inmeta

Final Report

A Bachelor's Thesis in Computer Science and Technology (2023)

DSA - Digital Sustainability Agent
Analyzing the carbon footprint of Microsoft 365 services

OSLOMET

Group 42

Mikolaj Krzysztof Baran

Jon Presthus

Meron Berhe Kidane

Institutt for Informasjonsteknologi

Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo

Besøksadresse: Holbergs plass, Oslo

Telefon: 22 45 32 00

Project nr.
XX

AVAILABILITY
Open

BACHELOR PROJECT

MAIN PROJECT TITLE	DATE
Digital Sustainability Agent - Analyzing the carbon footprint of Microsoft 365 services	26.05.2023
	PAGE COUNT XXX
PROJECT MEMBERS Mikolaj Krzysztof Baran (s354608) Jon Presthus (s350188) Meron Berhe Kidane (s354594)	INTERNAL SUPERVISOR Raju Shrestha

CLIENT	CONTACT PERSON
Inmeta Consulting AS	Mario Vaz Henriques

SUMMARY
DSA - Digital Sustainability Agent is a Microsoft Teams application developed by a group of three students from OsloMet in cooperation with Inmeta Consulting AS. It gives Users within a Microsoft tenant the ability to get an overview over carbon emissions produced through the use of Microsoft 365 services such as Outlook, OneDrive and SharePoint.

The App will provide users with data and visuals that give the user insight as to how much CO2 is produced by Microsoft 365 services over the course of a week. And give actionable suggestions as to how the user might change behavior to decrease their own carbon production.

3 KEYWORDS
REACT

.NET

Microsoft Graph

Preface

This document represents our bachelor's thesis in Applied Computer Technology, which was completed during the spring of 2023 at OsloMet Metropolitan University. Our finished product is a web application in Microsoft Teams created in collaboration with Inmeta Consulting AS. We hope that this application will help people and companies become more aware of their own carbon footprint and lead to a reduction in the production of CO2 for both people and companies.

Acknowledgements

We want to thank our supervisor at Inmeta, Mario Vaz Henriques, for guiding us through the development of the application and helping us when we were having problems with the project. As well, we want to thank Abiel, an employee at Inmeta, for being a great programmer who was willing to take time from work to give us advice and teach us various methods of how we could solve our problems. Finally, we want to thank Raju Shrestha, our internal supervisor from OsloMet, who gave us useful feedback about how we would plan and conduct the project.

Summary

This report documents the process behind developing our product, and the final product that came as a result of following the process we developed for this project. The process documentation covers all the tools, requirement specifications, challenges, progress plans and planning that was conducted to be able to produce the final product. The product documentation covers the final product and how each component works together to give the final product the desired functionalities documented in the requirement specifications.

Abbreviations

DSA	Digital Sustainability Agent
SQL	Structured Query Language
UI	User Interface
URL	Uniform Resource Locator
API	Application Programming Interface
UX	User Experience
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
ACID	Atomicity Consistency Isolation Durability
NPM	Node Package Manager
CO2	Carbon Dioxide
GDPR	General Data Protection Regulation
ORM	Object Relational Mapping
kWh	Kilowatt hours
CSV	Comma-separated Values
DOM	Document Object Model

Table of Contents

Chapter 1 - Introduction

1.1 Background

In recent years, sustainability has become a growing concern among people, companies, and countries due to climate change. We are living in a time where we are experiencing drastic changes to our environment due to the rise in global temperatures and the consequences that come with them. The cause of this can largely be attributed to the amount of CO2 humanity produces. Every year we produce more CO2 that ends up in the earth's atmosphere, where it strengthens the greenhouse effect and leaves our planet in more extreme climate conditions.

One aspect that many people are still unaware of is the impact of CO2 produced by digital means. It is estimated that the internet produces about 3.7% of total global CO2 emissions.(Kilgore, 2023)

As the general public slowly grows more aware of these emissions, some individuals, companies, and governments have, as a reaction, started to become more concerned about their own carbon emissions and want to take steps to reduce them.

This is the main intent behind Inmeta giving us this project. It will help users, departments, and companies have a greater overview of their own carbon emissions, in addition to increasing awareness surrounding the problem.

1.2 Problem Statement

Global data production has been growing at an ever-increasing rate. And with more data comes more carbon production. This data includes email, files, video, audio, etc. (Duarte, 2023)

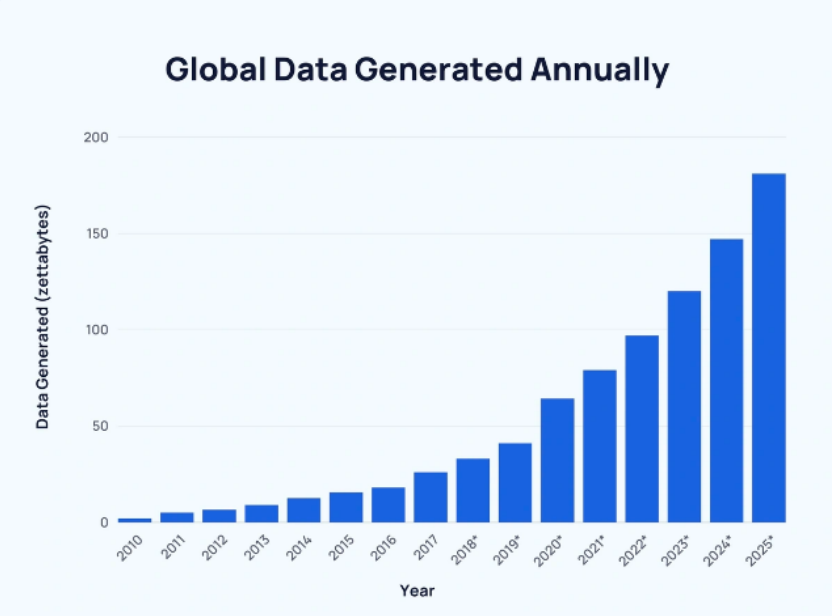


Fig 1.1 Global data generated each year (in Zettabytes)(Durate, 2023)

The aim behind this project is to create an application that will give users and companies within Microsoft Tenants an overview over their own digital data and corresponding impact on the environment, within Microsoft 365 services. And at the same time, provide them with suggestions on how to reduce this footprint.

1.2.1 Objective

Our proposed solution is an application for Microsoft Teams, the Digital Sustainability Agent, that gathers data and statistics from various Microsoft 365 services, like OneDrive, Outlook, SharePoint, and Teams, through the use of the Microsoft Graph API, and provide the users with data usage and associated energy consumption in the form of CO2 emissions. The core concept of the solution is a single-page application with multiple tabs that cleanly display the user's, their department's, and the whole company's CO2 production, respectively.

We envision the sustainability agent calculating and displaying what can be done to lessen the carbon footprint and achieve any objectives established by the user or their organization, for example, "You have 30 files that haven't been used since

2021; delete them?" or “You have sent 50 emails to johnsmith@example.com in the last week; can you try to send fewer?”.

If time allows us to develop the solution beyond being a way to present the data to the end-user, the application will also let the user ask a chatbot for more specific suggestions or how they can go about utilizing those solutions more effectively. We also want to develop a small pixel art game that tracks the progress of each employee in a company towards reaching their sustainability goal. Grouping the users by departments with the same colors, will give them a better overview of how their own department is progressing, making this pixel art game a fun motivator. We believe this pixel art game will promote healthy cooperation and competition between peers.

1.2.2 Intended use

Our applications intended use is for companies to have a greater overview over Files, Teams, emails and SharePoint sites within their organization. Older or obsolete projects and files can often be forgotten and left to occupy space within the company's systems. What we want our application to be used for is to make the users aware of how much CO2 is produced through their usage of Microsoft 365 services and suggest steps for the user to take to reduce CO2 production.

1.2.3 Product scope

What we hope to benefit with our application is a reduction in CO2 emissions of the company in a way that is not detrimental to the daily functions of the company. The application will not delete or set limitations on its own, but rather give suggestions.

1.3 Group presentation

We are a group of three students from OsloMet University that have worked together since we started studying “Anvendt Datateknologi” (Applied Computer Science) in August 2020. Our names are Jon Presthus, Meron Berhe Kidane, and Mikolaj Krzysztof Baran.

Our client for this project is Inmeta Consulting AS, a technology consulting company, with Mario Miguel Henriques as our external supervisor from Inmeta Consulting and Raju Shrestha as our internal supervisor from OsloMet University.

Meron Berhe Kidane, s354594

Meron is an applied computer science student from OsloMet, He was born and raised in Eritrea and has been living in Mysen, Norway, for the last seven years. He studied for 1 year as an accountant before leaving Eritrea and studied as a health worker in the upper secondary school in Norway, before switching to IT.

He has knowledge of frontend and backend development. This includes Java, C#, .Net SQL, Javascript, HTML, CSS, Spring Boot (the Java framework), Android Studio, the development of prototypes and principles related to user interface design, and now learning more C# and.Net.

Mikolaj Krzysztof Baran, s354608

Mikolaj is a 21-year-old Applied computer science student at OsloMet. He was born in Poland and now lives in Drammen, Norway, where he finished primary and upper secondary education within media and communication, and found an appreciation for developing websites and digital products.

Mikolaj has studied both front-end and back-end software development, and is most experienced with Java, HTML, CSS, JavaScript and C#. He has experience building the back-end for a small start-up company in university, and has gained a preference towards learning by doing.

Jon Presthus, s350188

Jon Preshus is 22 years old and is studying Applied computer science at OsloMet University. He was born in Oslo and has lived his entire life in the city. He grew up in

the district of “Søndre Nordstrand” and went to Bjørnholt High school where he chose the “SSØ” pathway. During High school he took a course in Information technology where he got a short introduction to web development and SQL-databases.

During his time at OsloMet he has acquired skills within both front-end and back-end development, as well as database management. His skill set includes: HTML, CSS, Javascript, Java, Android Studio, TypeScript, Angular, SQL, .NET C# and Principles related to website design.

Chapter 2 Process Documentation

The process documentation covers how we conducted the development of the product. We first present how we prepared for the project, then further describe our development process, requirement specifications, progress plan, tool usage and challenges that were part of the process.

2.1 Preparation

During the winter of 2022, we had a meeting with our external supervisor from Inmeta, where we discussed the general tool usage and structure of the project. From the discussion we were given requirements as to which tools and services we will use to accomplish our goals. These requirements included the use of C# and the .NET framework to form the backend of the project, which will be used for retrieving data and calculating the amount of CO2 emission that are produced by the users.

In the front-end, we will use React in combination with Fluent UI to form the UI of the web-application. This is because React is a standard used by Microsoft to create their own apps and Fluent UI is used as a standard template in Microsoft Teams.

To visualize the data, we intend to use graphs, pie charts and bar charts to present our collected metrics, as well as the calculated values of carbon emissions to the end user. Our idea is to use Microsoft Power BI to map the processed data from the Azure database to various graphs, and export to our end-user Teams application.

2.2 Privacy and GDPR

GDPR is a legislation ordered by the EU that gives certain rights to Individuals in regards to their personal information (PrivacyPolicies, 2022).
In the early stages of the project we were worried that collecting data from users within the company would break the rules set in the legislation. However, since we only collect statistical data that exists within the company and no other personal data besides company email. We concluded that we would not break the law considering

the legislation does not extend company data only personal information. (EU Commission, 2023)

2.3 Pre-Project Report

We started to work with a pre-project report on 18th January, which involved investigations of the general structure of the project. We worked with the progress-plan for the rest of the project, which includes everything about design, desired functionality, documentation, reporting, meetings, testing, and logging implementation.

2.4 Development Process

During our initial planning, we discussed how the development process should be conducted. We know it’s not easy to schedule a clear development process, which requires a lot of technical knowledge and experience to get right. For that reason we chose an agile development process for our project.

Agile development is a development process that revolves around principles written in “The agile Manifesto”. It is practiced in turbulent environments that are prone to changes in requirements and provides a method of dealing with eventual changes in the project.(Agile Alliance, 2023) We therefore chose Agile as our development process as the project we were given was prone to change depending on the wants of our supervisors. Agile provides frameworks and practices which include scrum, kanban, sprints, stand-ups and more which we will all use throughout the development of the product as they help keep the process organized and structured.

According to Microsoft, “agile development is a term that’s used to describe iterative software development. Iterative software development shortens the DevOps life cycle by completing work in short increments, usually called sprints. Sprints are typically one of four weeks long. Agile development is often contrasted with

traditional or waterfall development, which plans larger projects up front and completes them according to the plan” (Microsoft, 2023).

2.4.1 Meeting

We were fortunate to have a dedicated place to work on our project at the Inmeta company location, so we met 3-4 days a week in person to work together at the Inmeta office, where we could utilize group rooms as needed, and the remaining day(s) we worked independently from home or at the office and documented our work so we could all stay up to date. In addition, we had bi-weekly meetings with our internal supervisor where we were given feedback and guidance as to how we would conduct our project, check that our progress plan was going as expected and make possible adjustments as to what we needed to prioritize in any given sprint.

After each meeting with our internal supervisor, we routinely participated in group discussions to strategize how best to apply the advice we received. When further refinement was needed, we sought additional guidance from our external supervisor.

2.4.2 User Stories

The requirements of software development always change. “To handle these changes, agile methods do not have separate requirements for engineering activity." Rather, they integrate requirement elicitation with development. To make this easier, the idea of *user stories* was developed, where a user story is a scenario of use that might be experienced by a system user” (Sommerville, I. 2015, s. 79).

2.4.3 Scrum

“The scrum methodology is an agile framework that helps organizations facilitate team collaboration and simplifies complex projects by indicating what the scrum team needs to do, how they should do the tasks, but not in detail” (Jim Campbell, 2020).

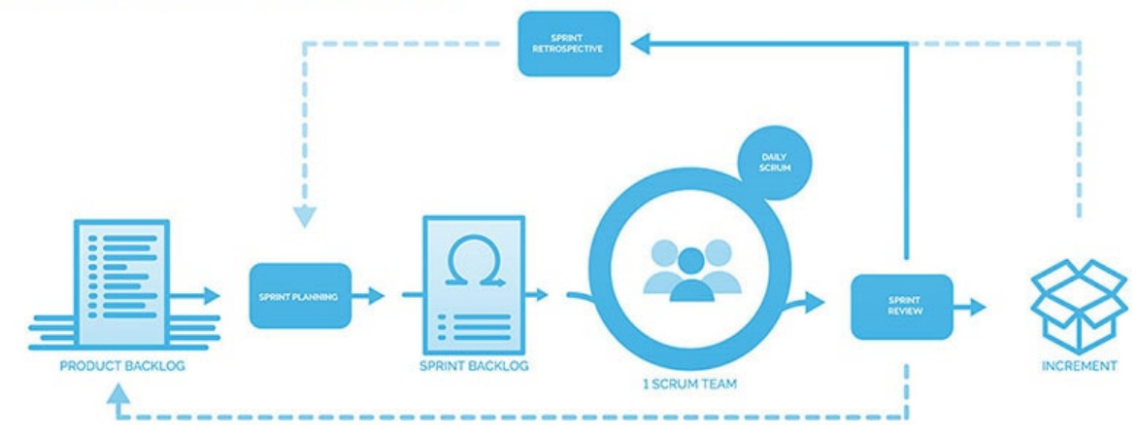


Fig 2.1 - The Scrum framework

As you see in the iterative scrum lifecycle in the figure above, which means the sprints were completed within the fixed period of time.

2.4.3.1 The Scrum terms

Scrum term	Principles
Product backlog	“This is a list of “to do” items that the Scrum team must tackle." They may be feature definitions for the software, software requirements, user stories, or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation” (Sommerville, I. 2015, s. 85)..
ScrumMaster	“The ScrumMaster is responsible for ensuring that the scrum process is followed and guides the team in the effective use of scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference” (Sommerville, I. 2015, s. 85).

Scrum	A daily meeting of the scrum team that reviews progress and prioritizes work to be done that day (Sommerville, I. 2015, s. 85)..
Sprint	In a development iteration, sprints are usually 2 to 4 weeks long (Sommerville, I. 2015, s. 85).

2.4.3.2 Stand up meeting

Before we started working on the project, we had a short stand-up meeting where we discussed what had been done since last time, what needs to be done moving forward, as well as any obstacles.

2.4.3.3 Sprint plan

We scheduled each sprint to last two weeks. We then created four epics that included features and work items for us to divide amongst the sprints. Each epic will cover two sprints meaning that we have a total of 8 sprints that cover 16 weeks of development. At the end of every sprint we had a sprint review with our supervisors from Inmeta and other students from OsloMet. Our agenda for sprint review is as listed below:

- Demo: we demonstrate what we have achieved during the sprint and how it will interact with the rest of our system. As we demonstrate the demo, customers are given the opportunity to provide input and feedback both with features and design.
- Assistance: here we have the chance to get help from our supervisor and other staff with some technical obstacles.

- Retrospect: in this section, we go through what the team did well, what didn't go as planned, and what can be improved in the next sprint.
- Planning: we show what the plan for the next sprint is and how it potentially deviates from our previous plan, and we also discuss if there is anything we need to change further about our plan.

2.5 Requirement Specification

The objective, intended use, and product scope can be found in Chapter 1 Introduction.

2.5.1 Preface

This section of the process documentation is dedicated to the requirements specification, which aims to give the reader an overview of the contents and functionality of the application. This section was co-developed jointly with Inmeta, with the goal of defining requirements for the application to fulfill the needs of the user to a satisfactory level.

As mentioned in the introduction, we are building a digital sustainability agent that tracks users' and their departments' carbon emissions from Microsoft 365 services. This solution is needed, as sustainability and carbon footprints become an ever increasingly prevalent issue in our society and new, stricter laws and regulations force organizations and companies to be more aware of the effect they are having on the environment. The DSA is built specifically for such companies, tenants as we call them, to fulfill their needs.

Our solution can be considered a brand new product on the market. The only other similar software available is Microsoft's own *Microsoft Sustainability Manager* service, which we found to be inaccessible, complex, extensive, modular and requiring too much syntax- and software knowledge of the end user to get any actionable results. As instructed by our external supervisor, that service doesn't fulfill the same needs (detailed below in chapter User needs) of readability and accessibility of our intended solution.

2.5.2 Guidance for reader

The requirements specification describes the common requirements that were defined in the planning stages of the project, and then potentially changed, iterated upon and elaborated on further in later stages of development. To give an clean, accessible overview of our requirement specification, we have separated them into three parts: the functional requirements, the non-functional requirements, and optional requirements that don’t necessarily fall into either category and can consist of components, additional features or alternative solutions that could be implemented if our group has enough time or resources at the end of development and fulfilling all the basic user needs of our program.

The optional requirements can also work as a basis for further iterations of the software that can be updated later by us or anyone at Inmeta who would take over the development after the end of the bachelorproject period.

2.5.3 Overall Description

2.5.3.1 User needs

The primary users our solution is aimed at are corporate employees. These are busy people that want instant and easy to read results. To fulfill their needs, our solution needs to be reliable, consistent, running fast calculations, and presenting results in an accessible way that is comprehensible and actionable to the end user.

The secondary users for our solution are company executives and administrators. While employees must be able to see data for themselves and their department, authorized administrators supervise the whole tenant, and must be able to look over each employee and department registered under them.

2.5.3.2 Assumptions and dependencies

Our assumption is that our end-users and customers will already be working primarily, and well-integrated, with Microsoft 365 services such as OneDrive,

Outlook, Sharepoint and Teams, since this solution relies on gathering data from these apps for the entire tenant to provide and present actionable statistics, charts and metrics.

In line with our assumption, this product solution is dependent on external software, namely Microsoft Teams, as the application will be built to run natively on that online service, and will in the future be commercially available on the Teams Store.

2.5.4 Functional Requirements

The backend of the software will be created through the use of Microsoft Graph and Microsoft Azure. The backend code will be hosted in Microsoft Azure and through the use of Azure Functions, will be executed once a week and collect data from various Microsoft 365 services. This data is then added to a database, and we will have a user interface that presents that data to the user.

The user's interaction with the application remains minimal, as the focus of the application is to present data to the user and give feedback as to what can be done differently to reduce their carbon footprint.

2.5.4.1 Common Functional Requirements for DSA

Functional requirements for Users		
No	Action	Levels
1	The users should be able to see the total CO2 emissions they have produced per week.	Medium
2	The DSA application should collect the old files and new files.	High
3	The methods used to fetch data will be executed in Microsoft Azure through the use of Azure Functions	High
4	Data Collected through Microsoft Graph will be stored in a Database located in Microsoft Azure	High

5	The DSA apps will Display my department's overall status on carbon footprint.	Medium
6	Users will have oversight over how much Co2 is produced by each department and company as a whole	High

2.5.5 Non-Functional Requirements

As described in the Functional Requirements section, the interaction of the user with the application is minimal, and we aim to make that interaction process smooth, clean, easy to digest and accessible, so that all users can get their primary needs of the application fulfilled. This includes things like making all components readable, the navigation on the Teams pages intuitive, and response times fast.

The following specifications are product-specific and project-specific, as external requirements like long-term scalability or support for non-department structured companies fall outside the scope and relevance of the project. These requirements don’t have levels of orders of importance, as it can be difficult to calculate and determine what requires the most work or reaches a level that is satisfying to the end user, so the list of non-functional requirements are simply marked with an “OK” when implemented, indicating they are qualities of the product that meet our demand.

2.5.5.1 Common Non-Functional Requirements for DSA

Non-functional requirements for Users		
No	Action	Level
1	The front-end application should be readable and split information into different tabs	OK

2.5.6 Optional Requirements

Early in the planning stages, our group, together with our supervisor, brainstormed ideas for the functions and features of the product application that aren’t the main core features of the software, and were ideas that the group could follow up on and

implement if we still have enough time and resources at the end of the project period. These requirements are primarily functional and are based on wishes for future functionality, shared during our dialogue with the internal supervisor.

As with non-functional requirements, a level of “OK” in the following figure indicates that an optional requirement feature has been implemented into the solution.

Optional requirements		
No	Action	Level
1	The application should feature a “pixel art game” that lets employees and departments partake in friendly cooperation and competition to reach the company emission goals in a fun way	
2	The application should include an AI chat bot that can give suggestions on what the user can do to help reduce their carbon emissions within Microsoft 365 services.	
3	Only the authorized administrator(s) should be able to view the company level emissions	OK
4	The authorized administrators(s) should be able to check the carbon emissions of each employee/department	OK

2.5.7 UML Diagrams

To get a general idea of how the individual parts of our solution would together we created some Sequence Diagrams to visualize how the parts of our solution would interact.

The sequence diagram provided below shows the step-by-step sequence of actions that take place once a user opens the Teams application. By following this sequence, users may efficiently use the Teams program to engage in communication,

cooperation, and teamwork within their various work contexts.

