

27.03.2020

# IPERKA M151

Projektdokumentation zum Modul  
151 «Datenbanken in Webauftritt  
einbinden»

RaviAnand Mohabir  
BBBADEN

# Inhaltsverzeichnis

1	Informieren .....	3
1.1	Einleitung .....	3
1.2	Mögliche Erweiterungen.....	3
1.3	Anforderungsanalyse .....	4
1.4	Ideen .....	5
1.5	Technologien.....	5
1.5.1	Web-Frameworks.....	5
2	Planen .....	6
2.1	Systemgrenze .....	6
2.2	Use-Case.....	6
2.3	Klassendiagramm .....	6
2.4	Mock-Up (GUI) .....	7
2.4.1	Quiz-Seite .....	7
2.4.2	Admininterface .....	8
2.5	Storyboard .....	9
2.5.1	Quiz-Seite .....	9
2.5.2	Admininterface .....	9
2.6	Konzeptionelles Datenmodell (Datenbank) .....	10
2.7	Logisches Datenmodell (Datenbank) .....	11
2.8	Testfallspezifikationen .....	12
3	Entscheiden.....	14
3.1	Entscheidungsmatrix.....	14
3.1.1	Web-Framework & Back-End .....	14
3.2	Entscheidungen.....	14
3.2.1	Dynamische Elemente der Anwendung.....	14
3.2.2	Dataserver .....	14
4	Realisieren.....	15
4.1	Programm .....	15
4.1.1	Ideenumsetzung.....	15
4.2	GUI .....	15
5	Kontrollieren .....	16
5.1	Testprotokoll.....	16
5.2	Funktionstest (Unit-Test) .....	17
5.3	Integrationstest (Selenium) .....	17
5.4	Testfazit.....	17
6	Auswerten .....	18

6.1	Reflexion .....	18
7	Anhang .....	19
7.1	Quellen .....	19
7.2	Code .....	19

# 1 Informieren

## 1.1 Einleitung

Ziel dieses Projekts ist es, eine Web-Applikation zu implementieren und diese mit einer Datenbank zu verknüpfen. Die Applikation muss in die 4 N-Tiers getrennt werden: Presentation, Webserver, Application Server, Dataserver.

Die Datenbank muss mit Hilfe einer Schnittstelle an die Applikation gebunden werden, welche Daten einlesen kann und alle CRUD-Operationen abdeckt. Sowie ein ORM Framework welches die eingelesenen Daten in Objekte mappt.

Wie gewohnt muss die Applikation client-seitig sowie im back-end abgesichert werden und alle Interaktionen mit der Datenbank sollen mit Prepared Statements durchgeführt werden damit Angriffsmöglichkeiten auf diese durch den Benutzer vermindert werden.

Spezifisch muss hier das Spiel «Wer Wird Millionär?» implementiert werden. Benutzer können Fragen mit einer Auswahl von 4 Antworten beantworten. Dazu müssen Systemadministratoren mit einem Admininterface Fragen hinzufügen bzw. entfernen können, sowie Kategorien verwalten und Einträge aus der Highscoreliste löschen.

## 1.2 Mögliche Erweiterungen

- Benutzer können ein Konto anlegen, mit welchem sie das Spiel starten. Somit gehört der Benutzername einer Person, welche mehrmals in der Highscoreliste erscheinen kann.
- Jede Spielstatistik wird einem Benutzer zugewiesen. Neben den Grundstatistiken werden alle gestellten Fragen, ob der Joker eingesetzt wurde, sowie die ausgewählte Antwort und dafür gebrauchte Zeit gespeichert.
- Benutzer können diese Spielstatistiken einsehen und sehen auch bei falsch beantworteten Fragen die richtige Antwort.
- Der Benutzer hat zur Beantwortung einer Frage 2 Minuten Zeit, geht er über die Zeit hinaus, wird das Spiel beendet und keine Punkte zugeschrieben -> Spiel ist verloren.
- Der gewonnene Betrag des Spiels wird dem Benutzerkonto zugewiesen, somit sehen Benutzer wie viel sie bis jetzt gewonnen haben.

