Hochschule für Technik und Wirtschaft Dresden

Fakultät Informatik/Mathematik

Bachelorarbeit

im Studiengang Informatik

Thema: Entwicklung und Untersuchung eines webfähigen Outlook-Add-Ins auf der Basis der Microsoft-Office-API

eingereicht von: Sergej Riel

eingereicht am: 5.August 2022

Betreuer: Dipl.-Wirtschaftsinformatiker (FH) Jens Gölker

Betreuender Hochschullehrer: Prof. Dr. -Ing. Jörg Vogt

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
CCERP	Cosmo Consult Enterprise Resource Planning
COM	Component Object Model
DWH	Data Warenhouse
ERP	Enterprise Resource Planning
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Objekt Notation
JWT	JSON Web Token
OBO	OAuth 2.0 On-Behalf-Of flow
PIA	Primäre Interop-Assemblys

Glossar

Begriff	Definition / Erklärung
Add-In	Ein Add-In ist eine Software-Komponente, die ein Hauptprogramm mit zusätzlicher Funktionalität erweitern kann.
Build	In der Softwareentwicklung wird mit Build der Vorgang oder dessen Resultat bezeichnet, bei dem ein Programm aus Quellcode automatisch erzeugt wird.
Client	Als Client wird ein Computer oder ein Programm bezeichnet, der mit einem Server in Kontakt steht.
Cloud-Anwendung	Eine Cloud-Anwendung ist ein Programm, das mindestens zum Teil auf einem Server verarbeitet wird.
Debugging	Als Debugging wird der Reinigungsprozess verschiedener Fehler eines Programms bezeichnet.
ERP-System	Als ERP-System bezeichnet man ein Programm zur Ressourcenplanung eines Unternehmens.
Fakturierung	Unter Fakturierung versteht man die Erstellung einer Rechnung über getätigte Leistungen.
HTTP Header	Im HTTP Header befinden sich die Kopfzeilen einer Web-Anfrage oder einer Antwort zwischen dem Client und Server. Diese können zusätzliche Informationen beinhalten.
Mandant	Als Mandant wird im Rahmen des Business Central ERP Systems ein separates Unternehmen betrachtet, dessen Datenbank von anderen, im selbem System angelegten, Unternehmen getrennt ist. Jeder Mandant hat innerhalb von COSMO CONSULT getrennte Projektnummern.
Server	Ein Server ist ein Computer, oder ein darauf laufendes Programm, dessen Funktionalität durch das Internet an die Nutzer zur Verfügung gestellt werden kann.

Abbildungsverzeichnis

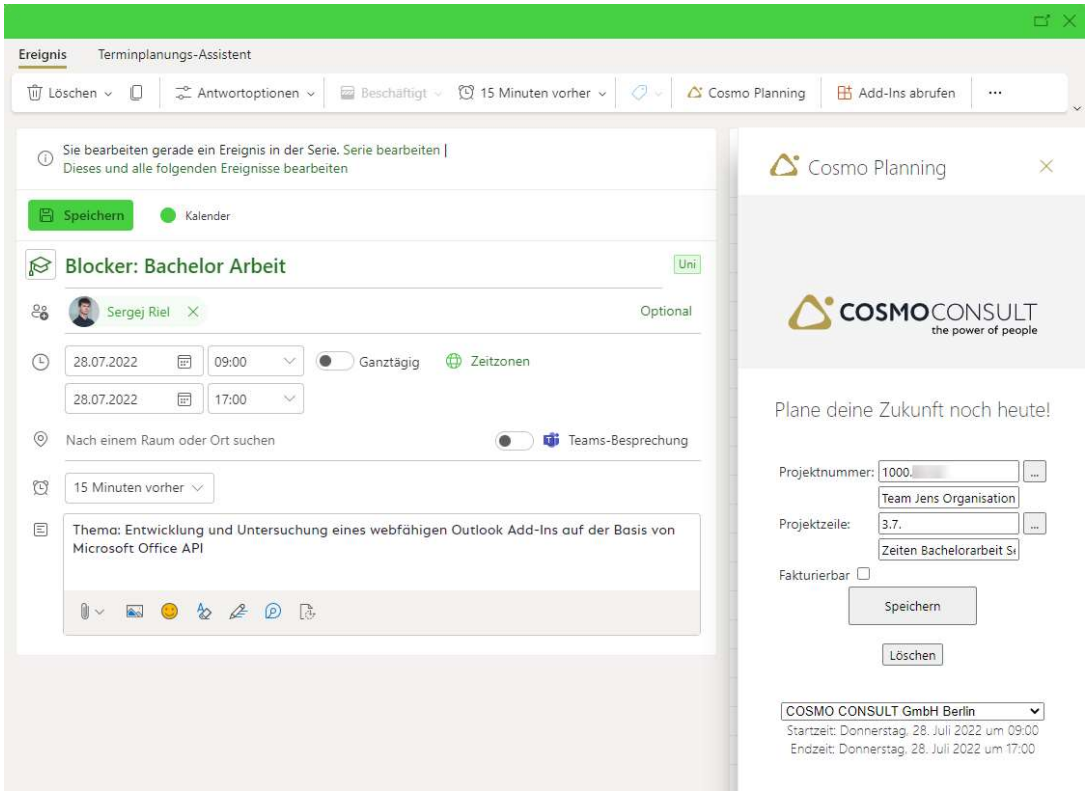

Abbildung 1: Outlook Add-In im Browser (Rechts)

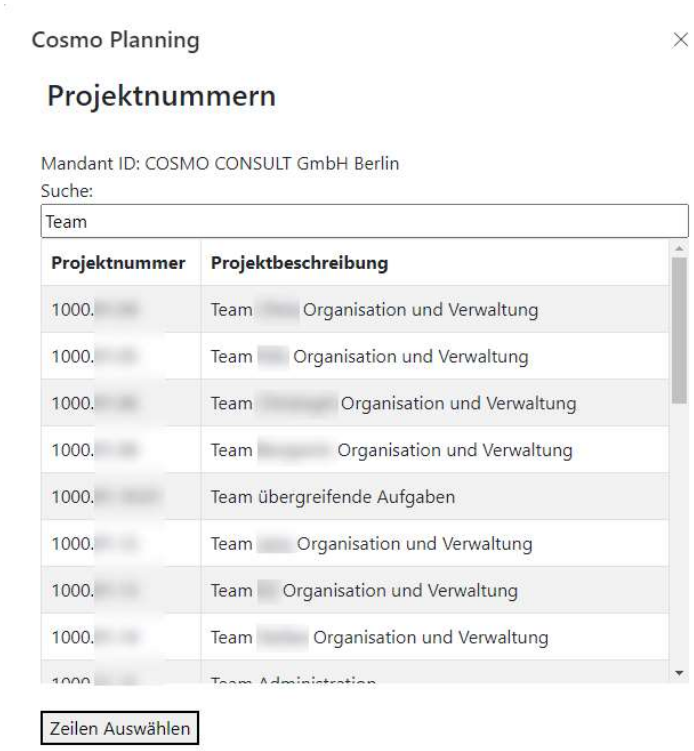

Abbildung 2: Add-In Dialogfenster

Codiert:

eyJ0eXAiOiJKV1QiLCJhb[redacted]
XBKM1VwYmpBWVhZR2FYRUpsOGxWMFRPSSJ9.eyJhdWQiOiJhcGk6Ly9jb3Ntby1vdXRsb29rLXBsYW5uaW5nLWFkZG1
uLmF6dXJld2Vic2l0ZXMubmV0LzRkYmI2NGUwLTU5YWItNDdkZS1hZGZhLWVmNTI3NjZkMmYxMSIsImlzcyI6Imh0dHB
zOi8vc3RzLndpbmRvd3MubmV0Lzk0ZDMwNTI4LTQ2NGYtNDVhNS1hOTVlLTgxNjQyOGY1N2I5MC8iLCJpYXQiOjE2NT
[redacted]
3JpZWw4OTY2QGNvc21vY29uc3VsdC5jb20iLCJ1cG4iOiJzcmllbDg5NjZAY29zbW9jb25zdWx0LmNvbSIsInV0aSI6ImlSZGJD
LWJ3TlUtWUFhSnE5ZU5CQUEiLCJ2ZXIiOiIxLjAifQ.OAVkhyozeq_2taOkcnWeCVA8vPYAYqIn-
IQySavElTvhvhDtqHAr8jB4iJmiibAMDlnXJtadp4qBPkJxA5ssOitWYuQBt1grMgdmkr73KtJIK75vEmTbWdVkf9YWMOYhNG
[redacted]
K6ZuGoBijOK8LQRGFQuJdbsInaF2JsC-
gk5oFpENvHJFYozRXYziCLDp8UaKmp4MvyKn_o40tqprBUtse6wqZabAR1yg2lqkRi8-yLnUNEXuokav6VSdTS2Q

Abbildung 3: Codiertes JWT

Dekodiert:

{
  "typ": "JWT",
  "alg": "RS256",
  "x5t": [redacted]
  "kid": [redacted]
}.{
  "aud": "api://[redacted]/4dbb64e0-59ab-47de-adfa-ef5[redacted]",
  "iss": "https://sts.windows.net/94[redacted]7b90/",
  "iat": 1657217985,
  "nbf": 1657217985,
  "exp": 1657222455,
  "acr": "1",
  "aio":
"[redacted]Uuj
a[redacted].
  "amr": [
    "pwd",
    "rsa",
    "mfa"
  ],
  "appid": "4dbb64e0-59ab-47de-adfa-ef[redacted]1",
  "appidacr": "1",
  "deviceid": "6bb4277e-[redacted]",
  "family_name": "Riel",
  "given_name": "Sergej",
  "ipaddr": "[redacted]",
  "name": "Sergej Riel",
  "oid": "[redacted]",
  "onprem_sid": "S-1-5-21-1196811695-[redacted]",
  "rh": "0.AUgAKAXTlE9GpUW[redacted].",
  "scp": "access_as_user",
  "sub": "uS4p_hbtzpXr5G[redacted]M",
  "tid": "94d3[redacted],
  "unique_name": "[redacted]l@cosmoconsult.com",
  "upn": "[redacted]@cosmoconsult.com",
  "uti": "i[redacted]",
  "ver": "1.0"
}.[Signature]

Abbildung 4: Decodiertes JWT

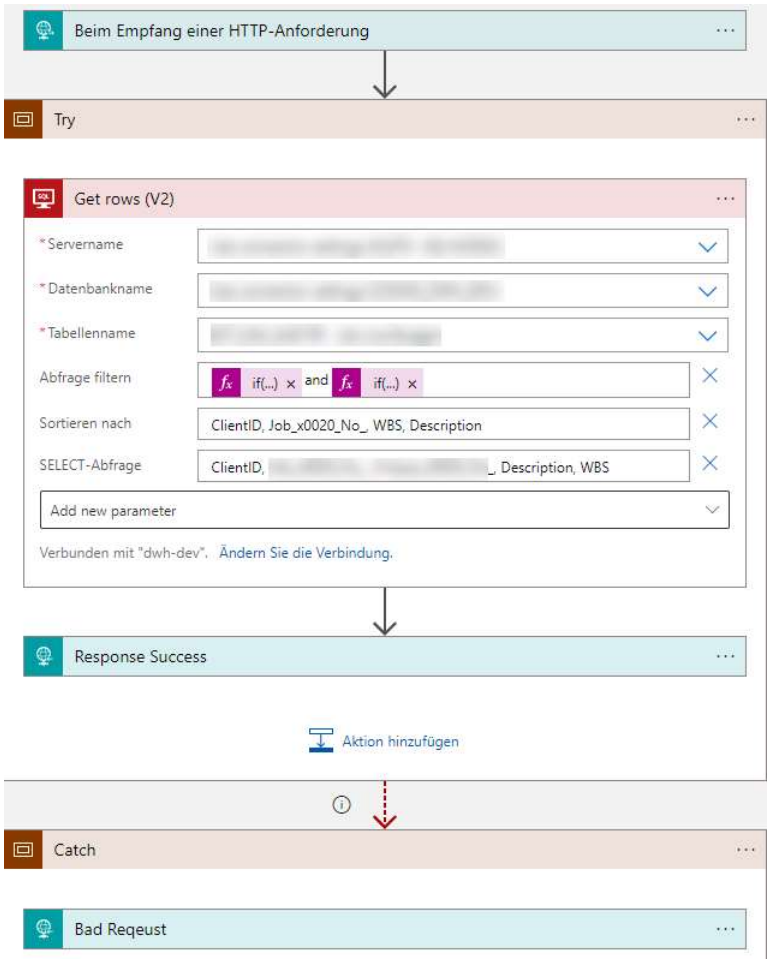

Abbildung 5: Logic App für Abrufen der Projektnummer-Liste

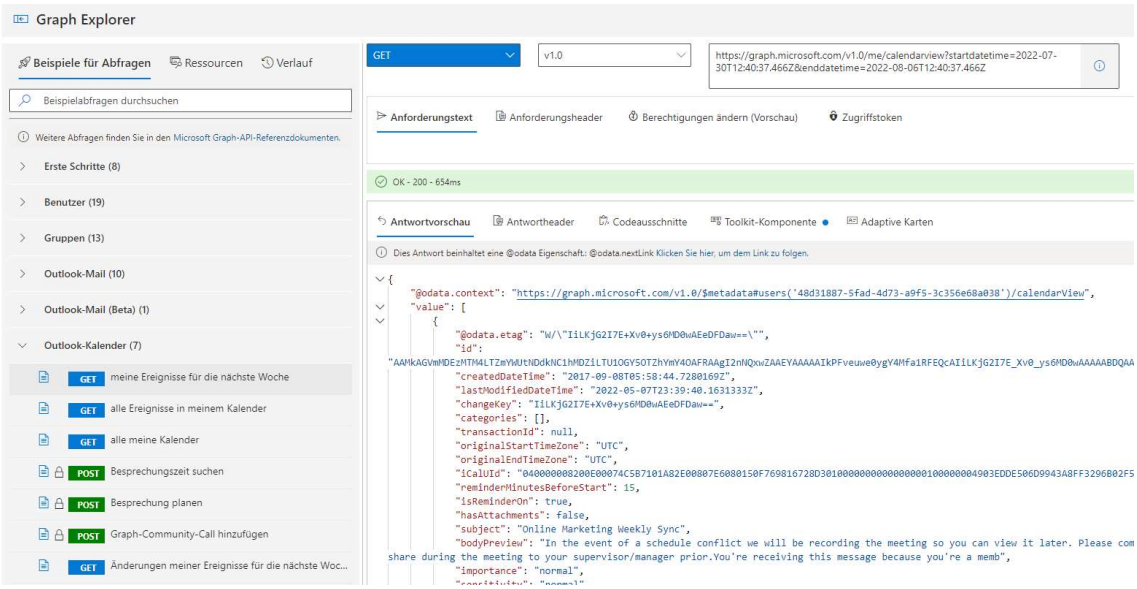

Abbildung 6: Graph Explorer, Quelle: https://developer.microsoft.com/de-de/graph/graph-explorer?request=me%2Fevents&version=v1.0 30.07.2022

1 Einleitung

Die vorliegende Arbeit befasst sich mit der Erstellung und Untersuchung eines auf der Microsoft Office API basierenden Outlook Add-Ins. Das Add-In wird für den Einsatz innerhalb der Firma COSMO CONSULT GmbH erstellt und später in Betrieb genommen. COSMO CONSULT ist ein weltweiter Anbieter von IT-Lösungen und Dienstleistungen für die Digitalisierung von Unternehmen. Das Unternehmen ist ein Microsoft Partner und befasst sich hauptsächlich mit der Enterprise-Ressource-Planning Software (weiter mit ERP abgekürzt) - Microsoft Dynamics 365 Business Central. Das Unternehmen verkauft Lizenzen und führt individuelle Anpassungsprojekte für verschiedene Kunden durch. Jedes Kundenprojekt besitzt eine eindeutige Projektnummer und die dazu gehörigen Projektzeilen, an die die erbrachte Leistung fakturiert wird. Um den Planungsprozess übersichtlicher zu gestalten, wurde in vergangenen Jahren das sogenannte Outlook Planungs Add-In eingeführt. Jeder Mitarbeiter plant seine Aufgaben im Outlook Kalender, um die Arbeit übersichtlicher zu gestalten. Sobald eine Aufgabe erbracht wurde, erfasst jeder Mitarbeiter seine erbrachte Zeit mit der zugehörigen Projektnummer und Projektzeile im COSMO CONSULT ERP System.

Für eine bessere Ressourcen Planung wurde das alte Outlook Add-In eingeführt. Das Add-In ermöglichte die Angabe von Projektdaten am jeweiligen Termin im Outlook Kalender. Diese Daten wurden direkt an das COSMO CONSULT ERP System übergeben und dort zur Analyse der Arbeitszeiten verwendet. Die Kalendereinträge der Mitarbeiter wurden dort maschinell, mit der vom jeweiligen Projektleiter erstellten Planung, abgeglichen. Damit wurde sichtbar, ob die Mitarbeiter, die von deren Projektleitern gestellten Aufgaben, tatsächlich im Kalender geplant hatten oder nicht. Weiter konnten die Projektleiter einsehen, wie sehr deren Mitarbeiter bereits ausgeplant waren. Eine weitere Funktion bestand darin, die freie Kapazität jedes einzelnen Mitarbeiters zu bestimmen.

