



LINZER
TECHNIKUM

htl paul-hahn-straße

Höhere Technische Bundeslehranstalt
Höhere Lehranstalt für Informationstechnologie

HTL Diplomarbeit

SmartQS
***Microsoft Teams Erweiterung für manuelles
Software Testing***

ausgeführt im

Schuljahr 2020/21

eingereicht von

Birngruber Dominik

5AHIT

Werner Lukas

5AHIT

Betreut durch Hofer Susanne



I. Haftungsausschluss

Die Inhalte dieser Diplomarbeit wurden nach bestem Wissen und Gewissen erarbeitet. Es kann jedoch keine, wie immer geartete Verantwortung oder Haftung für deren Aktualität, Vollständigkeit oder Richtigkeit übernommen werden. Es handelt sich bei dieser Diplomarbeit im Übrigen keinesfalls um eine sogenannte „Ingenieurbüroarbeit“.

II. Eidesstattliche Erklärung

Wir/Ich erkläre(n) Eides statt, dass wir/ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht verwendet und die den benutzten Quellen inhaltlich beziehungsweise wörtlich entnommenen Stellen als solche kenntlich gemacht haben.

Linz, im April 2021

.....
Name

.....
Name

III. Danksagung

Wir möchten uns an dieser Stelle herzlich bei der Firma smartpoint IT consulting GmbH bedanken, welche uns die Durchführung dieser Diplomarbeit ermöglicht hat. Unser weiterer Dank gilt unserer LiTec Betreuerin Frau Susanne Hofer und Herrn Daniel Burgstaller, welcher uns als Firmenbetreuer zur Seite stand.

IV. Einleitung [DB]

Es gibt schon einige Tools für manuelles Testen von Software, doch keine ist allen Anforderungen gewachsen. Manche haben einen großen Funktionsumfang, sind jedoch teuer und unübersichtlich und andere haben schlichtweg nicht alle benötigten Funktionen.

Deshalb hat die Firma smartpoint uns die Möglichkeit gegeben, in Zusammenarbeit mit ihnen eine Alternative zu Azure DevOps und ähnlichen Programmen zu entwickeln. Unsere Lösung hat jedoch den Vorteil, dass sie leicht in Microsoft Teams eingebunden werden kann, da Teams oft als firmeninterne Kommunikationsplattform verwendet wird.

SmartQS, das im Rahmen dieser Diplomarbeit entwickelte Tool zum manuellen Testen, kann durch die verwendeten Technologien leicht in Firmen verwendet werden. Die Voraussetzung ist, dass firmenintern SharePoint verwendet wird und optional über Microsoft Teams kommuniziert wird. Es sind auch keine zusätzlichen Server notwendig, da das Backend leicht über Azure gehostet werden kann.

V. Abstract [DB]

There are many tools for manual software testing on the market right now, but none of these fulfill all criteria. While some have lot of functionality, these often tend to be more expensive and harder to use. Others tend to just have not all needed functionalities.

Therefore, the company smartpoint gave us the chance to collaborate with them in creating an alternative to Azure DevOps and similar products. Our product has the advantage of being integrated into Microsoft Teams, a product often used for communication in companies.

SmartQS, the tool for manual testing developed for this diploma thesis, can be used in companies that use SharePoint and use Microsoft Teams as their communications platform, because the used technologies were chosen specifically for that use case. Furthermore, the backend does not need any dedicated servers because it can be hosted in Azure.

VI. Versionsverlauf

<i>Version</i>	<i>Bearbeiter</i>	<i>Änderungen</i>	<i>Datum</i>
1.0	Dominik Birngruber	Erstellung der Allgemeinen Struktur, Verfassen von Zusammenfassung und Abstract, Beginn der Dokumentation des Backends	12.02.2021
1.1	Dominik Birngruber	Hinzufügen von Ist-Situation und Zielbestimmungen Hinzufügen von Theoretischen Grundlagen des Backends und Erweiterung von Implementierung Backend	20.02.2021
1.2	Dominik Birngruber	Hinzufügen der Theorie zur Versionsverwaltung und Erweiterung der Dokumentation des Backends	21.02.2021
1.3	Lukas Werner	Hinzufügen der Theorie zu Frontend React	26.02.2021
1.4	Lukas Werner	Hinzufügen der Implementierung des Frontends und die genauere Beschreibung der Projektstruktur	28.02.2021
1.5	Dominik Birngruber	Anpassen von einigen Texten und Ändern von einigen Bildern. Hinzufügen von Codeschnipseln und hinzufügen von vergleich zwischen NoSQL und SQL-Datenbanken	10.03.2021
1.6	Dominik Birngruber	Einarbeiten von vorgeschlagenen Änderungen	11.03.2021
1.7	Lukas Werner	Hinzufügen vom Aufbau der Navigation	11.03.2021

1.8	Lukas Werner	Hinzufügen der Design Bibliotheken, Hinzufügen der Home Seite	15.03.2021
1.9	Lukas Werner	Hinzufügen Implementierung des Dashboards, Tests durchführen, Tests erstellen Seite	17.03.2021
2.0	Lukas Werner	Hinzufügen von der Generierung und Einbindung vom Projekt	19.03.2021
2.1	Lukas Werner, Dominik Birngruber	Hinzufügen Zusammenfassung, Codesnippets hinzugefügt, Theorie zu Testen erweitert	24.03.2021
2.2	Lukas Werner, Dominik Birngruber	Hinzufügen von Theorie zu Versionsverwaltung, Überarbeitung von Rechtschreibfehlern	29.03.2021
2.3	Lukas Werner, Dominik Birngruber	Ausbessern von Rechtschreibfehlern, Hinzufügen der Arbeitsprotokolle	30.03.2021

Inhalt

I. Haftungsausschluss	2
II. Eidesstattliche Erklärung.....	3
III. Danksagung	4
IV. Einleitung [DB].....	5
V. Abstract [DB]	6
VI. Versionsverlauf.....	7
1. Ausgangssituation und Zielsetzung [DB]	1
1.1. Auftraggeber	1
1.2. Zielbestimmungen.....	1
1.3. Anforderungen	1
1.4. Projektteam.....	1
1.5. Arbeitsaufteilung.....	1
2. Theoretische Grundlagen	2
2.1. Software Testing [LW, DB]	2
2.1.1. Test Case [LW].....	2
2.1.2. Test Run/Definition [LW]	2
2.2. Node.js [DB]	3
2.3. TypeScript [DB]	3
2.4. Backend [DB]	4
2.4.1. Datenbank – MongoDB	4
2.4.2. Express	4
2.4.3. NoSQL vs. SQL	5
2.5. Frontend [LW]	6
2.5.1. React.....	6
2.5.2. Design-Bibliotheken	6
2.6. Versionsverwaltung [DB]	7
3. Implementierung Backend [DB].....	8

3.1.	Projektstruktur	8
3.2.	Implementierung der Datenbank	10
3.3.	Deployment auf Azure	11
3.4.	Dokumentation des Backends	11
3.5.	Implementierung der APIs.....	12
3.6.	Datenbankaufrufe mit Mongoose	15
4.	Implementierung Frontend [LW]	17
4.1.	Projektstruktur	17
4.2.	Projekt Erstellung	18
4.2.1.	Generierung	18
4.2.2.	Anpassungen und Einbindung.....	19
4.3.	Navigation	21
4.4.	Tabs.....	23
4.4.1.	Home.....	23
4.4.2.	Dashboard	25
4.4.3.	Tests durchführen	27
4.4.4.	Tests erstellen.....	28
5.	Zusammenfassung [DB]	30
6.	Literaturverzeichnis.....	31
7.	Anhang.....	32
7.1.	Arbeitsprotokolle	32
7.1.1.	Lukas Werner.....	32
7.1.2.	Birngruber Dominik.....	35
7.2.	API-Dokumentation	36

Abbildungsverzeichnis

Abbildung 1: Projektstruktur des Backends in VS Code	8
Abbildung 2: Design einer Card	12
Abbildung 3: Property Pane	17
Abbildung 4: Projektstruktur	17
Abbildung 5: Registerkarte.....	21
Abbildung 6: Registerkarte hinzufügen	21
Abbildung 7: Navigationsleiste	22
Abbildung 8: Home Seite mit ausgewähltem Test.....	23
Abbildung 9: Popup bei Testfall Klick	24
Abbildung 10: Dashboard Seite	25
Abbildung 11: Drilldown Graph in Testfall-Ansicht	26
Abbildung 12: Popup bei Testfall Klick.....	26
Abbildung 13: Tests durchführen Seite	27
Abbildung 14: Popup nach Testdurchführung.....	28
Abbildung 15: Tests erstellen Seite	29
Abbildung 16: Testfall bearbeiten Popup	29

Codesnippetverzeichnis

Codesnippet 1: Aufbau einer API	9
Codesnippet 2: Datenmodell.....	10
Codesnippet 3: Anpassung des Datums für korrekte Filterung	13
Codesnippet 4: Aufbau des Datenmodells in mongoose	15
Codesnippet 5: Datenbankverbindung	16
Codesnippet 6: Aufbau eines Datenbankaufrufs	16
Codesnippet 7: render Methode	20
Codesnippet 8: gulpfile.js	21
Codesnippet 9: Route Element.....	22
Codesnippet 10: Pivot Element für die Navigation	22
Codesnippet 11: Nav Element für Tests	27

1. Ausgangssituation und Zielsetzung [DB]

Zurzeit werden im Unternehmen des Auftraggebers Software-Tests noch manuell, also ohne Tool-Unterstützung durchgeführt. Um das Software-Testing zu erleichtern, soll deshalb ein Tool entwickelt werden, welches direkt über Microsoft Teams verwendet werden kann.

1.1. Auftraggeber

Der Auftraggeber dieses Projekts ist die Firma smartpoint IT consulting GmbH.

1.2. Zielbestimmungen

SmartQS ist eine App für Microsoft Teams, in der manuelle Software-Tests definiert werden können. Diese können anschließend durchgeführt und in einem Dashboard graphisch angezeigt werden. Zusätzlich sind auch genaue Informationen zu den durchgeführten Tests abrufbar.

