Dokumentation

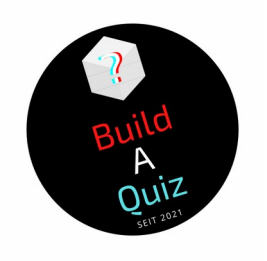
Projekt: Build-A-Quiz

Florian Vogl
Architekt

Aufgabengebiete

- Partielle Erstellung des Prototypen
- Festlegung der Grundlegenden Systemarchitektur
- Erstellung eines Objektorientierten Designs
- Erstellung der *„Build-Quiz“*-Seite

Gliederung



1. Planung des Projekts

Schon während der ersten Grobplanung wurde beschlossen, dass man die Website mit einer Datenbankanbindung ausstatten soll. Die so erstellten Quiz können sehr flexibel den verschiedenen Nutzern zugeordnet werden.

Im Laufe der Meetings mit Product Owner Stefan Müller wurde des Weiteren festgelegt, dass die Benutzer der Website sich in Gruppen organisieren können um somit gegeneinander „antreten“ zu können. Hierfür eignet sich eine Datenbank auch hervorragen.

1.1 Prototyp

Auf Basis der ersten Grobplanung konnte man nur daran gehen einen ersten Prototypen für das Projekt zu entwerfen. Dieser Prototyp erfüllt einzig den Zweck die grundsätzliche Idee der Seite mit dem Product Owner abstimmen zu können und hat noch keinerlei Funktion implementiert.

Sehr früh stand fest, dass man die Daten für die erstellten Quiz am besten im JSON-Format ablegen sollte, da dieses Format mittels PHP sehr einfach zu parsen ist.

1.1.1 Speicherung der Quizdaten

```
{
  [
      {
          "id":       1,
          "type":     "dropdown",
          "question": "Das ist die Frage",
          "answers":  ["Lösung 1", "Lösung 2", "Lösung 3"],
          "solution": 2,
          "points" :  1
      },
      {
          "id":       2,
          "type":     "multiplechoice",
          "question": "Das ist die Frage",
          "answers":  ["Lösung 1", "Lösung 2", "Lösung 3", "Lösung 4", "Lösung 5"],
          "solution": 2,
          "points" :  3
      },
      {
          "id":       3,
          "type":     "freetext",
          "question": "Das ist die Frage",
          "solution": "Lösung zum Vergleich",
          "points" :  2
      },
      {
          "id":       4,
          "type":     "multiplechoicema",
          "question": "Das ist die Frage",
          "answers":  ["Lösung 1", "Lösung 2", "Lösung 3", "Lösung 4", "Lösung 5"],
          "solution": [0, 2, 3],
          "points" :  3
      }
  ]
}
```

Listing 1: Prototyp für die Speicherung der Quizdaten im JSON-Format

Wie in Listing 1 zu sehen ist, wird für jedes Quiz ein mehrdimensionales Array angelegt. Dies bietet den Vorteil, dass man somit gleich sieht wie viel Fragen das jeweilige Quiz haben wird.

Innerhalb eines Array-Eintrags befinden sich dann die einzelnen Fragen. Diese haben eine *Id*, einen *Typ*, die *Frage*, die *Lösung*, die zu erreichenden *Punkte* und bei Multiple-Choice auch noch die *Antwortmöglichkeiten*.

Dadurch, dass JSON Plain-Text ist, kann es direkt an String in der Datenbank hinterlegt werden, was den Speicherbedarf auf ein Minimum beschränkt.

1.2 Verwendete Tools

Für den Entwicklungsprozess hat man sich darauf geeinigt *PHPstorm* von JetBrains zu nutzen. Die IDE ist für Studenten kostenlos erhältlich, sehr modern und bietet eine gute GIT-Integration.

Für die jeweiligen lokalen Tests am Server und der Datenbank wurde *LAMPP* verwendet. Dies ist auf allen gängigen Betriebssystem erhältlich und liefert für die Web-Entwicklung alles was man benötigt (einen Apache Server, eine MySQL-Datenbank, und einen PHP-Server).