In Rahmen meines Praktikums wurde zusammen mit der Geschäftsleitung der COSMO CONSULT Ost und sechs weiteren Teamleitern die Analyse des alten Outlook Add-Ins durchgeführt und ein Konzept für ein Neues entwickelt, was im folgenden Kapitel näher erläutert wird.

Das alte Outlook Add-In musste von jedem einzelnen Nutzer manuell installiert werden. Die Projektnummern sowie die Projektzeilen wurden als zwei große Dateien lokal runtergeladen und mussten regelmäßig vom Benutzer aktualisiert werden. Die erfassten Daten wurden bei dem jeweiligen Kalendereintrag im Outlook als weitere Eigenschaft gespeichert. Um die Daten weiter verarbeiten zu können, musste innerhalb des COSMO CONSULT ERP Systems ein SQL-Dienst gestartet werden, der alle Benutzerkonten durchlief und die Daten in die dazu gedachten Tabellen speicherte. Dieser Prozess war für gewöhnlich sehr zeitintensiv und wurde manuell nachts gestartet. Die Akzeptanz des Add-Ins war niedrig, da die Installation teilweise aufwendig war, eine VPN Verbindung benötigt wurde und das Add-In in der Outlook-Webvariante nicht funktionierte.

Innerhalb meines Praktikums wurde ich bereits nach kurzer Zeit vor einige Schwierigkeiten gestellt. Die Technologie, mit der das vorherige Add-In erstellt wurde, ließ sich erschwert recherchieren, da Microsoft bereits seit einigen Jahren an einer neuen

Technologie arbeitet.[1] Das von Microsoft entwickelte Office Application Programming Interface (im Folgende als API abgekürzt), ermöglicht es, ein Add-In zu erstellen, welches gleichzeitig auf Desktop funktioniert und im Web abrufbar ist. Da das Add-in web fähig sein sollte, konnten die Projektdaten nicht lokal abgelegt werden. Die Authentifizierung im Web ist nun auch ein wichtiger Bestandteil. Die Aufgabe lag nun darin eine Möglichkeit zu finden, die Daten sicher in das Add-In zu speisen und zur Auswertung im System zu speichern. Da es in der Firma keine auf die Programmierung von auf Microsoft API basierten Add-Ins spezialisierte Fachkraft gab, wurde die Entwicklung vorerst pausiert. Da ich selbst mit der Webentwicklung kaum vertraut war, konnte ich lediglich eine einfache Oberfläche mit mehreren Textfeldern gestalten, die später zur Erfassung von Daten dienen sollte. Zusätzlich habe ich eine einfache Tabellarische Anzeige von Projektnummern gebaut.

Ab diesem Zeitpunkt war ein kleines Team aus zwei Menschen damit beschäftigt einen Weg zu finden, um mir die Daten sicher aus den COSMO CONSULT ERP System zur Verfügung zu stellen und eine Auswertungsplattform zu gestalten. In dieser Zeit beschäftigte ich mich mit anderen Themen innerhalb des COSMO CONSULT ERP Systems und nahm an mehreren Schulungen teil. Am Ende meines Praktikums wurde die passende Auswertungsplattform eingerichtet und eine Datenquelle zur Verfügung gestellt.

Aufgrund des hohen Interesses meiner Firma, an der Fertigstellung dieses Outlook Add-Ins, habe ich nun die Möglichkeit innerhalb meiner Bachelorarbeit die Office API genauer zu untersuchen und alle Anforderungen umzusetzen, sofern dies mir möglich ist.

[1] Vgl. https://docs.microsoft.com/en-us/office/dev/add-ins/ (16.06.2022).

2 Zielstellung

Das Ziel dieser Bachelorarbeit ist es, erstens zu untersuchen, wie weit sich die Anforderungen mithilfe der gewählten Office API umsetzten lassen können und zweitens, ein sicheres, webfähiges und produktionsreifes Outlook Add-In zu erstellen.

2.1 Anforderungsanalyse

Das alte, und nicht mehr benutze Outlook Add-In muss komplett neu entwickelt werden. Das neue Add-In soll die Erfassung von Termin- und Projektdaten am jeweiligen Eintrag im Outlook Kalender ermöglichen. Die Projektdaten bestehen aus einer Projektnummer, Projektzeile, Client Identifikationsnummer und aus der Angabe der Fakturierbarkeit. Zu den benötigten Termindaten gehören die Terminnummer, Nutzer-E-Mail, sowie die Start- und die Endzeit. Das Add-In soll sowohl auf den Desktop Outlook Client, als auch im Browser voll funktionsfähig sein. Die Notwendigkeit einer VPN Verbindung für die Beschaffung der Projektdaten muss abgeschafft werden. Weiterhin soll es so wenig Benutzer Interaktion wie möglich erfordern. Die am jeweiligen Termin gespeicherte Daten müssen von den Terminersteller wieder abrufbar sein. Das Add-In soll im Kalender nur für den Terminersteller, sowie für die von den Terminersteller angegebenen Terminteilnehmer sichtbar sein. Aus Datenschutzgründen darf nur der Autor eines Eintrags seine Angaben einsehen und bearbeiten. Um die Angabe der Projektdaten zu erleichtern, muss eine Suche nach einer Projektnummer und einer Projektzeile implementiert werden. Jeder Benutzer muss in der Lage sein, sein derzeitiges Mandant auswählen zu können, der mit der entsprechenden Client Identifikationsnummer gespeichert wird. Die zu suchenden Projektnummern sollen nach ausgewähltem Mandanten gefiltert angezeigt werden. Es muss ein Weg gefunden werden, um die Projektdaten performant aus den CCERP System zu laden und zu speichern. Die Projektsuche und der Eintrag sollten innerhalb von zehn Sekunden möglich sein. Die Übergabe der Daten soll abgesichert verlaufen. Nur die Nutzer von COSMO CONSULT dürfen auf die Projektdaten zugreifen. Dafür muss eine passende Authentifizierung am System implementiert werden. Schließlich muss ein Weg gefunden werden, das Add-In ordnungsgemäß für die Endnutzer in die Produktion einzubauen. Das Add-In soll auch auf eine Terminverschiebung reagieren und diese möglichst automatisch speichern können. Weiterhin soll eine Kopierung eines bereits ausgefüllten Eintrags im Kalender erkannt und behandelt werden. Die Eingabe von Projektdaten sollte auch ohne eine Suche möglich sein. Dabei muss die Eingabe zwangsläufig mit Hilfe der existierenden Projektdaten auf Korrektheit abgeglichen werden. Nach der Projektnummer Neuauswahl muss der Faktura Hacken wieder gesetzt werden. Dieser soll standardmäßig immer angekreuzt werden. Weiter folgende Anforderungen sind optional. Das Add-In kann den Benutzer die Möglichkeit anbieten, seine Favoritenprojekte speichern zu können, um die Suche zu erleichtern. Das Add-In soll möglichst das Anlegen von Serienterminen unterstützen. Weiterhin sollte das Add-In beim Erfassen eines Kalendertermins immer eingeblendet sein. Alternativ kann das Add-In per Tasten-kombination aufgerufen werden. Das Add-In soll auf das Speichern und Löschen eines Kalendereintrags reagieren können. Die Veränderung der Zeit am Termin soll abgefangen werden. Der Teamleiter sollten in der Lage sein, Einträge für seine Mitarbeiter einzutragen. Mit anderen Worten sollte das Add-In die globalen Abfragen der gespeicherten Projektdaten von anderen Terminteilnehmern ermöglichen.

3 Technologieauswahl der Outlook Add-In Entwicklung

In diesem Kapitel werden mehrere von Microsoft Corporation, dem internationalen Hard- und Softwareentwickler, unterstützten Technologien untersucht, mithilfe deren sich Outlook erweitern lässt. Outlook ist eine, von Microsoft bereit gestellte Software, die das Versenden und Empfangen von E-Mails sowie das Verwalten von Terminen ermöglicht. Es muss eine Technologie gewählt werden, die möglichst alle Anforderungen der Anforderungsanalyse abdecken kann.

Die ersten beiden Technologien sind das Outlook-Objektmodell und die Primäre Interop-Assembly (PIA). Das Outlook-Objektmodell stellt Schnittstellen bereit, die das Bearbeiten von Outlook Benutzeroberflächen, sowie den Zugriff auf den lokalen Mail Ordner ermöglichen. Die Primäre Interop-Assembly ist auch eine Schnittstelle, die für ein COM-basiertes Objektmodell die Interaktion mit einer Microsoft Office-Anwendung, unter anderen mit Outlook, ermöglicht.[2] Ein COM-Objekt, abgekürzt von Microsoft Component Object Model, ist ein plattformunabhängiges und objektorientiertes System, mit der Softwarekomponente entwickelt werden können.[3] Beide Technologien besitzen ungefähr dieselbe Funktionalität und werden zusammen betrachtet. Auf das Objektmodell, als auch auf PIA basierende Outlook Add-Ins werden direkt in die Outlook Oberfläche eingebaut und sind somit ein fester Bestandteil des Clients. Die Prozessabarbeitung passiert somit zusammen mit dem restlichem Outlookclient seriell. Damit können verschiedene, benutzerdefinierte Lösungen entwickelt werden. Die Implementierung erfolgt mit Visual Studio .NET Framework in den Programmiersprachen Visual Basic oder C#. Das Add-In kann den jeweiligen Outlook Desktop Client anpassen und auf die zugehörigen Ordner zugreifen. Mit dieser Technologie wurde das Alte Outlook Add-In von COSMO CONSULT entwickelt. Die Technologie bietet den vollen Zugriff auf alle Nutzerdaten inklusive der Kalenderdaten, funktioniert aber ausschließlich lokal. Somit wird die offline Funktionalität gewährleistet. Das alte Add-In konnte zwar nach dem Nachladen von Projektdaten die erfassten Daten lokal speichern und erst mit einer wieder aufgebauten Internetzugang die Daten unter einer VPN Verbindung wieder abgeben, funktioniert jedoch im Outlook Webclient nicht. Da viele COSMO CONSULT Mitarbeiter Laut einer internen Umfrage den Desktop Client gar nicht benutzten, wurde die Weiterentwicklung des alten Add-Ins eingestellt.

Die dritte Technologie ist das MAPI Clientprotokoll, welches den Zugriff auf das Outlook Postfach ermöglicht. E-Mail-Clients, die ein MAPI Protokoll unterstützen, können ebenfalls die Nachrichten aus dem Outlook Postfach empfangen und versenden.[4] Es konzentriert sich Hauptsächlich auf das Versenden von E-Mails und wird meistens für die komplexe E-Mail-Verarbeitung verwendet.

[2] Vgl. https://docs.microsoft.com/de-de/visualstudio/vsto/office-primary-interop-assemblies?view=vs-2022 (22.07.2022).
[3] Vgl. https://docs.microsoft.com/de-de/windows/win32/com/the-component-object-model (22.07.2022).
[4] Vgl. https://docs.microsoft.com/de-de/exchange/clients/mapi-mailbox-access?view=exchserver-2019 (22.07.2022).

Die letzte Technologie, die zurzeit ständig aktualisiert wird, ist das Entwickeln von Apps für die Office-Plattform. Ein Office Add-In ist eine auf HTML, CSS und JavaScript basierende Webseite, die sich in einer Microsoft Office Anwendung, wie zum Beispiel Outlook, aufrufen lässt und lokal oder auf einem Webserver ausgeführt werden kann. Ein auf der Office-Plattform basierendes Add-In wird auch als Mail-App bezeichnet. Sowohl die Endbenutzer als auch die Administratoren können das Add-In in ihrem Exchange-Postfach installieren. Ein Microsoft Exchange-Postfach ist ein Server, der für die Verwaltung von E-Mails, Kalender und dessen kooperative Terminplanung zuständig ist. Darin werden alle Postfach spezifischen Daten gespeichert.[5] Nach der Installation kann das Add-In an vorher festgelegten Orten auf der App-Leiste im Outlook eingeblendet werden, sobald eine bestimmte Bedingung erfüllt ist. Im Gegensatz zum vorher erwähnten Objektmodell wird das Office Add-In nicht direkt in die Anwendung eingebaut, sondern nur eingeblendet und besitzt somit seinen eigenen Abarbeitungsprozess.

Da ein Office Add-In grundsächlich eine Webseite ist, kann dieses sowohl auf Desktop- als auch auf der Outlook Web-Variante ausgeführt werden. Ein Office Add-In kann außerdem auf einem Tablet, sowie ebenfalls, unter bestimmten Anpassungen, auf Mobilgeräten ausgeführt werden. Im Vergleich zu den vorher beschriebenen COM-APIs zeigen die Office Add-Ins einen niedrigeren Administrationsaufwand. Das Office Add-In kann von der IT für eine gewünschte Arbeitsgruppe zugewiesen werden. Sobald das Add-In einmal installiert ist, muss es bei weiteren Updates nicht für jeden Benutzer erneut manuell installiert werden. Die entwickelten Add-Ins können auch über den Office Store veröffentlicht und weiterverkauft werden. Zusätzlich bieten die auf Office basierenden Add-Ins mehr Datenschutz als die Objektmodell basierten Add-Ins, die einen uneingeschränkten Zugriff auf den gesamten Inhalt aller Konten im Benutzerprofil besitzen.[6] Als Referenz darauf, bietet das frühere Objektmodell eine einfache Implementierung von Serienterminen, die in Office Add-Ins eine komplette Authentifizierung erfordern.[7] Es wird nur eine eingeschränkte Datenmenge zur Verfügung gestellt. Für weitere Nutzerspezifische Daten muss eine Authentifizierung durch Identitätstokens implementiert werden.

Da diese Technologie das Entwickeln von Webfähigen Outlook Add-Ins ermöglicht, ist es somit eine passende Option für das nächste COSMO CONSULT Outlook Planung Add-In. Zunächst muss jedoch die Office Add-In API auf das Erfüllen der, bei der Anforderungsanalyse gestellten, Anforderungen untersucht werden.

[5] Vgl. https://www.techtarget.com/searchwindowsserver/definition/Microsoft-Exchange-Server (23.07.2022).
[6] Vgl. https://docs.microsoft.com/de-de/office/client-developer/outlook/selecting-an-api-or-technology-for-developing-solutions-for-outlook (23.07.2022).
[7] Vgl. https://docs.microsoft.com/en-us/office/client-developer/outlook/pia/how-to-find-a-specific-appointment-in-a-recurring-appointment-series (23.07.2022).

3.1 Untersuchung der Microsoft Office API

In diesem Kapitel werden sowohl die aktuellen Möglichkeiten, als auch Limitierungen der Microsoft Office API in Rahmen von Outlook anhand der Anforderungsanalyse untersucht.

Tabelle 1: Tabellarische Untersuchung der Microsoft Office API

Technologische Anforderung	**Office-API Unterstützung und Umsetzung**
Das Erfassen von Projektdaten am jeweiligen Eintrag im Outlook Kalender	**Ja**, die mit Office-API erstellten Add-Ins können im Outlook Kalender separat für den Terminersteller und für den Terminteilnehmer beim Lesen und Editieren eines Termins angezeigt werden. Die Einstellung erfolgt in der Manifest Datei.
Termin Identifikationsnummer	**Ja**, API-Aufruf: *Office.context.mailbox.item.itemId*
Zugriff auf Nutzer-E-Mail	**Ja**, API-Aufruf: *Office.context.mailbox.userProfile.emailAddress*
Start und Endzeit	**Ja,** API-Aufrufe: *Office.context.mailbox.item.start.getAsync* *Office.context.mailbox.item.end.getAsync*
Funktionsfähigkeit in Desktop Outlook Client, als auch im Browser	**Ja**, die Office Add-Ins werden mit den Websprachen JavaScript / TypeScript sowie mit HTML und CSS implementiert und von Browser unterstützt.
Das Speichern von Daten an jeweiligen Terminen	**Ja**, aber nur für den Ersteller sichtbar. API-Aufruf: *customPropsCallback(asyncResult)*
Das Speichern von Konfigurationsdaten für alle Termine	**Ja**, bis 32KB pro Benutzer. API-Aufruf: *Office.context.roamingSettings.get(settingName)*
Authentifizierung an externen Schnittstellen	**Eingeschränkt,** die Office-API stellt eine beschränkte Nutzer Datenmenge zur Verfügung und ist nur für eine interne Datenverarbeitung innerhalb der Microsoft Office Produkten ausgestattet. Die Authentifizierung kann mit Hilfe von weiteren Microsoft Schnittstellen, wie z.B. mit Single-Sign-On, realisiert werden. Dazu mehr in Kapitel 4.5.
Serientermine einzeln zurückgeben.	**Eingeschränkt**, die Office-API kann Serientermine anhand eines Serienmusters erstellen und bei bereits angelegten Serienterminen nur den genauen Start- und Endzeitpunkt zurückgeben.
Add-In im Kalender automatisch einblenden lassen	**Nein**, die Add-In Oberfläche kann nur beim Erfassen und Lesen einer E-Mail automatisch eingeblendet werden. [8]

[8] Vgl. https://docs.microsoft.com/en-us/office/dev/add-ins/outlook/pinnable-taskpane (21.07.2022).

