24.01.2020

RaviAnand Mohabir

BBBaden

IPERKA M151

Projektdokumentation zum Modul 151 «Datenbanken in Webauftritt einbinden»

Inhaltsverzeichnis

[1 Informieren 3](#_Toc30737743)

[1.1 Einleitung 3](#_Toc30737744)

[1.2 Anforderungsanalyse 4](#_Toc30737745)

[1.3 Ideen 5](#_Toc30737746)

[2 Planen 6](#_Toc30737747)

[2.1 Systemgrenze 6](#_Toc30737748)

[2.2 Use-Case 6](#_Toc30737749)

[2.3 Klassendiagramm 6](#_Toc30737750)

[2.4 Mock-Up (GUI) 7](#_Toc30737751)

[2.4.1 Quiz-Seite 7](#_Toc30737752)

[2.4.2 Admininterface 7](#_Toc30737753)

[2.5 Storyboard 8](#_Toc30737754)

[2.5.1 Quiz-Seite 8](#_Toc30737755)

[2.5.2 Admininterface 8](#_Toc30737756)

[2.6 Konzeptionelles Datenmodell (Datenbank) 8](#_Toc30737757)

[2.7 Logisches Datenmodell (Datenbank) 8](#_Toc30737758)

[2.8 Testfallspezifikationen 9](#_Toc30737759)

[3 Entscheiden 10](#_Toc30737760)

[3.1 Entscheidungsmatrix 10](#_Toc30737761)

[3.2 Entscheidungen 10](#_Toc30737762)

[3.2.1 Dynamische Elemente der Anwendung 10](#_Toc30737763)

[4 Realisieren 11](#_Toc30737764)

[4.1 Programm 11](#_Toc30737765)

[4.1.1 Ideenumsetzung 11](#_Toc30737766)

[4.2 GUI 11](#_Toc30737767)

[5 Kontrollieren 12](#_Toc30737768)

[5.1 Testprotokoll 12](#_Toc30737769)

[5.2 Funktionstest 13](#_Toc30737770)

[5.3 Integrationstest (Selenium) 13](#_Toc30737771)

[5.4 Testfazit 13](#_Toc30737772)

[6 Auswerten 14](#_Toc30737773)

[6.1 Reflexion 14](#_Toc30737774)

[7 Anhang 15](#_Toc30737775)

[7.1 Quellen 15](#_Toc30737776)

[7.2 Code 15](#_Toc30737777)

# Informieren

## 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.

## Anforderungsanalyse

**Anforderungsanalyse:** WerWird Millionär?

|  |  |  |
| --- | --- | --- |
| Anforderungs-nummer | Anforderung | Beschreibung |
| 1 | Grundanforderungen |  |
| 1.1 | Tiers | Die verschiedenen Komponenten des Systems werden voneinander und der Datenbank getrennt. Presentation-Layer wird für jeden Client geladen und Daten davon nicht getraut, Business-Logik wird einmal im dritten Tier programmiert und die Datenbank ist nur für den Server sichtbar. |
| 1.2 | 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. |
| 1.3 | 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. |
| 1.3.1 | Zeitmessung | Während dem Quiz wird die vergangene Zeit seit dem Start gemessen und dem Benutzer angezeigt, um die Recherchemöglichkeiten einzuschränken. |
| 1.3.2 | 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. |
| 1.3.3 | 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. |
| 1.4 | 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. |
| 1.5 | Admininterface |  |
| 1.5.1 | Quiz verwalten | Über das Admininterface können Quizfragen hinzugefügt werden, ihre Kategorie eingerichtet werden und Antworten zu jeder Quizfrage zuweisen. |
| 1.5.2 | Highscoreliste verwalten | Administratoren können einzelne Einträge von der Highscoreliste löschen. |
| 2 | Erweiterte Anforderungen |  |
| 2.1 | Fragen/Antworten kommentieren | Admins können Fragen und Antworten mit Kommentaren versehen als interne Notiz. |
| 2.2 | Statistiken | Fragen speichern wie oft sie hervorgekommen sind, Kategorien wie oft sie gewählt wurden und Antworten speichern wie oft sie von Benutzern als die richtige Antwort gewählt wurden. |

## Ideen

## 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.

### Web-Frameworks

#### 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 Simplizität der Programmierung und Effizienz im Vergleich zu modernen Frameworks.

