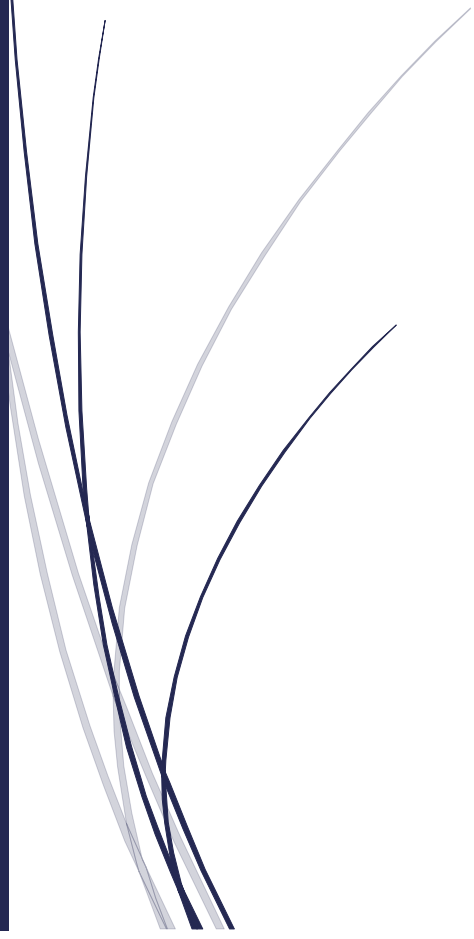
17.02.2020

IPERKA M151

Projektdokumentation zum Modul 151 «Datenbanken in Webauftritt einbinden»



RaviAnand Mohabir
BBBADEN

Inhaltsverzeichnis

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.
- Spielstatistiken speichern auch die ausgewählten Kategorien, somit sieht man auch die populärsten Kategorien und die, die am meisten ausgewählt wurden.
- 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, endet das Spiel und der bis jetzt errungenen Betrag wird ausbezahlt.
- 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?

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.

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 Simplizität 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

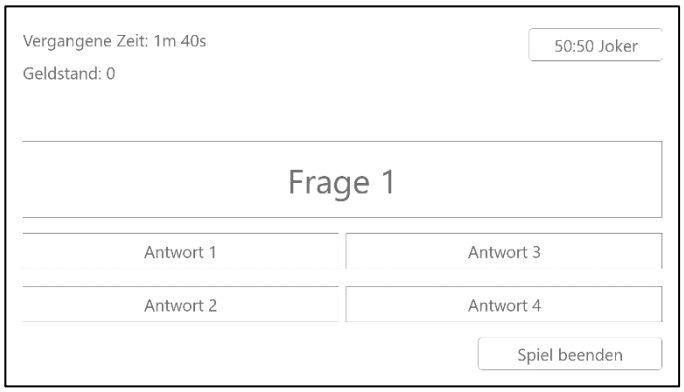
2.4 Mock-Up (GUI)

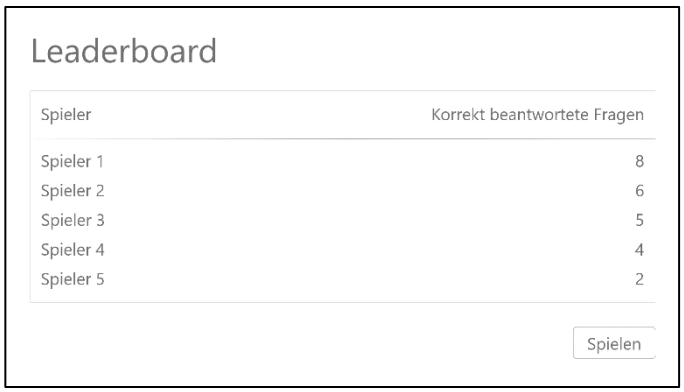
2.4.1 Quiz-Seite

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.