Unterstützung von benutzerdefinierten Tastenkombinationen	**Nein**, zurzeit werden die benutzerdefinierten Tastenkombinationen nur in Excel unterstützt.[9]
Das Speichern eines Kalendereintrags erkennen	**Eingeschränkt**, die Office-API unterstützt das Abfangen von bestimmten Ereignissen. Es kann auf das Versenden eines Kalendereintrags an weitere Teilnehmer beim Abfangen von *OnAppointmentSend* Event reagieren. Sonst kann das Add-In das Speicherstatus manuell abfragen lassen.
Das Löschen eines Kalendereintrags erkennen	**Nein**, zurzeit existiert kein Event, welches beim Löschen eines Kalendereintrages ausgelöst wird. Der Nutzer muss somit das DWH Eintrag manuell löschen, bevor der Kalendereintrag aus Outlook entfernt wird.
Die Zeitverschiebung eines Kalendereintrags erkennen	**Ja**, dazugehöriges Event: *OnAppointmentTimeChanged*
Das Abfragen der gespeicherten Projektdaten anderer Mitarbeiter	**Nein**, die in den *customProps* gespeicherte Informationen sind nur für die Autoren zugänglich.
Terminteilnehmer müssen auf die Projektdaten des Termin Organisators zugreifen können	**Nein**, dieses Szenario wurde überprüft. Alle Kalendereinträge der Teilnehmer desselben Termins besitzen unterschiedliche Termin ID´s. Somit können die in DWH gespeicherten Daten den Teilnehmern nicht eindeutig zugeordnet werden.

Alle hier erwähnten API-Aufrufe werden im Kapitel 4: „Entwicklung eines Outlook Office Add-Ins“ mit entsprechenden Belegen genauer beschrieben.

Die Office Add-In API unterstützt auch weitere Funktionen, die in diesem Fall nicht verwendet wurden. Das Add-In kann nicht nur im Kalender, sondern auch beim Erfassen und Lesen einer E-Mail aufgerufen werden. Es ermöglicht weiterhin das Anpassen des Titels und des Textes innerhalb der Terminbeschreibung, sowie innerhalb einer E-Mail-Textnachricht. Im Kalender kann die API die Teilnehmer und den Terminort anpassen, sowie das setzten der bereits in der Tabelle „Tabellarische Untersuchung der Microsoft Office API“ erwähnten Start- und Endzeit. Ebenso wie im Nachrichten Modus, kann das Add-In im Kalendertermin die Eigenschaften einer eingefügten Anlage auslesen. Außerdem können die allgemeinen Informationen nicht nur vom Autor, sondern auch über die Teilnehmer, wie zum Beispiel Name oder die Telefonnummer, zur Verfügung gestellt werden. Beim Erfassen einer E-Mail können zusätzlich Eigenschaften wie Empfänger und Zweitempfänger angepasst werden.

Um auf weitere Nutzerspezifische Daten zugreifen zu können, muss Microsoft Graph, eine weitere Schnittstelle, verwendet werden. Weiteres dazu im Kapitel 4.5.4.

[9] Vgl. https://docs.microsoft.com/en-us/office/dev/add-ins/design/keyboard-shortcuts (21.07.2022).

4 Entwicklung eines Outlook Office Add-Ins

4.1 Voraussetzungen

Für die Entwicklung benötigt man zunächst ein Microsoft 365 Konto, mit welchem man auf Outlook zugreifen kann. Im Rahmen dieser Arbeit wurde dafür ein Arbeitskonto von COSMO CONSULT verwendet.

Im nächsten Schritt muss man sich für eine der zwei möglichen Entwicklungsumgebungen entscheiden, welche von Microsoft angeboten werden. Die erste basiert auf Visual Studio, eine Entwicklungsumgebung für verschiedene Projekte, die mit .NET Sprachen entwickelt werden. Ein Nachteil von Visual Studio ist es, dass es nur unter dem Betriebssystem Windows lauffähig ist. Weiterhin muss die Serverseite des Add-Ins in einer der .NET Sprachen geschrieben werden. Für diese lassen sich jedoch nur eine mangelhafte Anzahl an Funktionsbeispielen finden. Schließlich kann das Programm nicht in Visual Studio direkt debuggt werden, sondern nur aus der Clientsicht im Browser.

Die zweite Variante ist die Node.js Umgebung. Diese ist eine plattformunabhängige, auf der Websprache JavaScript basierende, Open-Source-Laufzeitumgebung, die von Microsoft empfohlen wird. Diese Laufzeitumgebung wird ständig aktualisiert und beinhaltet viele nützliche Tools von Drittanbietern. Eines davon ist das Add-In Gerüsterstellungstool Yeoman Generator von Yeoman.[10] Im nächsten Kapitel wird dieses näher erläutert. Das Add-In kann unter Node.js auch in der Programmiersprache TypeScript geschrieben werden. Da jedoch die meisten Add-In Funktionsbeispiele von Microsoft in JavaScript verfasst sind, wurde diese Programmiersprache zusammen mit der Node.js Laufzeit auch für das Outlook Add-In verwendet. Diese kann von der offiziellen Webseite heruntergeladen und installiert werden.[11]

Zusammen mit Node.js wird das Paket Manager Tool npm automatisch mit installiert. Es ist für das Nachladen von Paketen zuständig, die für die Office-Add-Ins verwendet werden.[12] Es ist ratsam, das Tool auch regelmäßig manuell durch den Terminal Befehl „*npm install npm -g*“ im aktuellen Arbeitsverzeichnis zu aktualisieren.

Zum Schluss ist es notwendig einen beliebigen Code-Editor installiert zu haben. Für diese Arbeit wurde der Code-Editor Visual Studio Code verwendet.[13]

[10] Vgl. https://yeoman.io/ (23.06.2022).
[11] Vgl. https://nodejs.org/en/ (23.06.2022).
[12] Vgl. https://www.npmjs.com/ (23.06.2022).
[13] Vgl. https://docs.microsoft.com/de-de/office/dev/add-ins/overview/set-up-your-dev-environment?tabs=yeomangenerator (23.06.2022).

4.2 Erstellung der Add-In Basis

Das Grundgerüst eines Office Add-Ins kann mithilfe des, von Microsoft empfohlenen, Yeoman Generator generiert werden. Dieser kann auch mit „Yo Office“ abgekürzt werden. Es ist ein interaktives Node.js basierendes Terminal Befehlswerkzeug, welches Office Add-In Entwicklungsprojekte anlegt. Diese Projekte beinhalten für die Entwicklung notwendige Skripte zum Bauen des Projekts für die spätere Nutzung in der Produktion.
Eine weitere wichtige Funktion, die der Yeoman Generator mit sich bringt, ist das Starten eines lokalen Add-In Servers. Dieser stellt die Add-In Funktionalität für die Clientseite, in diesem Fall in Outlook, zur Verfügung. Dabei wird das Add-In automatisch in der jeweiligen Office Umgebung für den Nutzer installiert.[14] Dieser Server ist nur für die lokale Entwicklung und zum Testen geeignet. Für die Produktion wurde ein Server auf der Azure Web Apps Plattform gemietet. Azure ist eine Plattform, die verschiedene Computerressourcen online zu Verfügung stellt. Mehr dazu in Kapitel 5.

Der Yeoman Generator wird mit der Befehlszeile „*npm install -g yo generator-office*“ im Arbeitsverzeichnis installiert. Microsoft empfiehlt hierbei, das Tool regelmäßig mit Hilfe der Befehlszeile zu aktualisieren.[15] Nach Ende der Installation kann die Generierung des Projektes im gleichen Verzeichnis mit dem Befehl „*yo office*“ im Terminal gestartet werden. Das Programm schlägt mehrere Optionen zur Auswahl vor. Die zwei Möglichkeiten, die für die Outlook Add-In Entwicklung in Betracht gezogen wurden, sind *Office Add-in Task Pane project* und das *Office Add-in Task Pane project supporting single sign-on*. Im Gegensatz zu der ersten Möglichkeit, bietet die Single sign-on -unterstützende Variante, zusätzliche Funktionen und eine Bibliothek für eine sichere Authentifizierung im Microsoft Konto. Das Single-Sign-On ist eine Authentifizierungsmethode. Übersetzt steht das Single-Sign-On, abgekürzt mit SSO, für eine einmalige Anmeldung. Es gewährt dem Nutzer mit nur einer Anmeldung den Zugriff auf mehrere Webressourcen. Dieses Verfahren wird im Kapitel 4.5.3Authentifizierung genauer erläutert. Eine spätere im Kapitel 4.3.5 durchgeführte Untersuchung hat gezeigt, warum genau die von Yeoman generierte SSO Unterstützung im Fall dieses Projektes ungeeignet ist. Für die Entwicklung wurde die ersteMöglichkeit ausgewählt. In den folgenden Schritten muss sich der Entwickler zwischen den Programmiersprachen JavaScript und TypeScript entscheiden. Aus den, im letzten Unterkapitel genannten Gründen, wurde für die Entwicklung die Programmiersprache JavaScript verwendet. Danach vergibt man einen Namen an das Add-In. Im letzten Schritt wird eines der Office Programme gewählt, in der das Add-In laufen soll. Dafür stehen Excel, OneNote, Outlook, PowerPoint, Project und Word zur Auswahl. Im Gegensatz zur Outlook, kann die Art des Add-Ins für alle anderen Programme leicht ausgetauscht werden. So kann ein generiertes Word Add-In Grundgerüst zu einem PowerPoint Add-In umgeschrieben werden. Nachdem alle Angaben bestätigt wurden, generiert Yeoman Generator ein einfaches und lauffähiges Outlook Add-In. Das Add-In kann dann sofort mit dem Terminal Befehl „*npm start*“ lokal ausgeführt werden.[16]

[14] Vgl. https://docs.microsoft.com/en-us/office/dev/add-ins/develop/yeoman-generator-overview (24.06.2022).
[15] Vgl. https://docs.microsoft.com/en-us/office/dev/add-ins/develop/yeoman-generator-overview#install-the-generator (24.06.2022).
[16] Vgl. https://docs.microsoft.com/en-us/office/dev/add-ins/quickstarts/outlook-quickstart?tabs=yeoman-generator (24.06.2022).

4.3 Projektstruktur und dessen Bestandteile

4.3.1 Manifest

Das Projekt besteht aus mehreren Dateien, die mithilfe des Yeoman Generator erstellt wurden. Die Entwicklung beginnt mit der XML-Manifestdatei, in der die Eigenschaften des Add-Ins und dessen Anzeigeort deklariert werden. Genauer betrachtet, beinhaltet eine XML-Manifestdatei folgende Bestandteile: zuerst die klassischen Eigenschaften wie eine ID, eine Version, die Bezeichnung des Add-Ins, seine Beschreibung und der Aufbewahrungsort des Add-Ins. Im nächsten Kapitel wird das Office Programm, in diesem Fall Outlook, bestimmt, in dem das Add-In ausführbar sein soll. Gleich darunter kann die Standardabmessungen des Anzeigefensters angegeben werden.[17] Darauffolgend muss unter der „*Mailbox MinVersion*“ die benötigte JavaScript API Mindestanforderung an die Version angegeben werden. Es werden immer neue Office API-Versionen entwickelt, dessen Struktur sich manchmal von den vorherigen unterscheidet. Die Version der API selbst ist von der benutzten Outlook Version abhängig. Jede Funktion ist in der Microsoft Dokumentation mit einer minimalen JavaScript Office API-Version versehen. Man muss also die höchste Version von den in Add-In benutzten Grundfunktionen in Manifest angeben, um die minimale Funktionalität zu gewährleisten.[18] Weiterhin besteht die Möglichkeit, optionale Funktionen aus der höheren JavaScript Office API-Version zu benutzen, wenn die aktuell benutzte Outlook Version es erlaubt. In den *VersionOverrides* Element kann die Funktionalität deklariert werden, die von der Minimalversion nicht unterstützt werden. Die zurzeit unterstütze Version kann beim Ausführen des Add-Ins abgefragt werden. Das ermöglicht dem Entwickler, eine bestimmte Funktion für die verschiedenen API-Versionen explizit anzupassen. Somit kann das Add-In rückwärtskompatibel programmiert werden.[19]

Danach kann man den *ExtensionPoint*, übersetzt den Erweiterungspunkt, bestimmen. Darunter versteht man die Stelle, an welcher das Add-In aufgerufen werden soll. Im Outlook gibt es zwei Orte, in denen das Add-In eingebaut werden kann. Einmal in der E-Mail-Abteilung mit dem Erweiterungspunkt für das Verfassen einer Nachricht *MessageComposeCommandSurface* und für das Lesen einer Nachricht mit dem Erweiterungspunkt *AppointmentReadCommandSurface*. Beim Verfassen einer Nachricht kann ein Office Add-In dafür benutzt werden, den geschriebenen Text zu analysieren und automatisch bestimmte Textblöcke einzufügen. Die zweite Stelle, an welcher das Add-In im Outlook eingebaut werden kann, ist der Kalender. Hier kann das Add-In in der oberen Menüleiste beim Erfassen eines Kalendereintrags eingeblendet werden. Dieser kann separat für den Terminorganisator mit dem Erweiterungspunkt *AppointmentOrganizerCommandSurface* selbst, sowie für alle im Termin angegebenen Teilnehmer, mit dem Erweiterungspunkt *AppointmentOrganizerCommandSurface* angezeigt werden. Diese Erweiterungspunkte stehen ebenso für mobile Geräte zur Verfügung. Ein weiterer Punkt, welcher als

[17] Vgl. https://docs.microsoft.com/en-us/office/dev/add-ins/develop/add-in-manifests?tabs=tabid-1 (25.06.2022).
[18] Vgl. https://docs.microsoft.com/en-us/javascript/api/requirement-sets/outlook/outlook-api-requirement-sets?view=common-js-preview (25.06.2022).
[19] Vgl. https://docs.microsoft.com/en-us/javascript/api/manifest/versionoverrides?view=common-js-preview (25.06.2022).

Erweiterungspunkt zählt, ist das Abfangen von Ereignissen. [20] Mit Hilfe von Ereignissen können bestimmte Funktionen ausgeführt werden, ohne dass das Add-In explizit gestartet werden muss. Ereignisse, die für das Outlook Add-In in Frage kommen, sind das *OnAppointmentTimeChanged*, welches bei einer Zeitänderung am Termin eine gewünschte Funktion ausführt, und das *OnNewAppointmentOrganizer*, welches beim Anlegen eines neuen Termins reagieren wird. Es gibt allerdings noch weitere Ereignisse, die abgefangen werden können. Beim Erfassen einer Nachricht, beim Ändern von E-Mail-Empfängern oder der Termin-Teilnehmer, beim Hinzufügen von Anhängen bei einer E-Mail, im Kalendereintrag, beim Bearbeiten einer Terminserie und beim Speichern oder Abschicken der Nachricht oder des Kalendereintrages. Die Funktionalität selbst muss in einer anderen JavaScript Datei implementiert werden. [21]

Ein weiteres Kapitel der XML-Manifestdatei ist die Verlinkung von Dateien. In der XML-Manifestdatei werden alle implementierten Benutzeroberflächen direkt verlinkt. Die Verlinkungen der restlichen Dateien erfolgt in der *webpack.config* Datei, die sich im Add-In Verzeichnis befindet. Genau werden in der XML-Manifestdatei alle benutzten Grafiken sowie die Logos in verschiedenen Auflösungen verlinkt, die unter anderen im Auswahlmenü und in der Add-In Benutzereinstellung erscheinen.

