Fachhochschule für die Wirtschaft Hannover
- FHDW -

PRAXISARBEIT

Ressourcen- und Terminplanung in Office365 mit Hilfe von Microsoft Graph API

Name: **Adham Aijou**
Brinker Straße 72
30851, Langenhagen

Betreuer: **Prof. Dr. Ing. Klinger**

Studiengruppe: **HFI421IN**

Matrikelnummer: **600142**

Ausbildungsbetrieb: **DOOH media GmbH**
Frankenring 18
30855 Langenhagen

Eingereicht am: xx.xx.xxxx

Inhaltsverzeichnis

1 Einleitung

Die Praxisarbeit befasst sich mit der Ressourcen- und Terminplanung in Office365 mithilfe von Microsoft Graph API [microsoftGraphApi]. Die Arbeit ist in drei Teile gegliedert. Im ersten Teil wird die Theorie der Ressourcen- und Terminplanung in Office365 mithilfe von Microsoft Graph API erläutert. Im zweiten Teil wird die praktische Umsetzung der Theorie beschrieben. Im dritten Teil wird die Arbeit abschließend bewertet.

1.1 Fragestellungen der Arbeit

Die Fragestellungen der Arbeit lauten:

- Wie funktioniert die Ressourcen- und Terminplanung in Office365 mithilfe von Microsoft Graph API?
- Wie kann die Ressourcen- und Terminplanung in Office365 mithilfe von Microsoft Graph API praktisch umgesetzt werden?
- Wie kann die Arbeit abschließend bewertet werden?

Konkret bedeutet dies:
Inwiefern und wie sinnvoll, ist der Einsatz von Microsoft Graph API für die Ressourcen- und Terminplanung in Office365, im Vergleich zu anderen APIs. Dabei sollte berücksichtigt werden, dass dies auf den Kundenauftrag bezogen ist. Wie sollte sowas umgesetzt werden und welche Aspekte der Microsoft Graph API sind dafür relevant? Letztenendes soll herausgefunden werden, wie man die Arbeit quantitativ und qualitativ bewerten kann.

1.2 Ziele der Arbeit

Die Ziele der Arbeit sind wie folgt:

- Die Theorie der Ressourcen- und Terminplanung in Office365 mithilfe von Microsoft Graph API zu erläutern.
- Die praktische Umsetzung der Theorie anhand eines Kundenauftrags zu beschreiben.
- Die Arbeit abschließend zu bewerten.

Für die Ziele bedeutet dies, dass die Fragestellungen der Arbeit beantwortet werden müssen. Sowohl die Theorie als auch die praktische Umsetzung der Theorie, müssen in der Arbeit beschrieben werden. Es muss immer wieder auf die Kundenanforderungen zurückgegriffen werden.

1.3 Ergebnisse der Arbeit

Die Arbeit hat alle Ziele erreicht. Die Microsoft Graph API ist erfolgreich eingesetzt und die Ressourcen- und Terminplanung in Office365, mithilfe von der Microsoft Graph API, wurde erfolgreich als **Single Page Application (SPA)** umgesetzt. Die **Total-Blocking-Time (TBT)** der Applikation liegt, aufgrund von hoher Komplexität, bei ca. 500ms. Nachdem die Seite jedoch erstmal geladen wurde, dauert beispielsweise das Öffnen des Buchungsdialogs nur 2ms, welche die hauptsächliche Interaktion des Anwenders mit der Seite darstellt.
Die detaillierten Aspekte Ergebnisse der Arbeit sind in Kapitel **??** und Kapitel 4.2 beschrieben. Das ausformulierte Ergebnis und die Schlussfolgerung dieser, sind in Kapitel 5 erläutert.

1.4 Zusammenfassung

Die Praxisarbeit wurde durch einen Kundenauftrag ins Leben gerufen. Es sei umständlich für gewisse Kunden für Konferenzräume Termine zu buchen. Jedes Mal muss nämlich ein Mitarbeiter die Verfügbarkeit eines Konferenzraumes auf einer Applikation mit zu vielen Zwischenschritten prüfen und dann einen Termin buchen. Dieser Prozess ist sehr zeitaufwendig und kann zu Fehlern führen. Falls beispielsweise 30 Räume zur Verfügung stehen und der Mitarbeiter ausschließlich diesen einen Raum auf Verfügbarkeit prüfen möchte, ist dies einerseits umständlich und andererseits gegebenenfalls durch den Administrator eingeschränkt. Wenn der Administrator dann den Raum für den Kunden freigibt, muss der Mitarbeiter den Raum nochmal auf Verfügbarkeit prüfen, bevor er den Termin bucht. Falls zu viele Rechte freigegeben werden, kann es zu Fehlern kommen, die dann von einem Administrator behoben werden müssen. Zudem hätten dann Mitarbeiter Rechte, die sie nicht benötigen und könnten außerhalb ihrer Zuständigkeit Fehler machen.

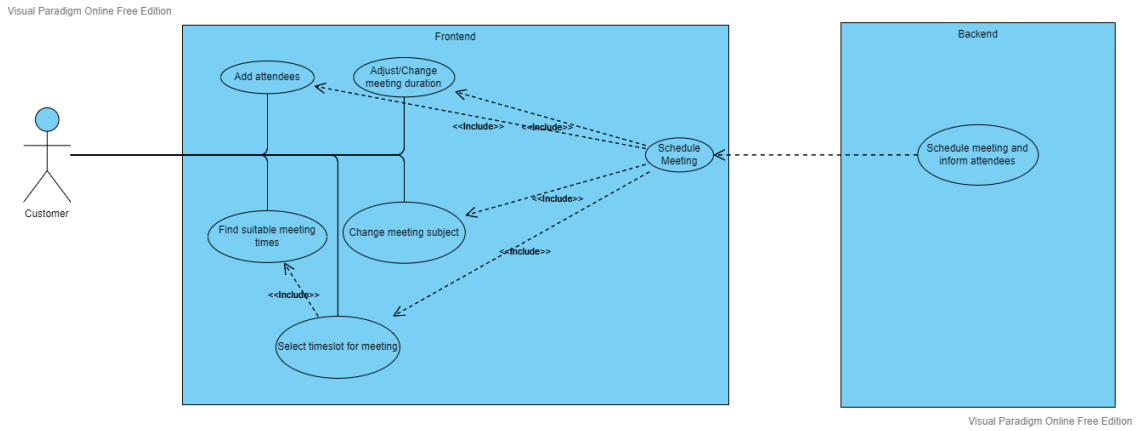
Daher soll eine Anwendung entwickelt werden, die es ermöglichen soll, dass der Mitarbeiter selbstständig einen Termin buchen oder einsehen kann. Außerdem soll überprüft werden können, wann Termine für den Raum stattfinden sollen, wenn die Person schon im Gebäude und sich vor dem Raum befindet.
Bei diesem Kunden sind viele Mitarbeiter nicht aus der gleichen Abteilung oder dem gleichen Subunternehmen und besitzen folglich unterschiedliche Firmennamen und Logos. Wenn ein Kalender keine detaillierteren Termininformationen darstellt, müsste der Mitarbeiter immer noch nachschauen, ob der Termin für ihn gedacht sei. Die Buchungen direkt vor dem Raum sind dafür gedacht, dass ein Mitarbeiter den Raum sofort buchen kann.
Sollte dies nicht der Fall sein, müsste der Anwender, im Voraus, an einem anderen Endgerät, welches Outlook oder Teams besitzt, einen detaillierten Termin buchen, damit alle Beteiligten wissen, wann der Termin stattfindet und wer anwesend sein wird. Dort kann der Anwender dann auch angeben, welche Firma der Gastgeber und welche der Gast sein soll.

1.5 Use Case

Hierzu wurde ein Use Case erstellt, welcher die Anforderungen an das System beschreibt. Dieser Use Case wurde in einem Tabellenformat erstellt, welches in der folgenden Tabelle dargestellt wird:

Use Case	Ressourcen- und Terminplanung in Office365 mithilfe von Microsoft Graph API
Titel	Finde freie Termine in einem Kalender für eine bestimmte Ressource
Beschreibung	Der Benutzer möchte einen Termin in einem Kalender für eine bestimmte Ressource finden.
Akteure	Benutzer, Ressource, Kalender
Vorbedingungen	Der Benutzer ist angemeldet.
Nachbedingungen	Der Benutzer hat einen Termin in einem Kalender für eine bestimmte Ressource gefunden und/oder hat einen Termin gebucht.
Normaler Ablauf	1. Der Benutzer geht zu einem Tablet, welcher vor dem Raum angebracht wurde, welcher die Ressource stellvertretend repräsentieren soll. 2. Der Benutzer sieht die verfügbaren Zeiten für die Ressource und den jetzigen Status der Ressource. 3. Der Benutzer wählt einen Termin aus. 4. Der Benutzer bucht den Termin. 5. Der Benutzer erhält eine Bestätigung oder eine Fehlermeldung.