#### 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.

#### 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.

# Planen

## Systemgrenze

## Use-Case

## Klassendiagramm

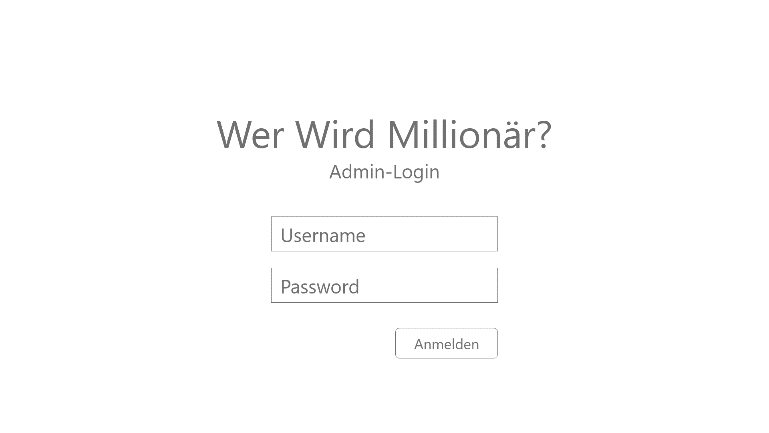
## Mock-Up (GUI)

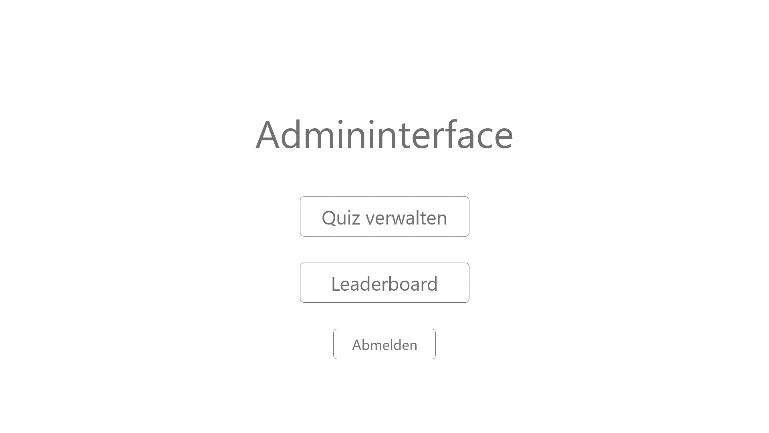
### Quiz-Seite

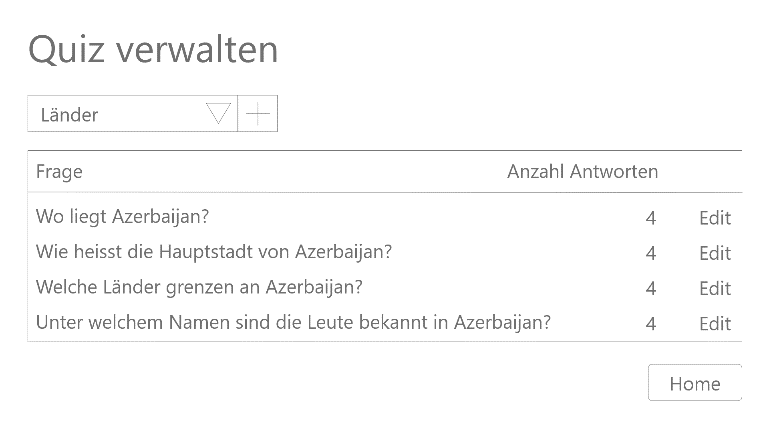
### Admininterface

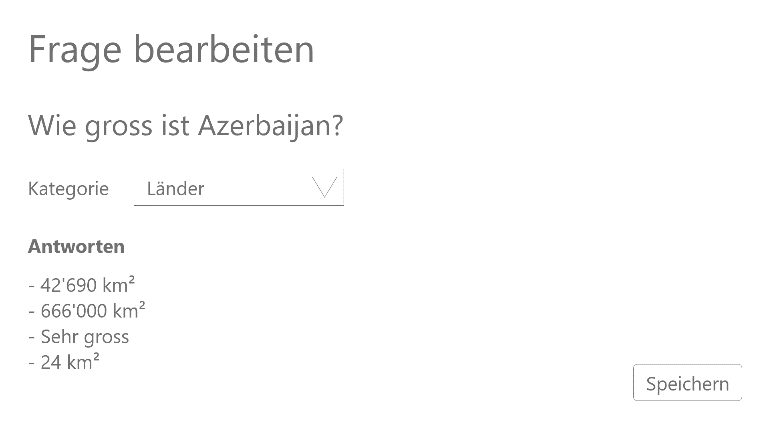
Nach dem Anmelden sehen Administratoren eine Auswahl welches ihnen erlaubt das Quiz zu bearbeiten oder Einträge aus der Highscoreliste zu löschen. Damit nicht zu viele Fragen auf einmal bei der Verwaltung des Quiz angezeigt werden, werden die Fragen unter ihrer Kategorie angezeigt und mit einem Dropdown 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.

