Begrüssung

Willkommen zu meiner Präsentation über meine Individuelle praktische Arbeit. Ich werde hier die Vorgehensweise und Umsetzung meiner IPA vorstellen. Diese fand über den Zeitraum von zehn Tagen vom 13. März bis zum 28. März statt.

Agenda

Die Präsentation ist folgendermassen aufgebaut:

Zuerst werde ich die Vorgaben und den Auftrag grob beschreiben.

Dann werde ich Aufzeigen wie ich von Anfang bis ende Vorgegangen bin, um den Auftrag umzusetzen.

Es folgt eine kurze Auswertung, wo ich die Vorgaben mit dem finalen Produkt vergleiche.

Zum Schluss noch mein persönliches Fazit und was ich aus dieser Arbeit mitnehme.

Agenda | Auftrag

Beginnen wir mit der Definition des Auftrags.

Projektauftrag

Hier ist was die Applikation können sollte. Es sollte Ein Verwaltungssystem erstellt werden, in dem Shops und ihre Produkte erfasst werden können. Aus diesen Daten sollen dann Bestellungen mit Produkten erstellt werden können. Dadurch soll eine Übersicht für ausgeführte Bestellungen vorhanden sein.

Funktionalität

Benutzer können sich mit einer E-Mailadresse Registrieren und Anmelden, was ihnen Zugang auf alle Daten gibt. Zur Authentifizierung wird ein Bearer-Token verwendet und die Kommunikation findet in JSON statt. Es existiert kein Frontend aber eine Dokumentation mit Swagger.

Mittel

Zur Umsetzung wurden folgende Mittel und Technologien eingesetzt.

Was ist XAMPP

Als lokale Server-Umgebung wurde XAMPP verwendet. XAMPP ist eine Abkürzung für Cross-Plattform, Apache, MySQL, PHP, Perl. Dadurch wurde also Apache als Webserver, MySQL als Datenbank und PHP als Programmiersprache zur Verfügung gestellt.

https://www.educba.com/what-is-xampp/

Was ist Laravel

Anstatt Vanilla-PHP wurde das Framework Laravel benutzt. Laravel bietet diverse nützliche Tools.

Die wichtigsten davon sind die MVC-Struktur von der hier nur der Model-Controller Teil benutzt wird, da wir keine Views haben,

Artisan ist das Comand-Line-Interface, das von Laravel bereitgestellt wird, damit können Migrationen durchgerührt werden und Boiler Plate-Code erzeugt werden,

Eloquent ist das ORM oder Object-Relational-Mapper, es verbindet die Modelle mit der Datenbank

Sanctum ist ein Authentisierungs System das Laravel bietet und eine einfache Integration von Bearer-Token erlaubt

https://builtin.com/software-engineering-perspectives/laravel

Was ist eine API

Die Umsetzung als API oder Application Programming Interface, es ist eine Sammlung von Definitionen und Protokollen, für das Erstellen von Software. Es erlaubt es ein Produkt oder Service bereit zu stellen, ohne zu wissen wie diese später implementiert werden. In diesem Fall eine Bestellungsverwaltung. Die Endpoints, die erstellt wurden, können von einer Mobile-App, einer Web-App oder auch von einer anderen API verwendet werden.

edhat.com/en/topics/api/what-are-application-programming-interfaces

JSON steht für JavaScript Object Notation und ist ein gängiges Format für das Speichern und Transportieren von Daten.

Was ist Swagger

Swagger ist Open Source und bietet Regeln, Spezifikationen und Tools für die Entwicklung und Beschreibung von RESTful-APIs. Das Swagger-Framework ermöglicht es Entwicklern, interaktive, maschinen- und menschenlesbare API-Dokumentation zu erstellen.

Das hier benutzte L5-Swagger ist eine auf Laravel angepasste Version.

https://www.techtarget.com/searchapparchitecture/definition/Swagger

Agenda | Vorgehen

Nun zum Vorgehen.

IPERKA

Bei meiner Vorgehensweise folgte ich der IPERKA Projekt-Methode. Diese stellt sich aus den sechs Schritten Informieren, Planen, Entscheiden, Realisieren, Kontrollieren und Auswerten zusammen.

Informieren

Im ersten Schritt der Informations-Phase, ging es hauptsächlich darum kleinere Unklarheiten zu beheben. Wie zum Beispiel, ob jeder Benutzer auf alle Daten jedes Benutzers zugreifen darf. Da ich mit den verwendeten Technologien schon weitestgehend vertraut war, hielt sich der Recherchier Bedarf aber in Grenzen.