Dazu gibt es auch ein Diagramm, welches den Ablauf des Use Cases beschreibt:



1.6 Systemschnittstellen

Hier wird ein Mal die benötigte Kommunikation mit der Microsoft Graph API als Systemschnittstelle tabellarisch erläutert.

Kurzbeschreibung:	In der UI kann der User sich mit Konto der Ressource anmelden, die für die Zukunft als Repräsentant für die reelle Ressource (z.B einem Raum) gelten soll. Daraufhin gilt dieser Raum immer als Teilnehmer der Buchungen und kann in seinen verfügbaren Zeiten gebucht werden. Basierend auf diesen Zeiten, bekommt der Benutzer verfügbare timeslots zur Buchung angezeigt. Die Dauer des Termins soll einstellbar sein. Es gibt zudem einen optionalen Betreff für den Termin. Der gebuchte Termin wird in der Terminübersicht, für den aktuellen und folgenden Tag, angezeigt. Zudem soll der Nutzer anhand farblicher Indikatoren (siehe Abschnitt 3.7.1) beurteilen können, ob die Ressource verfügbar ist. **REST API** Anfragen werden dann an die Microsoft Graph API geschickt, um diese Buchung zu kommunizieren.
Akteure:	Mitarbeiter und Software
Häufigkeit:	Abhängig von Firmengröße und Besprechungsfrequenz
Komplexität:	Mittlere bis hohe Komplexität
Vorbedingungen:	1. Der Benutzer muss eingeloggt sein und Zugriffsrechte akzeptiert haben. 2. Optionale Teilnehmer 3. Termindauer 4. Verfügbare Termine 5. Terminbetreff 6. Zeitzone 7. Access Token / Client ID zur Authentifizierung
Ausgaben:	1. Buchung erfolgreich oder nicht 2. Neue verfügbare Termine 3. Der neue Termin wird in der Liste der gebuchten Termine angezeigt
Use Case:	Siehe 1.5
Ausgeführte Aktionen:	Immer auszuführen: 1. Termin finden 2. Termin buchen Optional auszuführen: 1. Teilnehmer hinzufügen 2. Terminbetreff hinzufügen/ändern 3. Termindauer ändern

2 Grundlagen

Um die Arbeit in vollem Umfang zu verstehen, ist es wichtig, dass die Grundlagen von Webanwendungen und Office365 verstanden werden.

2.1 Webanwendungen

2.1.1 Was ist eine Webanwendung?

Eine Webanwendung ist eine Anwendung, die über einen Browser aufgerufen werden. Manche Webanwendungen sind auch Offline nutzbar.

2.1.2 Vor- und Nachteile von Webanwendungen

Eine Webanwendung ermöglicht es, dass die Anwendung von jedem Endgerät, welches über eine Internetverbindung verfügt, aus aufgerufen werden kann. Webanwendungen sind effizient, benötigen wenig Speicherplatz, sind heutzutage gut abgesichert und aufgrund von der Tatsache, dass vieles standardisiert ist, gut wartbar.

Ein Nachteil von Webanwendungen ist, dass nicht alle Offline nutzbar sind. Sie sind zudem nicht so schnell wie manche Desktopanwendungen und heutzutage auf die Verwendung von [JavaScript] und unterschiedlichen Frameworks, die damit verbunden sind, angewiesen, welche oftmals umfangreich sind und Dependenzen haben, die weiterhin gepflegt werden müssen. Dies führt oftmals dazu, dass falls eine Abhängigkeit nicht mehr gepflegt wird und andere geupdatet werden müssen und die Anwendung eventuell nicht mehr funktioniert. Javascript lässt Laufzeitfehler zu.

2.1.3 Was ist JavaScript?

JavaScript ist eine Skriptsprache, die auf ECMAScript basiert.

2.1.4 Was ist ECMAScript?

ECMAScript ist eine Skriptsprache, die von der European Computer Manufacturers Association (ECMA) entwickelt wurde.

2.1.5 Was ist Node.js?

Node.js ist eine JavaScript Laufzeitumgebung, die auf dem V8 JavaScript Engine von Google basiert.

2.2 Microsoft Graph API

Die Microsoft Graph API ist eine **RESTful** web API, die es einem erlaubt auf Daten von Microsoft 365 und Office 365 zuzugreifen. Mithilfe dieser API wurde das Projekt letztendlich umgesetzt. Weitere standen jedoch zur Verfügung:

- Microsoft Outlook API
- Microsoft Exchange API
- Microsoft SharePoint API
- Microsoft OneDrive API
- Microsoft Teams API
- Microsoft Power Automate

Einige dieser APIs, sind nur für bestimmte Microsoft 365 und Office 365 Abonnements verfügbar. Die Microsoft Graph API ist jedoch, derzeit, für alle Abonnements verfügbar. Zudem ist die Microsoft Graph API die einzige API, die es einem erlaubt auf alle Daten von Microsoft 365 und Office 365 zuzugreifen, da sie die meisten anderen APIs integriert. Der wichtigste Faktor bei der Entscheidung war es jedoch, dass die Microsoft Graph API, mithilfe von Azure AD, die Authentifizierung und Autorisierung von Benutzern erlaubt. Dies ist für die Anwendung von großer Bedeutung, da es dem Benutzer ermöglicht sich mit seinem Microsoft 365 oder Office 365 Account anzumelden und somit auf seine Daten zuzugreifen.

3 Abgrenzung zu ähnlichen Produkten

3.1 Microsoft Teams

Microsoft Teams ermöglicht Terminbuchungen genauso, erlaubt jedoch zu detaillierte Eingriffe in das Konto, welcher für den Raum vorgesehen ist. Einstellungen, die nicht verändert werden sollten, können dann erreicht werden.

3.2 Outlook

Outlook ist ein E-Mail-Programm, welches ebenfalls Terminbuchungen ermöglicht. Outlook ist jedoch nicht für die Terminbuchung in Konferenzräumen gedacht und kann je nach E-Mail-Konto-Typ nicht in vollem Umfang genutzt werden und ist zudem nicht immer vollständig synchronisiert. Auch hier können Einstellungen, die nicht verändert werden sollten, erreicht werden und Kalender eingesehen werden, die nur für Administratoren gedacht sind.

3.3 Microsoft Bookings

Microsoft Bookings ist eine weitere Applikation, die Terminbuchungen ermöglicht. Sie wird hauptsächlich für Terminbuchungen in beispielsweise Friseursalons, Fitnessstudios oder ähnlichen Geschäften genutzt. Dies ist jedoch der falsche Anwendungsfall für die Terminbuchung in Konferenzräumen. Auch die Einschränkungen und Freiheiten, die diese Applikation bietet, sind hier nicht passend.

3.4 Microsoft Power Automate

Microsoft Power Automate ist eine Applikation, die es ermöglicht, Workflows zu erstellen. Diese Workflows können dann automatisiert ausgeführt werden. Diese kann auch ähnliche Funktionen wie die Applikation, die in dieser Arbeit entwickelt wird, bieten. Jedoch ist die Freigabe dieser Applikationen nur für innerhalb der Organisationen vorgesehen und nicht für Kunden.

3.5 Spezifische Gründe für die Entwicklung einer eigenen Applikation

Wenn ein Kunde bei der DOOH media GmbH einkauft, kauft er im Regelfall ein umfangreiches Softwarepaket. Dieses Paket inkludiert die Möglichkeit die Geräte in einen sogenannten **Kiosk-Modus** zu versetzen. Dafür haben wir eine eigene Applikation, die Oxygen Player App, entwickelt. Zudem haben wir eine ”OMS”-Lösung (Oxygen Media Server), die es ermöglicht, die Inhalte auf den Geräten zu verwalten. Falls der Kunde also zu bestimmten Zeiten eigene Inhalte oder Inhalte von externen Anbietern, auf den Geräten, darstellen möchte, kann er dies über die ”OMS”-Lösung tun. Diese Inhalte werden dann auf den Geräten abgespielt.
Die Idee ist hier, dass der Kunde nicht an die Terminbuchungs-Webapplikation gebunden ist, sondern diese Applikation nur als zusätzliche Funktion angeboten bekommt. Viele der Softwarelösungen, die schon für genau diesen Anwendungsfall existieren, beinhalten Bindungen an andere Softwarelösungen, die der Kunde nicht benötigt. Sie machen es dem Kunden schwer, die Softwarelösung zu nutzen, da er sich mit anderen Softwarelösungen auseinandersetzen muss, die er nicht benötigt und die gegebenenfalls nicht in einem Kiosk-Modus laufen können oder parallel zu unserer restlichen Software, die der Kunde haben möchte. Außerdem erfordern einige dieser Softwarelösungen eine Installation auf dem Gerät oder erzwingen Konten-Registrierungen, die der Kunde nicht benötigt.
Die Alternativprodukte, die in den vorherigen Abschnitten beschrieben wurden, erfüllen nicht alle Anforderungen, die der Kunde an die Softwarelösung hat. Alles wird lokal auf dem Gerät abgespielt und es wird keine Installation auf dem Gerät benötigt. Es wird keine Registrierung eines Kontos benötigt und niemand kann die Daten von außerhalb, außer natürlich Microsoft, einsehen, da wirklich alles lokal gehostet wird. Die Inhalte werden dynamisch auf dem Gerät geladen und verarbeitet.