1.3. Anforderungen

- Integrierbare Lösung mit MS-Teams (als Tab)
- Test Runs definieren
- Runs durchführen und Ergebnisse graphisch anzeigen
- UI so simpel wie möglich
- Speicherung sollte in Azure passieren

1.4. Projektteam

Das Projektteam besteht aus Lukas Werner und Dominik Birngruber

1.5. Arbeitsaufteilung

Das Projekt ist in Frontend und Backend aufgeteilt, wobei der Frontend-Teil von Lukas Werner und der Backend-Teil von Dominik Birngruber übernommen wurde.

2. Theoretische Grundlagen

2.1. *Software Testing* [LW, DB]

Manuelles Testen von Hardware oder Software wird meist mit verschiedenen Testdokumenten beschrieben, genauer gesagt Test Plans, Test Cases und Test Runs. Am Ende soll dabei ein Dokument entstehen, in dem aufgeschrieben wird, ob das erwartete Ergebnis aufgetreten ist und im Fehlerfall, welche Schritte nicht funktionierten. Dabei können auch Kommentare angefügt werden. Außerdem wird aufgezeichnet, wer den Test durchgeführt hat und wann der Test durchgeführt wurde. In dieser Arbeit werden Test Runs und Test Cases behandelt, welche die Durchführung von Tests beschreiben (vgl. [1]).

2.1.1. *Test Case* [LW]

Unter einem Test Case (Testfall) versteht eine Bedingung, welche es dem Tester erleichtert zu überprüfen, ob alle Anforderungen ausreichend erfüllt sind. Beim Erstellen dieser Test Cases erkennt man auch oft Probleme in den Anforderungen oder in der Anwendung (vgl. [1]). Test Cases enthalten in SmartQS einen Titel und eine genaue Beschreibung was zu tun ist. Diese Beschreibung kann bei der Erstellung mit HTML-Tags erstellt werden, dadurch können zum Beispiel Aufzählungspunkte oder Überschriften verwendet werden. Beim Durchführen von Tests wird diese Beschreibung dann beim Klick eines der Test Cases angezeigt. Somit ist die Benutzung übersichtlich und falls der Benutzer mehr Informationen benötigt, bekommt er diese auch.

2.1.2. *Test Run/Definition* [LW]

Ein Test Run enthält zusätzliche Daten zu einer Sammlung von Test Cases, die gemeinsam ausgeführt werden können:

- Tatsächliches Ergebnis
- Status
- Anmerkungen
- Erstellt von
- Datum der Erstellung

- Durchgeführt von
- Datum der Durchführung

Manche dieser Daten werden direkt bei der Test-Erstellung hinzugefügt. Andere Felder hingegen werden erst bei der Test-Durchführung gesetzt. Bei SmartQS werden die wichtigsten Daten eines Test Runs, der Name des Tests, die Frist und das Erstellungsdatum, schon bei der Erstellung gesetzt.

Bei der Durchführung der Tests werden weitere Daten festgelegt.

Ein Test gilt als abgeschlossen, wenn alle Test Cases erfolgreich waren oder der Test nach einen fehlerhaften Testfall abgebrochen wird. Nachdem ein Test Run durchgeführt wurde wird der Name des Testers gespeichert. Weiters wird abgespeichert wann dieser durchgeführt wurde und ob dieser Test erfolgreich oder fehlerhaft war (vgl. [1]).

2.2. Node.js [DB]

Sowohl das Frontend als auch das Backend verwenden Node.js, eine Laufzeitumgebung für JavaScript. Node.js ist eventbasiert und kann besonders gut für skalierbare Webanwendungen verwendet werden. Im Gegensatz zu anderen Modellen für Nebenläufigkeit verwendet Node.js Callback Funktionen und keine eigenen Betriebssystem-Threads. Dies hat den Vorteil, dass es keine so genannten Deadlocks gibt, einen Zustand, in dem sich Threads gegenseitig blockieren da sie auf den jeweils anderen Thread warten und so das System stillsteht. Die Entscheidung für Node.js wurde getroffen, da die Technologie beiden Diplomanden bekannt war, und es sich um eines der meistgenutzten Frameworks für JavaScript handelt (vgl. [2]).

2.3. TypeScript [DB]

Als Programmiersprache wurde auf TypeScript gesetzt, ein Superscript von JavaScript, entwickelt von Microsoft. TypeScript hat den Vorteil, dass es schon während der Programmierung auf Datentypen achtet, sodass es seltener während der Laufzeit zu Fehlern kommt. Ein weiterer Vorteil ist, dass jeder JavaScript Code auch gültiger TypeScript Code ist, und es so leicht ist, bereits vorhandene Codeteile wiederzuverwenden. TypeScript muss zur Ausführung in JavaScript kompiliert

werden, bietet jedoch im Gegensatz zu JavaScript Typisierung und leichte Verwendung von Klassen (vgl. [3]).

2.4. Backend [DB]

Im Backend wird für die Datenbank MongoDB auf Azure verwendet und die API's werden in Node.js und Express entwickelt.

2.4.1. Datenbank – MongoDB

MongoDB ist eine moderne dokumentorientierte Datenbank von MongoDB, Inc., welche Daten in einem JSON-ähnlichen Format abspeichert. Das bedeutet, dass ein Objekt auch Unterobjekte oder Arrays von Unterobjekten beinhalten kann. Daten müssen auch nicht immer gleich aufgebaut sein, wodurch jedes Objekt nur jene Felder speichert, welche auch benötigt werden. MongoDB bietet jedoch auch einige Vorteile von relationalen Datenbanken, wie ACID Transaktionen (siehe 2.4.3. NoSQL vs. SQL) und, Joins in Queries und nicht nur eingebettete Unterobjekte, sondern auch Verweise auf andere Objekte.

In der Diplomarbeit wird Azure zum Hosting der Datenbank verwendet. Hierbei wird als Datenbank CosmosDB gewählt und als Datenbanksystem MongoDB (vgl. [4]).

Alternativer Lösungsweg

Die Entscheidung für eine dokumentbasierte Datenbank wurde anhand des nachfolgenden Datenmodells getroffen. Dadurch, dass jede Testdefinition mehrere Testfälle hat, würde eine relationale Datenbank mit Fremdschlüsseln schnell an Komplexität steigen. Es ist auch sehr vorteilhaft, dass die verwendete Schnittstelle Mongoose Daten als JSON zurückgibt, sodass keine Konvertierung mehr notwendig ist.

2.4.2. Express

Express ist ein JavaScript Framework, welches von der OpenJS Foundation entwickelt wurde. Es bietet die Möglichkeit, Routen und Zugriffsmethoden zu definieren, was es zu einer guten Wahl für eine API¹ macht. Die Entscheidung

¹ Application programming interface: Eine Funktion, welche es anderen Programmen ermöglicht mit dem Programm zu interagieren

für Express wurde getroffen, da diese Technologie dem Diplomanden schon bekannt war und dem Diplomanden keine Alternative bekannt ist (vgl. [5]).

2.4.3. NoSQL vs. SQL

Ein großer Unterschied zwischen SQL und NoSQL-Datenbanken ist die Art, wie Daten gespeichert werden. In SQL-Datenbanken werden Daten in einzelnen Tabellen gespeichert, während bei NoSQL-Datenbanken die Daten, je nach Art der Datenbank, in JSON-Dokumenten, Key-Value Paaren, Tabellen, welche die Spalten dynamisch anpassen, oder in Nodes gespeichert werden. NoSQL-Datenbanken sind moderner, da sie in den 2000er Jahren entwickelt wurden, um aufkommenden Problemen bezüglich Skalierung entgegenzuwirken. Sie sind auch besser auf agile Softwareentwicklung abgestimmt. Während SQL als Allzweck-Datenbank eingesetzt wird, gibt es verschiedene NoSQL Datenbanken für verschiedene Zwecke, zum Beispiel für sehr große Datenmengen Key-Value-Paar Datenbanken und Graph-Datenbanken, um Beziehungen zwischen Daten zu analysieren. Der weitaus größte Vorteil ist, dass die Schemen flexibel sind, also der Datenaufbau leicht verändert werden kann, auch während der Programmierung. Die Skalierung erfolgt bei NoSQL-Datenbanken und SQL-Datenbanken auch unterschiedlich. Während die meisten SQL-Datenbanken vertikal skalieren, also für größere Datenmengen größere Server benötigt werden, skalieren NoSQL-Datenbanken horizontal, können also auf viele Server aufgeteilt werden.

Ein weiterer Unterschied ist, dass NoSQL-Datenbanken normalerweise keine Joins benötigen, um mehrere Tabellen zu vereinen, viele bieten jedoch trotzdem die Möglichkeit. Das führt dazu, dass Queries typischerweise etwas schneller sind als bei SQL-Datenbanken, besonders bei großen Datenmengen. Viele NoSQL-Datenbanken sind auch sehr praktisch für Entwickler, da viele, wie MongoDB, den Aufbau aus Klassen von gängigen Programmiersprachen generieren können.

Ein Vorteil von SQL-Datenbanken ist hingegen, dass ACID-Transaktionen auch für mehrere Einträge durchgeführt werden können. Viele NoSQL

Datenbanken können diese ACID-Transaktionen nicht immer garantieren, jedoch gibt es manche NoSQL-Datenbanken, welche ACID unterstützen, zum Beispiel das von uns genutzte MongoDB. ACID steht für atomicity, consistency, isolation und durability. Das bedeutet, dass Transaktionen unteilbar sind (atomicity), alle Daten gültig abgespeichert sind (consistency), Transaktionen sich nicht gegenseitig beeinflussen (isolation) und dass Transaktionen auch nach einem Systemfehler gespeichert bleiben (durability). Auch die Speichernutzung ist meist bei SQL-Datenbanken besser, da besonders dokumentbasierte NoSQL-Datenbanken oft Daten doppelt speichern und dadurch mehr Speicher benötigt wird, wobei in SQL-Datenbanken Daten meist nur einmal gespeichert werden (vgl. [4]).

2.5. Frontend [LW]

2.5.1. React

React ist ein JavaScript Framework, welches zum Erstellen von performanten Web Anwendungen verwendet wird. Es gibt verschiedene Frameworks wie Angular oder Vue.js. Wir haben uns für React entschieden, weil das Framework sehr einfach zu konfigurieren ist, und es einen fertigen Yeoman Generator besitzt, mit dem sehr einfach ein Tab für Microsoft Teams erstellt werden kann. Diese Methode wird offiziell von Microsoft empfohlen, denn der Generator setzt ein fertiges Projekt auf, sodass nach der erfolgreichen Projekterstellung (Build) die Lösung direkt in Microsoft Teams eingebunden werden kann (vgl. [6]).

