HTL - Perg
Höhere Abteilung für Informatik

Diplomarbeit

Share:D

Projektteam: Noah Grillenberger
Michael Hintermayr
Paul Lanzinger
Paul Oprea

Projektbetreuerin: Prof[in]. OStR[in] Mag[a]. Gabriela Danner

In Zusammenarbeit mit Porsche Informatik Gesellschaft m.b.H.

Betreuer Herr Reiter Andreas

Bearbeitungszeitraum: 10.07.2023 – 03.04.2024

1. Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Perg, 03.04.2024 Unterschrift

(Noah Grillenberger)

Signiert von: Noah Kilian Grillenberger
Datum: 03.04.2024 15:11:15
TRUST
Dieses Dokument ist digital signiert!
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.
Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

Perg, 03.04.2024 Unterschrift

(Michael Hintermayr)

Signiert von: Michael Hintermayr
Datum: 03.04.2024 11:25:55
TRUST
Dieses Dokument ist digital signiert!
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.
Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

Perg, 03.04.2024 Unterschrift

(Paul Lanzinger)

Paul Lanzinger
Digital unterschrieben von Paul Lanzinger
Datum: 2024.04.03 10:58:55 +02'00'

Perg, 03.04.2024 Unterschrift

(Paul Oprea)

Signiert von: Paul Alexander Oprea
Datum: 03.04.2024 11:33:34
TRUST
Dieses Dokument ist digital signiert!
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.
Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

2. Gendererklärung

Die in dieser Diplomarbeit verwendeten Personenbezeichnungen beziehen sich immer gleichermaßen auf weibliche und männliche Personen. Auf eine Doppelnennung und gegenderte Bezeichnungen wird zugunsten einer besseren Lesbarkeit verzichtet.

Perg, 03.04.2024 Unterschrift Noah Grillenberger

(Noah Grillenberger)

Perg, 03.04.2024 Unterschrift Michael Hintermayr

(Michael Hintermayr)

Perg, 03.04.2024 Unterschrift Paul Lanzinger

(Paul Lanzinger)

Perg, 03.04.2024 Unterschrift

(Paul Oprea)

3. Danksagung

An dieser Stelle möchten wir uns bei allen Personen bedanken, die uns bei der Entwicklung der Diplomarbeit unterstützt haben.

Wir möchten uns herzlich bei unseren Betreuern und Kollegen während des Praktikums bei der Porsche Informatik GmbH bedanken.

Hier gilt unser Dank besonders Herrn Alexander Haslauer, der uns während des Praktikums immer wieder zur Seite stand. Darüber hinaus möchten wir auch noch Tobias Mayr und Jürgen Ninaus unsere Wertschätzung aussprechen.

Ein besonderer Dank gilt auch unserer Diplomarbeitsbetreuerin Frau Mag. Danner, die uns während der gesamten Diplomarbeitsdauer für organisatorische Fragen zur Verfügung stand.

4. Kurzfassung

Share:D ist ein Web-Tool, das die Buchung von Shared Desks durch die Mitarbeiter vereinfacht.

An allen Standorten der Porsche Informatik befinden sich diese Shared Desks und können grundsätzlich von allen Angestellten gebucht werden. Diese werden oft von Kollegen verwendet, die im Home-Office Modell arbeiten, wenn diese in der Firma sind, aber auch von jenen, die zwischen den Standorten der POI wechseln und somit keinen fixen Platz an den anderen Standorten haben.

Außerdem haben die Mitarbeiter von Porsche Informatik auch viele Termine, bei denen sie schnell den Überblick verlieren können, deswegen bietet unser Web-Tool auch eine Übersicht der anstehenden Termine an. Zusätzlich gibt es hier auch die Möglichkeit ein neues Event anzulegen.

Das Tool soll außerdem für Führungskräfte statistische Auswertungen der getätigten Buchungen liefern, wie z.B. die Auslastung der verfügbaren Shared Desks. Dies soll unter anderem dabei helfen, das benötigte Platzangebot feststellen zu können.

5. Abstract

Share:D is a web tool that makes it easier for employees to book shared desks.

These shared desks are located at all Porsche Informatik sites and can be booked by all employees. These are often used by employees who work in the home office model, but also by those who move between locations of the Porsche Informatik and therefore do not have a fixed desk at the other office.

The employees of Porsche Informatik also have a lot of appointments therefore it is easy to lose track, which is why our web tool also offers an overview of upcoming events. You can also create a new event here.

The tool is also intended to provide managers with statistical analyses of the bookings made, such as the utilization of the available shared desks. Among other things, this should help to determine the amount of space required.

The tool also offers the option of displaying and adding Outlook events.

6. Inhaltsverzeichnis

7. Einleitung

7.1 Auftraggeber

Unser Auftraggeber ist die Porsche Informatik GmbH. Hierbei handelt es sich um ein Tochterunternehmen der Porsche Holding Salzburg die sich seit März 2011 unter dem Dach des Volkswagen Konzerns befindet.

Die Firma spezialisiert sich auf die Digitalisierung des Automobilhandels und kombiniert langjährige Expertise in IT und Autohandel mit einer umfassenden Vision für die digitale Transformation in der Branche. Eine flexible Systemlandschaft und ein anpassungsfähiges Geschäftsmodell ermöglichen eine kontinuierliche Entwicklung sowie die Umsetzung innovativer Geschäftsstrategien für die Kunden. Sie haben bereits 180 Lösungen in 32 Ländern auf 4 Kontinenten entwickelt. Sie greifen in ihren Lösungen auf bewährte State-of-the-Art-Technologien wie Java, TypeScript oder Cloud zurück. Zudem vertiefen sie sich in Bereichen wie künstliche Intelligenz, Big Data, Virtual Reality und Voice Interface. (vgl. Porsche Informatik: Das Unternehmen, 2024)

7.2 Ausgangslage

Die Firma Porsche Informatik GmbH hat in Österreich mehrere Standorte, darunter in Wien, Salzburg und Hagenberg weshalb manche Mitarbeiter öfters von Standort zu Standort wechseln. Außerdem gibt es Angestellte, die im Home-Office arbeiten und nur wenige Tage der Woche ins Büro kommen. All diese Mitarbeiter brauchen einen freien Arbeitsplatz, wenn sie ins Büro kommen.

Daher hat die Porsche Informatik das Shared Desk System eingeführt. Dies bedeutet, dass es Arbeitsplätze gibt, die keiner fixen Person zugeteilt sind, welche „Shared Desks“ genannt werden. In der Praxis sind mehrere dieser Plätze nebeneinander, was dann „Shared Desk Group“ genannt wird.

7.2.1 Problemstellungen

Bevor ein Mitarbeiter, der keinen fixen Platz hat, ins Büro kommt, muss er sich einen in einer Shared Desk Group buchen. Dies erfolgt bis dato über Outlook und die Buchung wird im Outlook Kalender eingetragen. Dies kann aber recht umständlich werden da der Mitarbeiter über Outlook nicht genau sehen kann, wo sich eine bestimmte Desk Group befindet. Außerdem ist es für Teamleiter schwierig herauszufinden, wann wer vor Ort ist und wann im Home-Office.

Ein ähnliches Problem haben auch die Führungskräfte im Unternehmen. Das genaue Platzangebot in den einzelnen Standorten, also ob es zu viel oder zu wenig ist, ist für sie schwer zu ermitteln, da die Anzahl der besetzten Plätze dauernd schwankt.

Die Porsche Informatik möchte außerdem die Mitarbeiter motivieren, vermehrt ins Büro zu kommen, da viele sich an das Home-Office Modell, das durch Corona entstanden ist, gewöhnt haben und dies bevorzugen. Eine Möglichkeit, dies zu bewerkstelligen ist die Planung von Events. Im Arbeitsalltag der Firma tut sich nämlich einiges, z.B.: gemeinsames Frühstück, Brunch, Besuch und Schulung von Kollegen aus anderen Standorten und Arbeitsbereichen, Spieleabend, Geburtstagsfeier und so weiter.

7.3 Allgemeine Zielsetzung

Aus diesen Gründen wurde beschlossen ein Tool mit dem Namen Share:D für die Porsche Informatik zu produzieren, das…

- Das Buchen von Desk Groups für die Mitarbeiter erleichtert.
- Eine Übersicht für Teamleiter bietet, wann seine Team-Mitglieder im Home-Office, im Büro, krank oder anderswo sind.
- Eine Übersicht über die Auslastung der Standorte und der einzelnen Teams bietet.
- Eine Übersicht von bevorstehenden Events in seinem Standort bietet. Jeder Mitarbeiter kann auch neue Events anlegen.

Das Buchen in Outlook muss wegen einer Konzernvorgabe auch weiterhin unterstützt werden, soll jedoch nicht mehr durch den Mitarbeiter, sondern durch Share:D durchgeführt werden. Bucht der Mitarbeiter erfolgreich einen Tisch über Share:D, wird diese Buchung automatisch in Outlook eingetragen.

7.4 Ausgangssituation der Implementierung

Das Projekt wurde nicht gänzlich im Rahmen unserer Diplomarbeit erledigt, Mitarbeiter der Firma Porsche Informatik GmbH haben das Projekt aufgesetzt und einige Teilbereiche implementiert. Folgende Grundlagen wurden erledigt:

- Login:
  Der komplette Login-Prozess gehört zu dem erledigten Teil des Projekts. Dieser wurde über Azure-Authentication ermöglicht.
- Buchen von Plätzen in Shared Desk Groups:
  Die Funktion, die einem Mitarbeiter erlaubt einen Platz zu buchen, wurde implementiert. Dem Benutzer ist es also schon möglich, einen Standort und ein Stockwerk zu selektieren und hier dann einen der möglichen Desk-Groups auszuwählen. Außerdem kann er auch das Datum der Buchung wählen und für mehrere Tage buchen.
- Benutzeransicht:
  Eine einfache Benutzeransicht wurde ebenfalls vorgegeben. Diese bestand aus einer Seite mit dem Namen des Benutzers und einer Menüleiste am linken Rande der Seite.
- Statistik:
  Ein weiterer Punkt, der in dieser Menüleiste aufgelistet ist, ist die Statistik. Die grundlegenden Funktionen waren schon vorbereitet.
- Buchungsübersicht:
  Die Buchungen, die ein Benutzer getätigt hat, werden in einer extra Seite übersichtlich dargestellt.
- Help Center:
  Auch eine Hilfe-Seite wurde erstellt, die dem Benutzer eine kurze Bedienungshilfe zur Verfügung stellt und die Möglichkeit nach Fragen zu suchen bietet. Falls weitere Hilfe notwendig ist, kann ein Mitarbeiter, der sich mit dem Programm auskennt, über Mail kontaktiert werden.

Für diese Funktionen wurde sowohl alles Nötige um die Anzeige in der Weboberfläche, als auch um die Datenverarbeitung und Speicherung im Hintergrunds-Programm zu ermöglichen, programmiert.

Die Funktionen, die unsere Projektgruppe hinzugefügt haben, bauen auf diese auf. Folgendes wurde implementiert:

- die Raumplan-Funktion, welche das Buchen von Plätzen erleichtert.
- die Mitarbeiter-Info Ansicht, die in der Benutzeransicht für Team-Leiter zu finden ist. Hier wurde ebenfalls die Statistik um eine Teamansicht erweitert.
- das Einbinden der Graph-API, welche die Daten von Outlook mit ShareD synchronisiert.
- die Event-Ansicht, die dem Benutzer erlaubt, Events aus seinem Outlook-Kalender einzusehen und zu erstellen.

8. Theoretische und fachpraktische Grundlagen und Methoden

In den folgenden Kapiteln werden unsere verwendeten Technologien, Entwicklungssysteme, Bibliotheken und Plugins beschrieben.

8.1 Technologien

8.1.1 Angular

Für die Erstellung der Weboberfläche haben wir „Angular“ verwendet, was ein von Google entwickeltes Typescript-Framework ist. Dieses wird in der Praxis oft zur Entwicklung von sogenannten „Single-Page-Apps“, oft als SPAs abgekürzt, verwendet. Eine Single-Page-App ist dabei eine Webseite, welche zwar auf andere Seiten innerhalb derselben SPA verweist, aber beim Aufrufen einer dieser anderen Seiten den Inhalt der alten Seite durch den der neuen Seiten ersetzt. Im Gegensatz zu herkömmlichen Webanwendungen, welche nach jedem Wechsel der Seite die komplette Seite vom Server laden müssen, müssen SPAs nur den Inhalt austauschen, was zu einer besseren Benutzererfahrung durch den schnellen und flüssigen Wechsel führt. (vgl. Angular - Single Page Application, 2024)

Abbildung 8.1 - Angular Logo (angular.io, 2024)

8.1.2 Spring Framework

Abbildung 8.2 - Spring Framework (spring.io, 2024)

Mit dem Spring Framework für Java haben wir unser Backend realisiert. Das Spring Framework ermöglicht eine einfache Abhängigkeitsinjektion, wodurch Objekte die benötigten Ressourcen und Services direkt zugewiesen bekommen, und einen aspektorientierten Programmierstil ermöglichen. Dabei können technische Aspekte, wie Sicherheit, Datenbank-Transaktionen, API-Aufrufe, usw., vom komplexen Teil des Programmcodes getrennt werden. (vgl. Spring Framework & Spring Boot, 2024)

8.1.3 Spring Boot

Spring Boot erleichtert die Entwicklung von Spring Framework Applikationen, indem es die Erstellung und Konfiguration dieser rationalisiert. Es realisiert dies mithilfe einer Autokonfiguration, welche Aufgrund der getätigten Einstellungen die richtigen Pakete installiert und konfiguriert. Weiters hilft Spring Boot beim Erzeugen eines Projektes, indem es sogenannte „Spring Starters“ zur Verfügung stellt. Diese sind kleine Pakete, welche verschiedene Abhängigkeiten zusammenfassen, damit man diese nicht einzeln auswählen muss. Letztlich ermöglicht es auch, direkt beim Erzeugen der Anwendung einen Webserver in das Programm zu integrieren, wodurch es ohne weitere Konfigurationen gestartet werden kann. (vgl. Spring Framework & Spring Boot, 2024)

Abbildung 8.3 - Spring Boot (spring.io, 2024)

8.1.4 Docker Container

Docker Container sind kleine, virtuelle Maschinen, die die auszuführende Software von der eigenen Entwicklungsumgebung des Nutzers trennen. Dabei geben sie verschiedene standardisierte Schnittstellen für das Gerät, auf dem der Container läuft, frei. In unserem Fall haben wir unsere PostgreSQL Datenbank in einen solchen Container verfrachtet, da die Datenbank dadurch auf jedem Arbeitsgerät, auf dem auch Docker läuft, direkt laufen kann. (vgl. Docker-Container, 2024)

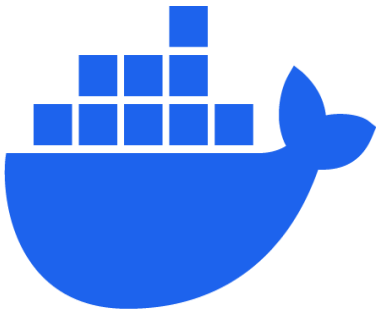

Abbildung 8.4 - Docker (docker.com, 2024)

8.2 Entwicklungssysteme

8.2.1 Intellij Ultimate

IntelliJ IDEA ist eine leistungsstarke integrierte Entwicklungsumgebung (IDE) der Firma JetBrains, die speziell für die Programmiersprachen Java, Kotlin, Groovy und Scala entwickelt wurde. Als Code Editor und Programmierwerkzeug ist IntelliJ IDEA ein unverzichtbares Tool für die Softwareentwicklung und bietet Entwicklern eine Vielzahl von Funktionen zur Unterstützung ihres Entwicklungsprozesses.

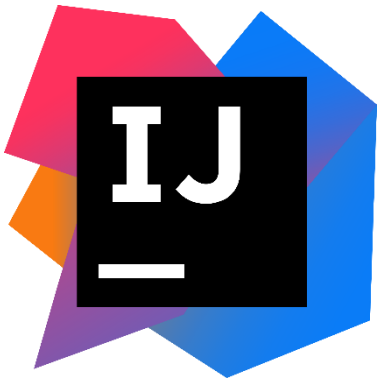

Abbildung 8.5 - IntelliJ IDEA (jetbrains.com, 2024)

Mit IntelliJ IDEA können Entwickler effizient und produktiv arbeiten, indem sie Features wie intelligentes Code-Refactoring, automatisches Code-Completion, Debugging-Funktionen und eine umfangreiche Bibliothek von Plugins nutzen. Damit ist IntelliJ IDEA nicht nur eine beliebte Java IDE, sondern auch ein unverzichtbares Entwickler-Tool für die gesamte Softwareentwicklung.

(vgl. Intellij Ultimate, 2024)

8.2.2 Webstorm

Abbildung 8.6 - Webstorm (jetbrains.com, 2024)

WebStorm von JetBrains ist eine auf JavaScript spezialisierte IDE (Integrated Development Environment) mit einem intelligenten Code-Editor und anderen Tools für Entwickler. Die Lösung ist mit Funktionen ausgestattet, die das Coding produktiver und angenehmer machen können. Der intelligente Editor von WebStorm kann Projektstrukturen verstehen und Codeschreiber mit Autovervollständigung, Qualitätsanalyse, sicherem Refactoring und anderen Funktionen unterstützen.

Andere Entwicklertools umfassen JavaScript-Debugging, Unit-Tests, lokale Änderungshistorie und mehr. Mit den Suchfunktionen können Nutzer beim Navigieren durch Codebases, Projekte und IDE-Einstellungen Zeit sparen. Zu diesen Suchfunktionen gehört auch ein „Search Everywhere“-Pop-up, um nach Aktionen, Dateien, Symbolen und anderen Daten zu suchen. Darüber hinaus ist WebStorm anpassbar und Nutzer können UI-Designs, Verknüpfungen und Plug-ins einrichten.

(vgl. Webstorm, 2024)

8.2.3 Visual Studio Code

Abbildung 8.7 - Visual Studio Code (visualstudio.com)

Visual Studio Code ist ein schlanker, aber leistungsstarker Quellcode-Editor, der auf Windows, macOS und Linux verfügbar ist. Das Entwicklungs-Tool bietet integrierten Support für JavaScript, TypeScript und Node.js und verfügt über ein umfangreiches Ökosystem an Erweiterungen für andere Programmiersprachen (etwa C++, C#, Java, Python, PHP und Go) und Laufzeitumgebungen (beispielsweise .NET und Unity).

(vgl. Visual Studio Code, 2024)

8.3 Bibliotheken und Plugins

8.3.1 Apache POI

Apache POI ist eine beliebte Java -Bibliothek, mit der man mit Microsoft Office -Formaten arbeiten kann, einschließlich Excel -Tabellen. Das XSSF (XML-Tabellenkalkulationsformat) des Apache POI funktioniert mit Excel-Dateien im neueren XML-basierten Format, das allgemein als . XLSX bekannt ist. Header und Fußzeilen befinden sich in Excel im oberen und unteren Bereich jeder Seite eines Arbeitsblattes. Darin können Inhalte wie Titel, Seitennummer, Daten und andere relevante Informationen platziert werden. Apache POI bietet Funktionen für die Arbeit mit Header und Fußzeilen mit dem XSSF -Modul.

(vgl. Apache POI, 2024)

8.3.2 Apache XML Graphics

Das Apache XML Graphics ermöglicht das Erzeugen und Bearbeiten von XML-Dokumenten in Java, welches im Raumplan Arbeitspaket verwendet wird, um SVG-Dateien im Programm erzeugen und gestalten zu können. SVG steht für „Scalable Vector Graphics“ und ist ein Dateiformat für Vektorgrafiken. (vgl. SVG Generator: Apache XML Graphics, 2024)

8.3.3 Microsoft Azure Plugins

Die Microsoft Azure Plugins werden benötigt, um im Code einen Benutzer über Microsoft anzumelden und auf die Ressourcen der Microsoft Graph API zuzugreifen. Weiters definieren diese Plugins bereits etliche Modellklassen, in welche die abgerufenen Daten abgespeichert werden können.

8.4 Sonstige Software

8.4.1 Outlook Schnittstelle

Um alle Daten über die Mitarbeiter und ihre Anwesenheit, sowie deren Kalenderevents zu bekommen, benutzen wir eine Microsoft Entra ID Applikation. Diese Applikation erlaubt es uns, die Anmeldung eines Nutzers über Microsoft abzuschließen, wodurch wir Zugriff auf die Daten des angemeldeten Mitarbeiters über die Microsoft Graph API bekommen.

8.4.2 Figma

Figma ist eine Software zur Erstellung von Design-Prototypen. Ziel dieser Anwendung ist es, seinen Nutzern dabei zu helfen, ihre Ideen und Vorstellungen mühelos auf den Bildschirm zu übertragen. Figma unterstützt auch die Zusammenarbeit von Teams, indem die einzelnen Mitglieder in Echtzeit alle Veränderungen mitbekommen und einsehen können. Es kann im Browser als Webanwendung genutzt werden. Es ist aber auch möglich sich dieses Tool lokal auf einen Rechner, als Desktop-App, herunterzuladen. Dasselbe gilt für Handys, wodurch der Nutzer von überall aus, auf seine Designs zugreifen kann. (vgl. Design: Figma, 2024)

9. Planung

9.1 Figma

Zur Erstellung von Mockups, für unsere Designs, wurde Figma verwendet. Durch diese Design-Prototypen war es möglich mit dem Projektpartner festzulegen, wie die implementierten Lösungen auszusehen haben. Verwendet wurde es für die Gestaltung der Events, des Raumplans und der Mitarbeiter Info. Für die Graph API war es nicht nötig, da dieses ausschließlich im Backend arbeitet und nicht auf irgendeine Weise vom Nutzer eingesehen wird.

Events:

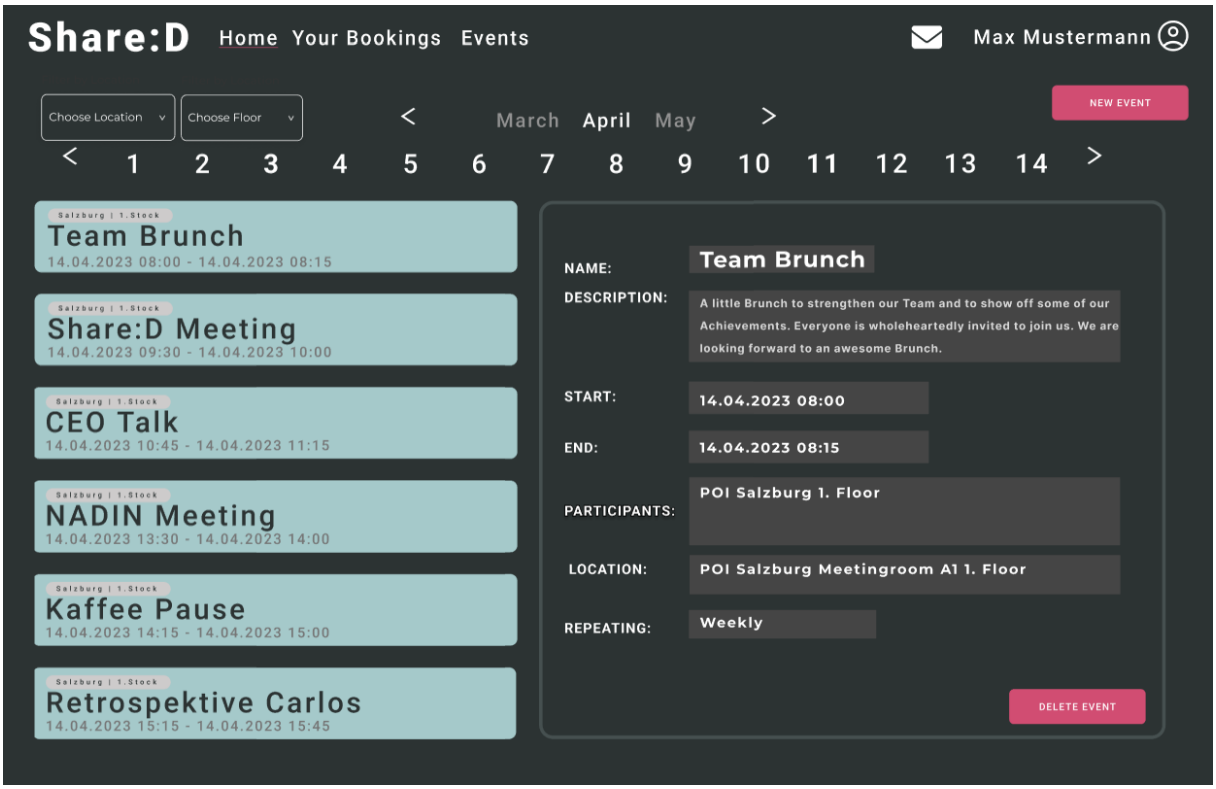

Abbildung 9.1 - Mockup für Events

Raumplan:

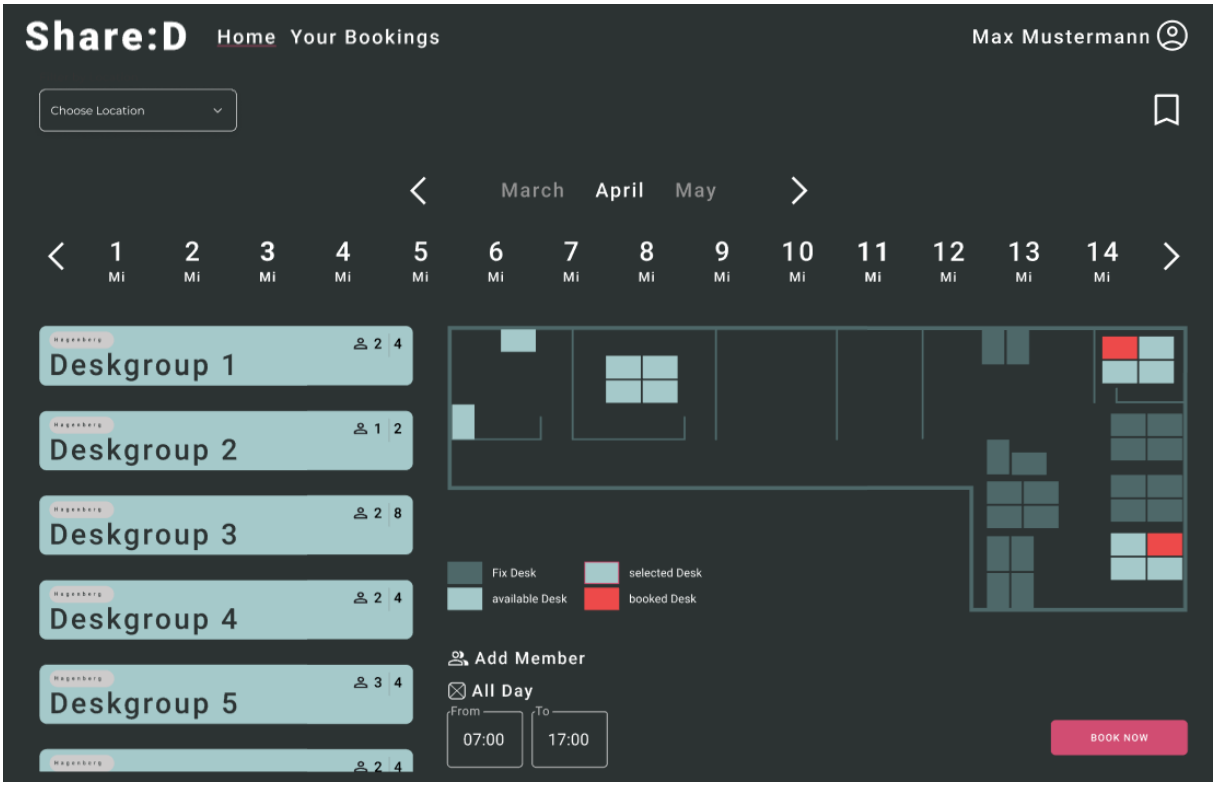

Abbildung 9.2 - Mockup für Raumplan

Team-Übersicht:

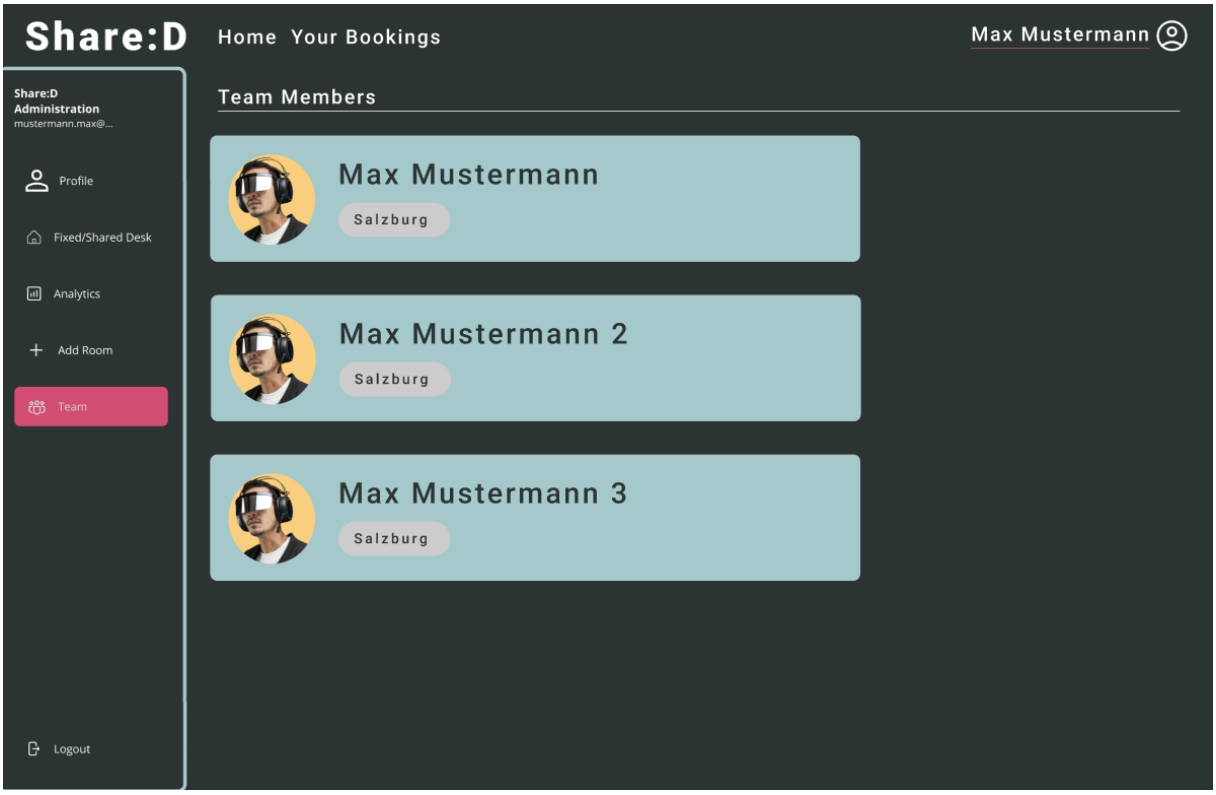

Abbildung 9.3 - Mockup für Team-Übersicht

9.2 Projektstrukturplan

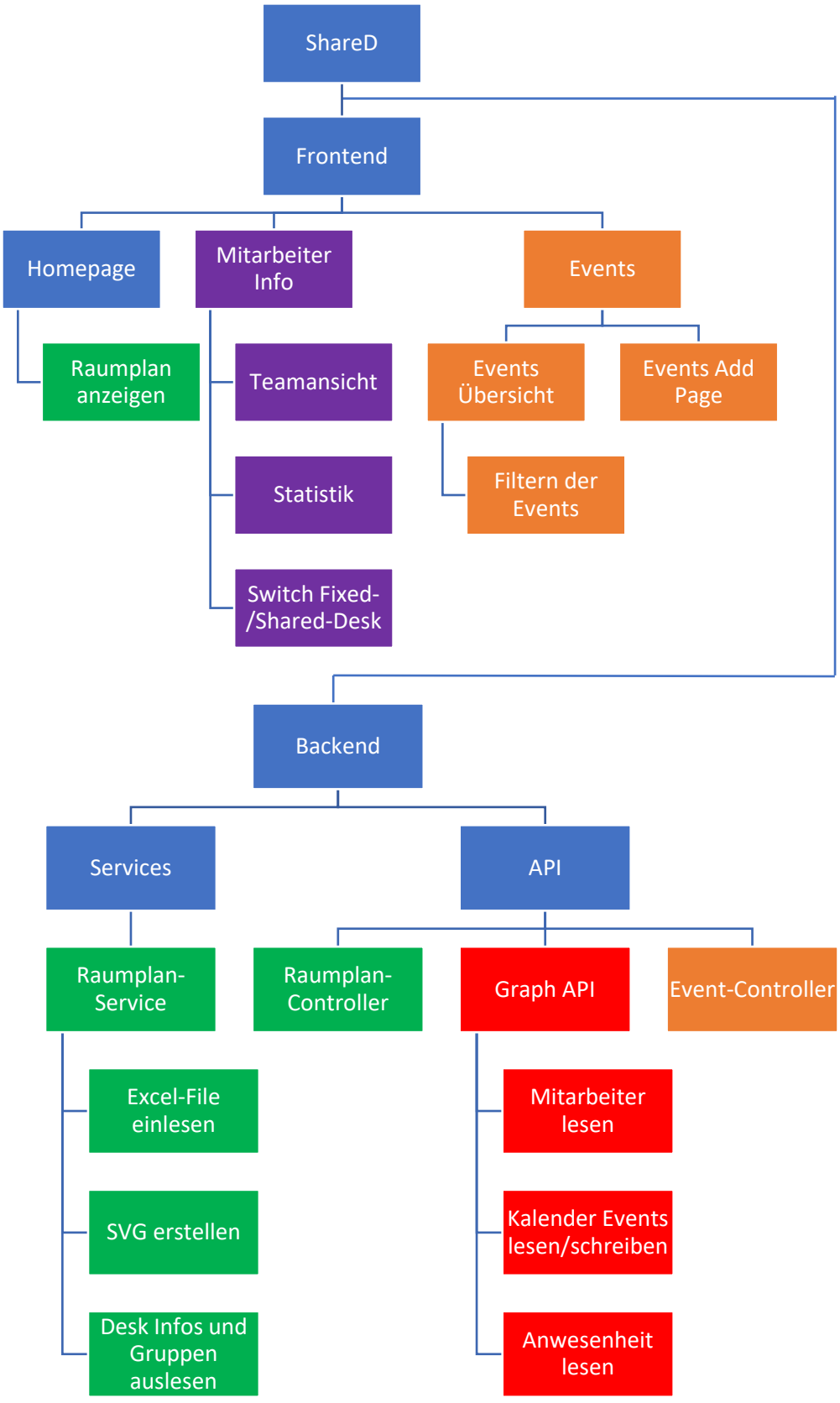

Abbildung 9.4 - Projektstrukturplan

Obiger Projektstrukturplan wurden entworfen, um unsere hergestellten Arbeitspakete und deren Zusammenhänge darzustellen. Im Frontend befindet sich der Raumplan, der in der Homepage zu finden ist, die Mitarbeiter Info, die aus der Teamansicht und den Statistiken besteht, und die Event-Seite. Im Backend befindet sich der Service für den Raumplan sowie die Einbindung der Graph API.

9.3 Projektorganisation

Ein Großteil unseres Projekts wurde während unseres 4-wöchigen Praktikums bei Porsche Informatik implementiert. Hier gab es wöchentliche Meetings mit unseren Kollegen und dem Vorstand des Projekts, wo wir unseren Fortschritt und Probleme besprachen. Eine offizielle Organisationsform gab es für uns jedoch nicht. Jeder hat ein Arbeitspaket zugewiesen bekommen, das individuell bearbeitet wurde.

9.4 Datenmodell

Das Programm speichert die Daten in einer Datenbank, welche dasselbe Modell wie in Abbildung 9.5 benutzt. Zum Zeitpunkt der Übernahme existierten die meisten Tabellen und Beziehungen bereits. Im Verlauf des Projektes wurden zunächst Änderungen an der “employee“ Tabelle vorgenommen, wie das „department“ Feld oder die Referenz zur „week_status“ Tabelle, welche auch implementiert wurde. Weiters wurde die komplette „event“ und „desk_svg_info“ Tabellen mit all ihren Referenzen und die „roomplan_path“ Tabelle zum Datenmodell hinzugefügt.

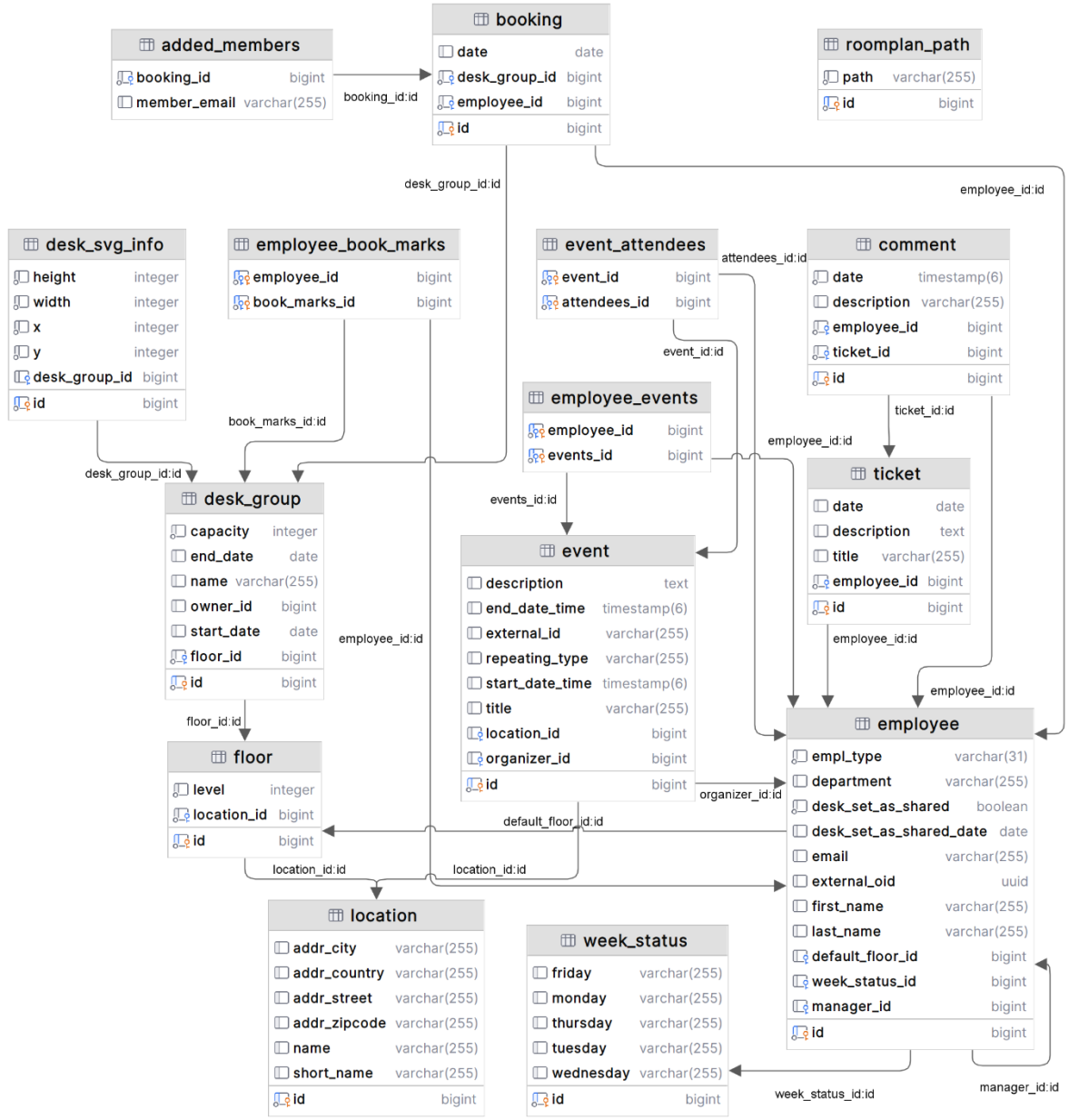

Abbildung 9.5 - Datenmodell

10. Funktionsweise

10.1 Raumplan

Ein wichtiges Hilfsmittel beim Buchungsprozess ist der Raumplan. Aus den Namen der Desk-Groups, zum Beispiel D-111, ist nicht ersichtlich, wo sich diese Gruppe befindet, und welche Schreibtische dazu gehören. Um dieses Problem zu beheben, wurde die Anzeige eines Raumplans implementiert. Sobald der Benutzer einen Standort und Stockwerk auswählt wird im Raumplanfeld der Plan des Stockwerks in Form eines SVG-Images dargestellt. Wenn der Benutzer über einer der aufgelisteten Desk-Groups hovert werden die dazugehörigen Schreibtische im danebenliegenden Plan markiert. Dadurch wird dem Mitarbeiter klar, wo sich diese Desk-Group befindet. [1]

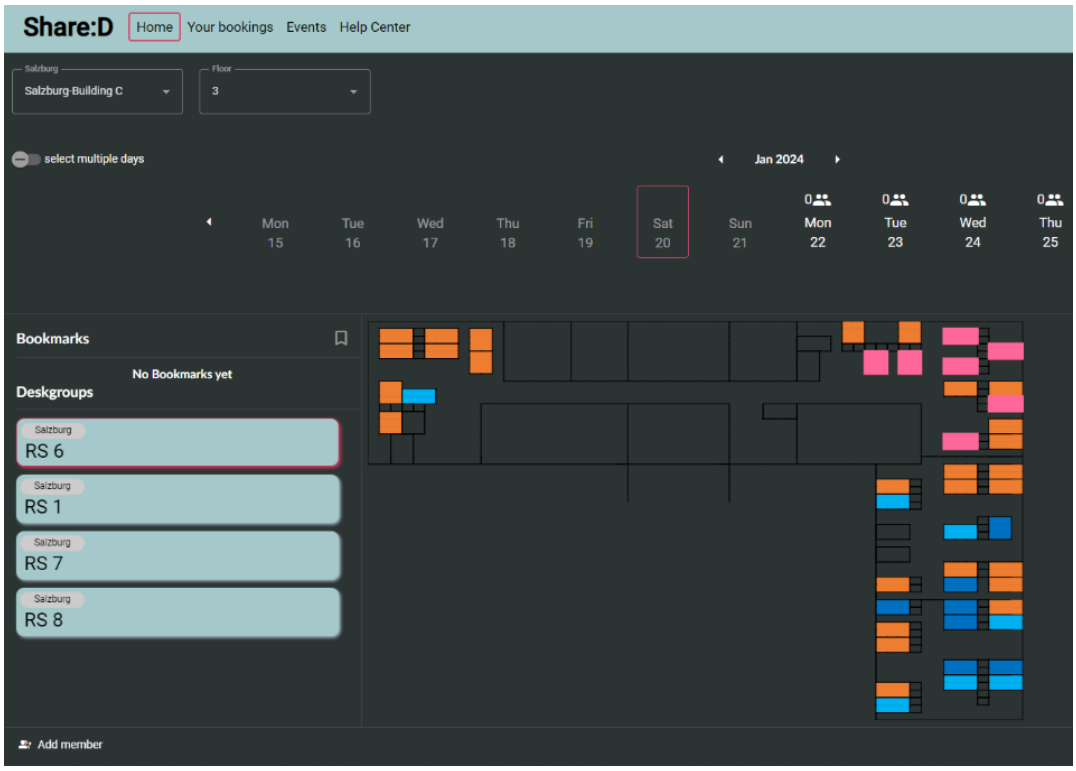

Abbildung 10.1 - Startseite mit Raumplan

Die Raumpläne werden im Hintergrund aus Excel-Files ausgelesen und abgespeichert und sobald sich der Benutzer anmeldet, werden diese Daten aktualisiert.

[1] Hinweis: Alle Raumpläne, die in der Arbeit dargestellt werden, enthalten lediglich Testdaten und entsprechen nicht den Gegebenheiten beim Auftraggeber.

Damit es Admins möglich ist den Pfad, in dem die Raumpläne abgelegt sind, zu aktualisieren, wurde dem Admin-Bereich eine Funktion hinzugefügt, wo man genau dies einstellen kann. Sobald der Pfad aktualisiert wird, werden alle alten Bilder im System gelöscht und die Pläne in dem neuen Pfad werden durchlaufen.

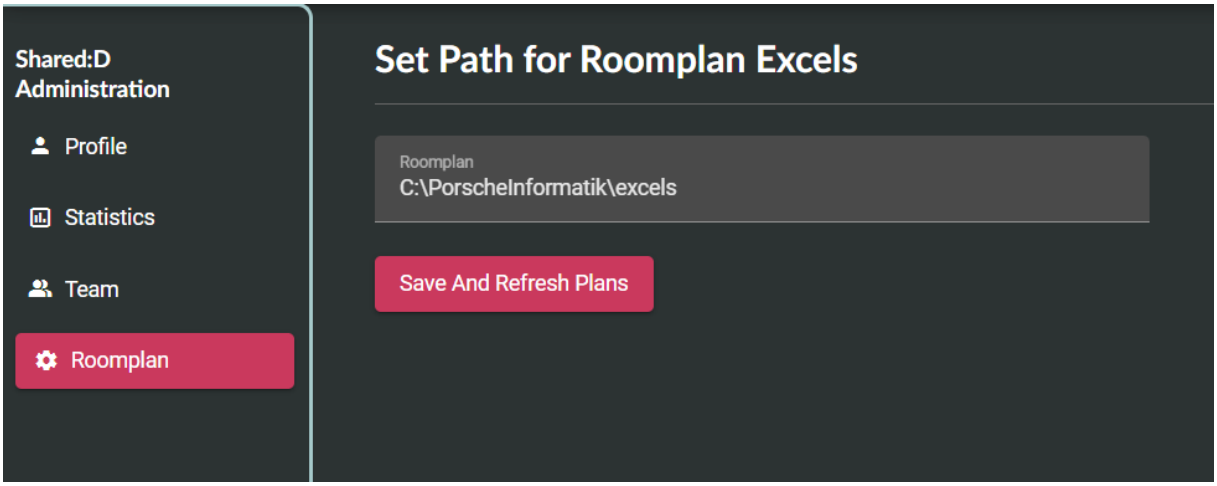

Abbildung 10.2 – Änderung des Raumplan-Pfads

10.2 Mitarbeiter Info

Dieser Teil der Applikation besteht aus drei Ansichten. Benutzer, die Admin-Rechte haben, werden hier mit Informationen über das eigene Team oder die eigene Abteilung versorgt, je nachdem ob es sich um einen Team- oder Abteilungsleiter handelt. In beiden Fällen stehen folgende Ansichten zur Verfügung:

- Benutzer-Ansicht (enthält den Fixed-Shared-Desk-Switch)
- Team-Übersicht
- Team-Statistiken

Fixed-Shared-Desk-Switch

Dem Benutzer ist es möglich seinen Schreibtisch als Shared-Desk für andere Mitarbeiter buchbar zu machen, wenn er ihn nicht braucht. Dies kann er bei seinen Benutzereinstellungen festlegen.

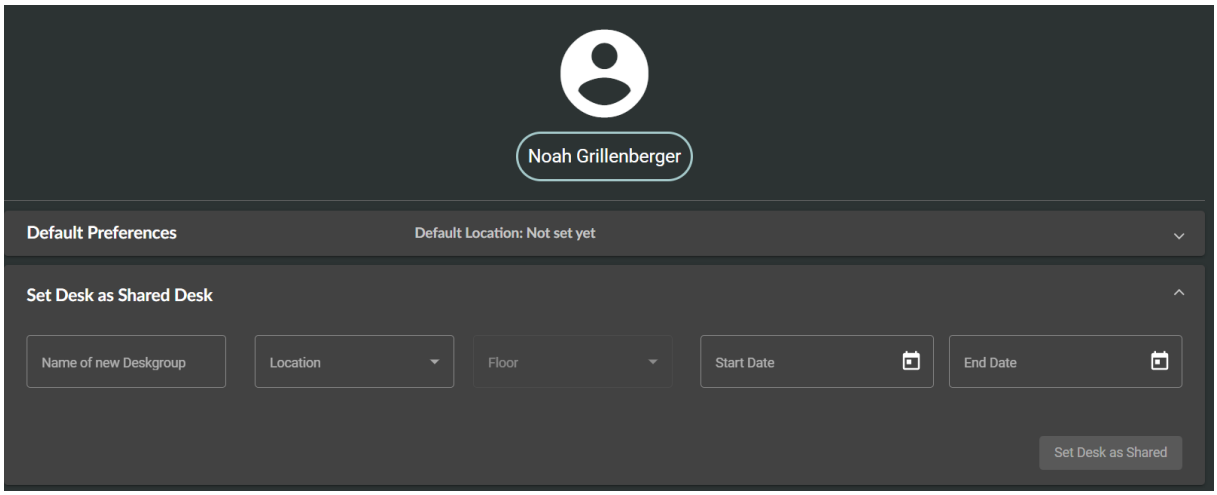

Abbildung 10.3 – Schreibtisch als Shared-Desk freigeben

Hierfür muss der Benutzer den Datumsbereich, für wann der Schreibtisch buchbar ist, und den Namen, der in der Webseite angezeigt wird, angeben. Sobald die Freigabe erfolgt, wird eine neue Desk-Gruppe mit Namen, Gültigkeitsdatum und Besitzer erstellt, die aber nur einen Schreibtisch beinhaltet. Die maximale Kapazität wird auf 1 gesetzt und die Anzahl der belegten Plätze auf 0. Der Benutzer kann dies nicht rückgängig machen.

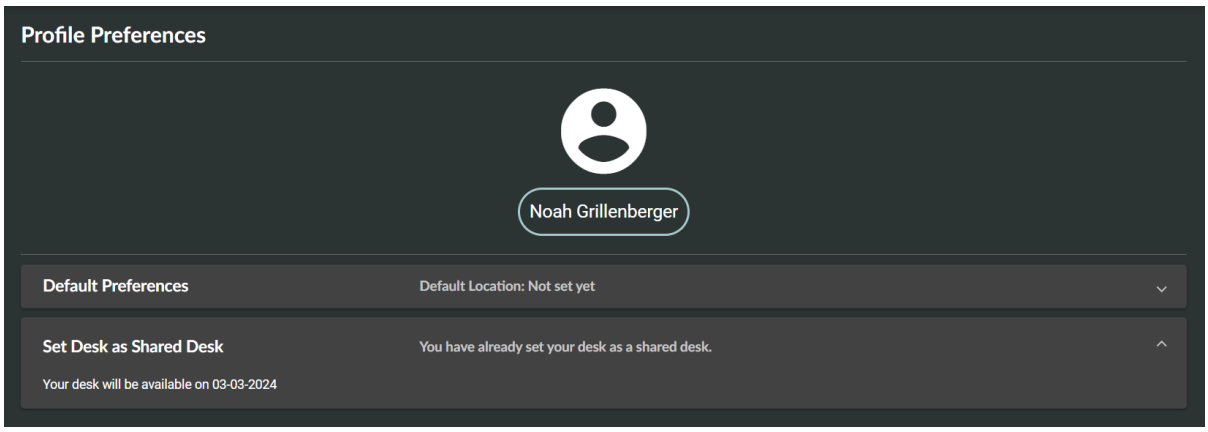

Abbildung 10.4 – Schreibtisch-Freigabe erfolgreich

Die Desk-Group wird dann mit den anderen verfügbaren Desk-Gruppen angezeigt.

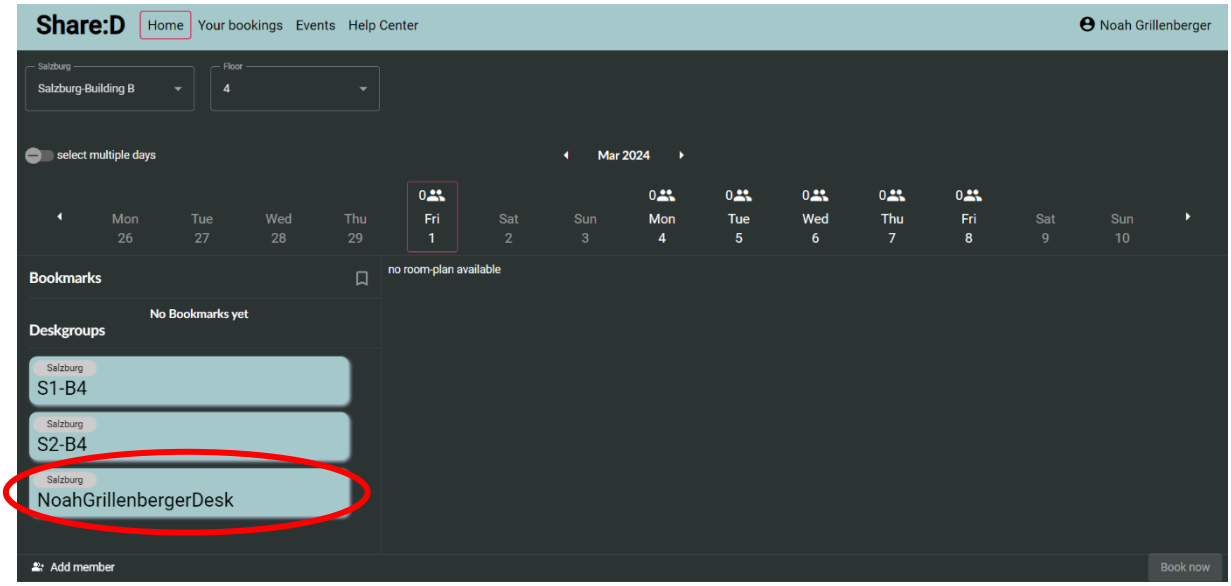

Abbildung 10.5 – freigegebener Schreibtisch buchbar

Team-Übersicht

Durch die Team-Übersicht kann der Teamleiter auf einen Blick erkennen:

- Wer alles zu seinem Team gehört
- Wer heute...
    - Im Büro ist
    - Im Home-Office ist
    - Abwesend ist
- Wo sie diese Woche waren
- Wo sie diese Woche sein werden

Dies ist besonders von Vorteil, da der Teamleiter dann weiß mit welchen seiner Arbeitskräfte er am heutigen Tag rechnen kann und über welches Medium diese zu erreichen sind (also in Person oder über digitale Kommunikationsmittel).

Dadurch wird auch klarer, welcher Meeting-Raum, bezüglich Größe und Sitzplatzkapazität, in Frage kommt für die täglichen Projekt-Meetings.

Abbildung 10.6 - Team-Übersicht als Teamleiter[2]

Dieselben Vorteile gelten auch für den Abteilungsleiter, nur dass der zusätzlich auch den "Status" der Angestellten einsehen kann, die nicht ihm direkt unterstellt sind, d.h. also auch die seinen Teammitgliedern unterstellten Mitarbeiter und in der Hierarchie folgenden Personen.

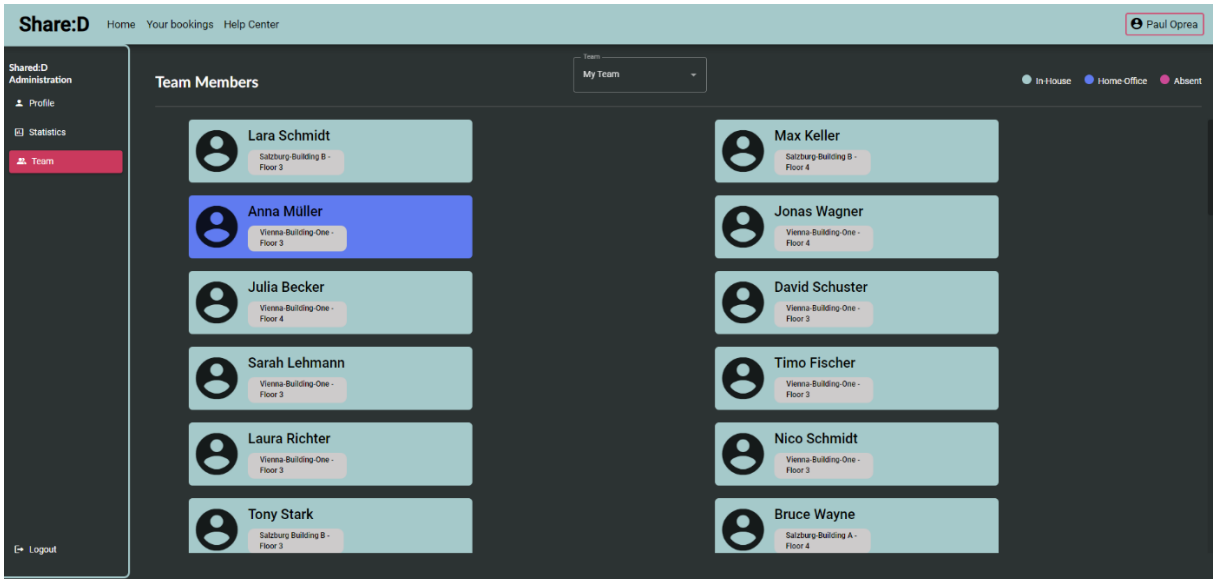

Abbildung 10.7 - Teamansicht als Abteilungsleiter

[2] Hinweis: Alle Bilder wo Mitarbeiter dargestellt werden, enthalten lediglich Testdaten und nicht echte personenbezogene Daten.