3.6 Ist-Zustand

Die Muttergesellschaft der DOOH media GmbH und einige Schwesterunternehmen nutzen Microsoft 365 und Office 365. Diese Produkte sind sehr umfangreich und bieten viele Funktionen. Die heutzutage gängige und weit verbreitete Terminplanung per Outlook oder Teams ist eines dieser Funktionen. Diese Funktion ist nicht so praktikabel wie sie es sein könnte. Vor allem nicht, wenn mehrere Unternehmen, das gleiche Gebäude und die gleiche Organisations-E-Mail besitzen. Microsoft versucht so viele Nutzer wie möglich abzudecken und bietet vorzugsweise mehr Funktionen an, als weniger. Dadurch ist die **User Journey**, also der Weg, den ein User durchläuft, um ein Ziel zu erreichen, sehr umständlich.

Falls ein User zufällig an einem Raum vorbeigeht oder vor einem Raum steht und sich fragt, ob dieser Raum belegt ist oder nicht, muss dieser erstmal Outlook oder Teams auf einem Gerät öffnen, zum Kalender des jeweiligen Raumes, falls er darauf überhaupt Zugriff hat und dann schauen, ob dieser Raum belegt ist oder nicht. Dies ist umständlich und kann viel Zeit in Anspruch nehmen.

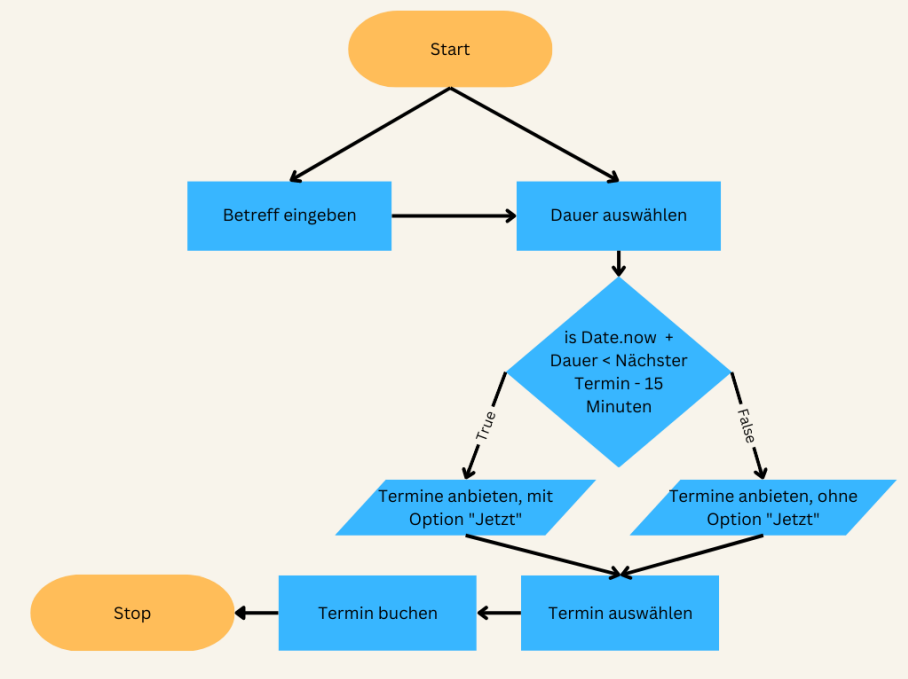
3.7 Soll-Zustand

3.7.1 Anforderungen

Ein User soll am Bildschirm eines Tablets, welches vor dem Raum angebracht wird, erkennen können, ob dieser Raum belegt ist oder nicht. Welche Termine am heutigen und nächsten Tag anstehen soll einfach ersichtlich sein. Zudem möchte der Kunde einen Termin, am Gerät, vereinbaren können. Der User soll also nicht mehr auf Outlook oder Teams angewiesen sein, sondern kann direkt am Bildschirm des Tablets sehen, ob der Raum belegt ist oder nicht. Zudem soll der Raumstatus farbig dargestellt werden, sodass der User sofort erkennen kann, ob der Raum belegt ist oder nicht, sowohl am User Interface, als auch an den LED Strips des Tablets.

3.7.2 User Interface

Das User Interface soll, so gestaltet sein, dass der User sofort erkennen kann, ob der Raum belegt ist oder nicht. Des Weiteren soll das **UserInterface** so gestaltet sein, dass der User auch Termine für den Raum vereinbaren kann. Dafür wird die Hintergrundfarbe des Containers, in dem der Raumstatus angezeigt wird, in Abhängigkeit vom Raumstatus, rot, gelb oder grün sein. Falls der Termin gerade stattfindet, soll dieser rot dargestellt werden, falls er innerhalb der nächsten 15 Minuten stattfindet, soll er gelb dargestellt werden und falls er in der Zukunft stattfindet, soll er grün dargestellt werden. Sollte am heutigen Tag kein Termin stattfinden, wird der Raumstatus weiterhin grün dargestellt, jedoch wird der Text ”Keine weiteren Termine für Heute” angezeigt. Das Userinterface soll also eine Übersicht über die Termine des Raumes und einen Button zum Vereinbaren eines Termins enthalten.
Hier ein Workflowdiagramm, welches den Ablauf der Terminbuchung beschreibt.



Die Rechtecke stellen die einzelnen Schritte des Anwenders dar. Pfeile zeigen den Ablauf des Anwenders. Entscheidungen der Software werden durch die Pfeile mit einem Pfeilspitze dargestellt. Falls diese Entscheidung zu einer Änderung der Daten führt, wird die Folge als Parallelogramm dargestellt.

Von den Grafikdesignerinnen unseres Unternehmens und der Muttergesellschaft wurde ein Design für das User Interface erstellt.

Dieses Design, ist jedoch in seiner Ausführung nicht praktikabel. Es wurden nicht die gängigen Konventionen für kontrastreiche Farben und Schriftarten berücksichtigt. Beispielsweise sind die dunkelgrauen Boxen mit einem fast schwarzen Hintergrund der Seite und einem fast weißen Text nicht gut differenzierbar. Des Weiteren wurde bei der Erstellung des Designs nicht genug auf die kleine Auflösung und Displaygröße des Tablets geachtet. Dies sind alles Eigenschaften und Variablen, die bei der Gestaltung des Userinterface noch beachten sollte.
Das Design existiert, in sechs Versionen, die sich in darin unterscheiden, ob sie gerade im ”Dark Mode” oder ”Light Mode” sind und basierend auf ihrem Belegungsstatus, mit den Farben rot, gelb und grün.

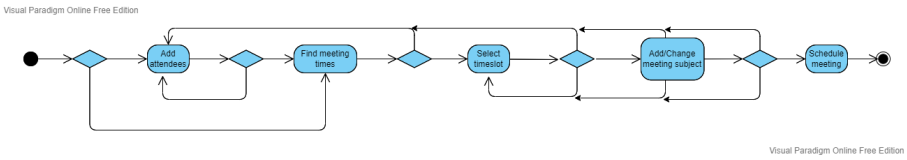
Responsive Design wurde hier nur für das Format berücksichtigt, da die Anwendung nur für Tablets und Desktops im Landscape-Modus gedacht ist.

3.7.3 LED Strips

Die **LED Strips** sollen in der Farbe leuchten, die dem Raumstatus entspricht.