### 1.3 Anforderungsanalyse

#### Anforderungsanalyse: Wer Wird Millionär?

Anforderungsnr.	Anforderung	Beschreibung
<b>1</b>	Grundanforderungen	
<b>1.1</b>	Tiers	Die verschiedenen Komponenten des Systems werden voneinander und der Datenbank getrennt. Daten vom Presentation-Layer sind nicht vertraulich, die Business-Logik wird im Application Server programmiert.
<b>1.2</b>	Persistenz	Quizfragen und Antworten, sowie Benutzerstatistiken werden in einer Datenbank hinterlegt und vom Back-End mit Hilfe eines Database-Connectors und einem ORM Framework eingelesen und an das Front-End weitergegeben.
<b>1.3</b>	Quiz	Der Benutzer kann seinen Namen eingeben und das Quiz starten. Fragen werden angezeigt und dazu 4 Antworten wovon nur eine richtig ist. Der Benutzer kann am Anfang die Kategorie, von welcher die Fragen gestellt werden, auswählen.
<b>1.3.1</b>	Zeitmessung	Während dem Quiz wird die vergangene Zeit seit dem Start gemessen und dem Benutzer angezeigt, um die Recherchemöglichkeiten einzuschränken.
<b>1.3.2</b>	50:50 Joker	Der Benutzer kann einmal den 50:50 Joker einsetzen welches zwei von den drei Falschen Antworten ausblendet und das Beantworten der Frage vereinfacht.
<b>1.3.3</b>	Fragen beantworten	Für richtig beantwortete Fragen erhält der Benutzer 30 Stück der Geldwährung und kann die nächste Frage spielen, bzw. das Spiel dann beenden. Bei falsch beantworteten Fragen geht das ganze Geld verloren und das Spiel wird beendet.
<b>1.4</b>	Highscoreliste	Benutzer können die Statistiken anderer Spieler anzeigen lassen und vergleichen; Der Rang in der Highscoreliste, die gewichteten Punkte, der Name des Spielers, der Zeitpunkt des Spiels, die Anzahl errungenen Punkte, die Dauer des Quiz und die gewählten Kategorien sollen alle ersichtlich sein.
<b>1.5</b>	Admininterface	
<b>1.5.1</b>	Quiz verwalten	Über das Admininterface können Quizfragen hinzugefügt und verwaltet werden. Kategorien können verändert werden.
<b>1.5.2</b>	Highscoreliste verwalten	Administratoren können einzelne Einträge von der Highscoreliste löschen.
<b>2</b>	Erweiterte Anforderungen	
<b>2.1</b>	Benutzerkonten	Benutzer können Konten anlegen und das Quiz mit diesem starten. Eine Übersicht des Profils kann eingesehen werden und Spielstatistiken zu allen gespielten Spielen werden gespeichert. Jeder Benutzer hat ein «Bankkonto», welches mit dem gewonnen Geld pro Spiel gefüllt wird.
<b>2.2</b>	Spielstatistiken	Spiele speichern neben dem Startzeitpunkt und den Kategorien auch jede einzelne Spielrunde welche die gestellte Frage, die ausgewählte Antwort (falls vorhanden), ob der Joker benutzt wurde und die Dauer kennen. Spieler können diese Statistiken über ihr Profil einsehen und diese Statistiken werden zur Kalkulation der Punktzahl eingesetzt. In der Statistik werden bei falsch beantworteten Fragen die richtige separat Antwort angezeigt.
<b>2.3</b>	Antwort-Countdown	Zur Beantwortung jeder Frage gibt es ein Zeitlimit von 2 Minuten. Geht man über die Zeit hinaus, wird das Spiel beendet und das Spiel ist verloren.

## 1.4 Ideen

## 1.5 Technologien

Um eine gute Entscheidung bei der Auswahl der Technologien für die verschiedenen Komponenten des Programms zu wählen, habe ich mir einige Möglichkeiten angeschaut und kurz zusammengefasst, um die Vor- und Nachteile übersichtlich aufzuzeigen.

### 1.5.1 Web-Frameworks

#### 1.5.1.1 *JavaServer Faces (JSF)*

JavaServer Faces ist eine Web-Technologie, welche von der Schule empfohlen wurde und es ermöglicht, ein sicheres back-end zu schreiben und die Models davon mit speziellen HTML-Dateien zu verknüpfen um dem Benutzer auf eine sichere Weise Daten anzuzeigen und dann auch Daten zurück an den Server zu senden per HTTP-GET oder -POST. Da es auf der client-Seite die gängigen Web-Technologien benutzt, ist es immer noch möglich moderne Seite damit zu erstellen, jedoch hat es die vielen Nachteile von Java bezüglich der Simplität der Programmierung und Effizienz im Vergleich zu modernen Frameworks.

#### 1.5.1.2 *React mit Node.js back-end*

React ist ein sehr populäres Web front-end Framework mit welchem man in JavaScript bzw. Typescript das GUI der Applikation erstellen kann und da es sich auf sogenannte «Components» verlässt gibt es online sehr viele Bibliotheken, mit welchen man innert kürzester Zeit ein modernes und performantes GUI erstellen kann. Dazu sind React Pages single-page, das heisst es wird auf dem Webserver nie eine neue Seite angefragt, was zu besserer Performance, besonders auf mobilen Geräten führen kann. Ein Nachteil davon ist jedoch die starke Abhängigkeit von JavaScript, was dazu führt, dass ältere Geräte React Pages oft nicht laden können und somit eine Benutzergruppe ausgeschlossen wird.

Da React nur das front-end löst, braucht es ein sicheres back-end welches mit der Datenbank verknüpft ist und dafür sind Programmiersprachen und Frameworks wie Python und Flask, oder in diesem Fall, Node.js sehr bekannt. Node.js ist eine Möglichkeit mit JavaScript back-end APIs zu schreiben. Es wird auf demselben Server wie die React-Applikation zur Verfügung gestellt mit welchem das front-end hin und her kommuniziert und hier werden erneut Eingaben überprüft und Daten eingelesen bzw. ausgegeben welches React dann dem Benutzer anzeigt.

#### 1.5.1.3 *ASP.NET Core MVC mit React*

Eine Alternative zu einem Node.js Server wie bereits in 1.4.1.2 geschildert ist ein ASP.NET Core MVC Server. Das Prinzip bleibt dasselbe wie bei der Kombination von React und Node.js; das front-end ist strikt vom back-end getrennt indem die Kommunikation über eine sichere API läuft, jedoch wird das back-end anstatt mit JavaScript mit C# implementiert und somit die Performance sowie Sicherheit verbessert. Dazu ist es deutlich angenehmer solche komplexe back-ends mit einer kompilierten Sprache zu schreiben im Vergleich zu interpretierten da Fehler bei diesen nicht bei Kompilation gefangen werden können.

## 2 Planen

### 2.1 Systemgrenze

### 2.2 Use-Case

### 2.3 Klassendiagramm

## 2.4 Mock-Up (GUI)

### 2.4.1 Quiz-Seite

Die Quiz-Seite soll möglichst simpel und intuitiv für die Benutzer aufgebaut sein. Da es neben dem Hauptspiel nur ein Auswahlfenster für die Kategorienwahl sowie die Highscoreliste gibt, braucht es kein Menü und das Design des Quiz sollte so stark wie möglich an das Spiel angelehnt sein.

