Dokumentation zur Arbeit von Serhat Güler im Praktikum Software Engineering im Rahmen des Projektes „CodeAcademy“

Einführung

Im Projekt namens „CodeAcademy“, welches ich mit meiner Gruppe „Engineer-Trio“ durchgeführt habe, habe ich stark bei der Entwicklung vom Backend und Frontend beigetragen. Bei meiner Arbeit im Projekt handelt es sich um die Authentifizierung der Nutzer, dem „Next Lesson“ Knopf im Frontend, einige Methoden in den Controllern sowie entsprechende Anpassungen in diesen bspw. Fehlerbehebungen sowie die Implementierung eines Badword Filters um unpassende Usernamen nicht zuzulassen. Zusätzlich habe ich Verbesserungen in der Datenbank vorgenommen, die davor einfach für das Vorhaben falsch geplant wurden. Ein Beispiel wäre, dass nur text statt longtext genommen wurde und wir schnell bei den Lektionsbeschreibungen ans Limit gestoßen sind. Im Frontend habe ich den eingebauten Code Editor implementiert und diesen in seiner Funktionalität ausgeweitet, damit dieser Zeilenangaben anzeigen kann. Als Letztes habe ich noch einen HTML-Tag Parser eingebaut, um eine Möglichkeit zu bieten, die Beschreibungen der Lessons aufzuhübschen mit schönen Formatierungen. Neben der Programmierarbeit habe ich den Python Kurs verfasst. Die Produktionsversion, die unter https://code-academy.online für einige Tage erreichbar war während der Präsentation, wurde von mir eingerichtet.

Inhaltsverzeichnis

1. Meine Rolle im Team

Dieses Schreiben gibt meine Sichtweise auf diese Thematik wieder.

Anfangs gab es bedauerlicherweise einige Komplikationen in unserer Gruppe, die wir zum Glück beilegen konnten und somit Schlimmeres verhindert wurde, wie beispielweise die Auflösung des Teams. Ich hatte anfangs privat leider viel Stress und auch gesundheitliche Probleme. Das Corona-Virus hatte mich schwer erwischt und ordentlich in meinem Körper gewütet, obwohl ich relativ frisch zweifach geimpft war und eigentlich sonst ein gutes Immunsystem habe. Unter anderem aus diesen Gründen konnte ich leider anfangs nicht jeder Pflicht nachkommen. Als sich später verbessert hatte, ging es endlich in unserem Projekt bergauf und das in einer erstaunlichen Zeit, da wir als ganze Gruppe viel Herzensblut und Zeit reingesteckt haben. Am Ende kam meiner Ansicht nach ein sehr gutes Produkt heraus. Ich empfand meine Rolle im Team als „Leading Developer“, da komplexe Entwicklungsaufgaben oft von meiner Seite gelöst worden sind.

2. Meine Arbeit

Aus programmtechnischer Sicht stammt einiges des bestehenden Produktes vor allem aus meiner Hand. Im Folgenden werden gemäß dem Inhaltsverzeichnis die Funktionen, die ich gemacht habe und zu denen ich stark beigetragen habe, beschrieben.

2.1 Backend

Das Backend, was einfach der Server bedeutet, der Daten validiert, verarbeitet und wiedergibt, wurde mithilfe des PHP-Frameworks Laravel verwirklicht. Dieses bringt eine Reihe an hilfreichen Komponenten mit, die bereits realisiert wurden und sehr einfach einbaubar sind im Produkt – nach dem Motto: „Don’t reinvent the wheel“. Im Deutschen bedeutet dies, das Rad nicht erneut zu erfinden, sondern einfach bestehende Konzepte zu nutzen und vor allem aus Effizienzgründen auszunutzen. Das tut man mit Laravel, da beispielweise die Authentifizierung größtenteils bereits eingebaut ist oder auch die Möglichkeit, mit der Datenbank zu kommunizieren. Für externe „packages“ (Pakete) haben

wir composer verwendet als Dependency-Manager (Abhängigkeiten-Manager), da es unserer Ansicht nach der beste Manager für PHP ist.