3.7.4 Objektorientierte Analyse

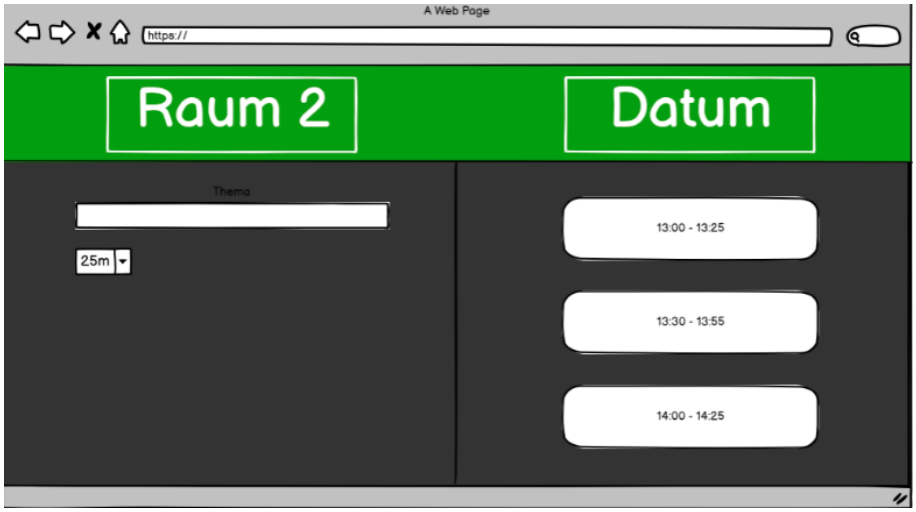
Anbei ein Aktivitätsdiagramm, welches den Ablauf der Anwendung beschreibt.



4 Vorgehensweise

4.1 Prototyp

Erst wurde ein Prototyp mithilfe von vuejs entwickelt, basierend auf folgendem Mockup, welcher erstellt wurde bevor die Grafikdesigner ein Konzept erstellt hatten:



4.2 Technische Umsetzung

4.2.1 Grobe Planung

Das Ganze wird als eine Azure App deployt. Diese lässt localhost Verbindungen zu und gibt einem die Möglichkeit das ganze als Single Page Application zu entwickeln, mit einem lokalen cookie cache für den eingeloggten Account. Somit ist keine Individualisierung notwendig.
Es wird, lokal, mithilfe von Dory-node.js gehostet. Diese lässt localhost Verbindungen zu und gibt einem die Möglichkeit die Web-Applikation wird als Single-Page-Application, mit einem lokalen cookie cache für den eingeloggten Account, entwickelt. Somit ist keine Individualisierung notwendig. Zudem kann dann mit dem Phillips Display die Web-Applikation aufgerufen werden und als eine Art Model eines Model-View-Controllers verwendet werden.
Der Player braucht auf der OMS den Content-Typen ”Web” mit der URL:
”http://localhost:3000/content”. Diese Seite wird dann lokal vom Player aufgerufen, worauf der Dory-nodejs Server antwortet und die tatsächliche Seite anbietet, die lokal auf dem Display liegt. So ist die URL durchgehend, auf allen Geräten, identisch, damit die OAuth 2.0 URI Bedingungen [OAuth-2.0-Simplified] erfüllt sind. Durch den Verzicht auf einen Webserver werden kosten eingespart, da Updates eher selten passieren sollten.
Zudem ist es für die Sicherheit der Nutzer so viel besser, da wir deshalb keinen Zugriff auf ihre Daten haben. Zeiten müssen, bei jeglichen API Anfragen an die Microsoft Graph API, ISO 8601 Konform sein . Dies ist wichtig, da Zeitzonen des Nutzers, ungleich der Zeitzonen der Server, sein können. Außerdem können Sommer- und Winterzeiten unterschiedlich sein, was zu Problemen führen kann. Es muss also bei der Datenverarbeitung darauf geachtet werden, dass die Zeiten immer ISO 8601 Konform sind, aber bei der Anzeige, kann man die Zeiten in die Zeitzone des Nutzers umrechnen. Beispielsweise, im Terminbuchungsmenü, muss dem Nutzer die Zeit angezeigt werden, die lokal für ihn gilt. Sobald dieser Termin aber gebucht wird, muss die Zeit in die Zeitzone des Servers umgerechnet werden, damit die Buchung auch dort korrekt verarbeitet wird.

4.2.2 Benutzte Hardware und Software

Hardware:

- Philips 10BDL4551T/00 (Firmware: FB01.13, SICP Version: v.2.05)

Die Hardware ist ein Philips [10BDL4551T/00] Display. Dieses Display ist ein 10 Zoll (ca. 25 cm) Touchscreen. Es hat eine Auflösung von 1280x800 Pixeln und benutzt Android. Dieses Display ist für Digital Signage gedacht und kann so auch als solches verwendet werden. Die eingebauten RGB LED Strips können per **SICP** angesteuert werden. Dafür braucht wird Dory-Node.js genutzt, um auf einen lokalen Server zu hören und die Anfragen an das Display weiterzuleiten. Es wird dafür keine Internetverbindung benötigt, da kein Websocket benötigt wird. Die Details der SICP Daten sind öffentlich schwierig zu finden, aber im Internet sind einige Beispiele zu finden. Befehle werden in Hexadezimal geschrieben und an das Display gesendet. Die Verschlüsselung passiert, indem ein XOR aller vorausgehenden Bytes genommen und als checksum mitgegeben wird.

Software:

- Vue.js 3.x
- Microsoft Graph API per NPM Package
- Dory-Node.js
- Azure App
- Babel
- Webpack
- Node.js
- NPM
- Git
- EsLint
- IntelliJ IDEA

Vue.js ist ein JavaScript Framework und transpiliert die Vue.js Dateien in JavaScript, damit sie von jedem Browser ausgeführt werden können. Es ist darauf spezialisiert, Single Page Applikationen zu entwickeln.

Dory-Node.js ist ein Node.js Server, der es ermöglicht, eine Web-Applikation lokal zu hosten. Dies wird benötigt, um einerseits die Raumbuchungsseite lokal anzubieten, damit die absoluten Redirect-URI Bedingungen von Microsoft und oAuth 2.0 erfüllt werden können und andererseits, um die Anwendung auf dem Display zu hosten und als Schnittstelle zwischen der Seite und dem den LED-Strips des Displays zu agieren.

Die Microsoft Graph API wird per NPM Package verwendet, um die Anfragen an die Microsoft Graph API zu vereinfachen und den User einzuloggen.

Babel ist ein JavaScript Compiler, der es ermöglicht, moderne JavaScript Features zu verwenden, die von älteren Browsern nicht unterstützt werden, um so die Kompatibilität zu erhöhen.

Webpack ist ein JavaScript Bundler, der es ermöglicht, mehrere JavaScript Dateien zu einer einzigen zusammenzufassen, um so die Ladezeiten zu verkürzen.

Node.js ist ein JavaScript Runtime, der es ermöglicht, JavaScript Code auszuführen, ohne einen Browser zu benötigen. Es ist das sogenannte Backend dieser Anwendung. Dieser wird per Dory-Node.js gehostet.

NPM ist ein Package Manager, der es ermöglicht, JavaScript Packages zu installieren und zu verwalten.

Git ist ein Versionskontrollsystem, welches es ermöglicht, Änderungen an Dateien zu verfolgen und zu verwalten. Dies wurde aber in Zusammenhang mit GitLab verwendet, da dies der derzeitige, etablierte Standard der DooH media GmbH ist.

EsLint ist ein Linter, der es ermöglicht, JavaScript Code zu analysieren und zu formatieren. Da Testfälle bei JavaScript Projekten nicht immer vollständig möglich sind, ist es wichtig, dass der Code einheitlich ist und keine Fehler enthält. Deshalb achtet EsLint auf die Einhaltung von Regeln, damit erst gar keine Fehler entstehen, die der JavaScript Compiler nicht erkennen kann. Beispielsweise wird überprüft, ob Variablen deklariert wurden, bevor sie verwendet werden und anders herum.

IntelliJ IDEA ist eine IDE, die es unter anderem ermöglicht, JavaScript Code zu schreiben und zu debuggen.

4.2.3 Weiterentwicklung der groben Planung

Es wurde eine Azure App erstellt, die die Authentifizierung der Anwendung und des Users übernimmt.
Dort wird der User eingeloggt und die Anwendung leitet ihn auf die Seite weiter, die er vorher besucht hat, welche in diesem Fall, die Raumbuchungsseite ist.
Es wurde jede Woche Rücksprache mit dem Kunden gehalten, um die Anforderungen zu besprechen und zu erfüllen. Aber auch intern wurde Rücksprache gehalten, was denn sinnvoll ist und was nicht.
So wurde jede Woche die Anwendung weiterentwickelt und verbessert. Teilweise wurden auch neue Features hinzugefügt, die nicht im Pflichtenheft standen, aber sinnvoll waren. Manche Features wurden auch wieder entfernt, da sie nicht sinnvoll waren oder von anderen Features abgedeckt wurden.

