



Erstellen eines Webshops

Studienarbeit

an der dualen Hochschule Baden Württemberg

von

Jens Gerle und Yaiza Gonzalo Alt

1. Juni 2017

Zeitraum

Oktober 2016 - Juni 2017

Matrikelnummern

2825585 (Jens Gerle)

1476581 (Yaiza Gonzalo Alt)

Kurs

TINF14B

Betreuer

Dominik Rietz

Gutachter

Dominik Rietz

Inhaltsverzeichnis

Abkürzungsverzeichnis	I
Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Quellcodeverzeichnis	V
Ehrenwörtliche Erklärung	VI
1 Einleitung	1
1.1 Aufgabenstellung	1
1.2 Projektziele	2
2 Planung	3
2.1 Projektphasen	3
2.2 Meilensteine	3
2.3 Aufgabenverteilung	4
3 Spezifikation der Anwendung	5
3.1 Anforderungen	5
3.2 Nichtfunktionale Anforderungen	5
3.3 Hauptanwendungsfälle	6
3.4 System-Abgrenzung	10
3.5 Inhaltliche Definition	11
4 Architektur von Webanwendungen	12
4.1 Serverseite	13
4.2 Clientseite	15
5 Architekturentscheidung	18
5.1 Spring Boot Anwendung	18
5.2 Angular 2 Frontend	19
5.3 REST Services	24
5.4 MySQL Datenbank	25
5.5 Hibernate ORM	26
6 Entwurf	27
6.1 Datenmodell	27
6.2 Design der Weboberfläche	29

7	Umsetzung	31
7.1	Initiales Aufsetzen der Anwendung	31
7.2	Backend-Klassen	34
7.3	Objektrelationale Abbildung (ORM) mit Hibernate	35
7.4	REST-Schnittstellen	38
7.5	Weitere Implementierungsdetails	40
7.6	Frontend	42
8	Deployment	47
8.1	Versionsverwaltung mit Git	47
8.2	Travis Continuous Integration	47
8.3	Heroku	48
8.4	AWS Datenbank Instanz	49
9	Fazit	50
9.1	Hürden und Stolpersteine	50
9.2	Ausblick	51
9.3	Kritische Würdigung	52
	Literatur	54
	Anhang A Meilensteine	57
	Anhang B Design-Mockups	58
	Anhang C Travis YAML-Datei	62

Abkürzungsverzeichnis

AJAX Asynchronous JavaScript and XML

AWS Amazon Web Services

Blob Binary Large Object

BSI Bundesamt für Sicherheit in der Informationstechnik

CRUD Create Read Update Delete

CSS Cascading Style Sheets

DAO Data Access Object

DOM Document Object Model

EJB Enterprise Java Bean

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

ISS Internet Information Server

JAXB Java Architecture for XML Binding

JAX-RS Java API for RESTful Web Services

JEE Java Enterprise Edition

JPA Java Persistence API

JSON JavaScript Object Notation

JWT JSON Web Tokens

MIME Multipurpose Internet Mail Extensions

NoSQL Not only SQL

NPM Node Package Manager

ORM Object Relational Mapping

POJO Plain Old Java Object

POM Project Object Model

REST Representational State Transfer

SPA Single Page Application

SQL Structured Query Language

URI Uniform Resource Identifier

XML Extensible Markup Language

Abbildungsverzeichnis

1	Anwendungsfalldiagramm mit den Hauptanwendungsfällen	6
2	Systemkontext	11
3	Drei-Schichten-Modell der Client-Server-Architektur	12
4	Lebenszyklus einer traditionellen Webanwendung	17
5	Lebenszyklus einer SPA	17
6	Grundlegende Architektur des Webshops	18
7	Arten des Data Bindings in Angular 2	22
8	Entity Relationship Modell	28
9	Mockup des Webshops	29
10	Mockup des Webshops Handy Version	30
11	Spring Initializr Weboberfläche	31
12	Ordnerstruktur des generierten Projekts	33
13	REST-Endpoints	39
14	Inhalt der Seite für die Kategorie Wolle	43
15	Bestandteile der Cloud-Servicemodelle	49

