Dokumentation

Teammitglied: Theresa Alt

Rollen:

- Github Admin
- Datenschutzbeauftragte

Aufgabengebiete:

- Accounzuordnung, Rollenverteilung
- Erster Prototype
    > Rangliste
    > Impressum & Datenschutz
- Github
- Erstellung generischer PHP-Klassen

1.Planung des Projekts

Im Rahmen der ersten groben Projektplanung wurde festgelegt, dass eine Webseite mit Datenbankanbindung erstellt wird. Die Kernaufgabe der Webseite liegt darin eigene Quizze erstellen zu können. Die einzelnen Komponenten wurden als folgende festgelegt:

- Login-Seite mit Registrierungsmöglichkeit
- Quiz-Builder-Seite zur Erstellung der Quizze
- Lets-Quiz-Seite zum Spielen der Quizze
- Rangliste-Seite zur Einsicht der Bestenliste
- Impressum und Datenschutz für die Kontaktierung

1.1 Prototyp

Auf Basis dieser Grobvorgaben wurde ein Prototyp zur Veranschaulichung erstellt. Dieser Prototyp hat jedoch keinerlei Funktionalitäten erfüllt, sondern nur gezeigt, wie die Umsetzung des Projekts zu einem späteren Zeitpunkt aussehen könnte.

1.1.1Design des Prototypen:

Wie zuvor im Team besprochen wird als Design Bootstrap genutzt. Bei Bootstrap handelt es sich um eine freies Fontend-CSS-Framework. Die Einbindung erfolgt dabei nicht durch eine lokale Installation, sondern lediglich über einen Link. Dieser Link ruft das Stylesheet von Bootstrap, welches sich genauso wie ein lokales CSS-Stylesheet verwenden lässt.

Dies ermöglicht vor allem ein einheitliches Design, unabhängig davon welches Teammitglied sich den verschiedenen Aufgaben annimmt.
Ein weiterer Entscheidungsgrund für Bootstrap war die einfache Umsetzung von responsive Webdesign. Anstatt für jeden Unterseite ein eigenes Grid-System implementieren zu müssen, ist dies mit Bootstrap durch die das Aufrufen von Klassen gelöst. Damit passt sich die Webseite geräteunabhängig die jeweilige zur Verfügung stehende Fensterbreite an.

Zu meinen persönlichen Aufgaben gehörten bei der Umsetzung des Prototypen das Impressum, der Datenschutz sowie die Rangliste.

1.1.2Rangliste:

Die Generierung der Rangliste erfolgte anhand der zuvor erstellten Datenbank. Die Rangliste wird als Tabelle dargestellt, innerhalb dieser Tabelle zeigt die erste Spalte den jeweilig erreichten Rang des Users an. Die zweite Spalte die korrespondierenden Punkte zu jeweiligen erreichten Rang. Innerhalb der dritten Spalte wird der Username des jeweiligen Spielers dargestellt. In der letzten und vierten Spalte befindet sich der Quiz-Name, passend zu dem abgelegten Quiz.
Innerhalb des ersten Prototypen sollte lediglich die Darstellung umgesetzt werden, jeweilige Sortierfunktionen sowie die Verbindung an die Datenbank werden erst innerhalb der Implementierung vollzogen.

1.1.3 Impressum & Datenschutz:

Das Impressum wurde mittels einer Tabelle dargestellt. In dieser Tabelle wird der Name, das Aufgabengebiet sowie die E-Mail Adresse für Kontakierungsmöglichkeiten gezeigt.

Im Anschluss wird die Datenschutz dargestellt. Dabei wird über Betroffenenrechte aufgeklärte und wie die allgemeinen Informationen während des Beuschs der Webseite erfasst werden. Beispielsweise kann nachgelesen werden, wie und zu welchem Zweck die Registrierungsdaten innerhalb des Projekts gespeichert werden.

Abschließend wird über das Widerspruchsrecht nach Art.21 DSGVO informiert sowie den Vorbehalt etwas an den Datenschutzbestimmungen zu ändern. Für eine klare Kommunikation mit den Nutzern der Webseite, wird zusätzlich noch einmal die Datenschutzbeauftragte benamt und durch die Angabe der E-Mail Adresse eine direkte Möglichkeit für die Kontaktierung geschaffen.