Bei der Kategorienauswahl für das Spiel werden mithilfe von Dropdowns die verschiedenen Kategorien angezeigt und wenn man weitere Kategorien hinzufügen möchte, kann das Pluszeichen angeklickt werden. Der Spieler muss nur noch den eigenen Namen eingeben und ab dann beginnt der Timer welche die vergangene Zeit misst und somit auch das Spiel. Fragen werden hintereinander angezeigt und falls der Spieler noch ein Joker zur Verfügung hat, kann mit einem Button dieser angewandt werden.

Am Ende des Spiels wird die Punktzahl angezeigt sowie die vergangene Zeit, ein Neustart wird auch zur Verfügung gestellt, welcher zurück zur Startseite führt und die Kategorienauswahl anzeigt. Die Highscoreliste zeigt alle Spieler in der Abfolge von den Besten Spielern zu den Schlechtesten, falls es zu viele Spieler für eine Seite hat, kann man die «Seite» wechseln und die nächsten Spieler anschauen.

### Wer wird Millionär?

Kategorien wählen:

Vergangene Zeit: 1m 40s

Geldstand: 0

50:50 Joker

Frage 1

### Spiel beendet!

Geldstand: 60

Vergangene Zeit: 2m 35s

Korrekt beantwortete Fragen: 2

### Leaderboard

Spieler	Korrekt beantwortete Fragen
Spieler 1	8
Spieler 2	6
Spieler 3	5
Spieler 4	4
Spieler 5	2



## 2.4.2 Admininterface

Nach dem Anmelden sehen Administratoren eine Auswahl welches ihnen erlaubt, das Quiz zu bearbeiten oder Einträge aus der Highscoreliste zu löschen. Damit nicht zu viele Fragen auf einmal bei der Verwaltung des Quiz angezeigt werden, werden die Fragen unter ihrer Kategorie angezeigt und mit einem Dropdown kann zwischen Kategorien gewechselt werden. Sind mehr als eine vordefinierte Anzahl Fragen definiert, kann man die Seite wechseln («Pagination») damit einzelne Seiten nicht zu viele Elemente auf einmal beinhalten.

Bei der Bearbeitung einer Frage kann die Kategorie verändert werden und die Antworten der Frage können bearbeitet werden. Da jedoch jede Frage 4 Antworten erfordert können keine Fragen hinzugefügt/gelöscht werden. Ganze Fragen können von Systemadministratoren gelöscht werden.

### Wer Wird Millionär?

Admin-Login

### Admininterface

### Quiz verwalten

Länder

▼

+

Frage	Anzahl Antworten	
Wo liegt Azerbaijan?	4	Edit
Wie heisst die Hauptstadt von Azerbaijan?	4	Edit
Welche Länder grenzen an Azerbaijan?	4	Edit
Unter welchem Namen sind die Leute bekannt in Azerbaijan?	4	Edit

### Frage bearbeiten

Wie gross ist Azerbaijan?

Kategorie

Länder

▼

**Antworten**

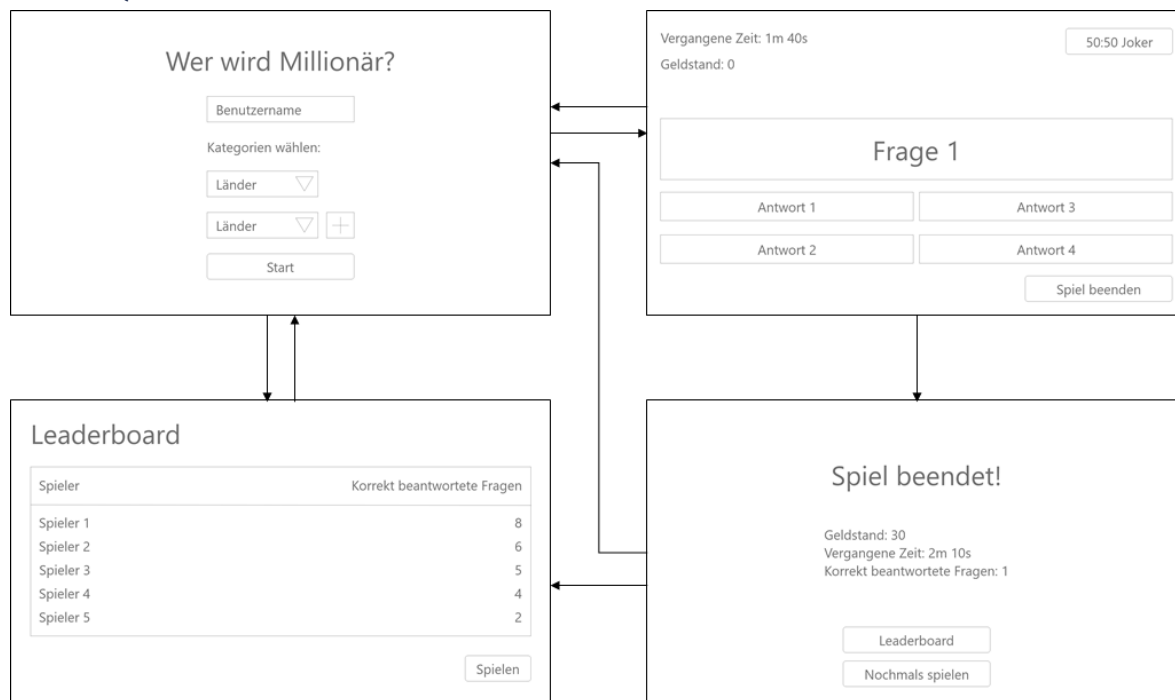
- 42'690 km<sup>2</sup>
- 666'000 km<sup>2</sup>
- Sehr gross
- 24 km<sup>2</sup>