Zum Schluss können die Berechtigungen zum Lesen und Schreiben von Daten innerhalb der benutzten Umgebung für das Add-In erteilt werden. Die Office-API kann nur eine eingeschränkte Menge von Informationen über den aktuellen Benutzer geben, die in dem Kapitel *Untersuchung der Microsoft Office API* bereits erwähnt wurde. Um auf weitere Nutzerdaten zugreifen zu können, stellt Microsoft die „Microsoft Graph API“ zu Verfügung. Das Add-In muss zuerst in Microsoft Verwaltungsportal Azure registriert werden und über eine funktionierende SSO Authentifizierung verfügen, um auf das Microsoft Graph zugreifen zu können. Dieses Verfahren wird in Kapitel 4.5.3 näher betrachtet. Nachdem das Add-In in Azure angemeldet wurde, kann das Add-In mit den registrierten Azure Daten inklusive der benötigten Berechtigungen bei Microsoft Graph, unter dem letzten Abschnitt *WebapplicationInfo* versehen werden.[22]

4.3.2 Node Modules und package.json

Dieses Verzeichnis beinhaltet alle Pakete und Bibliotheken, die durch den npm Befehl runtergeladen wurden und im Add-In benutzt werden. Das Verzeichnis beinhaltet eine große Menge von einzelnen kleinen Dateien, was für erhebliche Wartezeiten sorgt, wenn man das Verzeichnis hochladen oder kopieren möchte. Deshalb ist es ratsam, in einen Versionsverwaltungswerkzeug wie Git, das Verzeichnis *node_modules* auszuschließen. Bei jeglicher Übertragung ist es dringend empfohlen, das Verzeichnis komplett neu herunterzuladen. In den Dateien *package.json* und *package-lock.json* befindet sich eine Liste aller Pakete, die von npm in das Verzeichnis *node_modules* mit dem Terminal Befehl „*npm install*“ neu geladen werden können.

---

[20] Vgl. https://docs.microsoft.com/de-de/javascript/api/manifest/extensionpoint?view=common-js-preview (25.06.2022).
[21] Vgl. https://docs.microsoft.com/en-us/office/dev/add-ins/outlook/autolaunch (25.06.2022).
[22] Vgl. https://docs.microsoft.com/en-us/javascript/api/manifest/webapplicationinfo?view=common-js-preview (25.06.2022).

4.3.3 Source Verzeichnis

In dem „*src*“, abgekürzt von Source, Verzeichnis befinden sich die Quelldateien des Add-Ins. Die wichtigste davon ist die Datei mit den Namen Taskpane. Diese lässt sich auf drei Dateien aufspalten. Die taskpane.html ist für die Darstellung der Hauptoberfläche, die taskpane.css für die Formatierung und die taskpane.js für die Funktionalität verantwortlich. In der Letzteren findet die meiste Programmierung statt. Weiterhin gibt es innerhalb des Source Ordners das *commands* Verzeichnis, in welchem die im Outlook abgefangene Ereignisse behandelt werden können. In das entwickelnde Outlook Add-In wurde noch zusätzlich das Verzeichnis „jobdialog“ angelegt, wo das externe Fenster für die Projektnummernsuche implementiert ist.

4.3.4 Build Verzeichnisse

Um die einzelnen Add-In Dateien für die Produktion zusammen bauen zu können, wird der statische Modul-Builder *webpack* verwendet. Dieses ist ein auf der JavaScript Sprache basierendes Werkzeug, welches alle von dem Add-In benötigten Dateien mit Hilfe eines Abhängigkeitsgraphen kombiniert.[23] An dieser Stelle werden alle für den Build benötigten Dateien, zusammen mit den Grafiken aus den *assets* Verzeichnis, angegeben. Sobald das Projekt mit Hilfe des Terminal *npm build* Befehls gebaut wurde, entsteht ein neues Verzeichnis namens *dist*, in welchem die finale Add-In Version entsteht, die für die Produktion genutzt werden kann.

4.3.5 SSO Unterstützung

Dieser Kapitel beschreibt den Einbau der SSO Funktionalität für die Produktions-Variante des Add-Ins. Wie es bereits in Kapitel 4.2 erwähnt wurde, eignet sich die von Yeoman generierte Outlook Add-In Variante für das Add-In nicht. Bei der SSO Variante legt der Yeoman Generator ein zusätzliches *helpers* Verzeichnis neben der Taskpane im Source Ordner an. Es besteht aus mehreren JavaScript Dateien die folgende Funktion erfüllen: als Erstes wird ermittelt, ob der Nutzer bereits angemeldet ist. Falls das nicht der Fall ist, wird ein Dialogfenster erscheinen, in welchem der Benutzer sich mit seinem Microsoft Konto anmelden kann. Bei einer erfolgreichen Anmeldung empfängt das Add-In ein Access Token und macht damit eine Anfrage an Microsoft Graph, welcher einfache Nutzer Profildaten zurückgibt. Ein Access Token ist ein elektronischer Zugriffsschlüssel, der für eine Authentifizierung statt dem klassischen Namen und Passwort genutzt werden kann. Mehr dazu im Kapitel 4.5. Schließlich werden die Daten im Textblock des Kalendereintrags ausgegeben.

Für eine erfolgreiche Authentifizierung muss das Add-In zuerst im Microsoft Azure Portal angemeldet werden, mehr dazu in Kapitel 5. Nach der Registrierung erhält das Add-In einmalig ein *Client Secret*, ein Geheimschlüssel, der dem Add-In zugeordnet ist. Dieser

[23] Vgl. https://webpack.js.org/concepts/ (26.06.2022).

Schlüssel muss auf der Serverseite des Add-Ins gespeichert werden und wird für das Erhalten von Access Token verwendet. Das Problem ist, dass *Client Secret* durch das Yeoman Generator unter Windows interne Anmeldeinformationsverwaltung gespeichert wird.[24] An dieser Stelle befinden sich üblicherweise die im Browser gespeicherten Passwörter für verschiedene Webseiten. Der Add-In Produktionsserver besitzt ein derartiges Verzeichnis nicht. Es wurde versucht die Stelle im Programmcode zu finden, welche für das Abrufen des *Client Secrets* verantwortlich ist. Es stellte sich heraus, dass dies direkt in der *office-addin-sso* Programmbibliothek stattfindet, auf die das generierte Add-In zugreift. Diese Bibliothek befindet sich im bereits erwähntes *node_modules* Verzeichnis. Eine übliche Programmbibliothek beinhaltet eine Sammlung von bereits fertigen Unterprogrammen, die für eine komplexere Softwareentwicklung verwendet werden kann. Die Bibliotheken sind fest und können normalerweise nicht extern editiert werden.

Als Alternative bietet Microsoft selbst ein SSO fähiges Beispiel Add-In mit einer Programmiereinleitung vor.[25] Es beinhaltet die in der *office-addin-sso* Bibliothek vorkommenden Funktionen, die für die Entwicklung kritisch sind. Dieses Beispiel Add-In der 2.1 Version wurde als Grundlage für die SSO fähige Add-In Variante verwendet.[26] Es besteht aus zwei Schichten: aus dem Add-In-Client und aus der Add-In-Server Zwischenanwendung. Die Zwischenanwendung, auch mit Add-In-Server abgekürzt, wird auf einen von COSMO CONSULT verwalteten Azure Server ausgeführt. Dort kann der *Client Secret* Schlüssel, neben den anderen für die Sicherheit relevanten Add-In Daten, sicher gespeichert werden. Im Outlook selbst wird nur der Add-In-Client ausgeführt. Die dazu gehörigen Dateien befinden sich im *public* Verzeichnis. Da das Add-In Beispiel nur für Excel, Word und PowerPoint gedacht war, musste die Manifest Datei komplett überarbeitet werden. Die Hauptseite des Add-In-Clients wurde ebenfalls entfernt und von neu entwickelt. Die Implementation des Add-In-Clients wird innerhalb des nächsten Kapitels 4.4 ausführlich beschrieben. Die SSO Add-In Variante verwendet den im letzten Kapitel 4.3.4 beschriebenen Modul-Builder *webpack* nicht. Stattdessen wurde das Add-In mithilfe eines Node.js Anwendungsframeworks namens *Express*[27] gebaut. Der genaue Aufbau der Add-In-Server Zwischenanwendung wird in der *app.js* Datei geregelt. Alle weiteren Funktionen, die nur innerhalb der Zwischenanwendung ausgeführt werden, befinden sich in der *authRoute.js* Datei innerhalb des *routes* Verzeichnisses. Dessen Funktionalität wird im Kapitel 4.5 Authentifizierung beschrieben.

[24] Vgl. https://docs.microsoft.com/de-de/learn/modules/office-add-ins-sso/7-exercise-outlook-sso (26.07.2022).
[25] Vgl. https://docs.microsoft.com/en-us/office/dev/add-ins/develop/create-sso-office-add-ins-nodejs (29.07.2022).
[26] https://github.com/OfficeDev/Office-Add-in-samples/tree/main/Samples/auth/Office-Add-in-NodeJS-SSO (15.06.2022).
[27] https://expressjs.com/ (29.07.2022).

4.4 Task Pane und dessen Funktionalität

Das Hauptfenster des Add-Ins wird, durch das Betätigen der dazugehörigen Schaltfläche in der oberen Menüleiste, rechts neben den Kalendereintrag eingeblendet. Dazu siehe *Abbildung 1*: Outlook Add-In im Browser (Rechts) . Das Hauptfenster trägt allgemein den Titel *Taskpane*, übersetzt Aufgabenbereich. Wie im Kapitel 4.3.3 erwähnt, zählt dieser Bereich zu den wichtigsten innerhalb des Add-Ins. Die Datei *taskpane.html* beschreibt zusammen mit der *taskpane.css*, die im Kopf der *taskpane.html* Datei deklariert wird, das äußere des Add-Ins. Damit das Add-In in die Produktion gebracht werden kann, müssen alle benötigten Ressourcen wie die *office.js* Bibliothek, im Kopf der Datei angegeben werden. Im Körperbereich befindet sich die eigentliche Oberfläche mit Textfeldern und dessen Beschriftungen so wie die Schaltflächen zum Speichern und Löschen von angegebenen Projektdaten. Unter dem Körper können die benutzten Java Scripte verlinkt werden. Es ist wichtig zu erwähnen, dass die *taskpane.js* Datei hier nicht verlinkt werden muss, wenn es bereits in *der webpack.config.js* eingefügt wurde. Es kann sonst dazu führen, dass die Datei mehrmals eingebunden wird. Die *taskpane.js* Datei besteht zum Teil aus Funktionen, die auf die Ereignisse der *taskpane.html* reagieren. Beim zweifachen Anbinden werden diese Funktionen somit mehrfach ausgeführt. Die folgenden Kapitel beschreiben die Funktionalität der *taskpane.js* Datei und die dazu gehörigen Suchfenster.

4.4.1 Initialisierung von CustomProperties

Zu Beginn muss der Office API-Status ermittelt werden. Die Funktion *Office.onReady()* signalisiert die Bereitschaft der API. Sobald diese Funktion ein Rückgabewert liefert, ist die API bereit. Zuerst wird die Liste der Mandanten gefüllt, aus denen der Benutzer einen auswählen muss. Alle im Add-In angegebenen Daten werden nicht nur zum Data Warenhouse (weiter als DWH abgekürzt) geschickt, sondern auch direkt am Termin gespeichert. So kann man die angegebenen Daten mit denen aus der DWH abgleichen, um ungespeicherte Daten zu erkennen. Dafür stellt Office das *CustomProperties* Interface zur Verfügung. Ein *CustomProperties* Objekt stellt benutzerdefinierte Eigenschaften dar, die an den bestimmten Elementen, in diesem Fall an einem Kalendereintrag, gespeichert werden können. Dieses Objekt kann nur in einen Outlook Add-In verwendet werden. Die Größe eines Eintrags darf nicht größer als 2500 Zeichen sein.[28] Dieses Interface bietet das Abrufen, Setzen, Löschen und Speichern von Eigenschaften in Form eines JSON Eintrages an. JSON, abgekürzt von „JavaScript Object Notation“, ist ein kompaktes Datenformat, welches oft für den Datenaustausch zwischen verschiedenen Anwendungen benutzt wird.[29] Während der Initialisierung lädt das Add-In alle gespeicherten Eigenschaften und schreibt diese in die Textfelder. Sobald ein Textfeld mit anderen Daten befüllt wird, speichert das Add-In den vorher geprüften Inhalt sofort in ein *CustomProperties* Objekt. Der ausgewählte Mandant muss für alle im Folgenden erstellten Kalendereinträge gelten. Da die *CustomProperties* nur an den spezifischen Terminen abrufbar sind, sind sie für diese Zwecke nicht geeignet. Das Office RoamingSettings Interface ermöglicht das Speichern von kleineren Datenmengen bis zu 32KB pro Add-In und pro Benutzer. Dieses Interface wird im Falle des in dieser Arbeit behandeltem Add-In für das Speichern von Mandanten

[28] Vgl. https://docs.microsoft.com/en-us/javascript/api/outlook/office.customproperties?view=outlook-js-preview (02.07.2022).
[29] Vgl. https://www.json.org/json-en.html (02.07.2022).

verwendet. Dies sollte nicht als sicherer Speicherort betrachtet werden, da dessen Inhalte auch von anderen Webdiensten abrufbar sind.[30]

4.4.2 Termin Identifikationsnummer

Das nächste benötigte Datum ist die Identifikationsnummer des Termins. Diese Nummer ist eine der spezifischen Kalender Informationen, die von der Office Schnittstelle *Office.context.mail.item*, weiter als *item* abgekürzt, zur Verfügung gestellt werden. Diese Schnittstelle ist Kontextsensitiv und kann beim Lesen einer Nachricht, sowie beim Anlegen und Auslesen eines Kalendereintrages benutzt werden. So liefert uns zum Beispiel *item.subject* den Titel einer Nachricht oder eines Kalendereintrags je nachdem, wo das Add-In laut der Manifest Datei angebunden ist.

Die Identifikationsnummer des Termins, kann nun durch das *Item.itemId* abgerufen werden. Dieser Wert ist nur bei den bereits angelegten Kalendereinträgen belegt. Dafür muss der Eintrag vor dem abrufen der Identifikationsnummer zwischendurch per *item.saveAsync* gespeichert werden. Die Identifikationsnummer ist eine lange Kette aus Zahlen, Buchstaben und Sonderzeichen. Eins davon ist der Querstrich „/“. Die Projektdaten werden mittels der HTTP Webanfragen an das DWH versendet und empfangen. Die Anfrage URLs besitzen die folgende Form:

https://..../appointments/{userId}/{appointmentId}

wobei die *userId* für die Benutzer-E-Mail und die *appointmentId* für die Termin Identifikationsnummer steht. Sobald die *appointmentId* einen unerwarteten Querstrich besitzt, kann das System die Anfrage nicht mehr richtig bearbeiten, da die entstehende URL auf das nicht existierende Verzeichnis zeigen wird. Dieses Problem kann mittels der Funktion *Office.context.mailbox.convertToRestId* behoben werden. Es ersetzt alle vorkommenden Querstriche auf das Minuszeichen. Die Kalendereinträge können auch in der Grafischen Oberfläche mit allen *CustomProperties* kopiert werden. Dabei entsteht eine neue Termin Identifikationsnummer. Wenn man diese mit der in *CustomProperties* abgleicht, kann man eine in DWH ungespeicherte Kopie erkennen.

4.4.3 HTTP Webanfragen

Sobald das Add-In die *CustomProperties* geladen und den Kalendereintrag auf eine Kopie geprüft hat, findet die erste Anfrage an das DWH statt. Die Anfrage wird mittels der XMLHttpRequest Bibliothek ausgeführt. Die bereits weiter oben beschriebene URL wird zusammen mit einem Header mittels eines HTTP GET Befehls angefragt. Während der Taskpane Entwicklungsphase wurden alle Anfragen nur mit den *Api-Subscription-Key* im Header ausgeführt. Der *Api-Subscription-Key* ist ein weiterer digitaler Schlüssel, der von der Schnittstelle zum DWH ausgestellt und für die Authentifizierung verwendet werden kann. Dieses Verfahren wird im Kapitel API Management genauer beschrieben.

[30] Vgl. https://docs.microsoft.com/en-us/javascript/api/outlook/office.roamingsettings?view=outlook-js-preview (02.07.2022).

Sobald die SSO Authentifizierung fertig war, wurde zusätzlich ein JWT-Token verwendet. Mehr dazu im Kapitel Authentifizierung und in Datenfluss zwischen Outlook Add-In und Data Warenhouse erfahren.

Nachdem eine Anfrage bearbeitet wurde, sendet das DWH eine Antwort zurück. Die später im Kapitel 5.2.2 beschriebene Azure Logic App, welche eine Schnittstelle zwischen dem Add-In und dem DWH darstellt, sendet einen Status Code, und beim Erfolg den angefragten Inhalt, an das Add-In. Die Logic App unterstützt mehrere Statuscodes, die im Add-In auf bestimmte Weise abgearbeitet werden:

Tabelle 2: Statuscodes

Statuscode	Tätigkeit
200	Das DWH hat den angefragten Termin gefunden und im JSON Format wieder zurückgegeben. Die Projektdaten werden in die Textfelder eingegeben. Die zurück gegebene UTC Zeit wird auf die lokale Zeit umgewandelt, formatiert und in der Add-In Oberfläche ausgegeben. Der Benutzer kriegt eine Mitteilung, sobald die im DWH gespeicherte Zeit, mit der im Kalendereitrag angegebene Zeit nicht übereinstimmt und fordert den Benutzer zum zusätzlichen Speichern auf, falls die Zeitänderungsevents nicht abgefragt werden.
204	Das DWH hat den Angefragten Termin nicht gefunden. Es erfolgt eine Nutzerbenachrichtigung je nachdem ob der Kalendereintrag ungespeichert oder eine Kopie ist.
401	Die Authentifizierung ist im API Manager vor der Logic App gescheitert.
404	Keine Internetverbindung oder eine der Schnittstellen ist nicht erreichbar.