2.5.2. Design-Bibliotheken

Fluent UI

Die Fluent UI Bibliothek von Microsoft wurde für fast alle Design Elemente verwendet. Die Bibliothek bietet viele Komponenten an, die für sämtliche React Projekte verwendet werden können.

Ein weiterer Vorteil dieser Bibliothek ist, dass sich die primären Farben der Design Elemente automatisch an das Design Schema der Microsoft SharePoint Organisation oder des Microsoft Teams Team anpassen. Das heißt

wenn in einem Team alle Buttons die standardmäßig enthalten sind, die Farbe Blau haben passen sich die Elemente dieser App automatisch an diese Farbe an, sodass dann alle Buttons die gleiche Farbe haben (vgl. [7]).

PnP SPFx Controls

Für die restlichen Design Elemente wurden die PnP SPFx Controls verwendet. Diese Bibliothek wurde nicht direkt von Microsoft entwickelt, sondern ist ein Open Source Projekt von der Microsoft 365 Community. Diese Bibliothek wurde hauptsächlich entwickelt, um schon vorhandene Design Elemente von SharePoint nachzuprogrammieren, um diese dann öffentlich zugänglich zu machen (vgl. [8]).

2.6. Versionsverwaltung [DB]

Als Versionsverwaltung wird GitHub verwendet, eine Plattform, welche es ermöglicht git repositories online zu hosten. GitHub wurde von GitHub, Inc. entwickelt, welche 2018 von Microsoft gekauft wurde. Git hingegen ist das dahinterliegende open source Versionsverwaltungstool. Git ist das am öftesten verwendete Versionsverwaltungstool. Dabei hat jeder Bearbeiter des repositories eine lokale Kopie und kann dadurch auch ohne Netzwerkverbindung an einem Projekt arbeiten und sobald wieder eine Netzwerkverbindung besteht können Änderungen mit anderen Bearbeitern synchronisiert werden. Git bietet auch die Möglichkeit, verschiedene Entwicklungszweige, so genannte „branches“ anzulegen, damit zum Beispiel die Arbeit am Frontend getrennt von der Arbeit am Backend erfolgen kann, ohne unabsichtlich Daten zu überschreiben. Da alle Änderungen gespeichert werden, kann jederzeit das Projekt auf eine vorherige Version zurückgesetzt werden (vgl. [9]).

3. Implementierung Backend [DB]

3.1. Projektstruktur

Das Backend wird als Node.js Server realisiert, welcher verschiedene APIs freigibt. Dieser Server dient als Schnittstelle zwischen der MongoDB Datenbank, welche in Azure gehostet wird, und dem Frontend. Jede API ist auf zwei Teile aufgeteilt, um die Übersichtlichkeit der Dateien zu gewähren. Ein Teil gibt die Route für Express an und beinhaltet einen Aufruf für die Funktion, welche die Funktionalität bietet. Der zweite Teil ist dafür zuständig, die Anfragen auf die Datenbank zu machen, die erhaltenen Daten weiterzuverarbeiten und die Daten

an den Aufrufer zurückzugeben. Über der Funktion für Express befindet sich als Kommentar die Dokumentation für die API (siehe 3.5 Dokumentation des Backends). Das Codesnippet 1 zeigt den Aufbau einer solchen Funktion.

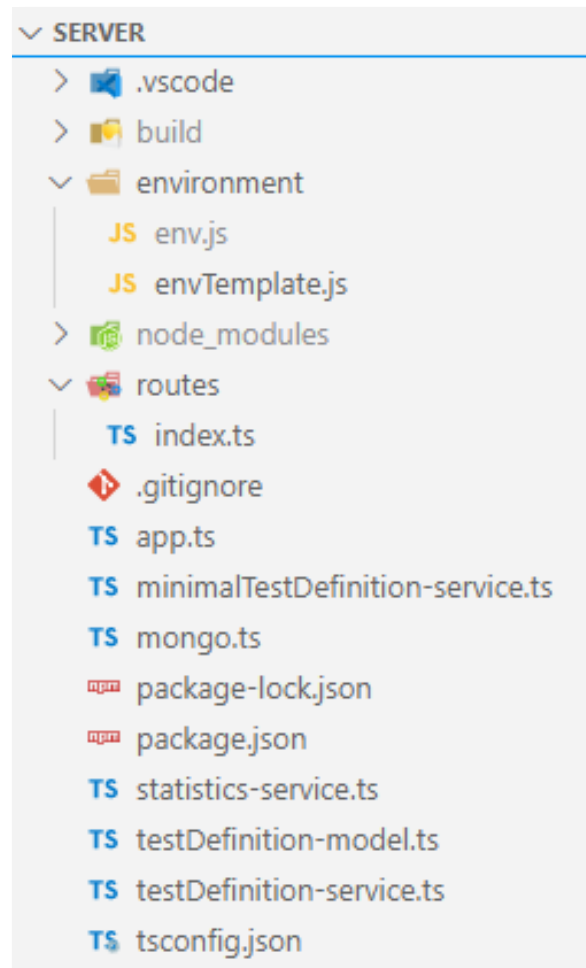


Abbildung 1: Projektstruktur des Backends in VS Code

```
//Router von Express
router.get("/testDefinitions", function (req: any, res: any) {
  testDefinitionService.get(req, res);
});

//Hilfsfunktion mit Datenbankaufruf
/**
 * Returns all Test Definitions from the Database
 * @author Dominik Birngruber
 * @param req
 * @param res
 */
function get(req: any, res: any) {
  //Datenbankaufruf und Rückgabe der Daten
}
```

Codesnippet 1: Aufbau einer API

Die Funktionen, welche die Verbindung mit der Datenbank herstellen sind auf 3 Dateien aufgeteilt, welche jeweils Funktionen für eine Art von Abfrage beinhaltet. Diese heißen *testDefinitionService.ts*, *statistics-service.ts* und *minimalTestDefinition-service.ts*. Einige Variablen wie der Datenbank-Verbindungsstring werden in einer eigenen Datei gespeichert, welche nicht in die Versionsverwaltung aufgenommen wird, um unberechtigten Zugriff zu verhindern.

3.2. Implementierung der Datenbank

Zur Nutzung von MongoDB muss nur der MongoDB Server von der offiziellen MongoDB Seite heruntergeladen werden, oder wie in unserem Fall bei einem Anbieter gemietet werden. Anschließend verbindet man sich mit dem Server über die festgelegten Anmeldeinformationen, es muss keine Tabelle erstellt werden oder Struktur angegeben werden, dies geschieht automatisch.

```
[
  {
    "_id": "string",
    "name": "string",
    "createdOn": "string",
    "tester": "string",
    "finished": true,
    "deadline": "string",
    "doneOn": "string",
    "channelID": "string",
    "__v": 0,
    "testCases": [
      {
        "_id": "string",
        "title": "string",
        "description": "string",
        "status": "string",
        "active": true,
        "comments": "string",
        "image": "string",
        "required": true
      }
    ]
  }
]
```

Codesnippet 2: Datenmodell

Die Datenbank hat nur eine Tabelle, welche die durchzuführenden und durchgeführten Tests beinhaltet. MongoDB hat den Vorteil, dass direkt Objekte in der Datenbank gespeichert werden, also sind keine weiteren Tabellen für Test Cases notwendig.

Die Felder `_id` und `__v` werden von MongoDB automatisch erstellt, wobei `_id` eine eindeutige Zeichenkette ist und `__v` angibt, welche Version des Objekts das

gespeicherte Objekt ist. `createdOn`, `doneOn` und `deadline` sind jeweils ein Datum, welches in der Form `YYYY-MM-DDTHH:mm:ss.sssZ` angegeben wird. Das Feld `finished` gibt an, ob ein Test bereits beendet wurde. Das Feld `testCases` ist ein Array, in welchem die Schritte zur Testdurchführung gespeichert sind. Jeder Schritt hat eine automatisch generierte `id`, einen Titel, eine Beschreibung, einen Status, ein Feld für Kommentare und ein Feld, in dem ein Link zu einem Bild eingefügt werden kann. Es gibt auch noch die Boolean Felder `active`, welches beschreibt, ob ein Test aktiv ist und `required`, welches angibt, ob ein Test benötigt oder optional ist. Jeder Datensatz wird genau einem Kanal in Teams mit dem Feld `channelID` zugeordnet.

3.3. Deployment auf Azure

Das gesamte Backend wird auf Azure gehostet. Die Entscheidung für Azure wurde getroffen, da das fertige Produkt über SharePoint benutzt wird und es möglich ist, Zugriffe auf die API über das Azure Active Directory ohne großen Mehraufwand einzuschränken.

Das Hochladen der API geschieht direkt über Visual Studio Code. Da das Backend in TypeScript programmiert wurde, muss zuerst ein `build` Ordner generiert werden, in dem der in JavaScript übersetzte Code enthalten ist. Außerdem muss ein `package.json` und das `environment-file` enthalten sind. Dieser Ordner wird anschließend hochgeladen und auf dem Server ausgeführt.

3.4. Dokumentation des Backends

Als API Dokumentationstool wurde Swagger von Smartbear eingesetzt. Dieses Tool bietet die Möglichkeit, die Dokumentation und die APIs über die gleiche Adresse zu erreichen. Dazu muss man nur Kommentare zum Code hinzufügen und das Tool erstellt automatisch eine interaktive Website, auf der die APIs auch direkt getestet werden können. In dieser Arbeit wurde die Open Source Implementierung namens Swagger UI verwendet. Diese kann aufgerufen werden, indem man an den Server-Pfad `/api-docs` anhängt, somit hat man eine Interaktive Dokumentation, in der man Aufrufe auch gleich testen kann.

Mit SwaggerUI wird genau beschrieben, welche Felder die Anfragen beinhalten müssen und ob ein Feld im body oder in der URL mitgegeben werden muss.

Diese Website ist auch in gespeicherter Form im Doc Ordner des git Repositories.

Die Funktionen, welche auf die Datenbank zugreifen wurden mit JSDoc dokumentiert (vgl. [10]).

