Entwicklung eines interaktiven Kundeninformationssystems für eine Bäckerei

Studienarbeit

Gregory Seibert

31. März 2019

Matrikelnummer, Kurs 9234269, TINF-16-ITA

Betreuer Enrico Keil

Name, Vorname: Seibert, Gregory
Matrikelnummer: 9234269
Studiengang/Kurs: TINF-16-ITA
Titel der Arbeit: Entwicklung eines interaktiven Kundeninformationssystems für eine Bäckerei

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Falls sowohl eine gedruckte als auch elektronische Fassung abgegeben wurde, versichere ich zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Unterschrift

Abstract

Inhaltsverzeichnis

Glossar

Backend Das Backend verwaltet die Daten.

DELETE Das Löschen eines bestehenden Datenobjektes.

Frontend Das Frontend stellt die notwendigen Informationen dar und ist zur Interaktion mit dem Benutzer.

GET Das Abfragen eines bestehenden Datenobjektes.

POST Das Erstellen eines neuen Datenobjektes.

PUT Das Aktualisieren eines bestehenden Datenobjektes.

Akronyme

ACID	Atomicity, Consistency, Isolation and Durability.
API	Application Programming Interface.
HTTP	Hypertext Transfer Protocol.
JSON	JavaScript Object Notation.
MVC	Model View Controller.
MVCC	Multi-Version Concurrency Control.
REST	Representational State Transfer.

Abbildungsverzeichnis

1 Einleitung

1.1 Stand der Technik

1.2 Zielsetzung

2 Theoretische Grundlagen

In diesem Kapitel werden die notwendigen Grundlagen der Technologien, die für das Backend und die iOS App notwendige sind, vermittelt.

2.1 REST API

Representational State Transfer (REST) Application Programming Interfaces (APIs) spielen mittlerweile eine große Rolle in der Entwicklung und finden bereits in zahlreichen Projekten namhafter Unternehmen Anwendung. Diese APIs sind sogenannte Programmierschnittstellen über das Hypertext Transfer Protocol (HTTP). Mit deren Hilfe lassen sich Daten per GET, PUT, POST und DELETE abrufen und verwalten.

Der Sinn dahinter ist, dass man das Frontend, also das, was der Benutzer sieht, und das Backend, die Verarbeitung und Verwaltung der Daten, trennt. Dieses Vorgehen ist nicht nur sicherer, sondern auch im späteren Verlauf einfacher anzupassen, da man nun in modularen Gebieten des Projektes arbeiten kann. Deshalb bedienen sich große und bekannte Internetanbieter, wie Google und Amazon, diesem nützlichen Prinzip.

Um REST Schnittstellen komfortabel und zeitsparend testen zu können, bietet sich das Programm Postman [Posa] hervorragend an. Die Request-Methode und die URL müssen angegeben werden. Der Request-Body ist nur für Requests zum Anlegen oder Bearbeiten von Daten notwendig. Wie das Ganze in Postman aussieht, lässt sich der Abbildung 2.1 entnehmen.