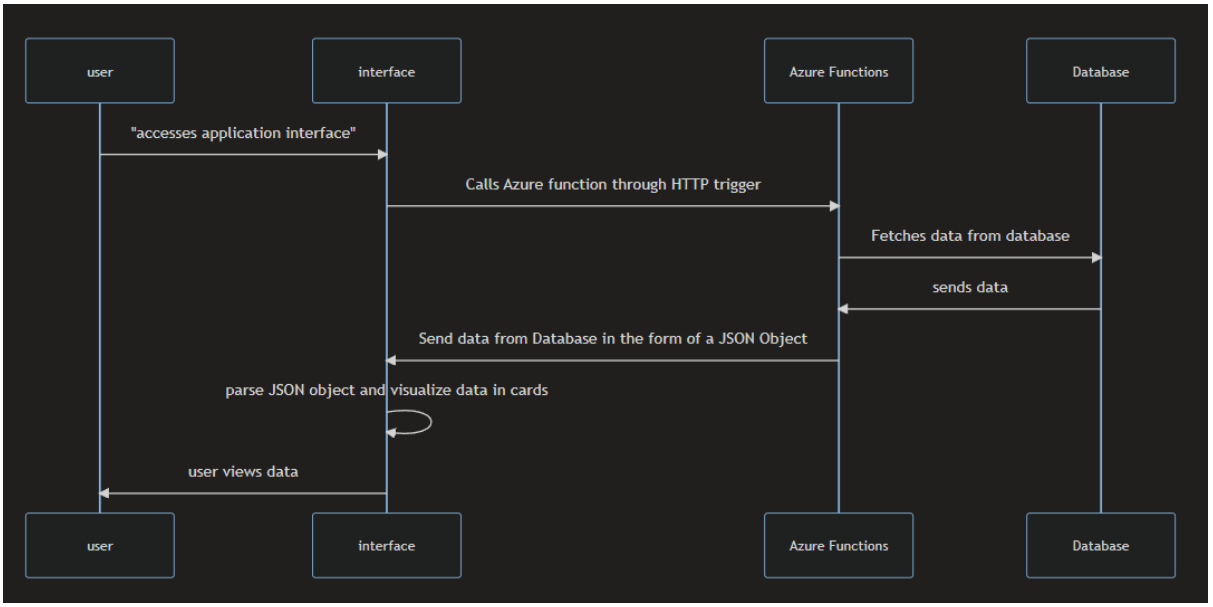


Fig 2.2 - Sequence Diagram User opens the application

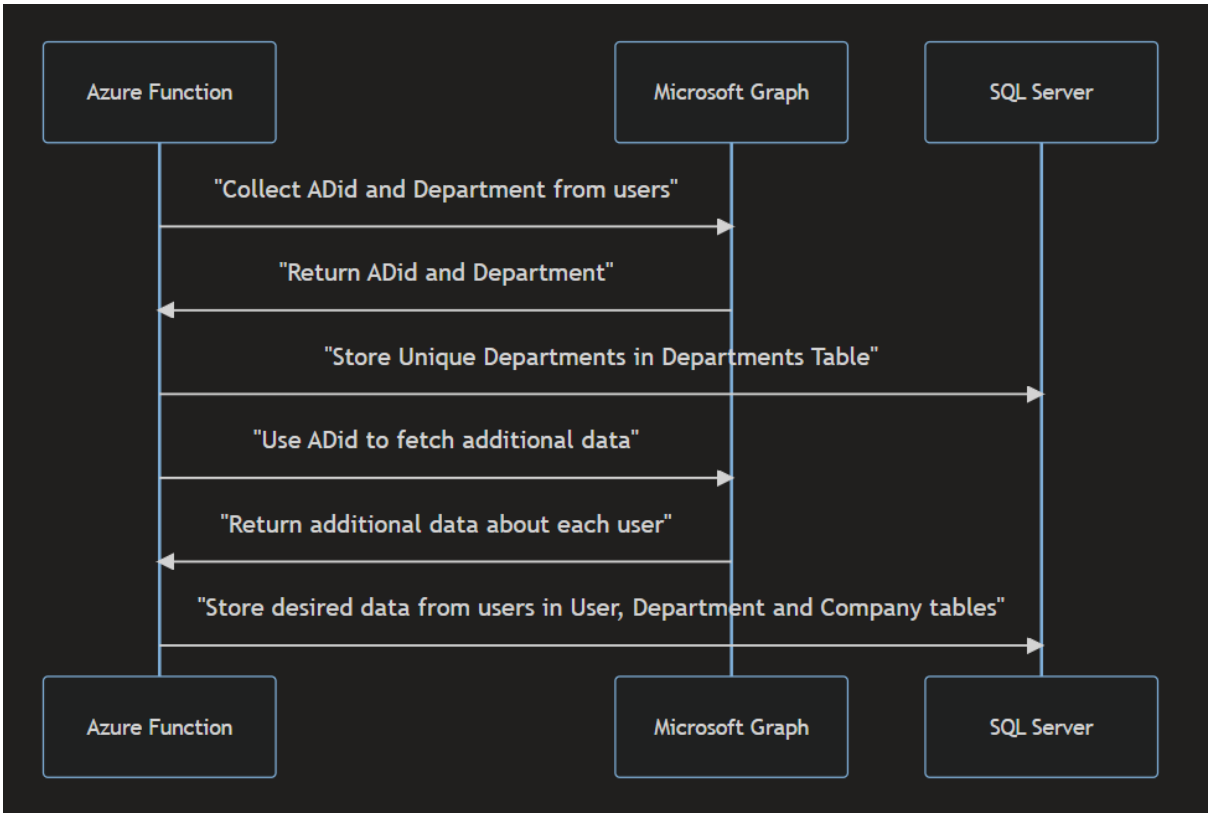


Fig 2.3 - Sequence Diagram- Data collection

2.6 Work and progress plan

Our Bachelor project is scheduled for 19 weeks, from the beginning of January until the end of May 2023, with our tasks split into 8 sprints (each sprint with 2 weeks duration). The initial two sprints will revolve around planning and the gathering of data from Microsoft Graph as well as creating the Teams application to be used by our personal assistant as well as data visualization. The second set of sprints will focus on storage, organizing and processing the data. Following that, the third set of sprints will focus on creating algorithms that calculate the amount of CO2 produced by the user, department and company. The last set of sprints will revolve around developing the front end of the application through the use of React, Teams Toolkit and FluentUI.

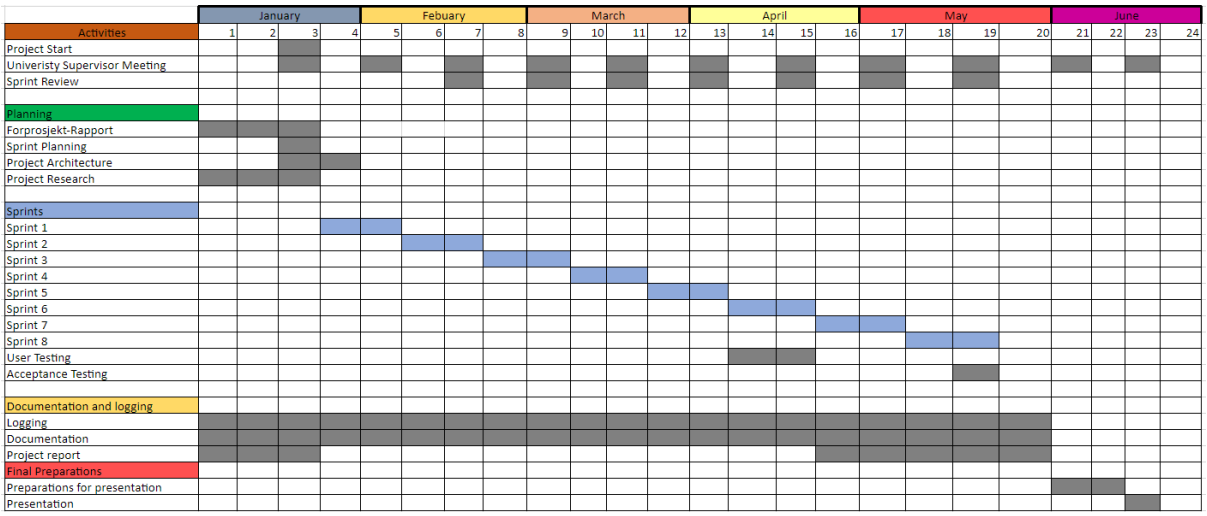


Fig 2.4 - work and progress plan

2.6.1 “Pre-sprint planning” period

In the beginning, the goal was to establish an overall system architecture and carry out thorough research on the project. This includes the choice of tools, programming language, planning, distribution of tasks in sprints and obtaining access to the organization etc.

We had a meeting with our internal supervisor from OsloMet to discuss our project and the content of the preliminary project report. We were informed that we would have bi-weekly meetings with the internal supervisor to review the project's progress and address any updates or challenges we encountered.

2.6.2 “Sprint 1” 23st January - 3rd February

On sprint one, we distributed the backlog and sprint with epics, features, user-stories and tasks. Further, we created a Team agreement on overall architecture and “Wiki” for documentation for every sprints and other relevant documents that might be needed for the report.

We had a quick review of how to set up an Azure project, and start an Azure Functions project in Visual Studio, and publish to the cloud service. We structured our DevOps to have three "main" Epics, consisting of 1. Teams app, 2. Backend, and 3. Database. We then linked Features, User Stories and Tasks under each area. We worked on getting a better understanding of the Microsoft Graph and API. We have used .NET C# and created an Azure Functions project that we use to run the API class.

2.6.3 “Sprint 2” 6th February - 17th February

Our goals for this sprint were to learn Microsoft Graph and use it for the Azure Function to collect data from tenant users’ OneDrive, SharePoint, Outlook and Teams.

Sprint two was the beginning of our introduction to Microsoft Graph. Our goal for the sprint was to be able to collect data specified in our work items to later be inserted into the database. Microsoft Graph was something we had never used before and had little understanding of, we therefore spent the early stages of the sprint getting a greater understanding of how Microsoft Graph works and in what ways we could collect desired data. We encountered some problems with collecting data related to activities in Microsoft Teams as it has a more convoluted way of collecting data than the other Microsoft 365 services.

We then created our Azure Functions where we could automatically collect wanted data in Graph through the use of loops and user IDs. We realized that the process of collecting data from all of the users in the tenant was a time-consuming process as the Azure functions would sometimes take minutes to collect all the wanted data. This is something we would improve upon later.

2.6.4 “Sprint 3” 20th February - 3rd March

Our goals for this sprint were to set up an SQL database and save processed data to it. At the start of the third sprint, we had finally gotten our final Azure Function, for Microsoft Teams, working and could push the entire backend code to the git repository for the first time. This sprint we also started working on the database through Microsoft SSMS (SQL Server Management Studio) and Azure Data Studio, and wrote the relevant C# code to initialize our database, that we then got running on Azure DevOps, connected to it, and built quick example tables.

Much of this sprint was spent on developing code so we could send the data from all our Azure Functions to the database, as well as creating all the dedicated User, Department and Company tables in the database. We achieved good success with OneDrive and Outlook, but Sharepoint and Teams proved more difficult until we could get actionable data and statistics out of our Functions.

2.6.5 “Sprint 4” 6th March - 17th March

In sprint four, we focused on implementing ORM (Object Relational Mapping) to our Azure Functions to be used for our database. This implementation allowed us to map all our retrieved data from Graph API reports to objects that would be sent as entries to the database tables later. OneDrive, Outlook and Teams functions were quickly set up to work as intended, correctly registering to the database with, but Sharepoint continued to prove difficult, as confusions over file and site ownership made it difficult to determine who individual elements should be related to, and we couldn’t find the correct keyphrases to use in our Graph API call. Sharepoint does work to gather data for the entire tenant when we don’t have to attach an owner to list each site and

file, it’s possible we may move on to only saving and calculating Sharepoint data for only the Company table.

It was also during this sprint that we started to visualize our retrieved database data in Power BI, and at the end of the sprint, we finally merged all our backend codebase branches together into one Staging branch that we will use as a basis for refining and implementing new code and branches going forward.

2.6.6 “Sprint 5” 20th March - 31st March

Our goals for this sprint were to define and create our calculations that measure the CO2 produced by Microsoft 365 services, store those calculated CO2 values in the database for later presentation and use within the Teams application, and to make changes in the Functions that collect the data.

During this sprint we had to make some changes to the data collection Azure Functions created in previous sprints. Our external supervisor showed us a more efficient and less time consuming way of collecting the wanted data meaning that we had to rework all of our Azure Functions. Luckily this was only a small setback as we were quickly able to integrate the necessary changes.

The latter part of the sprint revolved around creating algorithms that calculate the amount of CO2 produced by each User. What we first had to do was to find sources that give us a general idea of how much CO2 is produced by using the Microsoft 365 Services. We then used those sources to produce our calculations. We created an additional four Azure Functions that work in a very similar way to the previous Azure Functions that collect data. We query the database and fetch every user, then do calculations on the collected values and then insert the calculated values back to the database.

2.6.7 “Sprint 6” 3rd April - 14th April

We spent the majority of sprint number six working on retrieving the data from the Azure database through Azure Functions that return a JSON object that can be used

to visualize the data in the Teams application, as well as developing the Teams app itself. This sprint, we started working with Fluent UI to create “cards” that neatly present the retrieved data for the User, and created corresponding GetDeptData and GetCompData Azure Functions to retrieve data from the Department and Company tables, respectively.
Later in the sprint, we worked on fixing CORS (Cross-Origin Resource Sharing) in the backend, as our method of retrieving the User and Department data relied on it, and so that our data could correctly be read from the JSON and presented on the application. We followed official instructions from CodeSandbox to show our PowerBI report directly in React, and most importantly, we have started using TeamsUserCredentials to query the signed in Teams user’s email to be used in the API call to the Azure Functions to find the matching data in the database and present it to said user in their Teams application.

2.6.8 “Sprint 7” 17th April - 28th April

Our goals for this sprint were to implement FluentUI cards and graphs in the Teams application, implement proper authorization for administrators, and polish the UI and UX of the application.

During this sprint we were focused on creating the User interface through the use of React and with help of the library FluentUI. Fluent UI was chosen as a frontend library because it follows the UI standards set by Microsoft for applications within Microsoft Teams. (UXPin, 2023). With Fluent UI we created charts and cards that would visualize and display the data from the database to the user. We also implemented an admin check that gives the user administrator privileges based on the “adminCheck” value returned by sending a request to the Azure Functions. If the User has “adminCheck” set to true he is given access to view the company page as well as search for other users CO2 production.
This sprint concluded the development part of the project. The next two sprints revolved around the report writing.

2.6.9 “Sprint 8” 1st May - 12th May

Our goals for this sprint were to deliver a first draft of the project report to our supervisor for review by May 12th.
Sprint eight is the de facto final sprint, after our proposed deadline for focusing our work on the product solution to April 30th, our work now is exclusively dedicated to writing this bachelor project report.

2.6.10 “Final Sprint” 13th May - 26th May

Our goal for this sprint is to finish the bachelor report.
This final stretch of sprint after delivering the first draft and until the final deadline will be focused entirely on writing and reviewing the project report, as well as conducting some quick test cases to add to said report. Our aim is to be complete and deliver a satisfactory product on May 26th that answers all questions about the solution, that is neat, concise, spell-checked, avoids unnecessary repetition and gives the reader a perfect understanding of the application solution.

2.7 Overarching architecture

We had a meeting with our external supervisor where we discussed architecture and general plan around how the architecture of the system we would develop would look like. The discussion resulted in us splitting the architecture of the system into four main parts. These parts include Teams application in the frontend, Azure Functions as a serverless solution where our backend code would live, a database located in Azure and Microsoft Graph API as our source of data.

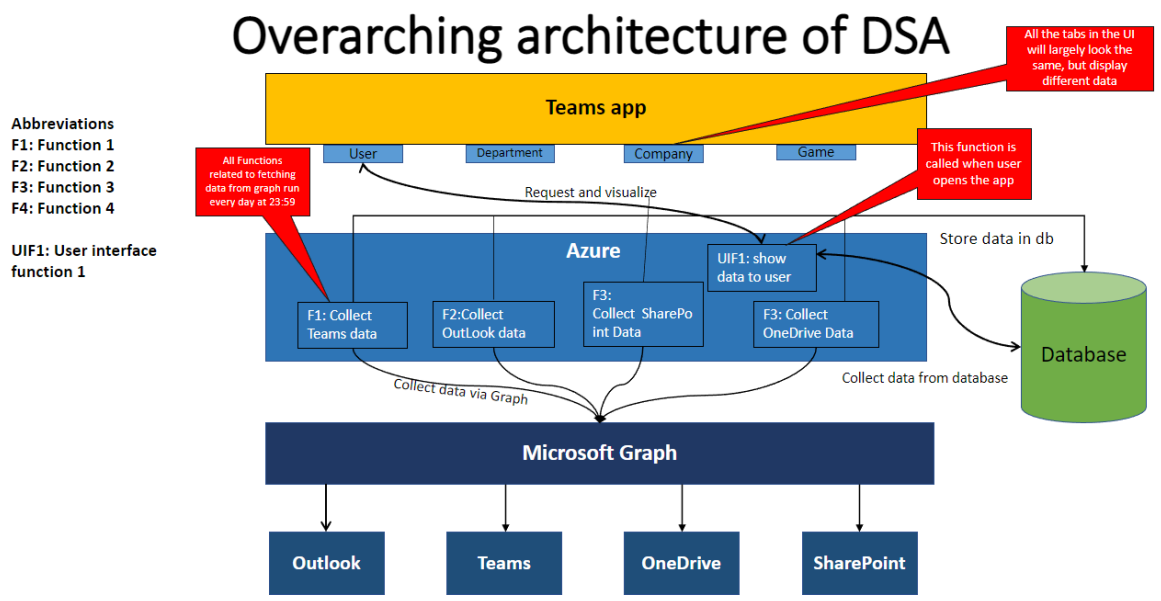


Fig 2.5 - DSA Architecture

2.8 Software tools and programming languages

We chose our tools and programming languages based on what has been made available to us for the project and what the client wanted to use for implementation and documentation.

2.8.1 C# and .NET framework

C# is a high-level, object oriented programming language developed by Microsoft to create durable applications. C# has its roots in the C family of programming languages sharing similarities with languages like C, C++ and Java. Some notable features C# supports include Garbage Collection, Nullable types, Exception handling and Lambda expressions (Microsoft, 2023).

The .NET Framework is an open-source development platform used for building a variety of different applications (Andrea, Chiarelli, 2021). It was originally developed by Microsoft in 2002 to create Windows applications but has evolved over the years to be a cross-platform framework that can be used to create both mobile, as well as, desktop applications. .NET components are based on three main key components

which include; Common Language Runtime(CLI) as a platform within .NET architecture from which .Net programs are executed. Class library, which contains a selection of methods and functions used for core purposes. And languages which include the types of applications that can be built in the .Net framework. This includes forms-based applications, web-based applications and database interaction applications (Barbra T, 2022).

2.8.2 NuGet

NuGet is the packet manager for .NET libraries. It allows users to create, share and consume useful .NET libraries. A NuGet package is a ZIP file with the extension ".nupkg" that contains compiled code, files related to the code in the package and a manifest file. Developers can publish their code as packages either privately or publicly for other developers to use in their own projects (Microsoft, 2022).

2.8.3 Microsoft Graph

Microsoft Graph works as a gateway to data located in Microsoft 365. It is used to gain access to the large amounts of data that is located within Microsoft 365 applications. This is done by exposing REST API and client libraries which gives the user access to Microsoft 365 core services(Outlook, OneDrive, SharePoint, etc). To use Microsoft Graph you can either manually query after the sought after data you want(delegated) or let an application do it for you on your behalf.(Microsoft, 2023)

2.8.4 Microsoft Azure

We have used Azure as cloud computing for our project, but what is Azure actually? "Azure is a cloud computing platform with solutions including infrastructure as a service(IaaS), Platform as a service(PaaS), and Software as a Service (Saas) that can be used for services such as analytics, virtual computing, storage, networking, data backup and disaster recovery, internet of thing, security and compliance and web hosting." (Logan Mccoy, 2023). We chose Azure because of its built-in integration with Microsoft services and flexibility.
According to Mccoy, Azure's inherent flexibility stems from the fact that it is a cloud-based solution; as such, it can back up your data in practically any language,

on any operating system, and from any location. In addition, you can schedule your backup on a daily, weekly, monthly etc.

2.8.5 Azure Functions

Azure Functions is a serverless compute service by Microsoft that can run code in response to predetermined events or conditions (triggers), such as an order arriving on an IoT system, or a specific queue receiving a new message. It automatically manages all the computing resources those processes require (Jay L, 2021).

Azure Functions break apart your system logic into code blocks that are called “functions”. These functions can be made to respond to a number of triggers depending on the needs of the program or application (Microsoft, 2022).

Azure Functions also scale to the needs of the functions. If many requests occur, Azure Functions will scale with the increase. But only to meet the needs of the function. If the amount of requests decreases, the additional resources and computing power will immediately drop off (Microsoft, 2022).

2.8.6 Azure DevOps

Azure DevOps is a collaboration platform where developers, project managers, and contributors can work together to develop software. Azure DevOps contains six main services whose main tasks are to meet the needs of every person that is part of a project. Not only developers (Microsoft, 2022).

Azure Overview is the service where project participants document their work. It includes Summary, where the project is introduced. Dashboard, where each participant sees an overview of the tasks assigned to them. And Wiki where each part of the project is documented. This can vary from development model to code explanations.

Azure Boards provides the tools that development teams need to manage software projects. The service provides native support for Agile, Scrum, and Kanban

methodologies, which will scale depending on the needs of your business. The board allows for the tracking of several types of work items, including, epics, features, user stories, and bugs. Each of these work items are included with a standard set of controls that gives you the ability to link work items, view the history of the work item, create discussions, and view attachments. Controls like related work, deployment, and development are integrated to support when code is changed or released (Microsoft, 2020).

The Azure Repos part offers two types of version control system Git and Team Foundation Version Control (TFVC). (Microsoft, 2022). See more about Git version control on chapter 2.8.6 Git version control.

Azure Test Plans give you access to several tools that you may use to test the apps you've created. During the survey, we made use of this to compile information on the flaw, with the end objective of achieving an overall improvement in product quality. For more about the survey see chapter 2.9 Survey and 3.8 Survey.