3.5. Implementierung der APIs

Es stehen im Backend 14 Endpoints zur Verfügung, wobei acht Endpoints noch mit Übergabe eines zusätzlichen Parameters eingegrenzt werden können. Die Aufrufe können noch weiter unterteilt werden in APIs für Testdefinitionen, minimale Testdefinitionen und Erfolgsstatistiken. Für Testdefinitionen, minimale Testdefinitionen und die zwei Statistik APIs ist es auch möglich, sie nach Durchführungszeit einzugrenzen.

Die APIs, welche Testdefinitionen liefern, geben die gesamten Daten zurück (siehe Codesnippet 2: Datenmodell). Die APIs für die minimalen Testdefinitionen geben ähnliche Daten zurück, jedoch werden bei diesen die Test Cases weggelassen. So müssen weniger Daten übertragen werden, was die Performance steigert. Die Statistik-APIs geben im Falle von Testdefinitions-Statistiken die Anzahl der erfolgreichen, fehlgeschlagenen und noch nicht durchgeführten Tests zurück. Im Fall von Test Case-Statistiken wird die Anzahl von erfolgreichen, fehlgeschlagenen, optionalen und nicht durchgeführten Test Cases zurückgegeben.

Bei der API zum Updaten von Tests besteht die Möglichkeit, einen Webhook für einen MS Teams Chat mitzugeben. Wenn dieser vorhanden ist und der Test als

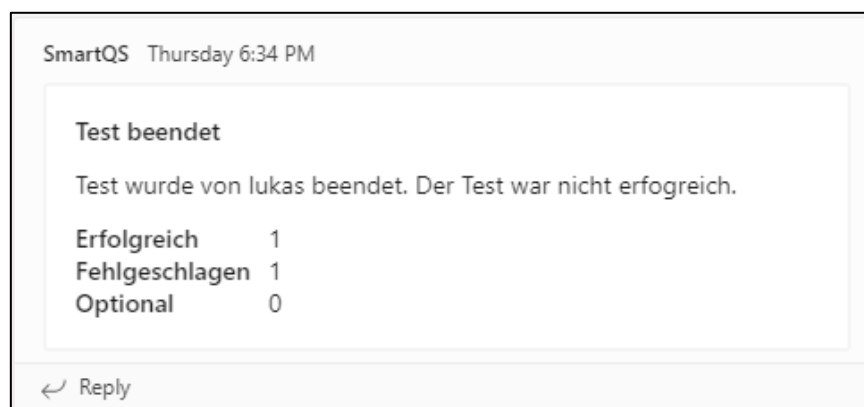


Abbildung 2: Design einer Card

fertiggestellt markiert wird, wird eine adaptive Card an diesen Webhook gesendet. In dieser Card werden der Name, der Tester, der Erfolgsstatus und die Test Case-Statistik für den jeweiligen Test angezeigt ohne nicht durchgeführte Test Cases.

Bei den APIs, bei denen man beim Aufruf ein Datum übergibt, ist anfangs das Problem aufgetreten, dass es nur bis zu einer bestimmten Uhrzeit filterte, obwohl der ganze Tag benötigt wurde. Deshalb musste das Datum so bearbeitet werden, dass das Startdatum immer 00:00:00 war und das Enddatum 23:59:59.

```
const startDate = Date.parse(
  req.body.startDate.split("T")[0] + "T00:00:00.000Z"
);
const endDate =
  req.body.endDate == null
    ? Date.now()
    : Date.parse(req.body.endDate.split("T")[0] + "T23:59:59.999Z");
```

Codesnippet 3: Anpassung des Datums für korrekte Filterung

GET-Requests

- `/testDefinitions/{channelID}`
Diese Anfrage gibt alle Test Definitionen zurück mit optionaler Einschränkung nach Kanal ID.
- `/finishedTestDefinitions`
Diese Anfrage gibt alle Test Definitionen zurück, welche schon als fertiggestellt markiert wurden.
- `/testDefinitionById/{_id}`
Diese Anfrage gibt die Test Definition zurück, welche die angegebene ID hat.
- `/testCasesByDefinitionId/{_id}`
Diese Anfrage gibt alle Test Cases der Definition zurück, welche die angegebene Definition hat.
- `/minimalTestDefintions/{channelID}`
Diese Anfrage gibt alle Definitionen ohne Test Cases zurück. Diese kann optional mit einer Kanal ID eingeschränkt werden.
- `/getSuccessStatistics/{channelID}`

Diese Anfrage gibt die Erfolgsstatistiken der Test Definitionen zurück. Diese kann optional mit einer Kanal ID eingeschränkt werden.

- `/getTestCaseSuccessStatisticsByTimePeriod(/{channelID})`

Diese Anfrage gibt die Erfolgsstatistiken der Test Cases zurück. Sie kann optional nach der Kanal ID eingeschränkt werden.

POST-Requests

- `/addTestDefinition`

Mit dieser Anfrage kann man eine neue Test Definition einfügen.

- `/updateTestDefinition/{_id}`

Diese Anfrage ändert die mit der `_id` angegebenen Definition.

- `/getDefinitionsByTimePeriod(/{channelID})`

Diese Anfrage gibt alle Test Definitionen, welche in der angegebenen Zeitspanne fertiggestellt wurden, zurück. Sie kann optional nach Kanal ID eingeschränkt werden.

- `/getMinimalTestDefinitionsByTimePeriod(/{channelID})`

Diese Anfrage gibt alle Test Definitionen ohne Test Cases zurück, welche in der angegebenen Zeitspanne fertiggestellt wurden. Sie kann optional noch nach Kanal ID eingeschränkt werden.

- `/getSuccessStatisticsByTimePeriod(/{channelID})`

Diese Anfrage gibt die Erfolgsstatistiken der Test Definitionen zurück, welche in der angegebenen Zeitspanne fertiggestellt wurden. Sie kann optional noch nach Kanal ID eingeschränkt werden.

- `/getTestCaseSuccessStatisticsByTimePeriod(/{channelID})`

Diese Anfrage gibt die Erfolgsstatistiken der Test Cases zurück, welche in der angegebenen Zeitspanne fertiggestellt wurden. Sie kann optional noch nach Kanal ID eingeschränkt werden.

DELETE-Request

- `/deleteTestDefinition/{_id}`

Diese Anfrage löscht die angegebene Test Definition.

3.6. Datenbankaufrufe mit Mongoose

Datenbankaufrufe werden mithilfe von Schemen gemacht. Dabei muss für jedes Dokument oder jedes Objekt in einem Dokument ein eigenes Schema erstellt werden. Ebenfalls muss, da das Projekt in TypeScript programmiert wurde, ein Interface für den Aufbau der Daten spezifiziert werden. Anschließend muss aus diesem Schema ein neues Model gemacht werden, welches anschließend mit export für andere Dateien nutzbar gemacht wird.

Schema	Interface
<pre>const testDefinitionSchema = new Schema({ name: String, createdOn: String, testCases: [testCaseSchema], tester: String, finished: Boolean, deadline: String, doneOn: String, channelID: String, });</pre>	<pre>export interface TestDefinition extends Document { name: String; createdOn: String; testCases: [TestCase]; tester: String; finished: Boolean; deadline: String; doneOn: String; channelID: String; }</pre>
<pre>export default model<TestDefinition>("TestDefintion", testDefinitionSchema);</pre> <p>Codesnippet 4: Aufbau des Datenmodells in mongoose</p>	

Die Nutzung des Models setzt eine erfolgreiche Verbindung mit der Datenbank voraus (vgl. Codesnippet 5: Datenbankverbindung).

```
import mongoose from "mongoose";
const env = require("./environment/env");
(<any>mongoose).Promise = global.Promise;
const mongoUri = env.dbName;

function connect() {
  var test = mongoose
    .connect(mongoUri, { useNewUrlParser: true,
      useUnifiedTopology: true, })
    .then(() => {
      console.log("Connected succesfully");
    })
    .catch((err) => {
      console.log("Could not connect");
    });
}
```

```
    return test;
  }
  module.exports = {
    connect, mongoose,
  };
};
```

Codesnippet 5: Datenbankverbindung

Ein Aufruf kann mit diesem Model erfolgen, nachdem man sich mit der Datenbank verbunden hat, indem man die verschiedenen Hilfsfunktionen verwendet. Beispielsweise kann man mit `model.find({})` alle Daten erhalten. Diese können anschließend mit `.then()` weiterverarbeitet werden oder fall ein Fehler auftritt, kann dieser mit `.catch()` behandelt werden.

```
TestDefinition.find({})
  .read(ReadPreference.NEAREST)
  .then((tests) => {
    res.json(tests);
  })
  .catch((err) => {
    res.status(500).send(err);
  });
```

Codesnippet 6: Aufbau eines Datenbankaufrufs

4. Implementierung Frontend [LW]

4.1. Projektstruktur

Das Frontend wird als NodeJS Server realisiert, welcher die Grundstruktur für die Benutzeroberfläche beschreibt. Nach der erfolgreichen Projekterstellung (Build) wird eine solution erstellt, welche dann in SharePoint eingefügt wird. Danach kann die App ganz einfach in Microsoft Teams ausgewählt werden. Das Projekt beginnt mit der `SmartqsWebPart.ts` Datei, dort werden die Property Pane Eigenschaften definiert und gelesen. Die Einstellungen können beim Bearbeiten des Web-Part aufgerufen werden, dort lassen sich die Graphen vom Dashboard an- und ausschalten. Nach den Web-Part Eigenschaften wird `Smartqs.tsx` aufgerufen, darin befinden sich grundsätzliche Initialisierungsschritte. Zu Beginn werden Icons initialisiert, die für den späteren Gebrauch wichtig sind. Weiters wird von den SharePoint Tenant Feldern der Organisation die URL für den Backend Server abgerufen. Diese sollte als Tenant Property mit den Namen `smartqsserviceurl` abgespeichert werden.

