

Prüfungsleistung Web-Programmierung

WWI2019A

Ergänzende Erläuterungen und
Ehrenwörtliche Erklärung

an der Dualen Hochschule Baden-Württemberg
Heidenheim
in der Fakultät Wirtschaft
im Studiengang Wirtschaftsinformatik

eingereicht von

5223626

5951911

6310429

Semester: 2019/4

Abgabedatum: 05.02.2021

1 Allgemeines

Im vorliegenden Projekt sollte ein vollständiger Webservice zur Buchung und Verwaltung von Stromtarifen entwickelt werden.

Im Speziellen wurden dabei folgende Punkte umgesetzt:

1. CRUD für alle Entitätstypen (User, Customer, Order, Plan)
2. Grafische Oberfläche für alle Funktionalitäten eines Kunden:
 - Registrierung, Login, Logout
 - Tarifsuche
 - Bestellprozess
 - Verwaltung von eigenen Bestellungen
 - Verwaltung der persönlichen Daten
3. Grafische Oberfläche zur Verwaltung durch einen Admin:
 - Verwaltung von Benutzern, Kunden, Bestellungen und Tarifen
 - Export von relevanten Daten
 - Statistiken
4. REST Schnittstelle für externe Zugriffe (insbesondere Portale)

In den folgenden Kapiteln werden für das Backend und Frontend Vorgehensweisen bei der Entwicklung und getroffene Entscheidungen und Annahmen kurz dokumentiert und erklärt.

2 Backend

2.1 Vorgehen

Nach dem **DRY**-Prinzip (dont repeat yourself) wurde von Beginn der Entwicklung an darauf geachtet eine möglichst große Wiederverwendbarkeit der Funktionen zu erreichen. Vorallem „Input-Checker“, also die Validierung und Vollständigkeitsprüfung von Benutzereingaben, und die Überprüfung von Zugriffberechtigungen standen dabei im Fokus.

Im ersten Schritt wurden die einzelnen Entitäts-/Relationstypen implementiert. Zu Beginn gab es hierbei noch keine „Customer“, sodass alle persönlichen Informationen zu einem Benutzer gespeichert wurden. Dies stellte sich in den ersten Wochen der Entwicklung aber schnell als problematisch dar, da es möglich sein sollte mit einem Benutzer-Account mehrere (eigene) Kunden(-daten/-accounts) verwalten zu können. Also mussten sowohl Datenbank als auch Models angepasst werden, um dies abbilden zu können.

Nach und nach wurden dann die Routen und Funktionen zum **Erstellen**, **Abrufen**, **Bearbeiten** und **Löschen** erstellt.

Wichtig war es dem Team dabei genauestens auf Zugriffsberechtigungen zu achten. Beispielsweise sollte ein Admin durch GET /orders sämtliche Bestellungen, bzw. Bestelldaten erhalten, ein normaler Benutzer dagegen nur seine eigenen Daten.

Um dem Anwender es zu ermöglichen zu filtern, zu suchen und zu sortieren, wurde ein eigener „Query-Builder“ entwickelt, der aus Benutzereingaben in der URL den entsprechenden Query zur Datenbankabfrage generiert. Zur Authorisierung wurden JSON Web Token¹ mit den Informationen *id*, *email* und *is_admin* und dem Hashalgorithmus SHA-256 eingesetzt.

Diese Token muss dann bei allen Anfragen, die eine entsprechende Berechtigung benötigen, mitgesendet werden.

Im weiteren Verlauf der Entwicklung wurden erst alle GET, dann alle POST, hierauf alle POST und abschließend alle DELETE Routen umgesetzt. Damit war es möglich, Synergien und etwaig doppelt verwendeten Code direkt zu identifizieren und auszulagern.

Beim Löschen der Daten wurde bei jedem Entitäts/-Relationstyp Soft- und Hard-Delete gegeneinander abgewägt. Nur bei „User“ hat man sich für den Hard-Delete entschieden. Gegen Ende der Entwicklungs-/Projektzeit wurden im Backend nur noch einzelne Bugs behoben und kleine Funktionalitäten für die Verwendung im Frontend angepasst.

2.2 Allgemeines

Statt JavaScript wurde sich für TypeScript entschieden, um eine typensichere Entwicklung zu gewährleisten.

Des Weiteren wurde das ORM Sequelize² verwendet. Modellbildungen und spätere Datenbankzugriffe wurden damit erheblich vereinfacht. Außerdem waren Datenbank-Migrationen und Seeder hiermit einfach umsetzbar und erforderten nur an sehr wenigen Stellen das explizite Schreiben eines SQL-Querys.