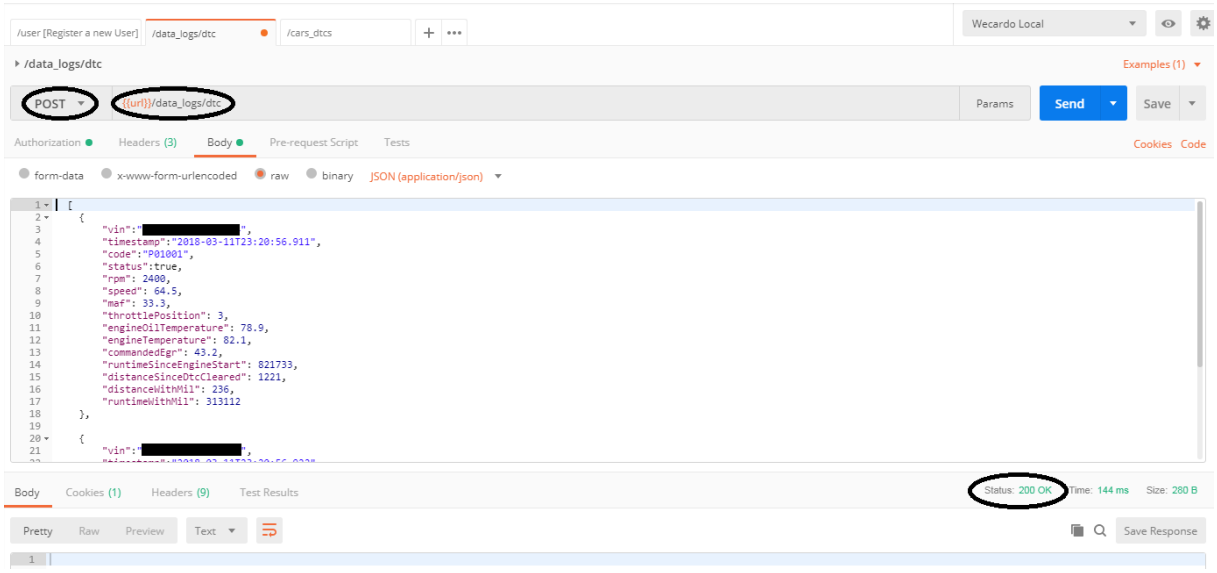

Abbildung 2.1: Das Programm Postman wird genutzt, um REST Schnittstellen zu testen

2.2 JSON

JavaScript Object Notation (JSON) ist ein leichtgewichtiges Format zum Austauschen von Daten, typischerweise zwischen Webserver und Client. Dieses Format wurde von der JavaScript Object Syntax abgewandelt, weswegen eine bidirektionale Umwandlung im JavaScript Code unkompliziert funktioniert. Doch auch in Programmiersprachen, wie zum Beispiel Java und Swift, gibt es standardmäßig eingebaute Funktionen, um JSON zu parsen. Die Syntax von JSON besteht aus folgenden Regeln:

- Arrays bestehen aus eckigen Klammern
- Objekte bestehen aus geschweiften Klammern
- Datenfelder von Objekten werden durch Key-Value Paaren dargestellt
- Datenfelder und Objekte werden durch Kommata getrennt

2.3 Dependency Injection

Dependency Injection ist ein Entwurfsmuster aus der objektorientierten Programmierung, welches dabei hilft, die Abhängigkeiten von Klassen zu organisieren [PP18, S. 27]. Die Abhängigkeiten werden an einem zentralen Ort einmalig instanziiert, identisch zum Singleton Pattern. Dabei werden die jeweiligen Klassen von ihren Abhängigkeiten entkoppelt und es wird verhindert, dass von ressourcen-intensiven Klassen mehrere Instanzen erzeugt wird. Des weiteren wird die Verwendung und Erstellung von Unit-Tests erleichtert.

Für die Umsetzung der Dependency Injection finden die drei Verfahren

- Constructor Injection
- Setter Injection
- Interface Injection

die meiste Verwendung.

Bei der Constructor Injection werden die Abhängigkeiten einer Klasse durch den Konstruktor übergeben und gesetzt [PP18, S. 119].
Durch jeweilige Methoden können die Abhängigkeiten bei der Setter Injection gesetzt werden [PP18, S. 120].
Die Interface Injection zeichnet sich dadurch aus, dass die abhängige Klasse eine Schnittstelle implementiert, durch die eine Methode vorgegeben wird, über die die Abhängigkeit zur Verfügung gestellt wird [PP18, S. 120].

2.4 Spring Framework

Das Spring Framework [Piv] ist ein Open-Source Framework für Java basierte Enterprise Projekte. Es wurde entworfen, um die Java EE Entwicklung zu vereinfachen und zu beschleunigen [Sch]. Spring implementiert Funktionalitäten wie

- Dependency Injection (siehe 2.3)
- Internationalisierung
- Datenbindung
- Validierung
- Typenkonvertierung

Insbesondere durch das Bereitstellen der Dependency Injection wird ein Einsatz von guten Softwarekonventionen gefördert, was für ein leicht wartbares und erweiterbares Projekt sorgt [Wol10, S. 20].

2.4.1 Spring Boot

Spring Boot ist eine Erweiterung zum Spring Framework mit dem Fokus auf Konventionen statt Konfiguration. Dadurch ist es deutlich schneller möglich, eine produktionsfähige Applikation zu erstellen, da bereits durch das Framework die meisten Entscheidungen getroffen wurden. Dennoch lassen sich jederzeit besagte Entscheidungen durch eine eigene Konfiguration überschreiben.

2.4.2 Spring Security

Spring Security erweitert das Spring Framework um einige Sicherheitsmechanismen, wie das Authentifizieren von Benutzern oder die Autorisierung in Form eines Rollenschemas mit verschiedenen Berechtigungen.

