

|  |
| --- |
| Höhere Technische Bundeslehranstalt  Höhere Lehranstalt für Informationstechnologie |

*HTL Diplomarbeit*

|  |  |  |
| --- | --- | --- |
| *SmartQS* | | |
| ausgeführt im | | |
| *Schuljahr 2020/21* | | |
| eingereicht von | | |
| Birngruber Dominik | | 5AHIT |
| Werner Lukas | | 5AHIT |
| Hofer Susanne |  | |
|  |  | |

# 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 „Ingenieursbüroarbeit“.

# 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 Juni 20xx

|  |  |
| --- | --- |
|  |  |
|  | *Name* |
|  | *Name* |

# 

# Danksagung

Wir möchten uns an dieser Stelle herzlich bei der Firma smartpoint IT consulting GmbH bedanken, welche mir 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.

Besonders möchte ich mich auch bei meiner Familie bedanken, welche mich in der stressigen Zeit der Diplomarbeit unterstützt hat.

# Einleitung

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 Azue DevOps und ähnlichen Programmen zu entwickeln, mit dem Bonus, dass unsere Lösung leicht in Microsoft Teams eingebunden werden kann, da Teams oft als Firmeninterne **K**ommunikationsplattform verwendet wird.

# Abstract

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, with the bonus of being integrated into Microsoft Teams, a product often used for communication in companies.

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Bearbeiter | Änderungen | Datum |
| 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 |

Inhalt

[I. Haftungsausschluss 2](#_Toc66280555)

[II. Eidesstattliche Erklärung 2](#_Toc66280556)

[III. Danksagung 3](#_Toc66280557)

[IV. Einleitung 3](#_Toc66280558)

[V. Abstract 3](#_Toc66280559)

[1. Ist-Situation 7](#_Toc66280560)

[2. Zielbestimmungen 7](#_Toc66280561)

[3. Theoretisch Grundlagen 7](#_Toc66280562)

[3.1. Software Testing 7](#_Toc66280563)

[3.1.1. Durchführen von Tests 7](#_Toc66280564)

[3.2. Backend 8](#_Toc66280565)

[3.2.1. Datenbank – MongoDB 8](#_Toc66280566)

[3.2.2. Express (OpenJS Foundation, kein Datum) 8](#_Toc66280567)

[3.2.3. NoSQL vs. SQL 8](#_Toc66280568)

[3.3. Frontend 10](#_Toc66280569)

[3.3.1. React 10](#_Toc66280570)

[(Microsoft, kein Datum) 10](#_Toc66280571)

[3.4. Versionsverwaltung 10](#_Toc66280572)

[4. Implementierung Backend 11](#_Toc66280573)

[4.1. Programmiersprache 11](#_Toc66280574)

[4.2. Projektstruktur 11](#_Toc66280575)

[4.3. Datenbankstruktur 12](#_Toc66280576)

[4.4. Deployment auf Azure 13](#_Toc66280577)

[4.5. Dokumentation des Backends 13](#_Toc66280578)

[4.6. APIs 14](#_Toc66280579)

[4.7. Datenbankaufrufe mit Mongoose 15](#_Toc66280580)

[4.8. Alternative Lösungswege 16](#_Toc66280581)

[5. Implementierung Frontend 16](#_Toc66280582)

[5.1. Projektstruktur 16](#_Toc66280583)

[6. Zusammenfassung 17](#_Toc66280584)

[7. Literaturverzeichnis 17](#_Toc66280585)

[8. Anhang 18](#_Toc66280586)

# Ist-Situation

Zurzeit wird im Unternehmen zum manuellen Software-Testing noch kein Tool verwendet. Um das Software-Testing zu erleichtern, soll deshalb ein Tool entwickelt werden, welches direkt über Microsoft Teams verwendet werden kann.

# Zielbestimmungen

Eine App für Microsoft Teams in der manuelle Software-Tests definiert werden können. Diese sollten anschließend durchgeführt werden können und e soll automatisch eine Dokumentation erstellt werden und auf SharePoint gespeichert werden. Danach soll eine „Card“ in Teams versendet werden. Die Daten werden in Azure SQL gespeichert.

