30. März 2021

Christopher OConnor

BBBaden

M151

Datenbanken in Web-Applikation einbinden

Inhaltsverzeichnis

[2 Aufgabenstellung 3](#_Toc67410896)

[2.1 Wer wird Millionär im Fernsehen 3](#_Toc67410897)

[2.2 Wer wird Millionär als Webapplikation 3](#_Toc67410898)

[2.3 Anforderungen 3](#_Toc67410899)

[2.3.1 Administration 3](#_Toc67410900)

[2.3.2 GUI 3](#_Toc67410901)

[2.3.3 Spiel 4](#_Toc67410902)

[3 Projektplanung 5](#_Toc67410903)

[3.1 Definition der Teilziele 5](#_Toc67410904)

[3.1.1 Datenbank 5](#_Toc67410905)

[3.1.2 Model Controller (CRUD) 5](#_Toc67410906)

[3.1.3 Routing 5](#_Toc67410907)

[3.1.4 Admin-User Account 5](#_Toc67410908)

[3.1.5 Spiel 5](#_Toc67410909)

[3.1.6 Input Validation 5](#_Toc67410910)

[3.1.7 Testing 5](#_Toc67410911)

[3.2 Zeitplan 6](#_Toc67410912)

[4 Analyse / Entscheidung 7](#_Toc67410913)

[4.1 Dynamische Elemente 7](#_Toc67410914)

[4.1.1 Tier 1 Presentation 7](#_Toc67410915)

[4.1.2 Tier 2 Webserver 7](#_Toc67410916)

[4.1.3 Tier 3 Application Server 7](#_Toc67410917)

[4.1.4 Tier 4 Data Server 7](#_Toc67410918)

[4.2 Verwendete Technologien 7](#_Toc67410919)

[4.2.1 Tier 1 Presentation 7](#_Toc67410920)

[4.2.2 Tier 2 Webserver 7](#_Toc67410921)

[4.2.3 Tier 3 Application Server 7](#_Toc67410922)

[4.2.4 Tier 4 Data Server 7](#_Toc67410923)

[4.2.5 Zusätzliche Technologien 7](#_Toc67410924)

[5 Realisierung 8](#_Toc67410925)

[5.1 Datenbank ERM 8](#_Toc67410926)

[5.2 Migrations 8](#_Toc67410927)

[5.3 Laravel Models 9](#_Toc67410928)

[5.3.1 Property & Keytype 9](#_Toc67410929)

[5.3.2 Fillable 9](#_Toc67410930)

[5.3.3 Relations 9](#_Toc67410931)

[5.3.4 Custom Get Method 10](#_Toc67410932)

[5.3.5 Syntax 10](#_Toc67410933)

[5.4 DB: SEED 11](#_Toc67410934)

[5.4.1 Factory 11](#_Toc67410935)

[5.4.2 Seeder 11](#_Toc67410936)

[5.5 Model Controller (CRUD) 13](#_Toc67410937)

[5.6 Routes 13](#_Toc67410938)

[5.7 Blade Templating Engine 13](#_Toc67410939)

/\* Management Summary \*/

# 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.

## 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.

## 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.

## Anforderungen

### 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.

### 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.

### 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 nutzten.
* 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: Einziger Tier, der eine direkte Verbindung zur Datenbank hat.

# Projektplanung

## 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.

### Datenbank

* ERM
* Migration
* Models generieren
* DB: Seed

### Model Controller (CRUD)

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

### Routing

### Admin-User Account

* Login / Logout
* Neuer Admin erstellen
* Authentifikation

### Spiel

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

### Input Validation

* Front-End
* Back-End

### Testing

* Unittests

## 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.

asdf

# Analyse / Entscheidung

## Dynamische Elemente

### Tier 1 Presentation

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

### Tier 2 Webserver

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

### Tier 3 Application Server

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

### Tier 4 Data Server

## Verwendete Technologien

### 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.

### Tier 2 Webserver

Routes werden in Laravel im File /routes/web.php definiert. Anfragen können geprüft und zum Entsprechenden Controller weitergeleitet werden. Die Back-End Eingabeüberprüfung erfolgt entweder direkt im Controller oder kann als Request-Klasse direkt überprüft werden. Front-End Überprüfung wird mit HTML und JS implementiert.