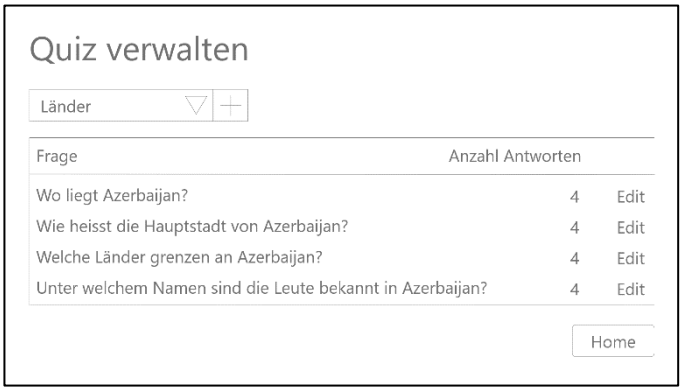


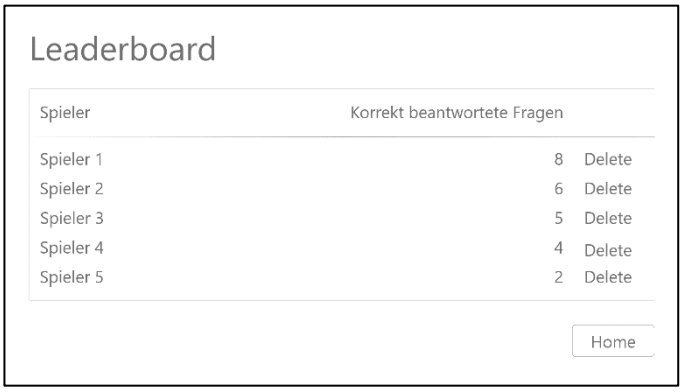


2.4.2 Admininterface

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.

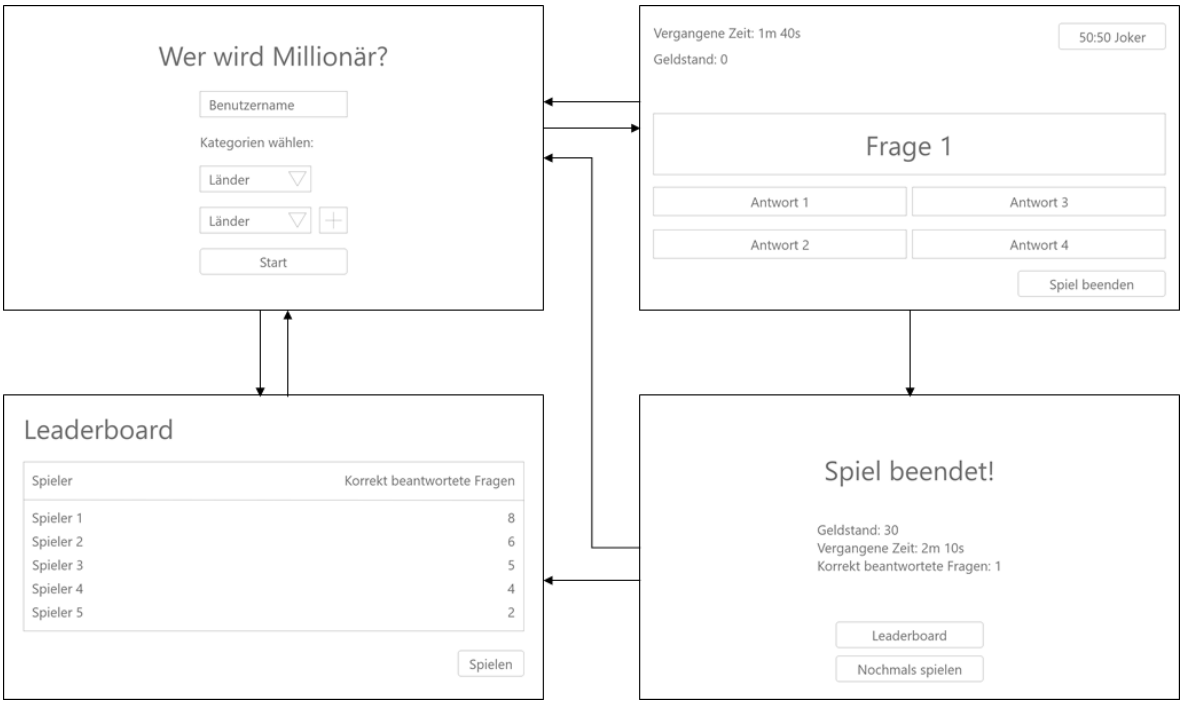
Bei der Bearbeitung einer Frage kann die Kategorie verändert werden und die Antworten der Frage können bearbeitet werden. Da jedoch jede Frage 4 Antworten erfordert können keine Fragen hinzugefügt/gelöscht werden. Ganze Fragen können von Systemadministratoren gelöscht werden.