Nach diesen Initialisierungsschritten wird die allgemeine Navigationsleiste, die aufgerufen wird. Diese ist in `SmartQSNV.tsx` enthalten, dort ist das Pivot Element mit den verschiedenen Links enthalten. Diese leiten dann auf die verschiedenen Seiten weiter: z.B.: Dashboard, Tests erstellen, Tests durchführen, usw. All diese Seiten und andere Komponenten, die verwendet werden, sind im `components` Ordner

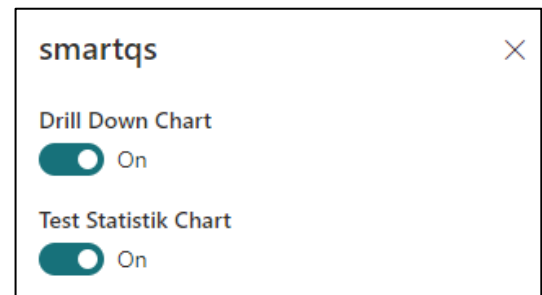


Abbildung 3: Property Pane

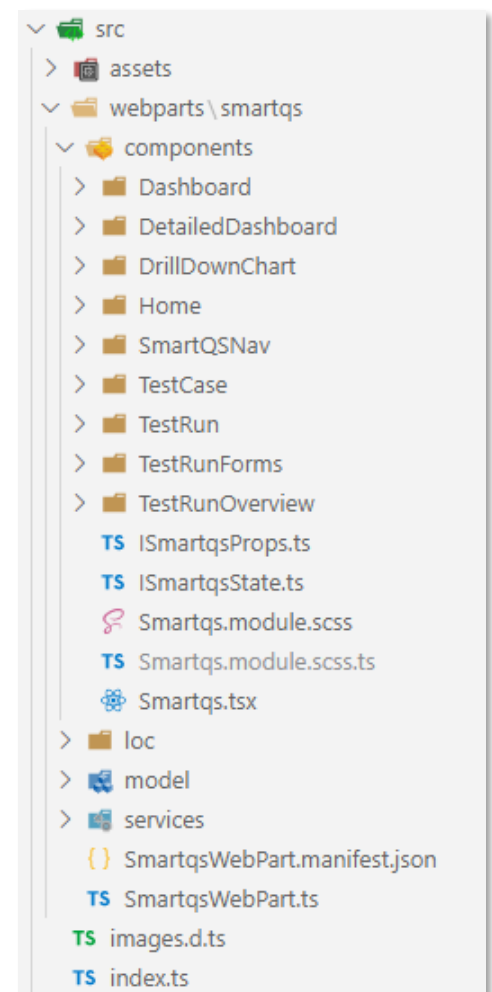


Abbildung 4: Projektstruktur

enthalten. Dort werden sie unter anderem von anderen Seiten aufgerufen, die diese Komponenten dann benötigen.

4.2. Projekt Erstellung

4.2.1. Generierung

Das Projekt wird mit einem Yeoman Generator erstellt. Diese Bibliothek bietet viel Möglichkeiten ein Node.js Projekt aufzusetzen.

Zu Beginn muss Node.js mit einer 10.x Version installiert werden.

Nun muss ein Ordner mit den bevorzugten Projektnamen erstellt werden und innerhalb eine Kommandozeile geöffnet werden. Danach muss der Yeoman Generator mit den npm package manager installiert werden: `npm install yo -global`. Danach muss gulp.js, ein Tool das Aufgaben automatisiert, mit `npm install gulp@3.9.1 -global` installiert werden. Jetzt muss der Generator, der speziell für SharePoint Framework benützt wird, installiert werden: `npm install @microsoft/generator-sharepoint -global`. Nun kann mit den Befehl `yo @microsoft/sharepoint` der Generator gestartet werden. Jetzt müssen folgende Schritte befolgt werden (vgl. [6]):

1. What is your solution name?
 - a. Enter Taste drücken, um den Ordner Namen zu wählen
2. Which type of client-side component to create?
 - a. WebPart auswählen
3. Which baseline packages do you want to target for your component(s)?
 - a. SharePoint Online only (latest) auswählen
4. Where do you want to place the files?
 - a. Use the current folder auswählen
5. Do you want to allow the tenant admin the choice of being able to deploy the solution to all sites immediately without running any feature deployment or adding apps in sites?
 - a. y auswählen

6. Will the components in the solution require permissions to access web APIs that are unique and not shared with other components in the tenant?
 - a. N auswählen
7. Which type of client-side component to create?
 - a. WebPart auswählen
8. What is your Web part name?
 - a. Den gewünschten Web part Namen eingeben.
9. What is your Web part description?
 - a. Die gewünschte Beschreibung eingeben.
10. Which framework would you like to use?
 - a. React auswählen

4.2.2. Anpassungen und Einbindung

Um die Webanwendung in Microsoft Teams einbinden zu können, müssen noch einige Anpassungen gemacht werden.

Zu Beginn muss in der [Web part Name]Webpart.manifest.json Datei beim Feld supportedHosts der Inhalt zu "supportedHosts": ["SharePointWebPart", "TeamsTab"], verändert werden. Danach muss in der I[Web part Name]Props.ts Datei die Zeile context: WebPartContext; zum Interface hinzugefügt werden. Anschließend muss in der [Web part Name]Webpart.ts Datei unterhalb von description: this.properties.description, die Zeile context: this.context, hinzugefügt werden. Danach muss in der [Web part name].tsx Datei die render Methode zu folgenden Inhalt verändert werden:

```
let title: string = "";
let subTitle: string = "";
let siteTabTitle: string = "";
if (this.props.context.sdks.microsoftTeams) {
  // We have teams context for the web part
  title = "Welcome to Teams!";
  subTitle = "Building custom enterprise tabs for your business.";
  siteTabTitle =
    "We are in the context of following Team: " +
    this.props.context.sdks.microsoftTeams.context.teamName;
```

```
} else {  
  // We are rendered in normal SharePoint context  
  title = "Welcome to SharePoint!";  
  subTitle = "Customize SharePoint experiences using Web Parts.";  
  siteTabTitle =  
    "We are in the context of following site: " +  
    this.props.context.pageContext.web.title;  
}  
  
return (  
  <div className={styles.sandbox}>  
    <div className={styles.container}>  
      <div className={styles.row}>  
        <div className={styles.column}>  
          <span className={styles.title}>{title}</span>  
          <p className={styles.subTitle}>{subTitle}</p>  
          <p className={styles.description}>{siteTabTitle}</p>  
          <p className={styles.description}>  
            Description property value - {escape(this.props.description)}  
          </p>  
          <a href="https://aka.ms/spfx" className={styles.button}>  
            <span className={styles.label}>Learn more</span>  
          </a>  
        </div>  
      </div>  
    </div>  
  </div>  
</div>  
>);
```

Codesnippet 7: render Methode

Für weitere Anpassung muss die Bibliothek gulp-sequence installiert werden: `npm install --save-dev gulp-sequence`. In der `gulpfile.js` Datei muss der Inhalt folgendermaßen geändert werden:

```
"use strict";  
  
// check if gulp dist was called  
if (process.argv.indexOf("dist") !== -1) {  
  // add ship options to command call  
  process.argv.push("--ship");  
}  
  
const build = require("@microsoft/sp-build-web");  
const gulp = require("gulp");  
const gulpSequence = require("gulp-sequence");
```



```

build.addSuppression(
  `Warning - [sass] The local CSS class 'ms-
  Grid' is not camelCase and will not be type-safe.`
);

gulp.task("dist", gulpSequence("clean", "bundle", "package-solution"));

build.initialize(gulp);

```

Codesnippet 8: gulpfile.js

Jetzt kann mit `gulp dist -ship` das Projekt gebaut werden. Nach der erfolgreichen Projekterstellung (Build) muss der App Catalog von SharePoint geöffnet werden. Diese befindet sich bei folgender URL:
[https://\[Tenant\].sharepoint.com/sites/apps/AppCatalog](https://[Tenant].sharepoint.com/sites/apps/AppCatalog).

Nun muss die `.sppkg` Datei, die sich im `/sharepoint/solution` Verzeichnis befindet in den AppCatalog reingezogen werden. Jetzt erscheint ein PopUp, das Kontrollkästchen muss angekreuzt werden und dann muss auf `Deploy` gedrückt werden. Danach muss die hochgeladene Solution markiert werden und in der Leiste oben unter Files der `Sync to Teams` Button gedrückt werden.



Abbildung 5: Registerkarte

Nun kann in Microsoft Teams überall die App eingebunden werden. Dies kann man durch das Hinzufügen einer Registerkarte machen, dort sollte dann die App auswählbar sein.

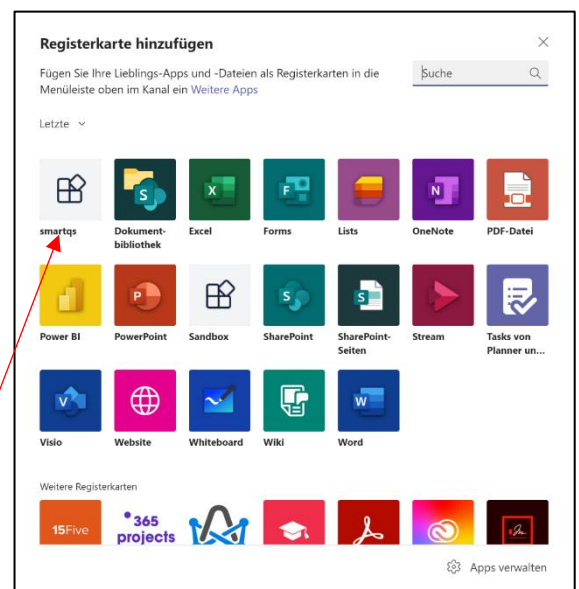


Abbildung 6: Registerkarte hinzufügen

4.3. Navigation

Um zwischen verschiedene Seiten zu wechseln wird eine Navigationsleiste verwendet.

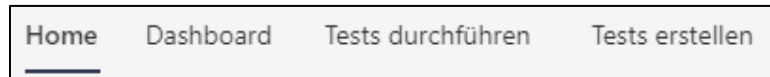


Abbildung 7: Navigationsleiste

Beim Klick eines der Links wird der Benutzer automatisch an den richtigen Inhalt weitergeleitet. Die grundsätzliche Navigation in einer React App wird meistens mit React-Routing umgesetzt. Bei herkömmlichen HTML Seiten funktioniert das anders, dort wird auf eine neue Datei weitergeleitet was vor allem die Ladezeiten verlängert. Das Prinzip dieser React-Routing Bibliothek ist es, dass sich der Inhalt der Seite in Abhängigkeit der URL ändert. Dies funktioniert so, dass man in einer Haupt Datei, in diesem Fall `SmartQSNV.tsx`, festlegt zu welcher URL welcher Inhalt angezeigt wird.

```
<Route path="/dashboard">
  <Dashboard/>
</Route>
```

Codesnippet 9: Route Element

Hier wird mit dem Route Element zum Beispiel festgelegt, wenn die Webseiten URL „/dashboard“ beträgt soll die Dashboard Komponente angezeigt werden. Im Gegensatz zu HTML Seiten wird hier nur der Inhalt ausgetauscht und nicht die komplette Seite neugeladen. Dadurch sind die Ladezeiten weniger und die Webseite ist auch mehr benutzerfreundlich.