Team-Statistiken

Anhand der Team-Statistiken können Team- und Abteilungsleiter einsehen, wie hoch die Auslastung der Arbeitsplätze vor Ort - bezogen auf die ihnen untergeordneten Teams - ist. Die Statistik kann die Auswertung sowohl prozentuell als auch numerisch anzeigen. Dabei kann gefiltert werden nach:

- Standort
- Stockwerk
- Team

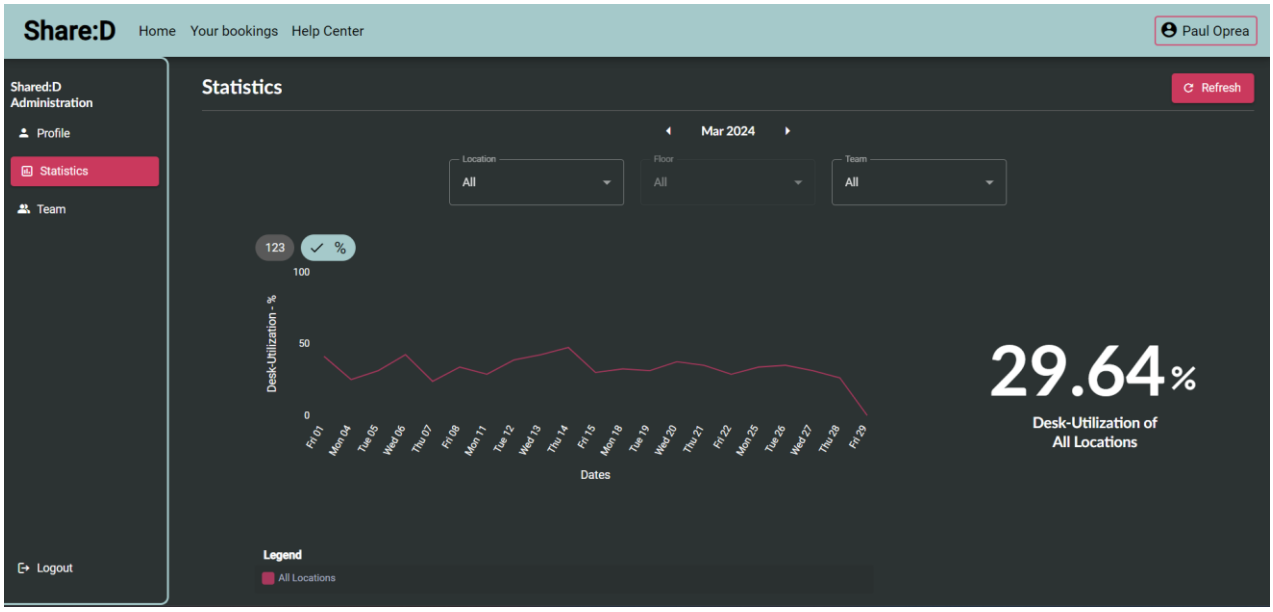

Abbildung 10.8 - Team-Statistiken[3]

10.3 Graph API

Damit sich Mitarbeiter anmelden, Arbeitsplätze buchen und die nächsten Events einsehen können, müssen die Daten von Outlook in unsere Datenbank gespeichert werden. Dazu wird die Microsoft Graph API, welche verschiedene Zugriffsmöglichkeiten auf die Daten der Mitarbeiter und Events bietet, verwendet. Der Zugriff auf die Daten aus unserer Datenbank wird über eine eigene API bereitgestellt, damit diese abgerufen, bearbeitet und gelöscht werden

[3] Hinweis: Alle statistischen Auswertungen, die in jeglichen Bildern zu sehen sind, basieren auf Testdaten.

können. Zusätzlich können auch neue Daten zurück an die Microsoft Graph API gesendet werden, um beispielsweise ein im Programm erstelltes Event im Outlook Kalender ansehen zu können.

10.4 Events

Ein weiterer Teil unserer Diplomarbeit sind Microsoft Events. Diese werden dem Benutzer übersichtlich in sogenannten Cards angezeigt. Diese Events kann der Benutzer mittels dem darüberliegenden Auswahlmenü auch noch nach dem Jahr, Monat und Tag sortieren. Durch einen Klick auf das gewünschte Event, werden dem Benutzer die Informationen des jeweiligen Events auf der rechten Seite angezeigt.

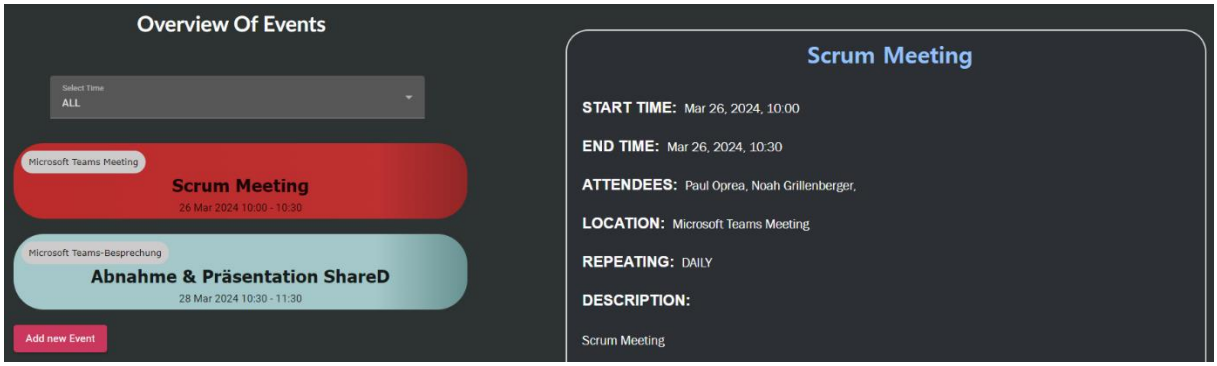

Abbildung 10.9 - Events Ansicht

Als nächstes kann der Mitarbeiter durch den Button „Add new Event“ ein neues Event anlegen. Dazu muss der Mitarbeiter von Porsche Informatik unteranderem den Titel, die Beschreibung, die Eingeladenen, die Startzeit und die Endzeit in die jeweiligen Eingabefelder eintragen. Weiters kann auch noch der Wiederholungstyp und der Ort der Besprechung eingeben werden. Fehlen die Angaben wird standardmäßig eine einmalige Microsoft-Team-Besprechung angenommen. Wenn der Benutzer danach auf den Button „Add Event“ drückt, wird dieses Event automatisch an die Graph API übergeben. Hier wird das Event durch Microsoft in

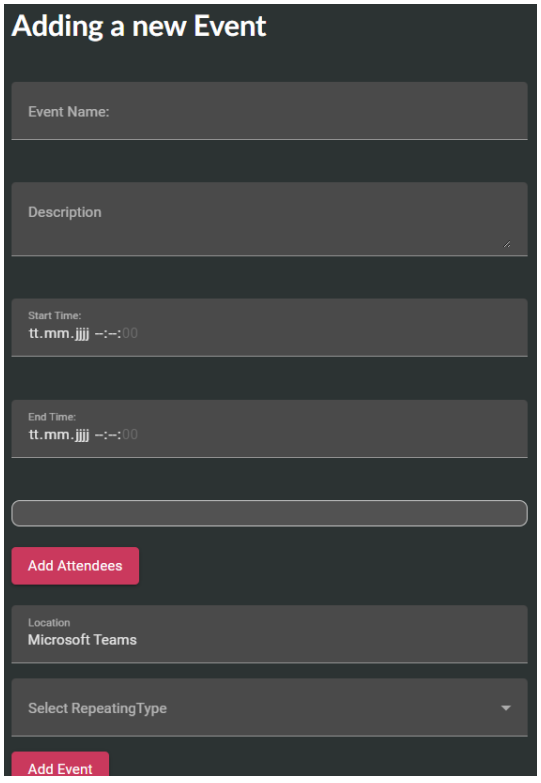

Abbildung 10.10 - Add Ansicht

Outlook eingetragen und die Eingeladenen erhalten zugleich auch eine Benachrichtigung.

11. Implementierung

11.1 Raumplan

Die Porsche Informatik speichert wie bereits erwähnt ihre Darstellungen der Stockwerke in Excel-Files ab. Hier werden die Excel-Zellen mit Grenzen, Farbfüllungen und Schriftzügen versehen, sodass die Excel-Arbeitsmappe einen für Menschen lesbaren Plan darstellt. Die Aufgabe besteht nun darin, die Excels auszulesen und in Share:D anzuzeigen.

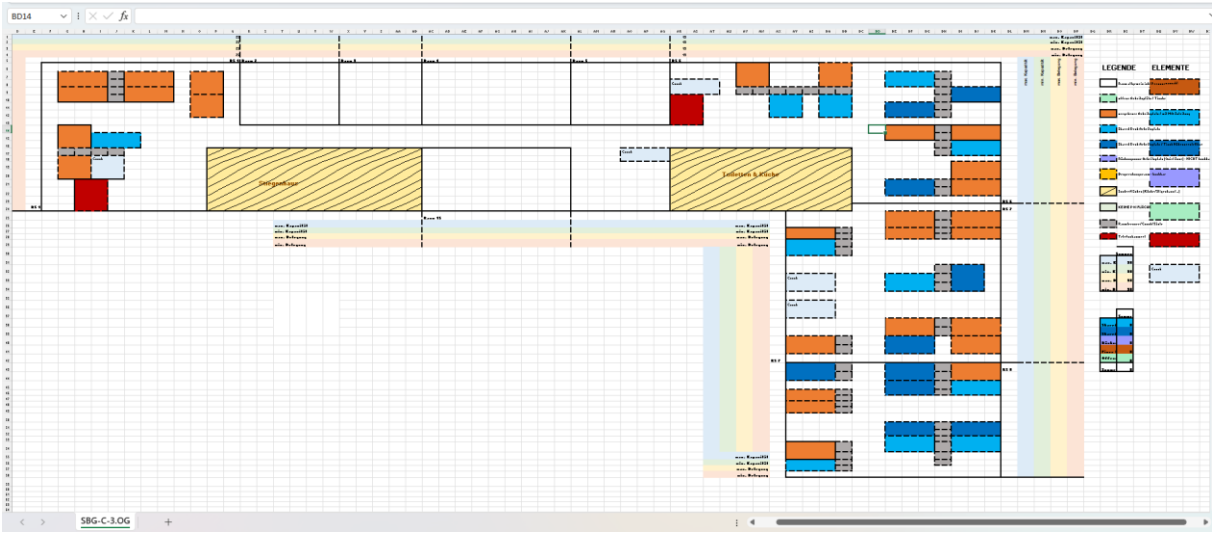

Abbildung 11.1 - Beispiel Excel

Man kann das Arbeitspaket in zwei Teilbereiche gliedern. Einerseits die Algorithmen für das Einlesen von den Excels und der Generierung von den SVG-Images und anderseits die Einbindung dieser Algorithmen in das Share:D Programm mitsamt der Anzeige der Bilder in der Weboberfläche.

11.1.1 Excel einlesen und verarbeiten

Dieser Teilbereich des Arbeitspakets beinhaltet vier Algorithmen:

- ExcelParse
- DeskIterator
- DeskGroupIterator
- SvgGenerator

Diese vier greifen auf ein gemeinsames Model zu, das nur von diesen Algorithmen verwendet wird. Das Model besteht ausfolgenden Klassen:

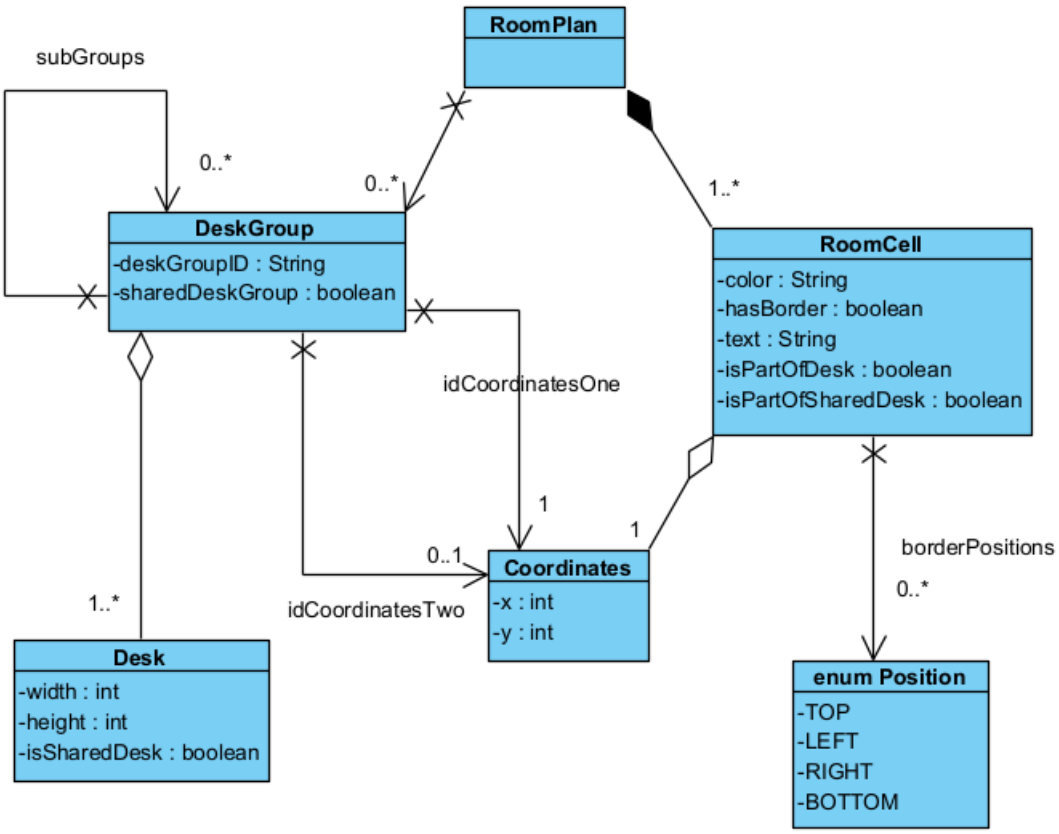

Abbildung 11.2 – Raumplan UML-Diagramm

Der Raumplan besteht aus `RoomCells` und `DeskGroups`. Ein `RoomCell` speichert alle Daten einer Zelle, die für das Abbilden und das Zuordnen zu Desk-Gruppen benötigt werden. DeskGroups speichern Schreibtische oder Sub-Gruppen die Schreibtische enthalten.

ExcelParse

Der „ExcelParse“ Algorithmus ist der erste Algorithmus, der aufgerufen wird. Er ist zuständig für die Grob-Auslesung und Speicherung aller benötigten Zellen und deren Eigenschaften.

Beim Erzeugen eines Objekts dieser Klasse wird im Konstruktor eine Map-Liste von DeskGroup Regulären Expressions übergeben. Diese regular expressions dienen zur richtigen Identifizierung und Zuordnung der Desk-Group Namen. Die Namen variieren nämlich von Standort zu Standort in den Excel Sheets und müssen daher im Voraus bestimmt und dem ExcelParser übergeben werden.

Außerdem wird im Konstruktor ein neues RoomPlan-Objekt angelegt wo alle Infos, die aus dem Excel gelesen werden, gespeichert werden.

Als Erstes werden Default-Werte bei vier Variablen, die verwendet werden, um den richtigen Bereich einzugrenzen, gesetzt. Wie in der bereits dargestellten Excel Tabelle zu erkennen ist, gibt es nämlich einige Schattierungen und Einrückungen rund um den eigentlichen Plan, der nicht angezeigt werden soll. Es gibt außerdem rechts eine Legende, die nicht gebraucht wird. Dieser Bereich muss also ignoriert oder im Nachhinein entfernt werden.

```
for (int rowIndex = 0; rowIndex < sheet.getLastRowNum(); rowIndex++) {
    Row row = sheet.getRow(rowIndex);

    if (rowHasMediumBorder(row)) {
        if (marginTop == -1) {
            marginTop = rowIndex;
        }
        marginBottom = rowIndex;
    }

    this.roomPlan.getRoomcells().add(new ArrayList<>());

    /*check if legend is already found, if not then set width to full width of the row*/
    if (!legendFound) {
        this.roomWidth = row.getLastCellNum();
    }
    /*Saves the last cell with a border, so we can delete unnecessary cells later*/
    int marginRight = 0;
```

Code 11.1 – Iteration über Zellen

Zuerst wird bei jeder neuen Zeile überprüft, ob diese Zeile eine Zelle mit einem vertikalen Medium-Border beinhaltet. Dadurch wird überprüft, ob der Plan schon in dieser Zeile beginnt. Wenn dies zutrifft, wird die `marginTop`-Variable, also die Einrückung von oben, gesetzt. Die `marginBottom`-Variable wird jedes Mal gesetzt, dadurch kennt das Programm nach der vollständigen Iteration den Index der letzten Zeile, wo eine vertikale Grenze vorgekommen ist. Obwohl alle Zeilen vor dem Planbeginn gelöscht werden, müssen sie trotzdem eingelesen werden, da es sein kann, dass sich die Legende-Überschrift in diesem Bereich befindet. Auf diese wird noch später eingegangen. Falls die Legende noch nicht gefunden wurde, wird die gesamte Länge der Zeile als Raumbreite angenommen.

Für jede Zelle werden erstmals die Infos Farbe und Grenze herausgesucht, angefangen mit der Farbe.

```
if (excelCell != null) {
    CellStyle cellStyle = excelCell.getCellStyle();

    if (cellStyle.getFillForegroundColorColor() != null) {
        XSSFColor xColor = (XSSFColor) cellStyle.getFillForegroundColorColor();
        if (isDeskColor(xColor.getARGBHex(), roomCell)) {
            /*
            There are cells around the map that have the same color as the fixed desks
            just with a different gradient. That's why I check if the cell above has the same color
            because these extra cells around the map don't have cells above and beside them
            with the same color
            Note: This doesn't work if the desks are made smaller than 4x4 in the Excel file
            */
            if (hasSurroundingWithSameColor(rowIndex, cellIndex)) {
                roomCell.setColor(xColor.getARGBHex());
            }
        }
    }
}
```

Code 11.2 – Setzen der Farbe

Die Farbe wird in der `isDeskColor`-Methode mit den Farb-Hex-Codes, die die Klasse als statische Werte abspeichert, verglichen. Dadurch werden nur die Schreibtische auf dem resultierenden Bild farblich dargestellt. Beim Vergleichen der Farben wird auch bestimmt, ob ein Tisch, je nach Farbe, Shared-Desk oder Fix-Desk ist. Diese Information ist für die spätere Erstellung und Zuordnung der Schreibtische notwendig.

An dieser Stelle ergibt sich jedoch das Problem, dass es im Excel-Plan Stellen gibt, die mit den gleichen Farben markiert sind wie die der Schreibtische und nicht weggeschnitten oder ignoriert werden können. Dies kann anhand der Tatsache gelöst werden, dass alle Schreibtische im Excel aus mindestens 4x4 Feldern bestehen. Daher muss eine Zelle, die Teil eines Schreibtisches ist, auch mindestens ein Feld mit der gleichen Farbe darüber oder darunter und gleichzeitig daneben haben. Diese Abfrage wurde in der `hasSurroundingWithSameColor`-Methode implementiert. Wenn diese „true“ zurückgibt, wird die Zellfarbe abgespeichert.

Als nächstes muss festgestellt werden, ob die Zelle eine Grenze hat und wenn ja, auf welcher Seite.

```
/* Set the Border Info */
if (hasBorder(excelCell.getCellStyle())) {
    roomCell.setHasBorder(true);
    roomCell.setBorderPositions(getBorders(excelCell.getCellStyle()));
} else {
    roomCell.setHasBorder(false);
}
```

Code 11.3 – Setzen der Grenze

Diese Information wird für alle Zellen abgespeichert, damit der Plan alle Grenzen abbildet und nicht nur jene der Schreibtische.

```
setCellInfo(roomCell, rowIndex, cellIndex);

if (hasRightBorder(excelCell, row.getCell( i: cellIndex + 1))) {
    marginRight = cellIndex;
    if (marginLeft == -1 || cellIndex < marginLeft) {
        marginLeft = cellIndex;
    }
}
roomCell.setText(getText(excelCell, cellIndex, rowIndex));
/*
These coordinates can't be used in the svg generator because after margins
and deletions of rows they are not accurate to the actual position
of the cell. They are only to be used for comparing cells in the ExcelParse class
*/
roomCell.setCoordinates(new Coordinates(cellIndex, rowIndex));

this.roomPlan.getRoomcells().get(rowIndex).add(roomCell);
```

Code 11.4 - margin, Text und Koordinate setzen

Nach dem Setzen der Zellen-Infos kontrollieren wir, ob die Zelle eine rechte Grenze besitzt. Wenn ja wird `marginRight` aktualisiert und falls `marginLeft` noch den Default-Wert hat, oder der aktuelle Zell-Index kleiner ist als der aktuelle Wert der Einrückung, wird dieser auch auf den neuen Wert gesetzt. Die zweite Klausel ist dazu da, dass wenn ein Teil des Plans unten nach links expandiert, dieser nicht gelöscht wird. Wenn so ein Fall auftritt, muss der `marginLeft` verringert werden.

```
String content = cell.getRichStringCellValue().getString();

/*Find width of map without the legend*/
if (content.toLowerCase().trim().equals("legende")) {
    this.roomWidth = cellIndex - 1;
    this.legendFound = true;
}

if (cell.getCellStyle() != null) {
    XSSFCellStyle cs = (XSSFCellStyle) cell.getCellStyle();
    XSSFFont font = cs.getFont();
    if (font.getBold()) {
        if (matchesRegex(content)) {
            addDeskGroupID(content.trim(), cellIndex, listIndex);
        }
    }
}
return content;
```

Code 11.5 – Text überprüfen

Zuletzt werden noch Text und die Koordinaten gespeichert. Auch hier müssen die Umstände im Excel berücksichtigt werden.

Einerseits ist die Überprüfung, ob der Inhalt dem Wort „Legende“ entspricht, wichtig. Wenn ja, weiß der Algorithmus das alles rechts danach und darunter nicht eingelesen werden muss und er kann die Breite des Raumes aktualisieren.

Nach dieser Überprüfung gibt es eine Abfrage, ob der Schriftzug fett formatiert ist. Alle Schreibtischnamen werden nämlich im Excel fett geschrieben. Um danach nochmal sicher sein zu können, dass es sich um einen Desk-Group-Namen handelt, wird der Name mit den Regular Expressions, die im Konstruktor übergeben worden sind, verglichen.

Die `addDeskGroupID`-Methode spielt beim Setzen der ID eine wichtige Rolle.

```
public void addDeskGroupID(String id, int cellIndex, int listIndex) {
    if (id.matches( regex: ".*[A-Z]$")) {
        addSubGroup(id, cellIndex, listIndex);
    } else {
        addOrUpdateDeskGroup(id, cellIndex, listIndex);
    }
}
```

Code 11.6 – Desk-Group-Namen hinzufügen

Wenn ein Desk-Group-Name mit einem Buchstaben aufhört, kann das Programm daraus schließen, dass dieser Teil einer Gruppe ist, die aus mehreren Sub-Gruppen besteht, zum Beispiel: D-111A, D-111B. Falls dies zutrifft, wird erstmal die Parent-Desk-Group geholt oder erstellt. Der Name dieser Eltern-Gruppe ist gleich der Kind-Gruppe ohne den Buchstaben am Schluss. Von der Kind-Gruppe werden die Koordinaten der Zelle, in der der Name der Gruppe gefunden geworden ist, gespeichert.

```
private void addSubGroup(String id, int cellIndex, int listIndex) {
    String parentId = id.substring(0, id.length() - 1);
    DeskGroup parentDeskGroup = getOrCreateGroup(parentId);

    DeskGroup childDeskGroup = new DeskGroup();
    childDeskGroup.setDeskGroupID(id);
    childDeskGroup.setIdCoordinatesOne(new Coordinates(cellIndex, listIndex));
    parentDeskGroup.getSubGroups().add(childDeskGroup);
}

1 usage   Noah Grillenberger
private void addOrUpdateDeskGroup(String id, int cellIndex, int listIndex) {
    DeskGroup deskgroup = getOrCreateGroup(id);
    if (deskgroup.getIdCoordinatesOne() != null) {
        deskgroup.setIdCoordinatesTwo(new Coordinates(cellIndex, listIndex));
    } else {
        deskgroup.setIdCoordinatesOne(new Coordinates(cellIndex, listIndex));
    }
}
```

Code 11.7 – Koordinaten der Gruppe abspeichern

Ein DeskGroup-Objekt speichert mindestens ein und maximal zwei Sets von Koordinaten. Der Name einer Kind-Gruppe kommt in einem Excel-Plan nur einmal vor, hier muss also nie ein

zweites Vorkommen abgespeichert werden. Bei einer normalen Desk-Group kann der Name zwei Mal vorkommen, muss aber nicht so sein. Die Koordinaten der Namen werden gebraucht, um später die Schreibtische richtig zuordnen zu können.

Nach dem erfolgten Auslesen des Textes müssen abschließend die Koordinaten der Zelle gespeichert werden.

Nach dem Iterieren der ganzen Zellen wird `cropRight` ausgeführt, wo alle Zellen nach der letzten Zelle, die eine vertikale Grenze hat, gelöscht werden. Hier werden also alle Felder zwischen Ende des Plans und Beginn des Legenden-Bereichs entfernt. Wenn alle nötigen Zeilen des Excels durchiteriert sind, kann die Applikation alle unnötigen Zellen oberhalb, unterhalb und links vom Plan löschen. Diese Veränderung wirkt sich auch auf die Koordinaten aus, die in den Zellen und Desk-Groups gespeichert sind. Daher werden diese noch nachkorrigiert.

Der ExcelParse Algorithmus ist nun fertig. Alle Infos sind in dem `roomPlan`-Attribut abgespeichert und können abgerufen werden.

SvgGenerator

Nach dem Ausführen des ExcelParse Algorithmus hat die Applikation alle nötigen Daten, um ein SVG-Image zu erzeugen. In Java werden Grafiken grundsätzlich durch die abstrakte Klasse Graphics2D erstellt. Wir können nun die SVGGraphics2D Klasse verwenden, eine Implementation von Graphics2D, um Grafiken in Form eines SVGs zu erzeugen. Es wird außerdem ein Objekt einer Document Klasse benötigt, damit das Image bearbeitet werden kann. (vgl. SVG Generator: Apache XML Graphics, 2024)

```
DOMImplementation impl = GenericDOMImplementation.getDOMImplementation();
String svgNS = "http://www.w3.org/2000/svg";

Document document = impl.createDocument(svgNS, qualifiedName: "svg", doctype: null);

SVGGraphics2D svgGenerator = new SVGGraphics2D(document);
svgGenerator.setSVGCanvasSize(new Dimension( width: getPlanWidth()*CELL_WIDTH,
        height: roomPlan.getRoomcells().size()*CELL_HEIGHT));
```

Code 11.8 – Erstellung SVG-Generator

Die statischen Variablen `CELL_WIDTH` und `CELL_HEIGHT` speichern die fixe Länge und Breite einer Zelle in Pixel ab, in diesem Fall sind sie auf 15px*10px gesetzt.

Nun gibt es eine Fläche, in der das Programm einen Plan zeichnen kann. Für jede eingelesene Zelle werden deren Grenzen und Farbe zu der Fläche hinzugefügt. Die Farbe einer Zelle wird in Form eines Rechtecks, dessen Größe den fixen Abmessungen entspricht, abgespeichert.

```
Coordinates coordinates = new Coordinates(cell.getCoordinates());
coordinates.convert(CELL_WIDTH, CELL_HEIGHT);

if (cell.getColor() != null) {
    Rectangle2D rectangle2D = new Rectangle2D.Double
            (coordinates.getX(), coordinates.getY(),  w: CELL_WIDTH+0.5,  h: CELL_HEIGHT+0.5);
    g2d.setColor(getColorFromHex(cell.getColor()));
    g2d.fill(rectangle2D);
}
```

Code 11.9 – Erstellung eines Rechtecks

Wenn die Applikation aber die genaue Zellgröße hernimmt und diese mit Farbe befüllt, ergeben sich dünne weiße Linien zwischen den Zellen.

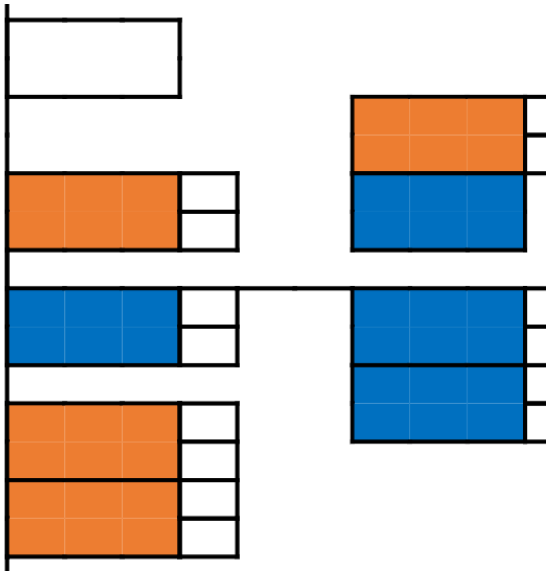

Abbildung 11.3 - SVG mit fehlerhafter Füllung

Um dies zu umgehen, müssen die Rechtecke etwas größer abgespeichert werden, damit sie sich überlappen.

Im Excel-Plan kommt es vor, das Grenzen unterschiedlich formatiert oder auf verschiedenen Seiten der Zelle gesetzt werden. Durch solche Inkonsistenzen ergeben sich manchmal unterschiedliche Dicken bei den Grenzen wie hier zu sehen ist.