Für die gemeinsame Entwicklung hat man sich dazu entschieden *GitHub* zu nutzen. Mittels Branches ist damit eine kollaborierte Entwicklung am einfachsten umsetzbar. Weitere Informationen zur Umsetzung mittels GitHub finden sich in der Dokumentation unseres GitHub-Admins *Theresa Alt*.

1.3 Nach der Entwicklung

Nach dem Entwicklungsprozesses soll das gesamte Projekt natürlich auch aus dem Internet aufrufbar sein.

Hierfür wurde die Website baq.byrdnest.de eingerichtet.

Nach Abschluss des Projekts wird die Seite über die URL erreichbar sein.

2. Grundlegende Architektur

Nachdem die grundsätzliche Projektplanung abgeschlossen war, konnte man sich nun an die Planung der Architektur, der Tools und der Funktionsweise der Website machen.

2.1 Technologien

Backend
Im Backend soll vorwiegend PHP eingesetzt werden um eine dynamische Website gewährleisten zu können. Des Weiteren eignet sich PHP auch sehr gut um die Quizdaten aus dem JSON-String zu parsen.

Die Daten (sowohl Userdaten, als auch Quizdaten) werden in einer MySQL-Datenbank abgelegt.

Für die temporäre Erstellung der Fragen in der Browser-Session eignet sich Javascript am besten.

Frontend
Beim Design hat man sich dafür entschieden Bootstrap zu verwenden. Somit können alle Entwickler unabhängig voneinander an verschiedenen Komponenten der Website arbeiten ohne sich Sorgen um das Layout machen zu müssen.

Bootstrap liefert außerdem schon Out-of-the-Box ein responsive Design mit zugehörigen JavaScript mit. Dadurch erschlägt man nebenbei auch den heutzutage gängigen Leitsatz *„Mobile first“*.

Durch Javascript kann die Interaktion zwischen Website und User deutlich besser gestaltet werden. So können beispielsweise Pop-Up-Fenster für eine erleichterte und intuitivere sorgen.

2.2 Funktionsweise von „Build A Quiz“

Die Grundfunktionalität der Seite ist schnell beschrieben:

„Der Nutzer soll sich ein Quiz zusammenstellen können“

Für dass stehen dem Nutzer verschiedene Frage-Typen zur Verfügung (Multiple-Choice, Freitext und Drop-Down), welche er nach Belieben in sein Quiz einbauen kann.

Nachdem man ein Quiz nicht nur alleine spielen möchte, gibt es die Möglichkeit Gruppen anzulegen um sich mit seinen Freunden messen zu können. Die Ergebnisse sind anschließend in einer Rangliste innerhalb der Gruppe einsehbar.

Um die Seite nutzen zu können muss man sich zunächst mit Benutzernamen und E-Mail-Adresse registrieren. Dies hat den weiteren Vorteil, dass man neben den Gruppen-Ranglisten auch eine persönliche Rangliste einsehen kann.

2.3 Erstellung OOD

Nachdem die Rahmenbedinugen abgesteckt wurden und die objektorientierte Analyse abgeschlossen war, gehörte zu meinen Aufgaben die Erstellung eines objektorientierten Designs. Das Ergebnis ist in Figure 1 zu sehen.