4.4.4 Zeitermittlung

Die Start- und Endzeit sind die wichtigsten Eigenschaften eines Kalender Eintrages.
Das Outlook Add-In sollte theoretisch im Web, als auch auf der Desktop Variante gleiche Funktionalität anbieten. Während der Entwicklung stellte es sich heraus, dass die *item.start* Schnittstelle nur in der Outlook Webvariante die erforderliche Startzeit des Kalendereintrages direkt beinhaltete. Bei der Desktopvariante entgegen bietet das Item Objekt die Prozeduren *getAsync()* und *setAsync()* an, um die Startzeit zu setzten oder auszulesen. Auch laut der Microsoft Dokumentation sollte das *item.start* Objekt die zwei Prozeduren beinhalten, statt das Zeitobjekt an sich.[31] Um das Problem mehr auf den Grund zu gehen, wurde die bereits in dem Kapitel Manifest beschriebenen requirement sets der Office API überprüft. Der COSMO CONSULT Exchange Server unterstützt stehts die aktuellste Set Version, die sowohl im Web, als auch auf der Desktop Variante ankommt. Nach einer weiteren Recherche fand sich die Ursache. Auf der Desktop Variante lässt sich das Add-In nur in den Bearbeitungsmodus eines Kalendereintrags öffnen. In der Webvariante hingegen gibt es zusätzlich eine Leseansicht, wo sich das Add-In auch starten lässt. Dieses Verhalten tritt nur auf, wenn das Add-In auch für die Terminteilnehmer in der Manifest Datei freigeschaltet wird. Ansonsten kann man das Add-In nur im Bearbeitungsmodus im Web aufrufen. Da es sich um eine Leseansicht handelt, kann die Zeit direkt aus dem *Item.start* als eine Eigenschaft gelesen werden. Innerhalb der Leseansicht kann man die „Bearbeiten“ Schaltfläche betätigen, um den geöffneten Kalendereintrag editieren zu lassen. In der Bearbeitungsansicht besitzt das Add-In im Web nun die zum

[31] Vgl. https://docs.microsoft.com/en-us/javascript/api/outlook/office.appointmentcompose?view=outlook-js-preview&preserve-view=true#outlook-office-appointmentcompose-start-member (03.07.2022).

Desktop identische Office API. Der Grund dafür ist, dass im Bearbeitungsmodus sowohl das Add-In, als auch der Nutzer gleichzeitig die Zeit ändern könnte. Deswegen muss die Zeit in diesen Fall asynchron abgerufen werden.[32]

Um diese API-Ungleichheiten abzufangen, schlägt Microsoft eine universale Lösung vor.[33] Es kann abgefangen werden, ob das *item.start.getAsync* definiert ist. Falls es undefiniert ist, dann weist es darauf hin, dass das Add-In im Lesemodus geöffnet wurde und die *item.start* Eigenschaft kann direkt ausgelesen werden. Falls das nicht der Fall ist, dann wird der Rückgabewert von der *item.start.getAsync* Prozedur direkt ausgelesen.

Es ließ sich keine passende Standard JavaScript Funktion finden, die das von COSMO CONSULT ERP System verwendete Zeitformat nachbilden konnte.

<Jahr>-<Monat>-<Tag>T<Stunde>:<Minute>:<Sekunde>Z
Beispiel: 2022-07-14T03:35:00Z

Die von der Office API zur Verfügung gestellte UTC Zeit wurde auf einzelne Teile zerlegt und in der richtigen Form konkateniert. Dabei mussten an den einstelligen Zahlen je eine zusätzliche Null rangehängt werden. Da der Kalender die Zeit nur in Stunden und Minuten speichert, wurden für den letzten Sekundenwert zwei Nullen ergänzt.

4.4.5 Behandlung der Asynchronität

Die meisten Office API-Aufrufe funktionieren asynchron. Es bedeutet, dass die Aufrufe zeitlich nicht koordiniert sind. Eine Webanwendung soll sich möglichst nicht aufhängen, deswegen werden die Anwendungen asynchron gebaut. Solche Funktionen werden parallel zu dem restlichen Programmablauf abgearbeitet und halten beim Absagen die restliche Anwendung nicht auf. Vorteilhaft können mehrere asynchrone Funktionen gleichzeitig abgearbeitet werden, doch man kann nicht vorhersehen, wann diese mit der Abarbeitung fertig werden. Das Problem wurde beim Abschicken von Daten zum DWH deutlich. Das Add-In schickte die Daten ab, bevor die Zeit asynchron überhaupt ermittelt werden konnte. Das Problem ließ sich mit mittels der *Office.Promise* Funktion lösen. Dies ist die vereinfachte *getStartTime* Funktion:

[32] Vgl. https://docs.microsoft.com/en-us/office/dev/add-ins/outlook/get-or-set-the-time-of-an-appointment (03.07.2022).
[33] Vgl. https://docs.microsoft.com/en-us/javascript/api/requirement-sets/outlook/outlook-api-requirement-sets?view=common-js-preview#using-apis-from-later-requirement-sets (03.07.2022).

```
function getStartTime() {
    return new Office.Promise(function (resolve, reject) {
        Office.context.mailbox.item.start.getAsync(function (asyncResult) {
            if (asyncResult.status !== Office.AsyncResultStatus.Failed) {
                appointmentTime(asyncResult.value, 'start');
                resolve(asyncResult.value);
            } else {
                reject(console.log(asyncResult1.error.message));
                return false;
            }
    });
}
```

Abbildung 7: Asynchrone getStartTime Funktion

Dabei baut die *appointmentTime()* Funktion den richtigen Zeitstring. Die Funktion muss beim Erfolg ein *resolve* oder beim Misserfolg ein *reject* Parameter zurückgeben. Die *getStartTime()* Funktion muss beim Aufrufen das Schlüsselwort *await* vorstehen haben. Um diese Synchronisationsmethode nutzen zu können, muss die Funktion, in der das *getStartTime()* gerufen wurde, selbst wiederum eine asynchrone Funktion sein.

4.4.6 Projektnummer-Liste

Die Projektnummer-Liste ist eine große Datei, welche alle Projektnummern von COSMO CONSULT beinhaltet. Das Laden einer solch großen Datei kann bis zu einer halben Minute dauern. Früher wurden die Projektnummern manuell nachgeladen und lokal abgelegt, doch im Web müssen die Projektnummern nun jedes Mal von neu geladen werden. Deswegen werden nur die aktuell laufenden Projektnummern, die zum von Benutzer ausgewählten Mandanten gehören, durch die Logic App beim DWH angefragt und an das Add-In zurückgegeben. Somit reduziert sich die Ladezeit auf 4 bis 12 Sekunden, je nachdem wie sehr das DWH im Moment belastet ist. Dieses Zeitintervall ist aus der Sicht der Benutzer immer noch lang. Deswegen werden die Projektnummern nicht im Suchfenster, sondern bereits beim Starten des Add-Ins sofort asynchron nachgeladen.

4.4.7 Dialog Fenster

Das Add-In verfügt über zwei Dialog Fenster. Das erste dient der einmaligen Anmeldung im Microsoft Konto und das zweite Fenster ermöglicht eine tabellarisch implementierte Auswahl der Projektnummern. Das zweite Fenster wird mit Hilfe der Office-Dialog-API[34] erzeugt und ist unter *Abbildung 2: Add-In Dialogfenster* zu sehen. In *Abbildung 8: Erzeugung eines Dialogfensters* wird das Aufrufen eines Fensters implementiert. Im erzeugten Dialogfenster kann eine weitere auf HTML basierende Seite angezeigt werden, die sich aus Sicherheitsgründen zwangsläufig auf derselben Domäne befinden muss.

[34] Vgl. https://docs.microsoft.com/de-de/office/dev/add-ins/develop/dialog-api-in-office-add-ins (15.07.2022).

```
function openDialog() {
    const url = `${window.location.origin}/src/jobdialog/jobdialog.html`;
    Office.context.ui.displayDialogAsync(url, { height: 61, width: 30, displayInIframe: true }, function (result) {
        if (result.status !== Office.AsyncResultStatus.Succeeded) {
            console.error(`open Dialog failed with message ${result.error.message}`);
            dialog = result.value;
            dialog.close();
        } else {
            console.log(`Dialog opened with status: ${result.status}`);
            dialog = result.value;
            //Dialog window message recieve eventhandler:
            dialog.addEventHandler(Office.EventType.DialogMessageReceived, processMessage);
        }
    });
}
```

Abbildung 8: Erzeugung eines Dialogfensters

Im *Office.context.ui.displayDialogAsync()* kann außer der Fenstergröße das Attribut *displayInframe* gesetzt werden. Dieses Attribut führt dazu, dass das Dialogfenster als unverankerter iFrame statt als separates Fenster geöffnet wird. Das erhöht die Geschwindigkeit, mit der das Fenster im Web gebaut wird. Ohne dieses Attribut fragt Outlook bei jedem Dialogfenster Aufruf nach, ob der Nutzer ein externes Fenster wirklich öffnen möchte. Auf die Desktop Outlook Variante hat es allerdings kein Einfluss und kann weggelassen werden, falls das Add-In nicht webfähig sein sollte.

Eine weitere wichtige Eigenschaft eines Dialogfensters ist, dass es keine gemeinsamen globalen Variablen mit der rufenden Taskpane besitzt. Allerdings können Nachrichten zwischen dem Dialog- und Hauptfenster ausgetauscht werden. Einer der zwei Office API-Aufrufe, die das Dialogfenster benutzen darf, ist das *messageParent*. Es kann eine beliebige Zeichenfolge an das Hauptfenster versenden. Damit das Hauptfenster eine Nachricht empfangen kann, muss es ein Event Händler implementiert haben, welcher in Abbildung 8: Erzeugung eines Dialogfensters, in der untersten Zeile, deklariert wurde.

Das Hauptfenster kann die Nachrichten nicht nur empfangen, sondern auch an das Dialogfenster mit Hilfe von *messageChild* versenden. Die im vorherigen Unterkapitel beschriebene Projektnummer-Liste sollte im Dialogfenster angezeigt werden. Das Problem ist, dass das rufende Hauptfenster nicht weiß, ob das Dialogfenster bereits initialisiert ist oder nicht. Die Daten wurden versendet bevor das Dialogfester in der Lage war es zu empfangen und kamen deswegen gar nicht an. Man kann Laut der Microsoft Dokumentation das Ereignis abfangen, um zu sehen ob die durch das Hauptfenster versendete Nachricht vor dem im Dialogfenster initialisierten Event Händler ankam oder nicht. [35] Das löst allerdings das Problem nicht.

Um das Problem zu beheben, wurde der sogenannte Drei-Wege-Handschlag eingebaut. Es wird zuerst das Dialogfenster ohne Daten aufgerufen. Sobald das *Office.onReady* Ereignis ausgelöst wurde, sendet das Dialogfenster eine Bestätigungsantwort an das Hauptfenster. Das Hauptfenster prüft, ob die empfangene Nachricht eine Empfangsbestätigung enthält. Sobald das der Fall ist, wird der Datentransfer gestartet.

Nachdem der Benutzer die oberste Schaltfläche, siehe Nr.1 in Abbildung 9: Dialogfenster Aufrufschaltflächen, betätigt hat und das Dialogfenster bereit ist, muss das Add-In die

[35] Vgl. https://docs.microsoft.com/de-de/office/dev/add-ins/develop/dialog-api-in-office-add-ins#handle-dialogparentmessagereceived-in-the-dialog-box (16.07.2022).

Projektnummer-Liste bereits empfangen haben. Um das zu prüfen, fragt das Add-In asynchron jede halbe Sekunde ab, ob die Daten bereits zur Verfügung stehen. Sobald ein Timeout von 20 Sekunden erreicht ist, wird eine Fehlermeldung ausgegeben. Das aktive Warten ist nicht die beste Lösung und sollte in Zukunft durch eine bessere Alternative ersetzt werden.

Das Dialogfenster besteht hauptsächlich aus einem Suchfeld, einer zweispaltigen Tabelle und zwei Schaltflächen. In der Tabelle werden die von dem Hauptfenster übergebenen Projektnummern und dessen Projektbeschreibungen angezeigt. Das Suchfeld filtriert die Tabelle nach gesuchtem Wert. Falls die Suche eindeutig ist, kann die Eingabe mit der Entertaste bestätigt werden, sonst muss die gewünschte Zeile per Doppelklick ausgewählt werden. Sobald eine Projektnummer ausgewählt wurde, fragt das Add-In die dazu gehörigen Projektzeilen in DWH nach. Diese werden beim Empfangen ebenfalls tabellarisch ausgegeben. Da nicht jede Projektnummer mindestens eine Projektzeile besitzt, sind diese optional. Sobald der Nutzer die „Senden“ Schaltfläche oder eine Projektzeile ausgewählt hat, werden die Daten an das Hauptfenster versendet und das Dialogfenster geschlossen. Das Hauptfenster besitzt eine zweite Schaltfläche neben der Projektzeile, siehe Nr. 2 in Abbildung 9: Dialogfenster Aufrufschaltflächen, die eine einzelne, bereits geprüfte, Projektnummer an das Dialogfenster sendet. In diesen Fall wird die Projektnummer Auswahl übersprungen.

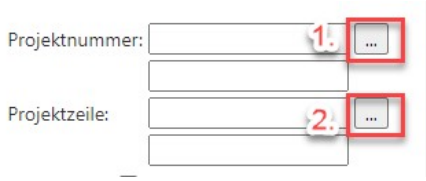

Abbildung 9: Dialogfenster Aufrufschaltflächen

Das Hauptfenster nimmt die zurückgegebenen Daten entgegen, befüllt damit die Textfelder, speichert diese in den *CustomProperties* und löst das Absenden von Projektdaten zum DWH aus.

4.4.8 Projektdaten Validierung

Der Nutzer muss in der Lage sein, die Projektdaten gleich in den Textfeldern eingeben zu können. In diesen Fall muss die Eingabe geprüft werden, bevor diese im DWH gespeichert wird. Es dürfen nur die bereits existierenden Projektnummern und dazu gehörigen Projektzeilen akzeptiert werden. Die Eingabeprüfung wird bei der Datenmenge durch eine einfache lineare Suche realisiert. Vor der Prüfung stellt das Add-In sicher, ob die Projektnummern oder die Projektzeilen entsprechend bereits nachgeladen sind. Es existieren mehrere Fälle, die eine Eingabeprüfung erfordern.

Im ersten Fall wird die Projektnummer direkt durch das Textfeld geändert. Dabei werden die *CustomProperties* der Projektnummer und der Projektzeile inklusive der Beschreibungen gereinigt. Dann wird die Eingabeprüfung der Projektnummer durchgeführt und beim Erfolg in den *CustomProperties* gespeichert und an das DWH versendet. Die

Projektnummer muss auch geprüft werden, falls der Nutzer eine Projektzeile durch das Dialogfenster auswählen möchte.

Falls eine Projektzeile angegeben wurde, muss diese ebenso wie die Projektnummer, geprüft werden. Die Projektzeile wird nur mit einer vorhandenen Projektnummer geprüft. Damit eine Projektnummer oder eine Projektzeile beim Betätigen der Speicher-Schaltfläche nicht erneut geprüft werden muss, merkt sich das Add-In, ob die aktuelle Eingabe bereits geprüft wurde oder nicht und holt es gegebenenfalls nach. Sobald die Daten korrekt angegeben sind, sendet das Add-In die gesammelten Projektdaten automatisch zum DWH und zeigt die im DWH gespeicherte Zeit auf der Oberfläche an.

4.5 Authentifizierung

Als Authentifizierung bezeichnet man das Nachweisen einer Identität. Nicht jeder Add-In Nutzer darf auf die sensiblen Projektdaten eines Unternehmens zugreifen. Es muss sichergestellt werden, dass nur die Mitarbeiter von COSMO CONSULT auf die Ressourcen zugreifen können. Microsoft bietet dafür die Single-Sign-On, weiter mit SSO abgekürzt, Schnittstelle an. Das SSO steht für ein einmaliges Anmelden, welches eine Authentifizierungsmethode ist, mit der sich Benutzer mit einem Satz von Anmeldeinformationen bei mehreren unabhängigen Softwaresystemen anmelden können. Um den Authentifizierungsprozess von SSO verstehen zu können, muss zuerst dessen Basis erläutert werden. Der SSO Anmeldeprozess ist eine Access Token, auch als Zugriffstoken bezeichnet, basierte Authentifizierung.

