp.h"        // basic tcp settings(auto generated)
```

6.3.2. Problems encountered

The main problem was that the lamp and buzzer had to work simultaneously, so we had to distribute the processes on 2 cores. To communicate with our application, we used the TCP server with http protocol for connection reliability.

6.3.3. Code – Algorithm

First, we initialized everything we were using.

1. For Raspberry Pi Pico W to work stdio_init_all();
2. LED + buzzer.
3. For wifi cyw43_arch_init() + enabled wifi station cyw43_arch_enable_sta_mode();

Then we launched second core multicore_launch_core1(core1_entry); which was simply waiting for command from the first core and then ran the buzzer and sent back signal to core one to continue.

```
while (true) {
        uint32_t command = multicore_fifo_pop_blocking(); // Wait for a
command from Core 0

        // Handle the command (e.g., run the buzzer)
        if (command == 1) {
            buz_run(buzMult); // Execute the buzzer task
        }

        // Signal back to Core 0 that the task is complete
        multicore_fifo_push_blocking(0);
    }
```

After this, we launched a TCP server and began to listen to Wi-Fi events. When the user clicked either up or down button (to move desk) in our web application the http request was sent to Raspberry Pi Pico W and it sent signal to second core and ran LED on the first core and buzzer on second, then waited for second core to finish and sent the response back to our application.

```
multicore_fifo_push_blocking(1); // run buzzer


    led_run(ledMult);

    // wait core 1
    multicore_fifo_pop_blocking();
```

```
// Prepare HTTP response
    const char *response =
        "HTTP/1.1 200 OK\r\n"
        "Content-Type: text/plain\r\n"
        "Content-Length: 13\r\n"
        "Connection: close\r\n"
        "\r\n"
        "Hello, World!";
```

6.4. CI/CD Pipeline

6.4.1. Tools and Libraries

Tools

We implemented CI/CD pipeline using GitHub actions and its official visual studio code extension. We also used Docker Desktop and DockerHub for the hosting of the images and were planning to develop a release pipeline using GitHub artifacts.

Libraries

For the testing part of our pipeline, we used the following libraries:

- **Microsoft.EntityFrameworkCore.InMemory:** For integration tests.
- **Moq:** For mocking the test data.
- **Xunit:** For writing Unit Tests.

The full list of the dependencies can be found in the DeskMotion.Tests.csproj file.

6.4.2. Automation testing

GitHub offers out of the box pipeline tools in the form of the GitHub Actions. We used them to trigger automatic tests after every push and pull request to main.

6.4.3. Integration with Docker Hub

During development we had planned to create a full release pipeline using Docker, Docker Hub and an on-cloud server but unfortunately due to real life issues and time constraints we were unable to finish this feature. We only managed to upload the webserver to Docker Hub.
The proposed architecture was to integrate within the pipeline GitHub actions that would build the relevant images, those being the front-end, the backend and the embedded, push them to a remote Docker Hub repository, and then deploy them from the repository to the chosen web service, for instance Azure.

6.5. Offices Plan page

We have accomplished the OfficesPlan Page, which is the dynamic interface for creating, managing, and visualizing office layouts. It offers admins the possibility of placing, rotating, and assigning desks to unique Mac-Addresses from a database. This is an interactive office space planning and resource management application, achieved with a drag-and-drop canvas, Json-based storage, and collision prevention algorithms.

6.5.1. Desks with unique MacAddress names:

The page enables administrators to assign **MacAddresses** to desks depending on data retrieved from the **DeskMetadata** table in the database. A dropdown list is loaded with all **MacAddresses** not already assigned to existing desks.

The mechanism for assigning **MacAddresses** starts with a query that retrieves all unused Mac-Addresses from the **DeskMetadata** table. These methods fill the dropdown list for user choice.

Once a **MacAddress** is selected and assigned, it is immediately removed from the list to prevent any duplication.

If the image desk is deleted afterward, the respective **MacAddress** will be restored to the dropdown menu, which provides both flexibility and accuracy in data management. The following is a part of the C# code:

```
// Fetch all MacAddresses from DeskMetadata
AvailableMacAddresses = await _context.DeskMetadata
    .Where(DeskMetadata dm => !string.IsNullOrEmpty(dm.MacAddress))
    .Select(DeskMetadata dm => dm.MacAddress)
    .Distinct()
    .ToListAsync();
```

6.5.2. Collision Detection Algorithm

Its purpose is to avoid collision among the desks in the drawing process as well as place them correctly at specified places that identify offices.

For drag and drop a new desk, its position and dimensions are compared against current desks. Collisions are prevented through geometric calculations. An alert is notified for the user if desks are placed outside the office zones, which are represented as rectangles on the canvas to simplify boundary checking.

Collision Validation code:

```
function isValidPlacement(rect) {
    return rect.x >= 0 && rect.x + rect.width <= fgCanvas.width &&
        rect.y >= 0 && rect.y + rect.height <= fgCanvas.height &&
        (isWithinBlueRect(rect, blueRect1) || isWithinBlueRect(rect, blueRect2));
}

function isWithinBlueRect(rect, blueRect) {
    return rect.x >= blueRect.x && rect.x + rect.width <= blueRect.x + blueRect.width &&
        rect.y >= blueRect.y && rect.y + rect.height <= blueRect.y + blueRect.height;
}
```

6.5.3. JSON Data Storage and Handling

The implementation uses the JSON format to store office layout information together with such details as desk position and orientation and the previously chosen **MacAddress.** Advantages of using JSON format include:

The **JSON.stringify** function makes it easy to store layout information simply because it converts it to JSON format. When the page loads, the JSON data is analysed using **JSON.parse** to rebuild the two canvas and dynamically update the list of available **MacAddresses**. The following is how to serialize and save desk data:

```
// Save form
function handleFormSubmit(event) {
        event.preventDefault();
        const form = event.target;

        // Set the hidden inputs
        document.getElementById('BgCanvasDataInput').value = bgCanvas.toDataURL();
        document.getElementById('FgCanvasDataInput').value = JSON.stringify(objects);
        document.getElementById('OfficeNameInput').value = officeNameDisplay.textContent.trim();

        // Create a FormData object from the form
        const formData = new FormData(form);

        // Send the form data using fetch
        fetch(form.action, {
            method: 'POST',
            body: formData
        }).then(response => {
            if (response.ok) {
                // Redirect to the same page to avoid form resubmission
                window.location.href = window.location.href;
            } else {
                console.error('Form submission failed:', response.statusText);
            }
        }).catch(error => {
            console.error('Form submission error:', error);
        });
    }
```

6.5.4. Canvas Redraw Algorithm

When the page loads, the algorithm processes JSON data to redraw the foreground (**fgCanvas**) and initialize the background (**bgCanvas**) by properties of the canvas context. Steps in Redrawing the Canvas:

JSON is analysed to obtain desk attributes (e.g., x, y, width, and height), where each desk is positioned using this data.

Draw desks by using **drawImage** to create a new Image object for each desk. Code already used to Canvas Redrawing:

```
// Redraw canvas
function redrawCanvas() {
    fgCtx.clearRect(0, 0, fgCanvas.width, fgCanvas.height);
    objects.forEach(obj => {
        const img = new Image();
        img.onload = () => {
            fgCtx.save();
            fgCtx.translate(obj.x + obj.width / 2, obj.y + obj.height / 2);
            fgCtx.rotate(obj.angle * Math.PI /
            fgCtx.drawImage(img, -obj.width /
            fgCtx.restore();
            if (obj.macAddress) {
                fgCtx.fillStyle = 'white';
                fgCtx.font = 'bold 13px Arial';
                fgCtx.fillText(obj.macAddress, obj.x - 8, obj.y + obj.height + 8);
            }
        };
        img.src = obj.img;
    });
    updateTotalDesks();
}
```

Merges front-end features with back-end data management, providing a good solution to the planning of office layouts. This is done through algorithms for collision detection, and the update of dropdown list data ensures functionality. The simple interface that allows administrators to plan offices in the best way.

6.6. Authentication and Authorization

6.6.1. Authentication

We have implemented the middleware Microsoft Identity into our project to handle Authentication, it does:

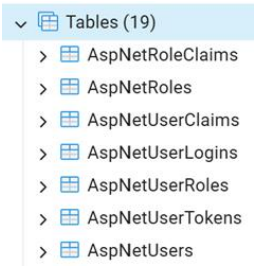
- **Security:** It has a built-in password hasher that uses pbkdf2 algorithm, Two-factor authentication, and a user lockout policy to protect against brute-force attacks. It also manages session state using secure cookies. These cookies store session data, such as authentication tokens.
- **Customizability:** Pages are scaffolded, where we build on top of it such as using a custom user and role class that inherits the identity user and role where we can add additional properties. And middleware settings set in *'program.cs'*.

Microsoft Identity uses cookies to store session data securely. These cookies are encrypted and include details about the logged-in user, ensuring tamper resistance. Sessions are managed programmatically via options configured in *'program.cs'*. For example:

- *options.LoginPath:* Redirects unauthorized users to the login page.
- *options.LogoutPath:* Handles user logouts gracefully.

- *options.AccessDeniedPath:* Redirects users without sufficient permissions to an error page.

Microsoft Identity generates a set of tables (From PGAdmin):



6.6.2. Authorization

We use the Role-Based Access Control (RBAC) model in our project, meaning access is determined based on the users' roles. Middleware policies like *'RequireAdministratorRole'* defined in *'Program.cs'* determine the access based on the user roles. Folders such as *'/Admin'* are restricted to only be used by their respective roles. And the entire site is restricted to only authorized users excluding the Login and Setup pages.

As roles are their own table *'AspNetRoles',* it means that there can be created as many as necessary without needing to modify the code, only the access is determined through code.

Role table:

Role	Actions
Admin	Full CRUD on all tables, users, roles etc.
User	Basic access: they can manage their own data and interact with content.

More roles were intended, such as Maintenance staff, but got cut due to time constraints.

6.7. Desk Details Page

The Desk Details page serves as the central control and monitoring page for desks in the DeskMotion system. This page combines functionality and user-friendly design to enhance desk management and usability for administrators and users.

6.7.1 Design and Layout

The Desk Details page is divided into distinct sections:

1. Desk Metadata:
   - MAC Address: A unique identifier for the desk's networked hardware.
   - Location: The physical placement of the desk within the office space.

- QR Code Data: Encodes desk-specific information for quick access or troubleshooting.

2. Desk Control Panel:
   - Move Up and Down Buttons: Allow users to adjust desk height in real-time.
   - Stop Button: Provides immediate halting of desk movement to ensure safety and prevent errors.
3. Desk State Monitoring:
   - Current Position (in mm): The precise height of the desk.
   - Speed (mm/s): The movement speed of the desk.
   - Status: Displays operational state (e.g., "Normal" or "Error").
   - Activations Counter: Tracks how often the desk has been adjusted.
   - Sit/Stand Counter: Logs transitions between sitting and standing modes.
4. Usage Analytics:
   - Daily Usage Pattern: A line graph visualizing desk usage trends over time.
   - Standing vs. Sitting Time: A pie chart providing insights into user habits and preferences.

6.7.2 Implementation Details

1. Frontend Development:
   - The page is implemented using ASP.NET Core Razor Pages for seamless integration.
   - Responsive design achieved through Bootstrap 5.
   - Chart.js is used for interactive and visually engaging data visualizations.
2. Backend Integration:
   - Real-Time Data: SignalR ensures live updates for desk state and control actions, maintaining low latency and high reliability.
   - Data Models:

```
60  public class DeskMetadata {
61      public Guid Id { get; set; }
62      public string MacAddress { get; set; }
63      public string Location { get; set; }
64      public string QRCodeData { get; set; }
65  }
66
67  public class State {
68      public int Position_mm { get; set; }
69      public int Speed_mms { get; set; }
70      public string Status { get; set; }
71  }
```

   - Data Access: Entity Framework Core is used for efficient data querying and management.
3. API Integration:
   - REST endpoints facilitate desk control and state retrieval, ensuring performance and scalability.
   - Asynchronous operations optimize responsiveness for users.

6.7.3 Error Handling and Optimizations

1. Error Handling:
   - Structured exception handling ensures resilience.
   - Detailed logging and user-friendly error messages improve system maintainability.
2. Performance Enhancements:
   - Efficient LINQ queries minimize database load.
   - Asynchronous data fetching reduces UI latency.
   - Optimized chart rendering enhances real-time updates.

6.7.4 Limitations and Future Enhancements

1. Advanced Analytics: Incorporating predictive models for usage trends.
2. Customizable Presets: Allowing users to save and apply preferred desk settings.
3. Mobile Optimization: Extending full functionality to mobile devices for improved accessibility.

The Desk Details page represents a core component of the DeskMotion system. It combines technical precision and usability to deliver a useful tool for desk management.

6.8. Admin Dashboard

The Admin Dashboard is the control center to view insights (via graphs) and manage site data.

6.8.1. Dashboard and Role Tools

- **Admin Index page (Pages/Admin/Index.cshtml):**
  It is the entry point for admins, providing a dashboard displaying graphs and statistics related to the desks.
- **Role Tools (Pages/Account/Manage/RoleTools.cshtml):**
  A page created for navigation to the Admin Dashboard if the user has the correct role. Users without the admin role will not get the redirect button but instead be told they have no tools.

6.8.2. CRUD Operations

The admin panel has **Create**, **Read**, **Update** and **Delete** operations for most tables in the database. We will take the example of the Reservations table to show how the CRUD logic is generally implemented.

Create

The create page (Pages/Admin/Reservations/Create.cshtml) allows the admin to add new reservations to the database. It has a form field with an input for selecting the user, the desk and specifying the start and end time of the reservation.

In the *'CreateModel',* you can see how the times are converted to **UTC** and saved into the database:

```
// Convert local time to UTC
Reservation.StartTime = StartTimeLocal.ToUniversalTime();
Reservation.EndTime = EndTimeLocal.ToUniversalTime();

context.Reservations.Add(Reservation);
await context.SaveChangesAsync();
```

Read

The Index and details pages handle reading/displaying the data.

- **Index Page (Pages/Admin/Reservations/Index.cshtml):**
  Displays a list of all Reservations.
- **Details Page (Pages/Admin/Reservations/Details.cshtml):**
  Shows more detailed information about a single reservation, including its associated User and Desk data.

Update

The **Edit** page (Pages/Admin/Reservations/Edit.cshtml) allows the admin to edit existing reservations. The time is also converted to UTC here before saving the modifications:

```
// Convert local times back to UTC
Reservation.StartTime = StartTimeLocal.ToUniversalTime();
Reservation.EndTime = EndTimeLocal.ToUniversalTime();

context.Attach(Reservation).State = EntityState.Modified;

try
{
    _ = await context.SaveChangesAsync();
```

On the **Delete** page (Pages/Admin/Reservations/Delete.cshtml), the admin can confirm they want to remove a reservation. The entity is found by its Id, and then removed from the database:

```
var reservation = await context.Reservations.FirstOrDefaultAsync(m => m.Id == id);
if (reservation != null)
{
    Reservation = reservation;
    _ = context.Reservations.Remove(Reservation);
    _ = await context.SaveChangesAsync();
}
```

Reservations is just one example of one CRUD pattern you would use with any table in your database; showing how to **Create**, **Read**, **Update**, and **Delete** table entities. The other CRUD pages (such as Desks, Roles, Users, etc.) follow a similar structure. following this structure.

6.9. Setup Page

Initially we seeded the initial admin account through *'DBInitializer.cs',* however since this wasn't very user-friendly, we introduced a Setup page so each organization can provide its own data and create the initial admin account.

6.9.1. Redirection

A small section of the *'Program.cs'* checks if the database is empty and redirects to *'/Setup':*

```
if (!dbContext.InitialData.Any() &&
    !ctx.Request.Path.StartsWithSegments("/Setup", StringComparison.OrdinalIgnoreCase))
{
    ctx.Response.Redirect("/Setup");
    return;
}
```

6.9.2. Setup

On the Setup page, the admin provides the organization name, admin email, and password. Below is a brief snippet from POST method showing the core logic:

```
var result = await userManager.CreateAsync(adminUser, AdminPassword);
if (result.Succeeded)
{
    _ = await userManager.AddToRoleAsync(adminUser, "Administrator");
}

// Save organization name
var initialData = new InitialData
{
    OrganizationName = OrganizationName
};
_ = context.InitialData.Add(initialData);
_ = await context.SaveChangesAsync();
```

Once setup is completed, the database is populated, and you no longer get redirected to *'/Setup'.* This avoids default credentials and provides a simple experience for new organizations.

6.10. Help Page

The Help Page is a central place to access frequently asked questions (FAQs) and report issues. This is inspired by GitHub Issues, utilizing the familiarity with GitHub's issues which many websites emulate.

The Help Page comprises three main components:

- **FAQs:** Find answers to common questions.
- **Report an Issue**: Users can submit reports about any issues they encounter.
- **My Reports**: Users can view and manage the issues they have reported.

6.10.1. CRUD Operations

The Create and Details functionalities within the Help Page follow the same CRUD logic as the Admin Dashboard, ensuring consistency.

- **Create**: Users can submit new issues through a form with inputs for the title, description, and optional attachments.
- **Details**: Users can view info about each issue, including comments and status updates. Admins can update the status of issues.

6.10.2. Attachments

Attachments submitted with issue reports are stored directly as byte arrays within the database. This was intended as a temporary solution; it became a permanent due to time constraints. SQL databases like PostgreSQL aren't intended to store files unlike content delivery networks (CDNs), which are essentially fileservers.

```
// Handle Attachments
if (attachments != null && attachments.Any())
{
    foreach (var file in attachments)
    {
        if (file.Length > 0)
        {
            using var memoryStream = new MemoryStream();
            await file.CopyToAsync(memoryStream);

            IssueReport.Attachments.Add(new IssueAttachment
            {
                FileName = file.FileName,
                MimeType = file.ContentType,
                Content = memoryStream.ToArray()
            });
```

6.11. API Services

The project has two API services. The RestRepository<T> handles standard HTTP requests, while DeskDataUpdater ensures desk data is regularly updated. *'Program.cs'* handles configuration.

6.11.1. RestRepository

RestRepository<T> is a generic implementation of HTTP operations. Data is retrieved from the API endpoint using the GET method. The JSON response is deserialized into type T when a GET request is sent.

- POST request to send data to an API endpoint. The hypertext is serialized into Json and added to the request body.
- PUT request that updates data at a given endpoint. It serializes the object T and sends it in the request body.
- DELETE request that removes data from the endpoint.

By making it a type T we don’t need to repeat the implementation, we can just register an instance in ‘program.cs’.

6.11.2. DeskDataUpdater

DeskDataUpdater is a background service that updates desk data every minute. It uses RestRepository<List<string>> to get a list of desk IDs from the API. For each desk ID, it ensures the desk metadata exists and retrieves the latest desk data using RestRepository<Desk> and updates the database accordingly.

6.11.3. Service Configuration

In Program.cs, API services are set up and integrated with the application's dependency injection system.

- **HttpClient Setup:**