### 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.

### 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.

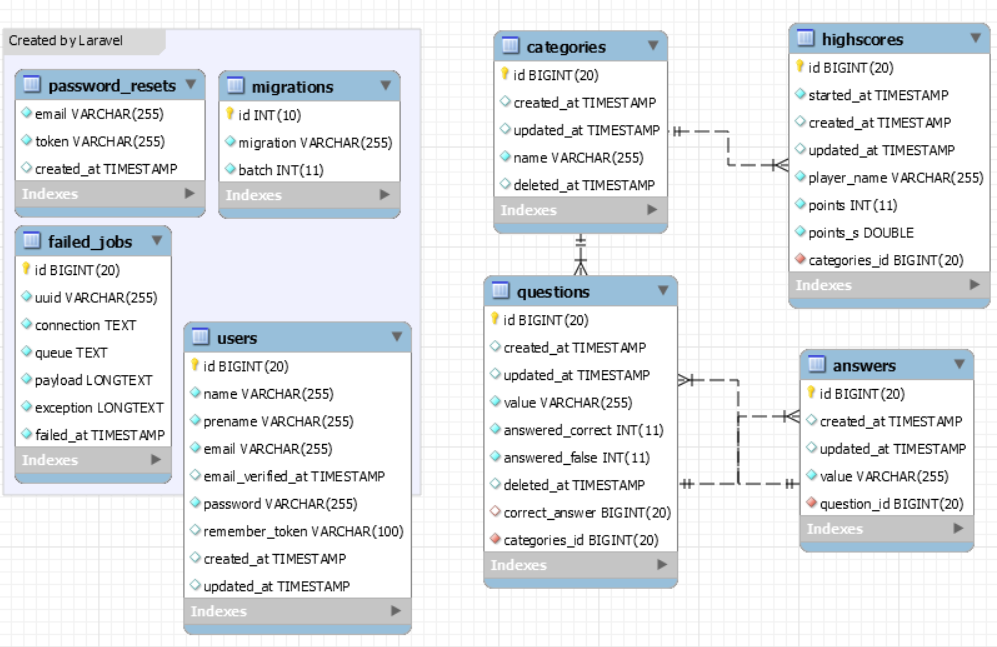
### Zusätzliche Technologien

* Composer – Dependancy Manager für PHP
* PHP Artisan – Laravel Konsole

# Realisierung

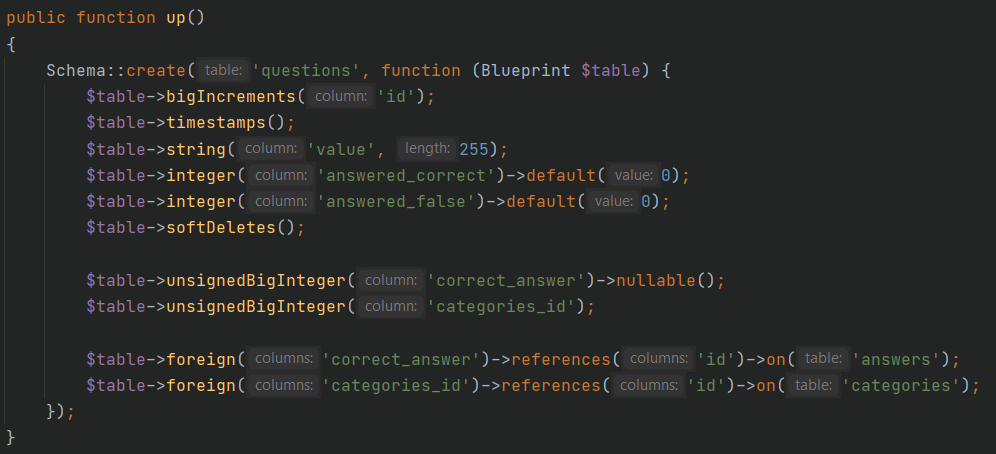
## Datenbank ERM

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



## 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»

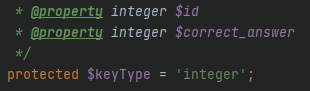
## Laravel Models

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.

### Property & Keytype



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

### Fillable

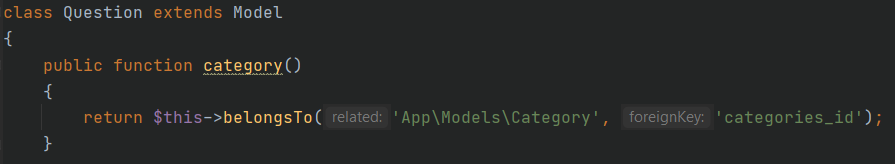


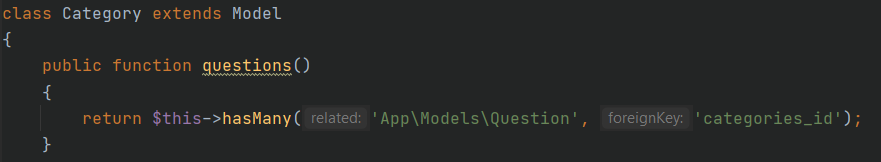
Im Array «Fillable» werden alle zugänglichen Attribute definiert. Für Attribute, welche nicht zugänglich sein sollen z.B. das Passwort wird noch ein zusätzlicher Eintrag im Array «$hidden» erstellt.