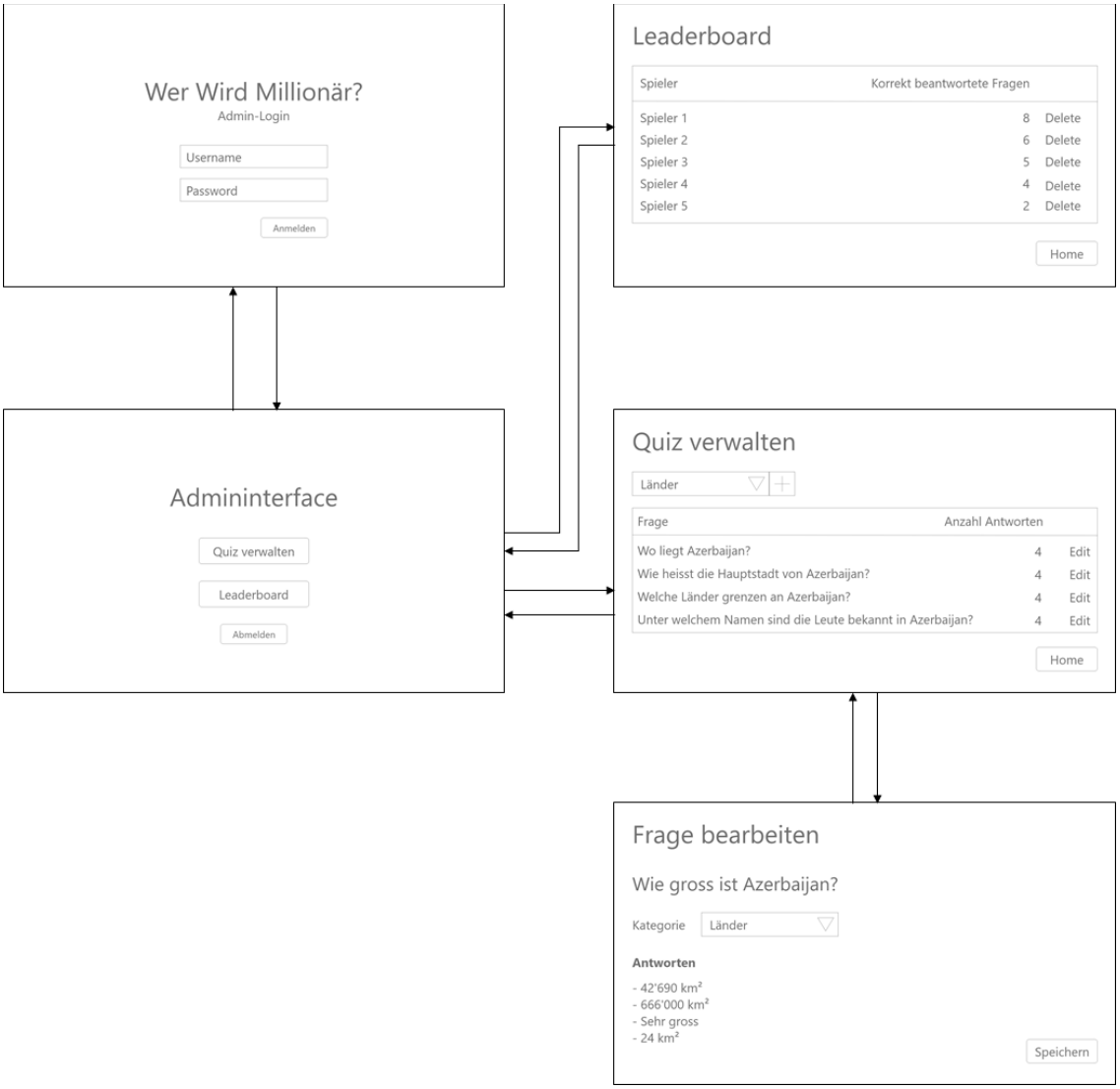


2.5 Storyboard

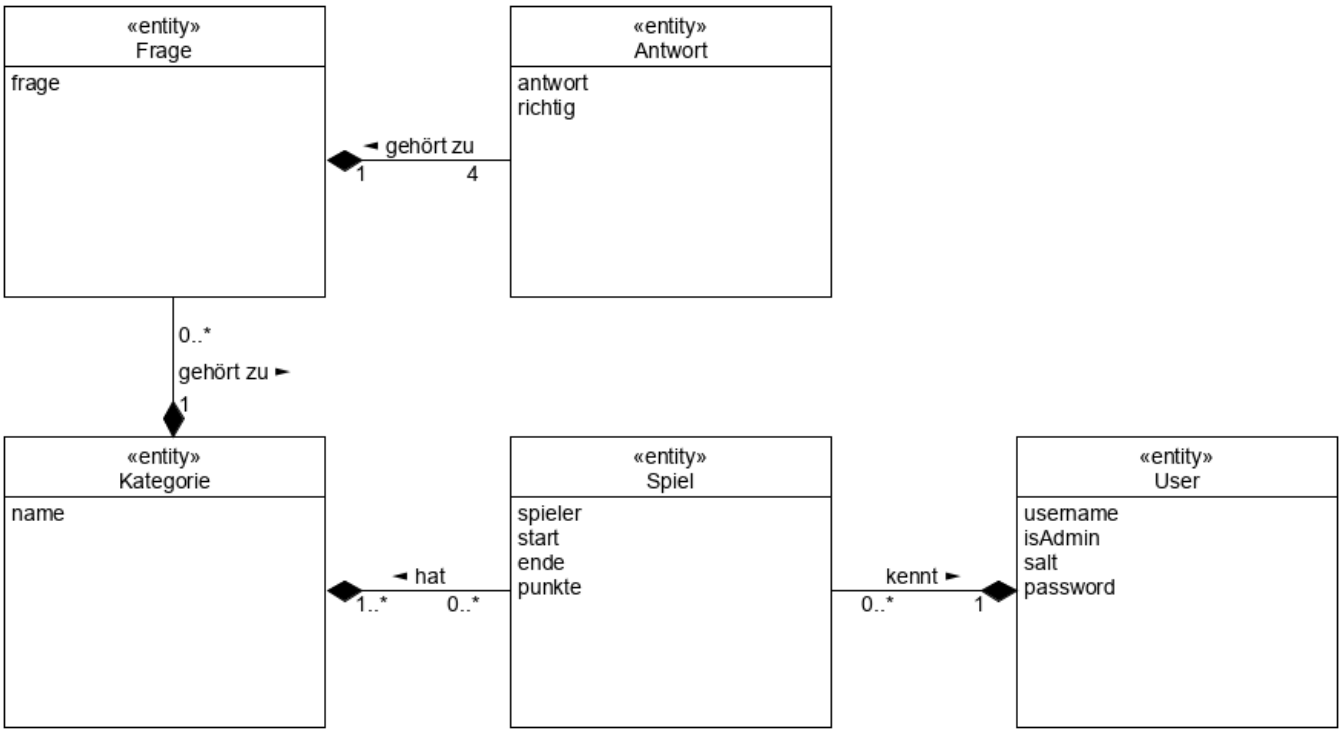
2.5.1 Quiz-Seite



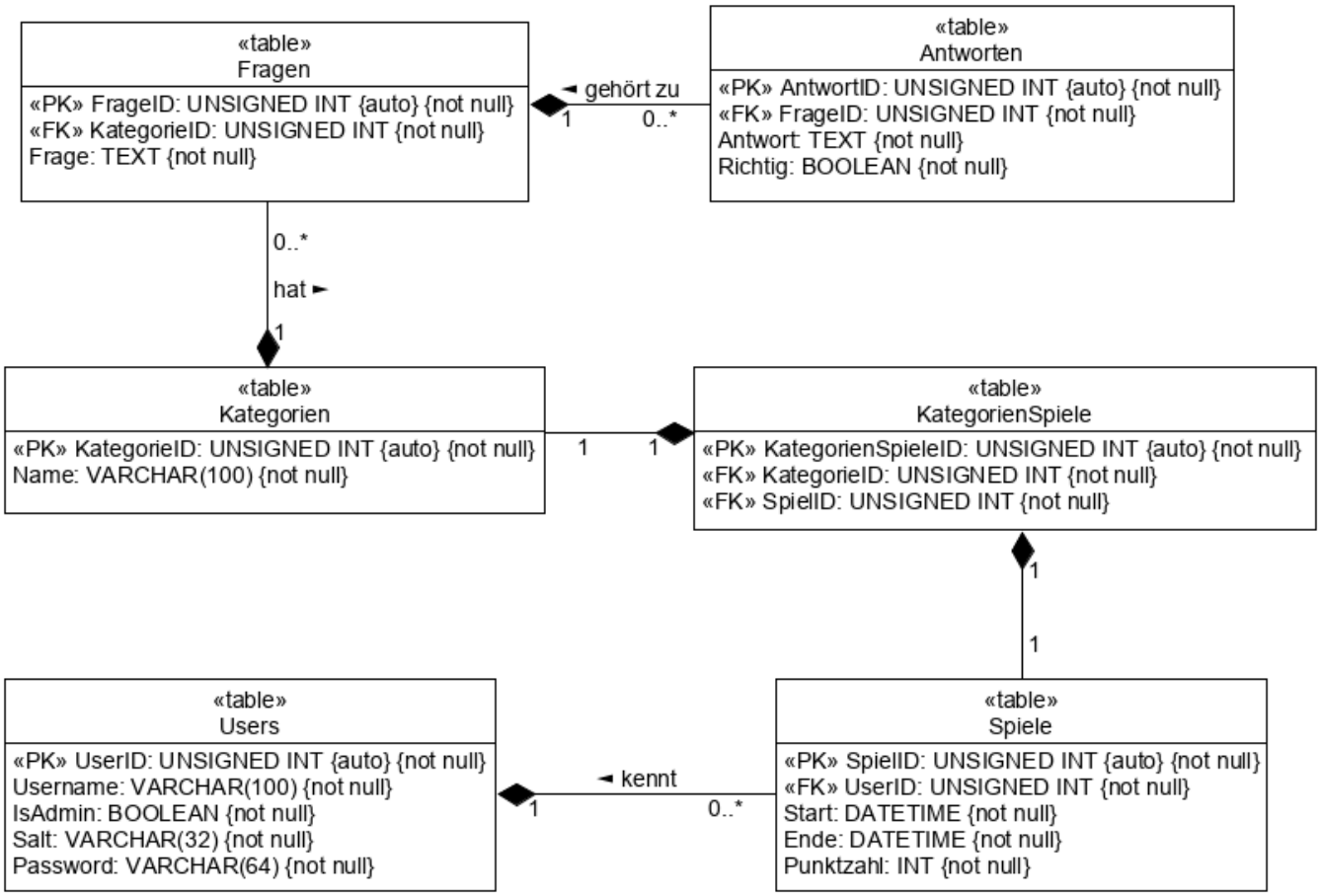
2.5.2 Admininterface



2.6 Konzeptionelles Datenmodell (Datenbank)



2.7 Logisches Datenmodell (Datenbank)



2.8 Testfallspezifikationen

Testfallspezifikation: [Projektname] **Betriebssystemversion:** Windows 10

Testfallnummer	Anforderung	Voraussetzungen	Eingaben	Erwartetes Resultat
1	1.3, 1.3.1	• Web-Projekt geöffnet & gestartet • MariaDB Server läuft	• Benutzername im Eingabefeld angeben • Quizfrage-Kategorien aussuchen • Auf «Quiz starten» drücken	Die erste Frage des Quiz wird angezeigt, sie sollte 4 Antworten zur Auswahl stellen. Oben wird die vergangene Zeit angezeigt.
2	1.3, 1.3.3	• Web-Projekt geöffnet & gestartet, Quiz wie in Testfall 1 geladen • MariaDB Server läuft	• Quiz wie in Testfall 1 starten • Antwort zu einer Frage auswählen	Ist die Antwort richtig: Das Quiz leitet den Benutzer zur nächsten Frage, ist sie falsch; Das Quiz wird beendet und die vergangene Zeit, sowie Statistiken des Spiels werden angezeigt.