1.2 Weiteres Vorgehen nach Review 1

Nach der Fertigstellung des ersten Prototyp, fand Review 1 mit den Product Owner statt. Innerhalb der gemeinsamen Besprechung, wurden neue Anforderungen an das Projekt herausgearbeitet. Die jeweiligen User sollen Gruppen erstellen können. Innerhalb dieser Gruppen werden die Quizze erstellt, damit man gegeneinander antreten kann. Um diese neuen Anforderungen innerhalb des Projekts sauber umzusetzen, wurden zur Vorbereitung der Implementierung eine OOA und OOD erstellt

2. Objektorientierte Analyse
Textuelle Aufarbeitung des Konzepts

2.1 Welche Objekte gibt es in der Miniwelt?

Innerhalb von BAQ werden folgende Objekte benötigt:

- User

- Gruppen

- Rangliste

- Quiz

- Anmeldung

2.2 Welche Attribute haben die Objekte?

Die benötigten Objekte müssen folgende Informationen beinhalten:

- User:
    > Username
    > E-Mail-Adresse
    > Avatar/Icon
    > Zugehörigkeit zu einer Gruppe
    > Erstellten Quizze
- Rangliste:
    > Gruppen
    > User
    > Punktzahl
    > Gespielte Quizze

- Gruppen:
    > Mitglieder
    > Erstellte Quizze
- Quiz:
    > Freitext-Fragen
    > DropDown-Fragen
    > Multiple-Choice-Fragen
- Anmeldung:
    > Username
    > E-Mail-Adresse
    > Passwort

2.3 Was tun die Objekte?

Die Funktionalitäten der Objekte und erforderlichen Ausgaben:
- User:
    > Anlegung und Änderung eines Avatars
    > Erstellung der Quizze
    > Einsehen der erstellten Quizze
- Gruppen:
    > Erstellung einer Gruppe
    > Zuordnung der Mitglieder
    > Einsehen der Quizze der zugehörigen Gruppe
- Rangliste
    > Anzeige der eigens erstellten und gruppenzugehörigen Quizze
    > Zuordnung der erreichten Punktzahlen zu Usernamen
    > Ordnung der Anzeige nach Quizzen
- Quiz
    > Erstellung von Freitext-Fragen
    > Erstellung von DropDown-Fragen

> Multiple-Choice-Fragen
> Angabe der korrekten Antworten
> Speicherung des Quiz
> Aufruf des Quiz seitens des Erstellers und der zugehörigen Gruppe
> Überprüfung der Korrektheit der gegebenen Antworten
> Vergabe und Speicherung der erreichten Punktzahlen

- Anmeldung:
> Möglichkeit zur Registrierung
> Angabe Username
> Angabe E-Mail-Adresse
> Angabe Passwort
> Anschließende Speicherung in der Datenbank
> Anmeldemöglichkeit

2.4 Welche Beziehungen gibt es zwischen den Objekten?

Festlegung der Abhängigkeiten:

User ⇄ Gruppen
- Ein User kann entweder einer oder keiner Gruppe zugehörig sein
- Eine Gruppe kann mehrere User enthalten, muss jedoch mindestens einen User beinhalten

User ⇄ Rangliste
- Ein User bekommt genau eine Rangliste, abhängig von den selbst erstellten Quizzen und der Gruppenzugehörigkeit
- Eine Rangliste kann identisch für mehrere User sein, solange sie derselben Gruppe angehören

User ⇄ Quiz
- Ein User kann beliebig viele Quizze erstellen
- Das erstellte Quiz wird genau einem erstellenden User zugeordnet

User ⇄ Anmeldung
- Ein User benötigt eine Anmeldung
- Eine Anmeldung kann nur von einem User durchgeführt werden

Quiz ⇄ Gruppen
- Ein Quiz wird einer Gruppe zugeordnet
- Eine Gruppe kann mehrere Quizze enthalten

Quiz ⇄ Rangliste
- Ein Quiz kann in mehreren Ranglisten vorkommen, da diese für jeden innerhalb der Gruppe ersichtlich ist
- Eine Rangliste kann keine Quizze enthalten, eines oder mehreren