2. 1. 1 Authentifizierung der Nutzer auf der Serverseite

Da unser gesamtes Produkt wie bereits erwähnt auf Laravel aufbaut, basiert die Authentifizierung, die ich eingebaut habe, ebenfalls darauf. Was ich umkonfiguriert habe, ist, dass der Hash des Passwortes, also die Verschlüsselung des Passwortes, mit argon2id statt bcyrpt stattfindet. Argon2id ist deutlich schneller, aber auch schwerer zu brechen als bcrypt und hat auch den Wettbewerb der besten bzw. sichersten Hashing-Algorithmen als klarer Sieger für sich gewonnen. In Laravel stellt man dafür ganz einfach eine Zeichenkette in der Konfigurationsdatei namens auth.php von „bycrpt“ auf „argon2id“ um und das war es schon. Das war sehr einfach, da bereits alles von Laravel gestellt wurde. Der komplexere Teil war, dies in eine sogenannte Route zu packen, die Eingabedaten der Nutzer zu validieren und die Authentifizierungs-API von Laravel anzusprechen mit der definierten Methode innerhalb der Route. Laravel hat auch eine REST API bzw. einen RESTful-Webservice, in dem man als Entwickler seine Router definieren kann, die auf Methoden innerhalb Controller zeigen und diese ausführen. Ich brauchte für eine Authentifizierung zwei Routen, diese wären register und Log-in. Ersteres logischerweise für das Registrieren und Letzteres für das Einloggen. Die Register-Route zeigt auf die Methode register im sogenannten UserController. Ein Controller ist eine Klasse, die Web-Anfragen verarbeitet. In dem UserController werden alle Anfragen verarbeitet, die etwas mit dem Nutzer zu tun hat, im LessonController alle, die etwas mit einer Aufgabe bzw. Lerneinheit zu tun haben usw. Im UserController wurden nun die Funktionen bzw. Methoden namens register und Log-in geschaffen, welche als Argument eine Variable der Klasse Request erhalten. Die Klasse Request ist wieder etwas von Laravel, in dem die Daten der jeweilig eingegangen Anfrage gehalten werden und mit der man ein paar hilfreiche Funktionen hat, um beispielweise zu überprüfen, ob ein Wert in der Anfrage mit übermittelt wurde oder nicht. Dementsprechend kann man handeln als Entwickler. Als sinnvollste Auswahl, wenn notwendige Felder in der Anfrage fehlen, wären es diese mit einer entsprechenden Begründung zurückzuweisen. Das überprüfe ich allerdings nicht händisch, sondern nehme wieder ein geniales Produkt von Laravel, die Klasse Validator. Diese kann Anfragen validieren, in dem man in einem einfachen, aber starken Syntax Regeln definieren kann, wie zum Beispiel, dass der Wert eine Nummer sein muss oder eine Zeichenkette mit einer Länge von sechs Zeichen. Wenn der Validator die Anfrage nicht validieren kann, weil Daten fehlen oder diese falsch übermittelt worden sind, wird die Anfrage mit der entsprechenden Begründung des Validators

zurückgewiesen. Wenn die Anfrage validiert wurde, wird das Passwort des Nutzers verschlüsselt und mithilfe eines sogenannten User-Modells, welches die Attribute der User beschreibt, ein User-Eintrag in der Datenbank erstellt. Das besondere an den Modellen ist, dass man nicht direkt dank der sog. Laravel Eloquent API mit der Datenbank zu tun hat und keine manuellen SQL-Eingriffe durchführt. Höchstens definiert man mit der Eloquent API seine Anfrage, beispielsweise etwas wie User::where(„id“, 1)->first(); um den Nutzer mit der ID eins zu bekommen. Nach dem der Usereintrag erstellt wurde, wird ein Token vom Server für den neu registrierten Account generiert und an den Client als Antwort gesendet. Ein Token ist ein Schlüssel zu einem Account.

Das Einloggen folgt einem ähnlichen Schema der Datenverarbeitung wie es die Methode beim Registrieren tut. Es werden aus der Anfrage das Feld email sowie password an die sog. Auth Klasse von Laravel weitergereicht, welche dann verifiziert, ob die eingegeben Daten existent und korrekt sind. Bei richtigem Login wird dem Client ein Token übermittelt. Tokens haben auch Verfallsdaten, zum Beispiel sind diese nach einer Woche nicht mehr gültig. Dann muss man sich erneut einloggen. Das ist eine Sicherheitsvorkehrung, falls der Token in falsche Hände gelangen sollte, damit diese den Token nicht lange nutzen können. Eine Woche ist zwar immer noch eine große Zeitspanne, aber lieber so als gar kein Ablaufdatum wie es Twitter für eine lange Zeit hatte. Wenn der eigene Token irgendwo entwendet wird, ist er nicht für immer verwendbar, sondern nur für eine Woche.

