



9. APRIL 2021

M151

DATENBANKEN IN WEB-APPLIKATION EINBINDEN

CHRISTOPHER OCONNOR

BBBADEN



1 INHALTSVERZEICHNIS

2	Aufgabenstellung.....	4
2.1	Wer wird Millionär im Fernsehen	4
2.2	Wer wird Millionär als Webapplikation	4
2.3	Anforderungen.....	4
2.3.1	Administration	4
2.3.2	GUI.....	4
2.3.3	Spiel	5
3	Projektplanung	6
3.1	Definition der Teilziele	6
3.1.1	Datenbank.....	6
3.1.2	Model Controller (CRUD).....	6
3.1.3	Routing.....	6
3.1.4	Admin-User Account.....	6
3.1.5	Spiel	6
3.1.6	Input Validation.....	6
3.1.7	Testing	6
3.2	Zeitplan.....	7
4	Analyse / Entscheidung.....	8
4.1	Dynamische Elemente.....	8
4.1.1	Tier 1 Presentation	8
4.1.2	Tier 2 Webserver	8
4.1.3	Tier 3 Application Server.....	8
4.2	Verwendete Technologien	8
4.2.1	Tier 1 Presentation	8
4.2.2	Tier 2 Webserver	8
4.2.3	Tier 3 Application Server.....	8
4.2.4	Tier 4 Data Server	8
4.2.5	Zusätzliche Technologien.....	8
5	Realisierung.....	9
5.1	Datenbank ERM	9
5.2	Migrations.....	9
5.3	Laravel Models.....	10
5.3.1	Property & Keytype	10
5.3.2	Fillable.....	10

5.3.3	Relations	10
5.3.4	Custom Get Method	11
5.3.5	Syntax	11
5.4	DB: SEED	11
5.4.1	Factory	11
5.4.2	Seeder	12
5.5	Model Controller (CRUD).....	13
5.5.1	Index	13
5.5.2	Create	13
5.5.3	Store	14
5.5.4	Show	14
5.5.5	Edit.....	14
5.5.6	Update	15
5.5.7	Destroy.....	15
5.5.8	Softdeletes	15
5.6	Routes.....	16
5.6.1	Get	16
5.6.2	Post.....	16
5.6.3	Return Funktion	16
5.6.4	Auth	16
5.6.5	Resource Models	17
5.6.6	Middleware	17
5.7	Blade Templating Engine	17
5.7.1	Template	17
5.7.2	Unterseiten	19
5.7.3	Blade PHP.....	19
5.8	Input Validation	21
5.9	Request Input Validation Backend	21
5.9.1	Request Validation Klasse.....	21
5.9.2	Request Validation Controller.....	22
5.9.3	Request Validation Failure Return	22
5.9.4	Custom Request Validation Return	23
5.9.5	Session Validation	23
5.10	Input Validation Frontend	23
5.11	Authentifikation	24
5.11.1	Controller	24

5.11.2	Request.....	24
5.11.3	Middleware.....	24
5.11.4	Blade Tag.....	24
5.12	Admin.....	25
5.12.1	Login	25
5.12.2	Register	25
5.13	Session.....	25
5.14	Spiel.....	26
5.14.1	Eingabe Benutzername.....	26
5.14.2	Auswahl der Kategorie.....	26
5.14.3	Frage.....	27
5.14.4	Richtige Antwort	27
5.14.5	Falsche Antwort	28
5.14.6	GameController.....	29
5.14.7	50/50 Joker	30
5.14.8	Beendung des Spiels.....	31
5.15	Highscores	31
5.16	Admin-Tool	32
5.16.1	Validation.....	32
5.16.2	Cookies.....	32
5.16.3	Kategorie.....	33
5.16.4	Fragen.....	35
5.16.5	Antworten.....	36
6	Testing.....	38
6.1	Unit-Testing	38
6.2	Selenium.....	38
6.3	Testfallspezifikation	39
6.4	Testprotokoll.....	46
7	Auswertung	51
7.1	Allgemein.....	51
7.2	Planungsdiskrepanz.....	51
7.3	Testing	51
8	Anhang.....	51

2 AUFGABENSTELLUNG

Ziel des Projektes ist es, eine abgewandelte Version des Quiz «Wer wird Millionär» zu implementieren. Dieses Quiz soll zu Lernzwecken und zur Unterhaltung gebraucht werden können.

2.1 WER WIRD MILLIONÄR IM FERNSEHEN

In jeder Spielrunde werden eine Frage und vier Antworten präsentiert, von denen nur eine korrekt ist. Für jede richtige Antwort erhält der Kandidat einen Geldbetrag, bei einer falschen Antwort verliert er alles, was er sich erspielt hat und scheidet aus. Der Kandidat kann bei jeder Frage bevor er die Antwort gegeben hat das Spiel selbstständig beenden und den Gewinn kassieren. In der Sendung gibt es drei Joker, die jeweils nur einmal eingelöst werden dürfen: Der Kandidat darf jemanden anrufen, das Publikum befragen oder zwei falsche Antworten ausblenden.

2.2 WER WIRD MILLIONÄR ALS WEBAPPLIKATION

In der Webapplikation soll die Frageform - eine Frage mit vier Antwortmöglichkeiten, von denen nur eine korrekt ist - beibehalten werden. Für jede korrekt beantwortete Frage erhält der Spieler 30 Punkte. Damit die Recherchemöglichkeit eingeschränkt wird, soll die Zeit zwischen dem Start des Quiz und dem Abschluss gemessen werden. Der 50:50-Joker soll dem Spieler einmal zur Verfügung stehen. Benutzt er ihn, werden zwei falsche Antworten ausgeblendet. Die Fragen werden von einem Administrator eingepflegt und unterhalten. Als Ansporn gibt es eine Highscoreliste, die einen Vergleich der Resultate ermöglicht.

2.3 ANFORDERUNGEN

2.3.1 Administration

- Der Administrator muss sich durch Username und Passwort authentifizieren.
- Der Administrator muss Fragen mit Antworten anlegen, ändern und löschen können.
- Der Administrator muss Kategorien anlegen und jede Frage einer Kategorie zuordnen können.
- Der Administrator kann einzelne Einträge der Highscoreliste löschen.

2.3.2 GUI

- Als Client dient ein Webbrowser.
- Zu jeder Frage wird gespeichert, wie oft sie richtig und wie oft sie falsch beantwortet worden ist.
- Zu jeder Frage wird beim Quiz prozentual angezeigt, wie oft sie richtig beantwortet wurde.
- Der Spieler sieht zu jeder Zeit seine aktuelle Punktzahl.
- Der Spieler sieht zu jeder Zeit, ob er den 50:50-Joker noch einsetzen kann.
- Der 50:50 Joker markiert zwei falsche Antworten und macht sie unauswählbar.
- War die vom Spieler gewählte Antwort richtig, so wird dies dem Spieler mitgeteilt und weiter zur nächsten Frage gegangen.
- War die vom Spieler gewählte Antwort falsch, so wird dies dem Spieler mitgeteilt und die richtige Antwort dargestellt.
- Hat der Spieler eine falsche Antwort eingegeben, so bricht das Quiz ab, der Versuch wird mit 0 Punkten gewertet und erscheint nicht auf der Highscoreliste, ansonsten schon.
- Der Spieler muss seinen Namen eingeben können, mit dem er auf der Highscoreliste erscheint.

- In der Highscoreliste werden folgende Daten aufgeführt:
 - Rang
 - Gewichtete Punkte: Punktzahl geteilt durch die Dauer des Quiz in Sekunden
 - Name des Spielers
 - Zeitpunkt des Spiels
 - Anzahl Punkte
 - Dauer des Quiz
 - Gewählte Kategorien
- Die Highscoreliste wird nach Rang, der durch die gewichteten Punkte bestimmt wird, aufsteigend sortiert.

2.3.3 Spiel

- Der Spieler kann die Kategorien wählen, aus denen die Fragen zufällig ausgewählt werden.
- Die Fragen mit den Antworten werden dem Spieler nacheinander präsentiert.
- Keine Frage soll während eines Spieles mehr als einmal gestellt werden.
- Zu jeder Frage gibt es vier Antworten: Eine korrekte und drei falsche.
- Der Spieler kann entweder eine Antwort auswählen oder aufhören und seinen Gewinn realisieren.
- Jede korrekt gewählte Antwort gibt 30 Punkte auf das Spielerkonto.
- Das Spiel soll mit einer spielbaren Anzahl Fragen gefüllt werden.
- Die Zeit zwischen dem Start des Quiz und dem Aufhören soll gemessen werden.
- Einfache Formulareingaben, wie leere Textfelder etc., sollen auf Client- und Serverseite geprüft werden.
- Die Wahl der Datenbank steht Ihnen frei (objektorientierte DB, NoSQL-DB, relationale DB mit stored Procedures und referentielle Integrität). Ein relationales Datenbankschema liegt als Empfehlung (Moodle M151_Daten.zip) bei. Es steht Ihnen frei dieses zu nutzen.
- Es soll eine Datenbankanbindung verwendet werden, die möglichst unabhängig vom tatsächlich eingesetzten Produkt ist.
- Transaktionsmanagement ist einzusetzen.
- Sicherheitsaspekte werden umgesetzt.
- Die Applikation soll als eine sessionbasierte professionelle 4-Tier Architektur implementiert werden. Dies bedeutet:
 - Webserver Layer: Templating System muss eingesetzt werden. Ein externes Framework bzw. eine Komponentenbibliothek werden eingebunden. Einsatz von HTML5 und CSS.
 - Business Logic: Einziges Tier, der eine direkte Verbindung zur Datenbank hat.

3 PROJEKTPLANUNG

3.1 DEFINITION DER TEILZIELE

Das Projekt wird in mehrere kleinen Teilziele aufgeteilt. Dadurch wird die Planung einfacher und die Teilziele sind überprüfbar und können somit abgeschlossen werden. Folgend werden die definierten Teilziele stichwortartig aufgelistet.