### Relations

Beziehungen müssen in beiden Models definiert werden.

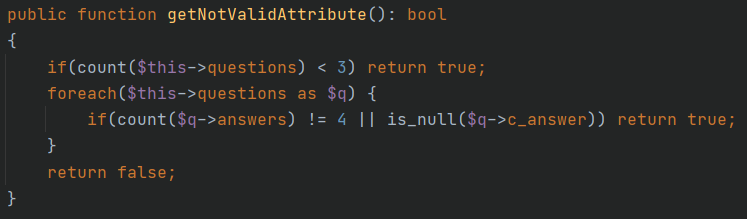




Die Relations werden beim Generieren miterzeugt. Laravel kann alle möglichen formen von Relations nachbilden und viele mehr! Beispielsweise «has-through». Diese müssen aber manuell entsprechend hinzugefügt werden.

### Custom Get Method

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



Möchte man diese aufrufen wäre dies die Syntax: «$model->not\_valid»

### Syntax

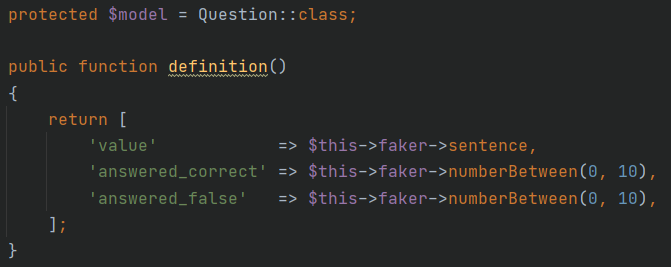
Als Beispiel wird ein Objekt des Models Question verwendet. Möchte man ein wert von einem Attribut haben wäre die Syntax: «$question->name». Wenn man nun von der Question alle Answers haben möchte, kann man dies so erreichen: «$question->answers».

## DB: SEED

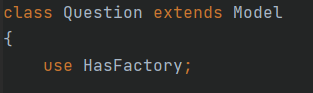
Um die Datenbank mit Dummy Daten zu füllen, benötigt man in Laravel sogenannte Factories und Seeders. Diese können mit dem Befehl: «php artisan migrate:fresh –-seed» ausgeführt werden.