4.2.4 Ablauf der Entwicklung

Nachdem der erste Prototyp fertig war und Rücksprache gehalten wurde, wurde angefangen die Anwendung zu entwickeln. Design und Funktionalität wurden dabei partiell parallel entwickelt, wobei die Funktionalität immer Priorität hatte. Über 95 Commits wurden die Änderungen an der Anwendung festgehalten.
Es wurde ein Testgerät benötigt, um die Anwendung zu testen. Ein Phillips 10BDL4551T/00 Display wurde dafür an die Wand gehängt und mit dem Internet verbunden. Zudem wurde es so eingerichtet, wie es auch beim Kunden laufen soll.
Es wurde täglich aktiv genutzt, um so Fehler zu finden und Feedback zu geben. Das Feedback wurde dann, falls sinnvoll, in die Anwendung eingearbeitet. Solche praktischen Tests sind sehr wichtig, um Intuitivität und Benutzerfreundlichkeit zu gewährleisten. Es war einerseits hilfreich, um vorgesehene Abläufe zu testen, andererseits aber auch um Fehler zu finden, die nicht

vorgesehen waren, indem Monkey Testing [Ofe10] betrieben wird.

4.2.5 Rest-Anfragen

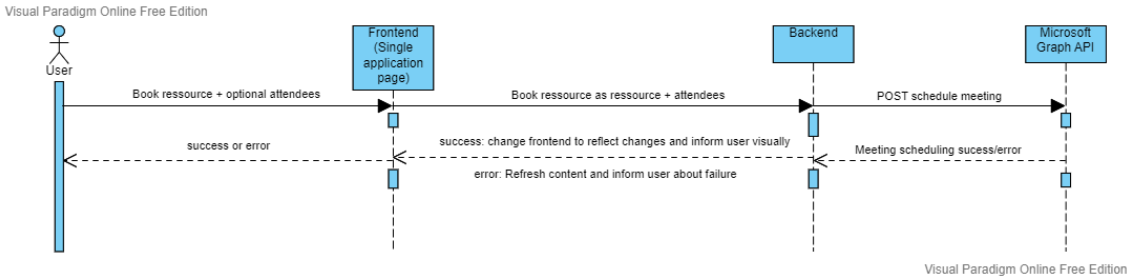
Mithilfe der oDatav4 ([oData]) API wurden die Daten aus der jeweiligen Microsoft Datenbank im Voraus gefiltert, um so nicht notwendige Daten zu vermeiden und Performance drastisch zu erhöhen, als auch Datenvolumen zu sparen. Die Daten wurden dabei in JSON Format zurückgegeben. Die oData v4 API verhält sich dabei wie eine SQL Datenbank, wobei die Daten in Tabellen gespeichert sind. Es wurden die Klauseln ”$filter” und ”$top” verwendet, um nur Termine für den aktuellen Tag und nächsten Tag zu erhalten und dies einzuschränken auf die top 300 Termine, falls jemand versucht das System zu überlasten. Eine Sortierung ist nicht notwendig, da die Termine bereits nach Startzeit sortiert sind. Hier sieht man die oData v4 API Anfrage:

```
1       let url =  "https://graph.microsoft.com/v1.0/me/findMeetingTimes/?
            $filter=start/dateTime" +  "ge"  + "${todayDate} and end/dateTime
            le ${tomorrowDate}&$top=300";
```

Es wurden im Voraus, Befehle im Microsoft Graph Explorer [Microsoft-Graph-Explorer] ausgeführt, um zu testen, ob die Anfrage funktioniert und ob die Daten in der gewünschten Form zurückgegeben werden. Jedoch hat sich herausgestellt, dass der Microsoft Graph Explorer nicht korrekt funktioniert und teilweise Befehle nicht entgegennimmt, die laut Microsoft Dokumentation funktionieren sollten. Evident ist dies, da die Anwendung korrekt funktioniert und die REST API Anfragen korrekt funktionieren und auch im Request-Response Zyklus keine Fehler auftreten.

Falls die Anfrage nicht funktioniert, wird eine Fehlermeldung angezeigt, die den User darauf hinweist, dass die Verbindung zum Server derzeit unterbrochen ist. Wie Fehlermeldungen behandelt werden sollten, findet man im folgenden Buch: [interaction-design-book1]. Fehlermeldungen sollten immer so gestaltet sein, dass sie den User nicht verunsichern, sondern ihn auf die Lösung des Problems hinweisen. Auch bei der Gestaltung der Fehlermeldung sollte darauf geachtet werden, dass sie nicht zu lang ist, da der User sonst nicht mehr weiß, was er tun soll. Bei Kiosk Anwendungen ist es besonders wichtig, dass die Fehlermeldungen kurz und prägnant sind, da der User ohnehin in einer Situation ist, in der er wahrscheinlich nicht viel Zeit hat. Genaueres dazu, befindet sich, im genannten Buch, unter dem Kapitel *Ërrors, Alerts, and Confirmation*” und unter dem Kapitel ”*Platform and Posture – Designing for kiosks*”. Das bedeutet, dass Fehler, die dazu führen, dass die Anwendung nicht mehr funktioniert, groß und auffällig dargestellt werden sollten, damit der User sie nicht übersehen kann und versteht, dass er die Anwendung nicht mehr nutzen kann, bis das Problem behoben wurde. Andererseits sollten Fehler, die nicht dazu führen, dass die Anwendung nicht mehr funktioniert, klein und unauffällig dargestellt werden, damit der User nicht gestört wird. Es liegt nicht in der Verantwortung des Nutzers, dass die Anwendung nicht funktioniert, sondern in der Verantwortung des Entwicklers, solange es um die lokale Verarbeitung geht.

Hier sieht man ein Sequenzdiagramm, welches die Kommunikation zwischen der Anwender und dem Server darstellt:



4.2.6 Timer

Der Timer, welcher die übrig bleibende Zeit bis zum nächsten Termin anzeigt funktioniert, wie folgt:

Es wird die Zeit bis zum Ende den jetzigen Termin berechnet, ausgehend von der Zeit, die zum Anfang des jetzigen Termins vergangen ist, und in Millisekunden umgerechnet. Dann wird eine sich drehende Animation erstellt, die für diese Dauer abläuft.
Da die Dauer angepasst werden kann, muss bei Datenänderungen die Animation neu berechnet werden. Damit dies nicht ohne Grund passiert, wird geprüft, ob die Dauer sich geändert hat. Die Daten dafür werden im Local Storage gespeichert. Falls eine Änderung stattgefunden hat, wird die Animation neu berechnet. Bei der Änderung werden, damit die Animation visuell nicht von vorne anfängt, die Animationsdauer und die vergangene Zeit als neue Animationsdauer aufaddiert und die vergangene Zeit als negative Animationsverzögerung gesetzt, damit die Animation visuell betrachtet dort ist, wo sie relativ zur neuen Dauer sein sollte.

Die Formel lautet, wie folgt:

$$\begin{aligned} \text{Vergangene Zeit} &= \text{Jetzt} - \text{Start} \\ \text{Animationsdauer} &= \text{Ende} - \text{Start} \\ \text{Animationsverzögerung} &= \text{-Vergangene Zeit} \end{aligned} \tag{1}$$

Man kann die Formel verkürzen und die Animationsdauer und -verzögerung direkt berechnen, aber es ist verständlicher, falls jemand in der Zukunft dies nachvollziehen möchte.

Hier sieht man den zugehörigen JavaScript Code:

```
1 let currentEventBeginningTime = localStorage.getItem('
    currentEventBeginningTime');
2 let timePassed = (new Date() - new Date(currentEventBeginningTime + 'Z')) /
     1000;
3 let animationDuration = ((new Date(currentEventEndTime + 'Z') - new Date(
    currentEventBeginningTime + 'Z')) / 1000);
4 firstHandSpan.style.animationDuration = animationDuration  + 's';
5 secondHandSpan.style.animationDuration = animationDuration + 's';
6 firstHandSpan.style.animationDelay = -timePassed + 's';
7 secondHandSpan.style.animationDelay = -timePassed  + 's';
```

Um die Animation während ihrer Laufzeit zu ändern, wird ein sogenannter Reflow[1] erzwungen, indem die offsetWidth Eigenschaft abgefragt, die Animationsdauer und -verzögerung neu gesetzt

[1] Reflow.

und der Animationsname erst entfernt und dann wieder hinzugefügt wird.

4.2.7 Persistente Datenspeicherung

Um die Daten der Anwendung zu speichern, wurde für kleinere Datensätze der Local Storage verwendet. Dieser ist in jedem Browser verfügbar und darf bis zu 5 MB groß sein.