3.1.1 Datenbank

- ERM
- Migration
- Models generieren
- DB: Seed

3.1.2 Model Controller (CRUD)

- CRUD erstellen für alle Models
- UI Implementierung für die Bearbeitung

3.1.3 Routing

3.1.4 Admin Account

- Login / Logout
- Neuer Admin erstellen
- Authentifikation

3.1.5 Spiel

- Sessionsaving: (Spielername, Kategorie, ...)
- 50:50 Joker
- Fragen / Antworten
- Auswertung

3.1.6 Input Validation

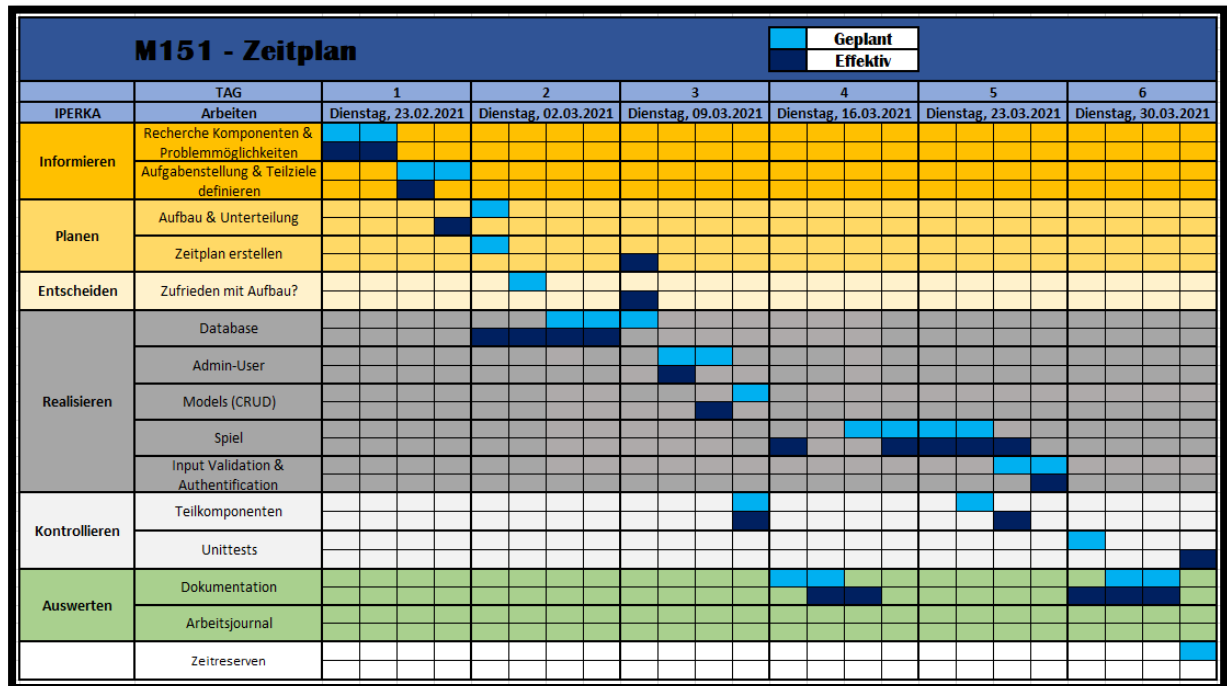
- Front-End
- Back-End

3.1.7 Testing

- Unittests
- Selenium

3.2 ZEITPLAN

Die Planung stellt ein wichtiges Hilfsmittel dar und wurde deshalb äusserst sorgfältig erstellt. Hier kann der Projektablauf mit der verfügbaren Zeit überprüft werden. Gegebenenfalls können Zeitknappheiten frühzeitig erkannt und angegangen werden.



4 ANALYSE / ENTSCHEIDUNG

4.1 DYNAMISCHE ELEMENTE

4.1.1 Tier 1 Presentation

Im Tier 1 werden Dynamische Informationen mithilfe von Blade-Templates ermöglicht. Zum Beispiel die Annotation `@auth` macht, dass nur ein eingeloggtter User nachfolgenden Inhalt sehen kann. PHP ermöglicht zusätzlich gewisse Elemente ein- oder auszublenden oder das Styling zu verändern.

4.1.2 Tier 2 Webserver

Auf der 2. Stufe wird auf Anfragen geantwortet. Entsprechend können verschiedene Contents zurückgegeben werden. Beispielsweise wenn eine Request-Validierung fehlgeschlagen ist.

4.1.3 Tier 3 Application Server

Mittels der Businesslogik wird das Spiel gesteuert und entsprechend unterschiedlicher Content zurückgegeben.

4.2 VERWENDETE TECHNOLOGIEN

4.2.1 Tier 1 Presentation

Blade ist eine einfache, aber dennoch starke Template-Engine welche in Laravel enthalten ist. Für das Styling wird zusätzlich Bootstrap verwendet und ergänzt mit eigenen CSS-Styles. PHP kommt hier auch zum Einsatz sowie JS/jQuery.

4.2.2 Tier 2 Webserver

Routes werden in Laravel im File `/routes/web.php` definiert. Anfragen können geprüft und zum Entsprechenden Controller weitergeleitet werden.

4.2.3 Tier 3 Application Server

Die Geschäftslogik findet in mehrere Controller statt. Typischerweise für jedes Model ein separater Controller für CRUD und zusätzliche für die Game-Logik.

4.2.4 Tier 4 Data Server

Für den Data Server wird eine MySQL Datenbank verwendet. Als ORM wird von Laravel Eloquent verwendet. Laravel beinhaltet auch einen eigenen DB:Seeder.

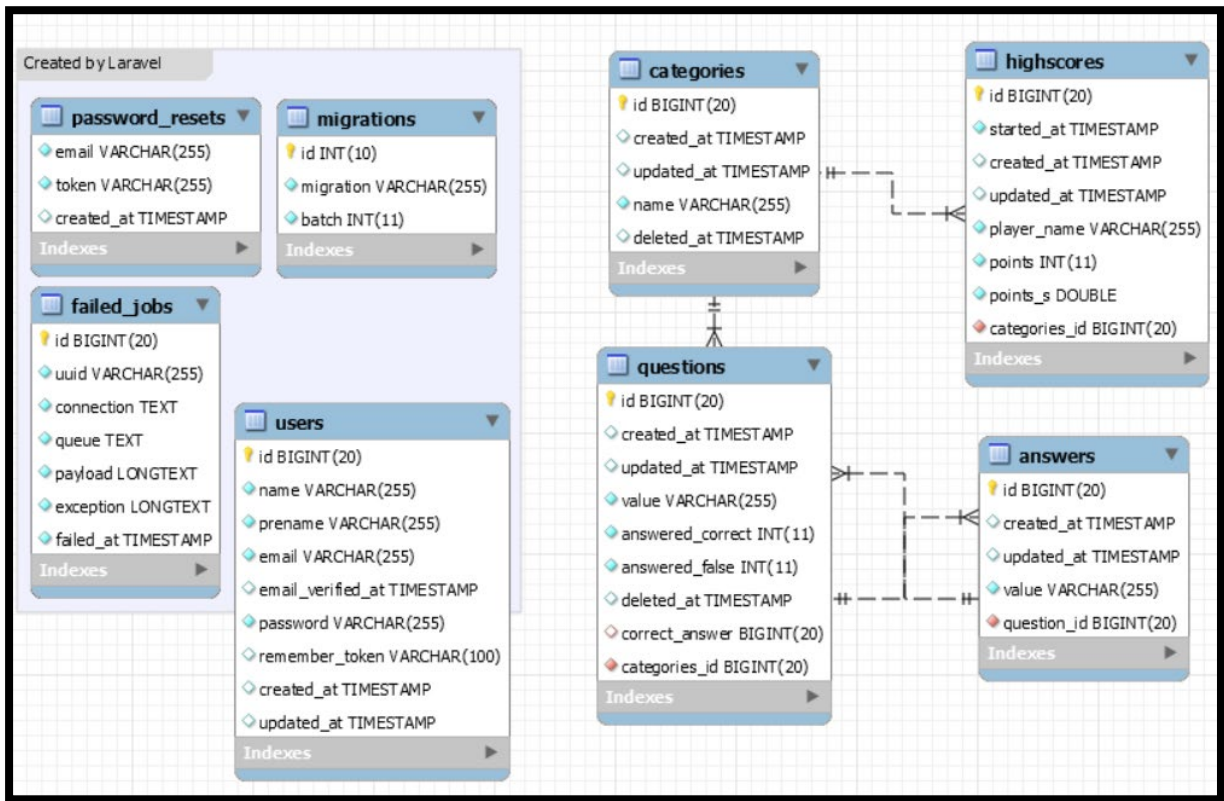
4.2.5 Zusätzliche Technologien

- Composer – Dependency Manager für PHP
- PHP Artisan – Laravel Konsole

5 REALISIERUNG

5.1 DATENBANK ERM

Folgendes ERM hat sich aus den Use-Cases und den Anforderungen ergeben. Erstellt wurde es in MySQL-Workbench.



5.2 MIGRATIONS

Die Datenbank könnte man durch das «Forward Engineer» erstellen. Ich habe mich dazu entschieden eigene Migrations in Laravel zu erstellen. Dadurch kann beim Testen die komplette Datenbank mittels eines Konsolenbefehls neu erzeugt werden innerhalb der IDE.

Das erstellte ERM gab hierzu die perfekte Vorlage. Für jede Tabelle wird eine eigene Migration erstellt. «php artisan make:migration create_questions_table»

```
public function up()
{
    Schema::create('questions', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->timestamps();
        $table->string('value', 255);
        $table->integer('answered_correct')->default(0);
        $table->integer('answered_false')->default(0);
        $table->softDeletes();

        $table->unsignedBigInteger('correct_answer')->nullable();
        $table->unsignedBigInteger('categories_id');

        $table->foreign('correct_answer')->references('id')->on('answers');
        $table->foreign('categories_id')->references('id')->on('categories');
    });
}
```

5.3 LARAVEL MODELS

Die Models können direkt von der Datenbank aus generiert werden. Dazu benötigt Composer eine externe Library. Diese kann dem Projekt mit dem nachfolgenden Befehl hinzugefügt werden:
«composer require krlove/eloquent-model-generator --dev»

Um die Models zu generieren benötigt man dann folgenden Befehl:
«php artisan krlove:generate:model Question --table-name=questions»

Wichtig! Reine N:M Verbindungstabellen müssen nicht in Laravel als Model implementiert werden. Diese kann man ignorieren.

5.3.1 Property & Keytype

```
* @property integer $id
* @property integer $correct_answer
*/
protected $keyType = 'integer';
```

Das Property beschreibt den Datentyp der Attribute und der Keytype den Datentyp der ID.

5.3.2 Fillable

```
protected $fillable = ['categories_id', 'correct_answer', 'created_at', 'updated_at', 'value'];
```

Im Array «Fillable» werden alle zugänglichen Attribute definiert. Für Attribute, welche nicht zugänglich sein sollen z.B. das Passwort wird noch ein zusätzlicher Eintrag im Array «\$hidden» erstellt.

```
protected $hidden = ['password', 'remember_token',];
```

5.3.3 Relations

Beziehungen müssen in beiden Models definiert werden.

```
class Question extends Model
{
    public function category()
    {
        return $this->belongsTo(related: 'App\Models\Category', foreignKey: 'categories_id');
    }
}
```

```
class Category extends Model
{
    public function questions()
    {
        return $this->hasMany(related: 'App\Models\Question', foreignKey: 'categories_id');
    }
}
```

Die Relations werden beim Generieren miterzeugt. Laravel kann alle möglichen Formen von Relations nachbilden und viele mehr! Beispielsweise «has-through». Diese müssen aber manuell entsprechend hinzugefügt werden.