### 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:

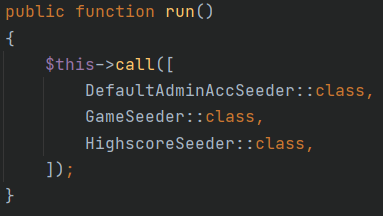


Ab Laravel Version 8.x benötigt die Factory das entsprechende Model dazu muss folgender Eintrag im Model hinzugefügt werden:



### 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.



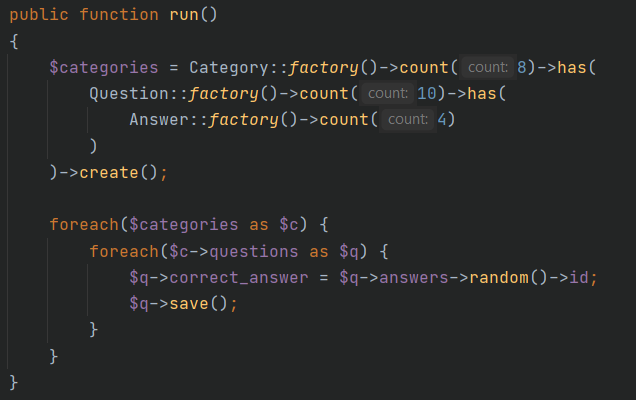
#### Statischer Seeder «DefaultAdminAcc»

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



#### Dynamischer Seeder «GameSeeder»:

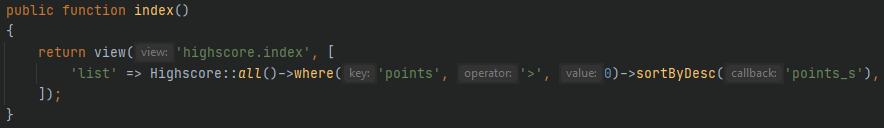
In diesem Seeder wird gleich die Kategorie, die dazugehörigen Fragen und Antworten erstellt.



## 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.

### Index



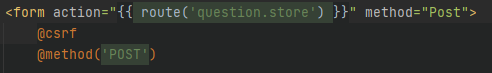
In der Index Methode wird definiert welche View dargestellt werden soll wenn man die Index URL des Models aufruft. Für das Model Question wäre dies beispielsweise: «m151/question/». In den meisten Fällen wäre dies eine Auflistung der vorhandenen Models wie bei der Highscoreliste.

### Create

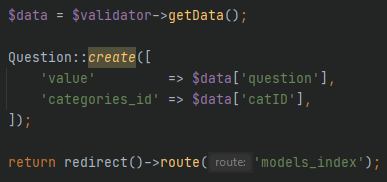
Create gibt die View zurück, mit welcher ein neues Objekt erstellt werden kann. In diesem Projekt wird dies für Category, Question und Answer an einem Ort getätigt und deswegen nicht relevant.

### Store

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



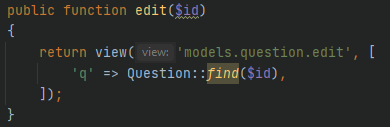
Das ausgefüllte Formular wird als Request an die Methode übergeben. Es gibt verschiedene Möglichkeiten für die Validierung der Request. Dazu mehr in einem späteren Kapitel. Danach kann das Objekt erstellt werden.



### 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.

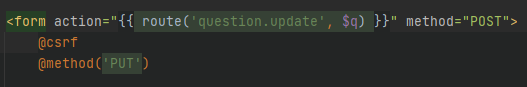
### Edit



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

### 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.

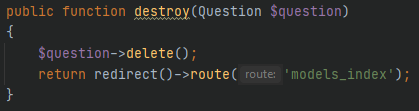


Nach der Validierung kann dann das Objekt angepasst werden:



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](#_Store) benötigt.