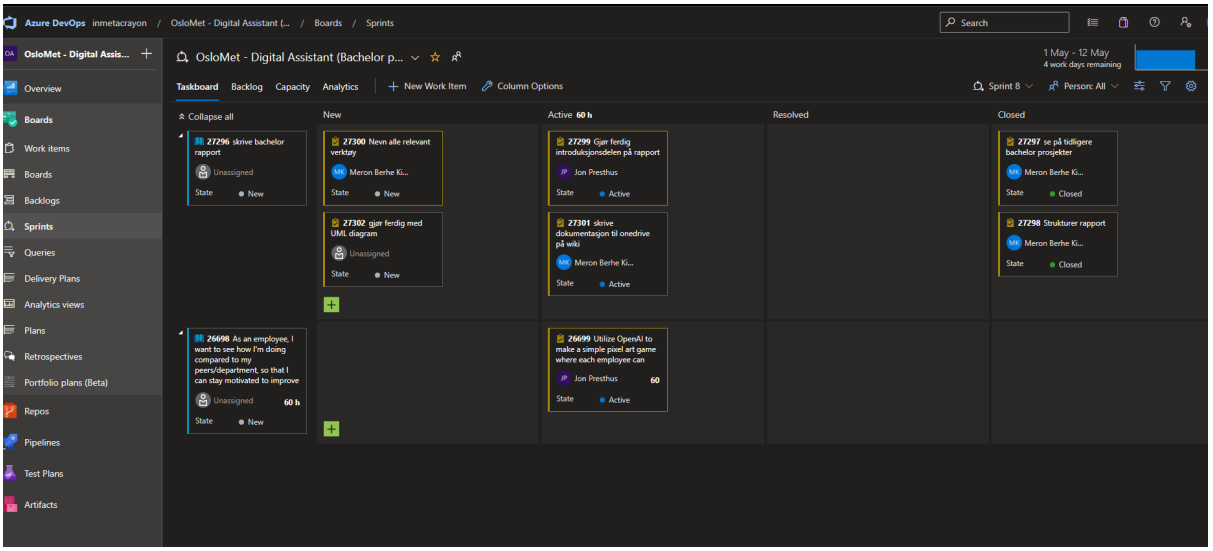


Fig 2.6 - This figure shows our tasks and user stories that will be completed during the sprint.

2.8.7 Git Version Control

As Atlassian defines: “Git is the most widely-used modern DVCS (distributed version control system) in the world. Originally developed in 2005 by Linus Torvalds, the creator of the Linux operating system kernel, this open-source project is still maintained to this day, and is relied upon for version control by a countless number of software projects, both commercial and open-source” (Atlassian, 2023).

Our decision to use Git for our project was obvious and immediately agreed upon, since it is the industry standard for fast, reliable and secure version control, that allows each group member to work on their own branches in the repository with compatibility for pull requests, checking out and merging branches, which is essential for developing software projects together.

2.8.8 SQL Server Database

According to the oracle: a database is an organized collection of structured information, or data, typically stored electronically in a computer system. Usually, a database management system (DBMS) is in charge of controlling a database. Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a "database system," often shortened to just "a database" (Oracle, 2023).

“Data within the most common types of databases in operation today is typically modeled in rows and columns in a series of tables to make processing and data querying efficient. The data can then be easily accessed, managed, modified, updated, controlled, and organized. Most databases use structured query language (SQL) for writing and querying data” (Oracle, 2023)

For this project we are using a SQL Server database located in Microsoft Azure to store data we collect from the Tenant(Inmeta) by using Microsoft Graph. We are following principles surrounding ACID to ensure that data transactions are processed reliably. We are also using ORM(Object-Relational Mapping) to reduce the amount of code produced as well as keep the data secure.

2.8.9 Postman

Postman is a widely used software testing tool which is based on API testing. It is a platform where you can create, test, share and document different API's. Using postman you can create test suites called collections where you can test different API's by sending different HTTP requests e.g PUT, POST, GET, PATCH requests.

2.8.10 SQL

Structured Query Language (SQL) is a programming language for storing and processing information in a relational database. A relational database stores information in a tabular form, with rows and columns representing different data attributes and the various relationships between data values (Amazon Web Services, 2023, February 22).

SQL commands can be divided into two main sub-languages; The Data Definition Language contains the commands used to create and destroy databases and database objects. After the database structure is defined with DDL, database administrators can use the Data Manipulation Language to insert, retrieve and modify the data contained within it (Chapple, Mike, 2021)

2.8.11 ORM

ORM is an acronym for Object Relational Mapping and is a technique used to create bridges between object-oriented programs and relational databases. When interacting with a relational database you have to be able to perform CRUD operations, which, by design, use SQL. While using raw SQL in code is not unheard of, using ORM makes the process of interaction with the database simpler (Abba, Ihechikara 2022).

2.8.12 Power BI

Microsoft created Power BI as a tool for data reporting and analysis. It makes it possible to collect data from a variety of sources, and use that data to create dashboards, reports, and analyses.

Power BI is a platform that provides tools that can be used by nontechnical business users to visualize, analyze, accumulate, and share data. Power BI provides a user interface that should be intuitive to use for users familiar with Microsoft Excel. It is deeply integrated with other Microsoft products, which makes it a flexible tool that requires little training for people to use.

PowerBI is often used to find observations within an organization's data. It can be used to collect different kinds of data sets, modify and clean that data into a data model, and then visualize that data for other users within the organization (Scardina, J , 2022).

We used PowerBI for analytics development on our project, where we got data from a database to analyze in the interactive visualization. According to Microsoft, Power BI is an integrated and scalable platform for business intelligence (BI) at both the self-service and enterprise levels. connect to any data source, visualize it, and then easily incorporate the resulting visualizations into the programs you use on a daily basis (Microsoft Power BI, 2023).

2.8.13 PowerPoint

Microsoft PowerPoint is a tool that is used to create presentations from scratch or a template and designing websites (Microsoft Office, 2023). This tool is used to illustrate the prototypes of our low-fidelity digital website.

2.8.14 TypeScript and React

Facebook created the open-source React.js framework and library for JavaScript. In comparison to using pure JavaScript, it is used to quickly and effectively create interactive user interfaces and web applications (Simplilearn, 2023).

By building reusable components—which you may conceive of as separate Lego blocks—you design your applications with React. These elements are separate parts

of a final interface that, when put together, make up the full user interface for the application (Simplilearn, 2023).

TypeScript is a superset of JavaScript and is an object-oriented, strictly typed programming language. TypeScript code can be used in any JavaScript-compatible setting, including web browsers, Node.js, and your own custom applications (Simplilearn, 2023).

Advantages of TypeScript with React

- The code in Typescript is simple and understandable. TypeScript's statically typed nature is a major selling point. The timing of type checking is what differentiates statically typed from dynamically typed computer languages. In static languages, variables are checked for errors in type (Simplilearn, 2023).
- TypeScript makes it possible to directly write HTML code within a React project. When combined with React, TypeScript's support for JSX in IntelliSense and code completion of the delivery. The short form of JavaScript XM is JSX (Simplilearn, 2023).
- Microsoft Teams supports TypeScript, an open source programming language; it‘s easy to get help if you get stuck while using TypeScript from the community portal (Simplilearn, 2023).

2.8.15 Teams Toolkit

Teams Toolkit is a set of development tools that makes it easier to build and deploy Microsoft Teams applications. The Visual Studio add-on for Teams Toolkit makes it easy to start new Teams app projects with access to cloud storage, integrated identity, content from Microsoft Graph, and others in Microsoft 365 and Azure. Teams Toolkit is a set of tools that makes it easier to build and deploy Microsoft Teams applications. (uctoday, 2022).

According Uctoday described, both a command line interface and a visual interface tool are included in this package. For the purpose of designing Teams applications, using the Teams Toolkit plugin for Visual Studio may result in a more intuitive user

experience. A command-line tool known as TeamsFx CLI is included with the Teams Toolkit. This tool supports CI/CD situations and gives you the ability to do operations using a user interface that is keyboard-centric.

The add-on for Visual Studio known as the Teams Toolkit makes it simple to initiate new projects using Teams. Users are able to set up cloud hosting, run and debug in Teams, quickly build up apps on the Teams Developer Portal, and make use of TeamsFx from inside their existing integrated development environments (IDEs).

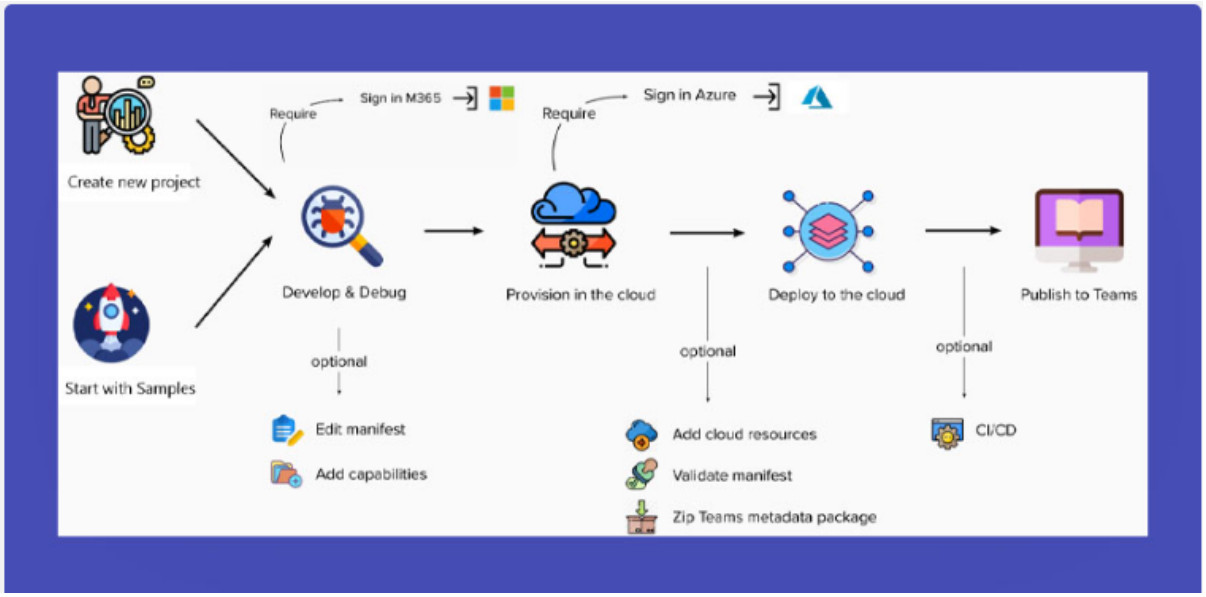


Fig 2.7 - Teams Toolkit process

2.9 Survey

A survey is used for collecting data, information, or opinions from a group of people. They are a fantastic decision to make when you want to learn about the qualities, preferences, opinions, or beliefs of a group of people because they provide this information (Shona, 2022). There are two types of surveys: questionnaires and interviews, but we are going to use an interview survey. “an interview, where the researcher asks a set of questions by phone or in person and records the responses” (Shona, 2022). The purpose of this strategy is to improve the overall quality of the research by obtaining accurate data from respondents. Specifically, this

will be achieved by reducing the likelihood of respondents misunderstanding the survey questions.

2.10 Ideation: Brainstorming and Sketches

Sharing our thoughts about what we wanted the site to be was the first step before we started building it. Figuring out which pages are necessary for the site, what kind of design it should have, and how we can make the product engaging to use were the first steps in our brainstorming process.

Our DSA firm came up with two preliminary designs for the website during the brainstorming session. First, we talked about the features and layout that the website should have. The aim was to build a website where consumers could easily access all the content that provided information about the company.

2.10.1 Low-Fidelity architecture

The first low-fidelity draft of the solution application was brainstormed on a blackboard after the initial concept was discussed with our external supervisor. The points that we discussed under the idea were developed into sketches. The opposite side of the belongs to have a horizontal navigation lane, and the object on the opposite side was oriented vertically (see Figure 2.8.1).

Fig 2.8 - sketching ideas on the whiteboard

2.10.2 Low-fidelity digital web application

After completing our initial sketch, we decided to use PowerPoint to illustrate the application's design. The layout will include charts and cards to visually represent data on each page, which is pulled from our database as you see on figure 2.8.2. We've strategically placed the cards on the right side of each page, with the charts placed directly below them for optimal presentation and user experience.

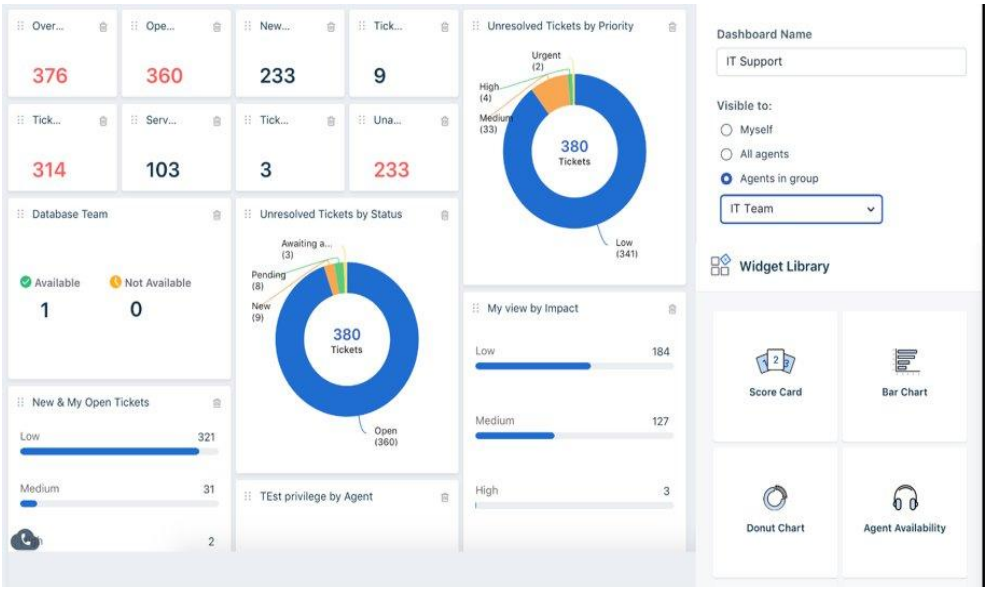


Fig 2.9 - Early prototype of web application layout.

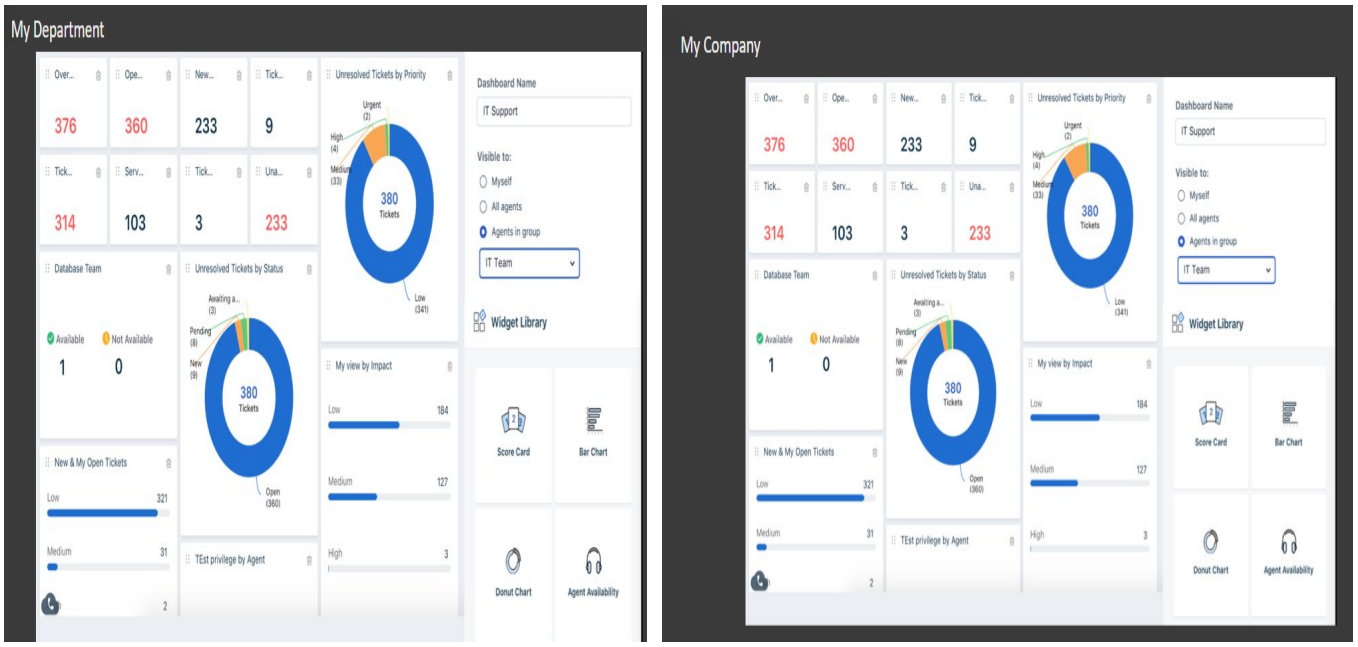


Fig 2.10 - Low-fidelity digital sketch of pages to User, Department and Company

2.10.3 Mid-fidelity

The mid-fidelity is developed in Power BI. We got data from the database to analyze in the interactive visualization (see chapter 2.7.11 Power BI). The website has three pages and each page gets data from the SQL-database.

As you can see below, we tried to visualize the retrieved data as different charts corresponding to these four applications (OneDrive, Teams, SharePoint and Outlook).

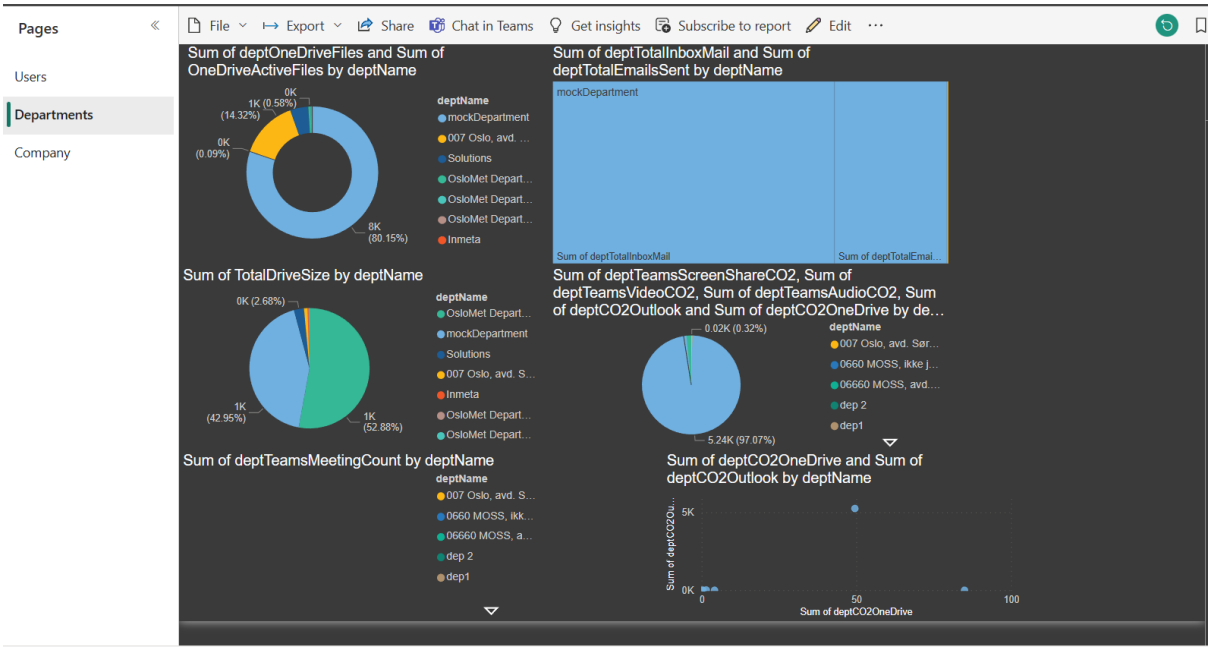


Fig 2.11 - Power BI application

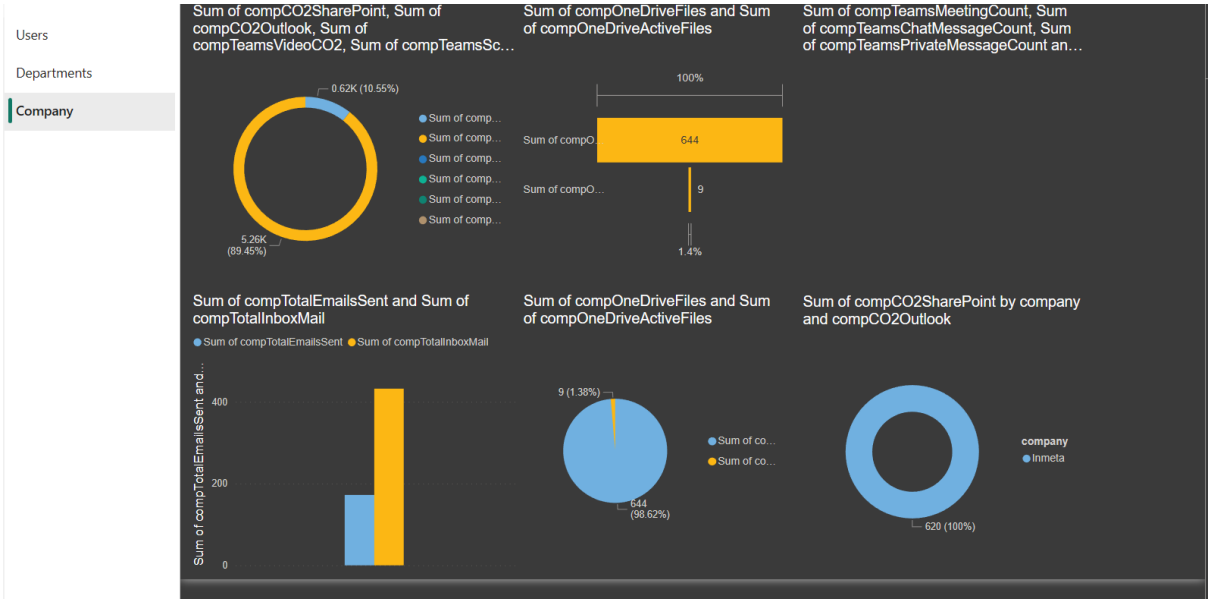


Fig 2.12 - Power BI charts

2.10.4 High-fidelity

Our final product, the high-fidelity website, is based on the Mid-fidelity version and the feedback we got from our external and internal supervisor, and the final product is created by using Typescript with React and UX framework FluentUI in Microsoft Teams. To iterate further on this, see chapter 3.2.2 Final Product and 3.3 Frontend.

2.11 Summary of process

The backend and frontend of the web application were created with the help of the programming languages C# and.Net framework, as well as React, Fluent UI, and Power BI. We worked according to a sprint plan, and our external supervisor participated in a demonstration. To finish the project, we decided to use an agile software development process. The DevOps lifecycle can be shortened using iterative software development by completing work in sprints that are typically between one and four weeks in length. Scrum is an agile framework that assists organizations in facilitating team cooperation and simplifies difficult projects by identifying what the team needs to do in order to complete the project.

The group held a stand-up meeting to talk about the project, as well as a sprint review, a requirement definition, and a plan for 19 weeks' worth of work and development. Because of the comments made by an external supervisor, the requirements were adjusted, and the plan for the work and its progression was divided into 10 sprints. The third round of sprints concentrated on the data collection process by employing Microsoft Graph and Azure Functions. Tenant users provided the information. In addition to conducting research on the project and developing a comprehensive system design, the objective was to hold meetings with an internal supervisor every two weeks. Although we developed Azure Functions to automate the data collection process in Graph, this undertaking was laborious.