5.3.4 Custom Get Method

Möchte man zusätzliche Methoden innerhalb des Models erstellen kann dies beispielsweise so aussehen:

```
public function getNotValidAttribute(): bool
{
    if(count($this->questions) < 3) return true;
    foreach($this->questions as $q) {
        if(count($q->answers) != 4 || is_null($q->c_answer)) return true;
    }
    return false;
}
```

Möchte man diese aufrufen wäre dies die Syntax: «\$model->not_valid»

5.3.5 Syntax

Als Beispiel wird ein Objekt des Models Question verwendet. Möchte man ein Wert von einem Attribut haben wäre die Syntax: «\$question->name». Wenn man nun von der Question alle Answers haben möchte, kann man dies so erreichen: «\$question->answers».

5.4 DB: SEED

Um die Datenbank mit Dummy Daten zu füllen, benötigt man in Laravel sogenannte Factories und Seeders. Diese können mit dem Befehl: «php artisan migrate:fresh --seed» ausgeführt werden.

5.4.1 Factory

In der Factory der jeweiligen Datenbanktabelle respektive des daraus entstandenen Models wird definiert was in welcher Spalte für ein zufälliger Wert generiert werden soll. Für die Factory für das Model Question sieht das folglich so aus:

```
protected $model = Question::class;

public function definition()
{
    return [
        'value' => $this->faker->sentence,
        'answered_correct' => $this->faker->numberBetween(0, 10),
        'answered_false' => $this->faker->numberBetween(0, 10),
    ];
}
```

Ab Laravel Version 8.x benötigt die Factory das entsprechende Model dazu muss folgender Eintrag im Model hinzugefügt werden:

```
class Question extends Model
{
    use HasFactory;
```

5.4.2 Seeder

Der Seeder ist dafür zuständig, wie oft welches Model erstellt werden soll oder falls statische Daten hinzugefügt werden sollen. Nachfolgende Abbildung zeigt das DatabaseSeeder.php file. In diesem werden alle anderen Seeder aufgerufen.

```
public function run()
{
    $this->call([
        DefaultAdminAccSeeder::class,
        GameSeeder::class,
        HighscoreSeeder::class,
    ]);
}
```

5.4.2.1 Statischer Seeder «DefaultAdminAcc»

In diesem Seeder wird ein Eintrag mit immer denselben Daten der DB hinzugefügt:

```
public function run()
{
    $users = [
        [
            'name'      => 'admin',
            'prename'   => 'admin',
            'email'     => 'admin@admin.ch',
            'password'  => Hash::make('123'),
        ]
    ];
    DB::table('users')->insert($users);
}
```

5.4.2.2 Dynamischer Seeder «GameSeeder»:

In diesem Seeder wird gleich die Kategorie, die dazugehörigen Fragen und Antworten erstellt.

```
public function run()
{
    $categories = Category::factory()->count( count: 8)->has(
        Question::factory()->count( count: 10)->has(
            Answer::factory()->count( count: 4)
        )
    )->create();
}
```

5.5 MODEL CONTROLLER (CRUD)

Die CRUD Controller können in Laravel sehr einfach generiert werden: «php artisan make:controller QuestionController --resource». Resource steht dafür, dass der Controller einem Model angehört und entsprechend die Methoden index, create, store, show, edit, update und destroy benötigt werden.

5.5.1 Index

```
public function index()
{
    return view( view: 'highscore.index', [
        'list' => Highscore::all()->where( key: 'points', operator: '>', value: 0)->sortByDesc( callback: 'points_s'),
    ]);
}
```

In der Index Methode wird definiert welche View dargestellt werden soll wenn man die Index URL des Models aufruft. Für das Model Question wäre dies beispielsweise: «m151/question/». In den meisten Fällen wäre dies eine Auflistung der vorhandenen Models wie bei der Highscoreliste.

5.5.2 Create

Create gibt die View zurück, mit welcher ein neues Objekt erstellt werden kann. In diesem Projekt wird dies für Category, Question und Answer an einem Ort getätigt und deswegen nicht relevant.

5.5.3 Store

Die Store Methode wird verwendet, um ein neues Model zu erstellen aus der Create View.

```
<form action="{{ route('question.store') }}" method="Post">
  @csrf
  @method('POST')
```

Das ausgefüllte Formular wird als Request an die Methode übergeben. Es gibt verschiedene Möglichkeiten für die Validierung der Request. Dazu mehr in einem späteren Kapitel. Danach kann das Objekt erstellt werden.

```
$data = $validator->getData();

Question::create([
    'value' => $data['question'],
    'categories_id' => $data['catID'],
]);

return redirect()->route('models_index');
```

5.5.4 Show

Show gibt die View zurück, um das selektierte Model anzuschauen. Dies wird in diesem Projekt nicht verwendet für ein einzelnes Model sondern zusammengetragen für Category, Question und Answer.

5.5.5 Edit

```
public function edit($id)
{
    return view('models.question.edit', [
        'q' => Question::find($id),
    ]);
}
```

Edit gibt die View zurück, mit welcher die ausgewählte Question editiert werden kann. Für die URL: «m151/question/edit/{id}»

5.5.6 Update

Die Update Methode funktioniert gleich wie die Store Methode mit dem Unterschied, dass als Parameter zusätzlich zu der Request noch das zu verändernde Objekt mitübergeben wird.

```
<form action="{{ route('question.update', $q) }}" method="POST">
    @csrf
    @method('PUT')
```

Nach der Validierung kann dann das Objekt angepasst werden:

```
$data = $validator->getData();
$q->fill($data)->save();
```

Es gibt hier unterschiedliche Möglichkeiten. Wenn die `<inputs>` im Formular den gleichen Namen haben wie das entsprechende Attribut des Models, dann kann das Erstellen bzw. Updaten eines Models in einer Zeile geschehen. Sollte dies nicht der Fall sein wird die Syntax vom [Store](#) benötigt.

5.5.7 Destroy

```
public function destroy(Question $question)
{
    $question->delete();
    return redirect()->route('models_index');
}
```

Die selektierte Question wird zerstört. In Laravel wird dies mit einer Post Request ausgeführt.

5.5.8 Softdeletes

In Laravel gibt es die Möglichkeit Softdeletes zu verwenden. Damit diese gebraucht werden können muss das Model eine entsprechende Spalte namens «deleted_at» haben.

```
Schema::create('categories', function (Blueprint $table) {
    $table->bigIncrements('id');
    $table->timestamps();
    $table->string('name', 255);
    $table->softDeletes();
});
```

Im Model selbst muss zusätzlich folgende Zeile hinzugefügt werden:

```
class Category extends Model
{
    use SoftDeletes;
```


Wird die Methode «Model->delete()» ausgeführt erkennt Laravel selbständig ob das Objekt gelöscht oder ein Timestamp Eintrag erstellt werden soll. Bei allen Anfragen werden die Objekte mit einem «deleted_at» Timestamp nicht aufgeführt ausser dies wird explizit mitgeteilt:

```
$cat = Category::withTrashed()->find($id);
```

5.6 ROUTES

Das Herzstück von Laravel, die Routes werden im File «routes/web.php» definiert.

5.6.1 Get

```
Route::get( uri: 'highscores' , action: 'HighscoreController@index' )->name( name: 'highscores.index');
```

Die URL wird an die Methode innerhalb eines Controllers weitergeleitet. Innerhalb des Projekts ist es möglich diese Route mit dem vergebenen Namen anzusprechen «route('{name}')».

5.6.2 Post

```
Route::post( uri: 'setPlayerName' , action: 'SessionController@setPlayerName' )->name( name: 'playername.set' );
```

Die Post Request wird gleich definiert mit dem Unterschied, dass eine Request Variable initialisiert wird und der Methode übergeben wird sowie der Prefix «post» lautet.

5.6.3 Return Funktion

```
Route::get( uri: '/', function () { return view( view: 'index'); } )->name( name: 'index');
```

Innerhalb der Route ist es möglich Funktionen zu schreiben und gleich die Weiterleitung an eine entsprechende View zu erstellen.

5.6.4 Auth

```
/** Auth Routes (Login, Register, Logout) */
Auth::routes();
```

Die Auth Routes beinhaltet das Login, Register und Logout.

5.6.5 Resource Models

```
Route::resources([
    'answer' => 'AnswerController',
    'category' => 'CategoryController',
    'highscore' => 'HighscoreController',
    'question' => 'QuestionController',
]);
```

Die Resource Controller können mittels dieser Route hinzugefügt werden. Beinhaltet werden automatisch folgende Methoden: index, create, store, show, edit, update und destroy.

5.6.6 Middleware

Die Routes können gruppiert werden. Nachfolgend wird dieser Gruppe die Middleware «auth» hinzugefügt. Dies bedeutet, dass alle Routes innerhalb der Gruppierung nur von Eingeloggten Usern zugänglich sind.

```
Route::group(['middleware' => 'auth'], function() {
    Route::get('cat/restore/{id}', function() {
        // ...
    }->name('category.restore'));
});
```

In den Gruppierungen könnten zusätzlich auch Namespaces oder Präfixes hinzugefügt werden und ineinander verschachtelt werden.

5.7 BLADE TEMPLATING ENGINE

5.7.1 Template

Das Template File bildet die Vorlage für alle anderen Seiten. In diesem wird der <head> definiert und wo welcher Inhalt der Unterseiten eingefügt wird.

5.7.1.1 Head

Wie für alle Webseiten werden im Head alles nötigen Imports definiert wie CSS, JS, Fonts etc. und der Titel. Für Bootstrap auch wichtig, der Viewport.

```

<!doctype html>
<html lang="{{ app()->getLocale() }}">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- CSRF Token -->
  <meta name="csrf-token" content="{{ csrf_token() }}">

  <title>M151</title>

  <!-- Scripts -->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
  <script src="{{ asset('js/app.js') }}" defer></script>
  <script>
    var app_url = "{{ env('app_url') }}";
  </script>

  <!-- Fonts -->
  <link rel="dns-prefetch" href="//fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet">

  <!-- Font Awesome -->
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.13.0/css/all.css">

  <!-- Styles -->
  <link href="{{ asset('css/main.css') }}" rel="stylesheet">
  <link href="{{ asset('css/app.css') }}" rel="stylesheet">
</head>

```

5.7.1.2 Include

Mit einem Include können andere Vorlagen dem Template eingebunden werden. In diesem Projekt sind das spezifisch die Navbar und der Footer.

```

<body style="...">
  <div id="app">
    @include('layouts.inc.navbar')

    <div class="container-fluid mt-1">
      @yield('content')
    </div>

    @include('layouts.inc.footer')

  </div>
</body>

```

5.7.1.3 Yield

Im Yield wird definiert, dass dort der Content mit dem Namen «content» eingefügt wird.

5.7.2 Unterseiten

5.7.2.1 *Extends*

Bei den Unterseiten muss in der 1. Zeile definiert werden, dass diese Seite das Template verwendet mit dem Befehl «extends».

```
@extends('layouts.app')
```

5.7.2.2 *Section*

Damit auch der Inhalt am Richtigen Ort angezeigt wird, benötigt es die Definition an welche Sektion welcher Inhalt eingefügt wird. In diesem Projekt gibt es nur eine Sektion. Alles innerhalb des «content» wird im Template an die entsprechende Yield-Stelle während der Laufzeit eingefügt.

```
@section('content')  
    <h1>Title</h1>  
@endsection
```

5.7.3 Blade PHP

Blade bietet shortcuts für PHP-Kontrollstrukturen. Diese werden in diesem Projekt natürlich verwendet.

