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. Die Arbeit ist in drei Teile gegliedert. Im ersten Teil wird die Theorie der Ressourcen- und Terminplanung in Office365 mit Hilfe 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?

1.2 Ziele der Arbeit

Die Ziele der Arbeit sind deshalb 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.

1.3 Ergebnisse der Arbeit

Die Arbeit hat alle Ziele erreicht. Die Microsoft Graph API wurde erfolgreich eingesetzt und die Ressourcen- und Terminplanung in Office365 mithilfe von Microsoft Graph API wurde erfolgreich umgesetzt als SPA (Single Page Application). 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.

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 ein Mitarbeiter des Kunden 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 man nur diesen einen Raum auf Verfügbarkeit prüfen möchte, ist dies einerseits umständlich und vielleicht 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 und 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 Applikation entwickelt werden, die es ermöglichen soll, dass der Mitarbeiter selbstständig einen Termin buchen, oder einsehen kann, wann ein Termin denn stattfinden soll, wenn er sowieso schon im Gebäude und sich vor dem Raum befindet.
Bei diesem Kunden sind viele Mitarbeiter nicht aus der gleichen Abteilung oder dem gleichen Subunternehmen. Alle besitzen ihre eigenen Logos und Namen. Wenn also ein Termin nur anzeigt, wann er stattfindet, müsste der Mitarbeiter immer noch nachschauen, ob der Termin für ihn gedacht ist. Die Buchungen direkt vor dem Raum sind jedoch nur dafür gedacht, dass der Mitarbeiter den Raum für sich reserviert, falls er ihn jetzt gerade oder bald benötigt und dann wissen die Teilnehmer auch im Regelfall, dass der Termin für sie gedacht ist.
Sollte dies nicht der Fall sein, sollte man im Voraus, an einem anderen Endgerät, welches Outlook oder Teams besitzt, einen detaillieren Termin buchen, damit alle Beteiligten wissen, wann der Termin stattfindet und wer anwesend sein wird und dort kann man dann auch angeben, welche Firma der Gastgeber ist und welche der Gast.

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 Titel	Ressourcen- und Terminplanung in Office365 mithilfe von Microsoft Graph API 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.

1.6 Systemschnittstellen

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

Kurzbeschreibung:	Im UI kann der User sich mit der Ressource einloggen, die für die Zukunft als Repräsentant für die reelle Ressource (z.B einen Raum) gelten soll. Dann gilt dieser User immer als Teilnehmer der Buchungen und kann in seinen verfügbaren Zeiten gebucht werden. Basierend auf diesen Zeiten, werden dem User verfügbare timeslots zur Buchung angezeigt. Auch die Dauer des Termins soll einstellbar sein. Es gibt zudem einen optionalen Betreff für den Termin. Der gebuchte Termin reflektiert sich dann im UI in der Liste der für den Tag schon gebuchten Termine und den Terminen vom nächsten Tag, für die Ressource. Zudem soll der Nutzer, anhand eines farblichen UIs (siehe Abschnitt 6.1) sehen können ob innerhalb der nächsten 15 Minuten ein Termin für die Ressource ansteht oder ob einer schon am Passieren ist. REST API Anfragen werden dann an die Microsoft Graph API geschickt, um diese Wünsche zu kommunizieren.
Akteure:	Mitarbeiter
Häufigkeit:	Variabel pro Ressource, Firma und Anzahl Ressourcen
Komplexität:	Mittlere bis hohe Komplexität
Vorbedingungen:	1. User muss eingeloggt sein (Nutzerdaten können nicht gesehen oder eingespeichert werden) und Zugriffsrechte akzeptieren. 2. Optionale Teilnehmer 3. Termindauer 4. Verfügbare Termine 5. Terminbetreff 6. Zeitzone 7. Für die Microsoft Graph API kommt noch der Access Token / Client ID von unserer Azure App zur Authentifizierung dazu
Ausgaben:	1. Buchung erfolgreich oder nicht 2. Neue verfügbare Termine 3. Der neue Termin wird in der Liste der gebuchten Termine angezeigt 4. Falls die Buchung nicht geklappt haben sollte, wird der User darüber informiert
Use Case:	1.5
Schnittstellen:	1. Microsoft Graph API 2. User Interface 3. Backend
Aufgerufene Aktionen:	Notwendige: 1. Termin buchen 2. Termin finden Optionale: 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 das Internet aufgerufen wird.