Als Letztes zur Authentifizierung gibt es noch das Logout, bei dem wird eine Anfrage mit Logout ohne jegliche Daten außer des Tokens im Header zur Identifikation übermittelt und dann im Anschluss widerrufen bzw. zerstört und somit unbrauchbar gemacht.

2. 1. 2 Routen

In Laravel werden REST APIs bzw. RESTful Webservices mithilfe von Routen beschrieben. Diese werden in der api.php Datei definiert. Routen sind eine Schnittstelle zwischen Web- und Anwendungslogik. In Laravel kann man Routen in Gruppen packen und beispielsweise ein Präfix vorsetzen, wie beispielsweise „user“. Dann fangen Routen innerhalb dieser Gruppe immer mit „/api/user/<route>“ anstatt „/api/<route>“ an. Das ergibt Sinn, um die Routen logisch zu untergliedern, auch aufgrund von technischen Aspekten. Beispielweise kann man mit einer Middleware definieren, dass nur eingeloggte Nutzer

diese Route benutzen dürfen. Eine Middleware ist eine Software zwischen der Route und dem Controller, welche bevor die Anfrage an den Controller gelangt verarbeitet wird - beispielsweise darauf, ob der Nutzer eingeloggt ist oder nicht. Die Middleware kann dann die Anfrage abweisen, beantworten oder an die nächste Middleware weiterleiten. Wenn keine mehr übrig ist, kommt diese im Controller an. Die meisten Routen und Middlewares habe ich definiert. Ich habe sogar eine eigene Middleware geschrieben und einige Routen eingruppiert.

2. 1. 3 Controller

Controller sind, wie bereits erwähnt, Klassen mit entsprechenden Methoden, die dafür zuständig sind Anfragen zu verarbeiten und zu beantworten. Die Anwendungslogik findet größtenteils in Controllern statt.

2. 1. 3. 1 Von mir verbaute Methoden

Ich habe in den Controllern die Methoden Login und Register verbaut, da ich für die gesamte Authentifizierung der Nutzer verantwortlich war. Zusätzlich habe ich noch ein extra eingebaut, eine Schnittstelle, um den Token zu verifizieren – nach Prüfung kann der Client somit abgelaufene Token feststellen und den Nutzer auffordern, sich erneut einzuloggen. Des Weiteren habe ich Schnittstellen eingebaut, welche nutzerrelevante Daten an den Client liefern, zum Beispiel view-profile. Diese teilt dem Client seinen Fortschritt mit, welche Errungenschaften erreicht worden sind, wie viel XP gesammelt wurden und welche Kurse der Nutzer abgeschlossen hat und wie weit er mit anderen Kursen ist. In den Routen habe ich eine Möglichkeit verbaut, die nächste Lektion aus der Datenbank auszulesen.

2. 2 Frontend

Das Frontend, auch als Client bekannt oder einfach gesagt: die Webseite wurde in React implementiert. React ist ein JavaScript-Framework, bei dem auch viel mit TypeScript gearbeitet wird. Mit React besteht die Möglichkeit Webanwendungen schnell zu erstellen. Durch die vielfältigen Möglichkeiten bietet React dem Anwender einige wichtige Werkzeuge in der Entwicklung. Das Beste an React ist meiner Meinung nach die strukturierte Weise, womit man übersichtlich programmieren kann ohne „Spaghetticode“ zu programmieren. Für externe Abhängigkeiten haben wir npm verwendet, da es meiner Ansicht nach der einfachste Dependency-Manager in der Nutzung ist.