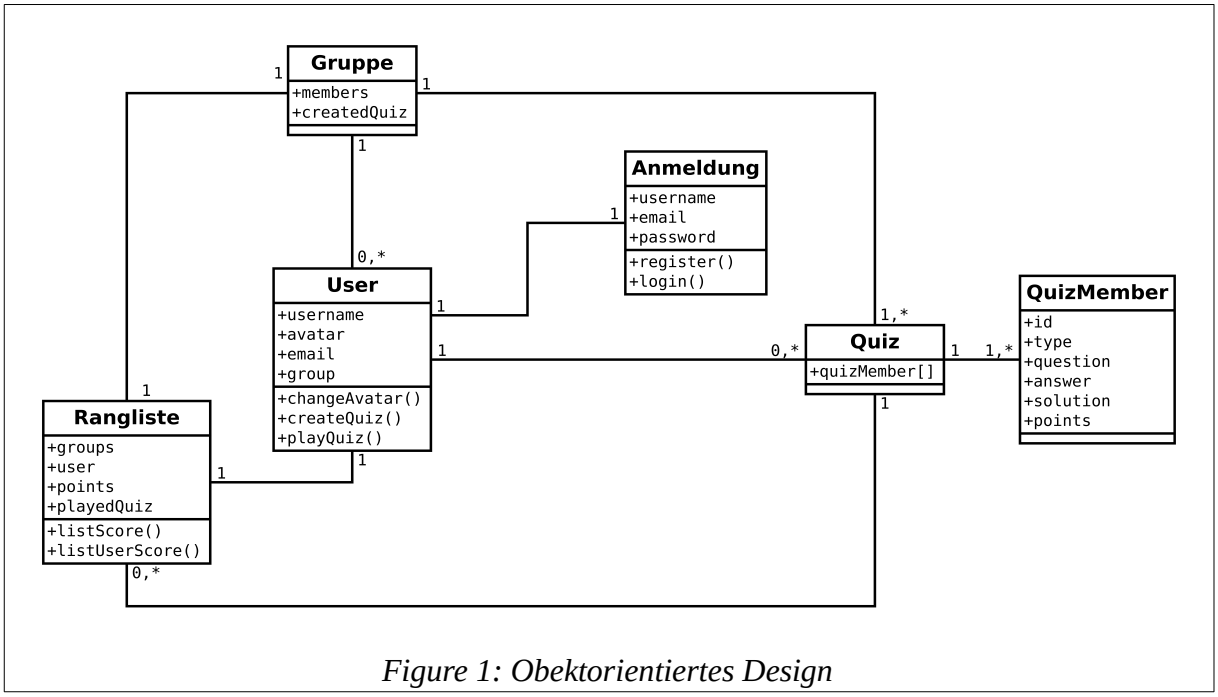


Figure 1: Obektorientiertes Design

2.4 *Komponenten der Website*

Build-A-Quiz besteht aus *sechs Hauptkomponenten* welche sich wie folgt aufteilen:

Landing-Page
Die erste Seite die der Nutzer sieht. Hier kann man sich mit Benutzername, E-Mail und Passwort registrieren bzw. anmelden falls man schon einen Account besitzt.

Beispielseite für ein Quiz
Nach dem Login wird der Nuzter zunächst auf diese geleitet um ihn kurz zu darzustellen, wie ein Quiz aussehen kann und welche verschieden Quiztypen es gibt.

Build Quiz
Das „Herzstück“ der Seite. Dort kann der User sich sein Quiz zusammenbauen und andere Teilnehmer hinzufügen, welche auch an dem Quiz teilnehmen sollen.

Profil Seite
Wie der Name schon vermuten lässt, kann der Benutzer hier sein Profil einsehen. Dort hat er dann unter anderem die Möglichkeit sein Profilbild zu ändern. Außerdem werden hier alle selbsterstellten bzw. spielbaren Quiz aufgelistet. Von hier kann der User dann ein Quiz starten oder sich die jeweilige Rangliste ansehen.

Quiz spielen und zugehörige Auswertung
Hat der User auf seiner Profil-Seite ein Quiz zum Spielen ausgewählt, wird dieses automatisch gestartet. Er beantwortet die Fragen und klickt auf Auswertung. Anschließend werden seine Antworten ausgewertet und ihm wird sein Ergebnis angezeigt, welches zugleich in der Datenbank hinterlegt wird.

Rangliste
Um die Rangliste für ein bestimmtes Quiz einsehen zu können, hat der User zwei Möglichkeiten. Entweder er wählt dies auf seiner Profil-Seite aus oder er lässt sich die Rangliste direkt nach dem absolvieren eines Quiz anzeigen. Anzumerken ist noch, dass jeweils nur der letzte Versuch in der Datenbank hinterlegt wird.

3. Programmierung der Build-Seite

Die Build-Seite gliedert sich in drei Teile auf. Znächst die build.php welche für die Darstellung der der Seite zuständig ist. Des weiteren eine build.js welcher für die Quizerstellung und die Übergabe an den Server zuständig ist. Die dritte Komponente bildet die send.php welche die vom Browser generierten Daten verarbeitet und in die Datenbank schreibt.