2.5 PostgreSQL

PostgreSQL, oft auch nur Postgres genannt, ist ein objektrelationales Datenbankmanagementsystem, welches als Open-Source Projekt angeboten wird. Es ist ein ehemaliges Projekt der „University of California, Berkeley“, das 1986 gestartet ist [Posb]. Als ein SQL-Interpreter im Jahre 1994 für Postgres geschrieben, woraufhin das gesamte Projekt als Open-Source unter dem Namen Postgres95 freigegeben wurde. Im Jahre 1996 wurde der aktuelle Name PostgreSQL gewählt, um die hinzugefügten SQL Fähigkeiten zu verdeutlichen [Posb].

2.5.1 Eigenschaften

Postgres ist Atomicity, Consistency, Isolation and Durability (ACID)-konform und fast vollständig mit dem SQL-Standard SQL:2011 konform, da mindestens 160 von 179 notwendige Funktionen erfüllt sind [Posc]. Es ist mit den Programmiersprachen C++, Delphi, Perl, Java, Lua, .NET, Node.js, Python, PHP, Lisp, Go, R, D und Erlang kompatibel [Bui]. Postgres unterstützt Multi-Version Concurrency Control (MVCC), ein Verfahren, um für eine gleichzeitige und Konsistenz wahrende Ausführung von konkurrierenden Zugriffen auf die Datenbank zu sorgen [Posc]. Die maximale Größe der Datenbanken wird entweder durch 32TB oder durch den tatsächlich verfügbaren Speicher begrenzt [Posc]. Postgres kann zudem durch selbst entworfene Funktionen, Operatoren und Datentypen erweitert werden und es unterstützt einige NoSQL Funktionen [Posc].

2.5.2 Vergleich zu MySQL

MySQL ist ebenfalls ein Open-Source Projekt. Es ist allerdings unter den relationalen Datenbankmanagementsystemen zu zählen. Genau wie PostgreSQL ist das oberste Element eine Tabelle. Die Funktionalitäten von MySQL und PostgreSQL sind beinahe identisch [Bui].

Zu den bekanntesten Unternehmen, die PostgreSQL als Datenbankserver verwenden, gehören unter anderen GitHub, US Navy, NASA, Tesla, YouTube, und Facebook verwenden MySQL als Datenbankserver [MyS]. Apple, Cisco, Skype, Uber, Groupon, Spotify, Netflix und Instagram verwenden PostgreSQL als Datenbankserver [Sta]. Auf beiden Seiten sind demnach größere und bekannte Unternehmen vertreten.

Da ein Vergleich der Performance stark abhängig von der jeweiligen Infrastruktur, des Einsatzzweckes und der Art der Abfragen ist, wird dies nicht weiter aufgeführt.

2.6 MVC Pattern

Das Model View Controller (MVC) Pattern ist ein Architekturmuster beziehungsweise Entwurfsmuster im Bereich der Software-Entwicklung. Hierbei wird das Programm in drei Schichten

- Model
- View
- Controller

aufgeteilt. Die Model Schicht enthält alle Klassen, die reale oder irreale Objekte modellieren, also jene, die nach der objektorientierten Programmierung entworfen wurden. Die View Schicht enthält die Oberfläche mit den Elementen zur Interaktion mit der Software. Da die Model Schicht und die View Schicht miteinander kommunizieren müssen und da die Eingaben des Benutzers verarbeitet werden müssen, ist die Controller Schicht gleichermaßen essenziell. Dieser Aufbau ist, wie beschrieben, in der Abbildung 2.2 ersichtlich.