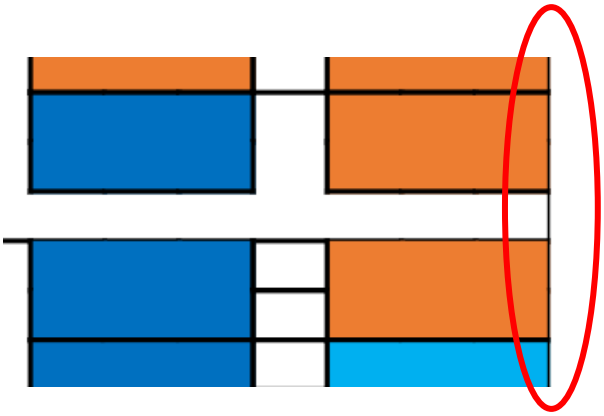

Abbildung 11.4 – SVG mit fehlerhaften Grenzen

Die dickeren Linien hier sind in Wahrheit zwei Grenzen, die nebeneinander liegen. Dieses Problem kann gelöst werden, wenn für jede Grenze eine zweite eingezeichnet wird. Umgesetzt wird dies durch eine Prüfung der umliegenden Felder der aktuellen Zelle. Wenn zum Beispiel die Zelle links, eine Trennungslinie auf der rechten Seite hat, wird für die aktuelle Zelle eine Linie links gezeichnet. Nachdem der Algorithmus fertig ist, gibt er das komplette SVG-Bild zurück:

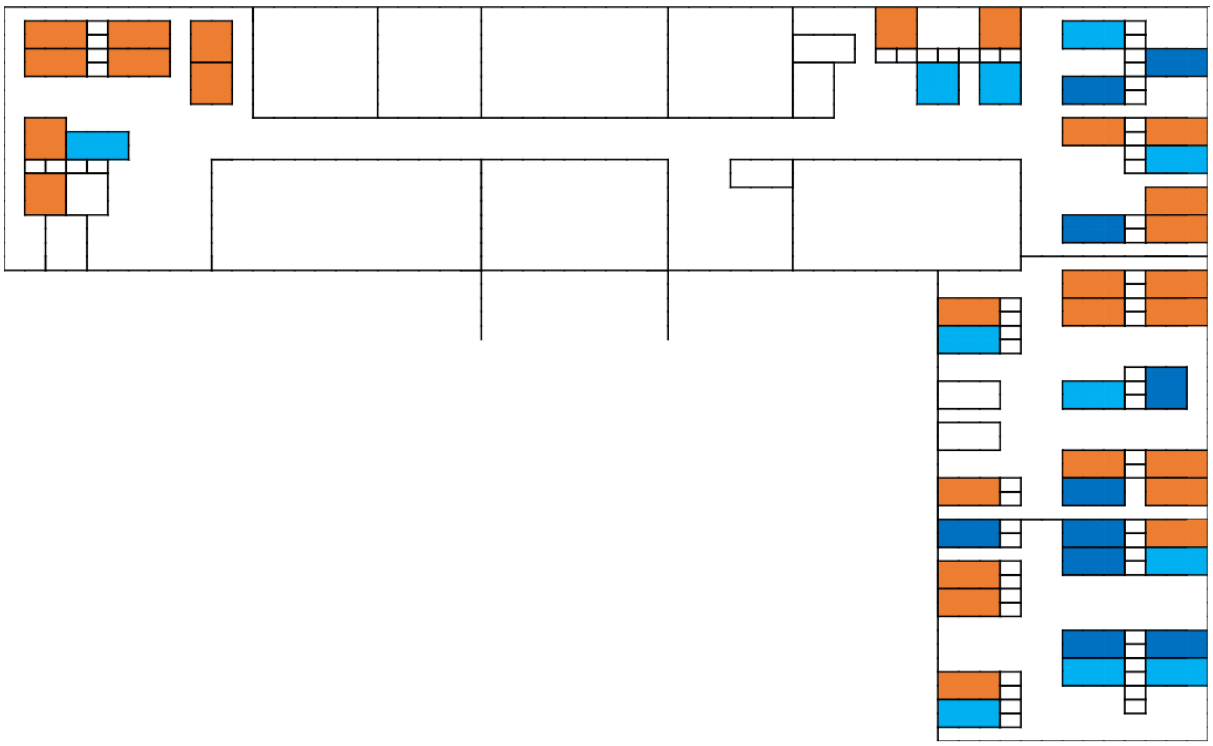

Abbildung 11.5 – Finales SVG

Leider bestehen immer noch leichte Inkonsistenzen, die nicht entfernt werden können. Die Qualität des Planes ist jedoch für die implementierten Anwendungen ausreichend.

DeskIterator

Die Aufgabe des `DeskIterator` Algorithmus besteht darin, über die ausgelesenen Zellen zu iterieren und daraus Desk-Objekte zu bilden. Dies ist notwendig, um die Schreibtische Gruppen zuzuordnen, deren Daten sauber abzuspeichern und in der Weboberfläche anzuzeigen.

Damit der Algorithmus richtig funktioniert, müssen ihm die Zell-Höhe und -Breite übergeben werden, die der SVG-Generator verwendet hat, damit er die passenden Koordinaten für die Schreibtische setzen kann. Außerdem benötigt er die Farb-Hex-Codes der Schreibtische, um Shared von Fixed-Desk unterscheiden zu können. Darüber hinaus benötigt er ein Raumplan-Objekt über dessen Zellen er iterieren kann.

Beim Iterieren über den einzelnen Zellen muss vorerst festgestellt werden, ob diese Zelle die erste Zelle eines Schreibtisches ist. Dank der `IsPartOfDesk` Variable, die beim Einlesen des Excels gesetzt wurde, kann die Applikation sofort erkennen, ob diese Zelle Teil eines Schreibtisches ist.

```
if (cell.isPartOfDesk()) {
    /* When a desk cell has a left border (or the cell before has a right border)
     and a top border it is the first cell of a desk **/
    if ((cell.hasSpecificBorder(Position.LEFT) ||
            (cellBefore != null && cellBefore.hasSpecificBorder(Position.RIGHT))) &&
            (cell.hasSpecificBorder(Position.TOP) ||
                    (cellAbove != null && cellAbove.hasSpecificBorder(Position.BOTTOM)))){
        return true;
    }
}
```

Code 11.10 - Schreibtischerkennung

Danach muss noch überprüft werden, ob die Zelle eine Grenze links und oben hat. Wegen der fehlerhaften Excel-Formatierungen kontrolliert der Algorithmus auch die Zelle darüber und links auf deren Grenzen.

Wenn der Algorithmus sich nun in einem Schreibtisch befindet, werden erstmals dessen Koordinaten gesetzt. Wie schon erwähnt, müssen diese in Koordinaten umgewandelt werden, die auch zum SVG passen.

```
desk.setSvgCoordinates(new Coordinates(cell.getCoordinates()));
desk.getSvgCoordinates().convert(this.cellWidth, this.cellHeight);
```

Code 11.11 – Konvertion der Koordinaten

Ein wichtiger Teil des Algorithmus ist die Bestimmung von Höhe und Breite der Schreibtische. Für die Höhe werden die Zellen, von der ersten Zelle des Schreibtisches aus, nach unten durchlaufen und nach einer unteren Grenze abgeprüft. Wenn diese gefunden wird, weiß der Algorithmus, dass er beim Schreibtisch ganz unten ist und sich die Höhe berechnen kann.

```
public int findHeight(Coordinates coordinates){
    int numberOfCells = 1;
    /* keep iterating until a cell with a bottom border is found **/
    int rowIndex = coordinates.getY();
    while (!this.roomPlan.getRoomcells().get(rowIndex).get(coordinates.getX())
            .hasSpecificBorder(Position.BOTTOM)
            && rowIndex < this.roomPlan.getRoomcells().size() - 1){
        numberOfCells++;
        rowIndex++;
    }
    return numberOfCells * this.cellHeight;
}
```

Code 11.12 - Auslesung der Schreibtischhöhe

Für die Breite des Schreibtisches wird das gleiche Vorgehen angewendet, nur auf horizontaler Ebene.

```
public int findWidth(Coordinates coordinates){
    int numberOfCells = 1;
    int x = coordinates.getX();
    List<RoomCell> allCellsFromRow = this.roomPlan.getRoomcells()
            .get(coordinates.getY());
    /* keep iterating until a cell with a right border is found **/
    while (!allCellsFromRow.get(x).hasSpecificBorder(Position.RIGHT)){
        numberOfCells++;
        x++;
    }
    return numberOfCells * this.cellWidth;
}
```

Code 11.14 – Auslesung der Schreibtischbreite

Nach dem Iterieren der Zellen sind alle Schreibtische in Objekten einer Liste gespeichert, die als nächstes zu Desk-Groups zugeordnet werden.

DeskGroupIterator

Der DeskGroupIterator-Algorithmus übernimmt die Aufgabe des Zuordnens der Schreibtische zu den Desk-Groups, die der Excel Parser erstellt hat. Zum Finden der korrekten Desks verwendet der Algorithmus die Koordinaten des Desk-Group Namen in den Excels. Bei den Vorkommnissen gibt es drei Szenarien, die berücksichtigt werden müssen:

- Der Name kommt zweimal vor.

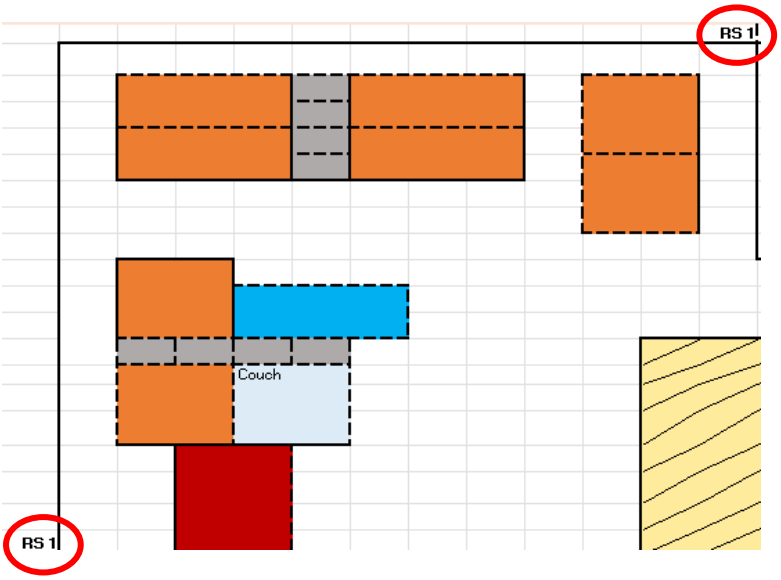

Abbildung 11.6 – Name kommt zwei Mal vor

- Der Name kommt nur einmal vor

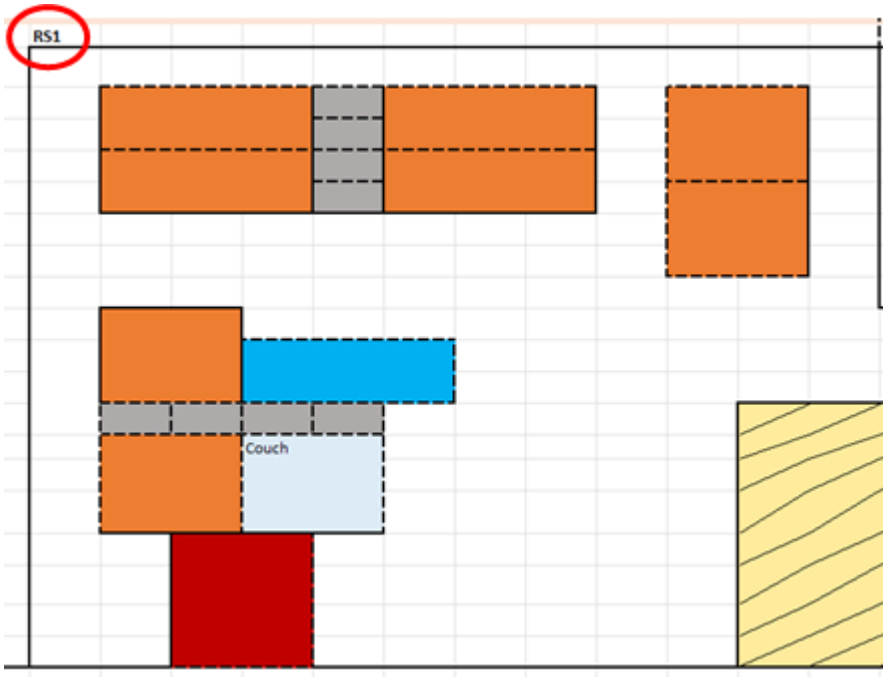

Abbildung 11.7 – Name kommt ein Mal vor

- Der Name kommt einmal vor, diese Desk-Group hat aber Subgruppen

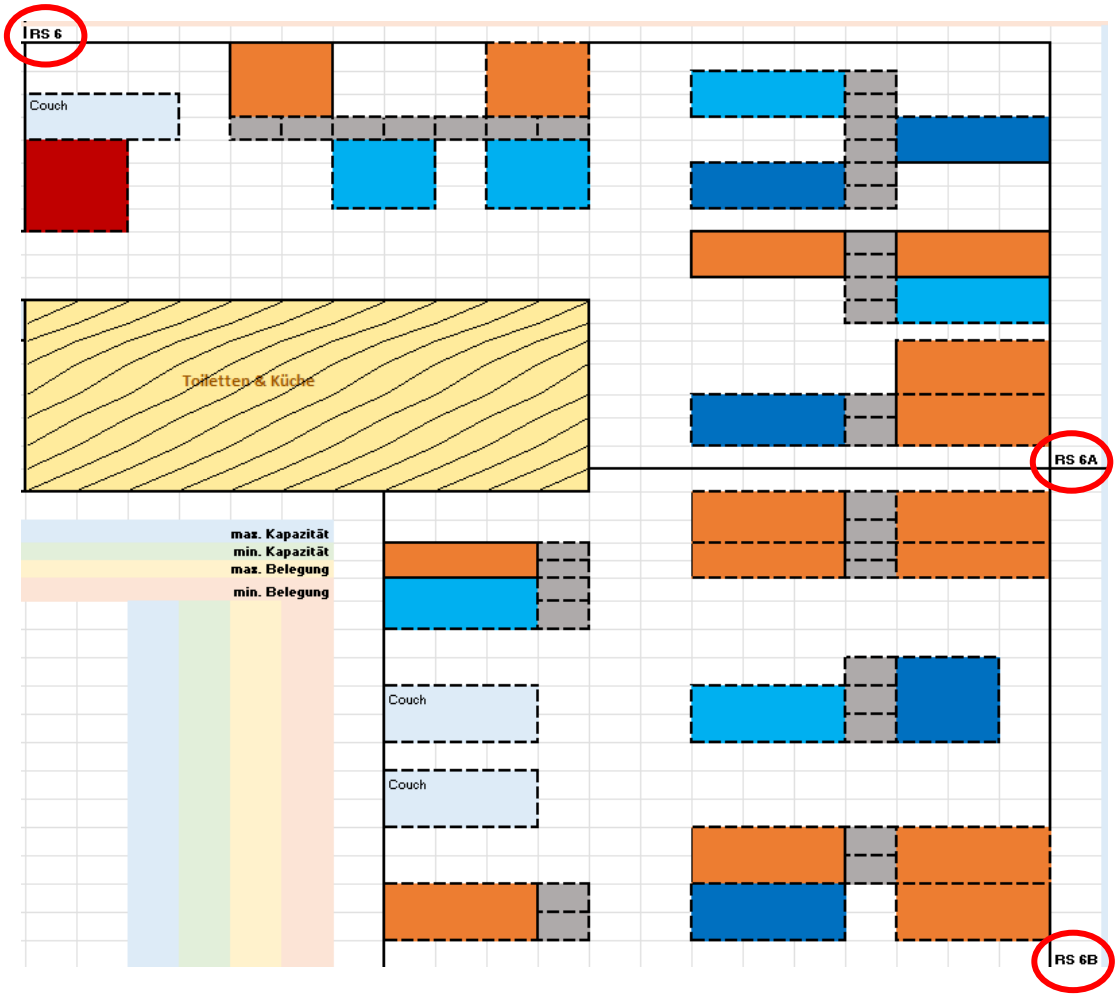

Abbildung 11.8 – Gruppe hat Subgruppen

Dieses Problem wurde wie folgt gelöst:

Name kommt zweimal vor

Bei zweimaligen Vorkommen des Namens kann das Programm einfach ein Rechteck mit den beiden Koordinaten der IDs bilden. Dies setzt voraus das die IDs sich nicht in der gleichen Reihe befinden. Beim Bilden der Koordinaten des Rechtecks muss berücksichtigt werden, dass das erste Vorkommen des Namens nicht unbedingt gleich der linken oberen Koordinate des Rechtecks sein muss. Deshalb muss beim Erzeugen des Rechtecks mit den Min- und Max-Werten der zwei Koordinatenpaare gearbeitet werden.

Bei jedem Schreibtisch wird nun überprüft, ob er sich innerhalb des Rechtecks befindet.

```
boolean deskInsideRect =
        // check if left corner OR right corner of desk is between left and right side of rectangle
        (desk.getSvgCoordinates().getX() >= rectTopLeft.getX() &&
                desk.getSvgCoordinates().getX() <= rectBottomRight.getX() ||
                deskRight >= rectTopLeft.getX() && deskRight <= rectBottomRight.getX()) &&
                // check if top corner AND bottom corner of desk is between top and bottom side of rectangle
                (desk.getSvgCoordinates().getY() >= rectTopLeft.getY() &&
                        desk.getSvgCoordinates().getY() <= rectBottomRight.getY() &&
                        deskBottom >= rectTopLeft.getY() && deskBottom <= rectBottomRight.getY());
```

Code 11.15 – Überprüfung, ob Schreibtisch Teil einer Gruppe ist

Auf der X-Achse wird überprüft, ob sich der linke oder der rechte Rand des Schreibtisches im Rechteck befindet, da es in den Excels manchmal vorkommt, dass die Tische über den imaginären Rechtecks-Rand herausragen. Dies ist aber nur auf der X-Achse der Fall, bei der Y-Achse gibt es immer eine klare Trennung und der Algorithmus kann deshalb die obere und die untere Seite des Tisches kontrollieren. Sobald alle Schreibtische zu der Desk-Group hinzugefügt worden sind, werden sie aus der Liste aller nicht zugeordneten Tische entfernt.

Name kommt einmal vor, diese Desk-Group hat aber Subgruppen

Falls die Desk-Group Subgruppen enthält, müssen ein paar extra Schritte gemacht werden. Vorab ist es wichtig zu wissen, dass Subgruppen in den Plänen ausnahmslos untereinander angeordnet sind. Desk-Groups mit Subgruppen, die nebeneinander angeordnet sind, kommen nicht vor.

Der erste Schritt ist, dass herausgefunden wird, ob die Kind-Gruppen der Elterngruppe über oder unter der ID der Gruppe angeordnet sind. In dem Bild, das oben als Beispiel angeführt ist, sind die Kind-Gruppen darunter. Diese Information wird benötigt, um die Rechtecke richtig bilden zu können. Für jede Subgruppe muss die Applikation nun die nächstgelegene ID suchen, um den Bereich richtig eingrenzen zu können. Wenn die Eltern-ID darüber ist, wird nach oben gesucht, wie beim angeführten Beispiel.

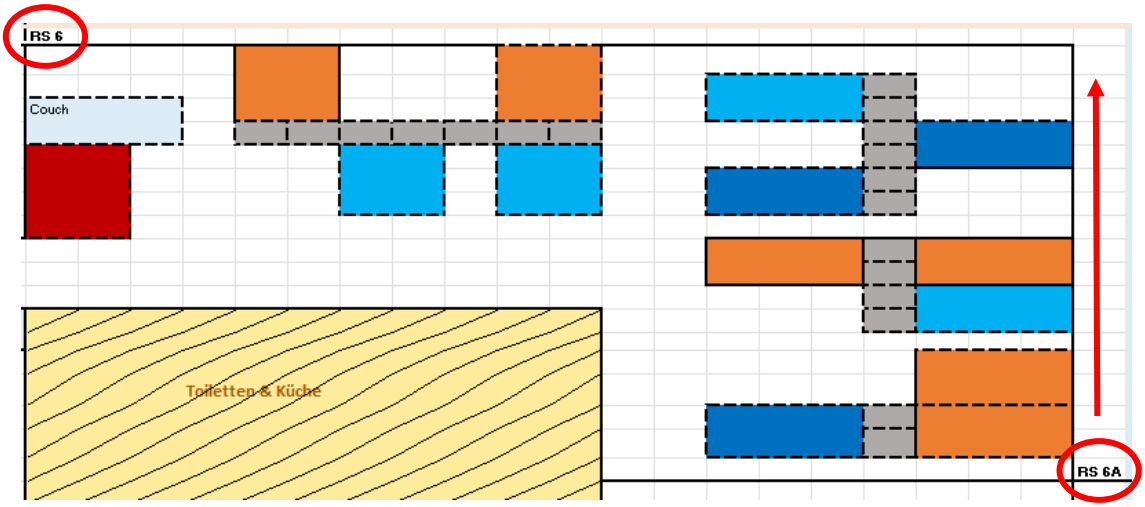

Abbildung 11.9 – ID-Suche nach oben

In diesem Fall gibt es aber keine Subgruppe, das sich darüber befindet. In diesem Fall werden die Koordinaten der Elterngruppe verwendet, um das Rechteck zu bilden. Bei der Gruppe RS6B sieht es anders aus. Wenn der Algorithmus hier nach oben sucht, stößt er auf die Gruppe RS6A. Doch hier ergibt sich ein weiteres Problem. Mit den Koordinaten von RS6B und RS6A lässt sich kein vernünftiges Rechteck bilden, da sie den gleichen x-Wert hat. Um dies zu lösen, wird der x-Wert von der Eltern-Gruppe hergenommen, mit dem alle Rechtecke der Gruppe eingegrenzt werden können.

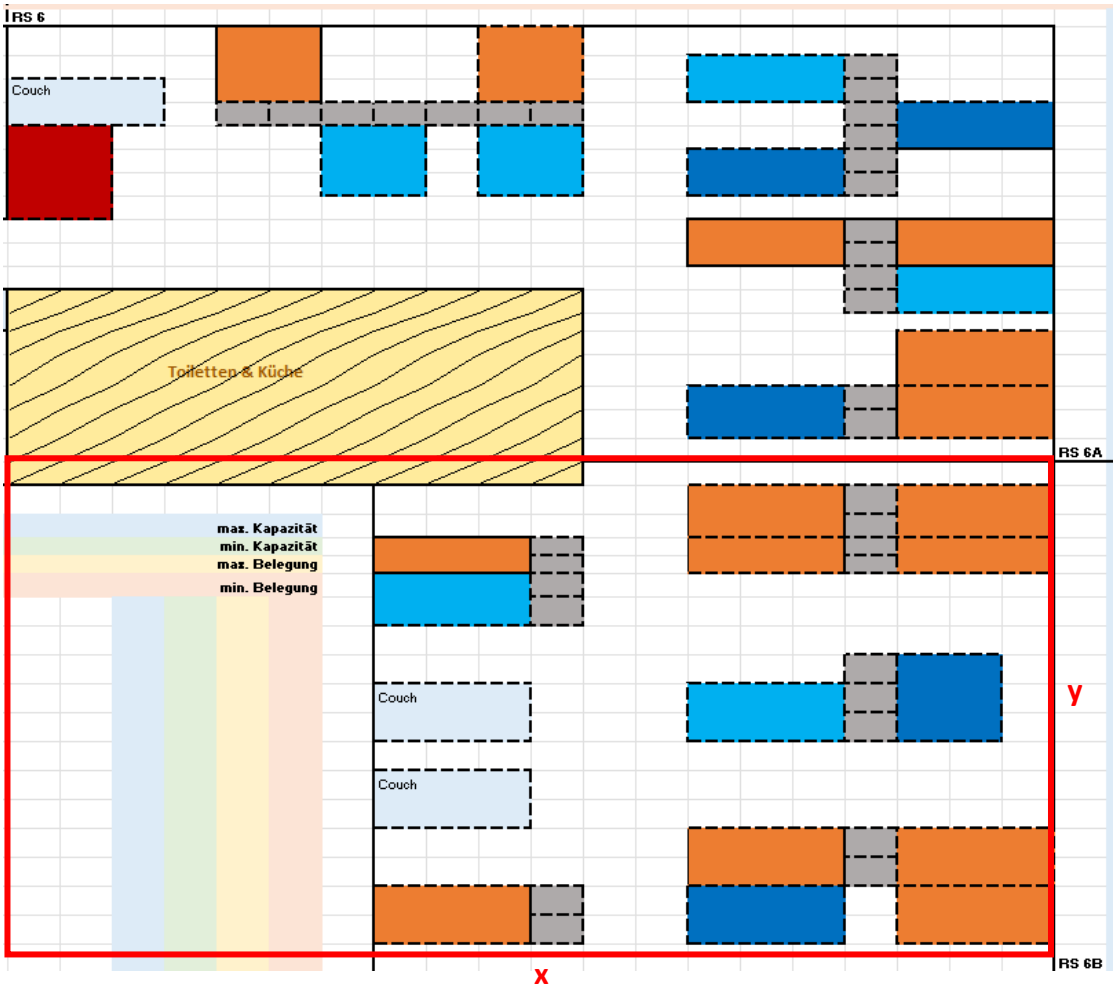

Abbildung 11.10 – imaginäres Rechteck einer Desk-Group

Dieses Verfahren wurde wie folgt implementiert.

Die Applikation sucht sich alle Koordinaten der IDs heraus, die durchsucht werden sollen.

```
List<Coordinates> coordinatesList;
/* get all the coordinates of the deskgroup ids above or below the subgroup */
if (searchAbove) {
    coordinatesList = getIDsAbove(parentGroup, subGroup);
} else {
    coordinatesList = getIDsBelow(parentGroup, subGroup);
}
```

Code 11.16 – ID-Suche

Vor dem Durchsuchen wird die y-Distanz zur Eltern-Gruppe festgehalten. Diese Distanz ist der Default und wird nur überschrieben, wenn es eine Subgruppe gibt, die eine kürzere Distanz zur aktuellen Gruppe besitzt.

```
int minYDistance = parentGroup.getIdCoordinatesOne().distanceBetweenY(subGroup.getIdCoordinatesOne());
Coordinates groupRectangleCords = new Coordinates(parentGroup.getIdCoordinatesOne());
for (Coordinates c : coordinatesList) {
    /* if the y is closer set the y in the rectangle coordinates */
    if (subGroup.getIdCoordinatesOne().distanceBetweenY(c) < minYDistance) {
        minYDistance = subGroup.getIdCoordinatesOne().distanceBetweenY(c);
        groupRectangleCords.setY(c.getY());
```

Code 11.17 – Bestimmung der kleinsten Distanz

Mit diesem Prinzip kann man alle nötigen Koordinaten heraussuchen und die Schreibtische innerhalb des Bereichs den Subgruppen zuordnen.

Name kommt nur einmal vor

Die dritte Möglichkeit, dass die ID der Desk-Groups nur einmal vorkommt, ist nur der Fall, wenn ein eindeutiger Abstand zwischen diesen und den restlichen Gruppen vorherrscht. Daher können alle übrigen Schreibtische, die nicht mit den schon genannten Verfahren zugeordnet worden sind, dieser Desk-Group zugeordnet werden. Wenn es mehrere dieser Art gibt, was durchaus vorkommt, werden die Schreibtische der am nächsten gelegenen Gruppen-ID zugeordnet.

```
/* if group has no second id and no subgroups add
remaining desks to the closest remaining group ids */
List<DeskGroup> groupsWithoutDesks = getGroupsWithNoDesks();
for (Desk desk: deskList){
    DeskGroup deskGroup = getClosestGroupWithoutDesks(desk, groupsWithoutDesks);
    if (deskGroup!=null){
        deskGroup.getDesks().add(desk);
    }
}
```

Code 11.18 - Schreibtische nächst gelegener Gruppe zuordnen

11.1.2 Anzeige im Frontend

Backend

Die beschriebenen Algorithmen werden von dem `RoomPlanService` verwaltet, dessen Interface folgende Funktionen beinhaltet:

```
public interface RoomPlanService {

    File getRoomplanSvg(long floorID) throws FileNotFoundException;
    void generateRoomplanSvg(File excelFile, int sheetNum, long floorID);
    void generateAllRoomplansSvg(String path);
    String getRoomplanPath();
    void saveRoomplanPath(RoomplanPath roomPlan);
}
```

Code 11.19 - RoomPlanService Interface

Die `getRoomplanSvg`-Methode retourniert das fertige SVG-Image File anhand der Stockwerk-ID. Die Files werden in der Ordnerstruktur der Applikation mit Standortnamen und Stockwerk abgespeichert und können dadurch leicht vom Programm verwaltet werden.

Hier wird beispielsweise das File aufgerufen und falls es nicht existiert, eine Exception geworfen.

```
roomPlanSvg = new File( pathname: "./svg/" + name + "_" + level + ".svg");
if (!roomPlanSvg.exists()){
    throw new ResourceNotFoundException("Svg does not yet exist. Generate it first");
}
```

Code 11.20 - SVG-Image aufrufen

Die `generateRoomplanSvg`-Funktion generiert und speichert das Image anhand des übergebenen Excel-Files. Hier werden auch die Regular-Expressions für das Einlesen der Schreibtischnamen festgelegt.

```
Map<String, String> deskGroupRegex = new HashMap<>();
deskGroupRegex.put("Salzburg", "^D-.*$");
deskGroupRegex.put("Hagenberg", "^RS.*$");
```

Code 11.21 - Regex für Gruppennamen erstellen

Die Funktion speichert auch sämtliche erforderlichen Daten der Schreibtische - ihre Koordinaten und Größe - und erstellt gegebenenfalls neue Desk-Gruppen in der Datenbank, falls diese noch nicht vorhanden sind.