Es wurde bewusst darauf geachtet, dass das Quiz zunächst komplett in der Browser-Session des User erzeugt wird und anschließend komplett an den Server geschickt wird. So wird vermieden, dass im Falle eines Verbindungsabbruches des Users Fragmente eines Quiz an den Server übertragen werden und die Datenbank korrupt wird.

3.1 build.php

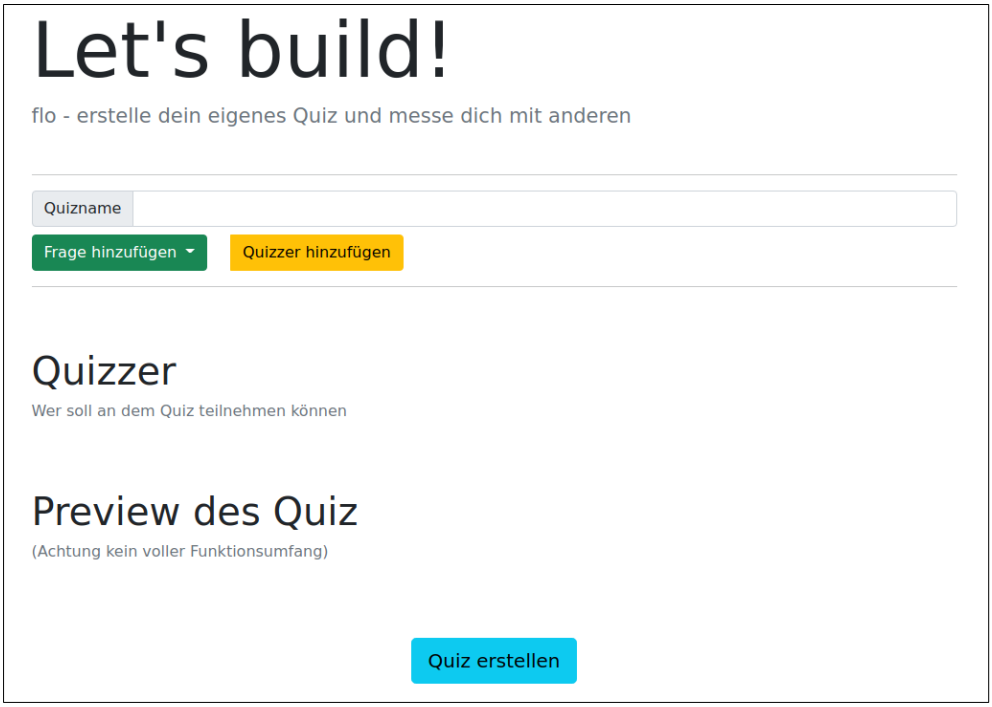


Figure 2: Darstellung von build.php

Die Build-Seite ist recht simpel gehalten und besteht nur aus einem Eingabefeld für den Quiznamen und den Buttons zum Hinzufügen von Fragen bzw. anderen Quizteilnehmern. Am Ende der Seite befindet sich noch ein Button zum finalen abschicken der Daten (siehe Figure 1).

3.2 build.js

Für die Interaktion mit dem User wurde SweetAlert2 verwendet um eine einheitliche Darstellung der Pop-Up-Fenster zu gewährleisten (siehe Figure 2).
Die Implementierung der Pop-Up-Fenster ist sehr *straight-forward*, weswegen darauf in dieser Dokumentation nicht genauer eingeganen wird. Die Implementierung kann im Github-Repo nachgelesen werden.

Für jeden Frage-Typen wurde eine eigene Funktion definiert, welche dem User nach und nach durch die Frage-Erstellung leitet.

Ist eine Frage fertig erstellt, wird diese in einem Javascript Array abgelegt und dem User im Preview-Bereich der Seite eine kurze Vorschau zur Verfügung gestellt.

Nach oben gibt es keine Begrenzungen wie viele Fragen der User pro Quiz erstellen kann.

Figure 3: Darstellung der Pop-Up-Fenster

Neben der Erzeugung der Pop-Up-Fenster übernimmt build.js insbesondere die Übermittlung der im Browser erzeugten Quizdaten an den Server. Hierfür wird ein *XMLHttpRequest* verwendet, welcher an send.php übertragen wird. Diesem Request werden die Quizdaten (Quiz-Teilnehmer, Quizname und die Fragen) hinzugefügt. Anschließend wird der Request dann an Server weitergeleitet und der User darüber informiert, dass sein Quiz nun erstellt und in der Datenbank abgelegt wurde.