2.1.2 Vor- und Nachteile von Webanwendungen

Eine Webanwendung ermöglicht es, dass die Anwendung von jedem Endgerät aus aufgerufen werden kann, welches über eine Internetverbindung verfügt. Webanwendungen sind relativ effizient, benötigen wenig Speicherplatz, heutzutage sehr sicher 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 sind heutzutage auf die Verwendung von [JavaScript] und all den Frameworks, die damit verbunden sind, angewiesen, welche oftmals sehr 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, die Anwendung eventuell nicht mehr funktioniert. Zudem ist die JavaScript Sprache nicht immer so fehlerabfangend wie andere Sprachen und lässt Kompilierungen zu, die dann im Laufzeitfehler enden.

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.

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 einigermaßen volles Paket. 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 diversen Anbietern, auf den Geräten, zu bestimmten Uhrzeiten, darstellen möchte, kann er dies über die ")OMS")-Lösung tun. Diese Inhalte werden dann auf den Geräten abgespielt.
Die Idee ist hier also, dass der Kunde nicht an die Terminbuchungs-Webapplikation gebunden ist, sondern diese Applikation nur als zusätzliche Funktion anbietet. 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 gegebenfalls nicht in einem Kiosk-Modus laufen können oder parallel zu unserer restlichen Software, die der Kunde haben möchte. Außerdem erforden einige dieser Softwarelösungen eine Installation auf dem Gerät oder erzwingen Konten-Registrierungen, die der Kunde nicht benötigt.
Die Applikation, die in dieser Arbeit entwickelt wird, soll es hingegen ermöglichen, dass diese Softwarelösung ohne weitere Softwarelösungen genutzt werden kann, aber auch ohne Probleme mit anderen Softwarelösungen genutzt werden kann, solange die anderen Softwarelösungen, von sich aus, keine Konflikte mit der Terminbuchungs-Webapplikation verursachen. 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.

4 Theorie

4.1 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. Mit Hilfe 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 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.

5 Ist-Zustand

Unsere Muttergesellschaft 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 jedoch nicht immer so praktikabel wie sie es vielleicht sein sollte, vor allem nicht, wenn mehrere Unternehmen, das gleiche Gebäude und die gleiche Organisations-E-Mail besitzen.
//UserJourney Beispiel einfügen
Falls ein User zufällig an einem Raum vorbeigeht oder vor einem Raum steht und sich fragt, ob dieser Raum belegt ist oder nicht, muss er 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 sehr umständlich und kann sehr viel Zeit in Anspruch nehmen.

6 Soll-Zustand

6.1 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 heute noch anstehen und spontan auch einen Termin 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.

6.2 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.
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 sehr 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 man 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.

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

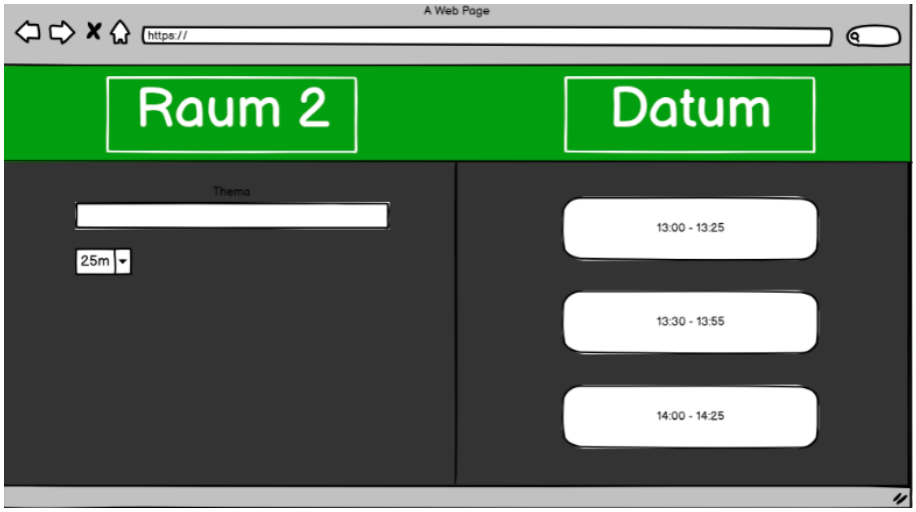
6.3 LED Strips

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

7 Vorgehensweise

7.1 Prototyp

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



7.2 Technische Umsetzung

7.2.1 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.Das ganze wird als eine Azure App deployt. 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 immer die Gleiche, auf allen Geräten, damit Microsoft’s redirect URI Bedingungen erfüllt werden können und Webserver kosten eingespart werden können, da Updates eher selten passieren sollten.
Zudem ist es auch für die Sicherheit der Nutzer so viel besser, da wir deshalb keinen Zugriff auf ihre Daten haben. Zeiten müssen ISO 8601 Konform sein bei jeglichen API Anfragen an die Microsoft Graph API.