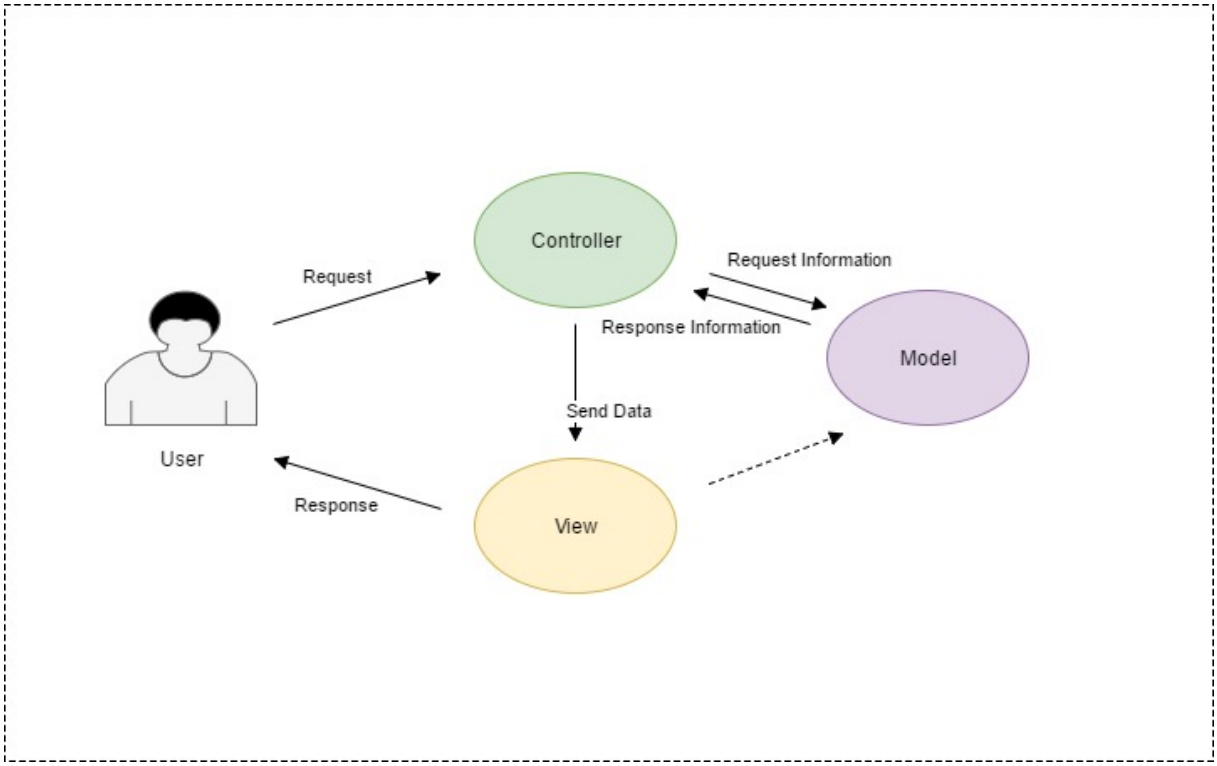

Abbildung 2.2: Der Aufbau des MVC Patterns [Cho]

Der Sinn dieses Architekturmusters ist eine lockere Kopplung der einzelnen Software-Module, um die Abhängigkeiten zu verringern und den Wartungsprozess, Erweiterungsprozess sowie Aktualisierungsprozess zu verbessern.

2.7 Swift

Swift ist eine Open Source Programmiersprache, die im Jahre 2014 von Apple veröffentlicht wurde. zum Entwickeln von iOS, MacOS und Linux Applikationen.

3 Anforderungsanalyse

3.1 Begriffsdefinitionen

Im Folgenden werden die Begriffe, die für die funktionalen und nichtfunktionalen Anforderungen notwendig sind, definiert.

3.1.1 Daten

Bei dem Begriff „Daten“ handelt es sich um Informationen zu Backprodukten oder Neuigkeiten.

3.1.2 System

Mit dem Begriff „System“ ist die Verwaltung für die hinterlegten Daten gemeint.

3.1.3 Administrator

Mit dem Begriff „Administrator“ ist die Person, die für das Pflegen der Daten zulässig ist, gemeint.

3.1.4 Benutzer

Mit dem Begriff „Benutzer“ ist eine Technologie zur Anzeige der angelegten Daten gemeint.

3.2 Funktionale Anforderungen

3.2.1 [FA10] Zentraler Speicherort

Das System muss die Daten zentral in einer Datenbank speichern.

3.2.2 [FA20] Zentrale Administration

Das System muss dem Administrator über eine zentrale Schnittstelle die Möglichkeit bieten, die Daten pflegen zu können.

3.2.3 [FA30] Alle Backprodukte anzeigen

Das System muss dem Benutzer und dem Administrator die Möglichkeit bieten, die Daten zu allen Backprodukten, sofern bereits welche angelegt wurden, anzeigen lassen zu können.

3.2.4 [FA40] Backprodukt anzeigen

Das System muss dem Benutzer und dem Administrator die Möglichkeit bieten, die Daten zu einem ausgewählten Backprodukt, sofern dieses bereits angelegt wurde, anzeigen lassen zu können.

3.2.5 [FA50] Backprodukt erstellen

Das System muss dem Administrator die Möglichkeit bieten, ein neues Backprodukt anlegen zu können.

3.2.6 [FA60] Backprodukt bearbeiten

Das System muss dem Administrator die Möglichkeit bieten, ein bestehendes Backprodukt bearbeiten zu können.

3.2.7 [FA70] Backprodukt löschen

Das System muss dem Administrator die Möglichkeit bieten, ein bestehendes Backprodukt löschen zu können.