Für größere Datensätze wurde IndexedDB [IndexedDB] verwendet. Dies ist eine NoSQL Datenbank, die in jedem gängigen Browser [caniuse-indexedDB] (**caniuse.com**) verfügbar ist. Die maximale Größe bei Chrome beträgt 80% des verfügbaren Speichers. Da es im Internet viele, sich widersprechende Angaben zu dieser Größe gibt, wurde diese manuell, mit folgendem Code, in der Konsole des Browsers, getestet:

```
1    if (navigator.storage && navigator.storage.estimate) {
2    const quota = await navigator.storage.estimate();
3    // quota.usage -> Number of bytes used.
4    // quota.quota -> Maximum number of bytes available.
5    const percentageUsed = (quota.usage / quota.quota) * 100;
6    console.log("You've used ${percentageUsed}% of the available storage.")
         ;
7    const remaining = quota.quota - quota.usage;
8    console.log("You can write up to ${remaining} more bytes.");
9    }
```

Die Festplatte, auf der dies getestet wurde, hatte ca. 828 GB freien Speicher. Das Ergebnis lautete wie folgt:

```
1    You've used 0.000002393592594674544% of the available storage.
2    You can write up to 599726105785 more bytes.
```

$$\text{verbleibender Speicher} = 599726105785 \text{ Bytes}$$
$$599726105785 \text{ Bytes} * 10^{-9} = 599{,}726105785 \text{ Gigabytes} \quad (2)$$

Das sind etwas über 70 % der verfügbaren Speicherkapazität. Hier ist davon auszugehen, dass die restlichen 10 % für andere Daten schon verwendet werden. Zudem ist zu betrachten, dass die ¨.¨ Schreibweise für Dezimalzahlen in JavaScript verwendet wurde, während wiederum die ¨,¨ Schreibweise für die mathematische Gleichung verwendet wurde. Dies hat den Hintergrund, dass die ¨,¨ Schreibweise in JavaScript nicht grundsätzlich unterstützt wird, beziehungsweise, die Konvention beim Programmieren verlangt, dass man auf Englisch programmiert und im Englischen die Dezimalzahlen mit einem Punkt geschrieben werden.

Die Datenbank befindet sich aufgrund ihrer geringen Komplexität in der 5. Normalform. Der Schlüssel besteht aus dem Namen der Firma, zu der, das Logo gehört und gibt das nicht Primärattribut zurück, welches das Bild, in Base64, enthält. Schlüssel müssen eindeutig sein. IndexedDB unterstützt die Eigenschaft ¨unique¨, welche dafür sorgt, dass der Schlüssel eindeutig ist. Jedoch gehen wir auch manuell noch einmal sicher, dass der Schlüssel eindeutig ist, also dass sich dieser, noch nicht in der Datenbank befindet. Eine weitere Aufteilung würde zu Informationsverlust führen, da der Firmenname eindeutig ist.

Zwar gibt es die Möglichkeit die Bilder für Firmen, zu updaten, welcher der User auch so wahrnimmt, jedoch wird der Schlüssel des alten Bildes entfernt und ein neuer Schlüssel mit dem neuen Bild hinzugefügt. Dies ist notwendig, da der Schlüssel eindeutig sein muss. Somit löschen wir den alten Eintrag und fügen einen neuen hinzu. Unabhängig von der Interaktion mit der Datenbank, wird sichergestellt, dass sinnfreie Anfragen an die Datenbank nicht gesendet werden. Falls der User beispielsweise ein Bild hochlädt, welches bereits in der Datenbank, für genau den gleichen Schlüssel, vorhanden ist, wird die Anfrage an die Datenbank nicht gesendet oder falls der User versucht ein Bild aus der Datenbank zu entfernen, welches gar nicht existiert.

4.2.8 Azure Authentifizierung

Um die Authentifizierung zu realisieren, wurden folgende Berechtigungen benötigt:

Calendars.Read	**Delegiert**	**Lesezugriff auf Benutzerkalender**
Calendars.Read.Shared	Delegiert	Benutzer und freigegebene Kalender lesen
Calendars.ReadWrite	Delegiert	Verfügt über Vollzugriff auf Benutzerkalender.
Calendars.ReadWrite.Shared	Delegiert	Benutzerdefinierte und freigegebene Kalender lesen und schreiben
email	Delegiert	E-Mail-Adresse von Benutzern anzeigen
IMAP.AccessAsUser.All	Delegiert	Read and write access to mailboxes via IMAP.
Mail.Read	Delegiert	Lesezugriff auf Benutzer-E-Mails
Mail.Send	Delegiert	E-Mails unter einem anderen Benutzernamen senden
Mail.Send.Shared	Delegiert	E-Mails im Namen anderer Benutzer senden
offline_access	Delegiert	Zugriff auf Daten beibehalten, für die Sie Zugriff erteilt haben
OnlineMeetings.Read	Delegiert	Read user's online meetings
OnlineMeetings.ReadWrite	Delegiert	Read and create user's online meetings
openid	Delegiert	Benutzer anmelden
profile	Delegiert	Grundlegendes Profil von Benutzern anzeigen
User.Read	Delegiert	Anmelden und Benutzerprofil lesen

Diese werden beim erstmaligen Login, vom User, seitens Microsoft, angefordert. Ohne die Zustimmung des Users, wird die Anwendung nicht entsprechend den Anforderungen, funktionieren und wird weiterhin darauf hinweisen, dass der User sich anmelden soll.

5 Ergebnis

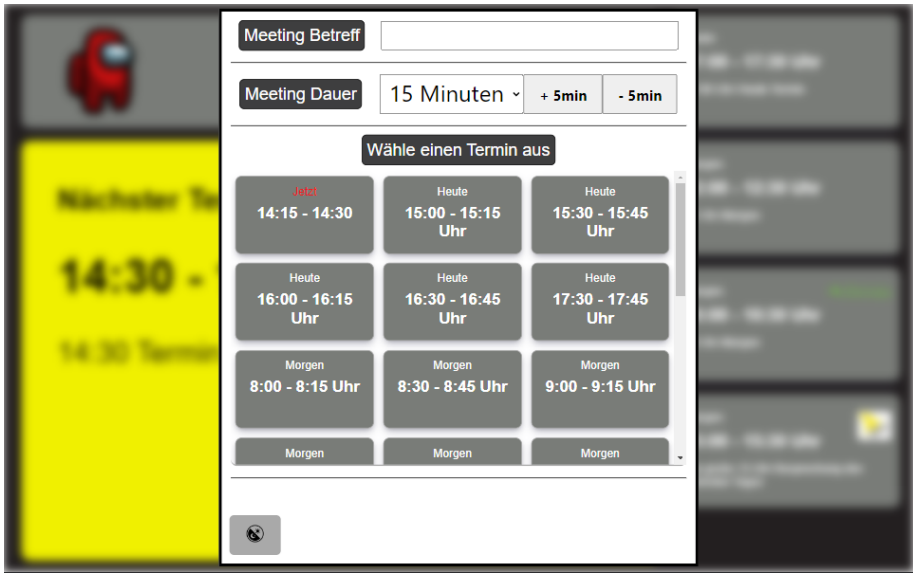
Das Ergebnis sieht wie folgt aus:

5.1 Visuelle Darstellung

Oben links im Bild, das Bild des kleinen roten Charakters, ist das Logo des Gastgebers des nächsten, beziehungsweise jetzigen, Termins zu sehen. Solch ein Logo kann dargestellt werden, indem beim Erstellen des Termins, außerhalb des Tablets, bei Outlook beispielsweise, ein Bild hochgeladen wird, welches im Betreff eine bestimmte Bezeichnung enthält, die hier aus Sicherheitsgründen nicht genannt werden kann.
Die anderen Logos sind alle Logos vom Gast des Termins. Diese Logos können an alle Geräte gleichzeitig versendet werden, indem ein spezieller Termin erstellt wird, der nur für die Logos

gedacht ist und eine einzigarte ID, sowie Befehle enthält, die dann das Bild, inklusive Firmennamen, in einer lokalen Datenbank abspeichert. Diese Logos können hinzugefügt, gelöscht oder aktualisiert werden. Aus Sicherheitsgründen werden die genaueren Befehle der Schnittstelle hier nicht genannt.
Es wird nur der erste Gast angezeigt, da das so vom Kunden gewünscht wurde.
Die Uhrzeit wird, immer, in der Zeitzone des Tablets angezeigt.

Hier sieht man nochmal das Menü, wo ein Termin gebucht werden kann, welches durch das Drücken des Plus-Symbols aufgerufen wird:



Wie zu sehen ist, ist beispielsweise die selbsterstellte Option ”Jetzt” deaktiviert, da das Ende des hypothetischen Termins innerhalb von 15 Minuten vom nächsten anstehenden Termin beginnt. Diese Pufferzeit ist so mit dem Kunden abgesprochen. Es werden alle möglichen Termine, innerhalb der Arbeitszeiten, für den Jetzigen und nächsten Tag angezeigt. Falls z.B. der nächste Tag ein Feiertag ist, werden für den nächsten Tag keine Termine angezeigt. Auf die anderen Optionen, wie z.B. Pufferzeiten, hat ein Anwender hier wenig Einfluss, da sie von den Einstellungen des Kalenders, des jeweiligen Nutzers, abhängen und es somit in der Verantwortung des Administrators liegt, diese zu ändern.