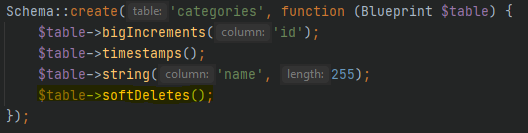
### Destroy



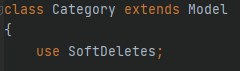
Die selektierte Question wird zerstört. In Laravel wird dies mit einer Post Request ausgeführt.

### 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.



Im Model selbst muss zusätzlich folgende Zeile hinzugefügt werden:



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 mittgeteilt:



## Routes

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

### Get



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}’)».

### Post



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.

### Return Funktion



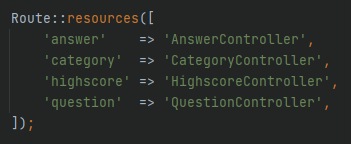
Innerhalb der Route ist es möglich Funktionen zu schreiben und gleich die Weiterleitung an eine entsprechende View zu erstellen.

### Auth



Die Auth Routes beinhaltet das Login, Register und Logout.

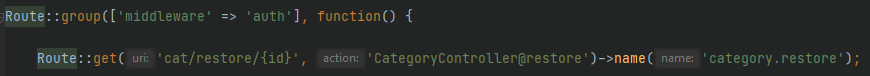
### Resource Models



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

### 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.



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

## Blade Templating Engine

### 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.

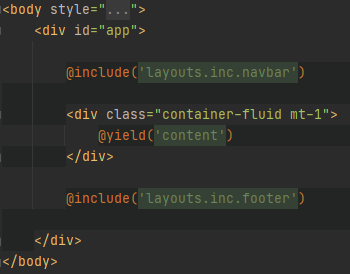
#### Head

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.



#### Include

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



#### Yield

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

### Unterseiten

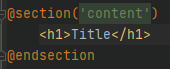
#### Extends

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



#### Section

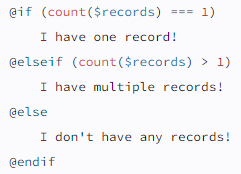
Damit auch der Inhalt am Richtigen Ort angezeigt wird, benötigt es die Definition an welche Sektion welcher Inhalt eingefügt wird. In diesem Projekt gibt es nur eine Sektion. Alles innerhalb des «content» wird im Template an die entsprechende Yield-Stelle während der Laufzeit eingefügt.