During the third sprint, we created a SQL database and saved all of the processed data to it. During the fourth iteration of our sprint, we added ORM to our Azure Functions so that we could map obtained data to objects. During the sprints, the primary focus was on developing algorithms for calculating the amount of CO2 produced by each User and on establishing computations to measure the amount of CO2 produced by Microsoft 365 Services. Calculated CO2 values were also stored in the database. They also worked on addressing the CORS issue in the backend of the application and retrieving data from the Azure database using Azure Functions.

Graphs, cards, and an administrative check were developed during the development of the project by utilizing React and FluentUI.

The next two iterations of the sprint were dedicated to writing the report and running the test cases. The 26th of May is the target date for the delivery of a product that meets all expectations. The architecture of the system was broken down into four primary components, which were a database, a Microsoft graph, an application called Teams, and Azure Functions. The selection of the software tools and programming languages was made after taking into account both the available resources and the requirements of the client. Microsoft's Azure Functions is a serverless computing solution that executes code in response to triggers specified by the user.

A short description of all software and services we have utilized during this whole development process can be found in Chapter 2.8 Software tools and programming languages.

2.12 Challenges

2.12.1 Difficulties with requirement specifications

During the project we had some problems with changes to the requirements for the project. These changes increased the development time and led to the group having to go back to previously completed tasks and sometimes completely redo the previous work. This came as a result from our external supervisor having problems with the code we had written. E.g we initially used raw SQL to query the database, however during a sprint review we were informed that we have to use ORM instead. Another example is a change in how we collect data. Some weeks after we were complete with the data collection part of the project we were given a new method of

collecting data by our external supervisor. This led to us having to redo the entire backend code to fit the new method.

2.11.2 Learning React

React is a javascript framework the group had no prior experience in using. Because of this we had to spend a large amount of time to get a better understanding of how to manipulate the React DOM. This included learning about how to transfer data between parent and child components and how to re-render individual components when changes occur. This led to us having to use large parts of the sprint finding ways to make our UI function as we envisioned and ending up not being as polished as we wanted.

Chapter 3 - Product documentation

The purpose of the Product documentation is to describe the functionalities and components of the final product. The following content of the product documentation envisions that the reader has some understanding of web-application development, relational databases and some knowledge surrounding object oriented programming.

In this chapter, we will explain and document the technical part of the project, which is essential for understanding how we developed our product. The section describes different considerations, choices, and solutions that were made throughout this phase, as well as reflections on the challenges we have encountered along the way.

3.1 Introduction to product solution

Our product, the Digital Sustainability Agent (DSA), is a digital application built to run natively within Microsoft Teams. This application and all of its components, make up our bachelor project for Inmeta Consulting. For more information about the project, see the process documentation (Chapter 2).

The purpose behind this solution is to offer companies a fast and simple to read overview of their carbon footprint, and to present the retrieved data in a clean and understandable way so that the subject can understand the reports and take action where needed.

We can divide our solution into three main parts which all have sub parts. The main parts are based on Azure Functions, Azure SQL Server, and React, respectively. These parts work together to collect, process, store and present actionable data.

The main part of the application is the *backend*, written in C# on the .NET Framework, that makes calls to Microsoft Graph to retrieve data and runs Azure Functions that will save the processed data to and update the SQL database. Azure is also utilized for authorizing the client user through Azure Active Directory (Azure AD) so that they can access their own Microsoft 365 records.

The client-side of the solution (*frontend*) is a React application written in TypeScript. With the help of HTTP-requests to the Azure functions, the application can retrieve data from the SQL database.

3.2 Development of product

3.2.1 Road from initial prototype

Originally, our vision for the product was to have four Azure Functions that run asynchronously and fetch the data of all users in multiple nested ‘forEach’ loops. In other words, each Function (OneDrive, Teams etc.) would first run one loop querying for users, then for each user, it would run another loop, making multiple calls to the Graph API to retrieve data.

Our solution soon changed from this initial vision because we found out the first version wasn’t efficient at all and took upwards of five minutes to run through and retrieve data. As we got feedback from an external supervisor, the main goal is that the function must not take a long time to get data.

We changed the Functions to get data from Microsoft Graph by using an alternative API request to query a complete CSV report instead, which was faster and simpler to use for our application.What previously took minutes, now had our data retrieved and saved in the database in seconds. The main reasons for this change were numerous benefits such as improving the performance of the application, scalability, resource optimization, competitive advantage, and encouraging best practices.

3.2.2 Final Product

The final product consists of many smaller, separate parts that work together to form the application. We have multiple Azure Functions that both fetch and enter data into the database. An SQL database that is scheduled to update once a week, and an application interface that displays the data from the database.

The Teams application is created with React and the UX framework FluentUI. It consists of four pages that can be navigated through the use of a navbar. This navbar has four elements that can be selected: “Your Report”, “Your Department”, “Your Company” and “Run Forrest Run”

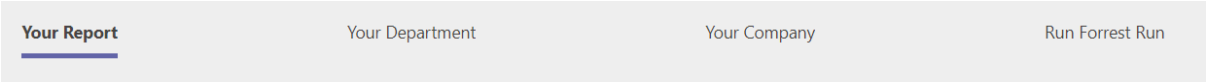


Fig 3.1 - Teams app navbar

The pages are separated based on the data that it shows. The user pages show data about the user. The department page shows data about the user’s department and finally the company page shows data about the company as a whole. The “Run Forrest Run” page was supposed to have a small game, but was cut out due to time constraints so it currently only holds a placeholder for the game.

On a page load we send a request to our azure function and the application will display a loading icon that is replaced with the cards and graphs that displays the data once the request gets a response.

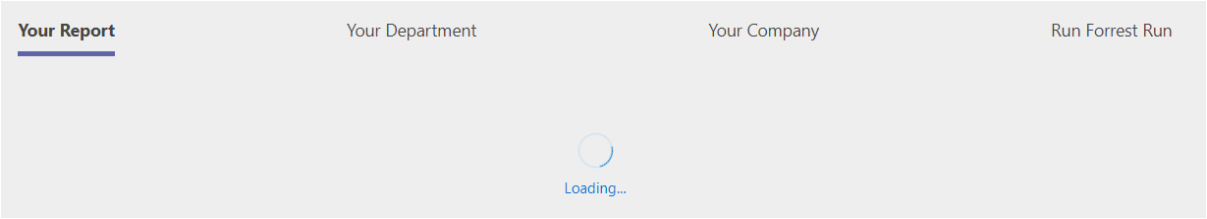


Fig 3.2 - Teams app loading

Once the API has made a successful API request we display the data in the before mentioned cards and charts.

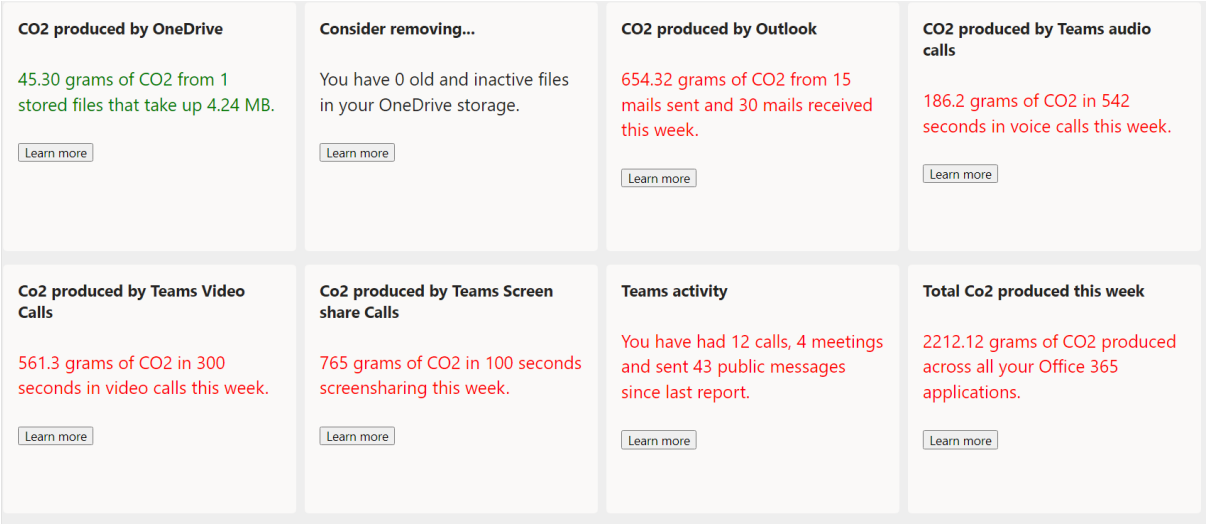


Fig 3.3 - Teams app report with FluentUI cards

The cards tell the user about their usage of the Office 365 applications and associate the activities the user has done throughout the week with CO2 produced. The text color used in the cards are either red or green to indicate to the user if their CO2 production is within predetermined limits or not. Green signifies that the user is within set limits while red indicates the opposite.

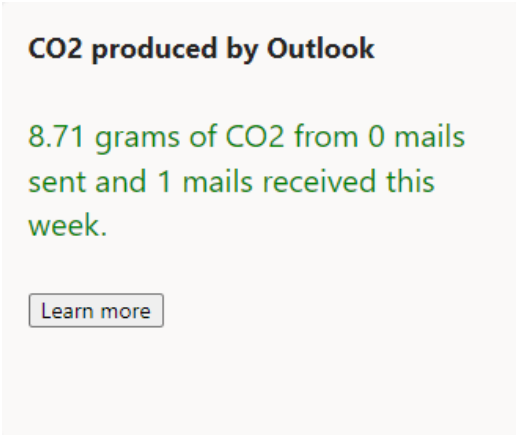


Fig 3.4 - Card example

While using the mouse to hover over the cards a text pop-up appears under the card. This text pop-up is used to give the user some perspective as to how much CO2 is produced by digital activities in comparison to other non-computer related activities like driving a car or boiling a kettle of water.

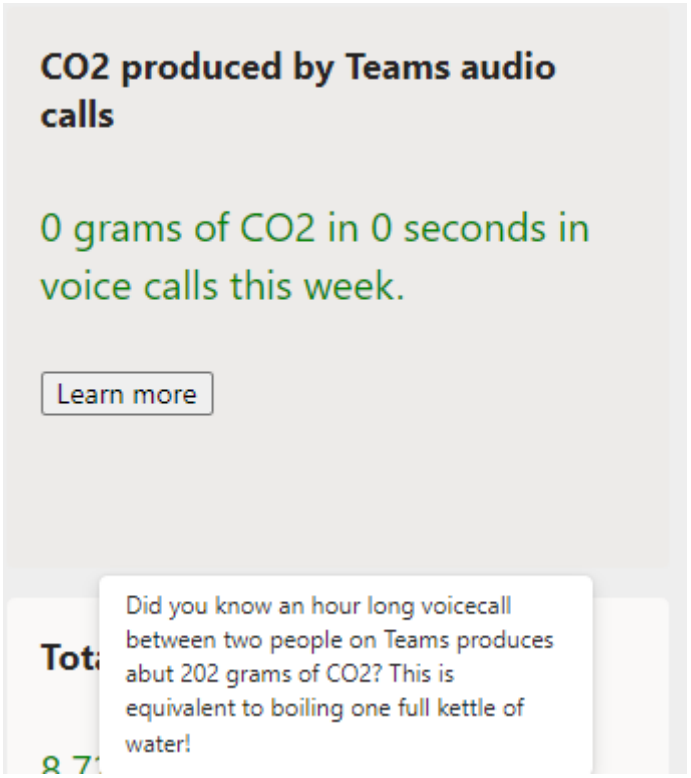


Fig 3.5 - Card description example

The graphs are there to visualize the data so that the user can get a better understanding of their own CO2 production. We wanted the graphs to be simple to understand and make it easy to get a quick overview over CO2 and data production.

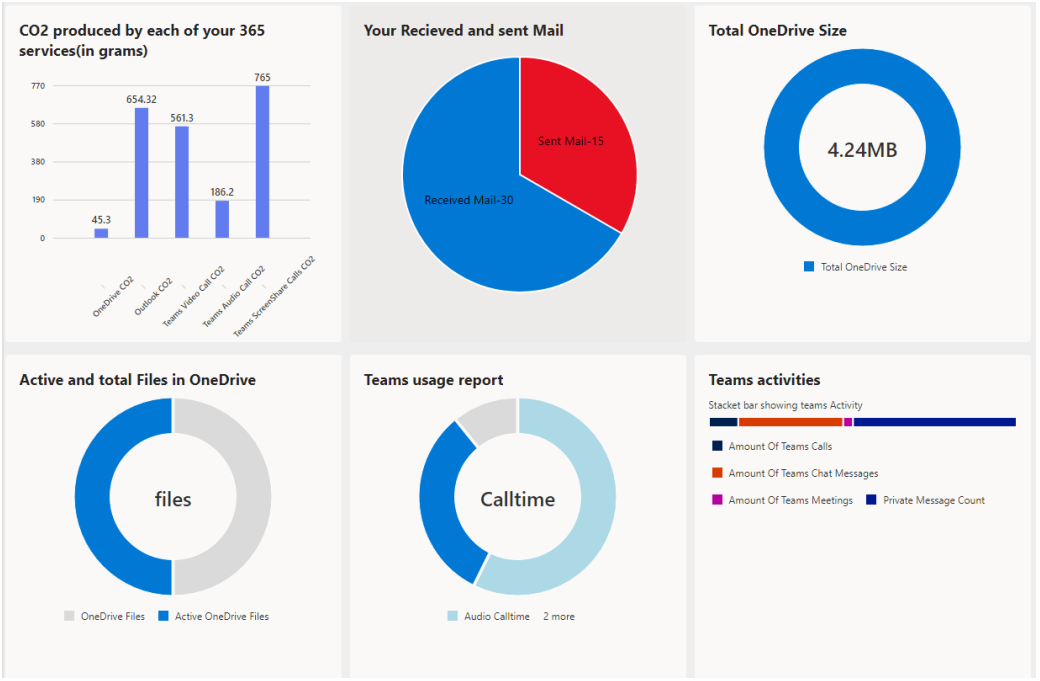


Fig 3.6 - Teams app with FluentUI graphs

Administrators are given additional functionalities so that they can get a greater overview of the company's CO2 production as a whole. These functionalities include:

- The ability to view other users reports
- Having access to the “Your Company” tab which contain data about the company as a whole
- See the weekly reports of all departments within the company

The admin panel of the “Your report” tab has an input field where the administrator can search the email of any person within the tenant and get the report of the user.

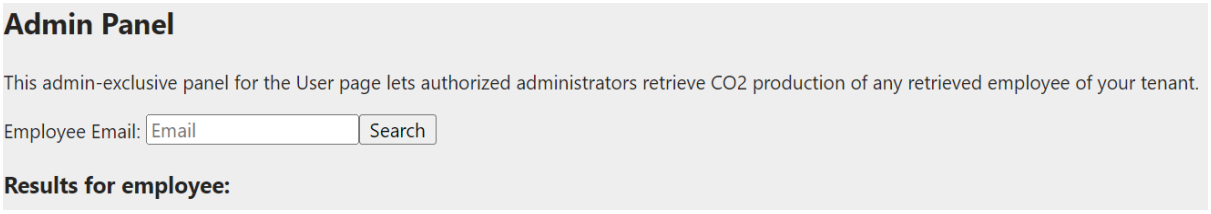


Fig 3.7 - Admin panel with email query

After querying for the wanted user the cards and charts will be populated with the weekly report stored in the database.

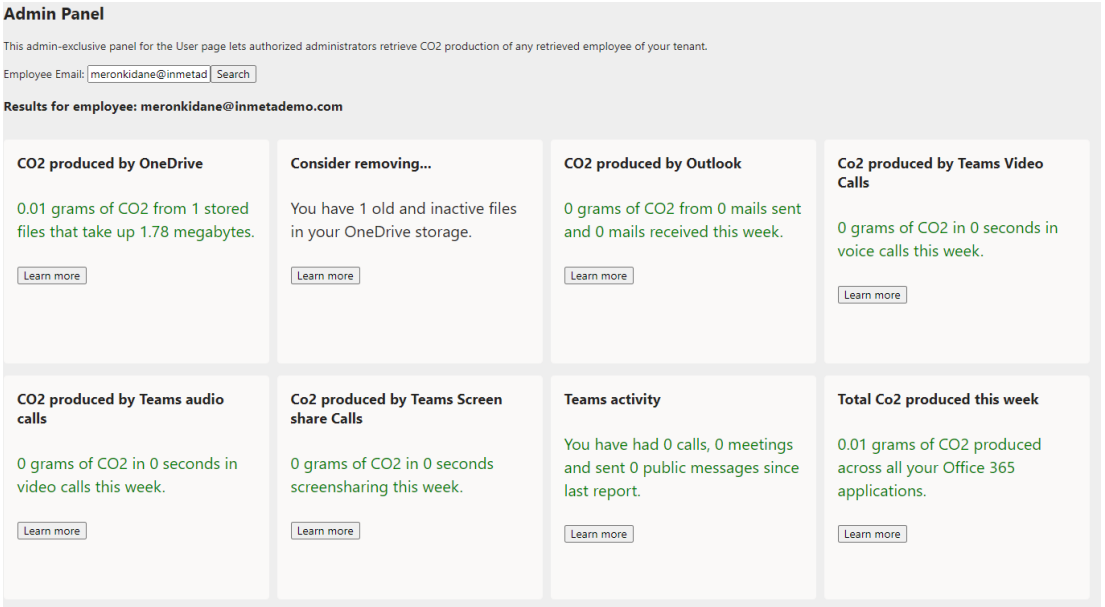


Fig 3.8 - Frontend page 1 “user report”

The same can be done in the “your Department” tab, only now the input field requires that you input the name of the department.

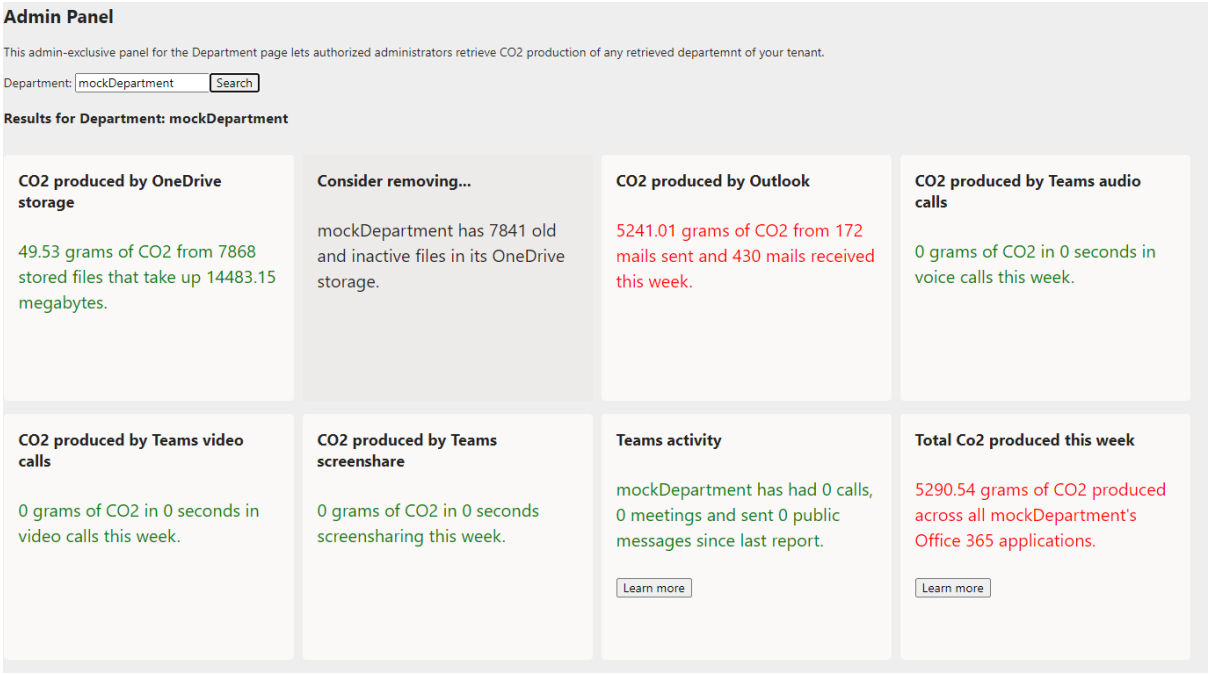


Fig 3.9 - Frontend page 2 “department”

Administrators also have the ability to view the “Your Company” page, which contains data that summarize every user's usage of Microsoft 365 services. This page also contains additional information about Sharepoint as it displays to the administrators the total size of the company's SharePoint in addition to how much CO2 is produced through the use of SharePoint.

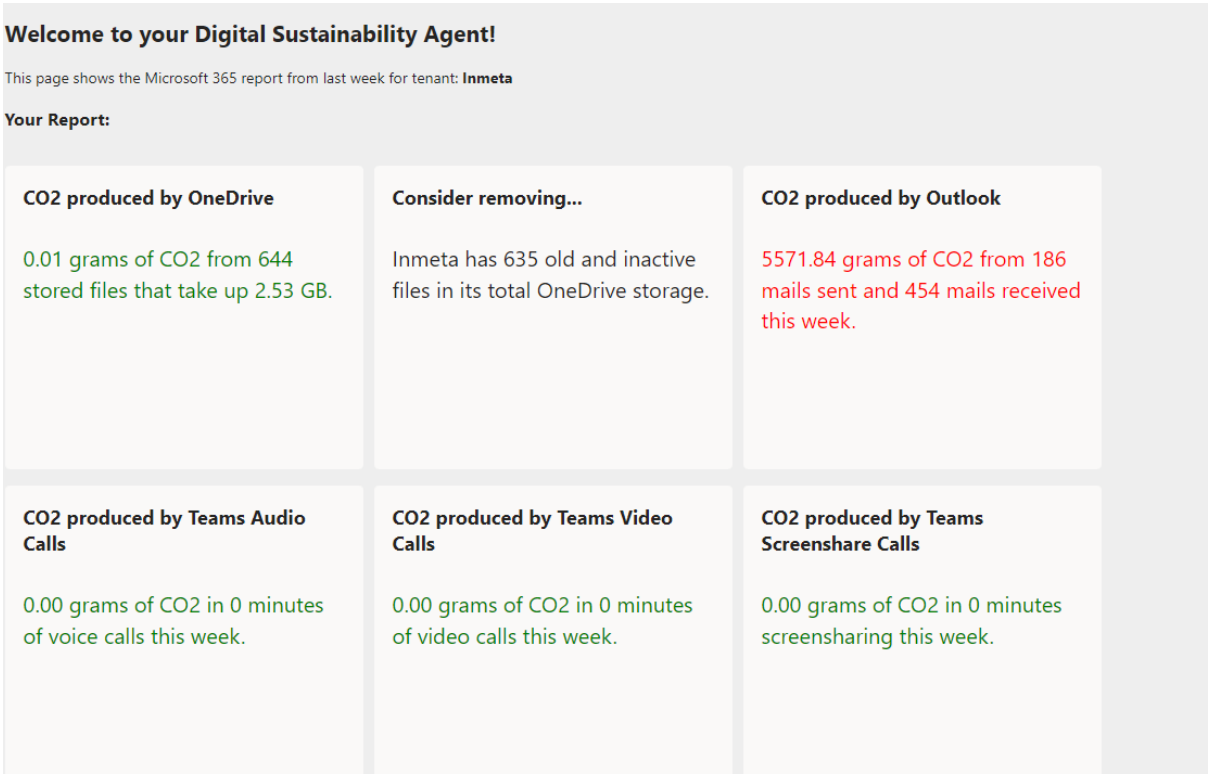


Fig 3.9 - Frontend page 3 “Company - Admin”

In cases where the user is not an administrator the “Your company” page will not fetch data from the database but rather tell the user that they are unauthorized to view the contents of this page.