3.2.8 [FA80] Alle Neuigkeiten anzeigen

Das System muss dem Benutzer und dem Administrator die Möglichkeit bieten, die Daten zu allen Neuigkeiten, sofern bereits welche angelegt wurden, anzeigen lassen zu können.

3.2.9 [FA90] Neuigkeit anzeigen

Das System muss dem Benutzer und dem Administrator die Möglichkeit bieten, die Daten zu einer ausgewählten Neuigkeit, sofern diese bereits angelegt wurde, anzeigen lassen zu können.

3.2.10 [FA100] Neuigkeit erstellen

Das System muss dem Administrator die Möglichkeit bieten, eine neue Neuigkeit anlegen zu können.

3.2.11 [FA110] Neuigkeit bearbeiten

Das System muss dem Administrator die Möglichkeit bieten, eine bestehende Neuigkeit bearbeiten zu können.

3.2.12 [FA120] Neuigkeit löschen

Das System muss dem Administrator die Möglichkeit bieten, eine bestehende Neuigkeit löschen zu können.

3.3 Nichtfunktionale Anforderungen

3.3.1 [NFA10] Authentifizierung zur Administration

Das System muss die Administration von Daten bei einem nicht autorisierten Zugriff verweigern, sofern es sich nicht um den Administrator handelt.

3.3.2 [NFA20] Authentifizierung zur Anzeige von Daten

Das System muss die Anzeige von Daten bei einem nicht autorisierten Zugriff verweigern, sofern es sich nicht um den Benutzer handelt.

3.3.3 [NFA20] iOS App mit iPad Kompatibilität

Das System muss eine iOS App mit iPad Kompatibilität bereitstellen, um die Daten für Endnutzer anzeigen lassen zu können.

4 Konzept

4.1 Softwarearchitektur

Die funktionalen Anforderungen [FA10] Zentraler Speicherort sowie [FA20] Zentrale Administration fordern eine zentrale Datenverwaltung. Um dies gewährleisten zu können, wird das System in zwei Komponentengruppen, dem Backend und das Frontend, aufgeteilt. Das Backend soll eine Anbindung zu einer Datenbank besitzen, um neue sowie bereits vorhandene Daten speichern zu können. Damit der Administrator die Daten möglichst komfortabel pflegen kann und es zusätzlich dem Administrator und dem Benutzer möglich sein soll, die Daten einzusehen, ist ein Frontend notwendig. Die Verwaltung der Daten durch den Administrator soll hierbei über ein Web Admin-Dashboard geschehen. Die Visualisierung der Daten als Information soll über eine iOS Applikation, gemäß der nichtfunktionalen Anforderung [NFA20] iOS App mit iPad Kompatibilität, ermöglicht werden. Diese Softwarearchitektur ist in der folgenden Abbildung 4.1 als UML Komponentendiagramm dargestellt.

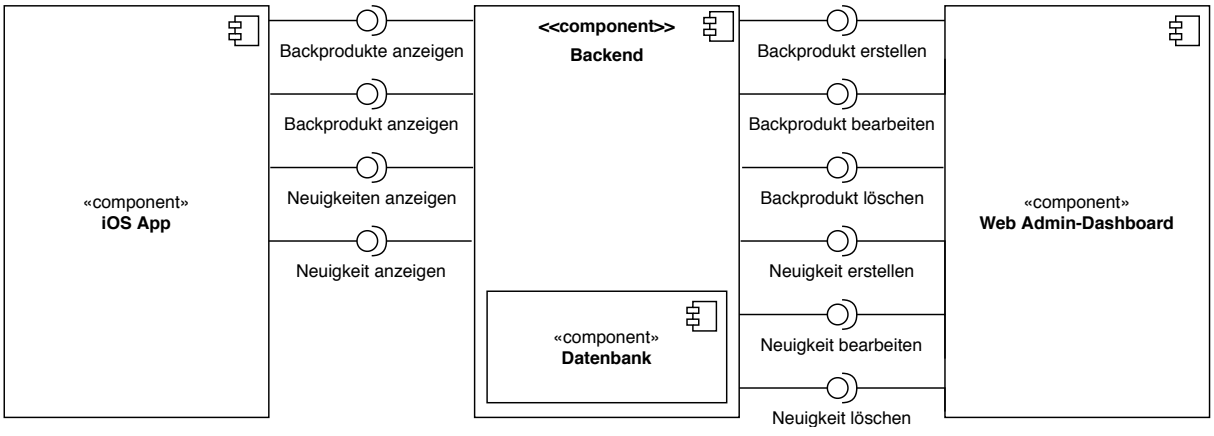

Abbildung 4.1: Das UML Komponentendiagramm für das System, bestehend aus Backend, iOS App und Web Admin-Dashboard