2. 2. 1 Authentifizierung der Nutzer auf der Clientseite

Da die Entwicklung leider nicht auf Backend-Seite für einen sog. Fullstack Developer endet (Fullstack = Back- und Frontend Entwickler), muss das Frontend ebenfalls ausgebaut werden. Anderenfalls bringt einem die ganze Arbeit im Backend nahezu nichts. Für die Authentifizierung habe ich zwei einfache Seiten erstellt. Für das Registrieren eine, bei der man seinen gewünschten Nutzernamen, E-Mail und Passwort (doppelt, um Tippfehler zu vermeiden) eingibt. Für das Login habe ich ebenso eine Seite erstellt, bei der der Nutzer sich mit den Daten, die zur Registrierung benutzt worden sind, einloggt. Die Seiten schauen für den Normalnutzer sehr einfach aus, für mich aber auch. Jedoch finden im Hintergrund komplexere Prozesse statt. Wenn ich mich registriere, wird dem Backend eine Anfrage an die register-Schnittstelle versendet, die die eingegeben Daten validiert und uns (also dem Client) antwortet. Diese kann einen Fehler beinhalten, aber auch einfach nur uns mitteilen, dass alles glatt gelaufen ist. Im Falle eines Fehlers erhält der Nutzer auf der Webseite eine visuelle Nachricht, die beispielsweise sagt, dass dem Server das Passwort zu kurz ist oder die Passwörter nicht übereinstimmen. Wenn alles glatt gelaufen ist, wird der Token vom Backend im an den Client übergeben und dort gespeichert. Ein Client kann zum Beispiel Firefox sein, bei dem der Token dann gespeichert wird. Dieser wird für künftige Anfragen immer wieder angegeben, denn damit kann der Server den Nutzer authentifizieren und entsprechend zuordnen. Beim Login passiert genau dasselbe, außer dass sich kein Account registriert, sondern nur einloggt.

2. 2. 2 Der spezielle next Lesson Knopf

In jeder Lesson, also Lerneinheit innerhalb eines Kurses gibt es einen sogenannten Next Lesson Knopf, der vermeintlich sehr einfach ausschaut. Dies ist es aber nicht. Beim Drücken wird nicht auf die nächste Seite geleitet, sondern die bestehende Seite wird aktualisiert mit den Daten der nächsten Lektion. Falls man am Ende angelangt ist, wird man wieder auf die Kursübersicht weitergeleitet. Das Aktualisieren habe ich so eingebaut, dass beim Drücken eine Anfrage mit der Position der jetzigen Lesson übermittelt wird, damit der Server ermitteln kann, wo die nächste Position ist. Da durch Löschungen von Lektionen Lücken in der Datenbank entstehen können, kann man nicht einfach die Position inkrementieren, da das nicht immer richtig wäre. Der Server sucht nach der nächsten Position in der Datenbank und antwortet mit der neuen Position inklusive Daten zur Lektion. Also beispielsweise kann es vorkommen, dass es auf Position 1 und 4 etwas gibt, aber nicht zwischen 2 und 3. In diesem Fall kann dann ein „404 – Not found“ vorkommen, wenn man stumpf inkrementieren würde. Das würde passieren, weil die zwei leer ist. Nach dem die Anfrage durch ist werden dann die neuen Daten im Frontend gesetzt, der Code Editor wird geleert und es wird ganz hochgescrollt. Zusätzlich wird ein redirect effect erzeugt, in dem ich die URL manipuliere mit der neuen, richtigen Lektion. In Wirklichkeit wurde der Nutzer nie redirected.

2.2.4 Akuter Problemfall vor Präsentation: Zu viele Anfragen wegen unsauberem Code

Kurz vor der Präsentation hatten wir ein großes Problem: Der Server hat nicht mehr auf Anfragen reagiert, da er vom Client, also der Webseite zu viele Anfragen innerhalb von kurzer Zeit erhalten hat. Irgendwann dachte der Server, er erhält gerade eine DDoS-Attacke und hat somit Anfragen von der Webseite ignoriert, doch in Wirklichkeit wurde leider von unserem SCRUM-Mater Daniel unsauberer Code eingebaut. Das wäre eine peinliche Situation geworden, wenn diese in der Präsentation aufgetreten wäre. Nach Spurensuche habe ich gefunden, dass er die Daten innerhalb der Komponenten angefragt hat. Das war zum Beispiel bei der Kursansicht mit Recent Courses und All Courses sechs Kärtchen, die jeweils eine Anfrage gestellt haben. Das bedeutet, dass sechs Anfragen simultan gesendet worden sind. Bei den Anfragen ging es darum, wie viel XP der Nutzer hat. Das war extrem ineffizient und hat den Server auch sehr gestört, weshalb er irgendwann einfach nicht mehr antwortete (der Ratelimiter hat angeschlagen). Daraus resultierte schließlich, dass die Webseite nach ein paar Mal durchnavigieren nicht mehr ging. Das sollte natürlich nicht so sein. Das gleiche Problem war innerhalb der Kursansicht, die Lektionskärtchen hatten dasselbe gemacht und das sogar jeweils zwei Mal pro Karte um XP und ob der Kurs abgeschlossen wurde zu ermitteln. Bei zehn