### Leaderboard

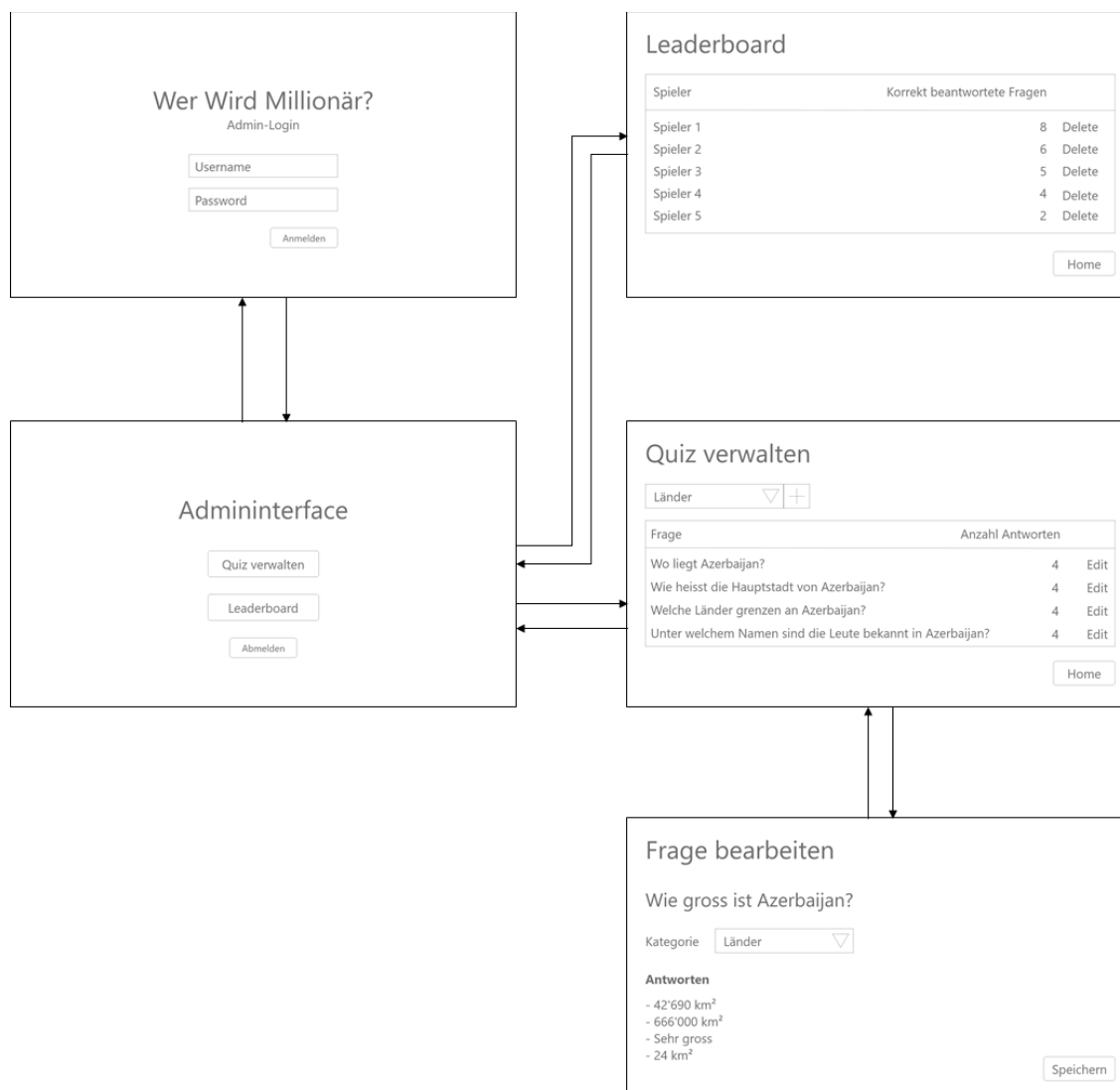
Spieler	Korrekt beantwortete Fragen	
Spieler 1	8	Delete
Spieler 2	6	Delete
Spieler 3	5	Delete
Spieler 4	4	Delete
Spieler 5	2	Delete