Um eine einfache Installation und gekapselte Ausführung zu ermöglichen wurde Docker genutzt.

¹<https://jwt.io>

²<https://sequelize.org/master/>

3 Frontend

3.1 Umsetzung

Das Frontend wurde mit Angular und Tailwindcss umgesetzt. Dabei wurde auf ein responsive Gestaltung der Seite Wert gelegt. Sämtliche Seiten, welche Server-Daten anzeigen, laden diese asynchron nach über den `ApiService`. Dabei war es dem Team wichtig, dass dem Nutzer durch eine Ladeanimation gezeigt wird, dass Daten aktuell geladen werden. Komponenten, welche logisch zusammengehören, wurden zu einzelnen Modules gebündelt, und werden bei Bedarf automatisch nachgeladen, um ein schnelleres initiales Laden zu ermöglichen. Außerdem werden so Module, welche vom Standardnutzer nicht benötigt werden (z.B. Adminseiten) gar nicht erst geladen. Um häufig verwendete Elemente der Seite wie etwa Buttons wiederverwenden zu können, wurde ein eigenes UI-Module erstellt. Beim Building der Angular-App wird ebenfalls das Stylesheet bereinigt, und CSS-Klassen, welche nicht verwendet werden, werden automatisch entfernt.

3.2 Startseite

Die Startseite beinhaltet ein Hintergrundbild, welches zufällig beim Laden der Seite gewählt wird und ein Eingabefeld, welches bei Drücken der Taste `Enter` | den Nutzer auf die Suchseite weiterleitet. Die gesuchte Postleitzahl wird dabei als Url-Routenparameter übergeben.

3.3 Tarifrechner und Buchung

Der Tarifrechner lädt die Tarifdaten der gewählten Postleitzahl und berechnet im Frontend die Kosten anhand des geschätzten Verbrauchs des Nutzers jeweils neu, um bei einer Änderung dieser jederzeit die Tarife neu nach günstigstem Preis zu sortieren. Klickt der Benutzer auf **Jetzt bestellen** so wird ihm die Bestellseite angezeigt. Diese wurde in mehrere Schritte eingeteilt. Um den Datenstand der Buchung zwischen Komponenten zu synchronisieren, werden die Buchungsdaten wie z.B. der aktuelle Tarif im `OrderService` zwischengespeichert.

3.4 Login und Benutzerprofil

Der Nutzer kann sich im Frontend mit seinen Nutzerdaten anmelden und registrieren. Ist dies erfolgreich, so wird durch den `LoginService` das vom Server generierte JWT in den `LocalStorage`³ gespeichert, um beim Neuladen der Seite ein automatisches eingeloggen zu ermöglichen. Anschließend wird in der Navigationsleiste ein Icon eingeblendet, welches beim Klicken den Nutzer zu seinem Profil leitet. Auf dieser Profilseite werden dem Nutzer seine aktuellen Tarifbuchungen und Kunden gezeigt.

³<https://developer.mozilla.org/de/docs/Web/API/Window/localStorage>

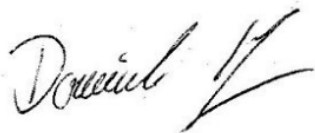
3.5 Admin

Der Adminbereich zeigt anschauliche Graphen und Diagramme, für diese wurde die Bibliothek `chart.js` verwendet. Sämtliche Datenbearbeitungsformulare wurden in Modals ausgelagert. Dafür wurde der `ModalService` verwendet, welcher einen Inhalt einer Adminkomponente (in diesem Fall das Formular) in die Modalkomponente projiziert. Der Export von Benutzerdaten wurde durch eine einfache Weiterleitung auf die entsprechende Serverroute umgesetzt. Da das Token lediglich über das Frontend abrufbar ist, wird dieses als Parameter weitergegeben. Für den Tarifimport wurde eine Drag-And-Drop-Komponente erstellt, welche eine Datei aufnehmen und durch mithilfe des `FileUploadService` auf den Server hochladen kann.

Ehrenwörtliche Erklärung

Wir versichern hiermit, dass unsere Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und erlaubten Hilfsmittel benutzt wurden. Die Arbeit wurde in gleicher oder ähnlicher Form bei keiner anderen Prüfung vorgelegt.

Heidenheim, 04.02.2021



Lisa Fremdt