5.7.3.1 *IF*

```
@if (count($records) === 1)  
    I have one record!  
@elseif (count($records) > 1)  
    I have multiple records!  
@else  
    I don't have any records!  
@endif
```

5.7.3.2 *For*

```
@for ($i = 0; $i < 10; $i++)  
    The current value is {{ $i }}  
@endfor  
  
@foreach ($users as $user)  
    <p>This is user {{ $user->id }}</p>  
@endforeach
```

5.7.3.3 *Auth*

```
@auth
    // The user is authenticated...
@endauth
```

5.7.3.4 *Isset*

```
@isset($records)
    // $records is defined and is not null...
@endisset

@empty($records)
    // $records is "empty"...
@endempty
```

5.7.3.5 *Echo*

```
{{ $cat->id . ' ' . $cat->name }}
```

5.8 INPUT VALIDATION

Die Back-End Eingabeüberprüfung erfolgt entweder direkt im Controller (T3) oder kann als Request-Klasse direkt überprüft werden (T2). Front-End Überprüfung wird mit HTML und JS implementiert (T1).

5.9 REQUEST INPUT VALIDATION BACKEND

Es gibt primär zwei Möglichkeiten für die Validation einer Request in Laravel.

5.9.1 Request Validation Klasse

In dieser Variante wird eine neue Request Klasse erstellt, welche von Request vererbt.

```
class QuestionRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return auth()->user();
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'catID' => ['required'],
            'question' => ['required|unique:questions|max:255'],
        ];
    }

    /**
     * Get the validation error message.
     *
     * @return string[]
     */
    public function messages()
    {
        return [
            'required' => 'x',
        ];
    }
}
```

In der «authorize» Methode wird angegeben wer die nötigen Rechte für diese Request besitzt.

In den «rules» werden die Regeln für eine Erfolgreiche Request definiert.

Mit dem Attribut «unique» kann man angeben in welcher DB-Tabelle dieser Eintrag einmalig sein soll.

In den «messages» wird die Antwort definiert. Es gibt bereits Standard Antworten, diese können aber überschrieben werden. In diesem Beispiel wird, falls die Regel «required» fehlschlägt die Antwort «x» zurückgegeben.

5.9.2 Request Validation Controller

Erfolgt die Validation innerhalb des Controllers folgt sie dem gleichen Schema:

```
$validator = Validator::make($request->all(), [
    'catID'    => ['required'],
    'question' => ['required|unique:questions|max:255'],
], [
    'required' => 'x',
]);

if ($validator->fails()) {
    return redirect(url()->previous())
        ->withErrors($validator)
        ->withInput();
}
```

Im ersten Array werden die Regeln definiert und im zweiten Array die überschriebene Message Antwort.

Bei dieser Variante benötigt es explizit noch den Return wie im Gegensatz zur [Variante 1](#) wird dies nicht automatisch gesteuert.

5.9.3 Request Validation Failure Return

Wurde die Request als nicht valid gekennzeichnet wird das weitere Vorgehen abgebrochen und der User wird zurück auf die letzte Seite geschickt mit einem Message-Bag. Dieser beinhaltet die Fehlermeldung und die vom Benutzer angegebenen Informationen der Inputfields.

Damit diese Eingaben wieder an ihren entsprechenden Ort eingefügt werden und eine Fehlermeldung angegeben wird benötigt es im Formular die Tags «@error» und die Methode «old()».

```
<div class="form-group row">
    <label class="col-md-2 col-form-label">Question <span class="text-danger">*</span></label>
    <div class="col-md-4">
        <input type="text" class="form-control @error('question') is-invalid @enderror" name="question" value="{{ old('question') }}">
    </div>
    @error('question')
        <div class="offset-2 invalid-feedback">
            {{ $message }}
        </div>
    @enderror
</div>
```

5.9.4 Custom Request Validation Return

Bei komplexeren Validierungen wie in diesem Projekt, genügen die Validierungsregeln nicht. Beispiel: Innerhalb einer Frage müssen die Antworten einmalig sein aber sie können in mehreren verschiedenen Fragen vorkommen.

Es ist möglich einen Eintrag dem Message-Bag hinzuzufügen und diesen an die Ursprüngliche Seite zurückzugeben:

```
if (in_array(strtolower($data['value']), $qValues)) {  
    $validator->getMessageBag()->add( key: "value", message: "This Question already exists.");  
    return redirect(url()->previous())  
        ->withErrors($validator)  
        ->withInput();  
}
```

Hat der Message-Bag einen Eintrag kann man diesen mit @error({key}) im Formular abfangen. Innerhalb des Error-Tags genügt {{ \$message }} um die Message auszugeben für den definierten Error.

5.9.5 Session Validation

Für die Validation in Laravel genügt ein Array. Diesbezüglich ist es auch möglich das Session Objekt als Array zu Validieren.

```
$validator = Validator::make(session()->all(), [  
    'started_at' => ['required'],  
    'player_name' => ['required'],  
    'cat' => ['required'],  
    'points' => ['required'],  
]);  
  
if ($validator->fails()) { return false;}
```

5.10 INPUT VALIDATION FRONTEND

Für die Frontend Inputvalidation genügt in diesem Projekt der HTML-Tag required:

```
<input type="text" class="form-control @error('value') is-invalid @enderror" name="value" value="{{ $q->value }}" required>
```


5.11 AUTHENTIFIKATION

Die Authentifikation in Laravel kann an mehreren Orten Vorkommen:

5.11.1 Controller

Diese Methode gilt als überflüssig, wenn die Middleware gepflegt und gut unterhalten wird.

```
public function __construct()
{
    $this->middleware('auth');
}
```

5.11.2 Request

Bei einer Request kann man gleich überprüfen, ob der User berechtigt ist diese Request auszuführen:

```
class QuestionRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return auth()->user();
    }
}
```

5.11.3 Middleware

Mittels der Middleware in den Routes kann die Authentifikation durchgeführt werden.

```
Route::group(['middleware' => 'auth'], function() {
    Route::get( uri: 'cat/restore/{id}', action: 'CategoryController@restore')->name( name: 'category.restore');
```

5.11.4 Blade Tag

Im Frontend können Elemente ein- oder ausgeblendet werden.

```
@auth
<li class="nav-item ml-lg-4">
    <a class="nav-link @if(explode('.', Route::currentRouteName())[0] == 'models_index') active @endif"
      href="{{ route('models_index') }}">Admin-Tool</a>
</li>
@endauth
```

5.12 ADMIN

In diesem Projekt gibt es nur zwei verschiedene User. Der Gast, welcher kein eigener Account besitzt und der Admin, welcher eingeloggt sein muss. Diesbezüglich gibt es keine Rollen innerhalb der eingeloggten Benutzer. Wer sich einloggen kann, ist ein Admin.

5.12.1 Login

Die Login-Logik / Validierung wird von Laravel übernommen, das Design vom Frontend wurde selbst entwickelt.

5.12.2 Register

Das Registrieren wird von Laravel bereits geliefert. Lediglich die [Authentifikation](#) im Controller musste angepasst werden, dass nur Admins das Recht haben neue Benutzer zu erstellen und das Formular sehen können.

5.13 SESSION

Die Session wird im Session-Controller verwaltet. Es werden alle nötige Session variablen initialisiert damit das Spiel starten kann. Dazu gehört das Setzen des Spielernamens und der Ausgewählten Kategorie sowie das Zerstören der Session.

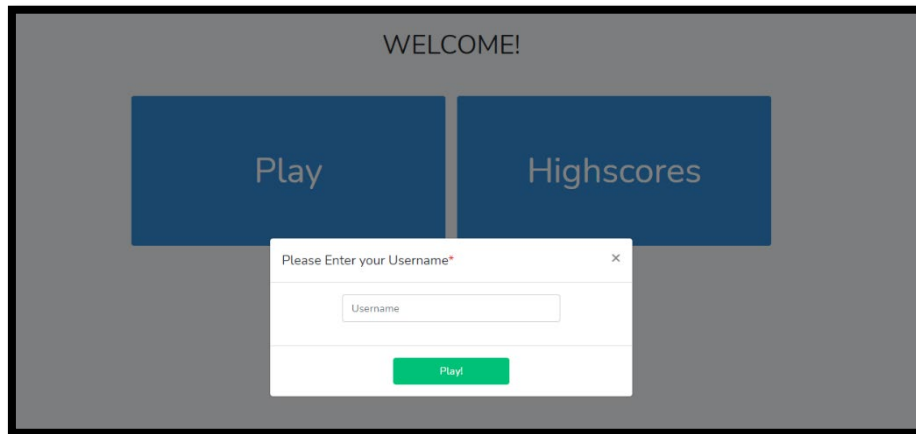
```
public function initGameSession()
{
    session([
        'q_completed' => array(),
        'points'       => 0,
        'started_at'  => Carbon::now()->format('Y-m-d H:i:s'),
        'joker'       => true,
        'activeQID'   => 0,
        'gameOver'    => false,
    ]);
}
```

5.14 SPIEL

Ich habe mich aus Sicherheitsgründen dafür entschieden, das gesamte Spiel unter einer URL durchzuführen. Entsprechend ist der Gamecontroller komplex und gross geworden mit der ganzen verbundenen Logik. Für Testzwecke können Sie sich als Administrator anmelden und die richtigen Antworten werden mit einem Mouseover angezeigt.

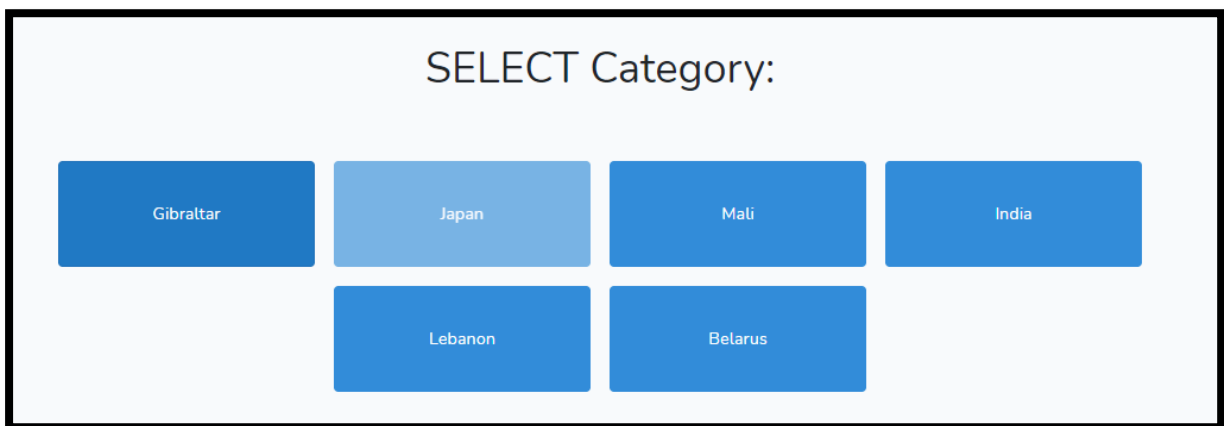
5.14.1 Eingabe Benutzername

Bevor das Spiel starten kann, wird der User aufgefordert einen Spielernamen einzutragen:



5.14.2 Auswahl der Kategorie

Besitzt die Session den Spielernamen dann kann eine Kategorie ausgewählt werden:



Wie gut ersichtlich, die Kategorien besitzen eine Validation. Hat beispielsweise eine Kategorie eine Frage ohne genügend Antworten oder keine Richtige Antwort, kann diese nicht zum Spielen ausgewählt werden.