```
// API Client
builder.Services.AddHttpClient("DeskApiClient", client =>
{
    client.BaseAddress = new Uri(apiBaseUri!);
    client.DefaultRequestHeaders.Add("Accept", "application/json");
});
```

The Base URI (Such as http://<host>/api/v2/<api_key>/) is passed to the program through the environment as an environment variable with docker compose.

- **Registering RestRepository Instances:** RestRepository<List<string>> and RestRepository<Desk>>

```
builder.Services.AddScoped<RestRepository<List<string>>>(provider =>
{
    var httpClientFactory = provider.GetRequiredService<IHttpClientFactory>();
    var httpClient = httpClientFactory.CreateClient("DeskApiClient");
    return new RestRepository<List<string>>(httpClient, jsonOptions);
});
```

- **Adding Hosted Services:**

```
builder.Services.AddHostedService<DeskDataUpdater>();
```

7. Validation

7.1. Introduction

Validation is a critical component in the software development process, ensuring that the system meets specified requirements and functions as intended. The following chapter outlines the testing and validation processes employed, including test cases, methodologies and the tools used within our solution.

7.2. Validation Objectives

The primary objectives of validation in the DeskMotion project are:

- To confirm that the system adheres to our functional and non-functional requirements.
- To ensure the integrity of data handling.
- To validate the security access solutions.

7.3. Testing Process

7.3.1. Testing Methodology

The testing process for DeskMotion includes Unit Testing, which validates individual components and functions within the ASP.NET stack, and Integration Testing, which ensures that modules interact correctly, focusing on database integration with PostgreSQL.

7.3.2. Testing Tools

- **xUnit**: For unit and integration testing within the ASP.NET framework.
- **Postman**: For the manual testing and validating RESTful APIs.
- **GitHub Actions**: For automated testing within our CI/CD pipeline.
- **pgAdmin**: For verifying PostgreSQL database operations and data integrity.

7.4. Test Cases

7.4.1. Unit Test Cases

Test Name	Unit under test	Description	Expected result	Actual result
OnPostAsync_ValidCredentials_RedirectsToReturnUrl	User Authentication	Validate login functionality with correct credentials	Login successful	Passed
OnPostAsync_InvalidCredentials_ReturnsPageWithModelError	User Authentication	Validate login functionality with incorrect credentials	Login failed; error page returned	Passed
OnPost_NoReturnUrl_RedirectsToPage	User Authentication	Check if the user gets fully logged out	User no longer logged	Passed

7.4.2. Integration Test Cases

Test Name	Unit under test	Description	Expected result	Actual result
PostAsync_SendsCorrectRequest	Rest repository	Validate that the API correctly handles the POST requests	Valid rest API response	Passed
PutAsync_SendsCorrectRequest	Rest repository	Validate that the API correctly handles the PUT requests	Valid rest API response	Passed
ExecuteAsync_UpdatesDeskData	Database	Check if the desk data gets correctly updated after adding a new desk	New desk gets correctly added	Passed
ExecuteAsync_HandlesExceptionGracefully	Database	Verify that the database handles exceptions	Exceptions gets correctly handled	Passed

7.5. Automated Testing

Automated testing in DeskMotion is implemented as part of the CI/CD pipeline using GitHub Actions. This process triggers automated unit and integration tests on every push or pull request to the main branch.

7.6. Validation Results

7.6.1. Functional Validation

- **Authentication Module**: Passed all test cases, ensuring secure and reliable user authentication.
- **Database Operations**: Validated successful CRUD operations, confirming data integrity and correctness.
- **API Functionality**: Verified accurate data retrieval and error handling.

7.6.2. CI/CD Validation

GitHub Actions reliably triggered and executed all automated test cases, significantly reducing manual intervention and ensuring consistent deployments.

7.6.3. Conclusion

The validation process for the DeskMotion project has thoroughly tested the functionality, and reliability of the system. The components passed the specified test cases, and the system met the outlined requirements. The combination of manual and automated testing ensures the solution’s coherence and its ability to deliver a seamless user experience.

8. Conclusion

8.1. Summary

8.1.1. Project Outcomes

Our Application, DeskMotion successfully solves the management of the motorized desk by implementing a desk distribution management system to address cases of desk users using it. The system included a web application, a back-end database, and embedded systems allowing for connected desk control and analytics. The main results of this project are:

- A web application, where the admin can manage the configuration of the desks available and connected to the system, monitor analyses and reports, and manage users.
- Through desk usage information analytics, users can make informed decisions about how to use desks and maintain better positions that count towards health and improving productivity.
- Automated Desk Control through the dashboard of adjusting heights of the desk after notice from the system analytics for convenient office user experience, control during maintenance, or office cleaning time.
- Implemented desk-move notification using the Raspberry Pi Pico, building a solid bridge between hardware and software systems.
- The manage office space interface can offer help with desk positioning that reflects the office layout, which helps office management optimize workspace and handle resources.
- Designed and developed solid back-end communication with servers, secure https for user connections, and the ability to enable two-factor authentication in user profiles.

8.1.2. Lessons Learned

This job project was completed based on knowledge from the previous and current semesters. As with every experience, there is benefit gained through work and practice, in addition to improving previous knowledge to become more experienced. These lessons were as follows:

- The importance of iterative development like Agile methodologies, especially when we are using Kanban, has been helpful in tackling unexpected challenges and keeping the project on track, as well as specific tasks to members of the team.
- Team collaboration daily updates through communication by Discord and weekly meeting reviews improved reliability and ensured task alignment with different abilities.
- Incorporating embedded systems with different Docker containers and the desk simulator required significant troubleshooting, particularly in syncing real-time controls with desk hardware.
- A team of people of different levels of experience in programing, more experienced people in that team would help share their experience and knowledge, providing multiple ideas and solutions.
- Using the Razor Pages framework for developing our web application gave us more experience to combine HTML, CSS, JavaScript, and C# code to develop a dynamic web page.

- Using Docker containers allows our project to easily scale applications horizontally by having multiple copies of the same application in containers that can handle a larger load easily.

8.1.3. Addressing challenges

During every project, there are some challenges that team members face. Knowing and identifying them is the way to avoid them in the future. Our project had some challenges, described in these points:

- Collaboration problem where some team members couldn't attend the weekly meeting to keep updating, find out problem solutions, and coordinate jobs with other team members, the result created a gap between knowledge and action.
- Differences of opinion can create new ideas and innovative job ways, but sometimes that causes sensitivities between team members that lead to delays in the workflow.
- Using Kanban inside GitHub was a good idea, but it was not enough to motivate the team members to keep moving in steady steps, as was our experience with SCREAM.
- Without deployment, our project is missing out on continued deployment, scalability, and real-world entertainment testing. Local Docker doesn't provide the same reliability, performance, and efficiency in production environments.

8.2. Future Work

After the initial version is released, each application is exposed to testing, which exposes the areas in which they can develop and appear to be better performing overall. These are the main aspects that need to be upgraded in the next release:

- Improved Graphical Interfaces to become more user-friendly without complexity in count of different tabs and short them in small number of tabs. In addition, keeping colour more popular will give a good user experience.
- Add AJAX to our web application pages, which brings several benefits, such as partial page updates for specific sections of the web page without refreshing the whole page. That means reducing data transfer between the server and the client by sending necessary data and faster response times.
- A full release pipeline with Docker Hub integration and cloud server hosting. This would require harder separation between development build and release build of the system.

Overall, the project met its established goals and objectives. Implementation of key deliverables including a web app, back-end integration, and embedded system functionality. But there's still scope for improvement in future, especially in advanced analytics and scalability. This working setup is a scalable model for more advanced iterations aimed at improving onscreen ergonomics and line efficiency.

University of Southern Denmark

Phone: +45 6550 1000
sdu@sdu.dk
www.sdu.dk