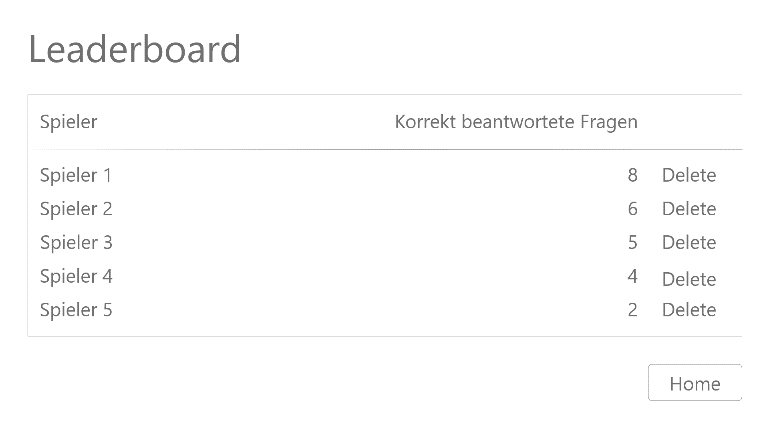
Bei der Bearbeitung einer Frage kann die Kategorie verändert werden und die Antworten der Frage können bearbeitet werden. Da jedoch jede Frage 4 Antworten erfordert können keine Fragen hinzugefügt/gelöscht werden. Ganze Fragen können von Systemadministratoren gelöscht werden.