Gruppen ⇄ Rangliste
- Eine Rangliste kann eine, mehrere oder keine Gruppen enthalten
- Einer Gruppe werden jedoch nur eine Rangliste zugeordnet

3. Versionsverwaltung über Github

3.1 Anlegung der Organisation und Repository

Für die Versionsverwaltung des Projekts wird Github in der Community-Version verwendet. Im ersten Schritt stand die Erstellung einer Organisation, dieser Organisation wurden als Mitglieder alle vier Teammitglieder hinzugefügt. Dabei wurde die rolle Owner gewählt, das bedeutet, dass jedes Teammitglied mit allen Rechten ausgestattet ist. Der Product Owner wurde dagegen als Outside Collaborator mit Lese- und Schreibrechten eingeladen. Dabei können Dateien problemlos angepasst, hochgeladen und heruntergeladen werden und die Gruppengröße von 5 wird nicht überschritten. Denn sobald die Gruppe die Anzahl von 5 übersteigt, sollte Github in der Pro-Version genutzt werden.
Nun da die Zugriffsrechte eindeutig zugeordnet waren, wurde das Repository angelegt, in welchem das gesamte Projekt umgesetzt werden wird.

3.2 Umsetzung des 4-Augen-Prinzips

Laut Vorgabe des Product Owners wird jede Aufgabe auf Grundlage des 4-Augen-Prinzips umgesetzt. Dies bedeutet, dass jedes Teammitglied die jeweiligen in den Sprint festgelegten Aufgaben immer in Absprache mit mindestens einem anderen Teammitglied ablegt. Dabei wurde für die organisatorischen, planerischen Aufgaben sowie die Erstellung des ersten groben

Prototypen noch kein technisches Werkzeug benutzt. Das Prinzip wurde mittels Absprachen umgesetzt.
Für die konkrete Implementierung der verteilten Aufgabenbereiche wird der Review-Prozess genutzt. Das bedeutet, dass ein Teammitglied, das auf seiner oder ihrer eigenen Branch arbeitet einen Merge-Request stellen muss. Dieser Merge-Request muss von mindestens einem anderen Teammitglied bestätigt werden, bevor die Neuerung mit in das Haupt-Repository eingebunden werden können. Sollte es noch Anpassungen geben, die vorher getätigt werden sollen, wird der Merge-Request nicht bestätigt, sondern lediglich kommentiert. Der Request wird erst zugelassen, sobald die benötigten Anpassungen umgesetzt worden sind