Die Methode `generateAllRoomplansSvg` erhält einen Pfad als Parameter, der entweder auf einen Ordner verweist, in dem mehrere Excel-Dateien liegen, oder auf eine einzelne Excel-Datei zeigt, die jedoch mehrere Arbeitsblätter enthält, auf denen die Pläne dargestellt sind. Dies kann man anhand der Benennung der Files erkennen.

Wenn man auf den Namen der Datei stößt, wo mehrere Pläne zu finden sind, wird über die einzelnen Arbeitsblätter iteriert und mit deren Namen weitergearbeitet. Diese sind jedoch nur Kürzel und nicht der ausgeschriebene Standortname. Daher wurde die Tabelle für Locations in der Datenbank um eine Spalte mit den Kürzeln erweitert, um der Applikation die richtige Zuordnung zu ermöglichen.

Der Pfad für diese Pläne wird in der Datenbank abgespeichert, damit die Applikation die Bilder bei Bedarf aktualisieren kann. Für das Ändern und Abrufen stehen Service Methoden zur Verfügung. Wenn bei der `generateAllRoomplansSvg`−Methode ein Pfad übergeben wird, der nicht dem in der Datenbank gleicht, wird der Datenbankeintrag aktualisiert.

Der Raumplan-Controller Klasse stellt die Verbindung zwischen Service und Frontend in Form einer REST-API her. Die SVG-Images werden auf folgende Weise verschickt:

```
Resource file = new UrlResource(roomPlanService.getRoomplanSvg(floorId).toURI());
return ResponseEntity.ok()
        .header(HttpHeaders.CONTENT_DISPOSITION,
                ...headerValues: "attachment; filename=\"" +
                        file.getFilename() + "\"").body(file);
```

Code 11.22 - Datei über REST-API verschicken

Es gibt noch weitere Endpoints, die auf die Funktionen des Services zugreifen, wie zum Beispiel der Update Endpoint, wo lediglich alle Excel-Files nochmal durchiteriert und die Bilder neu erstellt werden. Diese wird auch nach dem Ändern des Pfades in der Admin Ansicht aufgerufen, mit dem Unterschied, dass hier der neue Pfad mitübergeben wird. Der alte Pfad, der in der Datenbank abgespeichert ist, wird mit diesem ersetzt.

Frontend

Sobald ein Standort und ein Stockwerk im Home-Screen ausgewählt werden, muss der zugehörige Raumplan angezeigt werden. Sobald sich also die ID des ausgewählten Stockwerks ändert, wird das Image heruntergeladen. Mithilfe eines DOMParsers wird die Antwort, die von der API kommt, von einem String zu einem Document mit dem Typ SVG umgewandelt.

```
this.roomService.getRoomplan(this.floorID).subscribe( observerOrNext: response : string | Error => {
    if (typeof response === "string") {
        var dom : Document = new DOMParser().parseFromString(response, type: "image/svg+xml");
```

Code 11.23 - Datei empfangen

Damit die Markierungen zum Bild richtig hinzugefügt werden können, muss das `viewBox` Attribut gesetzt werden. Dadurch wird eine Art Koordinatensystem über das Bild gelegt und die Position sowie Dimension festgelegt. (vgl. Mediaevent: SVG-Viewport, SVG-Koordinaten und viewBox, 2023)

```
svgDom.setAttribute( qualifiedName: "viewBox", value: "0, 0," +
    svgDom.getAttribute( qualifiedName: "width") + ", " +
    svgDom.getAttribute( qualifiedName: "height"))
```

Code 11.24 - viewBox Attribut setzen

Danach werden noch ein paar Attribute gesetzt damit das Bild den Platz zur Gänze aufnimmt und skalierbar ist.

```
svgDom.setAttribute( qualifiedName: "preserveAspectRatio", value: "xMinYMin")
svgDom.setAttribute( qualifiedName: "height", value: "100%")
svgDom.setAttribute( qualifiedName: "width", value: "100%")
```

Code 11.25 - Attribute für Skalierbarkeit setzen

Wenn über einer Schreibtischgruppe gehovert wird, müssen die Markierungen dieser Tische dem Raumplan-Dokument hinzugefügt werden. Sobald das Stockwerk ausgewählt wird, werden alle Gruppen und ihre Desk-Infos abgerufen, daher ist kein zusätzlicher API-Aufruf erforderlich. Es wird über alle Schreibtische iteriert und für jeden wird ein Rechteck erstellt und gespeichert.

```
deskgroup.deskSvgInfos?.forEach( callbackfn: deskInfo : DeskSvgInfo  => {
    const e : HTMLElement  = document.createElement( tagName: 'rect');
    e.setAttribute( qualifiedName: "x", deskInfo.x!.toString())
    e.setAttribute( qualifiedName: "y", deskInfo.y!.toString())
    e.setAttribute( qualifiedName: "width", deskInfo.width!.toString())
    e.setAttribute( qualifiedName: "height", deskInfo.height!.toString())

    e.setAttribute( qualifiedName: "stroke-width", value: "6")
    e.setAttribute( qualifiedName: "style", value: "fill: rgb(255, 102, 153); " +
        "stroke-width: 3; stroke: rgb(255, 102, 153)");
    deskSvgDom.appendChild(e);
})
```

Code 11.26 - Rechtecke für die Schreibtische hinzufügen

Im HTML-Code wird dann das Bild wie folgt angezeigt.

```
<div style="width: 100%; height: 100%; padding: 8px;"
     [innerHTML]="completeSvg"></div>
```

Code 11.27 - Bild im HTML

Wenn der Mauszeiger die Gruppe verlässt, werden die Rechtecke wieder gelöscht. Sollte kein Raumplan vorhanden sein, wird eine entsprechende Nachricht in dem Bereich angezeigt, wo normalerweise der Raumplan dargestellt wird.

11.2 Mitarbeiter Info

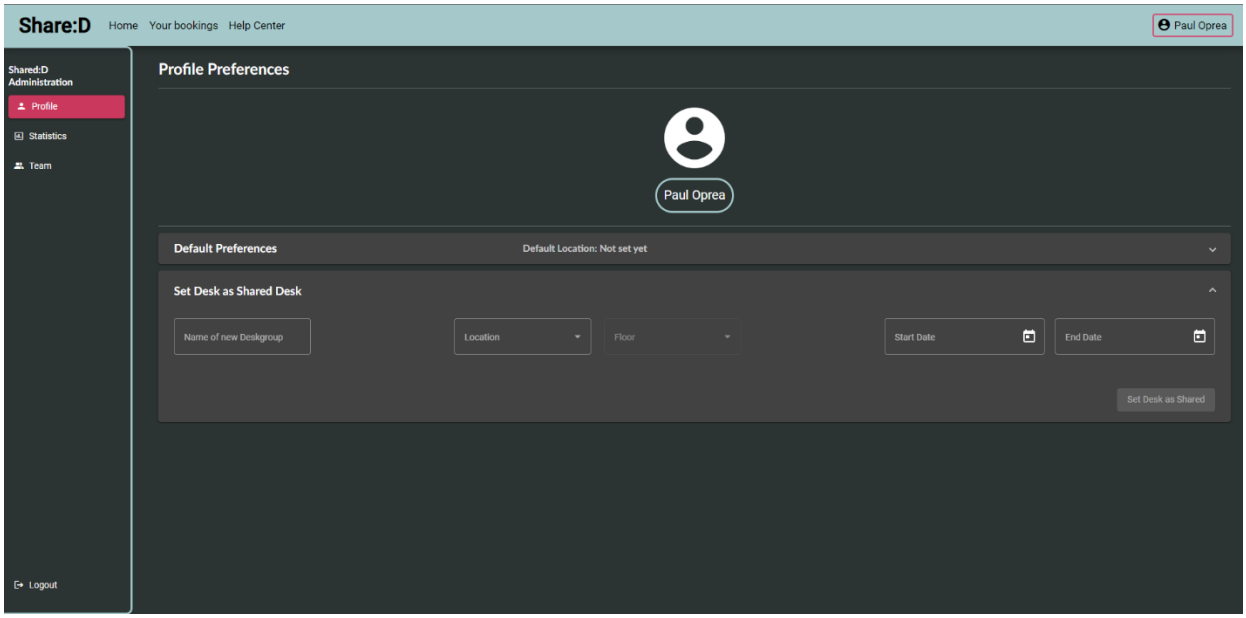

Abbildung 11.11 - Mitarbeiter Info

Einem normalen Angestellten ohne Leitungsfunktion steht nur die Benutzer-Ansicht zur Verfügung.

Diese kann er dann aber im vollen Ausmaß bedienen, d.h. er kann seine “Default Location” setzen. So muss er dann nicht mehr bei jeder Buchung auswählen, an welchem Standort und in welchem Stockwerk genau er einen Platz buchen will. Auf dieses Feature wird aber später nicht mehr eingegangen, da es kein Projektmitglied implementiert hat. Der Fixed-Shared-Desk-Switch steht ebenfalls jedem Nutzer zur Verfügung und wird im letzten der drei Unterkapitel näher beleuchtet.

11.2.1 Team-Übersicht

Die Team-Übersicht besteht aus zwei Angular-Komponenten:

- Team-Member
- Team-Page

Sie sind voneinander unabhängig und können nach Belieben ausgetauscht oder woanders wiederverwendet werden, ohne dass die anderen Komponenten dadurch verändert werden oder nicht mehr funktionsfähig sind.

Team-Member

Diese Komponente enthält alle Daten des Mitarbeiters, die für dessen Team- oder Abteilungsleiter von Interesse sein könnten, nämlich:

- Vor- und Nachname
- Ist er im Home-Office, vor Ort oder aktuell abwesend
    - Auf den ersten Blick für den heutigen Tag erkennbar, durch die Farbe des Mitarbeiter-Kärtchens
    - Sobald man mit dem Mauszeiger über den jeweiligen Angestellten hovered, erscheint eine Anzeige, die für jeden Wochentag wiedergibt, wo sich die Person befunden hat, befindet oder befinden wird.
- Wenn vor Ort:
    - Standort
    - Stockwerk

Abbildung 11.12 - Die Team-Member Komponente

Hellblau = In-House Dunkelblau = Home-Office Rosarot = Abwesend

Bei der Farbwahl für diese drei verschiedene Statusmöglichkeiten war es wichtig, dass sie sich klar voneinander unterscheiden, damit dies zu einer raschen Identifizierung des Status führt.

Gleichzeitig sollten sie aber immer noch angenehm zum Ansehen sein und nicht die Ästhetik des restlichen Farbedesigns korrumpieren. Dieses besteht bei Share:D vor allem aus blauen bzw. eher „kalten“ Farben. Um zum Beispiel die Auswahl des Benutzers herausstechen zu lassen, werden aber rötliche Töne verwendet.

Deswegen befinden sich zwei der Farben im blauen Spektrum und die dritte im roten, um besondere Aufmerksamkeit auf die abwesenden Mitarbeiter zu lenken. Dadurch wird dem Nutzer ein Gefühl der Ordnung und Harmonie vermittelt und dies ist gleichzeitig praktisch für die Arbeit.

Das Mitarbeiter-Kärtchen wird durch folgenden HTML-Code beschrieben:

```
<div fxLayout="row" fxLayoutAlign="center center">
    <div class="member-div" (mouseleave)="triggerLeftSlide()"
         [ngStyle]="{'background-color': getStatusColor(getCurrentWeekDay())}">
        <div fxLayoutAlign="center center" class="profile-div">
            <mat-icon class="profile">account_circle</mat-icon>
        </div>
        <div fxLayout="column" fxLayoutAlign="center start" class="info-div">
            <p style="...">{{ member.firstName }}  {{ member.lastName }}</p>
            <div class="location-div">
                <p>{{ location }}</p>
            </div>
        </div>
    </div>
```

Code 11.28 - HTML Code der Team-Member Komponente

Durch das `ngStyle` im Head vom div (eine HTML-Komponente, die meistens verwendet wird, um mehrere HTML-Komponenten zusammenzufassen), wird die Hintergrundfarbe des Kärtchens dynamisch auf die gesetzt, welche die `getStatusColor`-Funktion im Moment zurückgibt. Als Parameter der Funktion wird der Rückgabewert von `getCurrentWeekDay()` verwendet, welcher, wie der Name schon verrät, der aktuelle Wochentag in Form einer Zeichenkette ist.

Die Logik der `getStatusColor`-Funktion sieht wie folgt aus:

```
getStatusColor(weekday: string) {
  let inHouse: string = '#A5C9CA';
  let homeOffice: string = '#607bf0';
  let absent: string = '#cc4a96';
  var currentStatus: string = "Absent";

  switch (weekday) {
    case 'Monday':
      currentStatus = this.member.weekStatus!.monday;
      break;
    case 'Tuesday':
      currentStatus = this.member.weekStatus!.tuesday;
      break;
    case 'Wednesday':
      currentStatus = this.member.weekStatus!.wednesday;
      break;
    case 'Thursday':
      currentStatus = this.member.weekStatus!.thursday;
      break;
    case 'Friday':
      currentStatus = this.member.weekStatus!.friday;
      break;
  }

  switch (currentStatus) {
    case "InHouse":
      return inHouse;
    case "HomeOffice":
      this.location = ""
      return homeOffice;
    default:
      this.location = ""
      return absent;
  }
}
```

Code 11.29 - Implementation der getStatusColor-Funktion

Der Parameter weekday beinhaltet den Rückgabewert von `getCurrentWeekDay()`. Die ersten drei Variablen speichern die möglichen Farben die, der Hintergrund des Kärtchens annehmen kann, je nach Status. Die vierte Variable speichert standardmäßig den Status “Absent”, welchen Sinn das hat, wird am Ende des Absatzes klar werden.

Als nächstes wird anhand von `weekday` bestimmt, welcher Wochentag, des Status, der aktuelle ist und in `currentStatus` gespeichert.

Danach wird in Abhängigkeit von dem Wert von `currentStatus` bestimmt welche Farbe zurückgegeben werden sollte.

Würde aber nun jemand am Wochenende einen Blick auf die Team-Übersicht werfen, so würde `currentStatus` keine Veränderung erfahren und auf "Absent" bleiben. Dadurch würden wiederum alle Angestellten als abwesend angezeigt werden, was normalerweise der Realität entspricht, wodurch die Integrität dieser Anzeige gewährleistet wird.

Der unterschiedliche Status der Wochentage sieht im HTML-Code beispielhaft so aus:

```
<div fxLayout="row" fxLayoutAlign="space-evenly center" class="ext-info"
     [ngClass]="{'slide-left-vessel': hoverEnded == true}" (animationend)="resetHover()">
    <div fxLayoutAlign="center center" class="weekday-selected"
         [ngStyle]="{'color': mondayColor, 'border': '3px solid ' + mondayColor}">
        <mat-icon svgIcon="m-icon" class="week-day-cap"></mat-icon>
        <mat-icon svgIcon="o-icon" class="week-day"></mat-icon>
    </div>
    <div fxLayoutAlign="center center" class="weekday-selected"
         [ngStyle]="{'color': tuesdayColor, 'border': '3px solid ' + tuesdayColor}">
        <mat-icon svgIcon="d-icon" class="week-day-cap"></mat-icon>
        <mat-icon svgIcon="i-icon" class="week-day"></mat-icon>
    </div>
```

Code 11.30 - HTML Code der Wochenansicht

Für die Farbbeschaffung wird dieselbe Logik verwendet, wie vorher auch beim Mitarbeiter-Kärtchen. Beim Erstellen des Angestellten wird `getStatusColor` auf jeden Tag aus der Arbeitswoche angewendet. Die Ergebnisse werden dann in den Variablen `mondayColor`, `tuesdayColor` usw. gespeichert.

Damit die Übersichtlichkeit der Team-Übersicht nicht zu Schaden kommt, wird diese Wochen-Ansicht nicht statisch bei jedem einzelnen Angestellten von Anfang an angezeigt. Stattdessen besitzt diese Ansicht ein dynamisches Verhalten und wird erst angezeigt, wenn man mit dem Mauszeiger drüber hovered, also in anderen Worten: Nur dann, wenn es der Benutzer auch wirklich sehen will!

Dann wird eine Animation gestartet, die die Wochentage aus dem Kärtchen, nacheinander gleiten lässt. Verantwortliche CSS-Klasse:

```
.member-div:hover + .ext-info {
    animation: slide-right 0.8s;
    visibility: visible;
}
```

Code 11.31 - CSS Klasse für die Animation

Solange der Mauscursor innerhalb der Komponente bleibt, verbleibt die Anzeige des Status. Sobald diese Bedingung nicht länger zutrifft, wird die Animation invertiert und die Anzeige der Wochentage verschwindet.

Damit das Programm weiß, wann es Zeit für welche Animation ist, ist im Head vom div des Kärtchens das Event `mouseleave` an die Funktion `triggerLeftSlide()` gebunden.

```
triggerLeftSlide() : void  {
  this.hoverEnded = true;
  console.log("true");
}
```

Code 11.32 - Implementation der trigger-LeftSlide-Funktion

Dadurch wird immer die Variable `hoverEnded` auf `true` gesetzt, wenn die Maus nicht mehr Kontakt zum Mitarbeiter-Kärtchen hat. Da uns Angular ermöglicht CSS-Klassen dynamisch zuzuweisen, wird durch [`ngClass`] geprüft, ob die Maus noch über dem Mitarbeiter hovered, und weisen in dem Fall die CSS-Klasse `slide-left-vessel`. Diese startet die invertierte Animation und sorgt dafür, dass die Wochenansicht wieder verschwindet.

```
.slide-left-vessel {
    animation: slide-left 0.9s;
    visibility: visible;
}
```

Code 11.33 - CSS Klasse für die Invers-Animation

Die beiden Animationen “slide-left” und “slide-right” selbst sind als folgende Keyframes definiert:

```
@keyframes slide-right {
    0% {
      transform: translateX(-100%);
    }
    100% {
      transform: translateX(0);
    }
  }

  @keyframes slide-left {
    0% {
      transform: translateX(0);
    }
    100% {
      transform: translateX(-100%);
    }
  }
```

Code 11.34 - CSS Code der Keyframes

Dieser CSS-Code definiert zwei Animations-Schlüsselrahmen (keyframes), die jeweils eine Bewegung in horizontaler Richtung durchführen.

Beide Animationen haben zwei Schlüsselrahmen, einer bei 0% und der andere bei 100%. Diese geben den Start- und Endpunkt der Animation an. Innerhalb dieser Schlüsselrahmen wird der Wert der "transform" Eigenschaft verwendet, um das Element horizontal zu verschieben. Der Wert "translateX" wird verwendet, um die horizontale Translation zu steuern. Das Negative (-100%) bewirkt eine Verschiebung nach links, während das Positive (0%) das Element an seiner aktuellen Position belässt.

Obwohl also die ursprüngliche Position der Wochenansicht neben dem Mitarbeiter-Kärtchen ist, fällt das dem Nutzer nicht auf da die Wochentage erst mit Anfang der Animation sichtbar sind, da die `visibility` dieser Ansicht normalerweise auf `hidden` gesetzt ist.

```css
.ext-info {
    visibility: hidden;
    pointer-events: none;
    z-index: -1;
}
```

Code 11.35 - CSS Klasse der Wochenansicht

Aufgrund des dahinterliegenden Datenmodells wird beim Mitarbeiter nicht sein Standort als Zeichenkette gespeichert. Deswegen sind die Funktionen `getCurrentLocation` und `getAddressOfFloor` notwendig.

```
getCurrentLocation() : void  {
  this.bookingService.getLatestUserBooking(this.member.email!)
      .subscribe( observerOrNext: response : Booking | globalThis.Error  => {
    if (response instanceof HttpErrorResponse) {
      this.error = new Error(response.status, response.error.title, response.error.message);
      this.location = this.member.defaultLocation!.name!.toString();
    }

    else {
      var booking : Booking  = response as Booking;
      this.location = this.getAddressOfFloor(booking.deskGroup?.floor?.id!)
          + " - Floor " + booking.deskGroup?.floor?.level;
    }
  });
}

1+ usages  PORSCHE_PHDOM\D6IBK1A +1
getAddressOfFloor(floorId: number): string {
  var address: string = "";
  this.locations.forEach(location : Location  => {
    location.floors?.forEach( callbackfn: floor : Floor  => {
      if (floor.id === floorId) {
        address = location.name!;
      }
    });
  });
  return address;
}
```

Code 11.36 - Implementation der beiden Funktionen

Erstere der beiden tätigt einen API-Abruf, um die letzte Buchung des jeweiligen Angestellten zu erhalten. Diese Buchung enthält die einmalige Identifikationsnummer des Stockwerkes, in dem sich der gebuchte Arbeitsplatz befindet. Die Funktion `getAddressOfFloor` überprüft

dann welcher Standort innerhalb der `locations`-Variable dieses Stockwerk besitzt und gibt jenes zurück.

Team-Page

Hier werden, je nachdem ob der Nutzer ein Team- oder Abteilungsleiter ist, die Mitarbeiter des eigenen Teams oder der Abteilung spaltenweise angezeigt.

Abbildung 11.13 - Die Team-Page Komponente

Die unterstellten Mitarbeiter des Nutzers werden dynamisch geladen und dann wird für jeden dieser Angestellten ein `Team-Member` erstellt und in diesem div angezeigt:

```
<div class="content members-viewport">
    <mat-grid-list [cols]="breakpoint" (window:resize)="onResize($event)"
                   rowHeight="100px" gutterSize="20px">
        <mat-grid-tile *ngFor="let member of displayMembers">
            <SDP-team-member [member]="member"
                             [locations]="locations"></SDP-team-member>
        </mat-grid-tile>
    </mat-grid-list>
</div>
```

Code 11.37 - Auflisten der Mitarbeiter im HTML

Es wird für jedes Mitarbeiter-Kärtchen nicht nur der Mitarbeiter übergeben, sondern auch die verschiedenen Standorte der Porsche Informatik. Durch die vorherig beschriebene Funktion `getAddressOfFloor` wird dann herausgefunden an welchem Standort sich jener Angestellte

befindet. Dadurch muss die API nur einmal, am Anfang der Erstellung von Team-Page, die Standorte abrufen und nicht für jeden Mitarbeiter einzeln, was vor allem bei größeren Teams nicht effizient wäre.

In `breakpoint` wird gespeichert in wie vielen Spalten die Angestellten gestapelt werden sollten.

Zuerst wird die Größe des Browserfensters in Pixel übergeben. Danach wird davon die Breite der seitlichen Menüleiste abgezogen und durch die Größe eines Mitarbeiter-Kärtchens mit offener Wochenansicht dividiert.

Sobald das Fenster vergrößert oder verkleinert wird, wird die Funktion `onResize` durch Event-binding aufgerufen, welche `setGridColumns` ausführt und die Größe des veränderten Fensters übergibt.

```
onResize(event: any) : void  {
  this.setGridColumns(event.target.innerWidth);
}
```

Code 11.38 - Implementation der onResize-Funktion

Oberhalb des Bereiches, wo alle Mitarbeiter angezeigt werden, befindet sich eine Zeile, wo ganz links “Team” steht, in der Mitte sich ein Dropdown befindet und ganz rechts eine Legende dem Nutzer zu verstehen gibt welche Farbe, für welchen “Status” steht.

```
<div *ngIf="!error" fxLayout="row" fxLayoutAlign="space-between center">
    <h2 [ngClass]="{'teamleader-styling': isTeamLeader,
    'departmenthead-styling': !isTeamLeader}">Team Members</h2>

    <mat-form-field appearance="outline" *ngIf="!isTeamLeader">
        <mat-label>Team</mat-label>
        <mat-select [value]="selectedTeam" (selectionChange)="onTeamChange($event)">
            <mat-option value="My Team">
                My Team
            </mat-option>
            <mat-option *ngFor="let team of teams" [value]="team">
                {{ team }}
            </mat-option>
        </mat-select>
    </mat-form-field>

    <p fxLayout="row" [ngClass]="{'teamleader-styling': isTeamLeader,
    'departmenthead-styling': !isTeamLeader}">
        <span class="circle" style="background-color: #A5C9CA;"></span>
          
        <span>In-House</span>
              
        <span class="circle" style="background-color: #607bf0;"></span>
          
        <span>Home-Office</span>
              
        <span class="circle" style="background-color: #cc4a96;"></span>
         
        <span>Absent</span>
    </p>
</div>
<mat-divider class="custom-divider" />
```

Code 11.39 - HTML Code des Layouts

Anhand der `TeamLeader`-Variable wird dem Inhalt dieser Zeile dynamisch eine CSS-Klasse zugewiesen. Falls der Nutzer nämlich ein Abteilungsleiter ist, braucht er das mittlere Dropdown, damit er eines von den mehreren Teams, die ihm unterstellt sind, auswählen kann.

Ob es sich um einen Teamleiter handelt, wird folgend festgestellt:

```
this.employeeService.getTeamMembers(this.currentUser.email!)
    .subscribe( observerOrNext: res : globalThis.Error | Employee[]  => {
  this.members = res as Employee[];
  this.displayMembers = this.members;
  this.checkIfTeamLeader();
  if (!this.isTeamLeader) this.getAllUnderlyingTeams(this.members);
});
```

Code 11.40 - Implementation der Überprüfung

Innerhalb der `ngOnInit`-Funktion, welche automatisch ausgeführt wird, sobald diese Seite erstellt werden muss, wird über die API, nach den Angestellten aus dem Team des Nutzers gefragt und diese in der `members`-Variable gespeichert. Selbst bei einem Abteilungsleiter werden hier nur die Mitarbeiter zurückgegeben, die ihm direkt untergeordnet sind.

Danach wird überprüft, ob der Nutzer ein Teamleiter ist:

```
checkIfTeamLeader(): boolean {
  let isTeamLeader: boolean = true;
  this.members.forEach((employee : Employee ) : void  => {
    if (employee.isManager) isTeamLeader = false;
  })

  return isTeamLeader;
}
```

Code 11.41 - Implementation der checkIfTeamLeader-Funktion

Dafür wird durch eine Schleife überprüft ob zumindest einer, der direkt unterstellten Mitarbeiter, auch ein Teamleiter ist und dies dann zurückgegeben.

Ist dies der Fall wird die Funktion `getAllUnderlyingTeams` aufgerufen und diese Angestellten als Parameter übergeben.

Diese ist wie folgend implementiert:

```
getAllUnderlyingTeams(members: Employee[]) : void  {
  members.forEach( (employee : Employee ) : void  => {
    if (!this.teams.includes(employee.department!) && employee.department != null)
      this.teams.push(employee.department!);
    if (employee.isManager) {
      this.employeeService.getTeamMembers(employee.email!)
          .subscribe( observerOrNext: res : globalThis.Error | Employee[]  => {
        this.getAllUnderlyingTeams(res as Employee[]);
      })
    }
  });
}
```

Code 11.42 - Implementation der getAllUnderlyingTeams-Funktion

Es wird für jeden direkt unterstellten Angestellten überprüft, ob der Name seines Teams als Zeichenkette bereits in `teams` vorkommt, diese speichert nämlich die Namen aller Teams, die dem Abteilungsleiter unterstellt sind. Ist dies nicht der Fall wird jener Teamname hinzugefügt und beim Dropdown zur Auswahl stehen für den Benutzer. Falls es sich beim Mitarbeiter um jemanden handelt, der ebenfalls ein Team leitet, werden seine Teammitglieder durch die API abgerufen und als Parameter erneut an die `getAllUnderlyingTeams`–Funktion übergeben. Auf diese Weise werden rekursiv die Namen der Teams, unabhängig davon, ob dem sie dem Abteilungsleiter direkt oder indirekt unterstellt sind, herausgefunden und gespeichert.

Sobald der Nutzer ein anderes Team beim Dropdown auswählt, wird über Eventbinding die Funktion `onTeamChange` getriggert.

```
onTeamChange(event: any): void {
  this.selectedTeam = event.value;

  if (this.selectedTeam != 'My Team') {
    this.findTeam(this.members);
  }

  else {
    this.displayMembers = this.members;
  }
}
```

Code 11.43 - Implementation der onTeamChange-Funktion

Es wird überprüft, ob die neue Auswahl das direkt unterstellte Team des Nutzers ist. Ist dies der Fall wird die `displayMembers`-Variable mit den Angestellten von `members` befüllt. Der Nutzer sieht immer den Inhalt von `displayMembers`, während `members` immer die direkt unterstellten Mitarbeiter beinhaltet.