4.2 Rollenmodell

Die funktionale Anforderung [FA20] Zentrale Administration setzt voraus, dass der Administrator die Daten über eine zentrale Schnittstelle pflegen kann. Dabei muss dem Administrator die Möglichkeit gegeben werden, nach den funktionalen Anforderungen [FA50] Backprodukt erstellen, [FA60] Backprodukt bearbeiten, [FA70] Backprodukt löschen, [FA100] Neuigkeit erstellen, [FA110] Neuigkeit bearbeiten, und [FA120] Neuigkeit löschen, Backprodukte und Neuigkeiten erstellen, bearbeiten und löschen zu können. Des Weiteren muss es sowohl dem Administrator als auch dem Benutzer des Systems, gemäß den funktionalen Anforderungen [FA30] Alle Backprodukte anzeigen, [FA40] Backprodukt anzeigen, [FA80] Alle Neuigkeiten anzeigen und [FA90] Neuigkeit anzeigen, möglich sein, sich die Daten zu den Backprodukten und Neuigkeiten anzeigen zu lassen. Hierbei ist nach den nichtfunktionalen Anforderungen [NFA10] Authentifizierung zur Administration und [NFA20] Authentifizierung zur Anzeige von Daten zu beachten, dass die Administration der Daten lediglich bei Existenz einer autorisierten Sitzung des Administrators und das Abfragen der Daten lediglich bei Existenz einer autorisierten Sitzung des Benutzers erlaubt sein darf.
Hieraus folgt, dass es analog zwei Rollen geben soll. Zum einen der Benutzer und zum anderen der Administrator, welcher die für den Benutzer verfügbaren Funktionen erbt. Beiden soll es möglich sein, sich am System anzumelden. Dies ist in der folgenden Abbildung 4.2 als UML Usecase Diagramm visualisiert.