5.14.3 Frage

Eine Zufällige Frage aus der Kategorie wird gestellt. Unterhalb der Frage ist ersichtlich wie Oft diese Richtig oder Falsch beantwortet worden ist.

A screenshot of a quiz question interface. At the top, the question text is "A rerum est sint est laborum optio.?". Below the question is a progress bar showing 50% green and 50% red. There are four blue buttons arranged in a 2x2 grid: "Shields-Lesch", "Cartwright-Fadel", "Adams-Crona", and "Crona PLC". Below the buttons are three buttons: "50 / 50 Joker" (green), "Points: 0" (blue), and "Finish Quiz" (blue). At the bottom is a "Question Progress:" bar with 10 segments, the first of which is green.

Die vier Antwortmöglichkeiten stehen zur Auswahl und können angeklickt werden. Der 50/50 Joker, falls dieser noch nicht verwendet wurde, wird grün dargestellt. Möchte man das Quiz beenden kann dies mit dem Button «Finish Quiz» getan werden. Der Question Progress zeigt alle noch verfügbaren Fragen an und falls man diverse Fragen bereits korrekt ausgefüllt hat, werden diese grün dargestellt.

5.14.4 Richtige Antwort

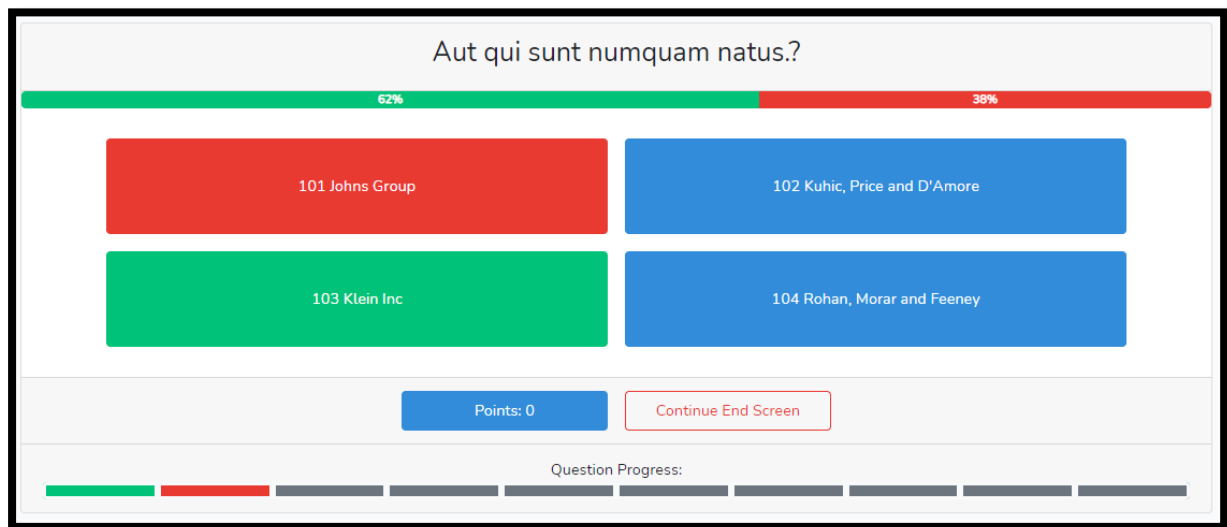
Wurde die Richtige Antwort ausgewählt wird dies dem Benutzer mitgeteilt:

A screenshot of a quiz question interface showing the correct answer. At the top, the question text is "Libero saepe officiis debitis ex quos tempora.?". Below the question is a progress bar showing 90% green and 10% red. There are four buttons arranged in a 2x2 grid: "85 Armstrong-Klocko", "86 Heaney, Batz and Ullrich", "87 Pollich Ltd", and "88 Marvin, Mraz and McGlynn". The button "88 Marvin, Mraz and McGlynn" is highlighted in green. Below the buttons are two buttons: "Points: 30" (blue) and "Continue Next Question" (green). At the bottom is a "Question Progress:" bar with 10 segments, the first of which is green.

Mit der Bestätigung «Continue Next Question» oder bei einem Page Reload (F5) wird zur nächsten zufälligen Frage weitergegangen.

5.14.5 Falsche Antwort

Wurde eine Falsche Antwort ausgewählt wird dies dem Benutzer mitgeteilt und die richtige Antwort eingeblendet:



Das Spiel wird abgebrochen und mit 0 Punkten gewertet. Mit der Bestätigung «Continue End Screen» oder Page Reload (F5) gelangt man zum «Game Over» Screen.

5.14.6 GameController

5.14.6.1 Logik der Darstellung

In der ersten Zeile werden alle Fragen der Kategorie ausgewählt, welche noch nicht beantwortet worden sind. Wurde eine Frage beantwortet wird dies in der Session gespeichert.

```
$questions = Question::all()->where( key: 'categories_id', session( key: 'cat'))->whereNotIn( key: 'id', session( key: 'q_completed'));

if(session( key: 'activeQID') == 0) {

    session()->forget( keys: 'errDisplayed');

    if($questions->isEmpty()) { return redirect()->route( route: 'play.end'); }

    $question = $questions->random();
    session(['activeQID' => $question->id]);

} else {

    if(array_key_exists( key: 'errDisplayed', session()->all()) && session( key: 'errDisplayed')) {

        if(array_key_exists( key: 'gameOver', session()->all()) && session( key: 'gameOver')) {
            return redirect()->route( route: 'play.over');
        }

        session()->forget( keys: 'errDisplayed');

        if($questions->isEmpty()) { return redirect()->route( route: 'play.end'); }

        $question = $questions->random();
        session(['activeQID' => $question->id]);

    } else {

        $question = Question::find(session( key: 'activeQID'));

    }

}
```

- Wurde die Frage bereits beantwortet, wurde diese Richtig oder Falsch beantwortet?
- Es gibt keine aktive Frage, nächste Frage wird zufällig ausgesucht.
- Gibt es bereits eine Aktive Frage, wird diese angezeigt. Zum Beispiel bei einem Reload (F5).

5.14.6.2 Antwort Validation

```
$data = $validator->getData();
$q = Question::find($data['question_id']);
$a = Answer::find($data['answer_id']);

if($q->c_answer->id == $a->id) {

    $validator->getMessageBag()->add( key: 'answer', message: 'c');

    $q->answer_count = true;
    $this->addPoints();

} else {

    $validator->getMessageBag()->add( key: 'answer', $a->id);

    $q->answer_count = false;
    $this->quizFailed();

}

$completed = session( key: 'q_completed');
array_push( $array: $completed, $q->id);

session(['q_completed' => $completed]);

return redirect(url()->previous())->withErrors($validator);
```

Es wird verglichen, ob die korrekte Antwort der Frage die gleiche ist wie die selektierte.

Ist dies der Fall wird eine Message an den Controller weitergeleitet in Form eines Message-Bags.

Stimmt die Antwort nicht wird die richtige Antwort dem Message-Bag hinzugefügt und weiter an den Logik-Controller übergeben.

Der daraus entstandene Message-Bag wird in der Methode «index» ausgewertet und die entsprechende Antwort dem Benutzer mitgeteilt in der View.

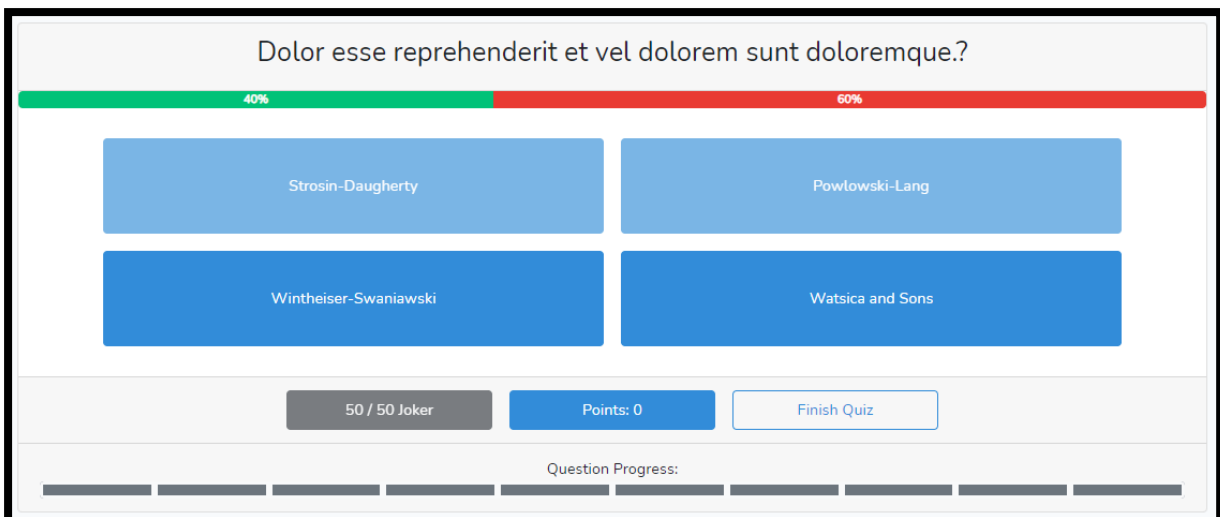
```

@error('answer')
@if($message == 'c')
    <span class="btn btn-primary btn-question-footer">Points: {{ session('points') }}</span>
    <a class="btn btn-outline-success btn-question-footer" href="{{ route('play.next', $q->id) }}">Continue Next Question</a>
@else
    <?php session(['gameOver' => true]) ?>
    <span class="btn btn-primary btn-question-footer">Points: {{ session('points') }}</span>
    <a class="btn btn-outline-danger btn-question-footer" href="{{ route('play.over') }}">Continue End Screen</a>
@endif
@else
    @if(session('joker'))
        <a class="btn btn-outline-success btn-question-footer" href="{{ route('joker') }}">50 / 50 Joker</a>
    @else
        <button class="btn btn-dark btn-question-footer" disabled>50 / 50 Joker</button>
    @endif
    <span class="btn btn-primary btn-question-footer">Points: {{ session('points') }}</span>
    <a href="{{ route('play.end') }}" class="btn btn-outline-primary btn-question-footer">Finish Quiz</a>
@enderror

```

5.14.7 50/50 Joker

Wird der 50/50 Joker betätigt werden zwei zufällig falsche Antworten ausgeblendet und der Joker für den Rest des Spiels deaktiviert.