3	1.3, 1.3.2	• Web-Projekt geöffnet & gestartet, Quiz wie in Testfall 1 geladen • MariaDB Server läuft	• Quiz wie in Testfall 1 starten • Oben den 50:50-Joker auswählen	Zwei der vier möglichen Antworten zur Frage verschwinden, diese können wie gewohnt ausgewählt werden.
4	1.5	• Web-Projekt geöffnet & gestartet • MariaDB Server läuft	• Oben auf «Admin» klicken und Admin-Infos angeben (Benutzername «Admin», Passwort «admin1234»)	Das Admininterface wird geladen.
5	1.5, 1.5.1	• Web-Projekt geöffnet & gestartet, Admininterface wie in Testfall 4 geladen • MariaDB Server läuft	• Auf der Startseite des Admininterface «Quiz verwalten» auswählen	Eine Liste aller Quizfragen/Kategorien wird geladen welche vom Admin verändert werden können.
6	1.5, 1.5.1	• Web-Projekt geöffnet & gestartet, Quizverwaltung wie in Testfall 5 geladen • MariaDB Server läuft	• In der Quizverwaltung eine Frage zur Verwaltung auswählen	Eine Detailübersicht der Frage mit ihren Antworten sowie Statistiken wird angezeigt, Antworten können verändert, sowie die Frage gelöscht werden.
7	1.5, 1.5.2	• Web-Projekt geöffnet & gestartet, Admininterface wie in Testfall 4 geladen • MariaDB Server läuft	• Auf der Startseite des Admininterface «Highscoreliste anpassen» auswählen	Eine Übersicht der Highscoreliste wird geladen. Neben jedem Eintrag ist ein Knopf mit welchem der Eintrag gelöscht werden kann.
8	1.5, 1.5.2	• Web-Projekt geöffnet & gestartet, Highscorelistenverwaltung wie in Testfall 7 geladen • MariaDB Server läuft	• Einen Eintrag auswählen und auf den Knopf, der mit «Löschen» markiert ist klicken.	Der Eintrag wird gelöscht und die Highscoreliste wird gespeichert; wenn man sie nun als Spieler ladet ist der Eintrag nicht mehr sichtbar.