Planen

Zeitplan (sieht man ok)

Die gesamte Planung für die Umsetzung der IPA, wurde in einem Zeitplan festgehallten. Wobei der geschätzte Aufwand in Orange und der tatsächliche Aufwand in Grün dargestellt ist. Wie man sehen kann, hatte ich eine grössere Fehleinschätzung, beim Punkt Testkonzept erstellen.

Klassendiagramm (sieht man ok)

Das Klassendiagramm sollte in der Dritten Normalform sein und musste bestimmte Werte in bestimmten Tabellen enthalten.

Aktivitätsdiagramm (sieht man nicht)

In diesem Aktivitätsdiagramm ist abgebildet, wie ein neues Produkt erstellt wird. Man sieht es hier zwar nicht so gut, ich möchte aber einfach die Schritte aufzeigen, die in dieser Anwendung bei jedem Request nötig sind.

Schritt 1, Route-Model-Binding

Der erste Schritt ist optional. Falls im URL eine ID mitgegeben wurde, wird mit dieser von Laravel das entsprechende Model aus der Datenbank geholt. Schlägt das fehl, wird eine Fehlermeldung zurückgegeben. Diese Funktionalität heisst Route Model Binding und ich werde dies später noch einmal aufgreifen.

Schritt 2, Validierung

Im zweiten Schritt werden alle im Post Request mitgegebenen Daten validiert.

Schritt 3, File Upload

Da Produkte ein Bild enthalten können, ist dieser zusätzliche Schritt nötig, um einen separaten File Upload zu machen.

Schritt 4, Datenbankspeicherung / Antwort

Am Ende dieses Vorganges, wird ein neuer Eintrag in der Datenbank erstellt.

Schritt 5, Fehlermeldung

Falls zu irgendeinem Zeitpunkt in diesem Prozess fehlerhafte oder ungültige Daten erkannt werden, muss eine Fehlermeldung generiert und zurückgegeben werden.

Aufbaudiagramm (sieht man die Details nicht)

Hier sieht man den Ablauf, einer Interaktion mit der API vereinfacht dargestellt.

Im Router werden die URLs definiert, die von aussen angesprochen werden können.

Diese Routes korrespondierend zu einer Methode im Controller.

Die Methoden akzeptieren als Parameter einen bestimmten Request Typ. Zum Beispiel einen Produkt Update Request. In diesem Request ist definiert, was alles Enthallten sein muss. Dinge wie Shop ID, Namen und Bild URL. Hier werden diese Daten auch gleich validiert.

In der Controller Methode wird dann eine Resource erzeugt. Diese definiert das Format der Antwort und wie dessen Daten präsentiert werden.

Zum Schluss wird die Resource als JSON-Response zurückgegeben.

Entscheiden

In der Entscheidungsphase kamen mir die Laravel-Konventionen, die genauen Vorgaben der Aufgabenstellung und meine Erfahrung mit den verwendeten Technologien zugute. So gab es nicht so viele Entscheidungen, die getroffen werden mussten.

Entscheidungs-Matrix mit Scout

Bei den Entscheidungen, die getroffen werden mussten, habe ich versucht so systematisch wie möglich vorzugehen. Dazu benutzte ich Tabellen mit gewichteten Werten, um die Optionen zu vergleichen.

Realisieren

Nach der ausführlichen Planungsphase ging es schlussendlich an die Realisationsphase.

Migrationen (Vorgabe)

Nach dem die Datenbank Verbindung sichergestellt ist, können über Migrationen die Tabellen erstellt werden. Hier ist die Migration für die Produkte-Tabelle dargestellt. In der Up- Funktion wird die Tabelle mit den Kolumnen ID, shop\_id, Name, Image und Time Stamps erstellt. In der Down-Funktion wird die Tabelle gelöscht.

Modell (Vorgabe)

Die Tabellen in der Datenbank können dann durch Modell-Klassen angesprochen werden. Diese Modelle können mit Methoden versehen werden, um ihr Verhältnis zu anderen Modellen verfügbar zu machen.

Routen mit Route-Model-Binding

Um dann auf diese Modelle zuzugreifen, müssen Routen definiert werden. Hier als Beispiel die Routen für die Produkte. Dabei werden die URLs den entsprechenden Controller Methoden zugeordnet.

Wobei die CRUD-Routen praktischerweise mit der Resource-Methode auf einmal erstellt werden können. In den URLs sieht man werte die in Klammern abgebildet sind. Dar Client fügt dort eine ID, des entsprechenden Modells ein, das er erhalten will. Laravel benutzt etwas namens Route-Modell Binding, wobei es dort automatisch das richtige Modell aus der Datenbank holt, noch bevor der Controller aufgerufen wird.