## 2.5 Storyboard

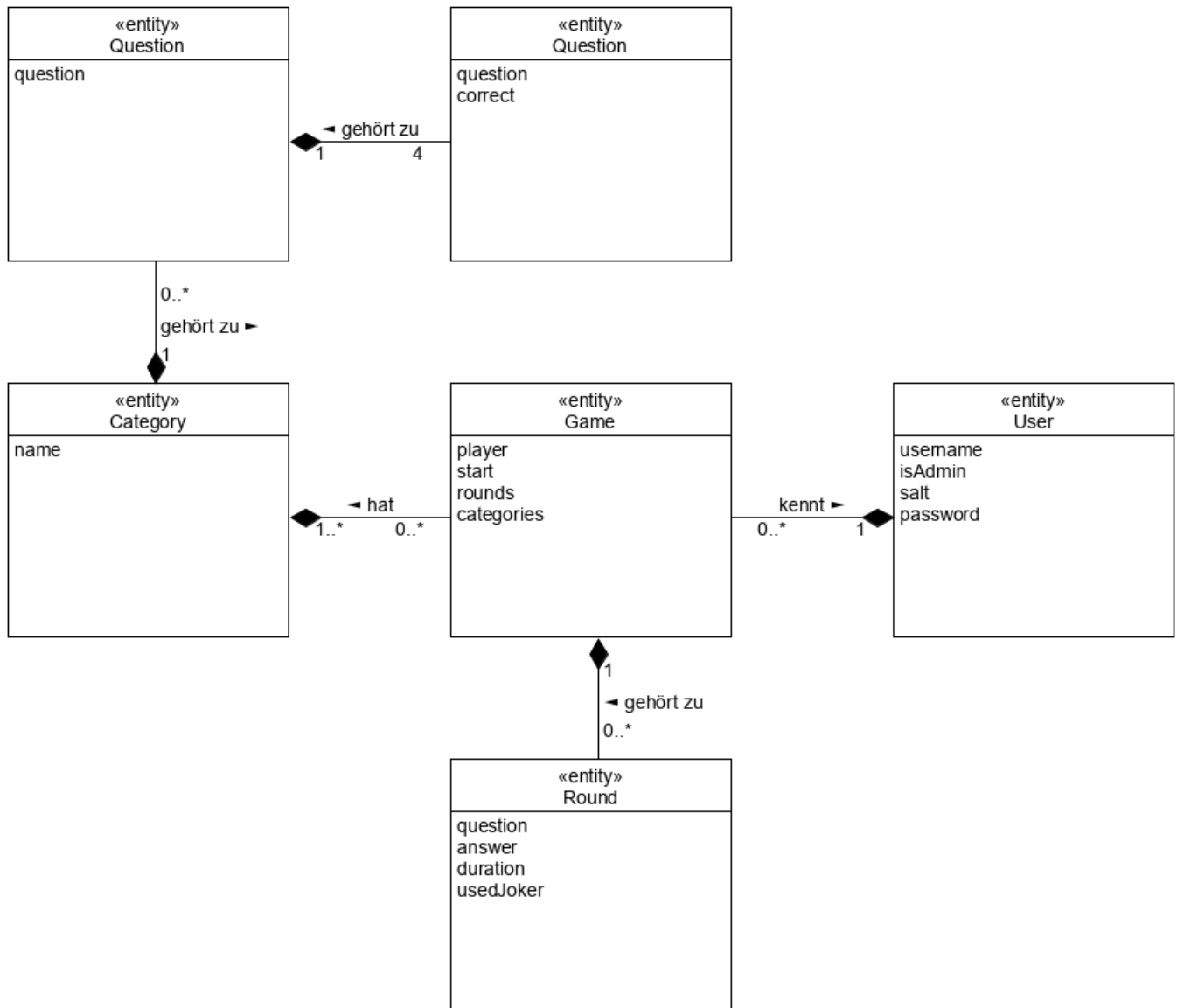
### 2.5.1 Quiz-Seite



### 2.5.2 Admininterface

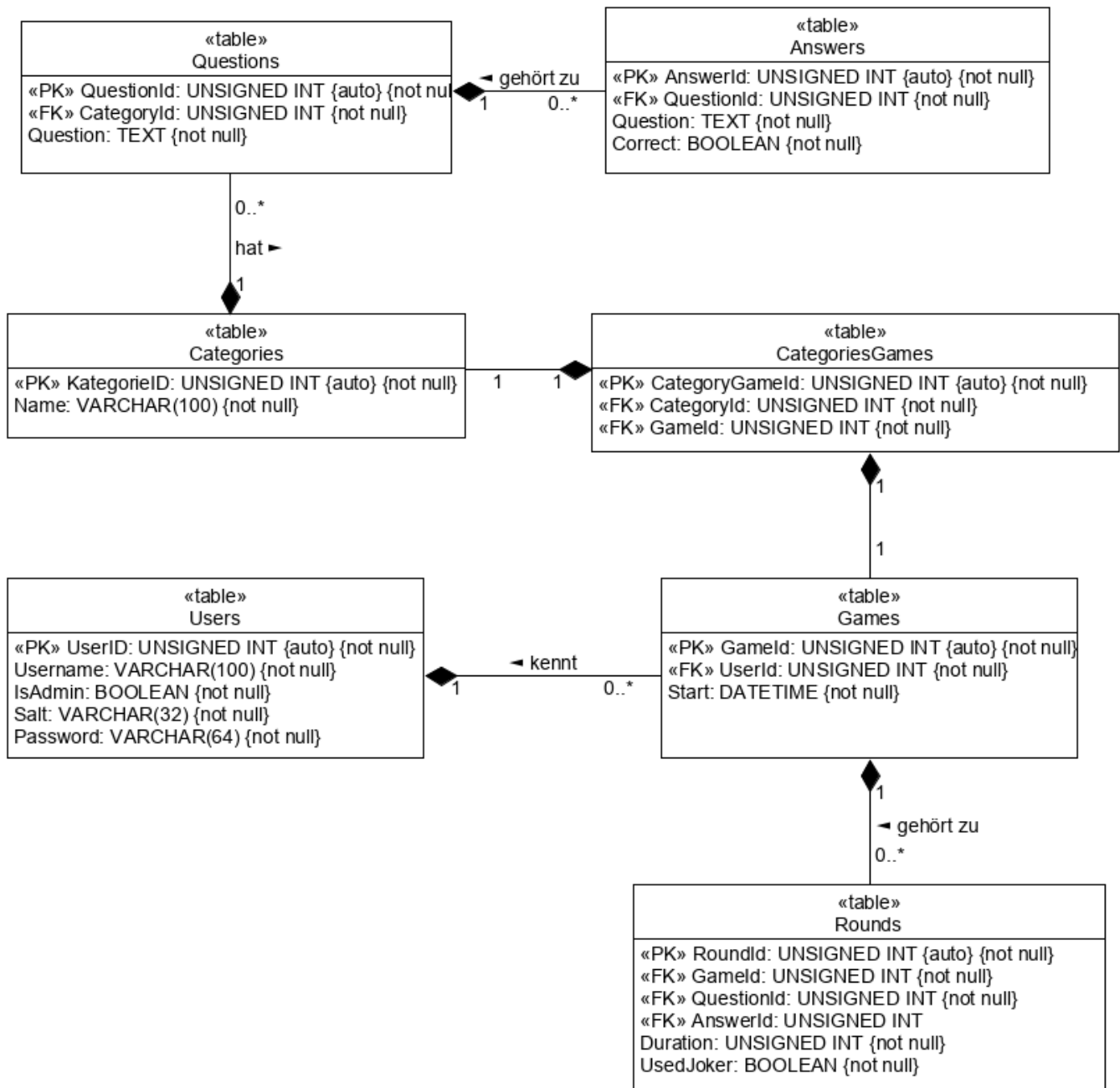


## 2.6 Konzeptionelles Datenmodell (Datenbank)



// fix Question -> Answer

## 2.7 Logisches Datenmodell (Datenbank)



// fix Answers.Question -> Answers.Answer, UserID -> UserId

## 2.8 Testfallspezifikationen

Testfallspezifikation: Wer Wird Millionär?

Betriebssystemversion:

Windows 10

TestfallNr.	Anforderung	Voraussetzungen	Eingaben	Erwartetes Resultat
1	2.1	<ul style="list-style-type: none"> <li>Web-Projekt geöffnet &amp; gestartet</li> <li>MariaDB Server läuft</li> </ul>	<ul style="list-style-type: none"> <li>«TestUser» als Benutzername eingeben</li> <li>«user123» im Passwort-Feld eingeben</li> <li>Auf «Log in» klicken</li> </ul>	Eine Übersicht der verfügbaren Kategorien wird angezeigt.
2	1.3, 1.3.1, 2.1	<ul style="list-style-type: none"> <li>Web-Projekt geöffnet</li> <li>MariaDB Server läuft</li> </ul>	<ul style="list-style-type: none"> <li>Wie in Testfall 1 in das Quiz anmelden</li> <li>«Automotives» als Kategorie auswählen</li> <li>Auf «Play» klicken</li> </ul>	Die erste Frage des Quiz wird angezeigt, sie sollte 4 Antworten zur Auswahl stellen. Oben wird die vergangene Zeit angezeigt, sowie die verfügbare Zeit für die aktuelle Frage.