4.5.1 Access Tokens Allgemein

Die Zugriffstokens werden von gewählten Autorisierungsserver generiert und validiert. Dafür muss der Benutzer seine Identität vorher an diesen Server verifizieren lassen. Ein Zugriffstoken ist ein digitaler Schlüssel, welcher das Anmelden an dafür bestimmten Anwendungen ermöglicht. Eines der Vorteile eines Zugriffstokens liegt darin, dass es nur innerhalb eines Zeitintervalls, oder optional bis zur ersten Verwendung gültig ist. Innerhalb des angegebenen Zeitrahmens kann der Benutzer den Token frei nutzen, ohne sich jedes Mal erneut von neu anmelden zu müssen. Die Token-basierte Authentifizierung ist aufwendig in der Realisierung, bietet aber dafür eine hohe Sicherheit. Um einen klassischen Token Fluss darzustellen, wurde folgendes Diagramm erstellt.

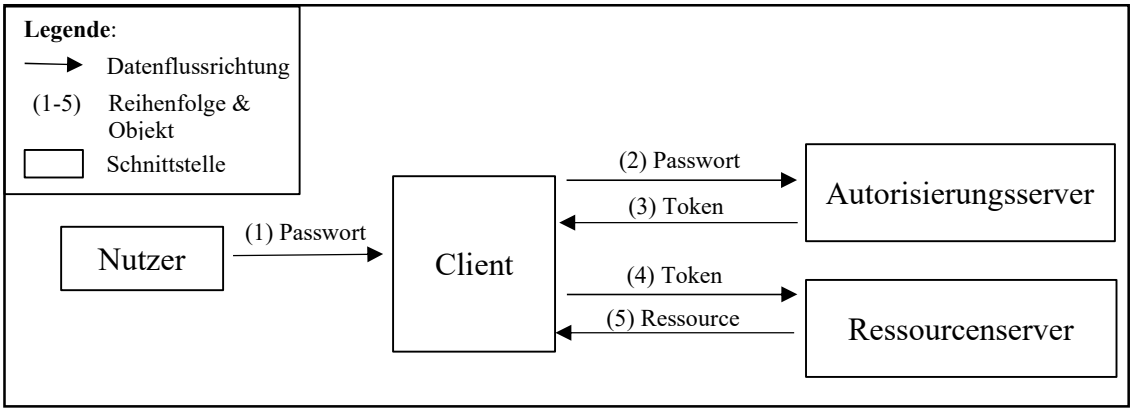

Abbildung 10: Token Authentifizierung

Falls ein nichtauthentifizierter Nutzer eine Ressource anzufordern versucht, kriegt er eine Fehlermeldung zurück. Um auf die Ressource zugreifen zu können, muss der Nutzer seine Login Daten (1) im Client angeben. Dieser sendet die Anmeldedaten an das Autorisierungsserver (2). Sobald die Daten korrekt validiert wurden, sendet dieser ein Token (3) zurück. Der Client fordert die Ressource an, in dem es den Token im Kopf der Anfrage mitsendet (4). Falls der Token kryptisch gesichert wurde, kann die Validierung gleich beim Ressourcenserver erfolgen, sonst muss es zuerst mit dem Autorisierungsserver abgeglichen werden. Sobald der Token korrekt ist, übergibt der Ressourcenserver die angeforderte Ressource (5) an den Client.

4.5.2 JSON Web Token

Es gibt eine Vielzahl von Token Arten, angefangen von physischen Speicherkarten bis hin zu elektronischen Zertifikaten und Passwort Hashes. Die SSO Authentifizierung benutzt eine besondere Form von Authentifizierungstoken. Diese ist der JSON Web Token, der gängig mit JWT abgekürzt wird. Für eine erleichterte Lesbarkeit wird das JWT im Laufe dieser Arbeit auch mit *JWT-Token* bezeichnet.

Ein JSON Web Token ist ein offener und nach RFC 7519[36] Standard genormter Access-Token. Es ist ein kompaktes, URL-sicheres Mittel zur Darstellung von Forderungen, die zwischen zwei Endpunkten übertragen werden sollen. Die Angaben in einem JWT werden als JSON-Objekt kodiert, welches als Nutzerlast einer JSON Web Signature Struktur oder als Klartext einer JSON Web Encryption Struktur verwendet wird. Die SSO Authentifizierungs-Tokens verwenden die JSON Web Signature Struktur. Diese Tokens bestehen aus drei Komponenten, die in der Base64 kodierten Form mit einem Punkt getrennt werden.

Die erste Komponente ist der Header, der den Typ des Tokens beinhaltet. Weiterhin enthält der Header den Signaturalgorithmus, der an die Header und Payload Komponente angewendet wurde.
Die zweite Komponente ist das Payload. Das ist der informationelle Inhalt des Tokens. Es ist ein JSON Element, der aus Claims besteht. Als Claim wird ein Name/Wert Paar bezeichnet, die als Attribute eines Gegenstands betrachtet werden. Es gibt drei Klassen von JWT-Claim-Namen: Registrierte, öffentliche und private Claim-Namen.

Tabelle 3: Die Registrierten Claim Namen

Typ	Bezeichnung	Beschreibung
"iss"	Issuer	Identifiziert die Authentifizierungsstelle, die das Token erstellt und zurückgibt.
"sub"	Subject	Identifiziert das Subjekt, über den das Token Informationen zusichert, wie zum Beispiel der Benutzer einer Anwendung.
"aud"	Audience	Identifiziert den vorgesehenen Empfänger des Tokens.
"exp"	Expiration	Beinhaltet das Ablaufdatum des Tokens.
"nbf"	Not Before	Diese Angabe gibt den Zeitpunkt an, vor dem der JWT nicht zur Verarbeitung angenommen werden darf.
"iat"	Issued At	Das ist der Zeitpunkt, an den die Authentifizierung für das betrachtete Token erfolgte.
"jti"	JWT ID	Die Angabe liefert einen eindeutigen Bezeichner für das JWT.

Die Registrierten Claim Namen sind laut RFC 7519 Standards alle optional. Die von SSO generierten Tokens beinhalten beispielhaft das „jti“ Claim nicht.
Die letzte Komponente ist die Signatur. Es ist ein Hashwert, der mit den ersten beiden vorher mit Base64 kodierten Komponenten erstellt wurde. Damit kann die Integrität eines Tokens sichergestellt werden.

[36] Vgl. https://datatracker.ietf.org/doc/html/rfc7519 (07.07.2022).

Die Microsoft-Identitätsplattform stellt für die benutzerdefinierte API´s zwei JWT-Formate mit den Namen „v1“ und „v2“ zur Verfügung, die sich mit Base64 nicht einfach dekodieren lassen. Die Entwickler können Ihre JWT Tokens auf der von Microsoft bereitgestellter Webseite[37] zum Testen dekodieren lassen.[38] Für das Add-In wird die „v2“ JWT-Token Variante verwendet. Ein entschlüsseltes Beispiel Token kann im Abbildungsverzeichnis unter Abbildung 3: Codiertes JWT und Abbildung 4: Decodiertes JWT angesehen werden.

4.5.3 Single Sign On

Das Single-Sign-On ist eine auf Token basierende Authentifizierungsmethode. Übersetzt steht das Single-Sign-On, abgekürzt mit SSO, für eine einmalige Anmeldung. Es gewährt dem Nutzer mit nur einer Anmeldung den Zugriff nicht nur auf eine, sondern gleich auf mehrere Webressourcen. Dabei braucht der Benutzer keine zusätzlichen Konten für die zusätzlichen Ressourcen anlegen, die das SSO unterstützen. Die SSO Authentifizierungsmethode kümmert sich selbst um die Ermittlung der Identität für alle Anwendungen, für die der Nutzer Zugriffsrechte besitzt. Nach einer Anmeldung erhält der Nutzer einen digitalen Ausweis in Form eines Access Tokens, der eine Art Zugangsschlüssel für bestimmte Webressourcen repräsentiert.[39] Die Funktionalität eines Access Tokens wurde im Kapitel Access Tokens Allgemein bereits beschrieben.

Die im Kapitel 5 beschriebene Microsoft Azure Plattform, auf der sich unter anderen die Verwaltung der Office Add-Ins befindet, stellt ein Unternehmensidentitätsdienst namens Azure Active Directory zu Verfügung. Die Azure Active Directory, oft auch mit Azure AD abgekürzt, ist für die Multi-Faktor- und für die SSO-Authentifizierung innerhalb der Azure Plattform zuständig. Die Azure AD wird innerhalb der Microsoft Dokumentation oft mit den Begriff *Microsoft Identity Plattform* erwähnt, da es sich hierbei um die nächste Entwicklung der Azure AD Developer Plattform handelt.

Das im Add-In einprogrammiertes SSO Verfahren erweitert die im Kapitel 4.5.1 bereits beschriebene Token Authentifizierung mit zusätzlichen Schritten. Die Azure AD übernimmt in unseren Fall die Rolle des Autorisierungsservers für die registrierten Anwendungen. Das Add-In wird ebenfalls vorher im Azure AD registriert. Dazu siehe Kapitel 5.1. Das folgende erstellte Diagramm beschreibt den SSO Authentifizierungsdatenfluss zwischen dem Add-In und der externen Schnittstelle zum DWH.

[37] https://jwt.ms/ (07.07.2022).
[38] Vgl. https://docs.microsoft.com/de-de/azure/active-directory/develop/access-tokens?WT.mc_id=ad-demystify-blog-masoucou (28.07.2022).
[39] Vgl. https://www.security-insider.de/was-ist-single-sign-on-sso-a-631479/ (24.07.2022).

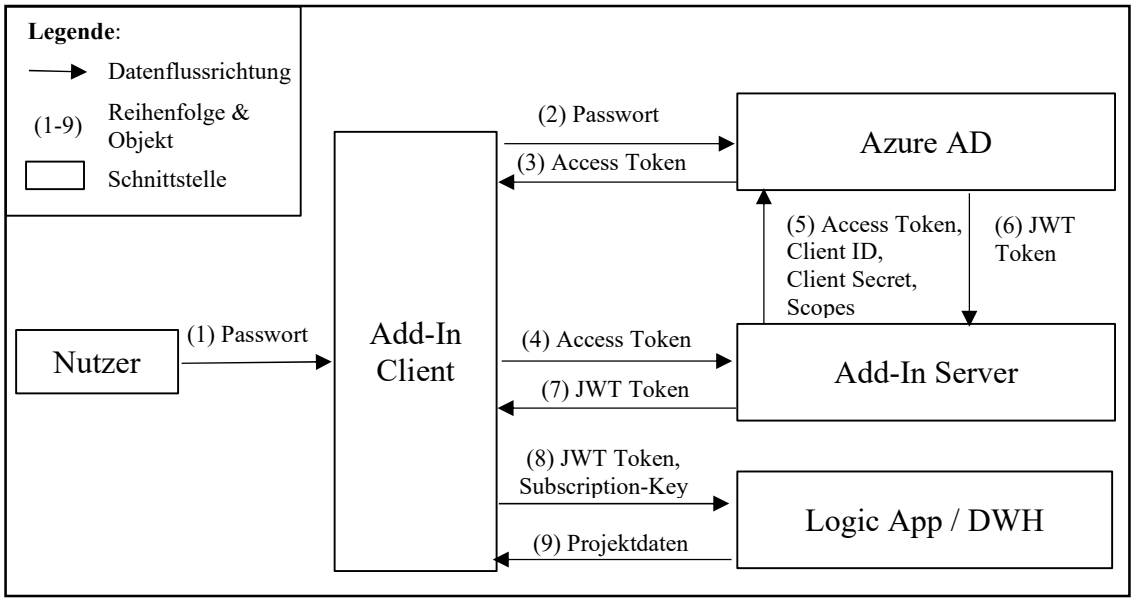

Abbildung 11: SSO Authentifizierungsablauf

Sobald der Nutzer im Outlook Add-In Client (1) angemeldet ist, kann das Add-In ein SSO Access Token bei der Azure AD (2) anfordern. Beim Erfolg wird das Access Token zurück an das Add-In Client (3) übergeben und im Outlook selbst aufbewahrt.[40] Alle weiteren Anfragen greifen zuerst auf den im Outlook gespeicherten Access Token zu, sobald dieser noch nicht abgelaufen ist. Das Access Token enthält in unseren Fall nur die Anmeldeinformationen von dem Benutzer selbst. Um auf die DWH Ressourcen zugreifen zu können, muss ein weiteres JWT-Token erzeugt werden. Dazu sendet das Add-In Client den Access Token zum Add-In Server (4). Für die Anfrage des finalen JWT-Zugriffstokens müssen weitere Parameter angegeben werden, die sich aus den Sicherheitsgründen auf den Add-In Server befinden. Diese sind die Client-ID des Add-Ins, als auch das Client Secret, dass durch die im Kapitel 5.1 durchgeführte Add-In Registrierung in Azure AD erstellt und erläutert wurden. Die letzten Parameter sind die Scopes. Die beinhalten die Geltungsbereiche, die für die Zielgruppe des JWT-Tokens entscheidend sind. Die Rolle der Scopes wurde im Kapitel 5.2.1 untersucht. Sobald die Anfrage mit allen benötigten Parametern wieder an das Azure AD Identitätsplattform abgeschickt wurden (5), empfängt der Add-In Server den gewünschten JWT-Token zurück und leitet es an den Add-In Client weiter (7). Der JWT-Token kann nun für alle Anfragen (8), wie im nächsten Kapitel 5.2 beschrieben, gemeinsam mit den Projektdaten an das DWH durch die Logic App übermittelt werden. Der JWT-Token kann auch, unter in Kapitel 5.2.1 erwähnten Bedingungen, für die Authentifizierung am Microsoft Graph verwendet werden, der im nächsten Kapitel beschrieben wird.

Ohne das Single-Sign-On wäre der Nutzer gezwungen, sich nicht nur einmal im Outlook, sondern auch jedes Mal im Add-In gegenüber der Logic App, die den Datenfluss zum DWH ermöglicht, anzumelden.

[40] Vgl. https://docs.microsoft.com/de-de/office/dev/add-ins/develop/sso-in-office-add-ins (24.07.2022).

Diese Vorgehensweise wird von Microsoft als *OAuth 2.0 On-Behalf-Of*, kurz OBO, bezeichnet. Es dient dem Anwendungsfall, wenn eine Anwendung eine Schnittstelle zur Mittelschicht ruft, die wiederum eine weitere Schnittstelle anfragt. Dabei werden die Benutzerberechtigungen und dessen Identität an die Zwischenschicht delegiert.[41] Der Add-In Server stellt in unseren Fall die Zwischenschicht da. Das im Kapitel 4.3.5 als Vorlage verwendetes SSO Add-In 2.1 Beispiel versendet das JWT-Token im Schritt (7) der Abbildung 11: SSO Authentifizierungsablauf zum Add-In Client, wo dieser für die weiteren Anfragen verwendet wird. Im Laufe der Entwicklung wurde das SSO Beispiel am 07.07.2022 zur Version 2.2 aktualisiert.[42] Dieses wurde an die aktuellen OBO Richtlinien angepasst.[43] Laut diesen sollte das JWT-Token die Zwischenschicht nicht verlassen und die damit betätigten Aufrufe ebenfalls zuerst auf der Zwischenschicht abgearbeitet werden und dann dessen Resultat an den Add-In Client weitergegeben.

Diese Abarbeitungsweise sollte in späteren Add-In Updates außerhalb dieser Arbeit nachgeholt werden.

4.5.4 Microsoft Graph

Microsoft Graph ist eine mächtige Schnittstelle zum mächtigen Teil von Microsoft 365, Windows und Enterprise Plattform Daten. Die meisten bekannten Microsoft Produkte sind im Microsoft Graph enthalten.[44] Alle dazu gehörigen Daten können, mit richtigen Geltungsbereichen, durch die einzige Microsoft Graph API *https://graph.microsoft.com* erreicht werden. Eins davon ist der Outlook/Exchange Server, wo alle Outlook relevanten Daten gespeichert sind. Damit eine Anfrage an den Microsoft Graph erfolgreich ist, muss ein im letzten Kapitel erzeugtes JWT-Token mit verwendet werden. Es ist wichtig zu beachten, dass der erzeugte JWT-Token explizit für den Microsoft Graph bestimmt ist. Dieses Verfahren wird im Rahmen des Kapitels API Management weiter untersucht. Im Laufe dieser Arbeit war das Add-In in der Lage, die Profilinformationen wie Name, Arbeitstitel, Mail, Handynummer und Office Aufenthaltsort des angemeldeten Benutzers bei der Graph API anzufragen und in die Textbeschreibung des Kalendereintrags einzufügen. Die dazugehörige Funktionalität wird im *helpers* Verzeichnis aufbewahrt, wie in Kapitel 4.3.5 SSO Unterstützung bereits erklärt.

Neben der Profildaten ist der Microsoft Graph auch in der Lage, die Informationen über alle Kalender des Benutzers auszugeben. Diese Daten können genauso sortiert angefragt werden. Hier ist ein Beispielaufruf für die Kalendereinträge einer bestimmten Woche:

*https://graph.microsoft.com/v1.0/me/calendarview?startdatetime=**2022-07-30**T11:50:00.375Z&enddatetime=**2022-08-06**T11:50:00.375Z*

[41] Vgl. https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-on-behalf-of-flow#middle-tier-access-token-request (29.07.2022).
[42] Vgl. https://github.com/OfficeDev/Office-Add-in-samples/tree/main/Samples/auth/Office-Add-in-NodeJS-SSO (29.07.2022).
[43] Vgl. https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-on-behalf-of-flow#middle-tier-access-token-request (29.07.2022).
[44] Vgl. https://docs.microsoft.com/de-de/graph/overview#whats-in-microsoft-graph (30.07.2022).