Der Joker merkt sich die ausgeblendeten Antworten in der Session und bei einem Page Reload (F5) werden die genau gleichen Antwortmöglichkeiten ausgeblendet.

```

public function joker()
{
    $validator = Validator::make(session()->all(), [
        'joker' => ['required'],
        'activeQID' => ['required'],
    ]);

    if ($validator->fails()) {
        return redirect(url()->previous());
    }

    if(session( key: 'joker')) {
        session(['joker' => false]);

        $q = Question::find(session( key: 'activeQID'));

        $a1 = Answer::all()->where( key: 'question_id', $q->id)->whereNotIn( key: 'id', $q->e_answer->id)->random();
        $a2 = Answer::all()->where( key: 'question_id', $q->id)->whereNotIn( key: 'id', [$q->e_answer->id, $a1->id])->random();

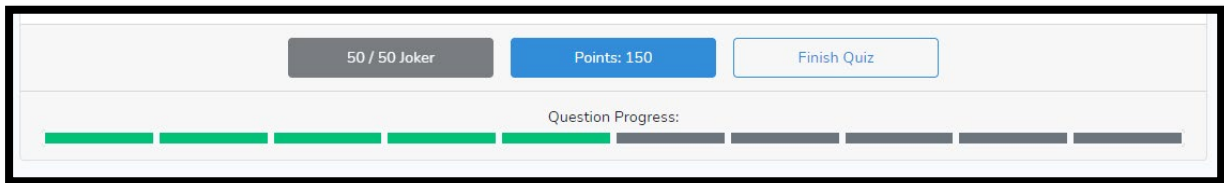
        session(['jokerAnswers' => [$a1->id, $a2->id]]);
    }

    return redirect(url()->previous());
}

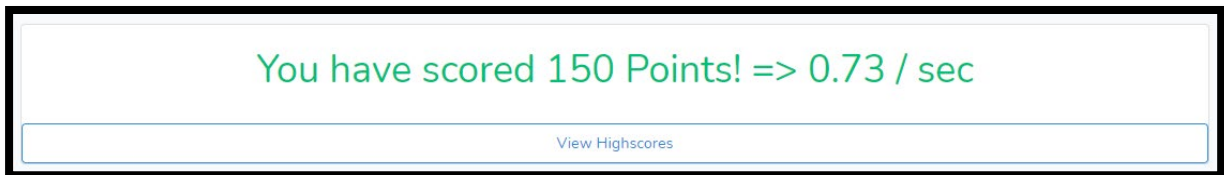
```

5.14.8 Beendung des Spiels

Wenn man genug vom Spiel hat, kann dies beendet werden mit dem Button «Finish Quiz»:



Der End-Screen erfolgt:





5.15 HIGHSCORES

In der Highscoreliste werden alle gültigen Versuche angezeigt und nach Punkte / Sekunde sortiert. Der eigene Versuch wird dabei zusätzlich hervorgehoben:

##	Name	▼ Points / s	Points	Duration	Category	Date
01	Doku :)	0.73	150	205s	Mali	14:24, 30/03/21
02	Alexandre Larkin	0.42	253	600s	Mali	17:06, 29/03/21

Der Administrator kann die Highscoreliste verwalten und einzelne Einträge löschen:

##	Name	▼ Points / s	Points	Duration	Category	Date	
01	Doku :)	0.73	150	205s	Mali	14:24, 30/03/21	
02	Alexandre Larkin	0.42	253	600s	Mali	17:06, 29/03/21	

5.16 ADMIN-TOOL

Der Administrator kann die Kategorien, Fragen und Antworten beliebig mutieren. Weil diese drei verschiedenen Objekte so stark abhängig voneinander sind, kann er alles gleichzeitig in einer View:

5.16.1 Validation

Durch die Validierung der Kategorie ist sofort ersichtlich in welcher Kategorie es einen Fehler gibt anhand der beiden roten Ausrufezeichen.

Die Fehlermeldungen sind dann entsprechend bei der jeweiligen Frage angezeigt:

5.16.2 Cookies

```
var initComplete = false;

function setCookie(cname, cvalue) {

    if (initComplete) {
        var cookie = getCookie("cat");

        if (cvalue == cookie) {
            // Reset Sub Cat
            if (cvalue.includes("-")) {
                var cat = cvalue.split("-");
                document.cookie = cname + "=" + cat[0] + ";";
            } else {
                // Reset Cat if only Cat was selected
                document.cookie = cname + "=" + 0 + ";";
            }
        } else {
            if (cookie.includes("-")) {
                var cat = cookie.split("-");

                if (cvalue == cat[0]) {
                    // Reset Cat if sub Cat was selected and Cat is pressed.
                    document.cookie = cname + "=" + 0 + ";";
                } else {
                    // Set new Cat if sub Cat was selected and other Cat is pressed.
                    document.cookie = cname + "=" + cvalue + ";";
                }
            } else {
                // Set new Cat if other Cat is pressed.
                document.cookie = cname + "=" + cvalue + ";";
            }
        }
    }
}
```

Damit diese Verschachtlung für Benutzer nicht mühsam wird, wird der aktuelle «Standort» in einem JS Cookie gespeichert. Dadurch ist es möglich das Admin-Tool zu verlassen und wieder beim zurückkommen

```
$(document).ready(function() {

    var str = getCookie("cat");

    if (str) {
        var x = str.split("-");
        $("#btn-" + x[0]).trigger('click');

        if (x.length > 1) {
            setTimeout(function () {
                $("#btn-" + str).trigger('click');
                initComplete = true; // complete Load if sub Cat was selected.
            }, 500);
        } else {
            initComplete = true; // complete Load is only Cat was selected.
        }
    } else {
        initComplete = true; // complete Load if none Selected.
    }
});
```

The screenshot shows a web application interface. At the top, there is a 'Create Category:' section with a text input field labeled 'Category Name' and a green plus icon button. Below this, the category '1. Gibraltar' is selected and expanded. It shows a 'Create Question:' section with a text input field labeled 'Question' and a green plus icon button. Underneath, there is a list of questions and answers. The first question is 'Q: Fugiat omnis numquam quo sint ipsum eum.' with a blue edit icon and a red delete icon. Below it are four answers: 'A: Ratke, Ernser and Dare', 'A: Kuhlman, Heathcote and Spinka', 'A: Schneider-Erdman' (which is highlighted with a green border), and 'A: Russel, Ankunding and Lind'. Each answer has a red delete icon. At the bottom, there is another question 'Q: Sit repudiandae assumenda enim aperiam nihil non ducimus.' with a blue edit icon and a red delete icon.

Die Kategorien können aufgeklappt werden und die Fragen werden angezeigt sowie die Möglichkeit eine neue hinzuzufügen. Die Fragen können wiederum auch aufgeklappt werden und die Antworten werden angezeigt.

5.16.3 Kategorie

Im Eingabefeld kann eine neue Kategorie erstellt werden mit dem grünen plus.

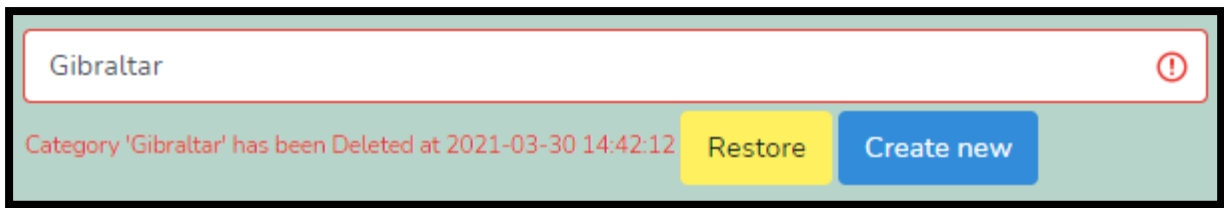
The screenshot shows the 'Create Category' section of the web application. It features a text input field labeled 'Category Name' and a green plus icon button. Below the input field, the category '1. Gibraltar' is listed, with a blue edit icon and a red delete icon to its right.

Bereits bestehende Kategorien können mittels des blauen Edit-Buttons editiert oder mittels des roten Lösch-Buttons gelöscht werden.

Die Eingabeüberprüfung für die Kategorie ist die ausführlichste. Es können keine Duplizierte Kategorien erstellt werden.

The screenshot shows a text input field with the value 'Gibraltar'. To the right of the input field is a red circle with an exclamation mark icon. Below the input field, a red error message is displayed: "'Gibraltar' is already used."

Zusätzlich wurde eine Kategorie gelöscht bspw. «Gibraltar» und wird neu erstellt passiert folgendes:

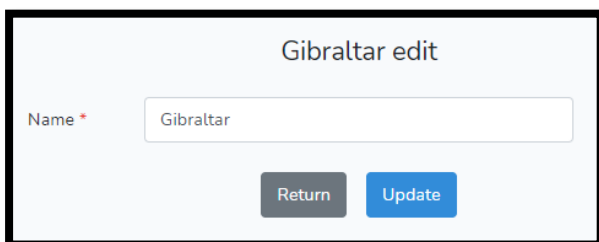


A confirmation dialog box with a light green background. At the top, there is a white input field containing the text "Gibraltar" and a red warning icon (exclamation mark inside a circle) on the right. Below the input field, a red message states: "Category 'Gibraltar' has been Deleted at 2021-03-30 14:42:12". To the right of this message are two buttons: a yellow "Restore" button and a blue "Create new" button.

Der Admin wird gezwungen eine Entscheidung zu treffen. Die gelöschte Kategorie kann wiederhergestellt werden mit allen dazugehörigen Fragen sowie Antworten oder komplett leer neu erstellt werden.

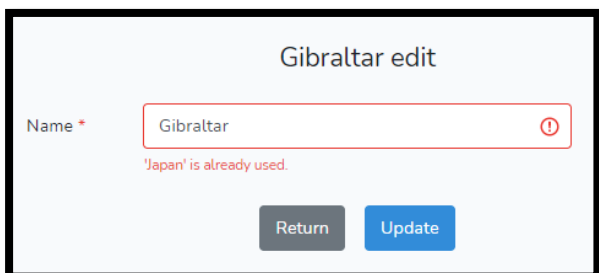
5.16.3.1 Edit

Mittels des Edit-Buttons gelangt man zu folgender Seite:



A form titled "Gibraltar edit" with a light blue background. It features a "Name *" label next to a white input field containing the text "Gibraltar". Below the input field are two buttons: a grey "Return" button and a blue "Update" button.

Der Name kann geändert werden von der Kategorie. Auch hier gibt es die Eingabeüberprüfung:



The same "Gibraltar edit" form as above, but with a validation error. The "Name *" input field, which still contains "Gibraltar", is outlined in red and has a red warning icon on the right. Below the input field, a red message states: "'Japan' is already used." The "Return" and "Update" buttons remain at the bottom.

5.16.4 Fragen

Die Fragen werden unter der dazugehörigen Kategorie aufgelistet. Im Eingabefeld kann eine neue Frage erstellt werden mit dem grünen plus für diese Kategorie.