Dies ist wichtig, da die findTeam-Funktion ausgehend vom höchsten Team in der Hierarchie sucht, aus diesem Grund werden die direkt unterstellten Mitarbeiter zusätzlich in members gespeichert und dadurch jederzeit zugänglich für `findTeam`.

```
findTeam(members: Employee[]) : void  {
  members.forEach((employee : Employee ) : void  => {
    if (employee.department === this.selectedTeam && employee.isManager) {
      this.employeeService.getTeamMembers(employee.email!)
          .subscribe( observerOrNext: res : globalThis.Error | Employee[]  => {
        this.displayMembers = res as Employee[];
        console.log("new team size is: " + this.displayMembers.length);
        this.displayMembers.push(employee);
      });
    }

    else if (employee.isManager) {
      this.employeeService.getTeamMembers(employee.email!)
          .subscribe( observerOrNext: res : globalThis.Error | Employee[]  => {
        this.findTeam(res as Employee[]);
      })
    }
  })
}
```

Code 11.44 - Implementation der findTeam-Funktion

Es wird jeder Angestellte in einer Schleife durchlaufen. Dabei wird kontrolliert, ob er dem Team zugeteilt ist, welches auch vom Benutzer ausgewählt wurde und ob er der Leiter dieser Gruppe ist.

Wenn beide Bedingungen zutreffen, werden über die API seine Teammitglieder abgerufen und als die neuen `displayMembers` gespeichert. Der Teamleiter selbst wird aber auch noch hinzugefügt, sodass er ebenfalls dem Abteilungsleiter angezeigt wird.

Im Fall, dass der zu überprüfende Angestellte ein Teamleiter ist, aber nicht des gesuchten Teams, wird die Funktion `findTeam` rekursiv aufgerufen und seine direkt unterstellten Mitarbeiter als Parameter übergeben. Dadurch wiederholen sich die bisher beschriebenen Schritte, bis die Person gefunden wird, welche das gesuchte Team leitet und jene Mitarbeiter die diesen als direkten Vorgesetzen haben, in `displayMembers` gespeichert wurden.

Testen

Da die Team-Übersicht nur von Team- und Abteilungsleitern eingesehen werden kann, sind im Backend Änderungen vorgenommen worden, durch die es dem dafür zuständigen Entwickler ebenfalls möglich ist, diese Ansicht zu verwenden.

Dies ist wichtig damit

- während der Entwicklung, der Zuständige kontrollieren kann ob die durchgeführten Änderungen, die gewünschten Ergebnisse erzielt haben.
- beim Aufkommen eines Fehlers nachgeschaut und getestet werden kann, wodurch dieser entstanden ist.

Beim Start des Backends werden die nötigen Daten über die API abgerufen und so aufbereitet, dass das Frontend dann mit diesen arbeiten und diese anzeigen kann.

Bei den Mitarbeitern ist es unter anderem nötig zuzuweisen, ob diese Manager sind, also Teamleiter oder Abteilungsleiter. Im Rahmen der Entwicklung wird also auch überprüft, ob die E-Mail des Angestellten mit, der des Entwicklers übereinstimmt. Außerhalb davon wird nur kontrolliert, ob die Identifikationsnummer des Mitarbeiters, die eines Team- oder Abteilungsleiters ist.

Als Mitarbeiter, die dem testenden Entwickler direkt und indirekt unterstellt sein sollten, werden alte Testdaten gewählt, deren Mitarbeiter keinen Vorgesetzten haben.

Diese werden dem Entwickler entweder direkt oder indirekt in der Hierarchie unterstellt.

```
for (Employee employee : testEmployees)
{
  if (employee.getDepartment() != null && !departments.contains(employee.getDepartment()))
  {
    departments.add(employee.getDepartment());
    testManagers.add(new Manager(employee.getFirstName(), employee.getLastName(),
            employee.getEmail(), employee.getDepartment(), employee.getExternalOid()));
    testManager.getEmployees().add(employee);
  }

  else if (employee.getDepartment() != null)
  {
    for (Manager manager : testManagers)
    {
      if (manager.getDepartment().equals(employee.getDepartment()))
      {
        manager.getEmployees().add(employee);
      }
    }
  }
}
```

Code 11.45 - Test-Mitarbeiter werden zugewiesen

Es wird für jeden Mitarbeiter überprüft, ob er ein Team hat und ob der Name dieses Teams in `departments` vorkommt. Anhand dessen wird entschieden, ob er dem Entwickler direkt oder indirekt unterstellt sein sollte.

Wenn beide Bedingungen erfüllt werden, wird sein Teamname zu `departments` hinzugefügt, er selbst zu einem `Manager` gemacht und der Entwickler zu seinem Vorgesetzten erklärt.

Wenn aber der Teamname von jenem Angestellten bereits in `departments` enthalten ist, werden alle bereits zugewiesenen Test-Manager durchlaufen. Es wird nach dem `Manager` gesucht, der zum selben Team gehört. Sobald dieser gefunden ist, wird jener Angestellte ihm direkt untergeordnet als Teammitglied.

Dadurch hat dann der Entwickler sowohl direkt als indirekt untergeordnete Teams, was ihm ermöglicht die Team-Übersicht zu testen, ohne dass er selbst eine hohe Position innerhalb der Hierarchie des Unternehmens belegen muss.

11.2.2 Team-Statistiken

Falls es sich beim Nutzer um einen Teamleiter handelt und nicht um einen Abteilungsleiter, werden hier nur zwei Dropdowns angezeigt.

Überprüft wird das, indem bei der Erstellung der Statistik-Ansicht folgendes ausgeführt wird.

```
this.employeeService.getTeamMembers(this.currentUser.email!)
    .subscribe( observerOrNext: res : globalThis.Error | Employee[]  => {
  let members : Employee[]  = res as Employee[];

  members.forEach( (employee : Employee ) : void  => {
    if (!this.teams.includes(employee.department!) && employee.department != null)
      this.teams.push(employee.department!);
  });

  if (this.teams.length > 1) this.showThirdDropdown = true;
});
```

Code 11.46 - Implementation derÜberprüfung

Nachdem über die API die Teammitglieder des Nutzers abgerufen wurden, werden sie mit Hilfe einer Schleife durchlaufen. Wenn jenes Teammitglied ein eigenes Team führt, wird dieses zu `teams` hinzugefügt. Wenn danach `teams` mehr als einen Wert enthält wird das dritte Dropdown angezeigt und stellt dem Nutzer den Inhalt der Variable zur Auswahl.

Beim Verändern der Auswahl, durch den Benutzer, wird die `onTeamChange`-Funktion aufgerufen:

```
onTeamChange(event: any): void {
  this.selectedTeam = event.value;

  if (this.selectedTeam === 'My Team') this.selectedTeam = this.currentUser.department!;

  this.onSelectionChange.emit(
      value: { location: this.selectedLocation,
        floor: this.selectedFloor,
        team: this.selectedTeam });
}
```

Code 11.47 - Implementation der onTeamChange-Funktion für Statistiken

Es wird überprüft ob alle untergeordneten Teams ausgesucht wurden oder nur ein einzelnes. Ist Ersteres der Fall muss der Platzhalterwert „all“ durch den richtigen Teamnamen des Benutzers ersetzt werden. Danach werden alle drei Filterkriterien an die Business-Logik der Statistik weitergegeben.

Jedes Mal, wenn sich eins dieser Kriterien ändert, wird die `updateStatistics`-Funktion ausgeführt, dessen erste Bedingung folgend implementiert ist:

```
async updateStatistics(event: any) : Promise<void>  {
  if (event.location === undefined || event.location.name === undefined) {
      if (event.team !== 'all') {
        this.tmpData = [this.tmpData[0], this.tmpData[this.findIndexOfTeamInTmp()]];
        if (event.team != this.team) this.tmpData = [this.tmpData[0]];
        this.indexForTmpData = 1
      }

      else {
        this.tmpData = []
        this.indexForTmpData = 0
      }
    }

    else if (event.location !== this.location) {
      this.tmpData = [this.tmpData[0], this.tmpData[this.findIndexOfTeamInTmp()]];

      if (event.team === 'all') this.tmpData = [this.tmpData[0]];

      if (event.location.name !== undefined) this.indexForTmpData = 1;
      else this.indexForTmpData = -1;
    }
```

Code 11.48 - Implementation zur Überprüfung für kein Kriterium oder Standort-Kriterium

Durch diese Verzweigungen wird überprüft welches Kriterium sich geändert hat und `tmpData` dementsprechend angepasst.

Die Variable `tmpData` speichert, welche neuen Kriterien sich durch den Nutzer ergeben haben. Dies ist nötig damit bei der Erstellung der Statistik, die Graphen mit den richtigen Kriterien assoziiert werden.

Das Team-Kriterium kann unabhängig von anderen Kriterien gesetzt werden, weswegen sein Platz in `tmpData` nicht immer derselbe sein muss. Der Standort und dessen Stockwerk, wenn letzterer ausgewählt ist, befinden sich immer an zweiter und dritter Stelle in der Variable. An erster Stelle befindet sich immer „Alle Standorte“, wodurch auch immer ein Graph diesen beschreibt.

Deswegen gibt es zur Aufsuchung des Teams in `tmpData` folgende Funktion:

```
findIndexOfTeamInTmp(): number {
  let result: number = 3;
  let index: number = 0;

  this.tmpData.forEach((data) : void  => {
    if (data.name.includes("Team")) result = index;
    index++;
  })

  return result;
}
```

Code 11.49 - Implementation der findIndexOfTeamInTmp-Funktion

Hier wird mit Hilfe einer Schleife `tmpData` nach dem Teamnamen gesucht. Dieser wird immer mit der Zeichenkette „Team“ davor gespeichert, also kann nach diesem gesucht werden.

Die Variable `indexForTmpData`, die in `updateStatistics` gesetzt wird, ist für die Funktion `convertToLineData` nötig. Durch diese wird bestimmt ob in `tmpData`, die nötigen Daten für einen neuen Graphen hinzugefügt werden müssen oder nicht.

```
if (this.indexForTmpData != -1) {
  this.tmpData.splice(this.indexForTmpData,  deleteCount: 0,  items: {
    name: this.getSeriesForLegend(),
    series
  });
}
```

Code 11.50 - Überpüfung im Falle eines neuen Graphes

Falls der Wert von `indexForTmpData` negativ ist, wird nichts neues eingefügt. Andernfalls wird in `tmpData` an der Stelle die `indexForTmpData` angibt, ein neues Objekt mit den Attributen

- `name`: Name des Graphen bzw. das neue Filter-Kriterium
- `series`: Werte die den Verlauf des Graphen beschreiben

gespeichert.

Um den Inhalt für `name` zu bestimmen, wird die Funktion `getSeriesForLegend` verwendet. Diese heißt so, weil der Wert, der hier `name` zugewiesen wird, in der Legende unter der Statistik erscheint.

```
getSeriesForLegend(): string {
  if (this.location?.name && this.location !== this.oldLocation) {
    return this.location.name;
  } else if (this.floor !== this.oldFloor && this.floor.id) {
    return 'Floor ' + this.floor.level;
  } else if (this.team !== this.oldTeam && this.team !== 'all') {
    return 'Team: ' + this.team
  } else {
    return 'All Locations';
  }
}
```

Code 11.51 - Implementation der getSeriesForLegend-Funktion

Hier wird überprüft welches Kriterium sich zuletzt geändert hat, indem die drei Filterkriterien, mit ihren alten Werten verglichen werden.

Mit der Funktion `getSeriesName` werden hingegen alle Kriterien hintereinander gereiht und unter der großen Prozentangabe ausgegeben:

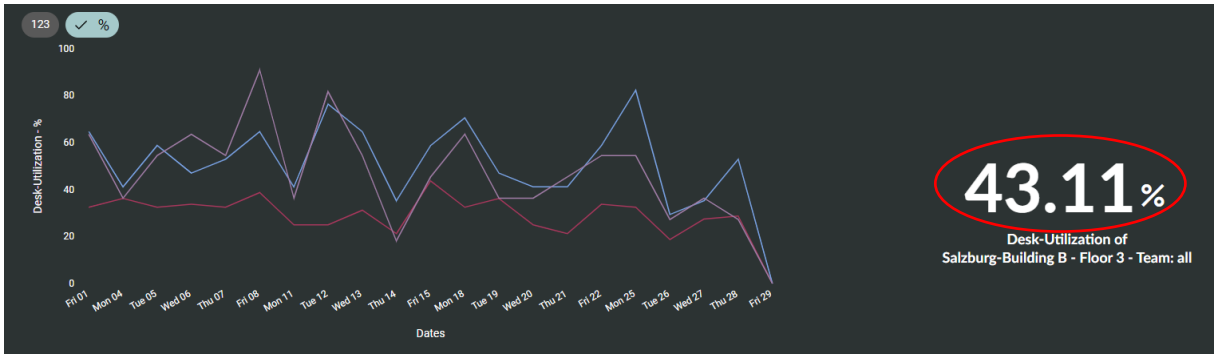

Abbildung 11.14 - Prozent-Auswertung der Filter-Kriterien

Hier wird überprüft welche Filterkriterien gesetzt sind und welche nicht. In Abhängigkeit davon wird eine Zeichenkette zusammengebaut, die die ausgewählten Kriterien beschreibt.

```
getSeriesName(): string {
  if (this.location?.name && !this.floor.id) {
    return this.location.name + ' - Team: ' + this.team;
  } else if (this.location?.name && this.floor.id) {
    return this.location.name + ' - Floor ' + this.floor.level + ' - Team: ' + this.team;
  } else if (this.team === 'all') {
    return 'All Locations'
  } else {
    return 'All Locations' + ' - Team: ' + this.team;
  }
}
```

Code 11.52 - Implementation der getSeriesName-Funktion

Die nötigen Daten für die Statistik werden in `getStatistics` besorgt:

```
async getStatistics() : Promise<void>  {
  let start : Date  = new Date(this.selectedDate.getFullYear(),
      this.selectedDate.getMonth(),  date: 1);
  let end : Date  = new Date(this.selectedDate.getFullYear(),
       monthIndex: this.selectedDate.getMonth() + 1,  date: 0);

  console.log("location: " + this.location?.id ?? undefined);
  console.log("floor: " + this.floor?.id ?? undefined);
  console.log("team: " + this.team ?? undefined);
  let response$ : Observable<Statistics | Error>  = this.statisticService
      .getDeskUtilization(
    this.datePipe.transform(start,  format: 'yyyy-MM-dd')!,
    this.datePipe.transform(end.setDate(end.getDate()),  format: 'yyyy-MM-dd')!,
     locationId: this.location?.id ?? undefined,
     floorId: this.floor?.id ?? undefined,
     team: this.team ?? undefined
  );
  let response : Statistics | Error  = await lastValueFrom(response$);

  if (response instanceof HttpErrorResponse) {
    this.error = new Error(response.status, response.error.title
        , response.error.message);
  } else {
    this.statistics = response as Statistics;
    this.convertToData();
  }
}
```

Code 11.53 - Implementation der getStatistics-Funktion

Diese Funktion wird beim Erstellen der Team-Statistiken und jedes Mal am Ende der Funktion `updateStatistics` ausgeführt. Es ruft die API über die Funktion `getDeskUtilization` von

`statistcService` ab und speichert die empfangenen Daten in der Variable `statistics`. Als Parameter werden der zuletzt genannten Funktion, die drei Filterkriterien übergeben. Anhand dieser wird dann entschieden welcher Endpoint der API kontaktiert werden sollte:

```
public getDeskUtilization(from: string, to: string, locationId?: number
    , floorId?: number, team?: string): Observable<Statistics | Error> {
  if (locationId === undefined && floorId === undefined) {
    return this.getDeskUtilizationByInterval(from, to, team!);
  } else if (floorId === undefined) {
    return this.getDeskUtilizationByIntervalAndLocation(from, to, locationId!, team!);
  }

  return this.http.get<Statistics>(
      url: `${environment.server}/statistics/deskutil/from/${from}/to/${to}
     /location/${locationId}/floor/${floorId}/team/${team}`)
    .pipe(map<any, Statistics>( project: res => res), catchError(this.errorHandler));
}
```

Code 11.54 - Implementation der getDeskUtilization-Funktion

Die Funktionen `getDeskUtilizationByInterval` und `getDeskUtilizationByIntervalAndLocation` sind ähnlich implementiert wie `getDeskUtilization`, nur dass sie keine Verzweigungen besitzen und direkt den API-Abruf ausführen.

Testen

Um die Team-Statistiken überprüfen zu können, sind wie bei der Team-Übersicht zusätzliche Änderungen im Backend nötig. Es muss gespeicherte Buchungen geben, die auf die untergeordneten Testangestellten des Entwicklers zurückzuführen sind. Ohne diese ergeben sonst alle Berechnungen des Backend null, wodurch die Statistik der Weboberfläche nichts zum Anzeigen hat. Das Generieren der Buchungen wurde folgendermaßen, innerhalb der Funktion `generateBookings`, implementiert:

```
for (Employee employee : employees) {
  DeskGroup deskGroup;

  if (employeeService.findManagerOfTeam(employee.getDepartment()) != null &&
          (employeeService.findManagerOfTeam(employee.getDepartment()).getEmail().contains("manager")
                  || employeeService.findManagerOfTeam(employee.getDepartment()).getEmail()
                  .equalsIgnoreCase( anotherString: "paul.oprea.ext@porscheinformatik.com"))) {
    do {
      deskGroup = deskGroups.get(deskGroupIndex % numDeskGroups);
      deskGroupIndex++;
    } while (deskGroup.getBookings().size() == deskGroup.getCapacity());

    for (int j = 0; j < 49; j++) {
      randomDay = random.nextInt( bound: (28 - 1) + 1) + 1;
      LocalDate date = LocalDate.of(LocalDate.now().getYear(), LocalDate.now().getMonth(), randomDay);
      var booking = new Booking(date);
      employee.addBooking(booking);
      deskGroup.addBooking(booking);
      System.out.println("Booking für Mitglied von Team: " + employee.getDepartment()
              + " - Location: " + booking.getDeskGroup().getFloor().getLocation().getName());

      bookings.add(booking);

      employee.setDefaultFloor(booking.getDeskGroup().getFloor());
    }
  }

  deskGroupIndex++;
}
```

Code 11.55 - Generation und Zuweisung von Test-Buchungen

Es werden alle Angestellten durch die Nutzung einer Schleife traversiert. Als erstes wird überprüft, ob jener Testmitarbeiter den Entwickler als Vorgesetzten hat.

Ist dies der Fall wird eine Deskgroup aus `deskGroups` ausgewählt, die noch Platz für weitere Buchungen hat. Falls dies nicht der Fall ist, wird deskGroupIndex inkrementiert und der Vorgang wiederholt. Dabei muss deskGroupIndex nie zurückgesetzt werden, weil immer ein Deskgroup mit dem Index `deskGroupIndex % numDeskGroups` ausgewählt wird. Das liegt daran, dass das Ergebnis dieser Rechenoperation nie die Anzahl der Deskgroups, also `numDeskGroups`, übersteigen wird.

Danach wird beliebig oft – in diesem Fall 50-mal – eine Buchung, an einem zufälligen Tag im aktuellen Monat erstellt und sowohl dem Angestellten als auch der Deskgroup zugewiesen. Damit die Buchung auch in der Datenbank gespeichert wird, wird sie zu `bookings` hinzugefügt. Diese Variable enthält alle Buchungen, die innerhalb von `generateBookings` erstellt wurden.

Die Funktion `findManagerOfTeam`, deren Rückgabewert bei der Überprüfung des Angestellten verwendet wird, ist folgend implementiert:

```
public Manager findManagerOfTeam(String team) {
  for (Manager manager : this.employeeRepo.findAllManagers()) {
    if (manager.getDepartment().equalsIgnoreCase(team)
            || manager.getEmail().equalsIgnoreCase(testingDeveloperMail)) {
      return manager;
    }
  }

  return null;
}
```

Code 11.56 - Implementation der findManagerOfTeam-Funktion

Es werden alle Team- und Abteilungsleiter, die es gibt, durchlaufen. Stimmt der Teamname des Vorgesetzten mit dem Parameter `team` überein, wird jener Manager als Ergebnis zurückgegeben.

11.2.3 Fixed-Shared-Desk-Switch

Die Methode für die Anlegung der neuen Gruppe sieht wie folgt aus:

```
this.deskGroupService.createDeskGroup(deskGroup).subscribe( observerOrNext: res : globalThis.Error | DeskGroup => {
    console.log(res);
    if (res instanceof HttpErrorResponse) {
        console.log(res);
    }else{
        const deskfreeDate : Date = new Date(this.endDate!)
        deskfreeDate!.setDate(deskfreeDate!.getDate() + 1);
        this.user.deskSetAsSharedDate = this.datePipe.transform(deskfreeDate, format: 'dd-MM-yyyy')!;
        this.user.deskSetAsShared = true;
        this.employeeService.updateEmployeeInfo(this.user).subscribe( observerOrNext: res => {
            if (res instanceof HttpErrorResponse) {
                console.log(res);
            }
        });
        this.setEndDateMessage();
    }
});
```

Code 11.57 – neue Schreibtischgruppe anlegen

Nachdem die Desk-Gruppe im Frontend erstellt und erfolgreich ans Backend verschickt wurde, wird beim Benutzer abgespeichert, dass er seinen Schreibtisch freigegeben hat und wann sein Schreibtisch wieder frei wird. Hierbei muss das Datum, in das richtige Format gebracht werden, damit es bei der Übertragung zu keinen Fehlern kommt.

Jedes Mal, wenn die Desk-Groups eines Stockwerks abgefragt werden, iteriert die Applikation über die Gruppen und überprüft, ob es sich hierbei um von einem Mitarbeiter erstellte Gruppe handelt und ob das heutige Datum zwischen dem Start- und Enddatum der Gruppe liegt.

```
if (deskGroup.getOwnerId()!=-1L) {
  if (deskGroup.getStartDate() != null && currentDate.isBefore(deskGroup.getStartDate())) {
    iterator.remove();
  } else if (deskGroup.getEndDate() != null && currentDate.isAfter(deskGroup.getEndDate())) {
    deleteDeskGroup(deskGroup);
    iterator.remove();
  }
}
```

Code 11.58 – Schreibtischgruppen filtern

Wenn das aktuelle Datum vor dem Startdatum liegt, wird die Gruppe aus der Liste der Gruppen, die angezeigt werden, entfernt, wenn es danach liegt, wird die Gruppe aus dem System gelöscht.

11.3 Graph API

Ein großer Bestandteil der Share:D Anwendung ist das Laden aller Daten, damit andere Komponenten mit diesen arbeiten können. Dieses Laden passiert im Backend, wobei beim Starten des Backends zuerst einmal eine Verbindung zur Microsoft Graph API hergestellt werden muss. Im Zuge dieser Verbindung muss sich ein Mitarbeiter der Porsche Informatik einmalig anmelden, wobei dieser Mitarbeiter möglichst derjenige sein sollte, der in späterer Folge für die Instandhaltung von Share:D verantwortlich ist, da alle Abfragen über diesen Mitarbeiter laufen.

Sollte die Anmeldung erfolgreich abgelaufen sein, werden die geladenen Daten in unserer Datenbank abgespeichert. Danach werden noch einige Daten generiert, die nicht von der Graph API abgerufen werden können, wie die Standorte der Porsche Informatik oder Buchungen für Testzwecke, da diese Daten sich nicht in der Graph API befinden und statisch sind, weswegen sie bei jedem Start der Applikation neu generiert werden. Anschließend werden noch die Mitarbeiter dynamisch über die Graph API abgerufen, wonach die Datenbank fertig initialisiert ist und Share:D benutzt werden kann.

```
@Override
public void run(String... args) {
  graphService.buildClient(); // Verbindung zur Graph API herstellen
  generateLocations(); // Standorte generieren
  databaseService.refreshEmployees(); // Mitarbeiter aus Graph API holen
  generateBookings(); // nur für Testzwecke
  generateTickets(); // nur für Testzwecke
  setDefaultPath(); // siehe 11.1 Raumplan
  LOGGER.info("Database initialized");
}
```

Code 11.59 - Initialisierung der Datenbank

11.3.1 Outlook Verbindung

Bevor nun eine Verbindung zu Outlook bzw. der Graph API von Microsoft hergestellt werden kann, benötigt die Anwendung, wie in 8.4.1 Outlook Schnittstelle beschrieben, eine Microsoft Entra ID Applikation. Diese App kann auf dem Azure Portal von Microsoft unter der URL „https://portal.azure.com/“ bei dem gleichnamigen „Microsoft Entra ID“ Azure-Dienst erstellt werden. So eine Applikation wurde bereits von der Porsche Informatik erstellt, allerdings mussten noch einige Änderungen bei den Menüpunkten „Authentifizierung“ und „API-Berechtigungen“ vorgenommen werden.

Authentifizierung

Im Menüpunkt „Authentifizierung“ müssen zwei Umleitungs-URIs hinzugefügt werden, eine Web-Umleitungs-URI und eine SPA-Umleitungs-URI. An diese URIs wird nach der erfolgreichen Benutzerauthentifizierung der Token, welcher für weitere Authentifizierungen innerhalb der Applikation benötigt wird, zurückgegeben.

Weiters muss weiter unten auf der Seite der öffentliche „Clientflow“ zugelassen werden. Dieser wird benötigt, damit später im Code der Anmeldebildschirm in einem Browser aufgerufen werden kann.

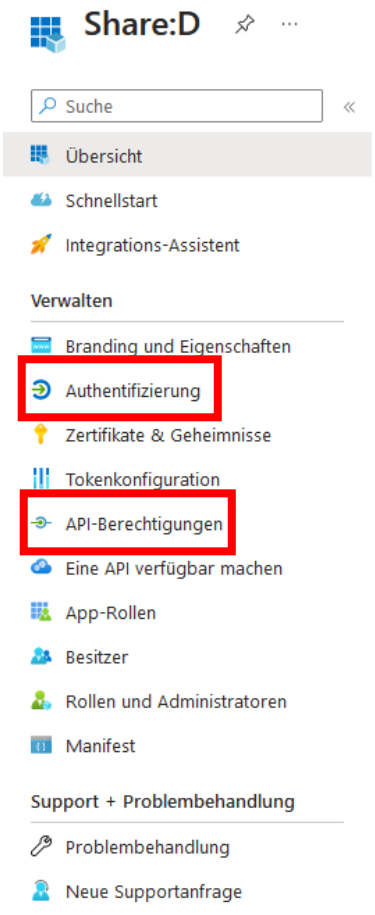

Abbildung 11.15 - Microsoft Entra ID Applikation Menüpunkte (azure.com)

API-Berechtigungen

Die Applikation würde jetzt schon ohne weitere Änderungen funktionieren, jedoch hat sie noch keinen Zugriff auf die Daten, die benötigt werden. Deswegen werden bei diesem Menüpunkt drei Microsoft Graph Berechtigungen hinzugefügt:

- `User.Read.All` – Hiermit können die vollständigen Profile von Nutzern abgerufen werden. Diese Berechtigung muss von einem Administrator der Porsche Informatik erteilt werden.
- `Calendars.ReadWrite` – Hiermit kann der eigene Kalender eingelesen und auch verändert werden. Dies wird für die Eventverwaltung benötigt.
- `Calendars.Read.Shared` – Hiermit können die Kalender von anderen Mitarbeitern eingelesen werden. Dies wird für das Einlesen der Anwesenheiten der Mitarbeiter benötigt.

API/Berechtigungsname	Typ	Beschreibung	Administratoreinwill...
Microsoft Graph (3)			
Calendars.Read.Shared	Delegiert	Benutzer und freigegebene Kalender lesen	Nein
Calendars.ReadWrite	Delegiert	Verfügt über Vollzugriff auf Benutzerkalender.	Nein
User.Read.All	Delegiert	Vollständige Profile aller Benutzer lesen	Ja

Abbildung 11.16 - Microsoft Graph Berechtigungen (azure.com)

Nach den Änderungen an der Microsoft Entra ID Applikation kann nun die Graph API im Code verwendet werden.

```
@Override
public void buildClient() {
  InteractiveBrowserCredential credential =
    new InteractiveBrowserCredentialBuilder()
      .clientId(clientId)
      .tenantId(tenantId.split( regex: "/")[3])
      .redirectUrl(redirectUrl)
      .build();

  TokenCredentialAuthProvider authProvider =
    new TokenCredentialAuthProvider(
      List.of( e1: "https://graph.microsoft.com/.default"),
      credential
  );

  GraphServiceImpl.graphClient =
    GraphServiceClient
      .builder()
      .authenticationProvider(authProvider)
      .buildClient();
}
```

Code 11.60 – Verbindung zur Graph API

Im obigen Codeausschnitt wird ein neuer Graph API Client erstellt, über welchen alle zukünftigen API-Aufrufe laufen. Dazu wird ein `AuthProvider` benötigt, welcher ein `Credential` benötigt. In diesem Fall wird das `InteractiveBrowserCredential` benutzt, welches den normalen Microsoft Anmeldebildschirm im Standardbrowser öffnet, wo sich der Nutzer wie gewohnt anmelden kann. Ist die Anmeldung erfolgreich, wird der Token für den `TokenCredentialAuthProvider` zur Verfügung gestellt, welcher wiederum dem Graph API Client übergeben wird, damit sich dieser bei der Graph API bei weiteren API-Aufrufe als der angemeldete Nutzer authentifizieren kann, ohne dass sich dieser nochmal anmelden muss.