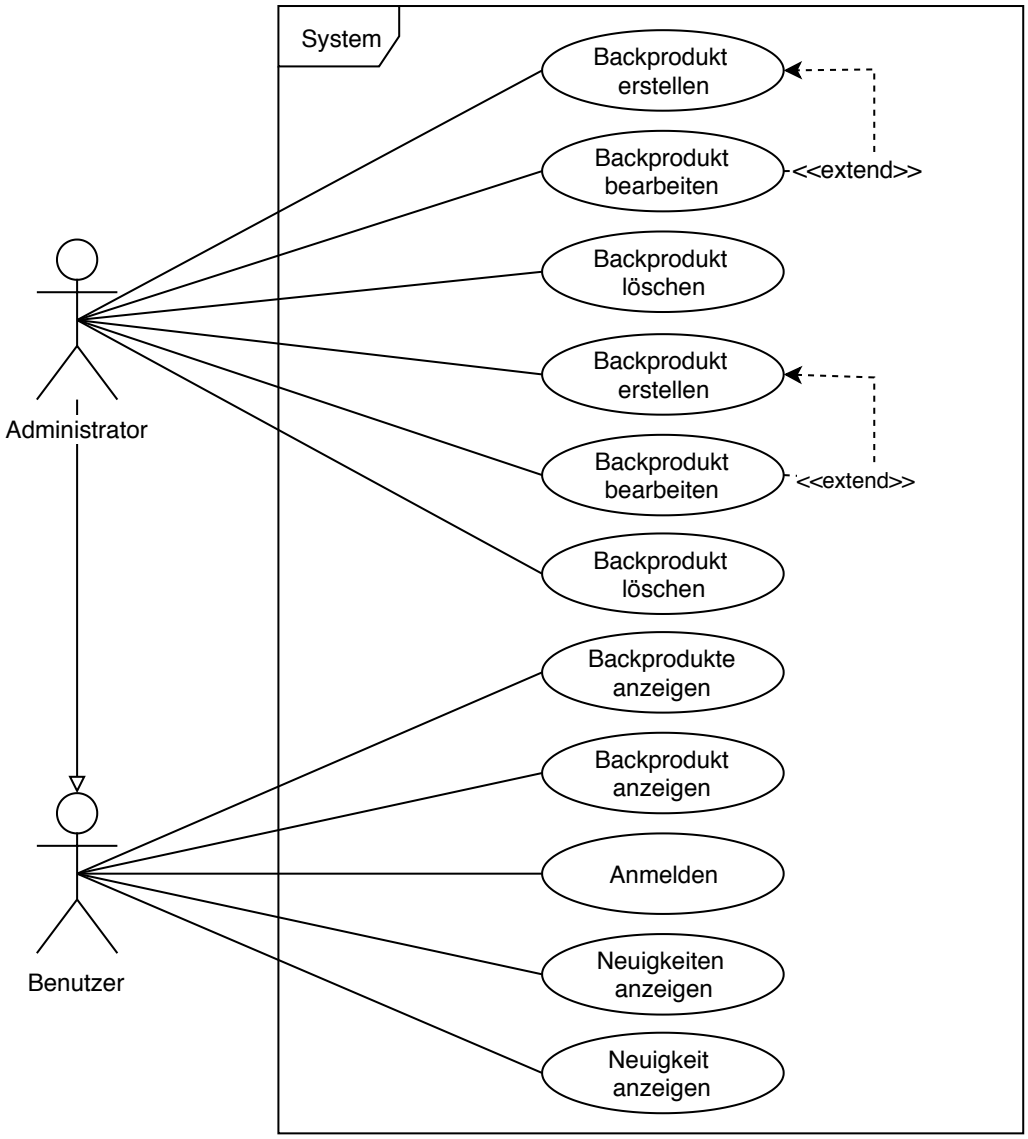

Abbildung 4.2: Das UML Usecase Diagramm für das System und als Aktor der Administrator sowie Benutzer zur Verdeutlichung des Rollenmodells

4.3 Datenmodell

Im Kapitel 4.2 wurde beschrieben, wie das Rollenmodell des Systems als Entwurf aussehen soll. Im Rahmen des Datenmodells ist hierbei eine Entität, der „ApplicationUser“ notwendig, siehe Abbildung 4.3. Diese Entität besitzt die Attribute „username“ für den Benutzernamen sowie „password“ für das Passwort zur Anmeldung, „name“ für den Namen der jeweiligen Person, „isActive“ für die Möglichkeit zur Deaktivierung und „isArchived“ für die Archivierung des dahinterstehenden Accounts. Des Weiteren besitzt die Entität „ApplicationUser“ eine Liste an „ApplicationUserRole“, wodurch das Rollenmodell vollständig implementiert ist.