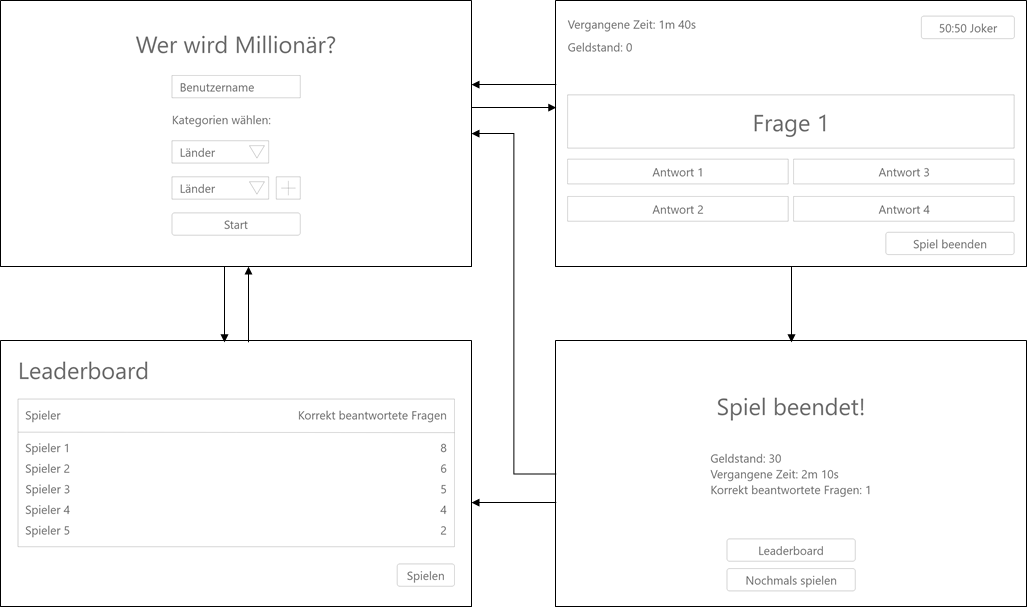




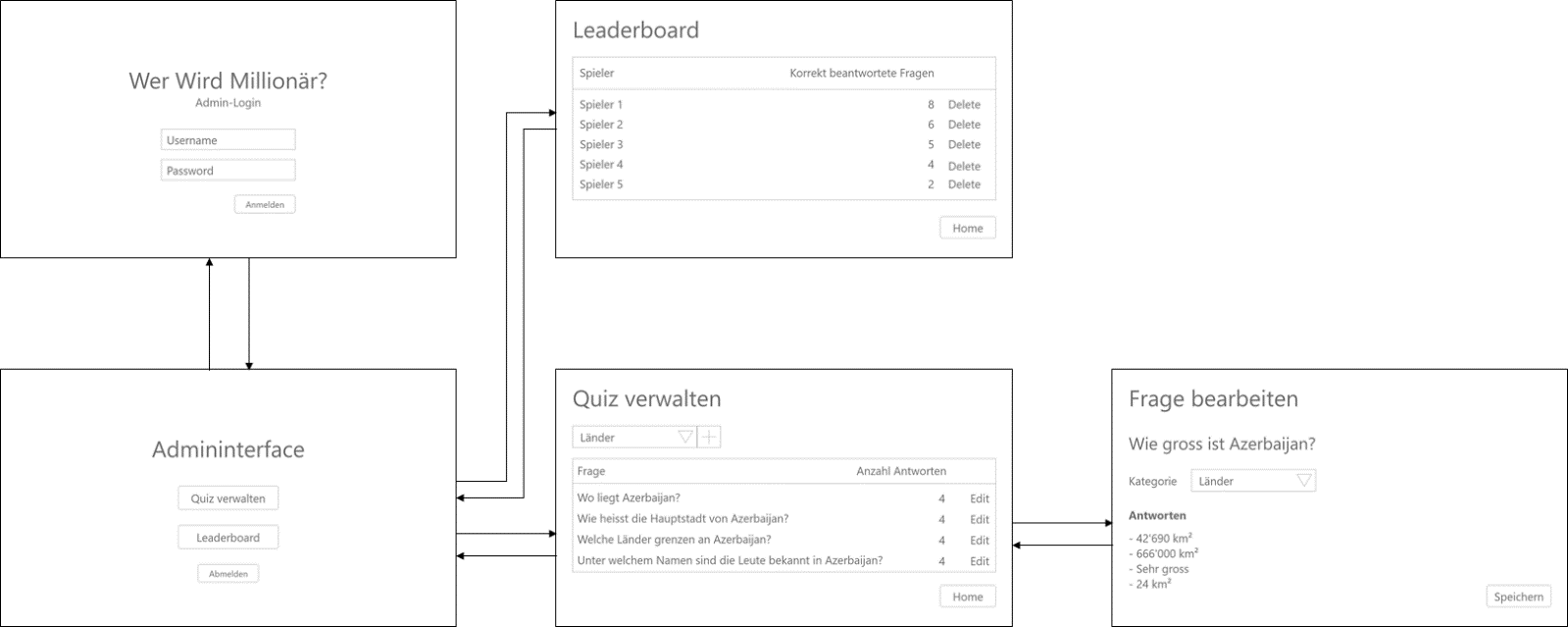


## Storyboard

### Quiz-Seite



### Admininterface



## Konzeptionelles Datenmodell (Datenbank)

## Logisches Datenmodell (Datenbank)

## Testfallspezifikationen

**Testfallspezifikation:** [Projektname]

**Betriebssystemversion:** Windows 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Testfallnummer | Anforderung | Voraussetzungen | Eingaben | Erwartetes Resultat |
| 1 |  |  |  |  |
|  |  |  |  |  |

# Entscheiden

## Entscheidungsmatrix

### Web-Framework

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 Nutzerschaft die einem bei Problemen oder Unklarheiten helfen können. Es wird auf einen Node.js Server verzichtet da dieser Fehler-anfälliger ist und nicht dieselbe Performance wie C# zur Verfügung stellt.

## Entscheidungen

### 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 Tier (Webserver)** übernommen wird, alle anderen dynamischen Elemente der Anwendung befinden sich, wie vorher erwähnt, auf dem **ersten Tier (Presentation)**.

# Realisieren

## Programm

### Ideenumsetzung

## GUI

# Kontrollieren

## Testprotokoll

**Testprotokoll:** Wer Wird Millionär?

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr. | Datum | Tester | Bemerkungen | Resultat | Unterschrift |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |

## Funktionstest

## Integrationstest (Selenium)

## Testfazit

# Auswerten

## Reflexion

# Anhang

## Quellen

## Code