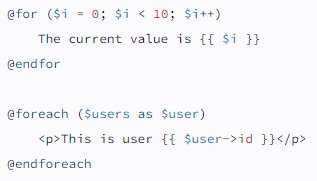
### Blade PHP

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

#### IF



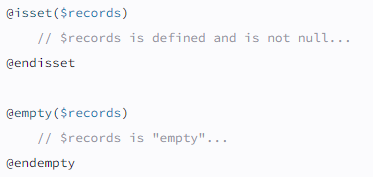
#### For



#### Auth



#### Isset



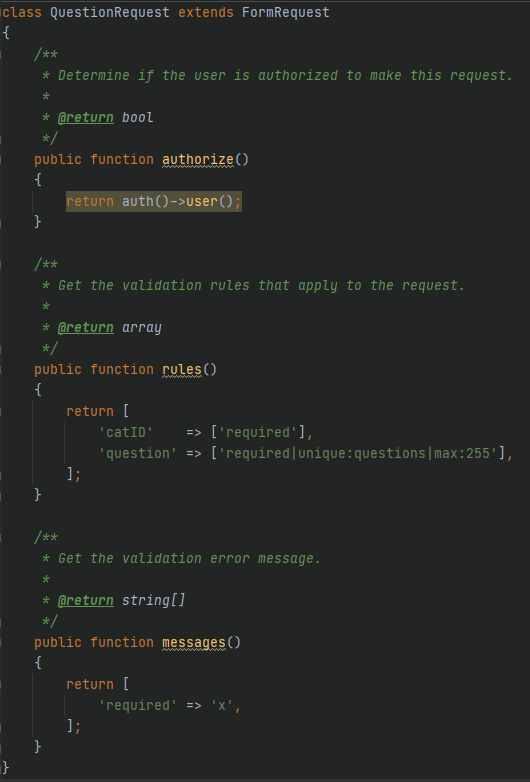
#### Echo



## Request Input Validation Backend

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

### Request Validation Klasse

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

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

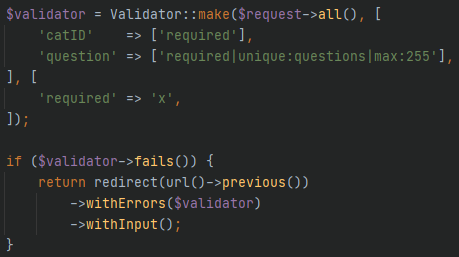
In den «rules» werden die Regeln für eine Erfolgreiche Request definiert.

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.

### Request Validation Controller

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

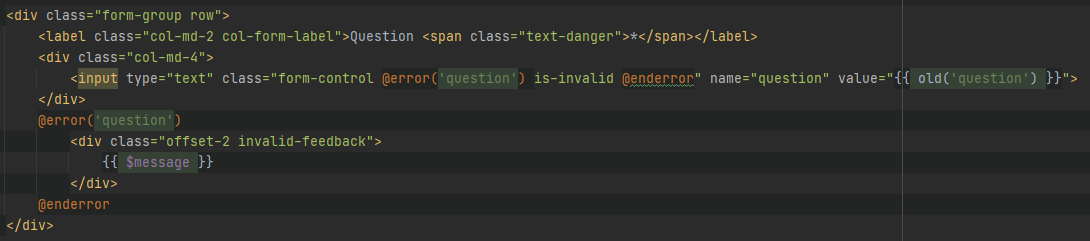


Im ersten Array werden die Regeln definiert und im zweiten Array die überschriebene Message Antwort.   
Bei dieser Variante benötigt es explizit noch den Return wie im Gegensatz zur [Variante 1](#_Request_Validation_Klasse) wird dies nicht automatisch gesteuert.

### 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.

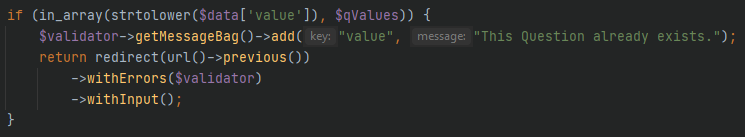
Damit diese Eingaben wieder an ihren entsprechenden Ort eingefügt werden und eine Fehlermeldung angegeben wird benötigt es im Formular die Tags «@error» und die Methode «old()».



### 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.

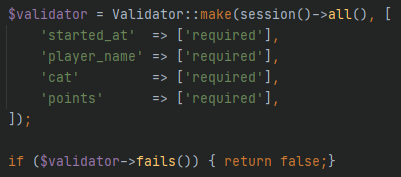
Es ist möglich einen Eintrag dem Message-Bag hinzuzufügen und diesen an die Ursprüngliche Seite zurückzugeben:



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.

### Session Validation

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



## Input Validation Frontend

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



Realisierung? -> Ansteuerung Datenbank

Aufbau Interface

Realisierung:

Administrationsbereich

(Login), (CRUD), (New Admin), (Update Admin Acc), Del Highscore …

Input Validation (Client & Server Side)

Tier 1

Tier 2

Tier 3

Tier 4

Spiel Dokumentieren

Bereiten Sie eine Abnahme Ihrer Applikation vor. Erstellen Sie alle dafür notwendigen Dokumente.

TESTING. + Unittests + Integrationstest mit Selenium