# Theoretisch Grundlagen

## Software Testing

## Durchführen von Tests

Manuelles testen von Hardware oder Software wird meist mit verschiedenen Testdokumenten beschrieben, genauer gesagt Test Plans, Test Cases und Test Runs. In dieser Arbeit werden Test Cases behandelt, welche die Durchführung von Tests beschreiben. 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 oder Bilder angefügt werden. Außerdem wird aufgezeichnet, wer den Test durchgeführt hat und wann der Test durchgeführt wurde.

## Backend

## Datenbank – MongoDB

(MongoDB, Inc., kein Datum)  
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, da bedeutet jedes Objekt speichert nur Felder, welche auch benötigt werden. MongoDB bietet jedoch auch einige Vorteile von relationalen Datenbanken, wie ACID Transaktionen und, Joins in Queries und nicht nur eingebettete Unterobjekte, sondern auch Verweise auf andere Objekte.

## Express (OpenJS Foundation, kein Datum)

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 API macht.

## NoSQL vs. SQL

(MongoDB, Inc., kein Datum)

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 NoSQL Datenbanken Daten, je nach Art der Datenbank, in JSON-Dokumenten, Schlüssel-Wert Paaren, Tabellen, welche die Spalten dynamisch anpassen, oder in Nodes gespeichert. NoSQL Datenbanken sind auch moderner, da sie in den 200er 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 Schlüssel-Wert-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. Auch die Speichernutzung ist meist bei SQL-Datenbanken besser, da besonders Dokumentbasierte NoSQL-Datenbanken auf Queries spezialisiert sind und somit manche Daten doppelt abgespeichert werden.

## Frontend

## React

## (Microsoft, kein Datum)

React ist ein JavaScript Framework, welches zum Erstellen von Performante Web Anwendungen verwendet wird. Das Framework ist sehr einfach zu konfigurieren, deshalb gibt es einen fertigen Yeoman Generator, um einfach ein Tab für Microsoft Teams zu erstellen. Dieser Generator setzt ein fertiges Projekt auf, sodass nach einen Projekt build die Lösung direkt in Microsoft Teams eingebunden werden kann.

## Versionsverwaltung

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.

# Implementierung Backend

## Programmiersprache

Als Programmiersprache wurde auf TypeScript gesetzt, ein Superscript von JavaScript entwickelt von Microsoft. TypeScript muss zur Ausführung in JavaScript transpiliert werden, bietet jedoch im Gegensatz zu JavaScript Typisierung und leichte Verwendung von Klassen.

## Projektstruktur

//Router von Express

router.get("/testDefinitions", function (req: any, res: any) {

  testDefinitionService.get(req, res);

});

//Hilfsfunktion mit Datenbankaufrug

/\*\*

 \* Returns all Test Definitions from the Database

 \* @author Dominik Birngruber

 \* @param req

 \* @param res

 \*/

function get(req: any, res: any) {

  //Datenbankaufruf

}

Aufbau von einer API

Das Backend wird als NodeJS 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. Über der Funktion für Express befindet sich auf die Dokumentation für die API (siehe 4.5. Dokumentation des Backends). Die Funktionen, welche die Verbindung mit der Datenbank herstellen sind auf 3 Dateien aufgeteilt, welche jeweils Funktionen für eine Art von Abfrage beinhaltet. Einige Variablen wie der Datenbank-Verbindungsstring werden in einer eigenen Datei gespeichert, welche nicht in die Versionsverwaltung aufgenommen wird, um unberechtigten Zugriff zu verhindern.

## Datenbankstruktur

[

  {

    "\_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

      }

    ]

  }

]

Datenmodell

Die Datenbank hat nur eine Tabelle, welche die 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 automatisch von MongoDbB automatisch erstellt, wobei \_id eine eindeutige Zeichenkette ist und \_\_v angibt, welche Version des Objekts das gespeicherte Objekt ist. Da direkt Objekte gespeichert werden, werden die Test Cases direkt bei der Testdefinition gespeichert

## Deployment auf Azure

Das gesamte Backend wird auf Azure gehostet. Es wurde Auf Azure gesetzt, da das fertige Produkt über SharePoint benutzt wird und es möglich ist, Zugriffe auf die API über das ActiveDirectory 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, ein package.json und das environment-file enthalten sind. Dieser wird anschließend hochgeladen und auf dem Server ausgeführt.