3.3 Arbeit mit Branches

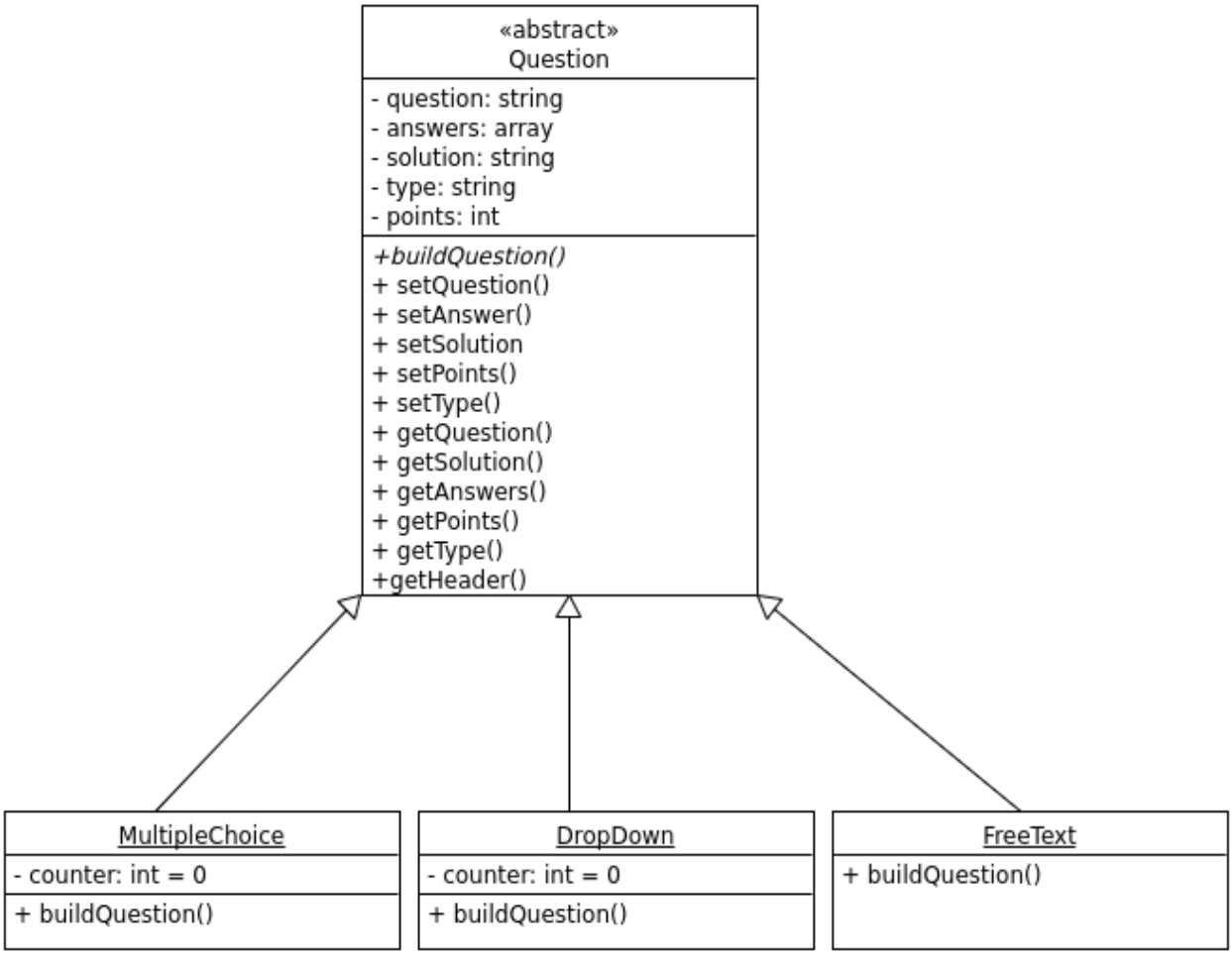
Alle organisatorischen Dokumente, gemeinsam Erarbeitetes sowie der erste Prototyp werden direkt in das main beziehungsweise Haupt-Repository von BAQ geladen. Genauso wie die Ressourcen bezüglich Corporate Design und Logo. Für die konkrete Implementierung des Projekts werden von jedem Teammitglied eigene Branches genutzt. Dies bedeutet, dass man quasi auf einer Kopie des Repositorys arbeitet, dort eigene Dateien und Veränderungen tätigen kann, ohne den Arbeitsprozess anderer Teammitglieder zu unterbrechen oder zu stören.
Die Branches werden nach Fertigstellung der Aufgaben durch einen sogenannten Merge mit dem Haupt-Repository vereint. Dieses Zusammenführen, muss von einem anderen Teammitglied bestätigt werden (siehe Punkt 3.3). Erst nach der Bestätigung und Beseitigung eventueller Konflikte, werden die Zweige (Branches) zusammengeführt, sodass schlussendlich nur ein vollständiges Repository besteht, in welchem alle Implementierungen zu finden sind.

4. Generische PHP-Klassen

4.1 Idee hinter der Erstellung

Um eine einheitliche Struktur der zu erstellenden Quizze, der Fragen sowie des Aus- und Einlesens zu garantieren wurden PHP-Klassen erzeugt, die mehrfach eingesetzt werden können. Benötigt werden sie für die Erzeugung des Quiz-Builders sowie für die Lets-Quiz-Seite. Da entweder Fragetypen zur Einlesung der Daten angelegt oder bereits erstellte Fragen aus der Datenbank ausgelesen und dem jeweiligen User angezeigt werden.