404 Fehlermeldung

Das führt zu einem Problem bei den Fehlermeldungen. Nämlich, dass diese automatisch generiert werden und nicht im Controller von mir bestimmt werden können. Das machte es nötig, Alle NotFoundExceptions für die entsprechenden Routen abzufangen und eine personalisierte JSON-Fehlermeldung zurückzugeben.

Controller

Hier zu sehen ist die Update-Funktion im Produkt-Controller. Die Logik in der Mitte überprüft, ob ein neues Bild mitgegeben wurde und wenn ja, speichert die neuen Daten für das Bild. Wie man ganz oben sehen kann, erwartet diese Funktion als Eingabe einen Request des Type UpdateProductRequest und gibt als Ausgabe eine Produkt Resource zurück.

Request mit Fehlermeldung (Vorgabe)

Sehen wir uns einmal den Request etwas genauer an. Der Request gibt die Daten und deren Struktur vor, die vom Client geschickt werden muss. In diesem Fall, um ein Produkt zu verändern. Im Bereich Regeln, sind die Variablen definiert. Laravel bietet Parameter, um die Validierung zu machen. Hier sagen wir, dass die shop\_id in der Shop-Tabelle existieren muss.

Im Messages Teil werden die individualisierten Fehlermeldungen definiert.

Ressource (Vorgabe)

In den Ressourcen wird definiert, was vom Controller zurückgegeben wird. Hier wird eine Produkt-Ressource erstellt.

Swagger-Annotation

Zur Erstellung der Swagger-Dokumentation, müssen im Code Swagger-Annotationen eingefügt werden. Das Passiert in diesem Projekt an verschiedenen Stellen und es würde eine lange Zeit dauern dies genau aufzuzeigen. Kurz gefasst funktioniert es aber so:

Es müssen Tags definiert werden also die Gruppen von Endpoints, die angefragt werden können. In diesem Fall sind das Dinge wie Produkte, Shops und Bestellungen.

Der wichtigste Teil ist dann, dass über der Controller-Funktion, die in der Dokumentation aufrufbar sein soll eine Swagger-Annotation eingefügt wird. Dort wird dann beschrieben, wie diese Funktion in der Dokumentation dargestellt werden soll.

Hier sieht man zum Beispiel die Parameter, die mitgegeben werden müssen, um die Update Product-Funktion aufzurufen. Also die ID des Produktes, die im Path also URL mitgegeben werden muss.

Slide 2

Hier werden weiter Parameter beschrieben, wie shop\_id, Name, Image. Also die Sachen die wir schon aus dem Request kennen.

Slide 3

Und hier werden noch die möglichen Rückgabewerte beschrieben.

Slide 4

So würde es denn alles zusammen aussehen, wie es über der Update-Funktion im Produkt-Controller steht.

Swagger Produkt

Die Produkt-Sektion in der Dokumentation sieht dann so aus. Oben links sieht man die verfügbaren End Points für Produkte, diese korrespondieren zu einer Funktion im Produkt-Controlller.

Rechts ist die Update-Funktion ausgeklappt. Wir sehen hier die Parameter die in der Annotation definiert wurden.

Unten links sind noch die Möglichen Rückgabewerte abgebildet.

~~Kontrollieren~~

Auswerten

Zum Auswertungsteil.

Vorgaben mit Produkt vergleichen

Ich habe mich bei der Umsetzung an den Vorgaben aus dem Arbeitsauftrag orientiert. Wobei ich alle Vorgaben abdecken konnte.

Fazit

Zum Schluss noch mein Fazit zu diesem Projekt.

Reflexion

Ich nehme aus diesem Projekt einiges mit und hallte es für eine wertvolle Erfahrung. Sowohl Dinge die noch verbesserungswürdig sind, als auch Dinge mit denen ich sehr zufrieden war.

Auf die Seite von verbesserungswürdig, würde ich sicher die Zeitplanung aber auch das Zeitplanerstellen selbst stellen. Zum einen habe ich mich in einigen Punkten ziemlich verschätzt, was den Zeitaufwand angeht. Der grösste Punkt dabei war das Testkonzept. Ich würde mir für das erstellen des Zeitplans aber auch mehr Zeit lassen.

Auf der positiven Seite sehe ich die Entwicklung nach Laravel-Standards an. Ich hatte den Eindruck, dass mir Laravel und die Laravel-Dokumentation einen roten Faden bot. Alles war sehr logisch und sauber. Bis auf das Route-Modell-Binding musste ich etwas machen das sich wie Flick anfühlte. Vielleicht noch die Bilder Speicherung ein bisschen.

Ende

Das beendet meine Präsentation.