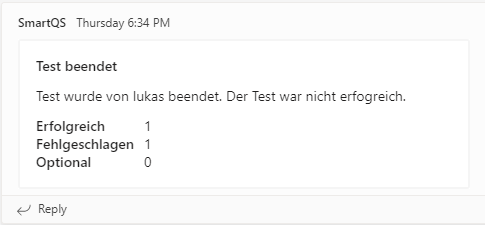
## Dokumentation des Backends

Als API Dokumentationstool wurde auf Swagger von Smartbear gesetzt. 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 auch gleich die APIs getestet werden können. In dieser Arbeit wurde die Open Source Implementierung namens Swagger UI verwendet.

Die Funktionen, welche auf die Datenbank zugreifen wurden mit JSDoc dokumentiert. Dabei wird genau beschrieben, was die Anfragen beinhalten müssen und ob es im body oder in der URL mitgegeben werden muss.

## 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 Test Definitionen, minimale Testdefinitionen und Erfolgsstatistiken. Für Testdefinitionen, minimale Testdefinitionen und den zwei Statistik APIs ist es auch möglich, sie nach Durchführungszeit einzugrenzen.

Die APIs, welche Test Definitionen liefern, geben die gesamten Daten zurück (siehe Abbildung bei Datenbankstruktur). Die APIs für die minimalen Testdefinitionen geben ähnliche Daten zurück, jedoch werden bei diesen die Test Cases weggelassen. Die Statistik-APIs geben im Falle von Testdefinitions-Statistiken die Anzahl der erfolgreichen, fehlgeschlagenen und noch nicht durchgeführten Tests zurück und im Fall von Test Case-Statistiken die Anzahl von erfolgreichen, fehlgeschlagenen, optionalen und nicht durchgeführten ****Tests.

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 fertiggestellt markiert wird, wird eine adaptive Card an diesen Webhook gesendet.

## 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 wird, da das Projekt in TypeScript programmiert wurde, muss ebenfalls 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 |
| const testDefinitionSchema =  new Schema({    name: String,    createdOn: String,    testCases: [testCaseSchema],    tester: String,    finished: Boolean,    deadline: String,    doneOn: String,    channelID: String,  }); | export interface TestDefinition  extends Document {    name: String;    createdOn: String;    testCases: [TestCase];    tester: String;    finished: Boolean;    deadline: String;    doneOn: String;    channelID: String;  } |
| export default model<TestDefinition>("TestDefintion",  testDefinitionSchema); | |

Ein Aufruf kann mit diesem model erfolgen, indem man die verschiedenen Hilfsfunktionen verwendet, man kann zum Beispiel 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);

});

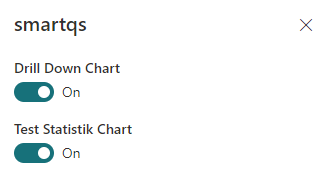
## Alternative Lösungswege

Datenbank

Die Entscheidung für eine dokumentbasierte Datenbank wurde anhand des Modells getroffen. Dadurch, dass jede Testdefinition mehrere Testfälle hat, wäre 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.

# Implementierung Frontend

## Projektstruktur

Das Frontend wird als NodeJS Server realisiert, welcher die Grundstruktur für dir Benutzeroberfläche beschreibt. Nach einem Projekt 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 bearbeiteten 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 abberufen. Diese sollte als Tenant Property mit den Namen „smartqsserviceurl“ abgespeichert werden.

Nach diesen Initialisierungsschritten kommt nun die allgemeine Navigationsleiste, die aufgerufen wird. Diese ist in `SmartQSNav.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 enthalten. Dort werden sie unteranderem von anderen Seiten aufgerufen, die diese Komponenten dann benötigen.

# Zusammenfassung

# Literaturverzeichnis

Microsoft. (2012). *TypeScript: Typed JavaScript at Any Scale*. Abgerufen am 20. 02 2021 von https://www.typescriptlang.org/

Microsoft. (kein Datum). *Create your first Microsoft Teams app using the Yeoman generator*. Von https://docs.microsoft.com/: https://docs.microsoft.com/en-us/microsoftteams/platform/tutorials/get-started-yeoman abgerufen

MongoDB, Inc. (kein Datum). *Die beliebteste Datenbank für moderne Apps | MongoDB*. Von https://www.mongodb.com/de abgerufen

OpenJS Foundation. (kein Datum). *Express - Node.js-Framework von Webanwendungen*. Von https://expressjs.com/de/ abgerufen

Smartbear. (kein Datum). *API Documentation & Design Tools for Teams | Swagger*. Von https://swagger.io/ abgerufen

# Anhang