Das Design der Navigationsleiste wurde mit den Pivot Element aus der Microsoft Fluent UI Bibliothek [7] umgesetzt.

```
<Pivot
  onLinkClick={(e) => {
    this.props["history"].push(e.props.itemKey);
  }}
>
  <PivotItem headerText="Home" itemKey="/home" />
  <PivotItem headerText="Dashboard" itemKey="/dashboard" />
  <PivotItem headerText="Tests durchführen" itemKey="/runTest" />
  <PivotItem headerText="Tests erstellen" itemKey="/createTest" />
</Pivot>
```

Codesnippet 10: Pivot Element für die Navigation

Hier wird mit dem „onLinkClick“ Event, das bei einem Klick eines der Navigationselemente gefeuert wird, die URL verändert. Diese URL wird dann von der vorigen erwähnten Route Element abgefangen und der Inhalt wird dann somit entsprechend angepasst.

4.4. Tabs

4.4.1. Home

Die Home Seite wird standardmäßig angezeigt und beinhaltet die grundsätzlichen Inhalte die wichtig sind. Der Hauptinhalt besteht aus einer Navigationsleiste von Tests. Im Gegensatz zu dem „Test durchführen“ Tab werden hier nur durchgeführte Tests angezeigt. Das heißt die Tests sind entweder erfolgreich oder fehlerhaft. Ein weiterer Zusatz im Vergleich zum Durchführen der Tests ist die Anzeige, wann der Test durchgeführt wurde. Die ganze Ansicht soll als schnelleren Überblick dienen, um alle letzten durchgeführten Tests auf einmal zu sehen. Weiters gibt es noch einen Filter Möglichkeit, um die angezeigten Tests zeitlich einzugrenzen.

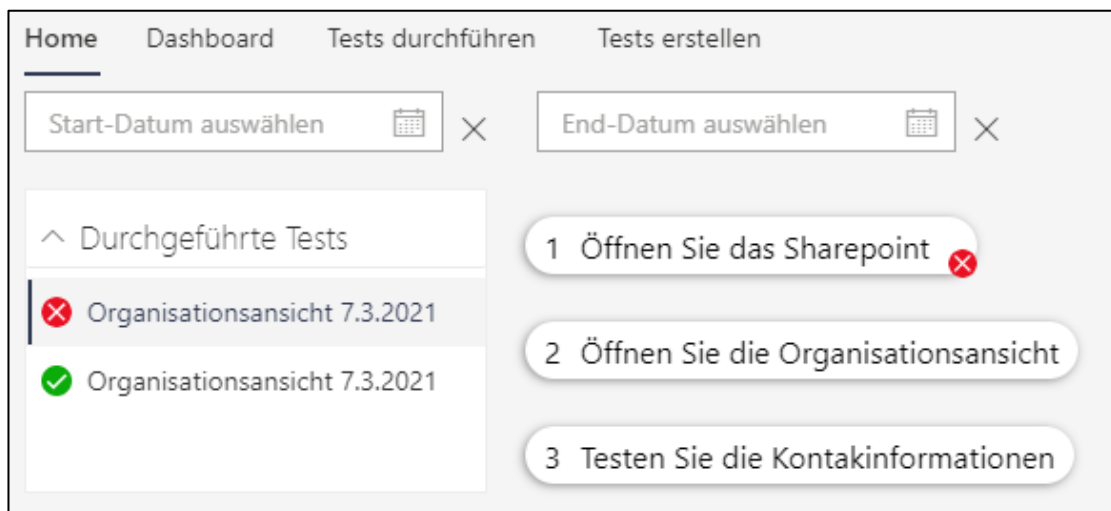


Abbildung 8: Home Seite mit ausgewähltem Test

Beim Klick eines der Tests in der Navigationsleiste werden die einzelnen Testfälle angezeigt. Dort lässt sich auch auslesen welche Fälle fehlerhaft oder erfolgreich sind. Beim Klick eines der Testfälle wird die Beschreibung des Testfalls angezeigt und falls der Test fehlerhaft bzw. optional ist, wird auch eine Fehlermeldung bzw. eine Info angezeigt.

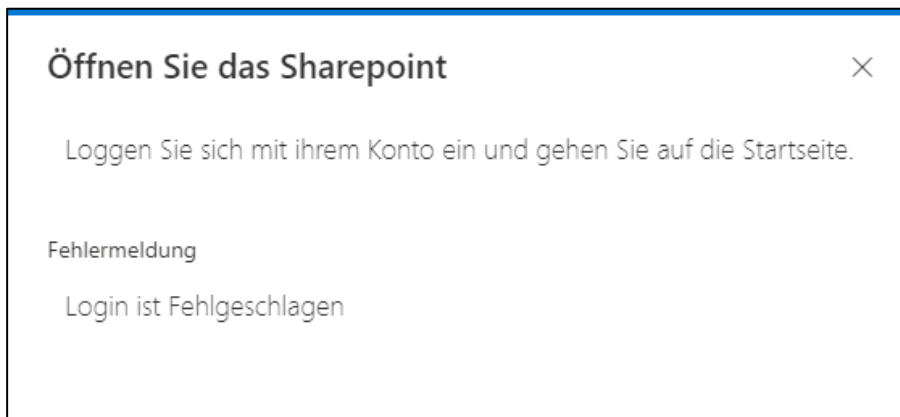


Abbildung 9: Popup bei Testfall Klick

4.4.2. Dashboard

Die Dashboard Seite dient zur Veranschaulichung sämtlicher Daten. Der obere Bereich beinhaltet zwei Statistik Graphen für eine schnelle Übersicht. Das linke Diagramm beschreibt wie viele Tests erfolgreich, fehlerhaft oder noch nicht durchgeführt sind. Im rechten Graph werden die einzelnen Testfälle beschrieben, d.h. wie viele von diesen als erfolgreich, fehlerhaft oder optional markiert worden sind. Außerdem wird angezeigt wie viele Testfälle noch nicht durchgeführt sind.



Abbildung 10: Dashboard Seite

Im Unteren Bereich wird ein sogenannter „Drill Down“ Graph für alle durchgeführten Tests angezeigt. Das Prinzip dahinter ist das der User durch Klick auf dem Graphen noch mehr Details erfährt.

In unserem Beispiel werden standardmäßig alle durchgeführten Tests angezeigt. Über die Legende können auch Tests ausgeblendet werden, falls diese nicht benötigt werden. Außerdem wird beim Hovern über den Tests der Name und der Status angezeigt.

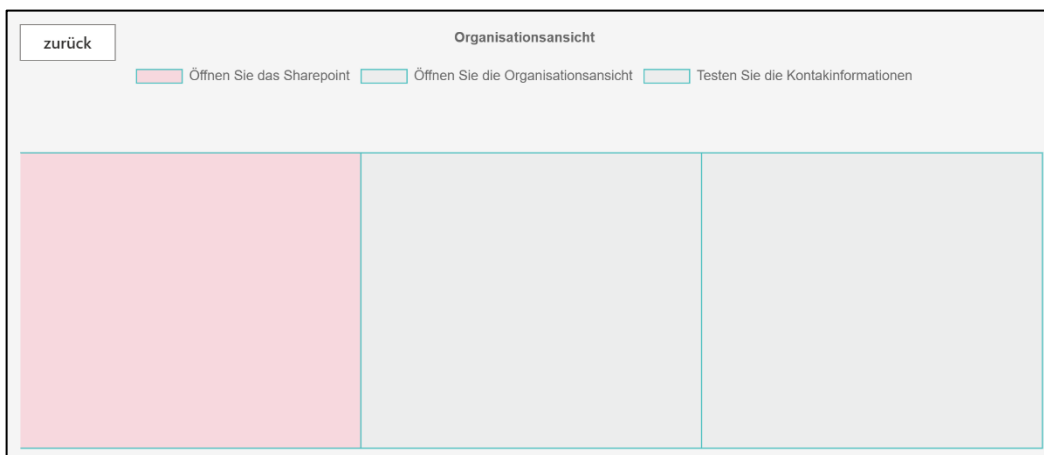


Abbildung 11: Drilldown Graph in Testfall-Ansicht

Beim Klick auf einen der Tests werden dann die einzelnen Testfälle angezeigt. Dadurch erreicht man eine gute Übersicht, wie zum Beispiel bei welchem Testfall der Test fehlgeschlagen ist.

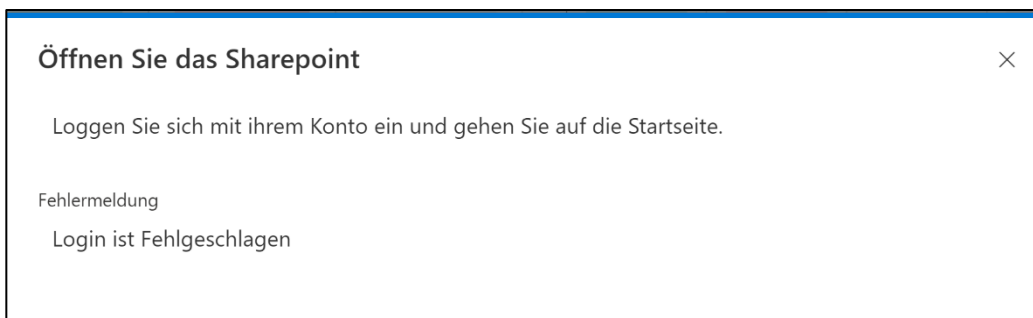


Abbildung 12: Popup bei Testfall Klick

Weiters wird beim Klick eines Testfalls eine Beschreibung und eine Info bzw. eine Fehlermeldung angezeigt, je nachdem ob der Testfall als optional oder fehlerhaft markiert wurde. Diese Anzeige ist identisch zu dem Pop-Up Fenster bei der Home Seite. Dadurch gibt es verschiedene Möglichkeiten Tests durchzuschauen, zu analysieren und Fehlerquellen zu ermitteln

4.4.3. Tests durchführen

Die „Test durchführen“ Seite wird von den meisten Benutzer verwendet. Sie beinhalten eine Übersicht über die verfügbaren Tests, die schnell und einfach abgearbeitet werden können.