4.2 Klassen-Diagramm und Strategy-Pattern



Da der Code weiterverwendet wird, wurde das Strategy-Pattern als Orientierung verwendet, um es möglichst übersichtlich zu gestalten. Wie im Klassen-Diagramm zu sehen ist, gibt es eine abstrakte Elternklasse, von der die konkreten Fragenklassen erben. Die Strategie oder auch Logik der Fragen wird in der abstrakten Klasse festgelegt. Dadurch, dass in Question, die Attribute festgelegt werden, können alle erbenden Klassen darauf zugreifen ohne diese selbst noch einmal anlegen zu müssen. Folglich wird

doppelter Code vermieden. Das bedeutet gleichzeitig, dass Getter- sowie Setter-Methoden lediglich einmal erstellt werden müssen, jedoch von allen Fragenklassen konfliktlos genutzt werden können. Die konkrete Strategie/Logik für jede einzelne Frage wird in den Unterklassen festgelegt, die von Question erben. Dabei muss die abstrakte Methode buildQuestion implementiert werden, da sich diese für jede Unterklasse unterscheidet.

4.3 Zweck der einzelnen Methoden

4.3.1 BuildQuestion

Die Methode buildQuestion() wird in der abstrakten Klassen Question lediglich deklariert und zwar als abstrakt. Da es sich um eine abstrakte Methode handelt, muss jede Unterklasse diese selbst implementieren.

Innerhalb dieser Methode wird HTML-Code generiert und zwar auf Basis der mitgegebenen Parameter. Einmal der Frage sowie den Antwortmöglichkeiten. Dieser HTML-Code dient dazu, die Frage passend zu dem jeweiligen Fragetypen darzustellen. Beispielsweise wird für die Freitext-Frage ein Eingabefeld generiert oder für die DropDown-Frage ein DropDown-Menü mit den jeweiligen Antwortmöglichkeiten darin.

Die Methode soll als Hilfsmittel für andere Teammitglieder fungieren, um nach dem Auslesen der Datenbank einfacher HTML-Code erstellen zu können. Dieser erstellte HTML-Code zeigt im Anschluss das erstellte beziehungsweise zu spielende Quiz des Users.

Um die Anwendung dieser Methode zu vereinfachen und darzustellen, wurde die Klassen ExampleBuild.php erstellt. Innerhalb dieser Klasse wird gezeigt, wie Fragen anhand der Methode erstellt und befüllt werden sowie wie es schlussendlich auf der Weibseite aussieht.

4.3.2 GetHeader

Bei der Methode getHeader() handelt es sich ebenfalls um eine Methode, die die Darstellung vereinfachen sollen. Dabei wird der Kopf der Frage anhand des Fragentyps genereriert. Der Fragentyp wird in einer Variable $type innerhalb des Konstruktors festgelegt. Sobald ein Objekt einer konkreten Fragenklasse erstellt wird, wird diese Variable befüllt.

Innerhalb der Methode wird anhand von $type unterschieden, welche Benutzeraufforderung über der jeweiligen Frage steht.

Dies wird ebenfalls innerhalb der Klasse ExampleBuild.php dargestellt, um die Nutzung zu veranschaulichen.

4.3.3 Getter- und Settermethoden

Für alle Attribute der Klasse Question wurden Getter- und Settermethoden festgelegt. Diese erleichtern das Einlesen für die Datenbank. Die Klassen sollen im JSON-Format hinterlegt werden. Daher erleichtert es die Arbeit, mit Objekten zu arbeiten und die Attribute anhand der gewünschten JSON-Datei anzulegen und zu befüllen.

4.4 Helper-Klasse

Innerhalb der Helper-Klassen befinden sich drei praktische Methoden, um das Design des Projektes zu vereinfachen. BuildButton(), buildHeader(), buildFooter() rufen jeweils eine tpl-Datei auf, in der der entsprechende HTML-Code für die gewünschten Elemente hinterlegt ist. Beispielsweise implementiert die Methode buildHeader() eine Navigationsleiste sowie das Logo von Build-A-Quiz. BuildButton() erzeugt einen Bestätigenbutton und buildFooter() erzeugt einen Footer, welcher auf die Datenschutzerklärung sowie das Impressum verlinkt.

Die Klasse implementiert statische Funktionen, damit keine Objekte der Klasse erzeugt werden müssen, um die Funktionen zu nutzen.

Die Notation lautet wie folgt, am Beispiel von buildHeader():

Helper::buildHeader();

Dies wird ebenfalls innerhalb von ExampleBuil.php genutzt und gezeigt.