Das Ergebnis ist eine JSON Datei, die alle, innerhalb der zeitlich festgelegten Rahmen, erstellten Kalendereiträge inklusive dessen Identifikationsnummern beinhaltet. Diese Daten können verwendet werden, um alle Kalendereinträge einer bestimmten Terminserie zu identifizieren. Somit könnte das Outlook Add-In alle zu einer Serie gehörten Termine abrufen und einzelne mit deren Identifikationsnummern in das DWH abspeichern.

Der Microsoft Graph kann auch außerhalb eines Office Add-Ins getestet werden. Dazu stellt Microsoft das sogenannte *Graph Explorer* Webtool zur Verfügung.[45] Die Oberfläche des Graph Explorers ist auch im Abbildungsverzeichnis auf Abbildung 6: Graph Explorer zu sehen. Dort können nicht nur vorgefertigte Beispieldaten, sondern auch die echten Nutzerdaten abgerufen werden. Dies kann durch die Anmeldung mit dem eigenen Microsoft Konto erfolgen. Dies ist jedoch nicht die einzige Möglichkeit. Die Authentifizierung kann zusätzlich mithilfe eines Zugriffstokens erfolgen. Diese Möglichkeit wurde für den JWT-Token Validierungstest verwendet, um das JWT-Token auf die Korrektheit zu prüfen.

[45] https://developer.microsoft.com/de-de/graph/graph-explorer?request=me%2Fevents&version=v1.0 (30.07.2022).

5 Microsoft Azure Portal

Microsoft Azure ist eine Cloud-Computing-Plattform. Es bietet ein vielseitiges Angebot an Webdienstleistungen, darunter Rechen-, Analyse-, Speicher- und Netzwerkdienste.[46] Es bietet nicht nur die Möglichkeit, ein Microsoft Add-In identifizieren zu lassen, sondern viele weitere Webdienstleistungen, die von COSMO CONSULT verwendet werden. Das Microsoft Business Central ERP System ist in den letzten Jahren auf die Cloud Basis umgestiegen und wird als ein „Software as a service" Dienst angeboten. Dabei befindet sich das gesamte ERP System nicht mehr auf den Kundenservern, sondern direkt bei Microsoft. Der Zugriff erfolgt dabei durch einen Webbrowser. Azure stellt alle für die Business Central ERP System Anpassung notwendigen Mitteln zur Verfügung. Ein Beispiel davon ist das Azure DevOps, ein Werkzeug für die parallele Softwareentwicklung und Systemadministration.[47]

Das Microsoft Azure Portal dient zur Verwaltung der von Azure angebotenen Dienstleistungen. Unter den komplexen Cloudbereitstellungen können im Azure Portal vielseitige Web-Anwendungen implementiert und verwaltet werden. Zu denen gehören die Azure Logic Apps, die automatisierte Workflows darstellen und ein Teil der Outlook Add-In Entwicklung sind.

Allgemein wird im Azure Portal die gesamte Webverwaltung des Add-Ins, angefangen von der App Registrierung, der Authentifizierung, bis hin zur Inbetriebnahme durch die Endnutzer, durchgeführt. Somit ist es einer der wichtigsten Bestandteile der Web-basierten Microsoft Office Add-In Entwicklung.

5.1 App Registration

Damit ein Microsoft Add-In das Single Sign-On unterstützten kann, muss das Add-In zuerst in der Azure Active Directory registriert werden. Die Azure Active Directory, auch mit Azure AD abgekürzt, ist ein Teil der Microsoft Identity Plattform und dient unter anderem als Autorisierungsserver und stellt bei einer erfolgreichen Authentifizierung ein JSON Web Token zur Verfügung.

Die App-Registrierung beginnt entweder in Azure Portal[48] oder direkt im, bereits in Kapitel 4.5.3 erwähnten, Azure Active Directory. Das Add-In wird mit einer passenden Bezeichnung angelegt. Dabei wird automatisch die Anwendungs-ID, die in der Microsoft Dokumentation an manchen Stellen auch als Client-ID bezeichnet wird. Als nächstes muss unter „Certificates & Secrets" ein Client Secret, auch als Clientschlüssel bezeichnet, erstellt werden. Dieser wird nur beim Anlegen sichtbar und sollte sofort sicher gespeichert werden. Der Clientschlüssel wird zur Identifizierung des Add-Ins an der Azure AD verwendet. Damit wird sichergestellt, dass nur das angemeldete Add-In ein JWT-Token anfragen darf. Der Clientschlüssel wird zusammen mit der Client-ID in den Anwendungseinstellungen auf der Serverseite des Add-Ins gespeichert und ist bis zu 24 Monate

[46] Vgl. https://azure.microsoft.com/de-de/overview/what-is-azure/ (14.07.2022).
[47] Vgl. https://azure.microsoft.com/en-us/services/devops/ (14.07.2022).
[48] https://portal.azure.com/ (10.07.2022).

gültig.[49] Die Anwendungseinstellungen sind im Microsoft Azure Portal verwaltbar und werden zur Laufzeit als Umgebungsvariablen für den Zugriff durch das Add-In verfügbar gemacht. Diese werden in Ruhestand verschlüsselt und über einen sicheren Kanal übertragen. Dank der Anwendungseinstellungen bleiben die sensiblen Daten nur auf der Serverseite des Add-Ins sichtbar und können im Browser nicht mehr ohne eine zusätzliche Einstellung eingesehen werden.[50]
Als nächstes wird die Add-Ins Web API mit Hilfe der Application-ID-URI zur Verfügung gestellt, die auch als „Application-GUID" bezeichnet wird. Die Application-ID-URI wird für die Authentifizierung benötigt und hat folgenden Aufbau:

api://<domain-name>/<application-id>

Diese wird aus der Domäne, in welcher das Add-In Server gehostet wird, und aus der *application-id* des Add-Ins zusammengefügt. Die *application-id* ist in den ersten Zeilen der Manifest Datei zu finden. Die Application-ID-URI wird ebenfalls für die SSO Authentifizierung benötigt und taucht sowohl in der Manifest Datei als auch bei der JWT-Token Anfrage mit auf. Als nächstes muss der Geltungsbereich für das Add-In deklariert werden. Zuerst wird ein Titel für den Bereich bestimmt. In diesem Fall lautet der Titel *access_as_user*. Danach muss eine Bereichs ID hinzugefügt werden, die einer der Office Anwendungen gehört. Für das Outlook im Web lautet diese *bc59ab01-8403-45c6-8796-ac3ef710b3e3.* Der Bereich erlaubt es dem Outlook Client die Web-APIs unseres Add-Ins mit den Zugriffsrechten des aktuellen Benutzers zu verwenden. Um gleich alle Microsoft Office Anwendungsendpunkte einzufügen, kann die Bereichs ID *ea5a67f6-b6f3-4338-b240-c655ddc3cc8e* verwendet werden.[51] Als letztens fügt man das Add-In zu dem erstellten Geltungsbereich hinzu. Somit kann der Bereich auch vom Add-In direkt angesprochen werden. Die Geltungsbereiche werden auch als Scopes bezeichnet.

Zum Schluss können die für das Add-In relevanten Microsoft Graph Berechtigungen delegiert werden. Das sind die Berechtigungen, die von der COSMO CONSULT Global IT für das Outlook Add-In zur Verfügung gestellt wurden:

Tabelle 4: An das Add-In erteilte Berechtigungen

API/Berechtigungsname	**Beschreibung**
User.Read	Anmeldung und Benutzerprofil lesen.
Profile	Grundlegendes Profil von Benutzern anzeigen.
Openid	Benutzer anmelden.
Office_access	Zugriff auf Daten beibehalten, für die Sie Zugriff erteilt haben.
Calendars.ReadWrite	Verfügt über Vollzugriff auf Benutzerkalender.

[49] Vgl. https://docs.microsoft.com/de-de/office/dev/add-ins/develop/register-sso-add-in-aad-v2#add-a-client-secret (12.07.2022).
[50] Vgl. https://docs.microsoft.com/de-de/azure/static-web-apps/application-settings (14.07.2022).
[51] Vgl. https://docs.microsoft.com/en-us/office/dev/add-ins/develop/register-sso-add-in-aad-v2#add-a-scope (14.07.2022).

Diese Berechtigungen müssen gemeinsam mit der Client-ID und der Application-ID-URI in der Manifest Datei unter der *WebApplicationInfo* eingefügt werden.

5.2 Datenfluss zwischen Outlook Add-In und Data Warenhouse

Während der Anforderungsanalyse war es zuerst geplant, die Projektdaten direkt aus den COSMO CONSULT Enterprise Resource Planning System zu laden und da auch zu speichern. Das COSMO CONSULT ERP System beinhaltet sensible Daten verschiedener Kundenunternehmen. Die Herausforderung bestand daran, den Datenaustausch entsprechend sicher zu gestalten. Deswegen wurde beschlossen, die Outlook Add-In relevanten Daten separat in einem Data Warenhouse zu verwahren und auszuwerten. Auch da muss die Sicherheit des Datentransfers gewährleistet werden.

Die ausgearbeitete Route zwischen den zwei Endpunkten wird im folgenden Diagramm vereinfacht dargestellt:

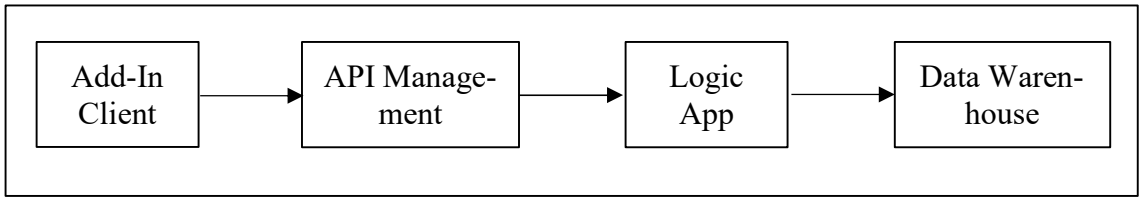

Abbildung 12: Datenfluss zwischen Add-In Client und Data Warenhouse

Im Kapitel 4.5.1 wurde das Anfordern von Zugangstoken mit Hilfe von Nutzer Anmeldedaten beschrieben. Mit diesem Token kann man auf die Microsoft Graph Daten zugreifen. In den folgenden Unterkapiteln wird der Versuch, mit den Zugangstoken auf die DWH Daten zuzugreifen, genau untersucht.

5.2.1 API Management

„Azure API Management ist eine hybride Multi-Cloud-Verwaltungsplattform für APIs in allen Umgebungen“[52] und setzt sich aus drei Komponenten zusammen. Die erste Komponente ist das API-Gateway. Es nimmt die API-Anrufe entgegen und gibt diese an vorher konfigurierte API-Anbieter weiter. In diesem Fall ist dies die Logic App. Unter anderen Aufgaben ist das API-Gateway für die Validierung der JWT-Token, API-Schlüssel und anderen Anmeldeinformationen verantwortlich. Die Verwaltungsebene ist die zweite Komponente. Diese bietet den API-Anbietern mit Hilfe von verschiedenen Azure Diensten, unter anderen durch das Azure-Portal oder durch die Visual Studio Code-Erweiterung, den interaktiven Zugriff auf das Azure API Management. Die letzte Komponente ist das Open-Source-basierte Entwicklerportal, welches von den Benutzern inhaltlich frei anpassbar ist. Das Entwicklerportal bietet unter anderem wichtige Optionen an, wie das Verwalten von API-Schlüsseln. Hier kann ein Subscription-Key generiert werden, mit dem man die Anfragen zwischen den Outlook Client und den API-Manager absichern kann.

[52] https://docs.microsoft.com/de-de/azure/api-management/api-management-key-concepts (09.07.2022).

Der Subscription-Key wurde zuerst für Testzwecke hart in den Quellcode des Outlook Add-Ins einprogrammiert und musste mit jeder an die Logic App gerichtete Anfrage im Header angegeben werden. Allerdings wäre jeder Nutzer mit genug Hintergrundwissen in der Lage, den Subscription-Key direkt im Browser auszulesen. Auf Grund dessen wurde eine alternative Authentifizierung gebraucht. Es wurde versucht, das für Microsoft Graph gedachte JWT-Token, für die API-Management Authentifizierung zu benutzen. Der API-Manager benötigt zusätzliche Informationen, um die verschlüsselten JWT-Tokens dekodieren zu können.[53] Dafür werden spezifische Richtlinien, auch policy genannt, benötigt, mit denen auch das JWT-Token erstellt wurde. Diese werden von der COSMO CONSULT Domäne, welche ebenso bei der direkten Anmeldung im Outlook Add-In verwendet wird, zur Verfügung gestellt.

Bevor die Validierung des JWT-Tokens eingerichtet wurde, wies der API-Manager darauf hin, dass die Anfrage zwei Authentifizierungsarten beinhaltet. Diese sind die „Bearer“ Authentifizierung, welche für das JWT-Token steht, und zusätzlich die SAS Authentifizierung. Das SAS steht für Shared Access Signature. Es ermöglicht einen eingeschränkten Zugriff auf die Azure Benutzer Ressourcen.[54] Der API-Manager verwendet diese Signatur standardmäßig, um auf die Logic App zugreifen zu können. Um das Problem beheben zu können, muss der JWT-Token aus dem Header der Logic App Anfrage nach der erfolgreichen Validierung wieder gelöscht werden.

Der API-Manager lehnte die Anfrage mit der Fehlermeldung *401 unauthorized* ab. Dies bedeutet, dass die Authentifizierung nicht erfolgreich war. Der API-Manager bietet Tools an, mit denen man die API genauer untersuchen kann. Mit Hilfe von Log Daten konnte bestätigt werden, dass die JWT-Token Validierung fehlgeschlagen ist. Die Richtlinien, mit denen das Token geprüft wird, enthalten die Claims, die im JWT-Token erhalten sein müssen. Die Claims und der Aufbau eines JWT-Tokens wurden im Kapitel JSON Web Token erläutert. In dem API-Manager wurden vorerst drei Claims zur JWT Validierung eingestellt. Das sind die „Issuer“, „Audience“ und „Expiration“ Claims. Mit Hilfe des von Microsoft bereit gestelltes Online Tools[55] ließ sich der JWT-Token dekodieren und in der für die Menschen lesbaren JSON Syntax anzeigen. Unter den drei Claims besaß das „Audience“ Claim eine ungewöhnliche Form: "aud":"00000003-0000-0000-c000-000000000000" statt der Applikation ID des Add-Ins aus der Azure App Registration, siehe Kapitel 5.1. Nach einer Recherche stellte es sich heraus, dass dieser Audience Claim laut der Microsoft Dokumentation[56] zum Microsoft Graph gehört. Die Audience identifiziert den vorgesehenen Empfänger des Tokens, und dies ist nicht der API-Manager.

Das JWT-Token wurde mit den Microsoft Graph Scopes angefragt. Unter Scopes versteht man die Geltungsbereiche, für die das Token angefragt wird. Um ein für den API-Manager korrekten JWT-Token zu erhalten, müssen die richtigen Scopes angefragt werden. Der Scope mit dem Titel *access_as_user* wurde bereits in der App Registration erstellt

[53] Vgl. https://docs.microsoft.com/en-us/azure/api-management/api-management-access-restriction-policies#ValidateJWT (09.07.2022).
[54] Vgl. https://docs.microsoft.com/en-us/azure/storage/common/storage-sas-overview (09.07.2022).
[55] https://jwt.ms/ (09.07.2022).
[56] https://docs.microsoft.com/en-us/troubleshoot/azure/active-directory/verify-first-party-apps-sign-in#application-ids-for-commonly-used-microsoft-applications (09.07.2022).

und muss nun in der Token Anfrage mit angegeben werden. Die Reihenfolge der Scopes spielt dabei eine wichtige Rolle. Bei einer Anfrage kann man mehrere Scopes angeben, wie zum Beispiel *Calendars.Read* und *profile*. Der *access_as_user* Scope muss an der ersten Stelle stehen, damit kein Microsoft Graph Token erzeugt wird. Damit der Scope von der Authentifizierungsplattform abrufbar ist, muss hinter ihm die Application-ID-URI, die während der App Registrierung erstellt wurde, mit angegeben werden. Der angefragte Scope hat somit folgende Gestalt:

api://<domain-name>/<application-id>/access_as_user

Der Audience Claim des angefragten JWT-Tokens bestand nun aus der Application-ID-URI und nicht aus dem Microsoft Graph Audience Claim. Mit dem neuen JWT-Token war die Authentifizierung im API-Manager erfolgreich.