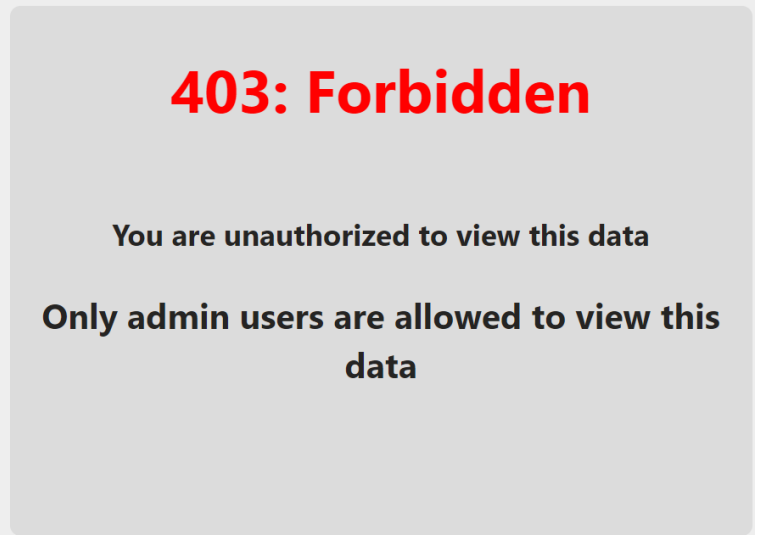


Fig 3.9 - Frontend page 3 “Company - Forbidden”

The “Run forrest Run” game is part of the requirement specifications that we were not able to complete in time. We therefore chose to have a placeholder image instead of a fully functioning game.

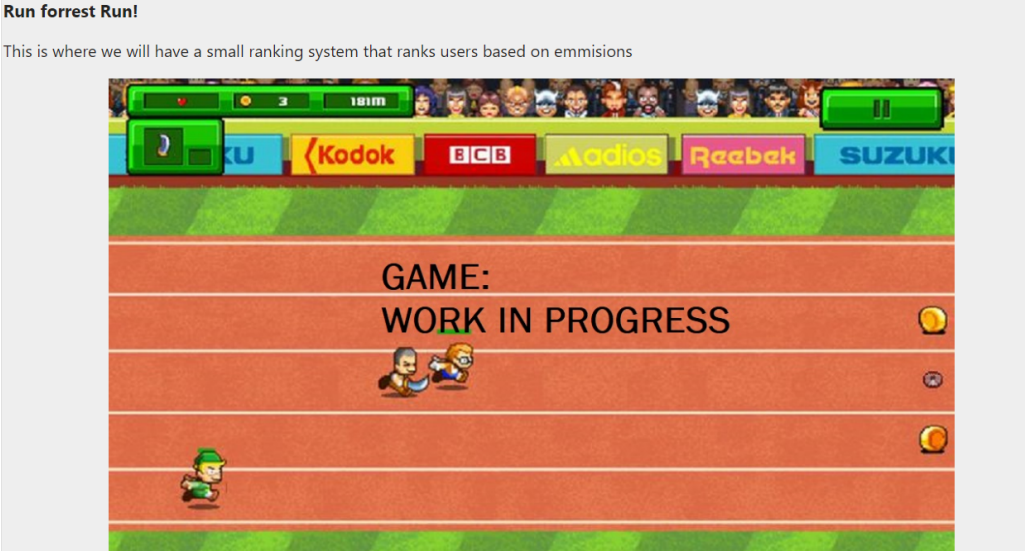


Fig 3.10 - Frontend page 4 “game”

3.3 Frontend

A variety of technologies are required to support the web application in order for it to perform efficiently. This section describes the web application's frontend as well as a variety of auxiliary utilities.

3.3.1 TypeScript with React

Our frontend application is developed in TypeScript using the JavaScript framework React in Microsoft Teams, due to its modularity, performance, and compatibility with Microsoft technologies. It is a good option for developing scalable and easily maintained applications in Microsoft Teams. The components are based on architecture and compatibility across platforms. (see chap. 2.6.13). We had extra packages that were used to develop this application, and those are as follows:

- Node v16.16.0
- Fluent UI v9.1.8
- Adaptive Cards v2.11.2

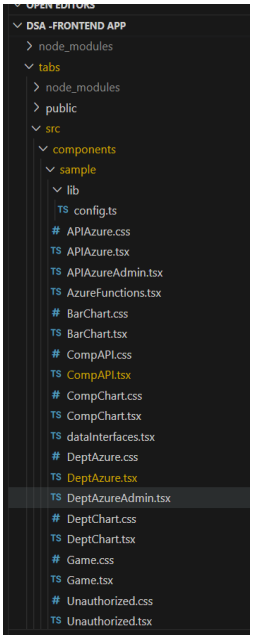


Fig 3.11 Frontend components

Node Package Manager (npm) is an open-source library of tools used by engineers to create applications and websites (UXPin, 2023). They are used for numerous development processes, such as dependency management, development server, build tooling, TypeScript transpilation, and package management.

3.3.2 Teams

Our application is developed to be used within Microsoft Teams. Microsoft Teams has many features, such as security and compliance, channels and teams, customization, and extensibility. In the customization and extensibility section, Microsoft Teams allows the customization of its users, and it supports the development of a platform that can be integrated into Teams. So, the DSA application is one part of the Teams application, which is developed in Microsoft Teams.

3.3.3 DSA - application

DSA is the abbreviation for "Digital Sustainability Agent." We have four pages that identify the user, department, company, and game page. Each page has its own function that retrieves data from the database.

3.3.3.1 Data Mapping

“Data mapping is the process of connecting a data field from one source to a data field in another source. This reduces the potential for errors, helps standardize your data and makes it easier to understand your data” (Informatica, 2023) . As seen in the following code snippet, we demonstrate how the map function is used to put associated values of the incoming JSON object with the values of a type object located within the code.

```
<Flex gap='gap.small'>
 <div>
 <Tooltip content="Did you know that 1 TB of data stored in OneDrive produces 178 KG of CO2 per year? This is equivalent to driving a
 car for about 1600km(1000 miles)!" position="below">
    <Card aria-roledescription='card avatar'
    styles={{height: "280px", width: "330px", marginTop:"15px"}}>
        <CardHeader>
            <Text content="CO2 produced by OneDrive" weight='bold' size="large"/>
        </CardHeader>
        <CardBody>      Jon-pre, last month • commit
            {message.map(item =>(
                <p style={{ fontSize: "20px", color: item.CO2OneDrive > 50 ? "red" : item.CO2OneDrive <= 50 ? "green" : "" }}>
                {(item.CO2OneDrive).toFixed(2)} grams of CO2 from {item.OneDriveFiles} stored files that take up {(item.TotalDriveSize).
                toFixed(2)} MB.

                </p>
            ))}
        </CardBody>
        <CardFooter fitted={true}>
        <form action="https://www.cibotechnologies.com/blog/calculating-the-carbon-footprint-of-zoom-meetings/
        #:~:text=The%20audio%2Donly%20calls%20would,emit%202.8%20kg%20of%20CO2/" method="get" target="_blank">
  <button type="submit">Learn more</button>
</form>
        </CardFooter>
    </Card>
    </Tooltip>
```

Fig 3.12 The highligheted code snippet shows how we mapping the data

The object “message” is a react state object of type “MyData”. Meaning that once the state of the object is changed, the component will re-render. The state first contains nothing as it waits for the fetch request to collect the data for the user. Once the JSON object arrives it is parsed and the state of “message” is updated to contain the data received by the fetch request. The component re-renders and maps all values within the “MyData” object to the desired places within the code.

```
export type Mydata = {
    id:string,
    email:string,
    OneDriveFiles:number,
    OneDriveActiveFiles: number,
    TotalDriveSize: number,
    totalInboxmail: number,
    totalEmailsSent: number,
    TeamsCallCount: number,
    TeamsMeetingCount: number,
    TeamsChatMessageCount: number,
    TeamsPrivateMessageCount: number,
    TeamsAudioDuration: number,
    TeamsVideoDuration: number,
    TeamsScreenshareDuration: number,
    CO2OneDrive: number,
    CO2Outlook: number,
    TeamsAudioCO2: number,
    TeamsVideoCO2: number,
    TeamsScreenShareCO2:number,
    admincheck:boolean,
    deptName: string
}
```

Fig 3.13 - “MyData” JSON object

3.3.3.2 User

The Fetch API is used to make HTTP requests «POST». An HTTP request instructs a server (connected to a database) to create, retrieve, remove, or modify database elements. In the web application, this is used to handle users, records, comments, and most other things that users have access to, and communicate with the backend. Consider the following example: Fig 3.4.3.1

```
useEffect(() => {
    setloadingSession(true)
    if(sessionData){
        setMessage(JSON.parse(sessionData))
        setloadingSession(false);
    }else{
    if(userName.includes('@')){
    const requestOptions = {
      method: 'POST',
      headers: { 'Content-Type': 'application/json'},
      body: JSON.stringify({
        email: userName      })
    };
    fetch('https://oslometazurefunctions.azurewebsites.net/api/GetUserData?
    code=8lwqP0eqHyXidGW2YHXdHuDhusrAUYS_Wgjfo-IGBGpvAzFuEk8ayA==', requestOptions)
      .then(response => response.json())
      .then(data =>{setMessage(data)
        window.sessionStorage.setItem("UserData", JSON.stringify(data))
        setloadingSession(false)
        console.log(data)})
      .catch(error => console.error(error));
    }
}
```

Fig. 3.14 - The highlighted code show the url to our azure function that collect data from the database about the teams user

3.3.3.3 Department

This page works very similarly to the “User” page shown above in sub-chapter 3.4.3.1. The Fetch API makes a “POST” HTTP Request to retrieve and present the correct data from the database “department” table. The department is queried with a Graph API call directed at a “/me/” address to quickly and automatically retrieve the delegated (signed in) user’s corresponding department.

```
useEffect(() => {
    const deptSessionStorage = window.sessionStorage.getItem("DeptValues");
    if(deptSessionStorage){
        setMessageDept(JSON.parse(deptSessionStorage))
    }else{
    if(deptName){
        const requestOptions = {
            method: "POST",
                headers:{'Content-Type': 'application/json'},
                body: JSON.stringify({deptName: deptName})
        };
        fetch(`https://oslometazurefunctions.azurewebsites.net/api/GetDeptData?
        code=s69n0paO-wjlvN05GlSDGAytdljW0cLe_C-B-xtcwkd1AzFuqcXRxA==`, requestOptions)
        .then(response => response.json())
        .then(data=>{setMessageDept(data)
            window.sessionStorage.setItem("DeptValues", JSON.stringify(data))
        console.log(data)})

        .catch(error => console.error(error));
    }
}
```

Fig. 3.15 - The figure shows how we use insert the department of the user as the url body and sends a request to the Azure Function

Additionally, if the signed-in user is a system administrator with the proper authorization, they can view and request the data of any preferred department by way of a text input form that sends a “POST” HTTP Request, and uses the Fetch API to return and present the designated department information, queried from the database table by the database name inputted into the form by the admin.

```
if(AdminDeptName){
    const requestOptions = {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        deptName: AdminDeptName     })
    };
    await fetch('https://oslometazurefunctions.azurewebsites.net/api/GetDeptData?code=s69n0paO-wjlvN05GlSDGAytdljW0cLe_C-B-xtcwkd1AzFuqcXRxA==', requestOptions)
      .then(response => response.json())
      .then(data =>{setMessageAdminDept(data)
        window.sessionStorage.setItem("AdminDeptValues", JSON.stringify(data))
        console.log(data)})
      .catch(error => console.error(error));
  }
}
```

Fig 3.16 - This figure shows how we check if the user is an admin and then give the admin the possibility to search for the report about wanted department

3.3.3.4 Company

This page presents the total sum of all statistics data, and the carbon footprint, for the entire tenant. Following the principle of least privilege, which states that every

user must only be able to access the information and resources that are necessary for its legitimate purpose (Saltzer, Jerome H et al,. 1975), the “company” page is made accessible only to authorized system administrators. Whether the user is an administrator or not is determined by the JSON object received in the HTTP-request. If the field “adminCheck” is true, the page will load with the data populating the fields. Anyone lacking the proper clearance to view its contents will be met with the following 403 HTTP response status code error:

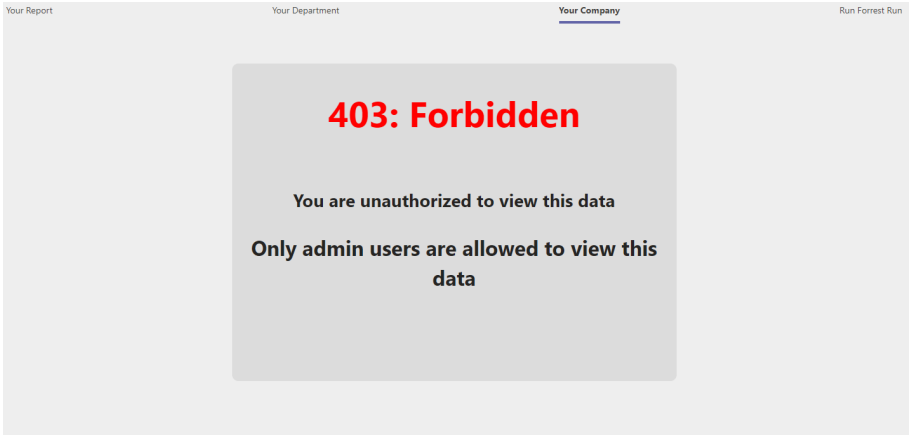


Fig 3.17 - Figure shows what a normal user is shown when clicking on the “Your Company” page

When viewed by an authorized admin, the page will display correctly and use the Fetch API to execute a “GET” HTTP Request and display the information and graphs for the entire tenant, as shown in the following figure:

```
useEffect(() => {
    const compSessionStorage = window.sessionStorage.getItem("CompValues");
    if(compSessionStorage){
        setMessage(JSON.parse(compSessionStorage))
    }else{
    const requestOptions = {
        method: 'GET',
        headers: { 'Content-Type': 'application/json'}
    };
    fetch('https://oslometazurefunctions.azurewebsites.net/api/GetCompData?', requestOptions)
    .then(response => response.json())
    .then(data =>{setMessage(data)
        window.sessionStorage.setItem("CompValues", JSON.stringify(data))
        console.log(data)
        console.log(data.company, data.compCO2OneDrive) })
    .catch(error => console.error(error))
}
},[])
```

Fig 3.18 - Figure shows the url used to fetch company data

3.3.3.5 Run Forrest Run

This page shows the ranking of users based on CO2 emissions produced last week. The reasoning behind this game is, it is important to consider the sustainability of the applications we use at work. This can motivate employees by seeing the ranking at this game. As we mentioned above, this will be one of our future development processes (see chapter 1.3 Objective).

3.3.4 Design

3.3.4.1 Fluent UI

Microsoft Fluent UI is a user interface design system. It seeks to provide a consistent and approachable design language for designing user experiences across several Microsoft products and platforms, including Windows Office and online apps (Microsoft 2023).
We decided to go with Fluent UI because it places an emphasis on visual clarity, responsive design, and smooth interactions. Fluent UI also makes use of elements such as light, depth, motion, and material to produce interfaces that are interesting and easy to use. The design system also contains a library of reusable components and standards to assist us as developers in the creation of unified user interfaces across our Microsoft Teams application.

3.3.4.2 Selection of Colors

The application employs a minimalistic use of colors, primarily focusing on matching the color palette of Microsoft Teams. The only exceptions are the Fluent UI cards and graphs that use complementary colors to highlight the data from the background and to differentiate it from each other.

3.3.4.3 Charts

We are using charts to display data to the users in a graphical, tangible way. According to FluentUi, "Charts is a set of modern, accessible, interactive, and highly customizable visualization libraries representing the Microsoft design system" (Microsoft FluentUI, 2023).

The component that contains the charts is a child-component of the main component that is used to retrieve the JSON object. We can therefore pass the JSON object as a prop to the child component for it to utilize the same data as the parent component uses to place data in the cards.

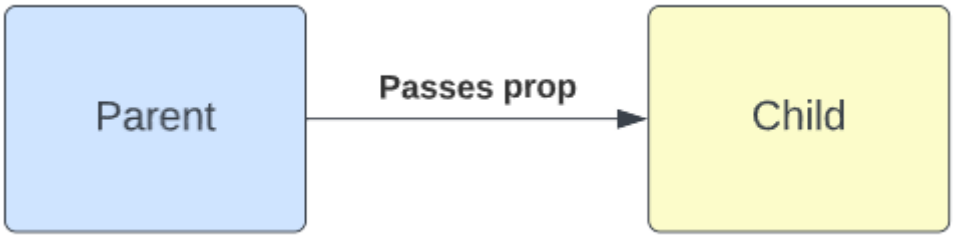


Fig 3.19 - Parent Child relation

To utilize the charts available in FluentUI we have to import the modules we desire in the component we want our charts to be in.

```
import { VerticalBarChart,
         IVerticalBarChartDataPoint,
         PieChart,
         IChartDataPoint,
         DonutChart,
         IChartProps,
         StackedBarChart} from '@fluentui/react-charting';
```

Fig 3.20 - Imported modules

```
export function BarChart(props: { arr: Mydata[]}){
```

Once the parent component has received the JSON object it is passed to the child component and we use the “map” function to place the desired data within the desired charts.

```
props.arr.map((arr)=>{
    pointsDonutDriveSize = [
       {legend: 'Total OneDrive Size', data: arr.TotalDriveSize, color:'#0078D4', xAxisCalloutData:'Total Size of your OneDrive (in MB)'}
    ]
    pointsDonutChart = [
       {legend: 'OneDrive Files', data:arr.OneDriveFiles, color: '#DADADA', xAxisCalloutData: 'Your total OneDrive files' },
       {legend: 'Active OneDrive Files', data:arr.OneDriveActiveFiles, color: '#0078D4', xAxisCalloutData: 'Files you actively use' },
    ];
    StacketPoints = [
       {legend:'Amount of teams calls', data:arr.TeamsCallCount, color:DefaultPalette.blueDark},
       {legend:'Amount of teams chat messages',data:arr.TeamsChatMessageCount, color:DefaultPalette.orange},
       {legend:'Amount of teams Meetings', data:arr.TeamsMeetingCount, color:DefaultPalette.magenta},
       {legend:'Private message count',data:arr.TeamsPrivateMessageCount ,color:DefaultPalette.blueMid}
    ];
    pointsPieChart = [
       {
           x: 'Sent Mail',
           y: arr.totalEmailsSent
       },
       {
           x:'Received Mail',
           y: arr.totalInboxmail
       }
    ];
```

We then create variables that extend “IChartProps” so that the chart recognizes that the data is a prop that contains some data and a title.

```
const donutData: IChartProps = {
    chartTitle: 'Donut chart showing Active files and total files',
    chartData: pointsDonutChart,
};
```

Finally the data is inserted into the HTML element and the data is displayed on the UI.

```
<Card styles={{height:"400px", width: "400px", marginTop:"15px"}}>
    <CardHeader>
        <Text content="Your Recieved and sent Mail" weight='bold' size='large' />
    </CardHeader>
    <CardBody className='cardbody'>
        <PieChart
        data={pointsPieChart}
        height={300}
        width={300}
        colors={colors}
        />
    </CardBody>
</Card>
```

3.3.4.4 UI Card

A card is a UI design pattern that groups related information in a flexible-size container visually resembling a playing card (Nielsen Norman Group, 2016). We used “cards” in our front-end user interface as you see below:

Fig 3.21 - Fluent UI Card HTML code

The card is using a card component with the specific properties and styles. This <card> indicates the use of the card components, which is imported from *@fluentui/react-northstar* framework. The aria stands for accessible rich internet applications, that describes the card’s role in terms of accessiblity, but in this section it implies the card is linked to an avatar. The “styles' ' defines the card component’s inline styling, and the styles property holds an object containing CSS attributes and values. In our case we have used a height of 280 pixels, a width of 330 pixels, and a top margin of 15 pixels for all cards we have implemented.

3.4 Backend

3.4.1 Azure Functions

We have a total of 11 Azure Functions, each of which serves a unique purpose and uses various triggers.

The reason for having many smaller Azure Functions instead of few larger ones is because we are following the principle of separation of concerns. In computer

science, separation of concerts is a design principle for separating a computer program into distinct sections. Each of our Azure Functions addresses a separate concern as to not interfere with other Azure Functions.

Four of the functions are used to collect data about the user's Office 365 usage within the tenant using Microsoft Graph and insert that data into a SQL server database located in Microsoft Azure. Another four retrieve data from the database, calculate the CO2 produced by each Microsoft 365 service, and then insert the calculated CO2 back into the database. The final three are HTTP triggers that are used in the Teams application to fetch data to display in the Teams application. All of the Azure Functions were created in Visual Studio 2022 and were later on published to Azure.

Name	Trigger	Status
calculateCo2Onedrive	Timer	Enabled
calculateCo2OutLook	Timer	Enabled
calculateCo2SharePoint	Timer	Enabled
CalculateCO2Teams	Timer	Enabled
GetCompData	HTTP	Enabled
GetDeptData	HTTP	Enabled
GetOnedriveReport	Timer	Enabled
GetOutLookReport	Timer	Enabled
GetSharePointReport	Timer	Enabled
GetTeamsReport	Timer	Enabled
GetUserData	HTTP	Enabled

Fig 3.22 - All Azure Functions for DSA application

To be able to communicate with the database each Function has to have a Database context. Database context represents a session of the Database and allows for querying and insertions into the database.

```
1 reference
public class GetOnedriveReport
{
    private readonly DSAContext _dsaContext;
    0 references
    public GetOnedriveReport(DSAContext dsaContext)
    {
        _dsaContext = dsaContext;
    }
```

Fig 3.23 - Figure shows how a class implements a constructor for DSAContext which is an instance of DbContext

When declaring a TimeTrigger Function it has to have CRON expression to determine at what time the code will execute.

```
[FunctionName("GetOnedriveReport")]
0 references
public async Task Run([TimerTrigger("0 0 23 * * SUN")]TimerInfo timer
    ,ILogger log, ExecutionContext context)
```

Fig 3.24 - This figure Shows the Azure Function method and cron expression used to determine at which time the function will execute (23:00 every sunday)

Our Azure Functions are scheduled to run once weekly, between 23:00 and 23:10 UTC on Sundays. The reason for this is that we've settled on displaying a weekly report to the user. Thus, data from Monday through Sunday is included in the data set. We could have the function run daily if we wanted, but a weekly report would provide the user with a better picture of their long-term carbon output.