3.2.1 Implementierung von send()

```
// global variables
let allQ = [];  // for all the questions
let allU = [];  // for all the quiz participant

// function to send the question to the server
function send() {

   // get information about quiz and players
   let players = allU.toString()
   let quizname = document.getElementById('quizname').value

   // generate new request
   let req = new XMLHttpRequest();

   // set target
   req.open("POST", 'php/send.php');

   // set additional data
   req.setRequestHeader("players", players);
   req.setRequestHeader('quizname', quizname);


   // check if quizname was given
   if (quizname) {

       // send request to server
       req.send(JSON.stringify(allQ));

       // inform the user
       swal.fire({
           icon: 'success',
           title: "Quiz " + quizname + " erfolgreich erstellt und in DB gespeichert",
           confirmButtonText: 'Alles klar!',
       }).then((result) => {
           if (result.isConfirmed) {
               document.getElementById('quizname').value = ''
               location.reload()
           }
       })
   } else {
       swal.fire({
           icon: 'warning',
           title: "Es wurde kein Quizname vergeben!"
       })
   }
}
```

Listing 2: funciton send() aus build.js

Für die Implementierung der *send()*-Funktion wurde zunächst ein Array `allQ[]` angelegt in welchen alle Fragen abgelegt werden. Die Quiz-Teilnehmer werden ebenfalls in einem Array (`allU[]`) abegelegt. Nachdem alle Daten zusammengetragen wurden, wird der XMLHttpRequest erstellt und mittels POST (so mit wird sichergestellt, dass die Daten auch verschlüsselt übermittelt werden) an den Server, im Speziellen an send.php übermittelt.

3.3 send.php

Nachdem die Verarbeitung am Client abgeschlossen wurde, müssen die Quizdaten nun noch am Server verarbeitet werden, sprich das Quiz muss in der Datenbank eingepflegt werden. Hierfür ist die send.php, welche die Daten vom Client empfängt verwantwortlich.

```php
// convert players to array
$players= explode(',',$players);
$quizzers = array();

// get players id from database
foreach ($players as $player){
    $query = "select user_id from users where username = '$player'";
    $result = $mysqli->query($query);
    $result = $result->fetch_assoc();

    // add player to quizzer
    $quizzers[] = $result['user_id'];
}

// create quiz in database
$newQuiz = "INSERT  INTO quiz(quiz_name, creator, quiz_json)
                    VALUES('$quizname', '$userId','$quizJSON')";
$mysqli->query($newQuiz);

// get quiz-id of the created quiz
$query = "select quiz_id from quiz where quiz_name = '$quizname'";
$result = $mysqli->query($query);
$result = $result->fetch_assoc();
$quiz_id = $result['quiz_id'];

// generate an entry in score for each quizzer
foreach ($quizzers as $quizzer){

    $newQuizzer = "INSERT  INTO score(user_id, quiz_id, points, counter)
                           VALUES($quizzer, $quiz_id, 0, 0)";
    $mysqli->query($newQuizzer);
}

// generate an entry in score for the creator
$mysqli->query("INSERT INTO score(user_id, quiz_id, points, counter)
                           VALUES($userId, $quiz_id, 0, 0)");
```

Listing 3: Auszug aus send.php

Zunächst wird für jeden Usernamen der vom Ersteller angegeben wurde die entsprechende User-ID aus der Datenbank geholt und in einem Array abgelegt. Im nächsten Schritt wird ein neuer Eintrag in der Quiz-Tabelle mit Quiznamen, Ersteller und des übermittelten JSON-Strings angelegt.

Um nun die Gruppenzugehörigkeit zu realisieren, wird für jeden Teilnehmer des Quiz ein Eintrag in der Score-Tabelle mit zugehöriger Quiz-ID angelegt und mit einem Score von 0 und einem Spielzähler von 0 angelegt.

Als letzes wird für den Ersteller ebenfalls ein solcher Eintrag angelegt.