9	1.5, 1.5.1, 2.1	• Web-Projekt geöffnet & gestartet, Quizfragenverwaltung wie in Testfall 6 geladen • MariaDB Server läuft	• Im Kommentarbereich der Frage einen Kommentar hinzufügen, bzw. den bereits verfassten Kommentar verändern und die Frage speichern.	Der Kommentar ist nun als Teil der Frage gespeichert, ladet man die Seite erneut ist der Kommentar weiterhin zu sehen.
10	1.5, 1.5.1, 2.2	• Web-Projekt geöffnet & gestartet, Quizfragenverwaltung wie in Testfall 6 geladen • MariaDB Server läuft	•	In der Detailübersicht der Frage sind Statistiken der Frage sowie ihrer Antworten zu sehen; Beispielsweise wie oft die Frage gestellt wurde und wie oft jede Antwort als richtig gewählt wurde.

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.

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

4 Realisieren

4.1 Programm

4.1.1 Ideenumsetzung

4.2 GUI

5 Kontrollieren

5.1 Testprotokoll

Testprotokoll: Wer Wird Millionär?

Nr.	Datum	Tester	Bemerkungen	Resultat	Unterschrift
1					
2					

5.2 Integrationstest (Selenium)

5.3 Testfazit

6 Auswerten

6.1 Reflexion

7 Anhang

7.1 Quellen

7.2 Code