Microsoft Graph calls need to be authenticated when being used from an application or service and not directly by an agent. To be able to use Microsoft Graph to collect user reports we have to create a Graph Client for us to use in our application. The figure below shows how we create a Graph Client by using a client-id, tenant-id and clientSecret. These three id’s are essential as they indicate where the Graph queries are coming from and which tenant Graph will collect data from.

```
IConfidentialClientApplication confidentialClientApplication = ConfidentialClientApplicationBuilder
        .Create(clientId)
        .WithTenantId(tenantId)
        .WithClientSecret(clientSecret)
        .Build();

// Acquire tokens for Graph API
var scopes = new[] { "https://graph.microsoft.com/.default" };
var authenticationResult = await confidentialClientApplication.AcquireTokenForClient(scopes).ExecuteAsync();


// Create GraphClient and attach auth header to all request (acquired on previous step)
var graphClient = new GraphServiceClient(
    new DelegateAuthenticationProvider(requestMessage =>
    {
        requestMessage.Headers.Authorization =
            new AuthenticationHeaderValue("bearer", authenticationResult.AccessToken);
        return Task.FromResult(0);
    }));
```

Fig 3.25 - Shows how we implement a graph client using clientId, tenantId and clientSecret.

When sending a request to Graph it has to be specified in the query what will be returned to the client. The Figure below shows how Graph fetches all the users in the tenant and from each user selects id, department and mail.

```
var allUsersOneDrives =
    microsoftGraphClient.
    Users.
    Request().
    Select("id, department, mail").
    Top(300).GetAsync().Result;

var adminGroup = config["adminGroup"];
```

Fig 3.26 - This figure shows how we query Microsoft Graph after all the users and only choose id, department and mail.

We use Microsoft Graph to collect a report about a user's OneDrive activity and it is returned as a stream in the format of a CSV file. To extract the report from the stream we have to use a Nuget package called “CSVhelper”. CSVhelper allows for the mapping of CSV headers to C# objects which let us extract and pick apart the headers together with the values associated with the headers.

```
// get all files from users OneDrive
var OneDriveReport =
    microsoftGraphClient.
    Reports.
    GetOneDriveUsageAccountDetail("D7")
    .Request().GetAsync().Result;
```

Fig 3.27 - Here we show how we use Microsoft Graph to collect a report containing data about the tenant user's OneDrive usage

We choose values we want from the CSV file and store it in variables so that we later can use those variables in a C# object to insert the data into the database.

```
using (var reader = new StreamReader(OneDriveReport as MemoryStream))
// Supports reading and writing CSV files using custom class and object through the help of csvHelper
using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))
{
    csv.Read();
    csv.ReadHeader();
    while (csv.Read())
    {
        var record = csv.GetRecord<OneDriveDetails>();
        string userEmail = record.OwnerPrincipalName;
        int oneDriveFiles = int.Parse(record.FileCount);
        int activeFileCount = int.Parse(record.ActiveFileCount);
        double storageUsedOnedrive = double.Parse(record.StorageUsedByte);

        storageUsedOndriveMB = Math.Round(storageUsedOnedrive / 1000000, 2);

        //int storageInBytes = storageUsedOnedrive;

        compOneDriveFiles += oneDriveFiles;
        compOneDriveActiveFiles += activeFileCount;
        compTotalDriveSize += storageUsedOndriveMB;
```

Fig 3.28 - This figure shows how we use a streamreader together with CSVReader to map csv headers to objects.

We then take the values collected from the CSV file and insert the data into the database using ORM. This is done by first acquiring a database session from "DSAContext" which returns an object containing the columns and rows of the database that we want to update. Then the values of the object returned by the database is updated with the values collected from the CSV file. Finally the updated object is inserted back into the database with the new values.

```
public async Task<IActionResult> updateUserTable(User user)
{
    try
    {
        var updateUser = _dsaContext.users.Where(u => u.email == user.email).ToList();
        foreach(var oneDriveUser in updateUser) {
        oneDriveUser.OneDriveFiles = user.OneDriveFiles;
        oneDriveUser.OneDriveActiveFiles = user.OneDriveActiveFiles;
        oneDriveUser.TotalDriveSize = user.TotalDriveSize;
    }
        await _dsaContext.SaveChangesAsync();
    }
    catch(Exception e)
    {
        logger.LogError(e, e.Message);
    }
    return new OkObjectResult(true);
}
```

Fig 3.29 - This method shows how we use a “User” object to update values in the SQL server database.

3.4.1.1 HTTP Triggered Azure Functions

We have used HTTP triggers to be able to invoke our functions through HTTP requests. When a request is sent the function is triggered and takes the body of the request to use further on. The function “getUserData” expects the request body to contain an email in the format of a JSON object that will then be used to query for data related to the email.

```
[FunctionName("GetUserData")]
  0 references
  public async Task<IActionResult> Run(
  [HttpTrigger("POST")] HttpRequest req,
  ILogger log)
```

Fig 3.30 - The highlighted part shows HttpRequest which is used to read the request body of incoming requests.

We then use a StreamReader that reads each of the lines in the request body. In our case the request only contains one line for the StreamReader to go through. The data from streamreader then has to be deserialized.

What deserializing does is to convert the JSON object received in the request to a string so that it can be used later on in the code. Finally the email is extracted and stored in a string variable.

```
var stream = await new StreamReader(req.Body).ReadToEndAsync();
log.LogInformation("");
dynamic data = JsonConvert.DeserializeObject<dynamic>(stream);
string email = data?.email?.ToString();
```

Fig 3.31 - Figure shows how we read body of request, deserialize body and then store body in a variable string

We then use the email to query to the database. The database columns and rows are then serialized, meaning that they are converted from strings to JSON objects, and returned to the location where the HTTP request was sent from.

```
var teamsUser = _dsaContext.users.Where(u => u.email == email);
string serializedObject = JsonConvert.SerializeObject(teamsUser, Formatting.Indented);
log.LogInformation(serializedObject);
return new OkObjectResult(serializedObject);
```

Fig 3.32 - The main highlight of the figure shows where we query to the database. And serialize the object before returning it.

3.4.1.2 Azure Functions Co2 Algorithms - Outlook

To be able to calculate how much CO2 each of the Microsoft 365 services produces we first had to find some sources that we could base our calculations on.

Outlook is primarily an email service provider so we wanted to get a general idea of how much Co2 is produced by sending and receiving email. We discovered that a single email can produce different quantities of CO2 depending on the type of email that is sent. Spam emails produce the least amount of CO2 while longer emails containing images and text produce more CO2.(Walkley, 2022)

Email type	Emissions(CO2)
Spam email	0,03 g/CO2
Short email sent by phone	0,2 g/CO2
Short email sent by laptop	0,3 g/CO2
Long email sent on a laptop	17 g/CO2

Long email containing images	26 g/CO2

The report we collect through the use of Microsoft Graph does not tell us the size of the email so we have to deal with averages when calculating the amount of CO2 produced by a user's Outlook. After summarizing the types of email we got an average of 8.706 grams of CO2 produced by one email.

The Azure Function is Time triggered in a similar fashion to previously mentioned Azure Functions and uses the average CO2 produced by an Email to create the results. The average CO2 produced by an email is placed in the “local.settings.json” file and retrieved through the use of a ConfigurationBuilder.

```
var config = new ConfigurationBuilder()
        .SetBasePath(context.FunctionAppDirectory)
        .AddJsonFile("local.settings.json", optional: true, reloadOnChange: true)
        .AddEnvironmentVariables()
        .Build();
double gramsOfCo2 = double.Parse
    (config["MailCo2ProductionGrams"],
    CultureInfo.InvariantCulture);
```

Fig 3.33 - ConfigurationBuilder is used to fetch environment variables located in local.settings.json

Then we use the Database context to retrieve the user table which contains the previously collected data about each user's sent and received mail. The total amount of emails in the user’s inbox is then multiplied with the average amount of CO2 produced by a single email. Finally the calculated value is inserted into the database. The same process is used to also calculate the CO2 produced by Outlook in both Department and Company tables.

```
int totalMailbox = (int)(user.totalEmailsSent + user.totalInboxmail);
double Co2Produced = totalMailbox * gramsOfCo2;
Co2Produced = Math.Round(Co2Produced, 2);
```

Fig 3.34 - Shows the simple calculation of Outlook Co2

3.4.1.3 Azure Functions Co2 Algorithms - OneDrive & SharePoint

OneDrive and SharePoint are both cloud based services that are used to store files and sites. Therefore we have used the same calculation for both of the Azure Functions.

Using a cloud based service to store files requires a lot more energy than storing files on a personal harddrive. In comparison, storing 1 GB of data on a personal harddrive requires about 0,00005 kWh while storing the same size in the cloud requires 7 kWh. It is estimated that having stored 100 Gigabytes worth of files in the cloud produces about 200 Kg of CO2 in a year(Adamson, 2015). We scaled down this calculation so it would show weekly CO2 production in grams because it would fit better with smaller user OneDrives.

Formula:
OneDrive calculation:

- User's Total OneDriveSize(in Megabytes) * Co2PerMegabyte / 52

SharePoint calculation:

- Company total Sharepoint Size (in Megabytes) * CO2PerMegabyte / 52

We then used the same approach as documented in Chapter 3.5.1.2 to fetch the user data, calculate CO2 and insert the calculated value back into the database.

3.4.1.4 Azure Functions Co2 Algorithms - Teams

Microsoft Teams works slightly differently than the other functions, as unlike OneDrive, Outlook or Sharepoint, there isn't a sum of files, emails or sites to calculate how many are stored or sent/received on that Microsoft 365 service. Instead, this Function retrieves a report on the user's activity within the platform, the aptly named "getTeamsUserActivityUserDetail" call from Graph API. This call retrieves a complete report in the form of a CSV file with data saved in headers, including: call count, meeting count, teams chat message count, private chat message count, teams audio duration, teams video duration and teams screenshare

duration, among other various results that we didn't find to be relevant for this calculation.

The CO2 calculation algorithm uses the activity data to calculate the projected emissions based on the time spent in Teams calls. Since we can't directly calculate carbon emissions per minute of Teams call or per Gigabyte of data consumed, we have researched online and discovered a formula that actually gives up actionable results. The time spent in a call can be multiplied with an average data consumption rate, times the amount of participants in the call. This sum can then be passed through the conversion rate of network traffic (in gigabytes) to electricity produced (in Kilowatts per hour (KWH)). Finally, KWH values can be directly calculated into the amount of kilograms CO2 emitted per kilowatt of energy generated. (Mytton, David. 2020).

Teams functions variables	Local.settings.json values
Audio GB consumed/hour	0.08
Video GB consumed/hour	0.54
Screenshare GB consumed/hour	0.40
Kilowatt energy produced/GB of data	5.00
Kilograms CO2/Kilowatt and GB of data	0.25358

Using variables saved in "local.settings.json", that can be later updated by the tenant to suit different environments or potential changes in energy efficiency in the future, we have written the following algorithm code for calculating our Teams emissions.

```
// retrieve user's time spent in Teams call in second, or get default zero time
double userAudio = (double)user.TeamsAudioDuration.GetValueOrDefault();
// Time measured in seconds, convert to hour
double userAudioHours = userAudio / 3600;
// Hours spent in call * average GB/hour consumption of online audio call * number of participants
double userDataUsed = (userAudioHours * audioGBuse) * callMembers;
// Amount of data used * Conversion rate of GB to kWh
double GBtoKilowattPerHour = userDataUsed * kilowattGB;
// Calculated GB/kWh used * emissions factor in kgCO2, converted to grams
double CO2produced = GBtoKilowattPerHour * kgCO2perKWH * 1000;
// round decimals to nearest two decimal points
double AudioCO2 = Math.Round(CO2produced, 2);
double AudioTime = Math.Round(userAudioHours, 2);
```

Fig 3.35 - Teams CO2 Algorithm

This example is showcasing the method of calculating the audio consumption and carbon footprint, but identical methods are used for calculating video and screen-share functions as well, with changes to the variables used, respectively.

3.4.1.5 Authorization

In the Teams application we have a page that will only be available for admin Users to view. Therefore we have split users into two different groups. One admin group and one normal user group. Since the Teams application has no login. We have to authenticate using the “global admin” group that exists within the tenants collection of Microsoft 365 Groups. When inserting users into the database we check if the Azure Active Directory Id of the user is equal to one of the Id’s of the user’s within the admin group. If this is the case, the user is inserted to the database with the “adminCheck” attribute being true. If the user does not exist within the group the “adminCheck” will be false. How this is used to authorize in the Teams application can be seen in chapter 3.3.3.4 Company

3.4.2 Azure SQL server

In the early stages of the project we were advised by our supervisor to use an Azure SQL database for our project because it provides a cloud based service that will function well together with our serverless Azure functions. It is a relational database which means that the data is organized within rows and columns that exist within tables. Our database solution for this project is a database located within the same Azure App Directory as the Azure Functions. It consists of four tables, users,

departments, company and tooltips, with the latter being left out of the project due to time constraints.

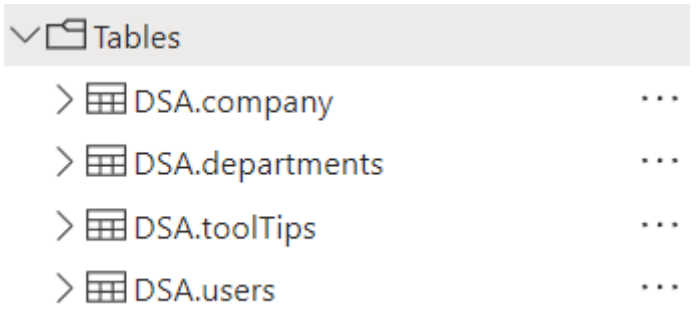


Fig 3.36 - The figure above shows the four tables that we have within our ‘DSA’ schema

We developed a very simple database structure that will fulfill our needs for the project. We have a User table with a foreign key to the Department table and a Department table with foreign key to the Company table. This structure meets the needs of our application as it correlates with the three pages we will have in the Teams application.

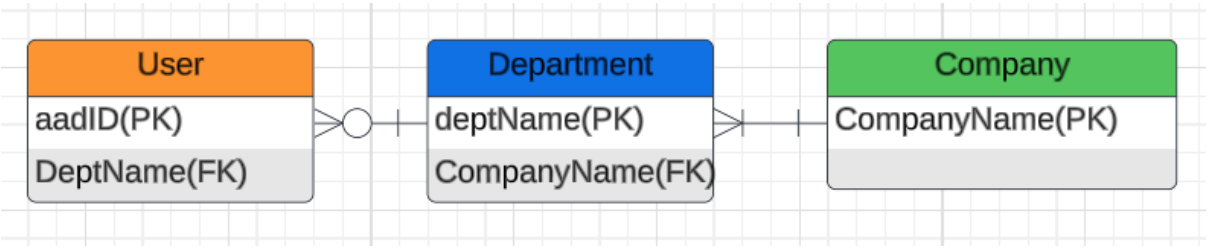


Fig 3.37 - The structure of the database

3.4.2.2 Entity Framework Core

Entity Framework Core is an ORM tool that is used to work with relational databases through the use of C# objects. To utilize Entity Framework Core we first have to create a connection between the relational database and the data access layer. This is done through the use of a connection string. The connection string is what establishes a connection between the application, in our case the Azure Functions,

and the SQL server database. The string is located in “local.settings.json” and is used in the code though “GetEnvironmentVariable”.

```
public override void Configure(IFunctionsHostBuilder builder)
{
    string connection = Environment.GetEnvironmentVariable("sqldb_connection");
    builder.Services.AddDbContext<DSAContext>(
        options => SqlServerDbContextOptionsExtensions.UseSqlServer(options, connection));
}
```

Fig 3.38 - Figure shows our startup.cs class where we utilize dependency injection to create a connection between database and Azure Function

Once a connection is established we need a class that extends Dbcontext which is used to map the database tables to objects. The context class contains three objects that each represent an entity within the database. These objects are “users”, “departments” and “company”. Inside these objects are methods that mimic each attribute in the database.

```
25 references
public class DSAContext : DbContext
{

    0 references
    public DSAContext(DbContextOptions<DSAContext> options) : base(options) { }
    15 references
    public DbSet<User> users { get; set; }
    14 references
    public DbSet<Department> departments { get; set; }
    12 references
    public DbSet<Company> company { get; set; }


}
```

Fig 3.39 - This shows our DSAContext class that extends DbContext to allow us to have a session of the SQL database available in the code.

Within the company object we have multiple methods that mirror the attributes of the database. E.g “compOneDriveFiles” has to be an integer because the database attribute expects the value to be an integer. In addition we use Data annotations to tell the context what abilities the attributes will have. In the figure below we use the [Key] annotation on the company method to annotate that the attribute has the properties of a primary key. Meaning that the value given to the attribute must be unique.

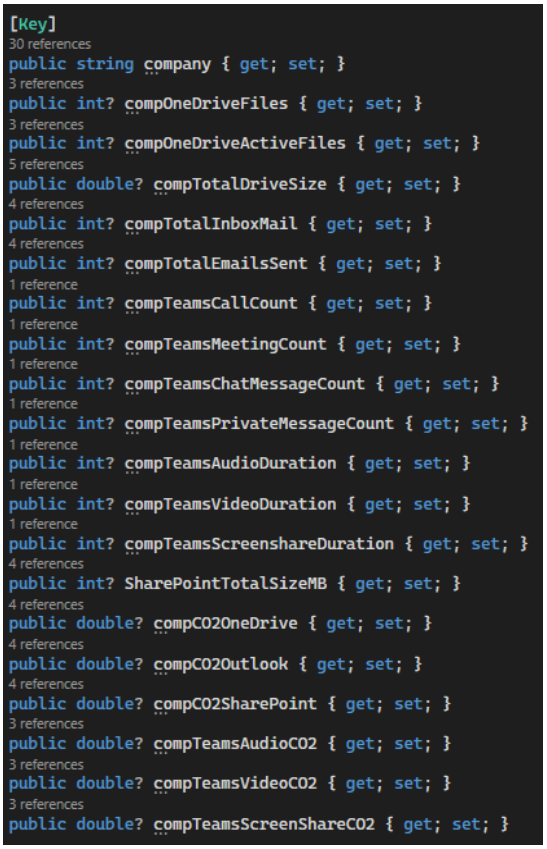


Fig 3.40 - This figure shows our Company.cs class that is used to map database attributes to objects

There are also cases where the user we collect data from has no data that correlates with the database attribute. Because of this we have allowed for the values to be null in such cases.

3.5 Test Documentation

Our primary goal for conducting the tests was to get an impression as to how user friendly and easy to understand our application is. To do this we did some user testing with people from different demographic backgrounds to see if our application is well understood by people from differing age brackets. After each user test was completed we asked them some questions and gave them a survey where they could give some feedback and rate the application based on how understandable and easy to navigate it was for them.

We also conducted some API testing using PostMan to see if our Azure Functions worked as expected both on localhost as well as after being published to Azure

3.5.1 API Testing

We tested the Azure Functions both before and after publishing them to Azure. To do this we used Postman, a platform for creating and testing API's. We created a collection where we could test each individual API to see if they worked as expected.

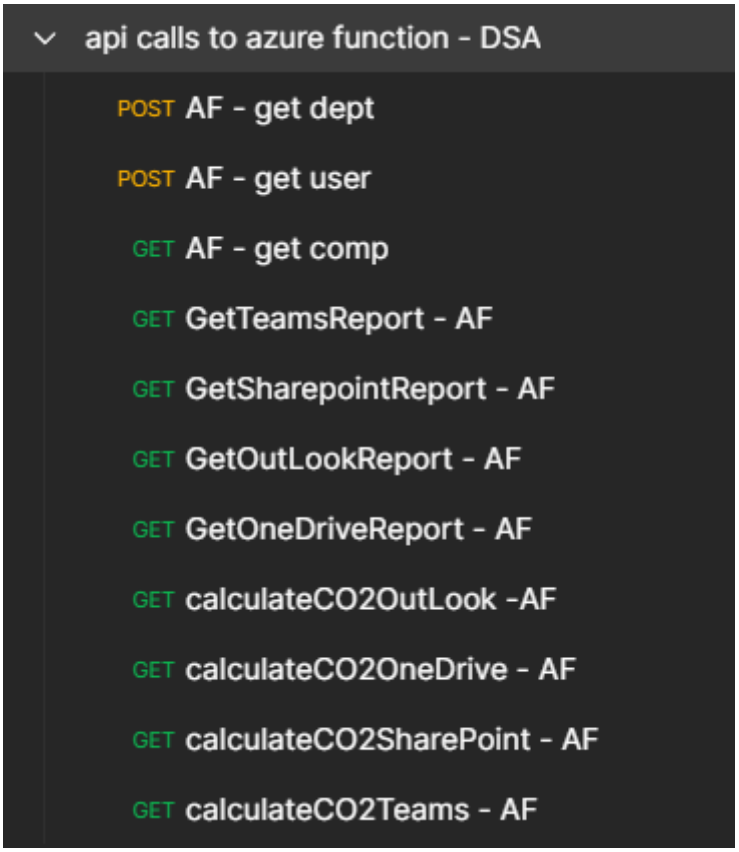


Fig 3.41 - Shows a collection in postman of API's we are testing

Some of the tests require that the API request contains a message body. In these cases we have to manually input a message body.

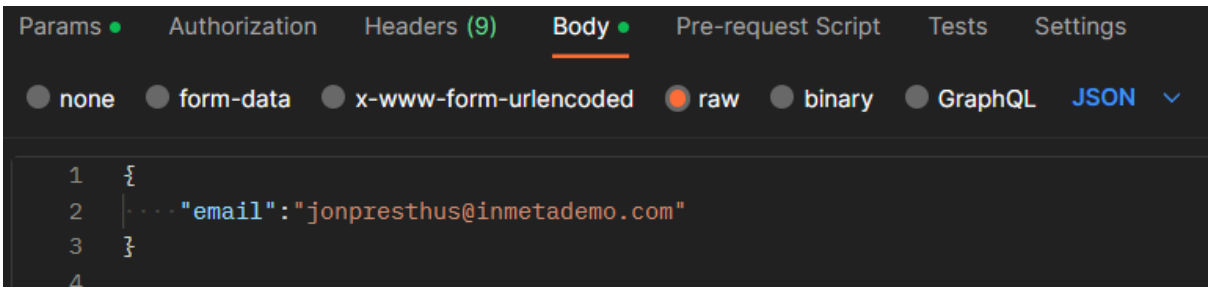


Fig 3.42 - Figure shows the body of the API request containing a email for testing

If the API request is successful it will return with the status code 200 OK as well as duration and Size of the received package.