Hier die normale Ansicht nochmal, aber im light-Modus:

5.2 Performance

Die Performance einer **SPA** zu messen ist trotz der geringen Komplexität der Anwendung, nicht simpel. Auch die gängige Total-Blocking-Time Messung ist nicht zwingend sinnvoll, da die Anwendung nicht für den Nutzer, während der Nutzung, blockiert ist, sondern lediglich die Daten vom Server geladen werden, um anschließend die Nutzung der Anwendung sinnvoll zu ermöglichen. Daher wurden manuell einige Tests durchgeführt, um die Performance zu messen.

5.2.1 Test 1

Wie lange braucht die Anwendung, um nach einem Klick auf den Button ”+” das Terminerstellungs-Menü zu öffnen? name=Event-Listener,description=Ein Event-Listener ist ein Objekt, welches auf bestimmte Ereignisse wartet und dann eine Funktion ausführt. Dieses Objekt braucht ein HTML Element, welches es überwachen soll. Die Antwort auf diese Fragestellung wurde gemessen, indem der Event-Listener für den Button ”+” registriert ausgab, wann die Taste betätigt und damit Zeit gemessen wurde, die vergeht, bis der das Menü darstellt. Es dauert im Durchschnitt 2ms, bis das Menü angezeigt wird.

5.2.2 Test 2

Wie viele Bilder pro Sekunde zeigt die Anwendung durchschnittlich an? Dies wurde mithilfe von LightHouse gemessen. Die Anwendung zeigt durchschnittlich 60 Bilder pro Sekunde an.

5.2.3 Test 3

Wie oft, kann die Anwendung theoretisch und praktisch pro Sekunde aktualisiert werden? Für jede Kombination aus Azure App und E-Mail-Adresse, dürfen 10000 Anfragen pro 10 Minuten gemacht werden. Dies sind 16,6 Anfragen pro Sekunde. Praktisch wurden eine bis drei Anfragen, je nach Bedarf, pro drei Sekunden gemacht. Pro Sekunde wären das also 0,33 bis 1 Anfragen.

Einerseits ist das schnell genug für die Anwendung und gibt den langsamen Geräten, genug Zeit, die Anwendung zu aktualisieren, andererseits hat man so genug Anfragen übrig, falls jemand sein Konto mit mehreren Geräten benutzt oder dies in Kombinationen mit anderen Applikationen verwendet. Sollte dieses Limit überschritten werden, verlangsamt Microsoft die Anzahl an neuen Antworten auf die Anfragen. Da Microsoft auch Zeit benötigt, um die Anfragen zu bearbeiten und die neuen Daten bei sich zu verarbeiten, würde der Anwender den Unterschied selten bemerken. Der **Bottleneck** der Anwendung ist also in diesem Fall die Anzahl an Anfragen, die Microsoft, in ihren Servern, verarbeiten kann oder will.

5.3 Fazit

Die Microsoft Graph API ist eine sehr mächtige Schnittstelle, die es ermöglicht, mit einer Vielzahl an Microsoft Diensten zu interagieren. Für die Anwendung wurde die Schnittstelle genutzt, um Termine zu erstellen, zu löschen und zu aktualisieren. Außerdem wurde die Schnittstelle genutzt, um die Logos der anderen Termine zu erhalten. Die Schnittstelle wurde nicht vollständig genutzt, da die Anwendung nicht alle Funktionen benötigt, aber bietet für die Zukunft viele Erweiterungsmöglichkeiten. Zudem ist die API recht neu und wird ständig weiterentwickelt [microsoft-graph-api-version]. Es Anwendung wurde mit der Version 1.0 der API entwickelt. Für den Anwendungsfall des Kunden ist die API ausreichend, da die Anwendung nur Termine erstellen, löschen und aktualisieren muss. Der einzige Nachteil ist, dass Azure AD, die Authentifizierung, für Ressourcenkonten nicht unterstützt, die beispielsweise, in Teams, genutzt werden. Daher muss der Anwender sich mit einem Microsoft-Konto anmelden, welcher die Ressource repräsentieren soll. Dies kann zu Verwirrung führen, da die Terminologie impliziert, man könne sich mit einem Ressourcenkonto anmelden.

Trotzdem ist die Anwendung eine gute Basis für weitere Erweiterungen und war die richtige Wahl für diesen Kundenauftrag. Ressourcen- und Terminplanung wurde mit der Anwendung vereinfacht, für Office365 Nutzer, definitiv vereinfacht. Ein Anwender braucht nur noch drei Klicks, um einen Termin zu erstellen und muss nicht mehr zwischen verschiedenen Kalendern hin und her wechseln. Zudem benötigt er nur ein Mal auf den Bildschirm zu schauen, um zu sehen, wann die Ressource, die durch die Software repräsentiert wird, frei ist und wann der Anwender gegebenenfalls einen Termin besitzt (Anhand der Logos).

Die Anwendung kann auch für menschliche Ressourcen genutzt werden, da die Anwendung keine Unterscheidung zwischen menschlichen und nicht-menschlichen Ressourcen macht. Solche Entscheidungen obliegen dem Anwender.

Zusammenfassend kann gesagt werden, dass die Anwendung eine gute Basis für weitere Erweiterungen ist und die Anforderungen des Kunden, so erfüllt. Damit geht einher, dass die Ziele der Praxisarbeit erreicht wurden.

Abschließend folgt noch ein Bild des Gerätes mit der Anwendung, welches im Kundenbetrieb genutzt wird.

6 Literaturverzeichnis

Literatur

[10BDL4551T/00] Phillips10BDL4551T/00. *Phillips10BDL4551T/00*. Sep. 2022. URL: https://www.download.p4c.philips.com/files/1/10bdl4551t_00/10bdl4551t_00_pss_enggb.pdf?_ga=2.87916248.193099575.1678716502-1541604121.1678716502.

[caniuse-indexedDB] caniuse-indexedDB. *caniuse-indexedDB*. 2023. URL: https://caniuse.com/indexeddb.

[IndexedDB] IndexedDB. *IndexedDB*. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API.

[interaction-design-book1] David Cronin Alan Cooper Robert Reimann. *About Face: The Essentials of Interaction Design*. Mai 2007.

[JavaScript] Mozilla. *JavaScript*. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[microsoft-graph-api-termin] microsoft-graph-api-termin. *microsoft-graph-api-termin*. Feb. 2022. URL: https://docs.microsoft.com/en-us/graph/api/resources/event?view=graph-rest-1.0.

[microsoft-graph-api-termin-aktualisieren] microsoft-graph-api-termin-aktualisieren. *microsoft-graph-api-termin-aktualisieren*. Feb. 2022. URL: https://docs.microsoft.com/en-us/graph/api/event-update?view=graph-rest-1.0&tabs=http.

[microsoft-graph-api-termin-bearbeiten] microsoft-graph-api-termin-bearbeiten. *microsoft-graph-api-termin-bearbeiten*. Feb. 2022. URL: https://docs.microsoft.com/en-us/graph/api/event-update?view=graph-rest-1.0&tabs=http.

[microsoft-graph-api-termin-erstellen] microsoft-graph-api-termin-erstellen. *microsoft-graph-api-termin-erstellen*. Feb. 2022. URL: https://docs.microsoft.com/en-us/graph/api/event-post-events?view=graph-rest-1.0&tabs=http.

[microsoft-graph-api-termin-loeschen] microsoft-graph-api-termin-loeschen. *microsoft-graph-api-termin-loeschen*. Feb. 2022. URL: https://docs.microsoft.com/en-us/graph/api/event-delete?view=graph-rest-1.0&tabs=http.

[microsoft-graph-api-version] microsoft-graph-api-version. *microsoft-graph-api-version*. Feb. 2022. URL: https://docs.microsoft.com/en-us/graph/overview?view=graph-rest-1.0.

[Microsoft-Graph-Explorer] Microsoft-Graph-Explorer. *Microsoft-Graph-Explorer*. 2023. URL: https://developer.microsoft.com/en-us/graph/graph-explorer.

[microsoftGraphApi] microsoftGraphApi. *microsoftGraphApi*. 2023. URL: https://docs.microsoft.com/en-us/graph/overview.

[NodeJS] NodeJS. *NodeJS*. 2023. URL: https://nodejs.org/en/.

[NPM] NPM. *NPM*. 2023. URL: https://www.npmjs.com/.

[oAuth] oAuth. *oAuth*. 2023. URL: https://oauth.net/2/.

[OAuth-2.0-Simplified] Aaron Parecki. *OAuth 2.0 Simplified*. Mai 2018.