3	1.3, 1.3.3	<ul style="list-style-type: none"> <li>Web-Projekt geöffnet &amp; gestartet</li> <li>MariaDB Server läuft</li> </ul>	<ul style="list-style-type: none"> <li>Quiz wie in Testfall 2 starten</li> <li>Antwort zu einer Frage auswählen</li> </ul>	Ist die Antwort richtig: Das Quiz leitet den Benutzer zur nächsten Frage, ist sie falsch; Das Quiz wird beendet und die vergangene Zeit, sowie weitere Statistiken des Spiels werden angezeigt.
4	1.3, 1.3.2	<ul style="list-style-type: none"> <li>Web-Projekt geöffnet &amp; gestartet</li> <li>MariaDB Server läuft</li> </ul>	<ul style="list-style-type: none"> <li>Quiz wie in Testfall 2 starten</li> <li>Auf «Use Joker» klicken</li> </ul>	Zwei der vier möglichen Antworten zur Frage verschwinden, diese können wie gewohnt ausgewählt werden.
5	1.5, 1.5.1	<ul style="list-style-type: none"> <li>Web-Projekt geöffnet &amp; gestartet</li> <li>MariaDB Server läuft</li> </ul>	<ul style="list-style-type: none"> <li>«TestAdmin» als Benutzername eingeben</li> <li>«admin123» im Passwort-Feld eingeben</li> <li>Auf «Log in» klicken</li> </ul>	Das Admininterface wird geladen. Eine Übersicht der Fragen in der zuerst ausgewählten Kategorie wird angezeigt.
6	1.5, 1.5.1	<ul style="list-style-type: none"> <li>Web-Projekt geöffnet &amp; gestartet</li> <li>MariaDB Server läuft</li> </ul>	<ul style="list-style-type: none"> <li>Admininterface wie in Testfall 5 starten</li> <li>Auf «Add Question» klicken</li> <li>Im Eingabefeld «Question» «Testfrage» eingeben</li> </ul>	Der Button, auf welchem «Add» steht, wechselt kurz auf «Saving changes...» und dann auf «Save». Beim Neuladen des Admininterface wird die Frage wie erwartet persistiert.