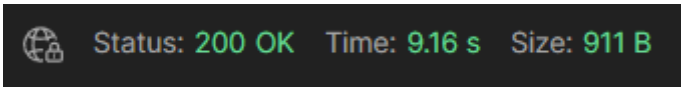


Fig 3.43 - Shows that a test results in success

If there are problems with the API or request we will receive the error code “500 Internal Server Error”. This can happen if the message body contains values or if the server encounters an unexpected condition.

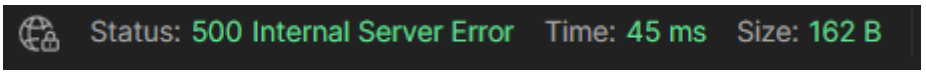


Fig 3.44 - Shows what postman reports if the test fails

Each of the Functions were tested both before and after deployment and in both cases the success rate was 11/11 successful tests.

3.5.2 User testing

The standard for user test completeness was established so that each member of the group was required to perform at least two user tests to meet the requirements. In addition, the person being tested should belong to a different age group. The test participants were tasked with testing the application in its entirety, and a thorough effort was made to follow all the test plan's instructions.

The test was performed in a predetermined order, and participants were given instructions at each step of the process. This ensured that the results of the test were comparable across the board. During this time, notes were taken, and later, the people tested were questioned about their general perception of the situation as well as whether they had any ideas on how certain aspects could be improved. We have two instructions as following:

DSA user test	DSA Admin test
Open application	Open application
Gå til user side	Go to the user page and review the reports intended for other users (employees).
Go to the department page	Go to department and review the reports intended for other departments
Go to the company	Go to Company page and check to have access
Use the cursor to hover over the charts and cards in the user page	Use the cursor to hover over the charts and cards in the user page.

3.5.2.1 Test Person 1 (Man 54 years old)

The test person had no IT background and he followed all steps from the test instructions and completed them all without any problems. He thinks it was easy to navigate between all the pages in the DSA application.

3.5.2.2 Test Person 2 (Man 40 years old):

The test person had some problems navigating around the DSA application. He thought that we had a button that could be pressed to go to the next page, and it took him a long time to understand where the header for the department was, while he was on the user page.

3.5.2.3 Test Person 3 (Man 47 years old):

The test person did not have any background in IT but is well experienced with computers and digital applications. He could easily navigate the Teams application and completed all test instructions without issue. He liked the layout of the application and thought it gave useful information to the user.

3.5.2.4 Test Person 4 (Woman 48 years old):

The test person is not very tech-savvy. She could follow the test instructions with a little help, though she did not really understand the placement of the user, department and company buttons. The cards and graphs didn’t tell her much, as she doesn’t use Microsoft 365 services herself, so maybe a little explanation of what the data presents on the front-end page could be appreciated.

3.5.2.5 Test Person 5 (Man 23 years old):

Test person did not have any problems with navigating the web app and understanding the variety of cards and graphs. However he had some feedback regarding how we present the data to the user. He said that the data and graphs were too impersonal and felt that it needed something more to relate better to the data he was shown.

3.5.2.6 Test Person 5 (Man 22 years old):

The test subject did easily complete every task set out for him without any complications. He also said the charts and cards were easy to understand and gave his feedback through a survey we gave to him later.

Chapter 4 - User Manual

4.1 Manual for users

4.1.1 Nav-bar

The navigation bar is located at the top of the application and contains four clickable buttons that the user can use to navigate through the application. These buttons include: “Your Report”, “Your Department”, “Your Company” and “Run Forrest Run”. When clicking on one of the buttons the content below the navigation bar will change depending on which button was clicked.

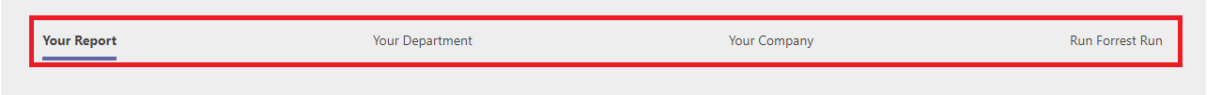


Fig 4.1 - Navigation bar

4.1.2 “Your Report” page

On the “Your Report” page you are presented with a series of cards and graphs that give you information about your usage of Microsoft 365 services and the CO2 that you have produced as a result of that usage. Here you can hover your mouse over the cards to get additional insights to your CO2 production. You can also use your mouse to hover over the charts if you want to get a highlight of the data points shown.

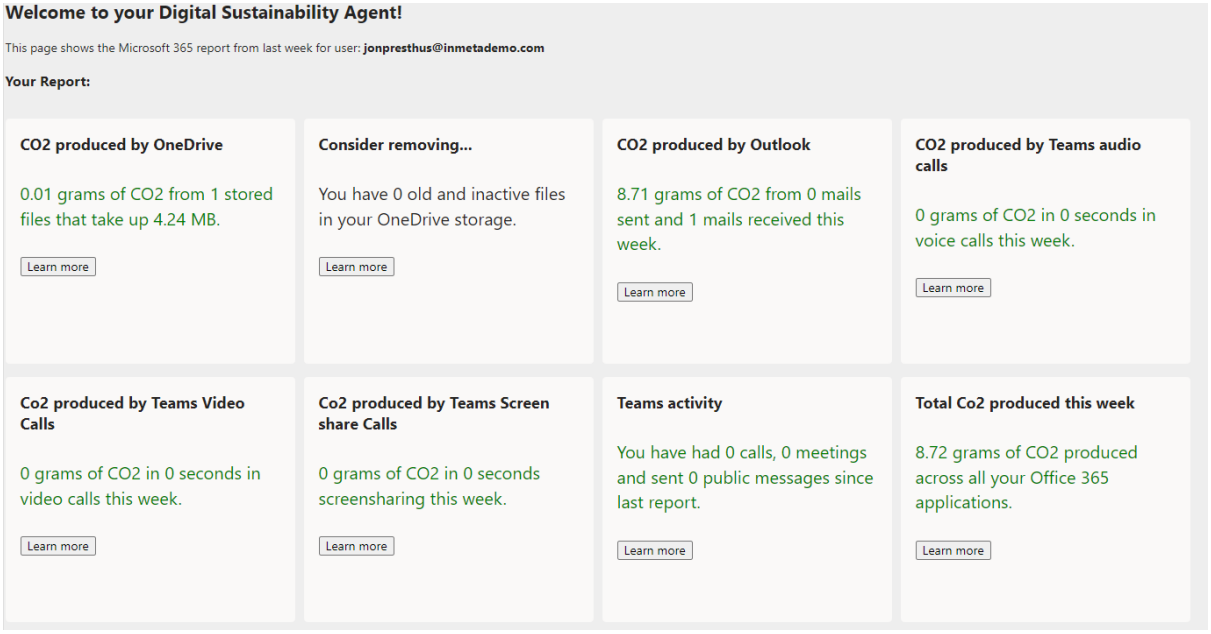


Fig 4.2 - Report cards

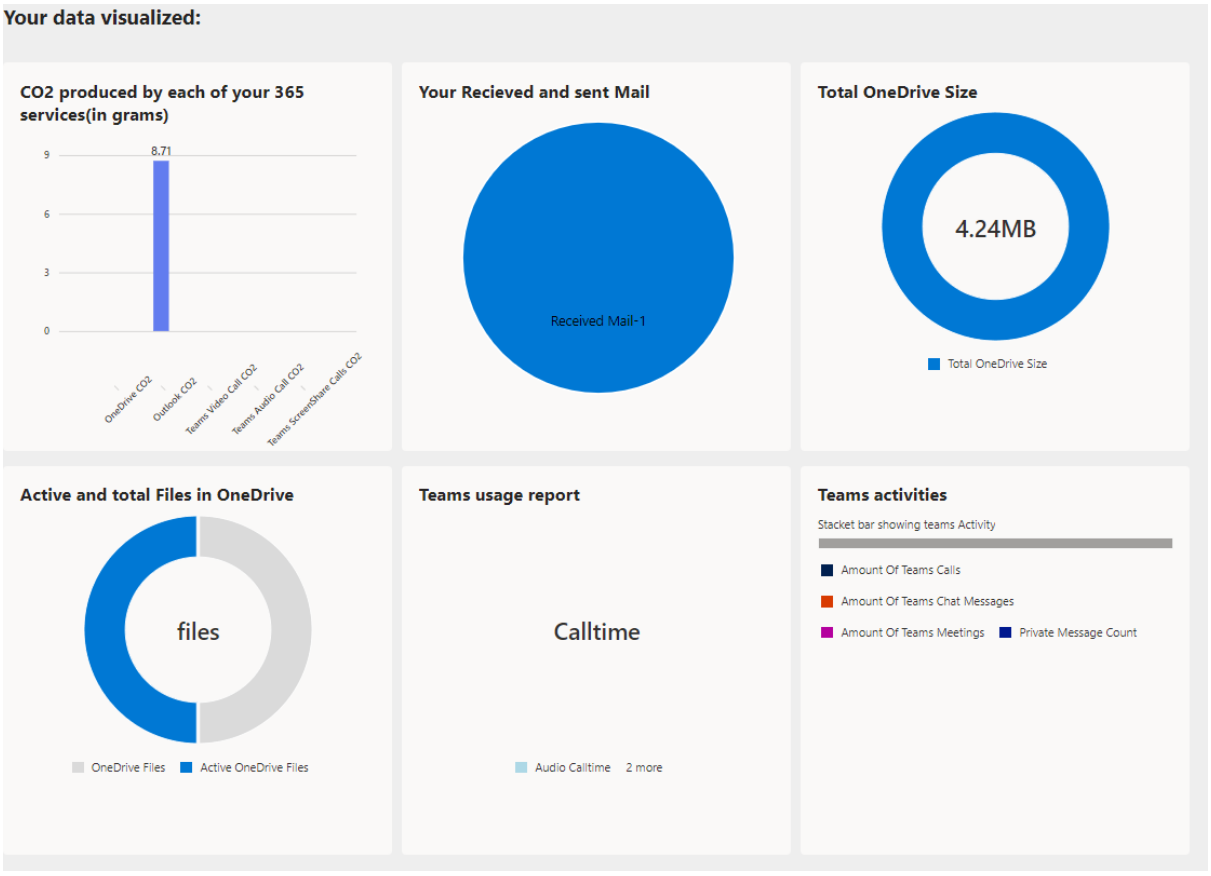


Fig 4.3 - Report graphs

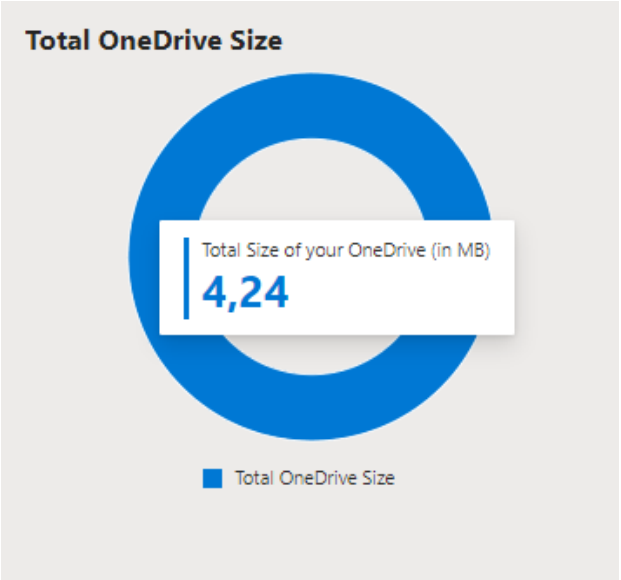


Fig 4.4 - Report graph with mouse-over description

4.1.3 “Your Department” page

The “Your Department” page is structured in a similar way to “Your Report”. It contains cards and graphs that you can hover over to get additional information about the subject the cards are displaying to you. However this page does not only contain data about the user, but about the users department.

4.1.4 “Your Company” page

The “Your company” page is unavailable to normal users as it contains data deemed accessible only to administrators. On this page you will be met with an HTTP status code error that says “403: Unauthorized” since you are not an authorized administrator.

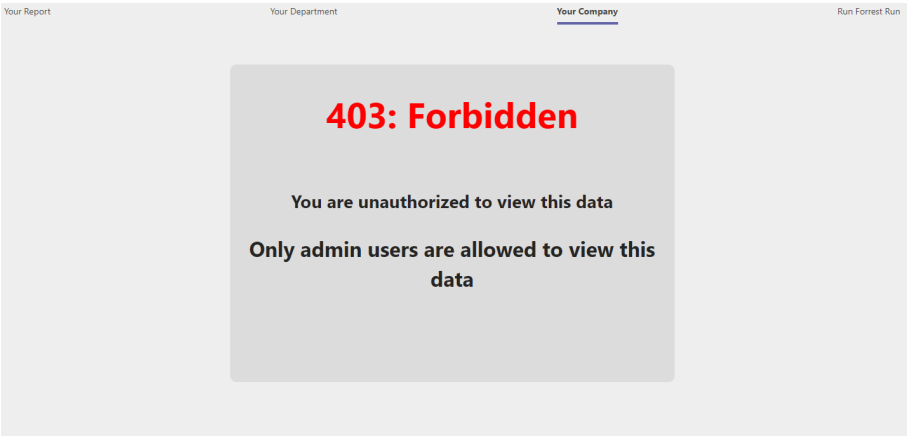


Fig 4.5 - Company page as seen by regular users

4.2 Manual for administrators

Administrators have more functionalities than a normal user as they are able to view other employees' data production, other departments and the "Your Company page"

4.2.1 Admin Panel - "Your Report"

As an administrator you are given an additional panel where you can search after users within your company. This panel is found below your own personal report. To use this you have to input the email of an employee within the same Microsoft tenant that you are in.

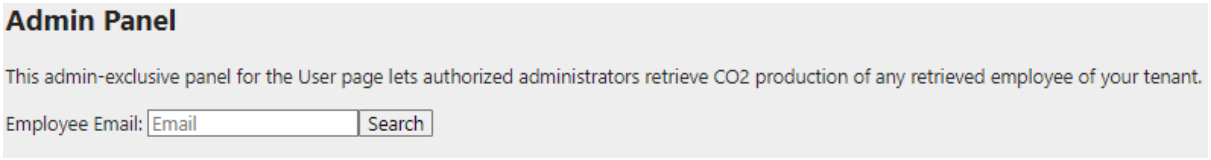


Fig 4.6 - Admin search panel

Once you have put the employee email in the input field you may click on the search button, and if the email is valid the data associated with the employee populates the cards and charts located below.

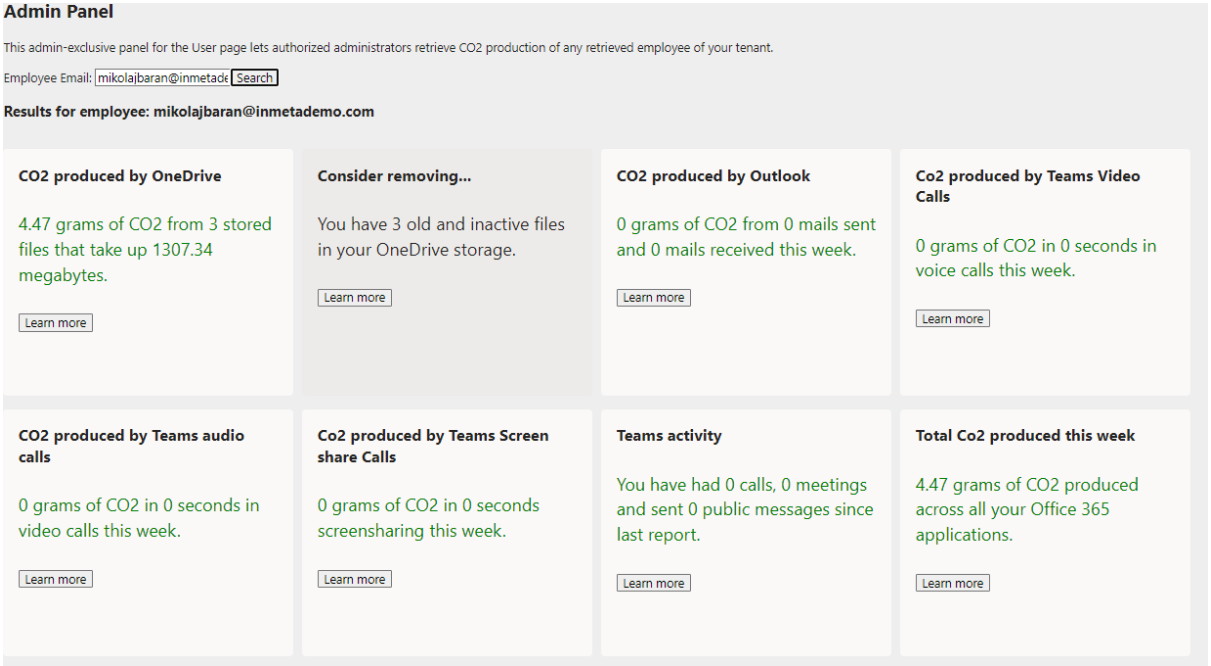


Fig 4.7 - Admin search result report

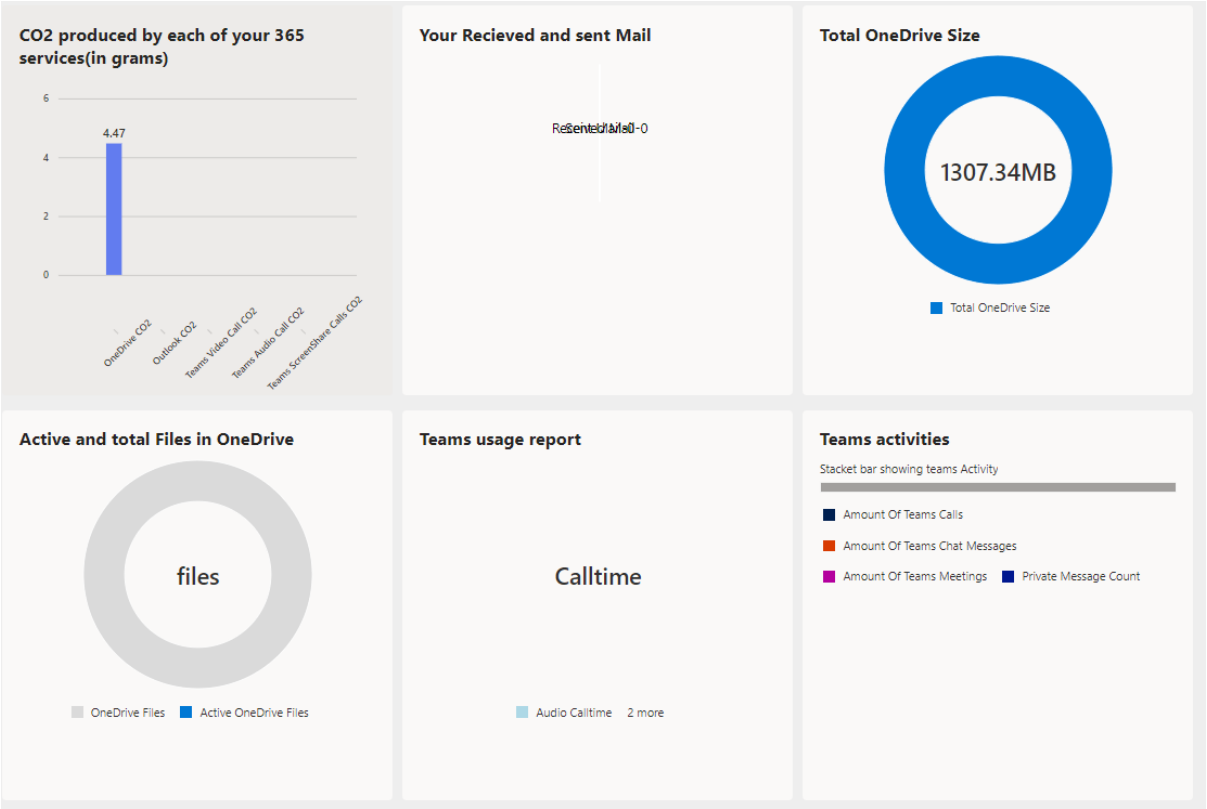


Fig 4.8 - Admin search result graph

4.2.2 Admin Panel - “Your Department”

The Admin panel within the “Your Department” page has the same functionality as the previously mentioned “Your Report”. The only difference is that on this page you may search after departments using the department name. After inputting a valid department name and clicking search, you will be presented with the data associated with the selected department.

4.2.3 Admin Panel - “Your Company”

When you are using the application as an administrator you are given access to the “Your Company” page. Here you will find data that summarizes every user and department to get a total overview over data and CO2 production within the company for every week.

Chapter 5 - Conclusion

5.1 Learning opportunities

This Bachelor's project has been a great learning experience. During this process we have gained much insight into how a "real-life" IT consulting company operates and conducts their projects, as well as learned a lot about industry technologies that are used to accomplish the goals of a project. Especially, since Inmeta Consulting AS primarily uses Microsoft technologies, we have learned much about Microsoft products and services.

5.2 Shortcomings

We finished the core functionality of the product solution and managed to deploy a working application. However, there were some goals defined in the requirements specifications that we were not able to implement during the project. The game showing how people in the company were ranked in comparison to each other based on carbon footprint was something that we were not able to accomplish within the given time constraint. In addition we were not able to integrate OpenAI into the Teams application due how new the technology is and our inexperience.

5.3 Future development

If we could give some recommendations to additional functionalities it would be the addition of historical data and gamification of the application. With historical data the user can compare current CO2 production to previous weeks and draw conclusions from that data. With the "game" implemented in the solution, it would create a general sense of cooperation and competition between the employees trying to reach the sustainability goals set by the company.
Another thing we would like to see implemented in the solution in the future, is to categorize and separate the data by each of the Microsoft 365 services we utilize in the application per page, for example OneDrive and all it's graphs and cards should be easily discernible from Outlook, Sharepoint or Teams.

5.4 Reflection

As developers, we have experienced significant professional growth through our work with the final product. It is important to note that the culmination of this project involved the use of a number of technologies that were initially unknown to us. Looking back, there are aspects that we would approach differently, if we had the opportunity to repeat the project. We acknowledge that the product's requirement specifications could have been further improved during the development process. For example, recording the stand-up meetings in greater depth would have provided valuable insight.

Despite the fact that the stand-up meetings were held less frequently throughout the process, it proved to be very beneficial when it came to preparing the product. Taking a step back and considering the project as a whole, we are genuinely pleased with the development of the process this semester and the final resulting outcome. We, as developers, will be able to look back on the project as a process of educational progress. We consider completing a project of this magnitude to be a significant accomplishment.