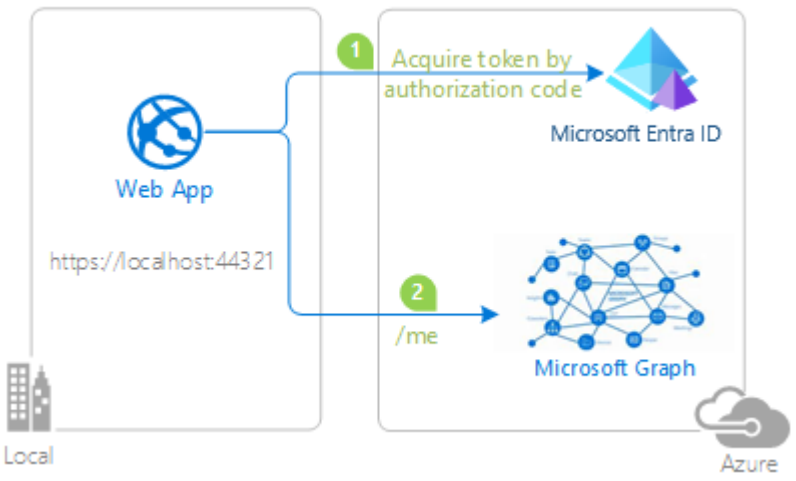

Abbildung 11.17 - Authentifizierungsablauf (microsoft.com)

11.3.2 Mitarbeiter und deren Anwesenheit einlesen

Für das erstmalige Laden und Aktualisieren von Mitarbeiterdaten in die Datenbank gibt es die `refreshEmployees` Methode. Am Anfang dieser Methode werden folgende Informationen eingelesen:

- Eine Liste aller Mitarbeiter der Porsche Informatik von der Graph API.
- Eine Liste aller vorhandenen Standorte der Porsche Informatik aus unserer Datenbank.
- Eine Liste aller vorhandenen Mitarbeiter der Porsche Informatik aus unserer Datenbank, falls diese schon einmal eingelesen wurden.

Beim Holen der Daten von der Graph API muss mit dem Graph Client eine neue Abfrage erstellt werden. Dabei bilden alle Methoden Aufrufe vor dem `buildRequest` Aufruf den abzurufend en Endpunkt. Also entspricht der Aufruf im Code 11.61 dem „https://graph.microsoft.com/v1.0/users“ Endpunkt. Zusätzlich werden noch drei weitere Methoden benötigt:

- `count`, um die Anzahl der Objekte, die das Programm empfängt, zu erhalten.
- `filter`, um die Objekte auf Mitarbeiter der Porsche Informatik einzuschränken.
- `select`, um die Anfrage um bestimmte Daten zu ergänzen, da diese sonst fehlen.

```
@Override
public UserCollectionPage getAllUsers(String filter) {
  for (int i = 0; i < 10; i++) {
    try {
      return graphClient
        .users() UserCollectionRequestBuilder
        .buildRequest(
          new HeaderOption( name: "ConsistencyLevel",  value: "eventual")
        ) UserCollectionRequest
        .count( value: true)
        .filter(filter)
        .select( value: "department,givenName,surname,mail,id,displayName,officeLocation,...")
        .get();
    } catch (Exception e) {
      LOGGER.error("Error in getAllUsers(): " + e.getMessage());
      LOGGER.info("Trying again in 5 seconds... (Attempt " + (i + 1) + "/10)");

      try {
        Thread.sleep( millis: 5000);
      } catch (InterruptedException interruptedException) {
        LOGGER.error(
          "Waiting got interrupted because of: " + interruptedException.getMessage()
          );
      }
    }
  }
  return null;
}
```

Code 11.61 – Alle Mitarbeiter abfragen

Sollte bei der Abfrage etwas passieren, wie ein etwaiger Internetausfall, wird die Abfrage nach fünf Sekunden noch einmal gesendet, bis die Abfrage zehn Mal probiert wurde. Sollte es nach dem zehnten Mal noch immer nicht funktioniert haben, wird die Abfrage abgebrochen.

Wenn die Graph API Abfrage erfolgreich war, füllt das Programm die anderen zwei Listen mit Daten aus der Datenbank. Sollte die Methode für das erstmalige Laden der Daten benutzt werden, ist die Liste der vorhandenen Mitarbeiter natürlich leer. Diese Liste wird nur zum Vergleichen benötigt, damit nicht derselbe Mitarbeiter öfters abgespeichert wird. Sollte die Liste also leer sein, werden einfach alle Mitarbeiter abgespeichert. Sobald dann diese Listen initialisiert sind, beginnt der erste von drei großen Abschnitten:

Mitarbeiter analysieren

Nun müssen die Mitarbeiter, die von der Graph API gesendet wurden, genauer analysiert und an unser Datenmodell angepasst werden. Zuerst aber empfängt das Programm nicht alle Mitarbeiter auf einmal von der API, sondern immer 100 Mitarbeiter, welche eine sogenannte „Page“ ausmachen. Im Programm werden also zuerst die 100 Mitarbeiter der empfangenen Page verarbeitet, bevor eine weitere Page abgefragt wird.

Ein zu verarbeitender Mitarbeitersatz enthält mindestens Vor- und Nachname, sowie eine E-Mail-Adresse. Weiters wird überprüft, ob der Mitarbeiter einen Standardstandort hat, und ob dieser ein Vorgesetzter von mindestens einem Mitarbeiter ist. Dabei wird in der Graph API nachgeschaut, ob der Mitarbeiter Untergeordnete besitzt:

```
return graphClient.users(userId).directReports().buildRequest().count( value: true).get();
```

Code 11.62 - Untergeordnete abfragen

Wenn das der Fall ist, wird der Vorgesetzte in einer eigenen, von normalen Mitarbeitern getrennten Liste abgespeichert, damit später die Vorgesetzten den Mitarbeitern zugeordnet werden können.

Falls es keinen einzigen Vorgesetzten in der Liste geben sollte, kann der nächste Abschnitt übersprungen werden.

Vorgesetzte analysieren

Anders als bei Mitarbeitern empfängt das Programm hier nicht die Vorgesetzten mittels der Graph API, sondern ihre Untergeordneten, in der Graph API auch „Direct Reports“ genannt. Die Abfrage für die Untergeordneten funktioniert ähnlich wie bei der letzten Abfrage, und kann in Code 11.62 eingesehen werden.

Die abgerufenen Untergeordneten werden nun einem Mitarbeiter aus unserem vorherigen Abschnitt zugeordnet. Sollte kein Mitarbeiter passen, werden auch noch die anderen Vorgesetzten geprüft, da ein Untergeordneter auch Vorgesetzter sein kann. Die gefundenen Mitarbeiter bzw. Vorgesetzten aus unserer Datenbank werden dem Vorgesetzten zugewiesen.

Sollte dies nicht der erste Aufruf der `refreshEmployees` Methode sein, und kein Unterschied zwischen den neuen und alten Mitarbeitern sein, kann der letzte Abschnitt übersprungen werden.

Anwesenheit der Mitarbeiter und Vorgesetzten analysieren

Ein Teil der Mitarbeiter-Info für Vorgesetzte ist das Einsehen der Anwesenheit der eigenen Untergeordneten. Diese Daten werden an diesem Punkt im Programm mithilfe der Graph API eingelesen, wofür drei Teile benötigt werden:

- Von wem?
- Von wann?
- Bis wann?

Für das „wem“ existiert schon die E-Mail-Adresse des aktuellen Mitarbeiters, fehlt also noch der Zeitraum. Die Mitarbeiter-Info verlangt, dass die Anwesenheit von der aktuellen Woche eingesehen werden kann. Also wird der Zeitraum von Montag in der Früh bis Sonntag am Abend benutzt. Die Abfrage selbst benutzt diesmal die Methodenkette: `me().calendar().getSchedule(parameters)`. Obwohl hier auch wieder mehrere Anwesenheiten empfangen werden, werden nur die der allerersten Woche benötigt.

Der letzte Schritt ist, die Anwesenheit auf unser Datenmodell anzupassen. Die Graph API sendet Werte wie `free`, `oof`, `workingElsewhere`, `tentative` und `busy`, allerdings unterscheidet das Programm nur zwischen `Frei`, `außer Haus` und `sonstiges`. Da „free“ nur englisch für „Frei“ und „Out of Facility“ so viel wie „außer Haus“ bedeutet, werden diese zueinander zugewiesen, wobei die restlichen Werte unter „Sonstiges“ eingeordnet werden.

```
public enum WeekStatusType {
  InHouse, HomeOffice, Absent
}

/* Outlook Typen
 * Frei - HomeOffice      (free)
 * Außer Haus - Abwesend  (oof)
 * Sonstiges - InHouse    (workingElsewhere, tentative, busy)
 */
```

Code 11.63 - Zuweisung der Anwesenheitswerte

Nachdem diese drei Abschnitte durchlaufen sind, ist das Initialisieren von den Mitarbeitern und Vorgesetzten mit ihren Anwesenheiten fertig. Allerdings kann dies noch optimiert werden. Im echten Programm wird nicht jeder Mitarbeiter nacheinander analysiert und verarbeitet, sondern es werden 100 Mitarbeiter, also eine Page, gleichzeitig verarbeitet. Diese Technik, mit der mehrere Programmteile gleichzeitig ablaufen können, wird „Multithreading“ genannt, wobei ein „Thread“ ein kleinerer, vom Hauptprogramm entstehender Prozess ist.

Das Multithreading wurde hier realisiert, indem pro Abschnitt jede Page einen eigenen Thread zum Analysieren von Mitarbeitern, Vorgesetzten und deren Anwesenheiten bekommt. Sobald alle Threads gestartet sind und arbeiten, wartet das Programm darauf, dass alle fertig sind, damit der nächste Abschnitt auch alle Daten nutzen kann. Beispielsweise könnten im Vorgesetzten Abschnitt nicht alle Untergeordneten einem Mitarbeiter zugeordnet werden, wenn nicht zuvor alle Mitarbeiter-Threads mit ihrer Arbeit fertig sind.

11.3.3 Kalender Events lesen und schreiben

Neben den Mitarbeitern und Vorgesetzen werden auch noch Events gespeichert, damit diese angezeigt, erstellt und gelöscht werden können. Um diese für die weitere Verarbeitung speichern zu können, wird ein Modell in unserer Datenbank, sowie API-Schnittstellen, welche für das Bearbeiten und Erstellen im Frontend benötigt werden, erstellt.

Das Modell

Ein Event besteht bei uns aus mehreren Eigenschaften:

- Einer ID zur einzigartigen Identifikation in der internen Datenbank und
- einer externen ID, welche von der Graph API festgelegt wird, sowie
- ein kurzer Titel und
- einer Beschreibung, welche länger sein darf, und
- einem Startzeitpunkt, sowie
- einem Endzeitpunkt.

Weiters speichert ein Event verschiedene Referenzen zu anderen Tabellen unserer Datenbank, wie:

- alle Mitarbeiter, die am Event teilnehmen, und
- der Mitarbeiter, der das Event veranstaltet, als auch
- der Standort und
- der Wiederholungstyp des Events.

```java
public class Event {
  @Id
  @GeneratedValue
  private Long id;

  private String title;

  @Column(columnDefinition = "TEXT")
  private String description;

  private LocalDateTime startDateTime;

  private LocalDateTime endDateTime;

  @ManyToMany(fetch = FetchType.LAZY)
  private Set<Employee> attendees;

  @ManyToOne(fetch = FetchType.EAGER)
  private Employee organizer;

  @ManyToOne(fetch = FetchType.EAGER)
  private Location location;

  @Enumerated(EnumType.STRING)
  private RepeatingType repeatingType;

  private String externalId;
}
```

Code 11.64 - Modell für Events

Die letzte Referenz ist allerdings anders als die anderen, da hier nur der Wert einer Enumeration gespeichert wird. Mögliche Werte für diese Referenz können dem Code 11.65 entnommen werden.

```java
public enum RepeatingType {
  NONE, WEEKLY, DAILY, MONTHLY, YEARLY
}
```

Code 11.65 - Enumeration für Wiederholungstyp

Die API-Schnittstellen

Die notwendigen API-Schnittstellen, um die Events aus unserer Datenbank anzeigen, neue erstellen und andere löschen zu können, werden in einer „Controller“-Klasse angelegt. Damit eine normale Java Klasse zu einer Controller-Klasse werden kann, benötigt die Klasse die `RestController` Annotation. Weiters legen wir noch mit der `RequestMapping` Annotation den Pfad für alle API-Schnittstellen dieser Klasse auf `/api/events` und dass alle API-Schnittstellen dieser Klasse die fertigen Objekte im „JSON“ Format senden sollen, damit die Daten in einem einheitlichen Format über das Netzwerk gesendet und empfangen werden können, fest. Zuletzt benötigen wir noch einen Weg, auf unsere Daten von der Controller-Klasse aus zuzugreifen. Dies ist mit einer „Service“-Klasse möglich, welche über die Spring Framework Annotation `AutoWired` in unsere Controller-Klasse injiziert werden. Anfangs wird nur eine Schnittstelle für die Service-Klasse entwickelt, welche den Namen, den Typ des Rückgabeobjektes sowie die Übergabeparameter für alle Methoden festlegt, wie in dem Code 11.66 zu sehen ist.

```
public interface EventService {
  List<Event> findAll();
  List<Event> findRelevant();
  List<Event> postAndSaveEvents(List<Event> events);
  void cancelEvent(Long eventID);
}
```

Code 11.66 - Schnittstelle für die Event Service-Klasse

Die Controller-Klasse sieht jetzt, ohne weitere Methoden, wie in dem Code 11.67 aus. Als nächstes müssen die API-Schnittstellen festgelegt, und zuletzt noch die Service-Klassen implementiert werden.

```
@RestController
@RequestMapping(
  value = "/api/events",
  produces = MediaType.APPLICATION_JSON_VALUE
)
public class EventController {
  private final EventService eventService;
  private final DatabaseService databaseService;
  private final ModelMapper mapper;

  @Autowired
  public EventController(
          EventService eventService,
          DatabaseService databaseService,
          ModelMapper mapper) {
    this.eventService = eventService;
    this.databaseService = databaseService;
    this.mapper = mapper;
  }
}
```

Code 11.67 - Event Controller-Klasse

Alle Methoden für die API-Schnittstellen haben zumindest zwei Annotationen:

- Eine `Mapping` Annotation, ähnlich wie die `RequestMapping` Annotation, welche die gültige Anfragemethode und den Pfad zu dieser API-Schnittstelle festlegt. In dieser Controller-Klasse werden `GetMapping`, `PostMapping` und `DeleteMapping` benutzt.
- Eine `Operation` Annotation, welche eine Beschreibung für diese Methode definiert.

Weiters geben alle API-Schnittstellen Methoden ein `ResponseEntity` Objekt zurück. In diesem Objekt verpackt sind der HTTP-Status-Code und, falls vorhanden, andere Objekte, wie Events.

Die Methoden für die API-Schnittstellen zur Abfrage von Events können in dem Code 11.68 eingesehen werden. Diese haben die zuvor erwähnten Annotationen und geben ein `ResponseEntity` Objekt mit dem HTTP-Status-Code 200 OK sowie den abgefragten Event Objekten in Form einer Liste mit `Data Transfer Objekten` (Siehe Seite 89) zurück. Bevor die Events abgefragt werden, werden die Referenzen zu diesen in unserer Datenbank erneuert, damit etwaige Änderungen berücksichtigt werden. Die Methode muss am Anfang feststellen, ob derzeit ein Nutzer angemeldet ist, da Events nur von einem bestimmten Nutzer abgefragt werden können. Sollte keiner angemeldet sein wird wie in dem Code 11.69 der Nutzer zum Anmelden aufgefordert. Danach benutzt die Methode dasselbe Verfahren wie das Einlesen der Mitarbeiter: Es werden alle Events des Nutzers von der Graph API abgefragt, welche in Form von Pages empfangen und mithilfe von Multithreading in die Datenbank eingelesen werden.

```java
@GetMapping(value = "/")
@Operation(summary = "Get all Events")
public ResponseEntity<List<EventDto>> getAll() {
  databaseService.refreshEvents();
  return ResponseEntity.ok(
    mapToDto(
      eventService.findAll()
    )
  );
}

@GetMapping(value = "/relevant")
@Operation(summary = "Get all relevant Events")
public ResponseEntity<List<EventDto>> getRelevant() {
  databaseService.refreshEvents();
  return ResponseEntity.ok(
    mapToDto(
      eventService.findRelevant()
    )
  );
}
```

Code 11.68 - API-Schnittstellen zum Abrufen von Events

```java
if (userService.getCurrentUser().isEmpty()) {
  graphService.buildClient();
  graphService.getMyself();

  if (userService.getCurrentUser().isEmpty()) {
    throw new ResourceNotFoundException("No one is logged in");
  }
}
```

Code 11.69 - Prüfen, ob ein Nutzer angemeldet ist

Neben den Methoden zur Abfrage für Events benötigen wir auch API-Schnittstellen für das Erstellen und Löschen der Events. Diese können in dem Code 11.70 eingesehen werden. Auch diese haben wieder die zwei Annotationen und die Methode zum Erstellen gibt auch wieder die erstellten Events an den Absender der Anfrage zurück. Die Methode zum Löschen ist allerdings anders: Sie benutzt eine `PathVariable`, da für das Löschen nur die ID des Events benötigt wird. Weiters gibt diese nur den HTTP-Status-Code 204 NO CONTENT, also ohne Inhalt oder Objekten, zurück.

```
@PostMapping(value = "/")
@Operation(summary = "Post multiple Events to Microsoft Outlook")
public ResponseEntity<List<EventDto>> postEvents(
  @RequestBody List<EventDto> input
) {
  return ResponseEntity.ok(
    mapToDto(
      eventService.postAndSaveEvents(
        mapToEvent(input)
      )
    )
  );
}

@DeleteMapping(value = "{eventID}")
@Operation(summary = "Delete existing Booking")
public ResponseEntity<Void> deleteBooking(
  @Schema(description = "ID of Event", example = "1") @PathVariable Long eventID
) {
  eventService.cancelEvent(eventID);
  return ResponseEntity.noContent().build();
}
```

Code 11.70 - API-Schnittstellen zum Erstellen und Löschen von Events

In der Controller-Klasse werden nun noch zwei Methoden benötigt, welche zur Zuweisung eines Events zu einem `Data Transfer Objekt`, kurz DTO, oder andersherum benutzt werden. Ein DTO ist das Objekt, das andere Programme, die auf die API-Schnittstellen zugreifen, benutzen, damit diese keine komplexen Referenzen zu anderen Objekten und keine Eigenschaften, die nur für interne Zwecke benötigt werden, verwalten müssen. Im Fall des Eventobjektes besitzt das DTO keine externe ID und nur eine einfache Liste für Teilnehmer statt einer Referenz zu einer anderen Tabelle.

```
private List<EventDto> mapToDto(List<Event> events) {
  return mapper.map(events, (new TypeToken<List<EventDto>>() {
  }).getType());
}

private List<Event> mapToEvent(List<EventDto> events) {
  return mapper.map(events, (new TypeToken<List<Event>>() {
  }).getType());
}
```

Code 11.71 - Methoden zum Wechseln zwischen Eventobjekten und Event DTOs

Damit die Methodenaufrufe der Event Service-Klasse innerhalb der API-Schnittstellen Methoden auch etwas bewirken, müssen diese implementiert werden. Dazu wird eine neue Klasse erstellt, die die Schnittstelle von dem Code 11.66 implementiert. Weiters wird die Klasse mit zwei Annotationen bestückt:

- Die `Service` Annotation, um die Klasse als Service-Klasse zu markieren, und
- die `Transactional` Annotation, damit alle Methoden der Klasse eine neue Transaktion zur Datenbank starten und diese am Ende der Methode zur Datenbank senden und beenden.

```
@Service
@Transactional
public class EventServiceImpl implements EventService {
  private final EventRepository eventRepo;
  private final EmployeeRepository employeeRepo;

  private final LocationService locationService;
  private final CustomUserService userService;
  private final GraphService graphService;

  @Autowired
  public EventServiceImpl(
          EventRepository eventRepo,
          EmployeeRepository employeeRepo,
          LocationService locationService,
          CustomUserService userService,
          GraphService graphService) {
    this.eventRepo = eventRepo;
    this.employeeRepo = employeeRepo;
    this.locationService = locationService;
    this.userService = userService;
    this.graphService = graphService;
  }
}
```

Code 11.72 - Event Service-Klasse

Nicht zuletzt benötigt die Service-Klasse auch noch andere Service-Klassen und sogenannte `Repositorien`, welche den direkten Zugriff auf die Tabellen der Datenbank ermöglichen. Die Referenzen zu diesen werden, wie schon bei der Controller-Klasse, über die `Auto-Wired` Annotation in den Konstruktor injiziert, was zu einem Aufbau wie in dem Code 11.72 führt.

Das zuvor erwähnte Repository ist eine Schnittstelle, die den Zugriff auf die Datenbank erlaubt. Dies funktioniert zum einen über die `Repository` Annotation, welche die Schnittstelle als Repository-Schnittstelle markiert, und zum anderen über die bereitgestellte Schnittstelle `JpaRepository`, welches das Modell, auf welches dieses Repository zugreifen soll, sowie den Typ der ID-Eigenschaft, benötigt. In diesem Fall will das Event Repository auf die Events zugreifen, wobei der Typ der ID-Eigenschaft ein `Long` ist. Die bereitgestellte Schnittstelle stellt auch einfache Methoden zur Verfügung, wie `findAll`, `save` oder `delete`, damit diese nicht implementiert werden müssen. Das Event-Repository muss aber noch eine eigene Methode implementieren, welche alle relevanten Events finden soll. Ein

Event gilt als relevant, wenn es sich entweder wiederholt oder nach der jetzigen Zeit stattfindet. Diese Bedingungen können in der `Query` Annotation über der Methode niedergeschrieben werden, damit diese implementiert wird.

```
@Repository
public interface EventRepository extends JpaRepository<Event, Long> {
  @Query("select e from Event e where e.repeatingType != 'NONE' or e.startDateTime > current_timestamp()")
  List<Event> findRelevant();
}
```

Code 11.73 - Event Repository-Schnittstelle

Nun können die Methoden der Event Service-Schnittstelle implementiert werden. Die zwei Methoden zum Abrufen der Events sind nur ein Aufruf der gleichnamigen Repository-Methode, wie in dem Code 11.74 zu sehen ist.

```
@Override
public List<Event> findAll() {
  return eventRepo.findAll();
}

@Override
public List<Event> findRelevant() {
  return eventRepo.findRelevant();
}
```

Code 11.74 - Methoden zum Abrufen von Events

Die Methode zum Speichern von neuen Events benutzt zuerst denselben Code wie in dem Code 11.69, um zu prüfen, ob ein Nutzer angemeldet ist. Zunächst wird für jedes neue Event die Referenz auf den Event-Veranstalter gesucht, indem alle Mitarbeiter auf die externe ID des angemeldeten Nutzers geprüft werden. Als nächstes wird auch die Referenz auf den Veranstaltungsort über den Namen gesucht, wobei ein neuer Ort in unserer Datenbank gespeichert wird, sollte keiner gefunden werden. Zuletzt wird noch die externe ID für das Event benötigt, welche durch das Senden des Events an die Graph API erhalten werden kann. Um das Event allerdings an die Graph API senden zu können, muss das Event Objekt von unserem Modell an das Modell der Graph API angepasst werden, da dieses eine andere Struktur und mehr Daten als unser Modell nutzt. Zum Schluss werden noch alle Events, die nun angepasst wurden, in unserer Datenbank mithilfe der `saveallAndFlush` Methode, die vom `JpaRepository` zur Verfügung gestellt wird, gespeichert.

Auch die Methode zum Löschen eines bestehenden Events benutzt den Code aus dem Code 11.69, um einen angemeldeten Nutzer zu garantieren. Wie oben erwähnt wird zum Löschen eines Events nur eine ID benötigt, da das `JpaRepository` die Methode `findById` zur Verfügung stellt. Nun muss das Event nur noch an das Modell der Graph API angepasst und von dort gelöscht werden, sowie aus unserer Datenbank mithilfe der bereitgestellten `delete` Methode.

11.4 Events

Ein weiterer Bestandteil unserer Diplomarbeit ist das Anzeigen und Hinzufügen von Events.

Die Events des Benutzers, der eingeloggt ist, werden von Microsoft Outlook zur Verfügung gestellt. Diese Events werden dann von dem Backend mittels einer weiteren API an das Frontend, also der Webseite, übermittelt. Danach werden die Events aufbereitet und dem Benutzer angezeigt. Durch das Drücken des "Add Event"-Buttons wird der Benutzer auf eine neue Seite weitergeleitet, die ihm die Eingabe der Eventdaten ermöglicht. Hier kann der Benutzer nun die Informationen des Events eingeben und diese absenden.

11.4.1 API-Aufruf

Die Methode in diesem Code Ausschnitt wird verwendet, um Daten von der API abzurufen.

```
getEvents(): Observable<Event[] | Error> {
    return this.http.get<Event[]>( url: `${environment.server}/events/relevant`)
        .pipe(map<any, Event>( project: res => res), catchError(this.errorHandler));
}
```

Code 11.75 - API Abfrage

Der API-Pfad wird durch die Kombination von `environment.server` und `/events/relevant` erstellt. Hierbei ist `environment.server` eine Konfigurationsvariable, die auf `"/api"` gesetzt ist. Das bedeutet, dass der vollständige Pfad für den API-Aufruf `"http://localhost:8080/api/events/relevant"` ist.

Die Methode gibt ein Observable, welches oft in JavaScript-Frameworks wie Angular für asynchrone Ereignisse verwendet wird, ähnlich einem Datenstrom, der im Laufe der Zeit Werte liefert, zurück, das entweder ein Array von Event-Objekten oder einen Fehler des Typs Error enthält. Der Fehler kann verschiedene Ursachen haben, wie z. B. die Nichtverfügbarkeit der Serverressource oder Serverfehler.

Das Observable wird durch die Verwendung des Angular HTTP-Clients und des get-Operators erstellt. Um die Serverantwort zu verarbeiten, wird die map-Funktion eingesetzt.

11.4.2 Aufbereiten der Events

Da die Applikation die Outlook Events erhalten hat, kann es diese Daten aufbereiten, um sie optimal anzeigen zu können.

Die Sortierung erfolgt in absteigender Reihenfolge, wobei Ereignisse mit früheren Startzeiten zuerst erscheinen. Dafür wird die Methode `sort` verwendet, und eine Vergleichsfunktion definiert, wie die Elemente verglichen und angeordnet werden sollen. Diese Vergleichsfunktion konvertiert die Startzeiten der Events in Datumsobjekte und verwendet dann die Methode getTime(), um den Zeitabstand in Millisekunden seit dem 1. Januar 1970 (UTC) zu erhalten. Dieser ist nämlich der Standardwert von Date. Das Ergebnis bestimmt die Reihenfolge der Events im sortierten Array.

```
const events :Event[]  = response as Event[];

// Sort the events based on startDateTime in ascending order
events.sort( compareFn: (a :Event , b :Event ) => {
    const dateA :Date  = new Date(a.startDateTime!);
    const dateB :Date  = new Date(b.startDateTime!);
    return dateA.getTime() - dateB.getTime();
});
```

Code 11.76 – Sortieren der Events

Weiters muss die Zeitzone der Events angepasst werden, welcher durch den Standort von Microsoft standardmäßig auf Eastern Time (US & Canada) eingestellt ist. Gelöst wird dieses Problem durch das folgende Code Segment.

```
events.forEach(event :Event  => {
    const options: Intl.DateTimeFormatOptions = {
        timeZone: 'CET',
        hour: '2-digit',
        minute: '2-digit',
        second: '2-digit',
        hour12: false,
    };

    event.startDateTime! = new Date(event.startDateTime!.toLocaleString( locales: undefined, options));
    event.endDateTime! = new Date(event.endDateTime!.toLocaleString( locales: undefined, options));
```

Code 11.77 - Zeitzonen Anpassung

Hier wird ein Konstantenobjekt namens `options` erstellt, um die Formatierung der Uhrzeit und die Zeitzone zu steuern. Die Zeitzone ist spezifisch auf 'CET' (Mitteleuropäische Zeit) eingestellt. Zusätzlich dazu wird die Uhrzeit im 24-Stunden-Format mit zweistelligen Stunden,

Minuten und Sekunden dargestellt. Danach werden den Start- und Endzeiten jedes Ereignisses neue Werte zugewiesen. Dafür wird die `toLocaleString`-Methode verwendet, um die UTC-Zeit des Events in ein lokalisiertes Datum und eine Uhrzeit gemäß den definierten `options` umzuwandeln.

Das resultierende lokalisierte Datum wird dann in ein neues JavaScript-Datumsobjekt (`new Date(...)`) überführt und den ursprünglichen Start- und Endzeiten des Ereignisses zugewiesen.

Weiters haben Events auch noch Wiederholungstypen, wie „Täglich“, „Wöchentlich“, „Monatlich“ oder Jährlich“. Leider wird bei genau diesen Wiederholungstypen nicht das nächste Datum des zu wiederholenden Termins übertragen, sondern nur das ursprüngliche Datum, an dem das Event zum ersten Mal stattfand. Dadurch muss das Programm anhand des ursprünglichen Datums und des Wiederholungstyp den nächsten Termin berechnen. Zusätzlich werden hierbei auch die schon vergangen Termine, die aber keinen Wiederholungstyp besitzen herausgefiltert.