			<ul style="list-style-type: none"> <li>Als Antworten «Antwort 1», «Antwort 2», «Antwort 3», «Antwort 4» angeben</li> <li>Auf «Add» klicken</li> </ul>	
<b>7</b>	1.5, 1.5.1	<ul style="list-style-type: none"> <li>Web-Projekt geöffnet &amp; gestartet</li> <li>MariaDB Server läuft</li> </ul>	<ul style="list-style-type: none"> <li>Admininterface wie in Testfall 5 starten</li> <li>Auf die erste Frage klicken</li> <li>Im Eingabebereich die Frage auf «Geänderte Frage» ändern</li> <li>Auf «Save» klicken</li> </ul>	Die Frage wird gespeichert und beim Neuladen des Interface die Änderungen persistiert.
<b>8</b>	1.5, 1.5.2	<ul style="list-style-type: none"> <li>Web-Projekt geöffnet &amp; gestartet</li> <li>MariaDB Server läuft</li> </ul>	<ul style="list-style-type: none"> <li>Admininterface wie in Testfall 5 starten</li> <li>Im Navigationsmenü «Leaderboard» auswählen</li> </ul>	Eine Übersicht der Highscoreliste wird geladen. Neben jedem Eintrag ist ein Knopf mit welchem der Eintrag gelöscht werden kann, der mit «Delete» beschriftet ist.
<b>9</b>	1.5, 1.5.2	<ul style="list-style-type: none"> <li>Web-Projekt geöffnet &amp; gestartet</li> <li>MariaDB Server läuft</li> </ul>	<ul style="list-style-type: none"> <li>Leaderboard mit Admin-Berechtigungen wie in Testfall 8 starten</li> <li>Beim ersten Eintrag «Delete» klicken</li> </ul>	Der Eintrag wird gelöscht und die Highscoreliste wird gespeichert; wenn man sie neu ladet ist der Eintrag nicht mehr sichtbar.
<b>10</b>	2.2	<ul style="list-style-type: none"> <li>Web-Projekt geöffnet &amp; gestartet</li> <li>MariaDB Server läuft</li> </ul>	<ul style="list-style-type: none"> <li>Quizbereich wie in Testfall 1 starten</li> <li>Im Navigationsmenü «My Games» auswählen</li> </ul>	Mind. 1 Spiel wird angezeigt; mit Klick auf den Titel («Game played at [Datum, Zeit]») wird eine Übersicht der Runden geladen.
<b>11</b>	1.3, 2.3	<ul style="list-style-type: none"> <li>Web-Projekt geöffnet &amp; gestartet</li> <li>MariaDB Server läuft</li> </ul>	<ul style="list-style-type: none"> <li>Quiz wie in Testfall 2 starten</li> <li>2 Minuten warten</li> </ul>	Das Spiel wird automatisch beendet.

## 3 Entscheiden

### 3.1 Entscheidungsmatrix

#### 3.1.1 Web-Framework & Back-End

Kriterien	Gewichtung	JavaServer Faces		React mit Node.js back-end		ASP.NET Core MVC mit React	
		Punkte	Total	Punkte	Total	Punkte	Total
Einfachheit der Programmierung	3	0	0	3	6	3	9
Performance	2	2	4	2	4	3	6
Sicherheit	3	3	9	3	9	3	9
Unterstützung von modernen Features	3	1	3	3	9	3	9
Total		16		28		33	

Hiermit ist klar ersichtlich, dass dank der Einfachheit der Programmierung und der starken Performance eines C# back-ends die Kombination dieser mit einem React Pages front-end die beste Wahl ist. Hinzu kommt der Support von tausenden modernen Bibliotheken, welche das Erstellen eines GUIs stark vereinfachen (Material UI, Bootstrap, Semantic etc.) und einer sehr aktiven Userbase die einem bei Problemen oder Unklarheiten helfen können.

Der Webserver ist bei React standardweise ein Node.js Server. Dies wird so beibehalten, jedoch ist der Application Server, welcher dann auch direkt mit der Datenbank (s. [3.2.2 «Dataserver»](#)) verbunden ist, eine ASP.NET Core C# Applikation, die über eine REST API mit dem Presentation Layer kommunizieren wird.

Object-relational mapping ist eine Programmiermethode mit welcher die Struktur eines Datenformats in die einer anderen konvertiert wird. Da die Struktur der Datenbanktabellen nicht 1:1 die der Klassen in C# entsprechen wird, werden die Datensätze aus der Datenbank mit Dapper in C# Objekte umgewandelt. Dapper erlaubt dem Programmierer weiterhin die SQL Statements selbst zu schreiben, und übernimmt nur die Transformation in die neue Struktur. Dies hat den Vorteil, dass bei Veränderungen der Datenbank nicht wie beim EntityFramework die Datenbank von einem Tool neu analysiert werden muss und nur die SQL Statements umgeschrieben werden müssen, was bei kleinen Projekten deutlich Zeit einspart und die Effizienz von diesen SQL Statements besser sein kann, als wenn sie von einem Tool generiert werden.

## 3.2 Entscheidungen

### 3.2.1 Dynamische Elemente der Anwendung

Die dynamischen Elemente der Anwendung werden grundsätzlich auf dem **ersten Tier (Presentation)** untergebracht da Benutzer (inklusive Administratoren) über das GUI, welches auf diesem Tier untergebracht ist, ihre Aufgaben erledigen werden.

Input-Validation passiert auf dem front- sowie back-end weshalb dieser Teil dann vom **zweiten und dritten Tier (Webserver & Application Server)** übernommen wird, alle anderen dynamischen Elemente der Anwendung befinden sich, wie vorher erwähnt, auf dem **ersten Tier (Presentation)**.

### 3.2.2 Dataserver

Als Dataserver wird die MariaDB, welche als Teil von XAMPP mitgeliefert wird, eingesetzt. Somit sind alle Daten lokal gespeichert und die Operationen mit ihr werden mit SQL geschrieben. MariaDB unterstützt sehr viele Datentypen und komplexe Lese- sowie Schreiboperationen die die Effizienz der Kommunikation zwischen Application Server und dem Dataserver erhöhen können. Da es in C# das MySqlConnection Framework gibt, muss keine eigene Implementation programmiert werden. Dieses Framework wird im Hintergrund die Authentifikation und Kommunikation mit dem Dataserver übernehmen und mit Dapper wird das object-relational mapping (s. [3.1.1 «Web-Framework & Back-End»](#)) realisiert.