Bibliography

[1]
Fabio Duarte(2023, April 3) Amount of Data Created Daily (2023) Retrieved from: https://explodingtopics.com/blog/data-generated-per-day

[2]
Joshua Melvin (2015, November 26). *What is the carbon footprint of the emails?* Retrieved from: https://phys.org/news/2015-11-carbon-footprint-email.html

Microsoft (2023, November 28). *What is Agile development*. Retrieved from: What is Agile Development? - Azure DevOps | Microsoft Learn

[3]
Microsoft (2023, February 13). A tour of C# - Overview | Microsoft Learn. Retrieved from: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[4]
Andrea, Chiarelli(2021, October 15) *What is .NET? An Overview of the Platform.* Retrieved from: https://auth0.com/blog/what-is-dotnet-platform-overview/

[5]
Barbra, T(2022, January 31) *What is .NET Framework? Explain Architecture & Components.* Retrieved from: https://www.guru99.com/net-framework.html

[6]
Jay, L(2021, April 6) *What is Azure Functions?* Retrieved from: https://www.dynatrace.com/news/blog/what-is-azure-functions/

[7]
Microsoft (2022, October 19) *Introduction to Azure Functions.* Retrieved from: https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview

[8]
Sommerville, I. (2015). Software Engineering, Global Edition (p.79). Pearson Education Limited

[9]
Jim Campbell (2020, June 5). *Scrum Methodology.* Retrieved from: What is the Scrum Methodology? A Thorough Guide to the Scrum Framework (scrumexplainer.com)

[10]
Microsoft(2022, October 12) *An introduction to NuGet.* Retrieved from: https://learn.microsoft.com/en-us/nuget/what-is-nuget

[11]
Oracle (2023). *What is a Database?* Retrieved from: What Is a Database | Oracle

[12]
Microsoft (2022, October 10) *What is Azure DevOps?* Retrieved from: https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?toc=%2Fazure%2Fdevops%2Fget-started%2Ftoc.json&view=azure-devops

[13]
Microsoft(2020, October 4) *What is Azure Boards?* Retrieved from: https://learn.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops

[14]
Oracle (2023), *Autonomous JSON database.* Retrieved from: Autonomous JSON Database | Oracle

[15]
Chapple, Mike. (2021, November 18). *The Fundamentals of SQL*. Retrieved from: https://www.thoughtco.com/sql-fundamentals-1019780

[16]
Microsoft (2023, January 27) *Use the Microsoft Graph API.* Retrieved from: https://learn.microsoft.com/en-us/graph/use-the-api?source=docs

[17]
Scardina, J(2022, December 27) *Microsoft Power BI.* Retrieved from: https://www.techtarget.com/searchcontentmanagement/definition/Microsoft-Power-BI

[18]
Amazon Web Services. (2023, February 22) *What Is SQL (Structured Query Language).* Retrieved from: https://aws.amazon.com/what-is/sql/

[19]
Abba, Ihechikara(2022, October 21) *What is an ORM - The Meaning of Object Relational Mapping Database Tools.* Retrieved from: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/
What is an ORM – The Meaning of Object Relational Mapping Database Tools

[20]
Atlassian (2023, March 08) - *What is Git.* Retrieved from: https://www.atlassian.com/git/tutorials/what-is-git

[21]
Microsoft Office (2023, March 15), *What is PowerPoint?* Retrieved from: What is PowerPoint? - Microsoft Support

[22]
CIBO Technologies (2021, March 15) *Calculating the Carbon Footprint of Zoom Meetings.*

https://www.cibotechnologies.com/blog/calculating-the-carbon-footprint-of-zoom-meetings/#:~:text=The%20audio%2Donly%20calls%20would,emit%202.8%20kg%20of%20CO2.

[23]
Kingsley-Hughes, Adrian (2021, April 23) How much CO2 are your Zoom meetings generating? Retrieved from: https://www.zdnet.com/home-and-office/work-life/how-much-co2-are-your-zoom-meetings-generating/

[24]
Zoom Support (2023, February 17) *Zoom System Requirements: Windows, macOS, Linux. Bandwidth Requirements*. Retrieved from: https://support.zoom.us/hc/en-us/articles/201362023#h_d278c327-e03d-4896-b19a-96a8f3c0c69c

[25]
Mytton, David (2022, November 26) *Zoom, video conferencing, energy, and emissions.* Retrieved from: https://davidmytton.blog/zoom-video-conferencing-energy-and-emissions/

[26]
Cassidy, Alex (2020, September 17) *Carbon Gaming: The games polluting the planet.* Retrieved from: https://www.uswitch.com/gas-electricity/guides/carbon-gaming/

[27]
McGovern, Gerry (2020, March 9) *The Hidden Pollution Cost of Online Meetings.* Retrieved from:

https://www.cmswire.com/digital-workplace/the-hidden-pollution-cost-of-online-meetings/

[28]
Microsoft (2023, April 4) *reportRoot: getTeamsUserActivityUserDetail.* Retrieved from:
https://learn.microsoft.com/en-us/graph/api/reportroot-getteamsuseractivityuserdetail?view=graph-rest-1.0

[29]
PrivacyPolicies.com Legal Writing Team(2022, July 1) GDPR Compliance for Apps Retrieved 03.05.2023 from:
https://www.privacypolicies.com/blog/gdpr-compliance-apps/

[30]
European Commision, Do the data protection rules apply to data about a company? Retrieved 03.05.2023 from:
https://commission.europa.eu/law/law-topic/data-protection/reform/rules-business-and-organisations/application-regulation/do-data-protection-rules-apply-data-about-company_en

[31]
Simplilearn (2023, February 24), What is React, What is TypeScript and its advantages? TypeScript with React.
Retrieved from:
https://www.simplilearn.com/tutorials/reactjs-tutorial/react-typescript#what_is_typescript

[32]
UXPin (2023, May 5) *What is npm?* Retrieved from:
https://www.uxpin.com/studio/blog/what-is-npm/

[33]

Saltzer, Jerome H.; Schroeder, Michael D. (1975). "The protection of information in computer systems". Proceedings of the IEEE. Institute of Electrical and Electronics Engineers (IEEE). 63 (9): 1278–1308. doi:10.1109/proc.1975.9939. ISSN 0018-9219. OCLC 5871551104. S2CID 269166 .

[34]
Shona McCombes (2022, November 30)
Retrieved from:
https://www.scribbr.com/methodology/survey-research/

[35]
Microsoft (2023, May 9) Fluent UI
Retrieved from:
https://developer.microsoft.com/en-us/fluentui#/

[36]
Walkley, Sarah (2022, september) The Carbon Cost of an Email: Update! Retrieved from https://carbonliteracy.com/the-carbon-cost-of-an-email/ (05.11.2023)

[37]
Microsoft (Microsoft FluentUI, 2023 May 11), FluentUi react-charting Retrieved from:
https://fluentuipr.z22.web.core.windows.net/heads/master/react-charting/demo/index.html

[38]
UXPin (2023, april 3) Fluent UI vs MUI – Designer's Comparison Retrieved From:
https://www.uxpin.com/studio/blog/fluent-ui-vs-mui/ (05.12.2023)

[39]
Microsoft (2023, May 11), what is Power BI?
Retrieved from: https://powerbi.microsoft.com/en-us/what-is-power-bi/

[40]

Agile Alliance, What is agile? Retrieved from https://www.agilealliance.org/agile101/ (18.05.2023)

[41]
Adamson, Justin(2017, May 15) Carbon and the Cloud, Retrieved From: https://medium.com/stanford-magazine/carbon-and-the-cloud-d6f481b79dfe

[42]
Logan Mccoy (2019, March 6) What is Azure, retrieved May 22, 2023 from https://ccbtechnology.com/what-microsoft-azure-is-and-why-it-matters/

[43]
Uctoday (2022, October 7), What is Teams Toolkit for Visual Studio?, retrieved May 22, 2023 from https://www.uctoday.com/unified-communications/what-is-teams-toolkit-for-visual-studio/

[44]
Informatica (2023, May 22), What is data mapping? Retrieved May 22, 2023 from https://www.informatica.com/resources/articles/data-mapping.html

[45]
Nielsen Norman Group (2016, November 6), Cards: UI-Component Definition, Retrieved May 24, 2023 from https://www.nngroup.com/articles/cards-component/

[46]
Microsoft(2023,March 16) Overview of Microsoft Graph https://learn.microsoft.com/en-us/graph/overview

[47]
Kilgore, Georgette(2023, March 30) Carbon Footprint of the Internet Over Time Since 1990 (With Graphics) https://8billiontrees.com/carbon-offsets-credits/carbon-footprint-of-the-internet/

References

Abba, Ihechikara. (2022, October 21). *What is an ORM – The Meaning of Object Relational Mapping Database Tools*. freeCodeCamp.org. Retrieved May 19, 2023, from https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/

Agile Alliance. (2015, June 29). *What is Agile Software Development?* Agile Alliance |. Retrieved May 19, 2023, from https://www.agilealliance.org/agile101/

Amazon Web Services. (2023 May 19). *What is SQL? - Structured Query Language (SQL) Explained - AWS*. Amazon Web Services, Inc. Retrieved May 19, 2023, from https://aws.amazon.com/what-is/sql/

Andrea, Chiarelli. (2021 October 15). *What is .NET? An Overview of the Platform*. Auth0 - Blog. Retrieved May 19, 2023, from https://auth0.com/blog/what-is-dotnet-platform-overview/

Atlassian. (2023 May 19). *What is Git | Atlassian Git Tutorial*. Atlassian. Retrieved May 19, 2023, from https://www.atlassian.com/git/tutorials/what-is-git

Barbra, T. (2020, January 5). *What is .NET Framework? Explain Architecture & Components*. Guru99. Retrieved May 19, 2023, from https://www.guru99.com/net-framework.html

Cassidy Alex. (2020 September 17). *Carbon Gaming: The games polluting the planet | Uswitch*. Uswitch. Retrieved May 19, 2023, from https://www.uswitch.com/gas-electricity/guides/carbon-gaming/

Chapple, Mike. (2020, April 22). *The Fundamental Guide to SQL*. ThoughtCo. Retrieved May 19, 2023, from https://www.thoughtco.com/sql-fundamentals-1019780

CIBO Technologies. (2021, March 15). *Calculating the Carbon Footprint of Zoom Meetings » CIBO Technologies*. CIBO Technologies.

Retrieved May 19, 2023, from https://www.cibotechnologies.com/blog/calculating-the-carbon-footprint-of-zoom-meetings/

European Commission. (2023 May 19). *Do the data protection rules apply to data about a company?* European Commission. Retrieved May 19, 2023, from https://commission.europa.eu/law/law-topic/data-protection/reform/rules-business-and-organisations/application-regulation/do-data-protection-rules-apply-data-about-company_en

Fabio Duarte. (2023, March 16). *Amount of Data Created Daily (2023)*. Exploding Topics. Retrieved May 19, 2023, from https://explodingtopics.com/blog/data-generated-per-day

Informatica. (2023, May 10). *What is Data Mapping?* Informatica. Retrieved May 22, 2023, from https://www.informatica.com/resources/articles/data-mapping.html

Jim Campbell (2020, June 5). *Scrum Methodology.* Retrieved from: (May 15, 2023)
What is the Scrum Methodology? A Thorough Guide to the Scrum Framework (scrumexplainer.com)

Joshua Melvin. (2015, November 26). *What's the carbon footprint of an email?* What's the Carbon Footprint of an Email? Retrieved May 19, 2023, from https://phys.org/news/2015-11-carbon-footprint-email.html

Justin, A. (2017, June 27). Carbon and the Cloud. Medium. Retrieved May 19, 2023, from:
https://medium.com/stanford-magazine/carbon-and-the-cloud-d6f481b79dfe

Kilgore, G. (2022, December 8). *Carbon Footprint of the Internet Over Time Since 1990 (With Graphics)*. 8 Billion Trees: Carbon Offset Projects & Ecological Footprint Calculators.

https://8billiontrees.com/carbon-offsets-credits/carbon-footprint-of-the-internet/

Kingsley-Hughes, Adrian. (2021, April 23). *How much CO2 are your Zoom meetings generating?* ZDNET. Retrieved May 9, 2023, from https://www.zdnet.com/home-and-office/work-life/how-much-co2-are-your-zoom-meetings-generating/

Livens, J. (2021, April 6). *What is Azure Functions?* Dynatrace News. https://www.dynatrace.com/news/blog/what-is-azure-functions/

McCoy, L. (2019, March 6). *Microsoft Azure Explained: What It Is and Why It Matters*. CCB Technology. https://ccbtechnology.com/what-microsoft-azure-is-and-why-it-matters/

McGovern, Gerry . (2020, March 9). *The Hidden Pollution Cost of Online Meetings*. CMSWire.com. Retrieved May 19, 2023, from https://www.cmswire.com/digital-workplace/the-hidden-pollution-cost-of-online-meetings/

Microsoft FluentUi react-charting. (2023, May 1). *Fluent UI React Examples*. Fluent UI React Examples. Retrieved May 1, 2023, from https://fluentuipr.z22.web.core.windows.net/heads/master/react-charting/demo/index.html

Microsoft MSGraphDocsvTeam. (2023, January 27). *Use the Microsoft Graph API - Microsoft Graph*. Use The Microsoft Graph API - Microsoft Graph | Microsoft Learn. Retrieved May 19, 2023, from https://learn.microsoft.com/en-us/graph/use-the-api

Microsoft Office. (2023, April 24). *What is PowerPoint? - Microsoft Support*. What Is PowerPoint? - Microsoft Support. Retrieved April 24, 2023, from https://support.microsoft.com/en-us/office/what-is-powerpoint-5f9cc860-d199-4d85-ad1b-4b74018acf5b

Microsoft Power BI. (2023, May 7). *What Is Power BI? Definition and Features | Microsoft Power BI*. What Is Power BI? Definition and

Features | Microsoft Power BI. Retrieved May 7, 2023, from https://powerbi.microsoft.com/en-us/what-is-power-bi/

Microsoft, O. (2023, May 14). *Home - Fluent UI*. Home - Fluent UI. Retrieved May 14, 2023, from https://developer.microsoft.com/en-us/fluentui#/

Microsoft. (2022, November 28). *What is Agile Development? - Azure DevOps*. What Is Agile Development? - Azure DevOps | Microsoft Learn. Retrieved May 19, 2023, from https://learn.microsoft.com/en-us/devops/plan/what-is-agile-development

Microsoft. (2022, October 10). *What is Azure DevOps? - Azure DevOps*. What Is Azure DevOps? - Azure DevOps | Microsoft Learn. Retrieved May 19, 2023, from https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops

Microsoft. (2022, October 11). *What is NuGet and what does it do?* What Is NuGet and What Does It Do? | Microsoft Learn. Retrieved May 19, 2023, from https://learn.microsoft.com/en-us/nuget/what-is-nuget

Microsoft. (2022, October 7). *reportRoot: getTeamsUserActivityUserDetail - Microsoft Graph v1.0*. reportRoot: Microsoft Learn. Retrieved May 19, 2023, from https://learn.microsoft.com/en-us/graph/api/reportroot-getteamsuseractivityuserdetail

Microsoft. (2023, April 14). *Azure Functions Overview*. Azure Functions Overview | Microsoft Learn. Retrieved May 19, 2023, from https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview

Microsoft. (2023, April 28). *What is Azure Boards? Tools to manage software development projects. - Azure Boards*. What Is Azure Boards? Tools to Manage Software Development Projects. - Azure Boards | Microsoft Learn. Retrieved May 19, 2023, from

https://learn.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards

Microsoft. (2023, March 16). Overview of Microsoft Graph. Retrieved May 25, 2023, from https://learn.microsoft.com/en-us/graph/overview

Microsoft. (2023, March 16). *Overview of Microsoft Graph*. Retrieved May 25, 2023, from https://learn.microsoft.com/en-us/graph/overview

Microsoft. (2023, May 4). *A tour of C# - Overview*. A Tour of C# - Overview | Microsoft Learn. Retrieved May 19, 2023, from https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Mytton, David. (2020, November 16). *Zoom, video conferencing, energy, and emissions*. David Mytton. Retrieved May 19, 2023, from https://davidmytton.blog/zoom-video-conferencing-energy-and-emissions/

Nielsen Norman Group. (2016, November 6). *Cards: UI-Component Definition*. Retrieved May 24, 2023, from https://www.nngroup.com/articles/cards-component/

Oracle . (2023, May 9). *Oracle Autonomous JSON Database*. Autonomous JSON Database | Oracle. Retrieved May 9, 2023, from https://www.oracle.com/autonomous-database/autonomous-json-database/

Oracle . (2023, May 9). *What is a database?* What Is a Database | Oracle. Retrieved May 9, 2023, from https://www.oracle.com/database/what-is-database/#relational

Sarah Walkley. (2022, September 17). *The Carbon Cost of an Email: Update! - The Carbon Literacy Project*. The Carbon Literacy Project - Relevant Climate Change Learning for Everyone. Retrieved May 19, 2023, from https://carbonliteracy.com/the-carbon-cost-of-an-email/

Scardina. (2022, December 1). *What is Microsoft Power BI? | Definition from TechTarget*. Content Management. Retrieved May 19, 2023,

from https://www.techtarget.com/searchcontentmanagement/definition/Microsoft-Power-BI

Shona McCombes. (2019, August 20). *Survey Research | Definition, Examples & Methods*. Scribbr. Retrieved May 19, 2023, from https://www.scribbr.com/methodology/survey-research/

Simplilearn . (2023, May 9). *Guide To Using Typescript With React | Simplilearn*. Simplilearn.com. Retrieved May 9, 2023, from https://www.simplilearn.com/tutorials/reactjs-tutorial/react-typescript

Sommerville. I. (2015). *Software Engineering, Global Edition* (10th ed., Vol. 79). Person Education Limited.

Team, UCToday, U. T. (2022, October 1). *What is Teams Toolkit for Visual Studio?* UC Today. Retrieved May 22, 2023, from https://www.uctoday.com/unified-communications/what-is-teams-toolkit-for-visual-studio/

UXPin. (2022, April 27). *What is npm?* Studio by UXPin. Retrieved May 19, 2023, from https://www.uxpin.com/studio/blog/what-is-npm/

UXPin. (2023, April 3). *Fluent UI vs MUI – Designer's Comparison*. Studio by UXPin. Retrieved May 19, 2023, from https://www.uxpin.com/studio/blog/fluent-ui-vs-mui/

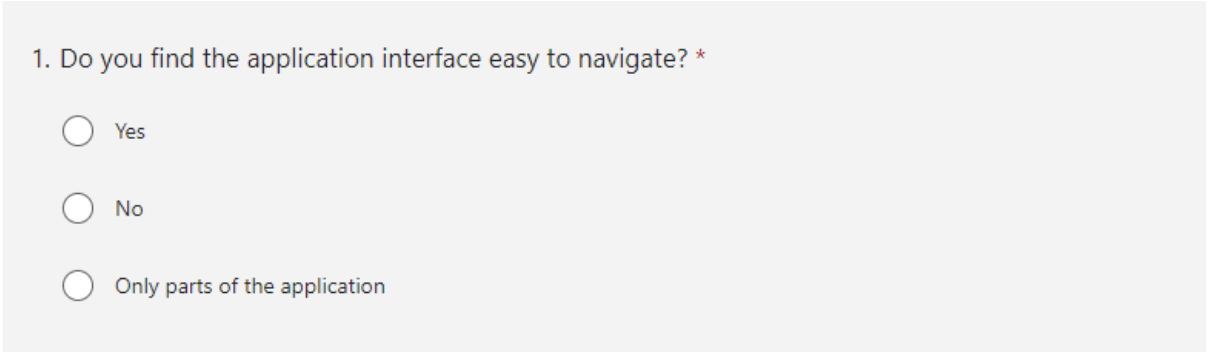
Zoom Support. (2023, February 17). *Zoom System Requirements: Windows, macOS, Linux*. https://support.zoom.us/hc/en-us/articles/201362023#h_d278c327-e03d-4896-b19a-96a8f3c0c69c

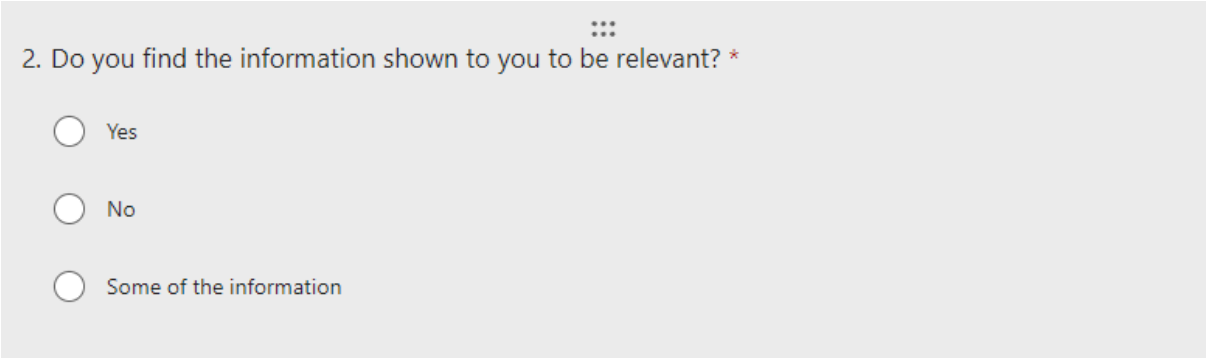
Appendix

The appendix includes additional files and code that may help the reader get a better understanding of the report, but it is technical and goes more in-depth on how the code and all functions of the application work. It is not mandatory to read the appendix.

Microsoft Forms

After conducting the user tests the participants were given a form to fill out that relates to their experience and opinions of the application. This form includes 10 questions that the user will answer to evaluate the experience they have with the application.





3. How likely are you to make changes to how you utilize Microsoft 365 services after finding out how much Co2 each service produces? *

- ◯ Very likely
- ◯ Somewhat likely
- ◯ Neither likely nor unlikely
- ◯ Somewhat unlikely
- ◯ Very unlikely

4. Will you use the app to get a better overview over own carbon production *

- Yes
- Maybe
- No

5. If you answered no to question 4. Can you give some feedback that might increase your likelyhood of using the app?

Skriv inn svaret

6. Is sustainability, both in private and public life, something that you value?

- ◯ Always
- ◯ Very often
- ◯ Often
- ◯ Maybe
- ◯ Seldom
- ◯ Never

7. Would you recommend others to use DSA? *

- ◯ Yes
- ◯ No
- ◯ Maybe

8. If you could rate this app in relation to how understandable the data shown is. How would you rate it? *

☆ ☆ ☆ ☆ ☆

9. Are the charts/graphs shown easy to understand? *

- ◯ Very easy to understand
- ◯ Somewhat easy to understand
- ◯ Neither easy nor hard to understand
- ◯ Hard to understand
- ◯ Very hard to understand

10. What are your thoughts about the application? *

Skriv inn svaret