The screenshot shows a web interface for a category named '1. Gibraltar'. At the top right, there are two buttons: a blue edit button and a red delete button. Below this is a 'Create Question' section with a text input field containing 'Question' and a green plus button. Below the form is a list of four existing questions, each with a blue edit button and a red delete button. The questions are: 'Q: Fugiat omnis numquam quo sint ipsum eum.', 'Q: Sit repudiandae assumenda enim aperiam nihil non ducimus.', 'Q: Voluptas cumque minus id tempora sequi reprehenderit.', and 'Q: Illo eius similique ipsum.'

Bereits bestehende Fragen können mittels des blauen Edit-Buttons editiert oder mittels des roten Lösch-Buttons gelöscht werden.

Die Eingabeüberprüfung prüft auf Duplikate innerhalb der Kategorie:

The screenshot shows the 'Create Question' form with the text 'Fugiat omnis numquam quo sint ipsum eum.' entered. A red border around the input field and a red warning icon indicate an error. Below the input field, the message 'This Question already exists.' is displayed. Below the form, the first question from the previous screenshot is visible with its edit and delete buttons.

5.16.4.1 Edit

Im Edit der Frage kann zusätzlich zur Änderung der Frage die richtige Antwort ausgewählt werden.

The screenshot shows the 'Edit Question' form. It has a title 'Edit Question' at the top. Below it, there is a 'Question' field with the text 'Fugiat omnis numquam quo sint ipsum eum.' and a 'CorrectAnswer' dropdown menu with 'Schneider-Erdman' selected. At the bottom, there are two buttons: 'Return' and 'Update'.

Auch hier folgt die gleiche Eingabeüberprüfung und prüft auf duplizierte Fragen innerhalb der Kategorie.

5.16.5 Antworten

Die Antworten werden unter der dazugehörigen Frage aufgelistet. Die richtige Antwort wird grün umrandet dargestellt.

Im Eingabefeld kann eine neue Frage erstellt werden mit dem grünen plus für diese Kategorie.

The screenshot shows a web application interface for creating and managing questions and answers. At the top, there is a header bar with the text "!! 2. Japan" and two icons: a blue square with a white plus sign and a red square with a white minus sign. Below the header, there is a section for creating a question. It has a label "Create Question:" followed by a text input field containing the word "Question". To the right of the input field is a green square button with a white plus sign. Below this section, there is a question entry: "Q: Exercitationem molestiae alias enim qui et numquam veniam." followed by "4 Answers needed..". To the right of the question text are the same blue plus and red minus icons. Below the question entry, there is a section for creating an answer. It has a label "Create Answer:" followed by a text input field containing the word "Answer". To the right of the input field is a green square button with a white plus sign. Below this section, there are three answer entries, each with a label "A:" followed by a name and a red square button with a white minus sign. The answers are: "A: Pfeffer-West", "A: Leffler Ltd", and "A: Kovacek, Fahey and Sanford". The "A: Leffler Ltd" entry is highlighted with a green border.

Bereits bestehende Antworten können mittels des Klicks auf die Antwort bearbeitet oder mittels des roten Lösch-Buttons gelöscht werden.

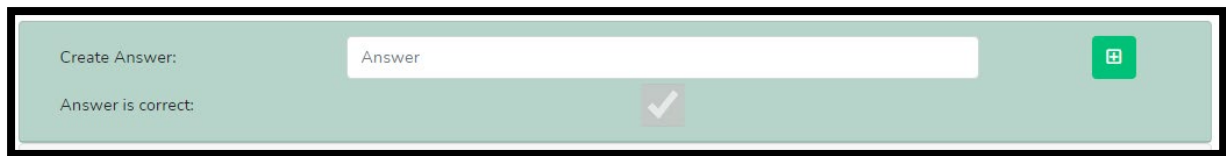
Beim Erstellen einer neuen Antwort wird zusätzlich unterschieden:

Es gibt bereits vier Antworten:

The screenshot shows a web application interface displaying a list of answers for a specific question. At the top, there is a header bar with the text "Q: Fugiat omnis numquam quo sint ipsum eum." and two icons: a blue square with a white plus sign and a red square with a white minus sign. Below the header, there are four answer entries, each with a label "A:" followed by a name and a red square button with a white minus sign. The answers are: "A: Ratke, Ernser and Dare", "A: Kuhlman, Heathcote and Spinka", "A: Schneider-Erdman", and "A: Russel, Ankunding and Lind". The "A: Schneider-Erdman" entry is highlighted with a green border.

M151 Datenbanken in Web-Applikation einbinden

Es gibt drei Antworten keine ist korrekt:



The screenshot shows a form titled 'Create Answer:'. It has a text input field containing the word 'Answer'. Below the input field, there is a label 'Answer is correct:' followed by a checked checkbox. To the right of the input field is a green button with a plus icon.

Es gibt bereits eine korrekte Antwort:



The screenshot shows the same 'Create Answer:' form with the input field containing 'Answer'. However, the 'Answer is correct:' checkbox is unchecked. The green button with a plus icon is still present.

Es gibt noch keine korrekte Antwort und noch nicht 3 Antworten:



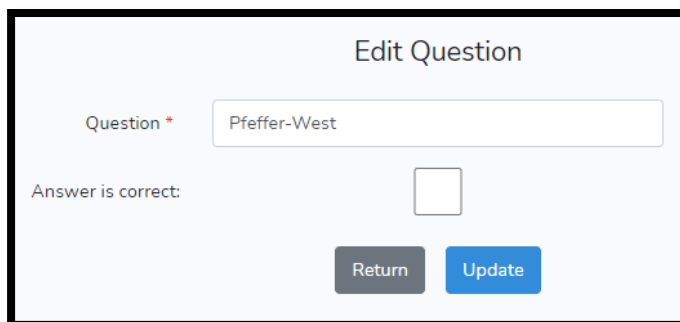
The screenshot shows the 'Create Answer:' form with the input field containing 'Answer'. The 'Answer is correct:' checkbox is unchecked. The green button with a plus icon is still present.

Die Eingabeüberprüfung prüft auch hier auf Duplikate innerhalb der Frage:



The screenshot shows the 'Create Answer:' form with the input field containing 'Pfeffer-West'. A red border surrounds the input field, and a red error message 'This Answer already exists.' is displayed below it. The 'Answer is correct:' checkbox is unchecked. The green button with a plus icon is still present.

5.16.5.1 Edit



The screenshot shows a form titled 'Edit Question'. It has a label 'Question *' followed by a text input field containing 'Pfeffer-West'. Below the input field, there is a label 'Answer is correct:' followed by an unchecked checkbox. At the bottom of the form are two buttons: 'Return' and 'Update'.

Die Eingabeüberprüfung funktioniert hier auch. Innerhalb der Frage wird auf Duplikate geprüft. Wird eine Antwort als richtig markiert aber eine andere ist bereits die korrekte, dann wird diese überschrieben mit der neuen.

6 TESTING

6.1 UNIT-TESTING

6.2 SELENIUM

Selenium ist ein Framework für automatisierte Softwaretests von Webanwendungen. Mit dieser ist es möglich Abläufe zu definieren und diese danach abzuspielen. Dadurch kann nach Belieben während des Entwickelns überprüft werden, dass alles funktioniert.

Nachfolgend ein Screenshot von den Selenium Testing:



Die erstellten Selenium-Tests waren alle Erfolgreich. Getestet wurden alle Anforderungen des Auftrags im Frontend. Dazugehören die Authentifikation respektive Redirect, CRUD zu den Categories, Questions und Answers sowie das Spiel selbst.

6.3 TESTFALLSPEZIFIKATION

<i>Testfallnummer</i>	1.01
<i>Getestete Anforderungen</i>	Admin Login
<i>Voraussetzung</i>	http://localhost/M151/m151/public/login
<i>Eingabe</i>	E-Mail: admin@admin.ch -> PW: 123
<i>Ausgabe</i>	Der User wird angemeldet und auf die Index Seite weitergeleitet.
<i>Testfallnummer</i>	1.02
<i>Getestete Anforderungen</i>	Admin Login Falsche Informationen
<i>Voraussetzung</i>	1. http://localhost/M151/m151/public/login
<i>Eingabe</i>	E-Mail: admin@a.ch -> PW: asdf
<i>Ausgabe</i>	Fehlermeldung: PW oder Benutzer Inkorrekt.
<i>Testfallnummer</i>	1.03
<i>Getestete Anforderungen</i>	Admin Login keine Informationen
<i>Voraussetzung</i>	1. http://localhost/M151/m151/public/login
<i>Eingabe</i>	E-Mail: «» -> PW: «»
<i>Ausgabe</i>	Fehlermeldung: Required.
<i>Testfallnummer</i>	2.01
<i>Getestete Anforderungen</i>	Kategorie erstellen
<i>Voraussetzung</i>	1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit
<i>Eingabe</i>	Test
<i>Ausgabe</i>	Neue Kategorie «Test» erstellt und in Liste hinzugefügt.

<i>Testfallnummer</i> 2.02	
<i>Getestete Anforderungen</i>	Kategorie erstellen Duplikat
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie Test existiert bereits
<i>Eingabe</i>	Test
<i>Ausgabe</i>	Test wird bereits verwendet.
<i>Testfallnummer</i> 2.03	
<i>Getestete Anforderungen</i>	Kategorie bearbeiten
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie Test -> Edit Button klicken
<i>Eingabe</i>	TestOK
<i>Ausgabe</i>	Kategorie Test wurde umbenannt.
<i>Testfallnummer</i> 2.04	
<i>Getestete Anforderungen</i>	Kategorie bearbeiten
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Nicht Kategorie TestOK -> Edit Button klicken
<i>Eingabe</i>	TestOK
<i>Ausgabe</i>	Existiert bereits.
<i>Testfallnummer</i> 2.05	
<i>Getestete Anforderungen</i>	Kategorie löschen
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit
<i>Eingabe</i>	Nicht Kategorie TestOK -> Delete Button klicken
<i>Ausgabe</i>	Kategorie wurde gelöscht

<i>Testfallnummer</i> 2.06	
<i>Getestete Anforderungen</i>	Frage erstellen
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen
<i>Eingabe</i>	TestFrage
<i>Ausgabe</i>	Neue Frage «TestFrage» erstellt und in Liste hinzugefügt. Fehlermeldung 4 Antworten notwendig & keine richtige Antwort
<i>Testfallnummer</i> 2.07	
<i>Getestete Anforderungen</i>	Frage erstellen Duplikat
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen
<i>Eingabe</i>	TestFrage
<i>Ausgabe</i>	TestFrage wird bereits verwendet.
<i>Testfallnummer</i> 2.08	
<i>Getestete Anforderungen</i>	Frage bearbeiten
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen 5. Frage TestFrage erstellt
<i>Eingabe</i>	<ol style="list-style-type: none"> 1. Frage TestFrage Button Edit klicken 2. Umbenennen zu TestFrageOK
<i>Ausgabe</i>	Frage TestFrage wurde umbenannt zu TestFrageOK.