Lektionen wurden 20 Anfragen gestellt und jeder Kurs hatte mindestens zehn Lektionen. Dort habe ich auch schnell entgegengelenkt und alles in eine Anfrage gepackt, damit der Server eine statt 20 Anfragen erhält. Danach hat alles schneller geladen und der Ratelimiter ist nicht angesprungen, da kein Verdacht auf DDoS mehr bestand.

3. Python Kurs

Den Python-Kurs innerhalb der Anwendung habe ich verfasst.

3. 1 Motivation

In meinem Python Kurs wollte ich schnell, viel Informationen mit wenig Text vermitteln. Das habe ich unter anderem durch Gebrauch von den HTML-Tags gemacht und beispielsweise durch Tabellen für eine schönere Darstellung. Somit konnte ich mehr Informationen mit einem geringen Textanteil anschaulicher gestalten. Mein Kurs hat sich von den anderen beiden in der Theorielast unterschieden. Das liegt daran, dass Python eine komplexere Geschichte hat und an sich spezieller ist.

3. 2. Ein wenig Theorie muss sein…

Ich mache mich zwar keine Fans mit dieser Aussage, aber so ist die bittere Realität: Theorie ist wichtig. Ohne die Theorie gäbe es nicht die ausgereifte Praxis.

3.3 Der gewünschte Lerneffekt

Der meinerseits angepeilte Lerneffekt in meinem Kurs ist schlichtweg, einfache, aber grundlegend wichtige Operationen aufzuzeigen, damit Lernende flott ein Programm schreiben können. Mit ein bisschen Theorie verstehen sie sogar, was im Hintergrund in Python eigentlich passiert.

4. Online-Inbetriebnahme der Produktionsversion der Webseite

Vor der Präsentation habe ich CodeAcademy für ein paar Tage in Betrieb genommen unter der Domain https://code-academy.online, damit Interessierte unsere Anwendung austesten können.

4.1. Server

Als Hosting-Maschine habe ich einen gemieteten Ubuntu VPS genommen mit 8GB RAM und einer 250 GB SSD. Als Webserver wurde nginx verwendet.

4.2. Domain

Die Domain war https://code-academy.online und wurde von dem Dienstleister STRATO bereitgestellt.

4.3. SSL-Zertifikat

Das SSL-Zerifikat für HTTPS habe ich mithilfe von certbot erlangt. Certbot erkennt automatisch wann das SSL-Zertifikat abläuft und verlängert es automatisch. Als Admin muss man sich dann um nichts mehr kümmern bezüglich SSL, da Certbot alles übernimmt.

4.5. Unterschied: Development und Production Build

Bei der Produktionsinbetriebnahme sollte man auf alle Fälle eine production build nehmen, statt die einfach und schneller kompilierbare development build. Die Development Build ist auf die Entwicklerbedürfnisse zugeschnitten und läuft dafür zwar schneller am Anfang und bietet mehr Möglichkeiten für Entwickler. Bei mehreren Nutzern bricht die Performanz erheblich ein. Genau das Gegenteil ist bei der Production Build, dort ist die Performanz sehr gut, dafür dauert das kompilieren etwas länger. Diese kann simultane Anfragen viel schneller und besser verarbeiten.

5. Selbstreflexion meiner Arbeit

Rückblickend empfinde ich, dass ohne alle Team-Mitglieder das Projekt nie so zustande gekommen wäre. Trotz anfänglicher Schwierigkeiten habe vor allem ich sehr viel bei der Realisierung des Projektes in der Entwicklungsphase beigetragen. Vom Backend bis zum Frontend stammt vieles von mir ab. Die Qualität des Gesamtpaketes habe ich stets durch diverse Maßnahmen gesichert. Fehler, die außerhalb meines Aufgabenbereiches waren und von anderen Teammitgliedern eingebaut worden sind habe ich behoben. Den anderen Gruppenmitgliedern habe ich bei deren Entwicklungsproblemen stets geholfen mit meiner Erfahrung im Web Development und sie wegweißend geleitet.