5.2.2 Azure Logic Apps

In Rahmen der Outlook Add-In Entwicklung wurden mehrere Azure Logic Apps für den Datentransfer zwischen Client und Data Warenhouse von der COSMO COSNULT IT-Abteilung für das Add-In zur Verfügung gestellt. Die Azure Logic Apps werden auf einer Azure Web-Plattform zur Automatisierung verschiedener Dienste entwickelt. Die Implementierung der Logic Apps basiert auf einer graphischen Basis. Der Ablauf der App wird mithilfe von visuellen Bausteinen, die verschiedene logische Ereignisse repräsentieren, miteinander verbunden. Die einzelnen Bausteine können optional Programmcode beinhalten.[57] Zur Veranschaulichung wurde die Logic App für das Abrufen von Projektdaten mithilfe einer SQL-Anfrage aus dem Data Warenhouse im Abbildungsverzeichnis hinterlegt. Dazu siehe Abbildung 5: Logic App für Abrufen der Projektnummer-Liste.

Für folgende DWH Aufrufe wurden separate kleine Logic Apps entwickelt. Das Abrufen der Projektnummern, Projektzeilen sowie das Speichern, Löschen und Abrufen eines DWH Eintrags.

Da die Logic Apps kein Bestandteil dieser Arbeit sind, werden diese nicht weiter erläutert.

[57] Vgl. https://docs.microsoft.com/de-de/azure/logic-apps/logic-apps-overview (23.07.2022).

6 Fazit

Als Erstes konnte bewiesen werden, dass die ausgewählte Office Add-Ins API von den Microsoft gängigsten Technologien für die gestellte Anforderung am besten geeignet ist. Das entscheidende Kriterium war die Webfähigkeit eines auf der Office API basierten Add-Ins. Im Kern unterscheidet sich die Office API mit der Objektbasierten, auf der das vorherige Add-In erstellt wurde, an zwei Stellen. Erstens durch die die bereits erwähnte Webfähigkeit. Das alte Add-In konnte nur lokal installiert werden. Die mit der Office API erstellten Add-Ins können hingegen reibungslos im Microsoft Store verkauft und für eine Gruppe von Mitarbeitern simultan durch das Azure Portal zur Verfügung gestellt werden. Zweiter Unterschied ist der Datenzugriff. Das alte Add-In wurde direkt in Outlook eingebaut und konnte auf alle Nutzerdaten zugreifen, die lokal auf dem Rechner gespeichert wurden. Die Office API bietet hingegen eine limitierte Datenmenge. Das hängt damit zusammen, dass das Add-In nicht direkt in das System eingebaut wird, sondern eine eigenständige App ist, die im Outlook eingeblendet werden kann. Dieses Problem lässt sich jedoch mit der zusätzlichen Microsoft Graph API lösen, die eine enorme Datenmenge von den meisten Microsoft Produkten greifbar macht. Im Gegenteil dazu wird eine sichere Authentifikation benötigt. Das Add-In verfügt über zwei Authentifizierungsverfahren, die einander ergänzen. Zuerst kann das Add-In mithilfe eines Dialogfensters mit den Nutzer Login und Passwort direkt im Microsoft Identitätsplattform angemeldet werden, sobald der Nutzer noch nicht eigeloggt ist. Die Authentifizierung wurde so eingestellt, dass sich nur Mitarbeiter von COSMO CONSULT anmelden können. In den meisten Fällen ist der Nutzer im Outlook bereits angemeldet. Das implementierte Single-Sign-On Verfahren greift durch die Office API auf die Microsoft Identitätsplattform zu und besorgt damit ein Zugriffstoken, mithilfe dessen man nun auf die Microsoft Graph Daten zugreifen kann. Diese Möglichkeit kann für eine eventuelle Add-In Erweiterung verwendet werden. Es hat sich herausgestellt, dass das für Microsoft Graph gedachtes Zugriffstoken für die Authentifizierung an der Schnittstelle zum DWH nicht geeignet ist. Nachdem der Zugriffstoken dekodiert wurde, stellte es sich heraus, dass innen auch der Empfänger des Tokens mit notiert ist. Mit den richtigen Angaben konnte ein weiterer Zugriffstoken angefragt werden, der schlussendlich akzeptiert wurde. Dafür musste zusätzlich die Schnittstelle zur Logic App, die den Datentransfer zwischen Add-In und DWH verwaltet, entsprechend angepasst werden. Alle nicht temporären Daten, die für die Authentifizierung notwendig sind, wurden auf der Add-In Zwischenschicht aufbewahrt und nicht an den Add-In Client versendet. Somit wurde die geforderte Sicherheit der Daten gewährleistet.

Um den Single-Sign-On und Microsoft Graph verwenden zu können, musste das Add-In zusätzlich im Azure Portal registriert werden. Bei der Registrierung wurden mehrere Schlüsseldaten erstellt, damit nur das registrierte Add-In auf die gewünschten Daten zugreifen kann. Diese Schlüsseldaten werden auf dem von Azure bereitgestellten Server geheim gehalten. Ohne dieser Daten ist das beigefügte Add-In funktionsunfähig.

Die Entwicklung der geforderten Hauptfunktionalität des Add-Ins konnte ebenfalls bis zur Abgabe dieser Arbeit fertig gestellt werden. Mit der eingebauten Authentifizierung ist das Add-In in der Lage, Projektnummern und dazugehörigen Projektzeilen je nach ausgewählten Mandanten anzuzeigen, durch die Daten zu suchen und in einer tabellarischen Form anzuzeigen. Die größte Herausforderung lag bei der asynchronen Abarbeitung des Programms. Mit dieser können mehrere Prozesse gleichzeitig abgearbeitet

werden. Ein Beispiel dafür ist die Add-In Hauptoberfläche, welche beim Laden der Projektnummern nicht einfriert. Anderseits müssen manche Prozesse zwangsläufig aufeinander warten. Die Projektdaten können nicht geladen werden, bevor ein Zugriffstoken, die Identifikationsnummer des Kalendereintrags und die Benutzer-E-Mail asynchron bestimmt wurden. Die Projektdaten können genauso nicht abgeschickt werden, bevor die aktuelle Zeit ermittelt ist. Die Daten konnten auch an ein Dialogfenster nicht übergeben werden, bevor das Fenster überhaupt geladen wurde. Zum Schluss ist es gelungen, diese und auch andere Prozesse, zu synchronisieren, um eine reibungslose Benutzererfahrung zu ermöglichen.

In Laufe dieser Arbeit wurde schnell bewusst, dass die benutzte Office API ständig weiterentwickelt wird. Wenn man das Erstelldatum von allen im Literaturverzeichnis erwähnten Artikeln aus der Microsoft Office Add-Ins Dokumentation betrachtet, stammen die meisten entweder aus diesem oder aus dem letzten Jahr. Manche Verfahren wurden erst im Laufe dieser Arbeit beschrieben. Ein Beispiel dafür ist der im Kapitel 4.5.3 erwähnte On-Behalf-Of Flow. Dank dem Yeoman Generator ist es ziemlich einfach, ein funktionsfähiges Add-In lokal zu erstellen und zu testen. Dafür wird eine sehr simple Einleitung angeboten, doch sobald man die Single-Sign-On Unterstützung für die Produktion gedachte Add-In Variante einplant, hat es sich im Kapitel 4.3.5 eher als ungeeignet erwiesen. Eine intensive Recherche war notwendig, um zu verstehen, wie man ein Single-Sign-On fähiges Add-In auf einen Webserver zum Laufen bringen kann. Dafür musste ein separates, SSO fähiges Add-In Beispiel von Microsoft, welches für das Outlook gar nicht bestimmt war, zuerst umgebaut werden. In Laufe der Add-In Implementierung wurde das von Microsoft bereitgestellte SSO Add-In Beispiel etwas umstrukturiert und an die bereits erwähnten On-Behalf-Of Flow Sicherheitsrechtlinien angepasst. Das ist ein weiterer Beweis dafür, dass die Office API und dessen Struktur regelmäßig aktualisiert wird und noch nicht final ist.

Diese Bachelorarbeit beschreibt den Entwicklungsprozess eines Outlook Add-Ins von Grundaufbau bis zum global einsetzbaren Zustand. Alle hier beschriebenen Methoden eignen sich genauso für die Erstellung eines beliebigen webfähigen Outlook Add-Ins auf Office-API Basis. Die Authentifizierung und die Registrierung sind für die anderen, von Office unterstützten, Anwendungen identisch. Somit kann diese Bachelorarbeit als eine Richtlinie für die Implementierung von ähnlichen Add-Ins verwendet werden.

7 Literaturverzeichnis

ANONYM (2022): Erstellen eines Office-Add-In für Outlook, welches Single Sign-On implementiert. Microsoft [https://docs.microsoft.com/de-de/learn/modules/office-add-ins-sso/7-exercise-outlook-sso, 2022-07-26]

ANONYM, L. CAPUTO, D. B. MONTERO u.a. (2022): Overview of Microsoft Graph. Microsoft [https://docs.microsoft.com/en-us/graph/overview#whats-in-microsoft-graph, 2022-07-30]

CANNON, L. (2021): Find a specific appointment in a recurring appointment series. Microsoft [https://docs.microsoft.com/en-us/office/client-developer/outlook/pia/how-to-find-a-specific-appointment-in-a-recurring-appointment-series, 2022-07-23]

CHESNUT, D., R. KIRKHAM, A. JERABEK u.a. (2022): Enable single sign-on (SSO) in an Office Add-in. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/develop/sso-in-office-add-ins, 2022-07-24]

FAN, E., ANONYM, H. ARYA u.a. (2022): What is Azure Logic Apps. Microsoft [https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-overview, 2022-07-23]

HART, J., G. HOGENSON, V. MAKANDAR u.a. (2022): Office primary interop assemblies. Microsoft [https://docs.microsoft.com/de-de/visualstudio/vsto/office-primary-interop-assemblies?view=vs-2022, 2022-06-16]

HENDRICKSON, J., D. COULTER, B. SHILPA u.a. (2022): Enable or disable MAPI access to mailboxes in Exchange Server. Microsoft [https://docs.microsoft.com/en-us/exchange/clients/mapi-mailbox-access?view=exchserver-2019, 2022-07-22]

HOLLAND, B., A. CHU, C. SHOEMAKER u.a. (2022): Configure application settings for Azure Static Web Apps. Microsoft [https://docs.microsoft.com/en-us/azure/static-web-apps/application-settings, 2022-07-14]

JONES, M., J. BRADLEY u. N. SAKIMURA (2015): JSON Web Token (JWT). Microsoft, Ping Identity u. NRI [https://datatracker.ietf.org/doc/html/rfc7519, 2022-07-07]

KIRKHAM, R. (2022): Create Office Add-in projects using the Yeoman Generator. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/develop/yeoman-generator-overview, 2022-06-24]

KIRKHAM, R., A. JERABEK, E. SAMUEL u.a. (2022): Implement a pinnable task pane in Outlook. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/outlook/pinnable-taskpane, 2022-07-21]

KIRKHAM, R., A. JERABEK, E. SAMUEL u.a. (2022): Add custom keyboard shortcuts to your Office Add-ins. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/design/keyboard-shortcuts, 2022-07-21]

KIRKHAM, R., A. JERABEK, L. L. CANNON u.a. (2022): Set up your development environment. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/overview/set-up-your-dev-environment?tabs=yeomangenerator, 2022-06-23]

KIRKHAM, R., E. SAMUEL, A. JERABEK u.a. (2022): VersionOverrides element. Microsoft [https://docs.microsoft.com/en-us/javascript/api/manifest/versionoverrides?view=common-js-preview, 2022-06-25]

KIRKHAM, R., E. SAMUEL u. A. JERABEK (2022): WebApplicationInfo element. Microsoft, [https://docs.microsoft.com/en-us/javascript/api/manifest/webapplicationinfo?view=common-js-preview, 2022-06-25]

KIRKHAM, R., E. SAMUEL, L. L. CANNON u.a. (2022): Office Add-ins XML manifest. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/develop/add-in-manifests?tabs=tabid-1, 2022-06-25]

KIRKHAM, R., E. SAMUEL, S. RAMON u.a. (2022): Outlook JavaScript API requirement sets. Microsoft [https://docs.microsoft.com/en-us/javascript/api/requirement-sets/outlook/outlook-api-requirement-sets?view=common-js-preview&tabs=xmlmanifest, 2022-06-25]

KIRKHAM, R., E. SAMUEL, S. RAMON u. A. JERABEK (2022): Outlook JavaScript API requirement sets. Microsoft [https://docs.microsoft.com/en-us/javascript/api/requirement-sets/outlook/outlook-api-requirement-sets?view=common-js-preview&tabs=xmlmanifest#using-apis-from-later-requirement-sets, 2022-07-03]

KIRKHAM, R., E. SAMUEL, D. CHESNUT u.a. (2022): Register an Office Add-in that uses single sign-on (SSO) with the Microsoft identity platform. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/develop/register-sso-add-in-aad-v2#add-a-client-secret, 2022-07-10]

KIRKHAM, R., S. RAMON, A. JERABEK u.a. (2022): Configure your Outlook add-in for event-based activation. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/outlook/autolaunch, 2022-06-25]

LARKIN, S., J. EWALD, G. GRISOGONO, u.a. (unb.): Concepts, Webpack [https://webpack.js.org/concepts/, 2022-06-26]

LEPOW, D. (2022): What is Azure API Management. Microsoft [https://docs.microsoft.com/en-us/azure/api-management/api-management-key-concepts, 2022-07-09]

LEPOW, D., P. OVHAL, S. SAVELL u.a. (2022): API Management access restriction policies. Microsoft [https://docs.microsoft.com/en-us/azure/api-management/api-management-access-restriction-policies#ValidateJWT, 2022-07-09]

LIN, T., S. XU, ANONYM u.a. (2022): Verify first-party Microsoft applications in sign-in reports. Microsoft [https://docs.microsoft.com/en-us/troubleshoot/azure/active-directory/verify-first-party-apps-sign-in#application-ids-for-commonly-used-microsoft-applications, 2022-07-09]

LUBER, S. u. P. SCHMITZ (2017): Was ist Single Sign-on (SSO). Security Insider [https://www.security-insider.de/was-ist-single-sign-on-sso-a-631479/, 2022-07-24]

LUDWIG, N., M. MACY, ANONYM u.a. (2022): Microsoft identity platform and OAuth 2.0 On-Behalf-Of flow. Microsoft [https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-on-behalf-of-flow#middle-tier-access-token-request, 2022-07-29]

MARTIN, J., S. HAAS, T. MYERS u.a. (2022): Grant limited access to Azure Storage resources using shared access signatures (SAS). Microsoft [https://docs.microsoft.com/en-us/azure/storage/common/storage-sas-overview, 2022-07-09]

MURRAY, D., N. LUDWIG, M. MACY u.a. (2022): Microsoft identity platform access tokens. Microsoft [https://docs.microsoft.com/en-us/azure/active-directory/develop/access-tokens?WT.mc_id=addemystify-blog-masoucou, 2022-07-28]

O´BRYEN, N. u. S. J. BIGELOW (2020): Microsoft Exchange Server. TechTarget [https://www.techtarget.com/searchwindowsserver/definition/Microsoft-Exchange-Server, 2022-07-23]

RAMON, S., R. KIRKHAM, D. CHESNUT u.a. (2022): Build your first Outlook add-in. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/quickstarts/outlook-quickstart?tabs=yeomangenerator, 2022-06-24]

SAMUEL, E., D. CHESNUT, R. KIRKHAM u.a. (2022): Create a Node.js Office Add-in that uses single sign-on. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/develop/create-sso-office-add-ins-nodejs, 2022-07-29]

SAMUEL, E., R. KIRKHAM, S. RAMON u.a. (2022): ExtensionPoint element. Microsoft [https://docs.microsoft.com/en-us/javascript/api/manifest/extensionpoint?view=common-js-preview, 2022-06-25]

SAMUEL, E., S. RAMON, A. JERABEK u.a. (2022): Get or set the time when composing an appointment in Outlook. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/outlook/get-or-set-the-time-of-an-appointment, 2022-07-03]

SAMUEL, E., S. RAMON, R. KIRKHAM u.a. (2022): Use the Office dialog API in Office Add-ins. Microsoft [https://docs.microsoft.com/en-us/office/dev/add-ins/develop/dialog-api-in-office-add-ins, 2022-07-15]

TURETZKY, K., A. BUCK., S. PETERS. u.a. (2022): Selecting an API or technology for developing solutions for Outlook. Microsoft [https://docs.microsoft.com/en-us/office/client-developer/outlook/selecting-an-api-or-technology-for-developing-solutions-for-outlook, 2022-07-23]

WHITE, S. u. M. SATRAN. (2022): The Component Object Model. Microsoft [https://docs.microsoft.com/de-de/windows/win32/com/the-component-object-model. 2022-07-23]

Selbstständigkeitserklärung

Ich versichere, dass ich die Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

_____	_____
Datum	Unterschrift