Der nachfolgende Code-Ausschnitt ist genau für das zuvor genannte zuständig. Hierbei wird für jeden Fall des Wiederholungstyp unterschiedliche Berechnungen angestellt.

```
let date: Date = new Date()
date.setTime(Date.now())
if (event.repeatingType?.toString() === "NONE" && event.startDateTime?.getTime() < date.getTime() && event.endDateTime?.getTime() < date.getTime()) {
    events.splice(events.indexOf(event),  deleteCount: 1)
}
if (event.repeatingType?.toString() === "DAILY") {
    event.startDateTime?.setMonth(date.getMonth())
    event.startDateTime?.setDate(date.getDate())
    event.startDateTime?.setFullYear(date.getFullYear())

    event.endDateTime?.setFullYear(date.getFullYear())
    event.endDateTime?.setDate(date.getDate())
    event.endDateTime?.setMonth(date.getMonth())
}
if (event.repeatingType?.toString() === "WEEKLY" && !this.areDatesInSameWeek(event.startDateTime!, date)) {
    const day : number  = event.startDateTime?.getDay()! - date.getDay()
    event.startDateTime?.setDate( date: event.startDateTime?.getDate() + day)
    event.endDateTime?.setDate( date: event.endDateTime?.getDate() + day)
}
if (event.repeatingType?.toString() === "MONTHLY") {
    if (date.getMonth() > event.startDateTime?.getMonth()!) {
        event.startDateTime?.setMonth(date.getMonth())
        event.endDateTime?.setMonth(date.getMonth())
    }
}
if (event.repeatingType?.toString() === "YEARLY") {
    if (date.getFullYear() > event.startDateTime?.getFullYear()!) {
        event.startDateTime?.setFullYear(date.getFullYear())
        event.endDateTime?.setFullYear(date.getFullYear())
    }
```

Code 11.78 - Wiederholungstypen Auflösen

Konkret wird hier zuerst für den Fall, dass ein Ereignis keinen Wiederholungstyp (NONE) aufweist und sowohl die Start- als auch die Endzeiten in der Vergangenheit liegen, das Event lokal entfernt.

Als nächstes werden bei Events mit einem täglichen Wiederholungstyp (DAILY) Start- und Endzeiten auf den heutigen Tag aktualisiert.

Bei einem wöchentlichen Wiederholungstyp (WEEKLY) wird überprüft, ob das Event in derselben Woche wie das aktuelle Datum liegt. Falls nicht, werden Start- und Endzeiten in die jetzige Woche in den richtigen Wochentag verschoben.

Ähnlich zu dem wöchentlichen Wiederholungstyp wird bei Events mit einem monatlichen Wiederholungstyp (MONTHLY) überprüft, ob das Ereignis im gleichen Monat wie das aktuelle Datum liegt· Falls nicht, werden Start- und Endzeiten auf den aktuellen Monat gesetzt.

Und zuletzt wird noch überprüft, ob bei einem Event mit einem jährlichen Wiederholungstyp (YEARLY) die Start- und Endzeiten im gleichen Jahr wie das aktuelle Jahr liegen, sonst werden diese wieder in das aktuelle Jahr verschoben.

11.4.3 Event Übersicht

Als nächstes sollen diese Events, die durch das Programm wie im vorigen Abschnitt zu sehen ist, aufbereitet wurden, durch das Angular Frontend angezeigt werden. Für diese Anzeige werden sogenannte „Cards“ verwendet, welche das Programm durch die Angular Bibliothek Material erhält. Mit dieser Bibliothek kann ein Frontend Entwickler auch ohne großes Knowhow eine Webseite gut designen.

Dafür wird eine eigene Komponente „Card“ erzeugt, welche zuständig ist eine Karte, also das einzelne Event, mit dem Titel, dem Ort und der Start- bzw. Endzeit anzuzeigen.

```
<mat-card [ngClass]="{'card': true, 'today': isTodayEvent(event.startDateTime)}" fxLayout="column" style="left: 20px">
    <div fxLayout="row" fxLayoutAlign="space-between center">
        <div fxLayoutAlign="center" class="card-location">
            {{ event.location?.name }}
        </div>
    </div>
    <div class="card-title">
        {{ event.title }}
    </div>
    <div fxLayout="row" fxLayoutAlign="center center">
        <div class="card-date">
            {{(event.startDateTime) | date: format: 'dd MMM yyyy HH:mm': timezone: 'UTC+2'}}
            -
            {{ event.endDateTime | date: format: 'HH:mm': timezone: 'UTC+2' }}
        </div>
    </div>
</mat-card>
```

Code 11.79 - Event Card

Der darüberliegende Code-Abschnitt zeigt nun den HTML-Aufbau einer „Card“.

Hierbei wird der Standort des Ereignisses in einem separaten Abschnitt angezeigt, um dem Benutzer einen schnellen Überblick zu geben, wo das Ereignis stattfindet.

Ebenfalls wird hier auch noch die Startzeit im Format 'Tag Monat Jahr Stunden: Minuten' und die Endzeit im Format 'Stunden: Minuten' angezeigt.

Konkret können sie auch das Label mat-card erkennen, welches sich am Anfang und am Ende befindet, dies stammt von der besagten Bibliothek Material. Dieses Label verleiht herbei der Komponente die Kartenähnliche Struktur.

In diesem Fall jedoch wurde die Karte noch weiter von ihrer Grundstruktur verändert, sodass die Karte nun abgerundete Ränder besitzt durch `border-radius:40px;`

Weiters wird die Karten auch noch eingefärbt durch den Abschnitt “`background:`” im darunter liegenden Code Ausschnitt. Diese Farben sind hierbei vordefiniert von Porsche Informatik. Zusätzlich wird auch noch ein Schatten hinzugefügt mittels des Abschnittes “`box-shadow:`” und es erhält auch noch das Attribut “`transition all 0.09s ease-out`" dies gewährleistet einen fließenden und ansprechenden visuellen Übergang, wenn sich das Styling der einzelnen Elemente ändern sollte.

```css
.card {
  width: 600px;
  border-radius: 40px;
  background: linear-gradient(90deg, rgba(165, 201, 202, 1) 0%, rgba(165, 201, 202, 1) 80%, rgba(105, 146, 147, 1) 100%);
  border: #eee;
  box-shadow: 7px 7px 13px 0 rgba(50, 50, 50, 0.22);
  transition: all 0.09s ease-out;
  padding: 0 5px;
  margin-left: 0;
  margin-bottom: 20px;
  margin-top: 0px;
}
```

Code 11.80 -Event Card Styling

```
<mat-card [ngClass]="{'card': true, 'today': isTodayEvent(event.startDateTime)}" fxLayout="column" style="left: 20px">
```

Code 11.81 – Event Today

Beim nächsten Code-Ausschnitt ist der Teil `[ngClass]="{'card': true, 'today': isTodayEvent(event.startDateTime)}"` von großer Bedeutung, da hierbei das „Card“- Objekt an einen Style-Typen gebunden wird, wenn das Event heute stattfindet. In diesem Fall wird die Karte in einem roten Ton eingefärbt. Damit der Benutzer auf den ersten Blick erkennen kann, dass das Event von höherer Relevanz ist.

11.4.4 Hinzufügen eines Events

Weiters kann der Benutzer noch ein Event über das Webinterface hinzufügen, dazu gibt es unter der Anzeige der verschiedenen Events einen "Add Event"-Button, der den Benutzer zu der „Add-Event“ Anzeige führt. Hier soll nun der Benutzer die verschiedenen Parameter eines Events eingeben. Unter anderem muss der Mitarbeiter von Porsche Informatik hier den Titel des Events, die Beschreibung, das Startdatum, das Enddatum und noch weitere Daten eingeben.

```
<div class="form-group" [ngClass]="{ 'missing-field': isTitleMissing }">
    <mat-form-field class="form-field">
        <mat-label>Event Name:</mat-label>
        <input matInput [(ngModel)]="event.title" id="eventName">
        <mat-error *ngIf="isTitleMissing">Event Name is required</mat-error>
    </mat-form-field>
</div>

<div class="form-group" [ngClass]="{ 'missing-field': isDescriptionMissing }">
    <mat-form-field class="form-field">
        <mat-label>Description</mat-label>
        <textarea matInput [(ngModel)]="event.description" id="eventDescription"></textarea>
        <mat-error *ngIf="isDescriptionMissing">Description is required</mat-error>
    </mat-form-field>
</div>
```

Code 11.82 - Event Parameter #1

Der darüberliegende Code Ausschnitt zeigt hier den Aufbau einer solchen Eingabe.

Es zeichnet ein Formularfeld für den Namen eines Events in einer Webanwendung unter Verwendung der Angular-Direktive ngClass, an die die Bedingung übergeben wird (`„isTitleMissing“`), und die abhängig von dieser Bedingung eine CSS-Klasse hinzufügt (`„missing-field“`), um das Feld im UI visuell anzuzeigen.

Innerhalb des Formularfeldes wird die Angular Material Design-Direktive `(mat-form-field)` verwendet, um eine konsistente Benutzeroberfläche zu liefern, wobei das Label „Event Name“ in dem Eingabefeld angezeigt wird. In dieses Feld kann der Benutzer den Namen des Events eingeben, und die Datenbindung `([ngModel])` synchronisiert die Benutzereingabe mit der Variable `„event.title“`.

Wenn das Eingabefeld leer ist und der Eventname fehlt, wird eine Fehlermeldung unter dem Eingabefeld `"Event Name is required"` angezeigt, die durch die Angular `ngIf-Direktive` gesteuert wird. Diese wird nur angezeigt, wenn `isTitleMissing` true ist. Dies ermöglicht eine benutzerfreundliche Validierung und Rückmeldung für den Benutzer, um sicherzustellen, dass alle erforderlichen Informationen eingegeben wurden.

```
<div class="form-group">
    <div style="...">
        <mat-list class="form-field-list" style="...">
            <mat-list-item *ngFor="let addedMember of addedMembers.slice(0, 3); let i = index">
                <p>{{ addedMember }}</p>
                <mat-divider [vertical]="true"></mat-divider>
            </mat-list-item>
        </mat-list>
        <button mat-raised-button color="accent" class="larger-button"
                style="margin-bottom: 20px; margin-top: 20px;" (click)="openAddMembersDialog()">
            Add Attendees
        </button>
    </div>
</div>
```

Code 11.83 - Event Parameter #2

Als nächstes muss der Benutzer die Teilnehmer (Attendees) des Events einladen.

Dazu wird ein HTML-Element mit der Klasse `'form-group'` definiert, in den Informationen zu den Teilnehmern angezeigt werden.

Innerhalb dieses Bereichs befindet sich ein weiteres Div-Element, das verwendet wird, um die Größe des Inhalts automatisch anzupassen.

In diesem inneren Div-Element wird eine Material Design-Liste (`mat-list`) erstellt, die die hinzugefügten Teilnehmer anzeigt. Die Liste hat einen dunklen Hintergrund (#525252), um sie hervorzuheben.

Für jeden hinzugefügten Teilnehmer wird ein Listenpunkt (`mat-list-item`) erstellt, der den Namen des Teilnehmers enthält. Die Namen werden dynamisch aus einem Array (`added-Members`) generiert und nur die ersten drei Teilnehmer werden angezeigt (`slice(0, 3)`).

Zwischen den Teilnehmern wird ein horizontaler Trenner (`mat-divider`) eingefügt, um sie voneinander abzugrenzen.

Unterhalb der Liste befindet sich ein erhöhter Material Design-Button (`mat-raised-button`), der die Möglichkeit bietet, weitere Teilnehmer hinzuzufügen. Der Button ist farblich hervorgehoben (`accent`) und größer (`larger-button`), um ihn leichter erkennbar zu machen.

Ein Klick auf diesen Button löst die Funktion `openAddMembersDialog()` aus, die ein Dialogfenster öffnet, in dem neue Teilnehmer hinzugefügt werden können. Dieses Dialog Fenster ist eine vordefinierte Komponente, die zuvor schon von Porsche Informatik zur Verfügung gestellt wurde.

```
    <div class="form-group">
        <mat-form-field class="form-field">
            <mat-label>Location</mat-label>
            <input matInput [(ngModel)]="location" id="eventLocation">
        </mat-form-field>
    </div>


    <div class="form-group">
        <mat-form-field class="form-field" style="...">
            <mat-label>Select RepeatingType</mat-label>
            <mat-select [(ngModel)]="event.repeatingType" id="repeatingType">
                <mat-option *ngFor="let repeatingOption of repeatingOptions" [value]="repeatingOption">
                    {{ repeatingOption }}
                </mat-option>
            </mat-select>
        </mat-form-field>
    </div>

</div>
```

Code 11.84 - Event Parameter #3

Zuletzt muss der Benutzer den Standort des Meetings und den Wiederholungstyp angeben.

Im ersten Abschnitt des Codes wird ein Formularfeld für die Eingabe des Veranstaltungsorts definiert. Dabei wird das Material Design von Angular `(mat-form-field)` verwendet, um eine einheitliche Benutzeroberfläche sicherzustellen. Das Label "Location" wird über dem Eingabefeld angezeigt, in das der Benutzer den Veranstaltungsort eingeben kann. Die Eingabe wird über eine zweifache Datenbindung `[(ngModel)]` mit der Variable `'location'` synchronisiert.

Im nächsten Abschnitt wird ein Dropdown-Menü erstellt, um den Wiederholungstyp für die Veranstaltung auszuwählen. Auch hier wird das Material Design von Angular verwendet, insbesondere das `mat-select-Element`. Das Label `'Select RepeatingType'` wird über dem Dropdown-Menü angezeigt. Die verfügbaren Wiederholungstypen werden aus einer Variable namens `'repeatingOptions'` geladen. Diese bestehen aus den Optionen täglich, monatlich, jährlich oder keins. Für jede Option wird eine Auswahlmöglichkeit im Dropdown-Menü angezeigt. Die ausgewählte Option wird über die `ngModel-Direktive` mit der Variable `'event.repeatingType'` synchronisiert.

Wenn der Benutzer alle erforderlichen Parameter eingegeben hat, kann er das Event durch Klicken auf den darunter liegenden `'Add Event'`-Button hinzufügen. Die Methode `onaddNe-`

`wEvent()` wird aufgerufen. Zunächst überprüft sie, ob alle erforderlichen Felder für das Ereignis ausgefüllt wurden, einschließlich Titel, Beschreibung, Start- und Enddatum/-uhrzeit. Wenn ja, wird das Ereignis weiterverarbeitet.

Wenn das Feld für den Wiederholungstyp (`repeatingType`) des Ereignisses nicht vorhanden ist, wird es auf den Standardwert `'NONE'` gesetzt.

Anschließend werden Teilnehmer hinzugefügt (`addAttendees()`).

Hierbei wird das Ereignis über einen `EventService` emittiert, und der Benutzer wird zur Seite mit den Ereignissen zurückgeleitet (`this.router.navigate(['/events'])`). Eine Erfolgsmeldung wird mithilfe eines `MatSnackBar` angezeigt, der dem Benutzer mitteilt, dass das Ereignis erfolgreich hinzugefügt wurde.

Wenn nicht alle erforderlichen Felder ausgefüllt sind, wird dem Benutzer eine Fehlermeldung angezeigt. Diese weist darauf hin, dass alle erforderlichen Felder ausgefüllt werden müssen, um ein Ereignis hinzuzufügen.

```
onaddNewEvent() :void  {
    if (
        this.event.title &&
        this.event.description &&
        this.event.startDateTime &&
        this.event.endDateTime
    ) {
        if (!this.event.repeatingType)
            this.event.repeatingType = 0;

        this.event.location = new Location();
        this.event.location.name = location.toString();
        this.addAttendees();
        this.eventService.emitEvent(this.event);
        this.router.navigate( commands: ['/events']);
        this.snackBar.open( message: 'Event added successfully',  action: 'Close',  config: {
            duration: 3000, // 3 seconds
        });
    } else {
        this
            .snackBar
            .open(
                 message: 'Please fill in all required fields'
                ,
                 action: 'Close'
                ,  config: {
                    duration: 3000
                    , // 3 seconds
                }
            )
```

Code 11.85 - Event Add Methode

Beim Wechseln auf die „Events-Page“ wird zu Beginn die Methode „ngOnInit()“ aufgerufen. Diese Methode versucht nun, das zuvor übergebene Event zum Backend abzuschicken.

Dafür wird die Methode „onEventPush“ aufgerufen. Zunächst wird überprüft, ob es bereits ein Ereignis mit dem gleichen Titel gibt. Hierfür wird in der Liste der vorhandenen Ereignisse

`(this.events)` nach einem Ereignis gesucht, das denselben Titel wie das empfangene Ereignis hat. Wenn ein Ereignis mit dem gleichen Titel gefunden wird, wird eine Benachrichtigung angezeigt, die besagt, dass ein dupliziertes Ereignis gefunden wurde.

Wenn kein Ereignis mit dem gleichen Titel gefunden wird, wird geprüft, ob alle erforderlichen Informationen für das neue Ereignis vorhanden sind. Dazu wird die Methode `checkEventInput(event)` aufgerufen. Wenn alle erforderlichen Informationen vorhanden sind, wird fortgefahren.

In diesem Fall wird das Ereignis durch einen Aufruf der Methode `addEvent(eventsArray)` des eventService an Server gesendet, um es zu erstellen. Das Ergebnis wird dann mit einem subscribe()-Aufruf verarbeitet. Wenn das Ereignis erfolgreich erstellt wurde, wird eine Erfolgsmeldung protokolliert und die Ereignisliste aktualisiert `(this.reloadEvents())`. Tritt ein Fehler auf, wird eine Fehlermeldung protokolliert.

Falls nicht alle notwendigen Informationen für das Ereignis vorhanden sind, wird eine Benachrichtigung angezeigt, die darauf hinweist, dass die obligatorischen Informationen fehlen und das Ereignis nicht erstellt werden kann.

```
onEventPush(event: Event) :void  {
    const tempEvent :Event | undefined  = this.events.find((e :Event ) :boolean  => e.title == event.title);
    if (tempEvent) {
        alert("Duplicated Event found!");
    } else if (this.checkEventInput(event)) {
        const eventsArray: Event[] = [event]; // Create an array with a single event
        console.log("eventsArray:", eventsArray);
        this.eventService.addEvent(eventsArray).subscribe(
            next: (response :Event[] | Error ) :void  => {
                console.log("Event added successfully:", response);
                // Update the 'this.events' array with the updated data from the server
                this.reloadEvents();
            },
            error: (error) :void  => {
                console.error("Error adding event:", error);
            }
        );
    } else {
        alert("The mandatory information is not given to make a new Event!");
    }
}
```

Code 11.86 – onEventPush()

```
addEvent(event: Event[]): Observable<Event[] | Error> {
    console.log("addEvent: " + `${environment.server}/events`, event)
    return this.http.post<Event[]>( url: `${environment.server}/events/postEvents`, event).pipe(catchError(this.errorHandler));
}
```

Code 11.87 – AddEvent()

Und zu guter Letzt muss das Event noch an die Graph API übergegeben werden, dies funktioniert durch den darüberliegenden Code Ausschnitt mit der Methode addEvent(). Hierbei wird der API-Pfad durch die Kombination von `environment.server` und `/events/postEvent` erstellt. Dabei ist `environment.server` eine Konfigurationsvariable, die auf `"/api"` gesetzt ist. Der vollständige Pfad für den API-Aufruf lautet daher `"http://localhost:8080/api/events/postEvents"`.

12. Ergebnis

Das Ergebnis der Diplomarbeit besteht aus den in 7.3 definierten Zielen: dem Erleichtern des Buchens von Desk Groups, einer Übersicht der Team-Mitglieder für Teamleiter, eine Auswertung der Auslastung von Standorten und einzelnen Teams und eine Übersicht von bevorstehenden Events, sowie das Anlegen dieser. Die Daten der Buchungen und Mitarbeiter werden laut Vorgabe mittels der Graph-API mit Outlook synchronisiert.

12.1 Buchen von Desk Groups

Die Funktion zum Buchen einer Desk Group wurde wie in 7.4 erwähnt bereits implementiert. Sie wurde jedoch mithilfe des Raumplans, dessen Implementierung in 11.1 beschrieben wird, verbessert. Der Benutzer kann dadurch beim Buchen einsehen, wo sich in einem Stockwerk sich die Gruppe befindet, bei der er einen Platz buchen möchte. Ihm fällt daher die Wahl des Platzes und das Auffinden dieser vor Ort leichter.

12.2 Übersicht der Teams

Als Hilfe-Tool für die Team- und Abteilungsleiter wurde eine Ansicht realisiert, welche alle Untergeordneten des Leiters übersichtlich auflistet. Falls mehrere Teams dem Nutzer in der Hierarchie untergeordnet sind, kann der Benutzer beliebige Daten dazu einsehen. Dabei kann er erkennen, welche Mitarbeiter, entweder im Büro oder im Home-Office arbeiten oder abwesend sind und dies über die ganze Woche.

12.3 Auswertung der Auslastung

Die Statistik-Ansicht wurde ebenfalls für die Team- und Abteilungsleiter des Unternehmens implementiert. Hier kann nachgesehen werden, wie viel von der gesamten Anzahl der Sitzplätze besetzt wird. Durch das Verwenden von den Filterkriterien, ist es dadurch möglich die Auslastung aller Standorte, eines Standortes, eines Stockwerkes und eines Teams graphisch zu veranschaulichen.

12.4 Events

Die Event-Ansicht und die dazugehörige Seite zum Hinzufügen von Events wurde für die Mitarbeiter implementiert, damit diese nicht nur ihre Desks buchen, sondern auch noch direkt die Events des Tages übersichtlich einsehen können. Hier kann man die heutigen, morgigen, aktuellen des Monats oder direkt alle Events und deren weitere Informationen einsehen. Zusätzlich können die Mitarbeiter durch eine weitere Seite neue Events hinzufügen.

13. Resümee

13.1 Allgemeines Resümee

Während des Praktikums wurden wir mit der Arbeitswelt der Softwarefirmen familiarisiert. Als Teil eines Projekt-Teams haben wir an den wöchentlichen Meetings teilgenommen und dadurch ein besseres Bild davon erhalten, wie diese abgehalten werden. Außerdem kam es oft vor, dass der Projektleiter oder sonstige Mitglieder des Teams entweder von zuhause oder von einem anderen Standort der Porsche Informatik aus an diesen Treffen teilnahmen. Dies ist mittlerweile keine Seltenheit mehr für die meisten Unternehmen, vor allem aufgrund der wachsenden Beliebtheit des Home-Offices. Dadurch wurde uns das Absprechen mit unseren Kollegen, in Bezug auf projektspezifische Details, egal ob in Person oder online, geläufig.

Die Implementierung unserer Diplomarbeit hat uns gezeigt, wie zeitaufwändig ausführliches Testen sein kann. In unserem Fall lag es unter anderem an dem vollständigen Neustarten des Backend-Programmes, was jedes Mal um die 20 Minuten brauchte. Außerdem machten wir auch Bekanntschaft mit der Tatsache, dass der Entwickler selbst nicht immer die Berechtigungen für die von ihm entwickelten Features hat.

13.2 Persönliches Resümee

Noah Grillenberger

Mir hat mein Arbeitspaket gut gefallen. Die Implementierung der Algorithmen war eine Herausforderung, jedoch auch ein interessantes Problem zu lösen, ähnlich wie ein Puzzle. Auch das Einlesen der Excel-Files war informativ. Ich kann mir gut vorstellen wieder einmal ähnliche Aufgaben zu erarbeiten.

Michael Hintermayr

Das Kennenlernen der Microsoft Graph API und das Erlernen vom Spring Framework haben einige Zeit in Anspruch genommen, allerdings sind diese Technologien weiterhin relevant und können mir im Laufe meiner Karriere noch behilflich sein. Weiters konnte ich im Laufe dieser Diplomarbeit auch mit einem zuverlässigen Team an einem großen Projekt arbeiten, wodurch mir die Wichtigkeit von Zusammenarbeit nochmals nähergebracht wurde. Alles in einem konnte ich neue Erfahrungen für Technologien und Teamwork sammeln.

Paul Lanzinger

Während meiner Diplomarbeit konnte ich nicht nur meine Angular-Fähigkeiten auf ein neues Level bringen, sondern auch wertvolle Erfahrungen im Team eines großen Konzerns sammeln. Diese Zeit hat mir nicht nur technisches Know-how vermittelt, sondern auch gezeigt, wie wichtig gute Teamarbeit für den Projekterfolg ist.

Ich freue mich darauf, dieses Wissen in Zukunft weiter auszubauen und mich sowohl beruflich als auch persönlich weiterzuentwickeln.

Paul Oprea

Es war eine interessante Abwechslung für mich, mal das Ästhetische als Schwerpunkt meiner Aufgabenpakete zu haben. Ich habe mir Gedanken darüber machen müssen, was nicht nur gut, aber sich auch intuitiv verwenden lassen würde. Um diese Ideen umzusetzen, habe ich Angular, von Grund auf, lernen und CSS auf einem Niveau nützen müssen, wie es bisher noch nicht an meiner Schule nötig war. Auf diese Weise habe ich mir durch die Diplomarbeit nicht nur viel technisches Wissen angeeignet, aber auch ein Bild davon bekommen, was es heißt, ein Frontend-Entwickler bei einem Unternehmen zu sein. Ich könnte mir auf jeden Fall vorstellen in Zukunft auch beruflich derartige Aufgabenstellungen zu bewältigen.

14. Abbildungsverzeichnis

15. Codeverzeichnis

16. Quellenverzeichnis

Angular - Single Page Application. (Januar 2024). Von https://www.studysmarter.de/schule/informatik/webentwicklung/single-page-application/ abgerufen

Apache POI. (März 2024). Von https://blog.fileformat.com/de/spreadsheet/insert-header-and-footer-in-excel-using-apache-poi-for-java/ abgerufen

Design: Figma. (März 2024). Von Figma: https://www.figma.com/de/design/ abgerufen

Docker-Container. (Januar 2024). Von https://www.ionos.at/digitalguide/server/knowhow/docker-container/ abgerufen

Intellij Ultimate. (März 2024). Von https://www.biteno.com/was-ist-intellij-idea/ abgerufen

Mediaevent: SVG-Viewport, SVG-Koordinaten und viewBox. (Mai 2023). Von Mediaevent.de: https://www.mediaevent.de/tutorial/svg-viewbox-koordinaten.html abgerufen

Monschau, A. (Januar 2017). *Kaffee in Zellen: entwickler.de*. Von entwickler.de: https://entwickler.de/software-architektur/kaffee-in-zellen abgerufen

Porsche Informatik: Das Unternehmen. (Januar 2024). Von Porsche Informatik: https://www.porscheinformatik.com/das-unternehmen/ abgerufen

Spring Framework & Spring Boot. (Januar 2024). Von https://www.ibm.com/de-de/topics/java-spring-boot abgerufen

SVG Generator: Apache XML Graphics. (Januar 2024). Von The Apache XML Graphics Project: https://xmlgraphics.apache.org/batik/using/svg-generator.html abgerufen

Visual Studio Code. (März 2024). Von https://www.computerwoche.de/a/so-finden-sie-zum-richtigen-entwicklungs-tool,3553962 abgerufen

Webstorm. (März 2024). Von https://www.getapp.at/software/2048273/webstorm abgerufen

17. Glossar

Anfragemethode
: Eine vom HTTP-Standard definierte Methode, die dem Empfänger mitteilt, was für eine Art von Aktion ausgeführt werden soll. Beispiele wären GET, POST, PUT oder DELETE. 87

AuthProvider
: Stellt die Authentifizierung für den Client zur Verfügung 80

AutoWired
: Eine Annotation, die Referenzen auf Service- und andere Objekte in den Konstruktor injiziert. 86

Credential
: Speichert zur sicheren Verwendung die Anmeldedaten des Benutzers 80

DeleteMapping
: Eine Annotation, die den Pfad zu einer API-Schnittstelle mit der Anfragemethode DELETE anlegt. 87

DTO
: Abkürzung für "Data Transfer Objekt" 88

Enumeration
: Eine vollständige Auflistung von zusammenhängenden Elementen 85

Event
: Ein Ereignis im Alltag eines Mitarbeiters, wie ein Meeting oder ein gemeinsames Frühstück 85

GetMapping
: Eine Annotation, die den Pfad zu einer API-Schnittstelle mit der Anfragemethode GET anlegt. 87

hovert
: Den Mauszeiger über etwas bewegen 24

HTTP-Status-Code
: Der Status der gesendeten Anfrage in Form einer dreistelligen Zahl. Beispiele wären 200 OK oder 404 NOT FOUND. 87

InteractiveBrowserCredential
: Fragt die Anmeldedaten über den Microsoft Anmeldebildschirm im Standardbrowser ab 80

JpaRepository
: Eine bereitgestellte Schnittstelle, die von anderen Schnittstellen zur Erstellung eines Repositorium erweitert werden kann. 89

Long
: Ein Datentyp zur Darstellung von großen Zahlen. 89

Mapping
: Eine Art von Annotation, die den Pfad zu einer API-Schnittstelle mit einer bestimmten Anfragemethode anlegt. 87

Operation
: Eine Annotation, um eine API-Schnittstelle kurz zu beschreiben. 87

Page
: Eine begrenzte Menge von Datensätzen, welche von einer API gesendet wird, um die Übertragungseffizienz zu erhöhen 82, 84

PathVariable
: Eine Annotation, um eine Variable, die in der URL der Anfrage übergeben wird, zu definieren. 88