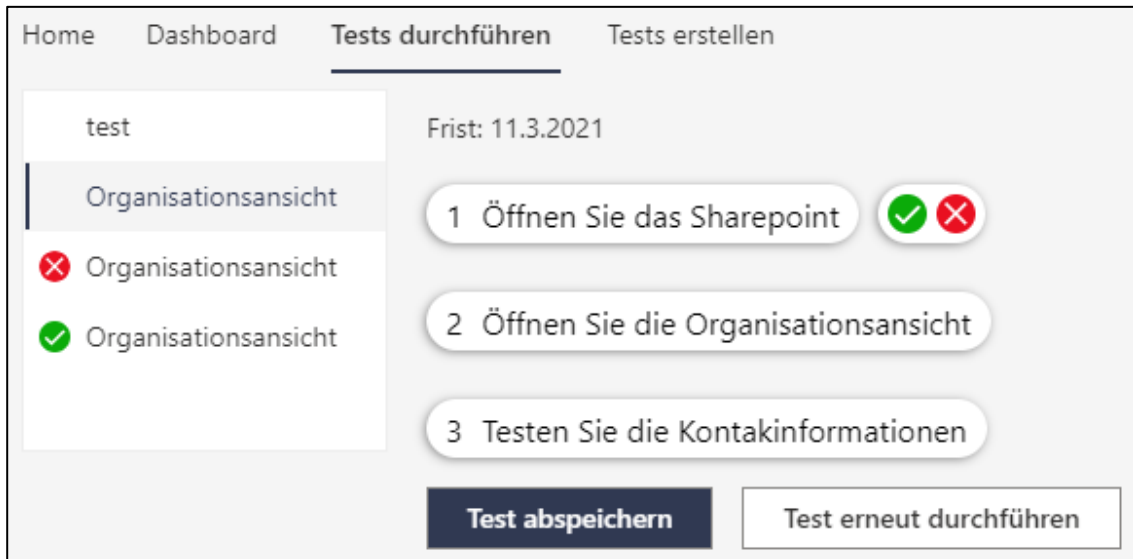


Abbildung 13: Tests durchführen Seite

Auf der linken Seite befinden sich alle verfügbaren Tests. Dieser Teil wurde mit den Nav Element der Fluent UI Bibliothek [7] umgesetzt.

```
<Nav
  className={styles.Nav}
  groups={navLinkGroups} // die Navigations Links
  selectedKey={id}
/>
```

Codesnippet 11: Nav Element für Tests

Die Navigation Links bestehen aus einem Titel und einen Link.

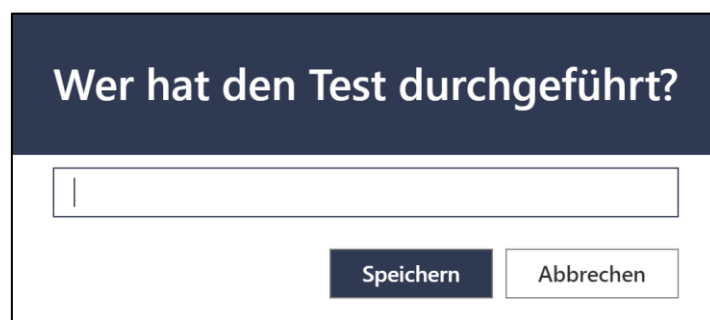
Der einzelnen Tests sind nach dem Erstell Datum sortiert, sodass die neuesten Tests ganz oben sind. Bei bereits durchgeführten Tests wird vor dem Titel mittels Symbols angezeigt, ob dieser erfolgreich oder fehlerhaft war.

Beim Klick eines der Tests wird der Inhalt rechts von der Navigation angepasst. Oben wird die Frist des Tests angezeigt, die bei der Test-Erstellung festgelegt wurde. Darunter werden die einzelnen Testfälle angezeigt. Beim Klick auf den Titel erscheint ein Popup, in dem die nähere Beschreibung aufgelistet wird. Falls der User einen Test durchführt, kann dieser entweder auf das rechts liegende

erfolgreich oder fehlerhaft Icon klicken. Weiters, wenn ein Testfall optional ist, was bei der Erstellung festgelegt wird, erscheint neben den beiden Symbolen ein lila Fragezeichen, mit dem man den Test als optional markieren kann. Beim Markieren von fehlerhaft oder optional, erscheint ein Popup, in das man einen kurzen Kommentar angeben kann, wieso der Test nicht durchgeführt wurde oder was der Grund vom Fehlschlagen ist.

Um den Status des durchgeführten Tests zu speichern wird der unten liegende „Test abspeichern“ Button verwendet. Somit kann der Test für späteres Weiterarbeiten zwischengespeichert werden. Falls der Test fertiggestellt wurde, wird bei der Navigation ein Symbol angezeigt, ob dieser erfolgreich oder fehlerhaft ist. Wenn der Test fertig ist, wird außerdem vom Benutzer erwartet seinen Namen über ein Pop-Up Fenster einzugeben, um festzulegen, welcher Tester den Test durchgeführt hat.

Wenn ein Test nochmal durchgeführt werden soll, wird der Button „Test erneut durchführen“ verwendet. Beim Klick erscheint ein Pop-Up Fenster, in dem eine



The image shows a modal dialog box with a dark blue header containing the text "Wer hat den Test durchgeführt?". Below the header is a white text input field. At the bottom right of the dialog are two buttons: a dark blue button labeled "Speichern" and a white button with a dark blue border labeled "Abbrechen".

Abbildung 14: Popup nach Testdurchführung

neue Frist -für den Test erwartet wird. Nach der Eingabe kopiert sich der ausgewählte Test und er kann erneut durchgeführt werden.

4.4.4. Tests erstellen

Das Tests erstellen Tab wird verwendet, um Tests einfach und schnell zu erstellen. Der Name vom Test ist ein einfaches Textfeld und ist der Text, der in der gesamten App für sämtliche Graphen und Navigation-Links verwendet wird. Die Frist ist ein Datumsfeld, wo man die Deadline für den Test festlegt, dies ebenfalls beim Durchführen der Tests angezeigt wird.

Das nächste Element ist eine ListView aus den PnP SPFx Controls [8]. Sie wird verwendet, um die Testfälle zu bearbeiten und einen schnellen Überblick zu haben.

The screenshot shows the 'Tests erstellen' (Create Tests) page. At the top, there is a navigation bar with links: Home, Dashboard, Tests durchführen, and Tests erstellen (which is underlined). Below the navigation bar, there are several input fields and buttons. The first field is 'Test-Name' with a text input containing 'Test'. Below it is the 'Frist' (Deadline) field with a text input containing 'Datum auswählen' and a calendar icon. Underneath the deadline field is a list of test cases. The first item is 'Titel' (Title). Below it are two items: 'Öffnen Sie das Sharepoint' and 'Testen Sie den Login', each with a pencil icon to its right. At the bottom of the form, there are four buttons: 'Testfall hinzufügen' (Add Test Case), 'Testfall entfernen' (Remove Test Case), 'Testfälle neu anordnen' (Reorder Test Cases), and 'Test abspeichern' (Save Test).

Abbildung 15: Tests erstellen Seite

Mit den ersten beiden der vier unten liegenden Buttons, können Testfälle hinzugefügt oder entfernt werden.

In der Liste der Testfälle können, mit dem Bearbeiten Icon neben dem jeweiligen Testfall, die Daten bearbeitet werden. Es erscheint ein Popup, in dem der Titel, die Beschreibung und ob ein Testfall optional ist, eingetragen werden kann.

The screenshot shows the 'Testfall bearbeiten' (Edit Test Case) popup. It has a title bar with 'Testfall bearbeiten' and a close button (X). Inside the popup, there are three sections: 'Erforderlich' (Required) with a checked checkbox, 'Titel' (Title) with a text input containing 'Öffnen Sie das Sharep...', and 'Beschreibung' (Description) with a text area containing 'Öffnen Sie das SharePoint der Organisation und Loggen Sie sich ein.'

Abbildung 16: Testfall bearbeiten Popup

5. Zusammenfassung [DB]

SmartQS ist ein Tool für manuelles Software-Testing. Es kann einfach in Microsoft Teams integriert werden und passt sich von der Farbgestaltung an die Farbe des firmeninternen SharePoints an. Das Backend kann leicht in der Azure Cloud gehostet werden und hat somit über das Azure Active Directory einen integrierten Zugriffsschutz. Das optionale Ziel, beim Abschließen eines Tests eine „Card“ zu verschicken ist im Backend bereits umgesetzt, konnte jedoch im Frontend noch nicht realisiert werden. Das Projekt wird ab Juni 2021 in der Firma smartpoint Consulting GmbH produktiv eingesetzt.

6. Literaturverzeichnis

- [1] T. Kok, „Test Case, Test Suite, Test Run, What’s the Difference?“, 31 Juni 2017. [Online]. Available: <https://www.testmonitor.com/blog/test-case-test-suite-test-run-whats-the-difference>. [Zugriff am 04 August 2021].
- [2] OpenJS Foundation, „Über Node.js | Node.js“, 11 März 2021. [Online]. Available: <https://nodejs.org/de/about/>. [Zugriff am 11 März 2021].
- [3] Microsoft, „TypeScript: Typed JavaScript at Any Scale“, 11 März 2021. [Online]. Available: <https://www.typescriptlang.org/>. [Zugriff am 20 Februar 2021].
- [4] MongoDB, Inc., „Die beliebteste Datenbank für moderne Apps | MongoDB“, 11 März 2021. [Online]. Available: <https://www.mongodb.com/de>. [Zugriff am 20 Februar 2021].
- [5] OpenJS Foundation, „Express - Node.js-Framework von Webanwendungen“, 11 März 2021. [Online]. Available: <https://expressjs.com/de/>. [Zugriff am 11 März 2021].
- [6] Microsoft, „Create your first Microsoft Teams app using the Yeoman generator“, 05 02 2021. [Online]. Available: <https://docs.microsoft.com/en-us/microsoftteams/platform/tutorials/get-started-yeoman>.
- [7] Microsoft, „Microsoft Fluent UI“, 11 03 2021. [Online]. Available: <https://developer.microsoft.com/de-de/fluentui?fabricVer=6#/controls/web>.
- [8] Microsoft 365 Community, „Reusable React controls for your SharePoint Framework solutions“, 11 03 2021. [Online]. Available: <https://pnp.github.io/sp-dev-fx-controls-react/>.
- [9] S. Chacon und J. Long, „Git“, 29 März 2021. [Online]. Available: <https://git-scm.com/>. [Zugriff am 29 März 2021].
- [10] Smartbear, „API Documentation & Design Tools for Teams | Swagger“, 11 März 2021. [Online]. Available: <https://swagger.io/>. [Zugriff am 20 Februar 2021].