## 4 Realisieren

### 4.1 Programm

#### 4.1.1 Ideenumsetzung

### 4.2 GUI



## 5 Kontrollieren

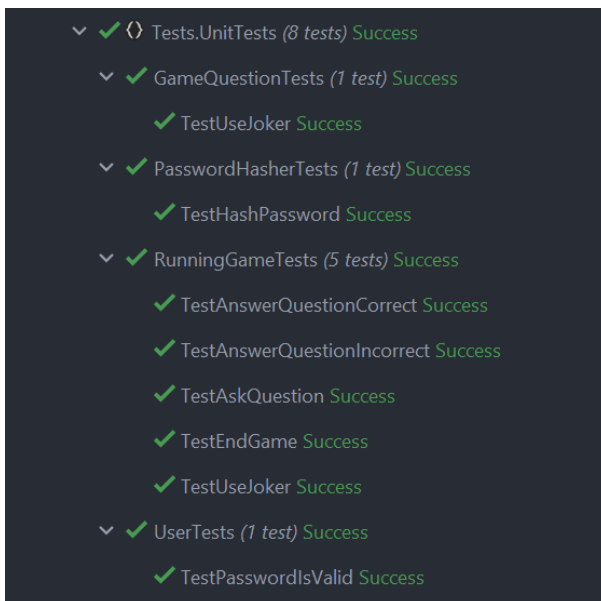
### 5.1 Testprotokoll

**Testprotokoll:** Wer Wird Millionär?

Testfallnummer	Datum	Tester	Bemerkungen	Resultat	Unterschrift
1	23.03.2020	RaviAnand Mohabir		OK	RaviAnand M.
2	23.03.2020	RaviAnand Mohabir		OK	RaviAnand M.
3	23.03.2020	RaviAnand Mohabir		OK	RaviAnand M.
4	23.03.2020	RaviAnand Mohabir		OK	RaviAnand M.
5	23.03.2020	RaviAnand Mohabir		OK	RaviAnand M.
6	23.03.2020	RaviAnand Mohabir		OK	RaviAnand M.
7	23.03.2020	RaviAnand Mohabir	- Beim Neuladen der Seite wird man ausgeloggt, man muss sich wieder einloggen, um das Resultat zu überprüfen.	OK	RaviAnand M.
8	23.03.2020	RaviAnand Mohabir		OK	RaviAnand M.
9	23.03.2020	RaviAnand Mohabir	- Beim Neuladen der Seite wird man ausgeloggt, weitere Änderungen können nur bei einer Neu-Anmeldung vorgenommen worden.	OK	RaviAnand M.
10	23.03.2020	RaviAnand Mohabir		OK	RaviAnand M.
11	23.03.2020	RaviAnand Mohabir		OK	RaviAnand M.

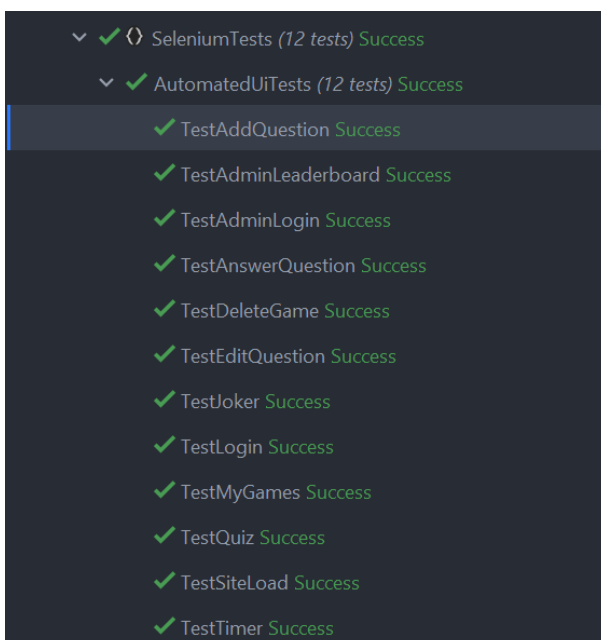
## 5.2 Funktionstest (Unit-Test)

Um die Business-Logik zu Testen, wurden für die verschiedenen Models Unit-Tests erstellt, die den Logikverlauf testen. In derselben Solution des Hauptprojekts wurde ein Projekt «WhoWantsToBeAMillionaire.Tests» erstellt, worin die Unit-Tests abgespeichert sind und mit einer beliebigen IDE wie Visual Studio oder JetBrains Rider durchgespielt werden können.



## 5.3 Integrationstest (Selenium)

Mit Selenium wurden alle Testfälle im selben Testprojekt, wie die Unit-Tests, einprogrammiert, und durchgeführt. Im Stil von TDD wurde jeder Testfall durchgeführt, und dann Änderungen am Projekt vorgenommen bis Issues behoben wurden. Diese Issues wurden auch auf dem GitHub Projekt verfolgt und nacheinander durchgearbeitet.



## 5.4 Testfazit

Beim durchlaufen der funktionalen sowie automatisierten Integrationstest kommen keine Fehler auf. Auch beim Testen der Applikation mit Hilfe der Testspezifikationen, wie im Testprotokoll zu sehen ist, funktioniert die Applikation wie gewollt. Die Anforderungen sind somit vollkommen gedeckt und die Applikation produktionsbereit.

## 6 Auswerten

### 6.1 Reflexion

## 7 Anhang

### 7.1 Quellen

### 7.2 Code