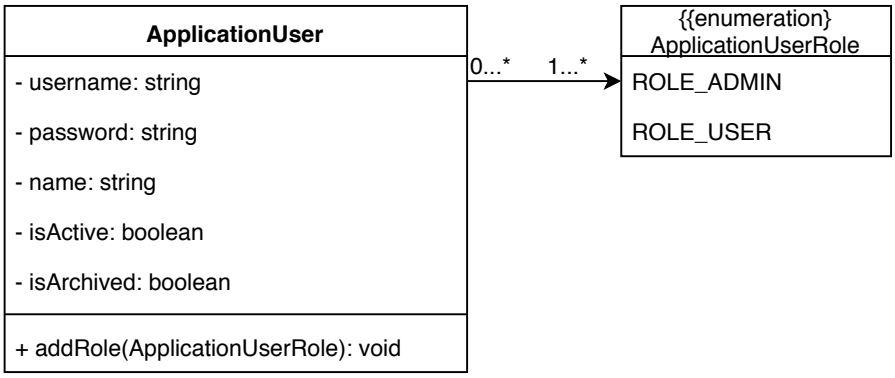

Abbildung 4.3: Das UML Klassendiagramm zur ApplicationUser Entität

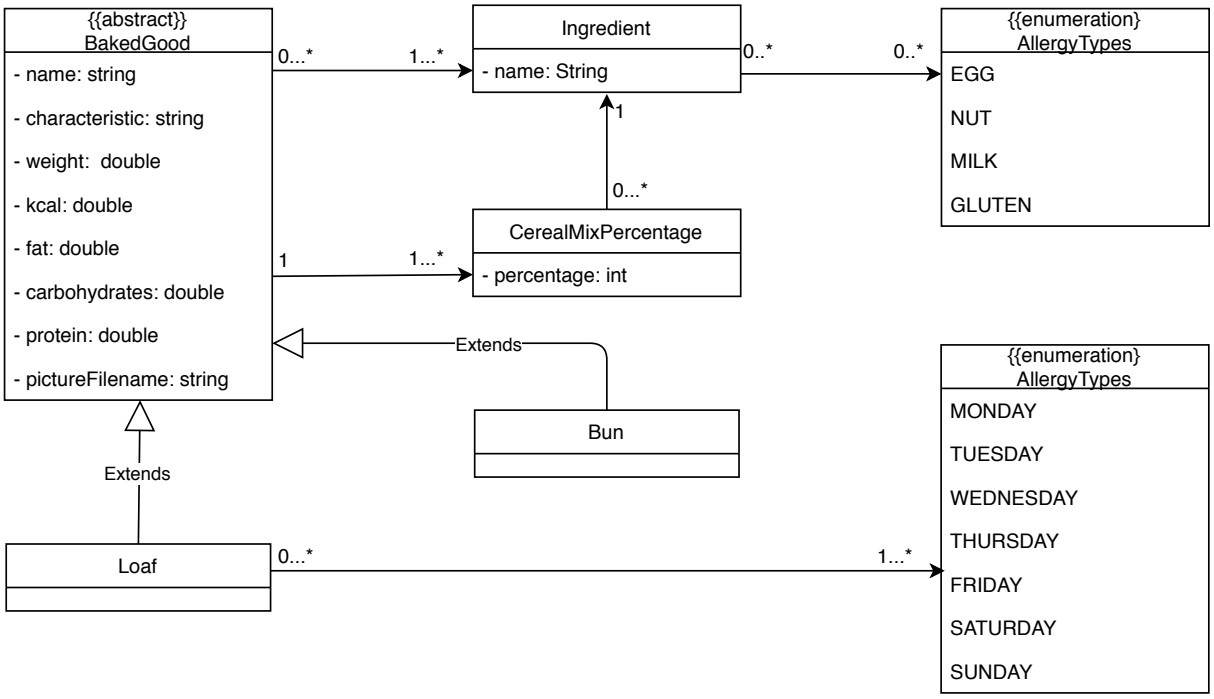

Abbildung 4.4: Das UML Klassendiagramm zur BakedGood Entität

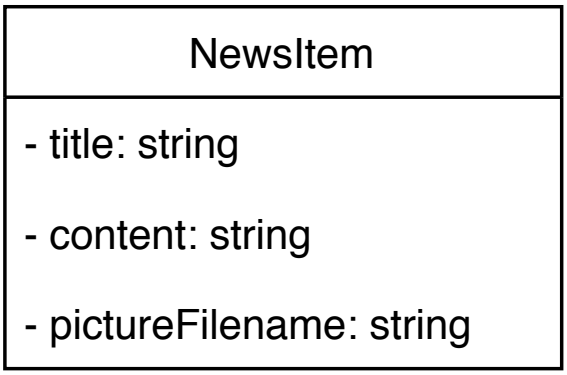

Abbildung 4.5: Das UML Klassendiagramm zur NewsItem Entität

4.4 Sicherheit

5 Implementierung

5.1 Backend

5.2 iOS Applikation

5.3 Web Admin-Dashboard

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

6.2 Ausblick

Literatur

[Bui] An Bui. *PostgreSQL vs MySQL*. URL: https://blog.panoply.io/postgresql-vs.-mysql (besucht am 23. 08. 2018).

[Cho] Kishan Choudhary. *Difference Between MVC And Web Forms*. URL: https://www.c-sharpcorner.com/article/difference-between-mvc-and-web-forms/ (besucht am 20. 08. 2018).

[MyS] MySQL. *MySQL Customers*. URL: https://www.mysql.com/customers/ (besucht am 23. 08. 2018).

[Piv] Pivotal. *Spring Framework*. URL: https://spring.io/ (besucht am 27. 08. 2018).

[Posa] Inc. Postdot Technologies. *Postman*. URL: https://www.getpostman.com/ (besucht am 03. 09. 2018).

[Posb] PostgreSQL. *A Brief History of PostgreSQL*. URL: https://www.postgresql.org/docs/current/static/history.html (besucht am 22. 08. 2018).

[Posc] PostgreSQL. *About PostgreSQL*. URL: https://www.postgresql.org/about/ (besucht am 22. 08. 2018).

[PP18] Nilang Patel und Krunal Patel. *Java 9 Dependency Injection*. Packt Publishing Ltd, 2018. ISBN: 9781788296250.

[Sch] Sebastian Schelter. *Das Spring Framework – eine Einführung*. URL: http://www.inf.fu-berlin.de/inst/ag-se/teaching/S-BSE/134_schelter_spring.pdf (besucht am 29. 08. 2018).

[Sta] Stackshare. *Companies that use PostgreSQL*. URL: https://stackshare.io/postgresql (besucht am 23. 08. 2018).

[Wol10] Eberhard Wolff. *Spring 3 - Framework für die Java-Entwicklung*. dpunkt.verlag, 2010. ISBN: 978-3-89864-572-0.