7. Anhang

7.1. Arbeitsprotokolle

7.1.1. Lukas Werner

DATUM	AUFGABE	DAUER
01.07 - 03.07	Kennerlernen und Testen der Technologien bei einem Seminar	15h
22.07	Zusammenfassung Agile Softwareentwicklung	4h
29.07	Zusammenfassung Grundlegende Prinzipien, Praktiken und Prozesse des agilen Testens	2h
29.07	Research Test Plans and Test Runs	1h
30.07	Zusammenfassung von Grundlegende Prinzipien, Praktiken und Prozesse des agilen Testens	2h
03.08	aufsetzen Sandbox Entwicklungsumgebung	3h
03.08	aufsetzen Sandbox Entwicklungsumgebung; schreiben von Dokumentation: setting up project	2h
03.08	Zusammenfassen von Test Techniken	3h
03.08	Formattierung von allen Markdown Dokumenten (Inhaltsverzeichnis, Auslagerung von Themen)	1h
04.08	Zusammenfassung von Test Methoden	3h
18.09	Mockup erstellen mit Adobe XD	4h
25.09	Pflichtenheft erstellen	1h
25.09	Mockup testen und rendering ausprobieren in sanbox projekt	4h
01.10	Rendering dynamisch erweitern; svgs statt pngs für hover effekt	3h
03.10	Migration von Sandbox Projekt auf SmartQS Projekt; Markieren von Erfolgreichen und Fehlerhaften Tests nun möglich	2h
09.10	Sichbarkeit von Status Buttons hinzugefügt; State erweitert; Rendering ausgelagert; Modal Interfaces hinzugefügt für wiederverwendbarkeit	5h
10.10	State Handhabung aktualisiert; callback Methode fürs State Handhabung hinzugefügt; rendering angepasst für logischen Aufbau; lokale Speicherung von Test Plans auf Array umgeschrieben	4h
14.10	Callback Function aktualisiert damit sie state nicht direkt ersetzt; State Handhabung aktualisiert und funktioniert nun korrekt	3h

15.10	svgs als React Component hinzugefügt; Element Rendering verbessert mithilfe von ausgelagerten Funktionen	2h
15.10	updated the User stories that define the project	1h
16.10	fixed project folder structure; fixed wrong terms for some words; fixed spelling mistakes	2h
18.10	added property active to display status buttons only if the test case is currently active; added functions in props interface	2h
20.10	added comments to all functions	1h
20.10	added Modal that shows the test case description when clicking on test case	1h
03.11	added message property in test case; added dialog when marking test case as failure to enter the issue; updated the style of the modal to fluent ui style	3h
20.11	added Richtext to Dialog; added the Ability to switch between test Runs	2h
5.12	added React Router to Navigate between the TestRunOverview and the dashboard;	3h
11.12	added React Router and NavBar to TestRunOverview to navigate between different TestRuns; navbar dynamically renders links from data; added TestRunOverview to list all available TestRun	7h
16.12	moved Modeldata; changed requests to channelid; added icon next to test run nav to display if a testrun is successfull or not	5h
20.12	added optional field to the test runs forms; added button to reorder all testcases (custom listview)	7h
04.01	outsourced nav component; added dashboard component; worked on pnpjs charts for drill down chart	6h
15.01	pnpjs charts did not work; used chartsjs to implement drill down chart successfully	4h
29.01	added date field for deadline; sortierung der testruns nach Erstelldatum; optionale tests beim Durchführen implementiert	5h
03.02	implementiert das nach fehler beim testcase automatisch abbrechen; button zum testrun erneut durchführen hinzugefügt, kopiert und setzt testrun zurück	3h
09.02	added functionallity to mark testcases as optional with comment popup; added tester field and popup when finishing a test; updated drill down chart colors to enable optional	3h
10.02	added doneOn property to models; doneOn property gets set when a testrun if either faulty or successful; add popup when copying testrun to set new deadline	1h

11.02	added readonly flag to set if the test runs should be read only or not; customized testrunoverview, testrun and testcase for generic use	3h
11.02	added statistics model; outsourced drill down component; added readonly testrunoverview to dashboard and fixed routing; added chart to dashboard page that displays the statistics of all test runs	4h
12.02	added second Graph to Dashboard that displays all statistics of test cases;renamed Statistics Model to TestRun; added Model for TestCaseStatistics	2h
13.02	added 2 datepicker to select the start and end date to filter the graphs and the testrun navbar; added reset button to reset the datepicker; added the ability to filter the statistics of testcases and testruns by date range	3h
13.02	added ability to filter the testrun navbar by start and enddate; added props to parse the date values	2h
16.02	added feature to configure server url of the backend; done via property pane; outsourced drilldown detailed chart to drilldown component; deleted unused code and comments	2h
17.02	added Code Documentation to all methods; added region splitting to all files for better code readability; formatted some code parts	1h
19.02	tried aligning statistics graphs side by side	3h
20.02	fixed issue with multiple charts by adding a redraw flag; moved testrunoverview readonly component to home page	2h
21.02	changed colors of drilldown chart to the same as statistics chart; tried fixing bug where a random margin at the top gets added when rendering drill down chart;added props to customize title of test run nav; added feature when test run overview is in readonly mode only completed test runs show	1h
22.02	updated pane properties and dashboard handling; added list view for creating test cases but without implementation of dynamic data	2h
23.02	fixed bug where initial value of toggle was undefined and not true	1h
25.02	changed local serving to real sharepoint site; removed server url pane property; added feature that receives the backend server url from a sharpoint tenant property; added service that allows to receive a specific tenant property	3,5h
28.02	added alert when tenant property url is empty; added listview that displays the test cases in the testrun forms; added button where to edit a single test case; modal appears with a edit mask where the test case can be edited; deleted regular rendering of test cases	3h

28.02	added Frontend Documentation part to document;added theory part of frontend that describes what react is and how to create a teams tab project; added project structure of frontend that describes the structure how the project works	1,5h
07.03	changed title of test overview; deleted necessary code; added comments	1h
11.03	added Structure of Navigation to frontend documentation	2h
15.03	added Design Libraries and Home Tab to documentation	3h
17.03	added implementation of "Dashboard", "Tests durchführen", "Tests erstellen" Pages	4h
18.03	fixed Drilldown chart not showing in ms teams; fixed only showing completed test runs	2h
19.03	added generating and integration of Project to documentation	3h
20.03	added Service that sets up custom http Client where azure authentication works; changed all fetch requests to custom http client; refactoring of some code	4h
24.03	added Codesnippets, extended theory of testing	2h
25.03	deleted unused listview component; fixed bug where request would not work in dashboard; added modal do testrun forms if saving request failed or succeeded; added clearing of form after saving	2h
26.03	fixed slider at property pane; changed solution name and serving location	1h
29.03	fixed formatting of documentation; fixed spelling mistakes	1h
30.03	rewrote and added work logs to documentation	1h
31.03	fixed spelling mistakes and wording; reformatted some parts	2h

7.1.2. Birngruber Dominik

DATUM	AUFGABE	DAUER
1.7 - 3.7	Kennenlernen und testen der Technologien in einem Seminar	15h
29.7	Erstellen des GitHub Repositories und einrichten der Ordnerstruktur, Zusammenfassen von Rest Plans und Test Runs	1,5h
3.8	Beschreiben von Testarten und Strategien	2h
2.10	Erstellung des Pflichtenheftes	4h
9.10	Erarbeiten eines CosmosDB Tutorials/React Tutorials	4h

15.10	Weiterarbeiten am CosmosDB Tutorial	2h
2.11 - 8.11	CosmosDB Tutorial angepasst mit eigenen Daten	4h
9.11 - 15.11	Backend weiter ausgebaut	4h
23.12	Design für Präsentation erstellt	2h
2.1	Datenmodell erweitert und APIs entsprechend angepasst	2h
4.1	Dokumentation für dox42 gelesen	1h
5.1-12.1	Mehrere spezialisierte APIs hinzugefügt, Backend vollständig in TypeScript übersetzt und Build-Prozess erstellt	15h
13.1 - 19.1	SwaggerUI für API-Dokumentation recherchiert und Dokumentation begonnen, API-Deployment für Azure recherchiert	12h
20.1 - 3.2	API-Dokumentation fortgeführt, Möglichkeit für Optionale Tests hinzugefügt, Punkte für Präsentation genauer spezifiziert	12h
4.2 - 11.2	APIs für Erhalt von Statistiken erstellt, Dokumentation weitergeführt	15h
12.2 - 17.2	Endpoints der Verschiedenen APIs pro Channel mit allgemeinen Requests zusammengeführt, Möglichkeit für Filtern nach Datum hinzugefügt, Dokumentation angepasst	20h
18.2 - 24.2	Code besser dokumentiert, Möglichkeit zum Senden von Teams-Nachrichten gesucht und erste Umsetzung	18h
25.2 - 3.3	Umsetzung für Senden von Teams-Nachrichten finalisiert und dokumentiert, Anpassungen für Deployment vorgenommen	7h
4.3 - 10.3	DA-Dokumentation ausgearbeitet und Backend deployed	10h
11.3 - 17.3	DA-Dokumentation ausgearbeitet (Theorie)	10h
18.3 - 24.3	DA-Dokumentation ausgearbeitet (Umsetzung)	10h
25.3 - 31.3	DA-Dokumentation finalisiert	10h

7.2. API-Dokumentation

Diese Dokumentation wurde als Website aufgerufen und anschließend als PDF exportiert.