[OAuth2.0$_R$*edirect*$_U$*ri*] OAuth2.0$_R$*edirect*$_U$*ri*. *OAuth2.0*$_R$*edirect*$_U$*ri*. 2023. URL: https://www.oauth.com/oauth2-servers/redirect-uris/.

[oData] oData. *oData*. 2023. URL: https://www.odata.org/.

[Ofe10] Stefan Szeider Ofer Strichman. *Theory and Applications of Satisfiability Testing - SAT 2010*. Springer-Verlag Berlin Heidelberg, Okt. 2010.

[Reflow] JavaScriptAnimationsReflow. *JavaScriptAnimationsReflow*. 2023. URL: https://www.html5rocks.com/en/tutorials/speed/animations/.

[REST] REST. *REST*. 2023. URL: https://restfulapi.net/.

[vuejs] vuejs. *vuejs*. 2023. URL: https://vuejs.org/.

7 Glossar

Bottleneck Der Bottleneck ist, in der Softwareentwicklung, die Stelle, an der die Leistung der gesamten Anwendung am meisten eingeschränkt wird.. 23

caniuse.com Caniuse is a website that shows you browser support for various features and includes references to the relevant specifications.. 18

Kiosk-Modus Der Kiosk-Modus ist ein Modus, in dem ein Gerät, wie beispielsweise ein Tablet, nur eine Applikation ausführen kann.. 9

LED Strips LED Strips sind eine Art von LED Leuchtmittel. LED Strips sind in der Regel lang und dünn und werden in Streifen angebracht.. 12

Responsive Design Responsive Design ist ein Begriff aus der User Experience Design. Responsive Design ist ein Design, das sich an die Größe des Bildschirms anpasst.. 12

REST API REST API ist eine Abkürzung für Representational State Transfer Application Programming Interface. REST ist ein Architekturstil, der die Kommunikation zwischen verschiedenen Systemen ermöglicht. REST ist ein Architekturstil, der die Kommunikation zwischen verschiedenen Systemen ermöglicht.. 6

RESTful RESTful ist ein Synonym für Representational State Transfer. RESTful ist ein Architekturstil für die Entwicklung von Webdiensten.. 7

SICP SICP steht für Serial (Ethernet) Interface Communication Protocol. SICP ist ein Protokoll, das es ermöglicht, mit dem Display, an der OS vorbei, zu kommunizieren.. 14

Single Page Application (SPA) Eine Single Page Application (SPA) ist eine Webanwendung, die nur eine HTML-Seite besitzt. Diese Seite wird beim Laden der Anwendung geladen und bleibt während der gesamten Nutzung der Anwendung bestehen.. 3, 22

Total-Blocking-Time (TBT) Die Total-Blocking-Time (TBT) ist eine Metrik, die die Gesamtzeit misst, die eine Seite blockiert wird, bis sie interaktiv ist.. 3

User Journey User Journey ist ein Begriff aus der User Experience Design. User Journey ist ein Weg, den ein User durchläuft, um ein Ziel zu erreichen.. 10

UserInterface UserInterface ist ein Begriff aus der User Experience Design. UserInterface ist die grafische Oberfläche, die ein User sieht und mit der er interagiert.. 11

8 Anhang

8.1 Spezifikationsblatt

Spezifikationsblatt - Raumbuchungsanzeige

Contents

1. Übersicht: Lokal gehostete Single-Page-Anwendung für Raumbuchungen. Mit Azure-Integration für die Anmeldung von Benutzern.
2. Benutzeranmeldung: Integration von Azure zur Anmeldung von Benutzern.
3. Funktionalität:

- Raumbuchungsanzeige für angemeldete Benutzer.
- Nach Anmeldung muss maximal 3-6 Sekunden gewartet werden, ohne mit dem Gerät zu interagieren oder die Seite muss manuell neu geladen werden
- Anzeige von Terminen für den aktuellen Tag und nächsten Tag
- Buchung von Terminen.
- Informationen aktualisieren alle 3 Sekunden
- Farbliche Kennzeichnung vom nächsten Termin bzw. dem aktuell laufenden:
    - Grün für Termine, die 15 Minuten oder mehr entfernt sind.

    - Gelb für Termine, die weniger als 15 Minuten entfernt sind.
    - Rot für laufende Termine.
- LED-Strip-Anzeige für farbliche Kennzeichnung von Terminen.
- Für Querformat vorgesehen
- Bei langem Klicken auf einen „kleinen“ Termin, werden die Details des Termins dargestellt. Der Terminkörpertext wird jedoch aus Sicherheitsgründen nicht dargestellt

4. Technische Anforderungen:

- Unterstützte Browser: modernste Versionen von Chrome, Firefox, Edge und Safari.
- Lokale Hosting-Umgebung erforderlich.
- Internetverbindung
- Microsoft Arbeits- oder Schulkonto erforderlich (Ressourcen-Konten sind derzeit nicht unterstützt)
- Automatisches Akzeptieren von Meetings unterstützt Microsoft nur für Exchange/IMAP/POP3. Falls das Konto lediglich IMAP/POP3 ist, ist diese Funktionalität nicht gegeben. Buchungen lokal auf dem Gerät sind weiterhin möglich. Die Einrichtung hierfür erfolgt Seitens der Nutzerorganisation.
- Für die LED Strip Kontrolle ist ein 10BDL4551T vorgesehen. Andere Modelle können funktionieren, falls sie das gleiche SICP Protokoll unterstützen.
- Die Navigationskontrolle des Displays sollte ausgeschaltet werden, damit diese bei Texteingaben nicht permanent danach die Seite überdecken. Es sollte vorher Teamviewer o.ä. eingerichtet werden, um die Displays weiterhin kontrollieren zu können, falls notwendig.
- SICP muss bei dem Display angeschaltet sein und auf localhost:5000 hören
- Es benötigt zudem eine Applikation, die nodeJs code lokal hosten kann, wie z.B. „Dory-node.js“. Folgende Version wird benötigt: https://dorynode.firebaseapp.com/v10.15.1_arm_release/node

- Bei dory-node.js auf download file -> url eingeben -> appfiles anklicken -> executable auch und dann ok.

- Das Dory-node.js Script sollte auf automatisches Starten eingestellt werden.
- Der nodeJs Ordner mit dem nodeJs code für die hostende Applikation (in diesem Fall, Dory-node.js) muss in einem Verzeichnis abgespeichert sein, wo auch der Ordner „bookingPage“ existiert. In bookingPage muss die Buchungsseite hinterlegt sein und muss „index.html“ heißen. Es ist egal wo diese zwei Ordner abgespeichert werden, solange sie relativ zu einander, sich im gleichen Verzeichnis befinden.

5. Logos darstellen:

- Unterstützt werden PNGs, JPEGs (JPGs sind identisch), WebPs GIFs und SVGs (svg+xml)
- Um ein Gastgeber-Logo darzustellen, sollte ein Bild angehangen werden, welches „Termin_Logo“ im Namen beinhaltet. Groß- und Kleinschreibung sind nicht wichtig. Es kann nur ein Gastgeber Logo angezeigt werden und nur das erste welches auftaucht. Andere Fotos sind beeinflussen diesen Vorgang nicht. Diese können weiterhin normal angehangen werden.
  Dieses Gastgeber-Logo taucht nur auf, falls der Termin, den es betrifft, der Nächste bzw. Jetzige ist.
- Gästelogos werden durch den Termintextkörper identifiziert (damit ist nicht der Betreff gemeint). Eine Firma muss lediglich namentlich erwähnt werden und dann taucht die Firma als Gast auf. Es wird nur der erste Gast angezeigt.
  Um den Geräten mitzuteilen, wie das Logo eines Gasts aussieht, wird ein Termin erstellt und die Geräte zum Termin hinzugefügt, die diese Information erfahren sollen.
  Dieser Termin benötigt ein angehangenes Bild, welches das Logo darstellen soll und einen spezifischen Betreff. Der Betreff setzt sich wie folgt zusammen:
  uniqueKey + ADD/REMOVE/UPDATE + “Firmenname zum

Identifizieren für die Logodarstellung“
Ein Beispiel könnte wie folgt aussehen:
uniqueKey *ADD “Dooh“*
Die Groß- und Kleinschreibung von ADD/REMOVE/UPDATE und dem Firmennamen sind nicht wichtig. Der Schlüssel jedoch ist aus Sicherheitsgründen genau so festgelegt. Dieser Schlüssel sollte nur vertraulichen Personen weitergegeben werden.

6. Wartung und Support: Regelmäßige Überprüfung und Wartung, um die reibungslose Funktion der Anwendung zu gewährleisten. Kundensupport steht zur Verfügung, um bei Problemen oder Fragen behilflich zu sein.