Testfallnummer 2.09

Getestete Anforderungen	Frage bearbeiten Duplikat
Voraussetzung	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen 5. Frage TestFrageOK erstellt
Eingabe	<ol style="list-style-type: none"> 1. Neue Frage erstellen: TestFrage 2. Frage TestFrage Button Edit klicken 3. Umbenennen zu TestFrageOK
Ausgabe	Existiert bereits.

Testfallnummer 2.10

Getestete Anforderungen	Frage löschen
Voraussetzung	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen 5. Frage TestFrageOK erstellt
Eingabe	Frage TestFrageOK -> Delete Button klicken
Ausgabe	Frage wurde gelöscht

Testfallnummer 2.11

Getestete Anforderungen	Antwort erstellen
Voraussetzung	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen 5. Frage TestFrageOK erstellt 6. Frage TestFrage aufklappen
Eingabe	TestAntwort
Ausgabe	Neue Antwort «TestAntwort» erstellt und in Liste hinzugefügt.

Testfallnummer 2.12

Getestete Anforderungen	Antwort erstellen Duplikat
Voraussetzung	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen 5. Frage TestFrageOK erstellt 6. Frage TestFrage aufklappen
Eingabe	TestAntwort
Ausgabe	TestAntwort wird bereits verwendet.

Testfallnummer 2.13

Getestete Anforderungen	Antwort bearbeiten
Voraussetzung	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen 5. Frage TestFrageOK erstellt 6. Frage TestFrage aufklappen
Eingabe	<ol style="list-style-type: none"> 1. Antwort TestAntwort Button Edit klicken 2. Umbenennen zu TestAntwortOK
Ausgabe	Antwort TestAntwort wurde umbenannt zu TestAntwortOK.

Testfallnummer 2.14

Getestete Anforderungen	Antwort bearbeiten Duplikat
Voraussetzung	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen 5. Frage TestFrageOK erstellt 6. Frage TestFrage aufklappen
Eingabe	<ol style="list-style-type: none"> 1. Neue Antwort erstellen: TestAntwort 2. Antwort TestAntwort Button Edit klicken 3. Umbenennen zu TestAntwortOK
Ausgabe	Existiert bereits.

<i>Testfallnummer</i> 2.15	
<i>Getestete Anforderungen</i>	Antwort löschen
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen 5. Frage TestFrageOK erstellt 6. Frage TestFrage aufklappen 7. Antwort TestAntwortOK erstellt
<i>Eingabe</i>	Antwort TestAntwortOK -> Delete Button klicken
<i>Ausgabe</i>	Antwort wurde gelöscht

<i>Testfallnummer</i> 2.16	
<i>Getestete Anforderungen</i>	Richtige Antwort löschen
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen 5. Frage TestFrageOK erstellt 6. Frage TestFrage aufklappen 7. Antwort TestAntwortOK erstellt als richtige Frage
<i>Eingabe</i>	Antwort TestAntwortOK -> Delete Button klicken
<i>Ausgabe</i>	Antwort wurde gelöscht

<i>Testfallnummer</i> 2.17	
<i>Getestete Anforderungen</i>	Frage Validierung
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen 5. Frage TestFrageOK erstellt 6. Frage TestFrage aufklappen
<i>Eingabe</i>	4 Antworten erstellen mit zufälligen werten.
<i>Ausgabe</i>	Keine Fehlermeldung bei der Frage mehr sichtbar.

<i>Testfallnummer</i> 2.18	
<i>Getestete Anforderungen</i>	Kategorie Validierung
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen
<i>Eingabe</i>	4 Fragen mit jeweils 4 Antworten erstellen.
<i>Ausgabe</i>	Keine Fehlermeldung neben der Kategorie sichtbar (!!).
<i>Testfallnummer</i> 2.19	
<i>Getestete Anforderungen</i>	Kategorie Validierung
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/models/edit 3. Kategorie TestOK erstellt 4. Kategorie TestOK aufklappen 5. Zufällige Antwort löschen 6. Spielernamen festgelegt 7. http://localhost/M151/m151/public/cat/select
<i>Eingabe</i>	-
<i>Ausgabe</i>	Kategorie TestOK ist deaktiviert.
<i>Testfallnummer</i> 3.01	
<i>Getestete Anforderungen</i>	Highscoreliste Eintrag löschen
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. http://localhost/M151/m151/public/highscores
<i>Eingabe</i>	Zufälliger Highscore -> Delete Button klicken
<i>Ausgabe</i>	Highscore wurde gelöscht.
<i>Testfallnummer</i> 4.01	
<i>Getestete Anforderungen</i>	Anzeige richtig & falsch beantwortet % von Frage
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Spielernamen festgelegt 2. Zufällige Kategorie ausgewählt 3. http://localhost/M151/m151/public/play
<i>Eingabe</i>	-
<i>Ausgabe</i>	Balken unterhalb der Frage wird angezeigt % richtig & falsch.

<i>Testfallnummer</i> 5.01	
<i>Getestete Anforderungen</i>	Aktuelle Punktzahl
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Spielernamen festgelegt 2. Zufällige Kategorie ausgewählt 3. http://localhost/M151/m151/public/play
<i>Eingabe</i>	-
<i>Ausgabe</i>	Aktuelle Punktzahl wird angezeigt im unteren Menü.

<i>Testfallnummer</i> 6.01	
<i>Getestete Anforderungen</i>	50:50 Joker
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. Spielernamen festgelegt 3. Zufällige Kategorie ausgewählt 4. http://localhost/M151/m151/public/play
<i>Eingabe</i>	50:50 Joker Button wird aktiviert
<i>Ausgabe</i>	2 Falsche Antworten werden ausgeblendet.

<i>Testfallnummer</i> 6.02	
<i>Getestete Anforderungen</i>	50:50 Joker
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. Spielernamen festgelegt 3. Zufällige Kategorie ausgewählt 4. http://localhost/M151/m151/public/play 5. 50:50 Joker bereits verwendet
<i>Eingabe</i>	
<i>Ausgabe</i>	50:50 Joker Button deaktiviert.

<i>Testfallnummer</i> 7.01	
<i>Getestete Anforderungen</i>	Auswahl richtige Antwort 30 Punkte pro richtige Frage
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. Spielernamen festgelegt 3. Zufällige Kategorie ausgewählt 4. http://localhost/M151/m151/public/play
<i>Eingabe</i>	<ol style="list-style-type: none"> 1. Mousover Antworten -> um die Richtige ausfindig zu machen. 2. Die richtige Antwort auswählen
<i>Ausgabe</i>	<ol style="list-style-type: none"> 1. Question Progress aktualisiert 2. % Frage richtig/falsch aktualisiert 3. Richtige Antwort wird dargestellt 4. Auswahl nächste Frage möglich 5. 30 Punkte wurden dem Konto gutgeschrieben
<i>Testfallnummer</i> 8.01	
<i>Getestete Anforderungen</i>	Auswahl falsche Antwort
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. Spielernamen festgelegt 3. Zufällige Kategorie ausgewählt 4. http://localhost/M151/m151/public/play
<i>Eingabe</i>	<ol style="list-style-type: none"> 1. Mousover Antworten -> um die Richtige ausfindig zu machen. 2. Eine falsche Antwort auswählen
<i>Ausgabe</i>	<ol style="list-style-type: none"> 1. Question Progress aktualisiert (roter Balken) 2. % Frage richtig/falsch aktualisiert 3. Richtige Antwort wird dargestellt und die falsch ausgewählte. 4. Auswahl «Continue End Screen» möglich
<i>Testfallnummer</i> 9.01	
<i>Getestete Anforderungen</i>	Spielernamen eingabe
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. http://localhost/M151/m151/public/ 2. Keine aktive Session vorhanden 3. Auf Play klicken.
<i>Eingabe</i>	<ol style="list-style-type: none"> 1. TestUser
<i>Ausgabe</i>	<ol style="list-style-type: none"> 1. Weiterleitung zu Kategorie auswählen

<i>Testfallnummer</i> 9.02	
<i>Getestete Anforderungen</i>	Spielername Eingabe leer
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. http://localhost/M151/m151/public/ 2. Keine aktive Session vorhanden 3. Auf Play klicken.
<i>Eingabe</i>	<ol style="list-style-type: none"> 1. «»
<i>Ausgabe</i>	<ol style="list-style-type: none"> 1. Fehlermeldung, Feld rot markiert.
<i>Testfallnummer</i> 11.01	
<i>Getestete Anforderungen</i>	Kategorie wählen Messung der Spielzeit Eintrag Highscoreliste
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. Spielername festgelegt 3. Zufällige Kategorie ausgewählt 4. http://localhost/M151/m151/public/play 5. Mind. 1 Frage korrekt beantwortet
<i>Eingabe</i>	<ol style="list-style-type: none"> 1. Finish Quiz klicken 2. View Highscores klicken
<i>Ausgabe</i>	<ol style="list-style-type: none"> 1. Weiterleitung zu Highscoreliste 2. Aktuelles spiel wird dargestellt und grün hervorgehoben 3. Spielzeit wurde korrekt gemessen
<i>Testfallnummer</i> 11.02	
<i>Getestete Anforderungen</i>	Eintrag Highscoreliste Fehlgeschlagenes Spiel
<i>Voraussetzung</i>	<ol style="list-style-type: none"> 1. Angemeldet mit Admin Account 2. Spielername festgelegt 3. Zufällige Kategorie ausgewählt 4. http://localhost/M151/m151/public/play 5. Frage falsch beantwortet
<i>Eingabe</i>	<ol style="list-style-type: none"> 1. Continue End Screen klicken 2. View Highscores klicken
<i>Ausgabe</i>	<ol style="list-style-type: none"> 1. Weiterleitung zu Highscoreliste 2. Aktuelles spiel wird nicht dargestellt

M151 Datenbanken in Web-Applikation einbinden

<i>Testfallnummer</i> 12.01	
<i>Getestete Anforderungen</i>	Zufällige Reihenfolge der Fragen
<i>Voraussetzung</i>	<ol style="list-style-type: none">1. Spielername festgelegt2. http://localhost/M151/m151/public/play
<i>Eingabe</i>	<ol style="list-style-type: none">1. Zufällige Kategorie wählen2. Frage notieren3. Session zerstören4. Gleiche Kategorie wählen
<i>Ausgabe</i>	<ol style="list-style-type: none">1. Frage überprüfen, war es die gleiche?2. Wiederholen 2-3x

6.4 TESTPROTOKOLL

Testfall	Resultat	Datum	Tester	Bemerkung
01	Funktioniert	02/02/21	CO	-
02	Funktioniert	02/02/21	CO	-
03	Funktioniert	02/02/21	CO	-
04	Funktioniert	02/02/21	CO	-
05	Funktioniert	02/02/21	CO	-
06	Funktioniert	02/02/21	CO	-
07	Funktioniert	02/02/21	CO	-
08	Funktioniert	02/02/21	CO	-
09	Funktioniert	02/02/21	CO	-
10	Funktioniert	02/02/21	CO	-

7 AUSWERTUNG

7.1 ALLGEMEIN

7.2 PLANUNGSDISKREPANZ

7.3 TESTING

8 ANHANG