7.2.2 Benutzte Hardware und Software

Hardware:

- Philips 10BDL4551T/00

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.

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, das es ermöglicht, Änderungen an Dateien zu verfolgen und zu verwalten.

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 ermöglicht, JavaScript Code zu schreiben und zu debuggen.

7.2.3 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.

7.2.4 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, aber auch um Fehler zu finden, die nicht vorgesehen waren, indem man monkey-testing betreibt.
Mithilfe der oData v4 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";
```

7.2.5 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 diese 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 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 natürlich verkürzen und die Animationdauer und -verzögerung direkt berechnen, aber so ist es verständlicher.
Hier sieht man den 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 wird, die Animationsdauer und -verzögerung neu gesetzt wird und der Animationsname erst entfernt und dann wieder hinzugefügt wird.

[1] **Reflow**.

8 Ergebnis

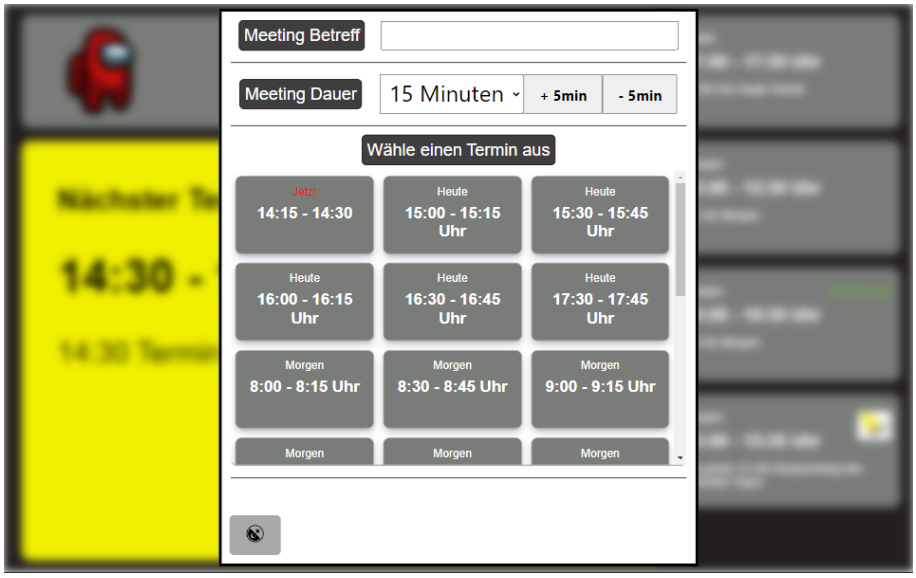
Das Ergebnis sieht wie folgt aus:

8.1 Visuelle Darstellung

Oben links im Bild, das Bild des kleinen roten Charakters, sieht man das Logo des Gastgebers des nächsten, beziehungsweise jetzigen, Termins. 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 immer 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 man sieht, ist beispielsweise die selbsterstellte Option ”Jetzt” deaktiviert, da das Ende des hypothetischen Termins innerhalb von 15 Minuten vom nächsten anstehenden Termin beginnt. Diese Pufferzeit wurde so mit dem Kunden abgesprochen. Auf die anderen Optionen hat man hier wenig Einfluss, da sie von den Einstellungen des Kalenders, des jeweiligen Nutzers, abhängen und es somit in der Verantwortung des Nutzers liegt, diese zu ändern.

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

8.2 Performance

Die Performance einer SPA zu messen ist nicht ganz einfach. Auch die gängige Total-Blocking-Time Messung ist nicht zwingend sinnvoll, da die Anwendung ja nicht blockiert, sondern nur die Daten vom Server lädt und danach die Nutzung der Anwendung möglich ist. Daher wurden manuell einige Tests durchgeführt, um die Performance zu messen.

8.2.1 Test 1

Wie lange braucht die Anwendung, um nach einem Klick auf den Button ”+” das Terminerstellungs-Menü zu öffnen? Diese Frage wurde gemessen, indem der Event-Listener auf den Button ”+” registriert wurde und die Zeit gemessen wurde, die vergeht, bis der Event-Listener aufgerufen wurde und das Menü dargestellt wurde. Es dauert im Durchschnitt 2ms, bis das Menü angezeigt wird.

8.2.2 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.

8.2.3 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 sowieso auch Zeit benötigt, um die Anfragen zu bearbeiten und die neuen Daten bei sich zu verarbeiten, würde man den Unterschied nicht merken. Der Bottleneck

[JavaScript]

Literatur

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

literatur