Tabellenverzeichnis

1	Projektmeilensteine	4
---	-------------------------------	---

Quellcodeverzeichnis

1	Beispiel Metadaten-Annotation für Komponenten	21
2	Root Modul	22
3	Beispiele zu den Data Binding Arten	23
4	Auszug der Datei angular-cli.json	34
5	CategoryDao.java	34
6	Vererbungshierarchie	37
7	REST-Beispiel	39
8	Routing-Module	42
9	Template für die Produklisten-Komponente	44
10	Methode für den Aufruf des Warenkorbes	44
11	Travis CI YAML-Datei	62

Ehrenwörtliche Erklärung

Wir erklären hiermit ehrenwörtlich, dass wir die vorliegende Arbeit selbständig angefertigt haben; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Stuttgart, 01.06.2017

Ort, Datum

A handwritten signature in blue ink, reading "Jens Gese".

Unterschrift

Stuttgart, 01.06.2017

Ort, Datum

A handwritten signature in blue ink, appearing to be "Klein".

Unterschrift

1 Einleitung

Die Ausbreitung des elektronischen Handels ging mit der Erfolgsgeschichte und Kommerzialisierung des Internets einher und wächst seitdem immer weiter. Für viele Endkunden ist der Einkauf von Waren in Online-Shops alltäglich geworden und bietet im Vergleich zum konventionellen Handel viele Vorteile.

Große Versandhäuser wie Amazon bieten weltweit ihre Dienste an und reagieren auf die sich ändernde Nachfrage ihrer Kunden. Wo früher Preisvergleiche und Informationsbeschaffung zu bestimmten Produkten einen erheblichen Zeitaufwand bedeuteten und nicht an einem Ort zu finden waren, bieten heute Suchmaschinen ihre Dienste an und ermöglichen Nutzern bequem von zu Hause aus diese Informationen abzurufen. Davon profitieren wiederum die Online-Händler, die dadurch Kunden gewinnen können. Einzelhändler und Kaufhäuser haben diese Entwicklung schmerzlich erfahren, da sich der Preisdruck erhöht hat und der E-Commerce zudem weniger Nebenkosten benötigt. Viele traditionelle Händler haben aber reagiert und ihr Geschäft durch eigene Online-Shops ergänzt [1].

Die Gestaltung und technischen Umsetzungsmöglichkeiten von Webshops haben sich mit der Zeit ebenso weiterentwickelt. Heute gibt es eine Vielzahl an Technologien und Herangehensweisen, um einen Online-Handel aufzusetzen und zu betreiben.

Diese Studienarbeit beschäftigt sich mit der Konzeption und Umsetzung eines solchen Webshops.

1.1 Aufgabenstellung

Die Erstellung eines Webshops mit der benötigten Infrastruktur ist die Aufgabenstellung der vorliegenden Arbeit. Dabei soll es einen Bereich für Benutzer (oder Kunden) für den Einkauf im Shop geben, sowie einen Bereich für Administratoren zur Verwaltung des Bestands und der Erfassung neuer Artikel.

Details zur Funktionalität und inhaltlichen Gestaltung finden sich in Abschnitt 3.

1. EINLEITUNG

1.2 Projektziele

Neben der Umsetzung der Anforderungen und Funktionalitäten für den Webshop gibt es weitere Ziele für das Projekt. Dazu gehört der Wunsch, neue Technologien zu evaluieren, einzusetzen und sich anzueignen, was auch in einem professionellen Umfeld immer wieder nötig ist, um dem schnellen Wandel und der Entwicklung der technischen Möglichkeiten gerecht zu werden.

Außerdem soll für die gemeinsame Projektarbeit auf eine strukturierte Arbeitsweise geachtet, sowie best practices herausgearbeitet werden, um einen möglichst reibungslosen und effektiven Ablauf des Projekts zu gewährleisten.

2 Planung

Für die Bearbeitung des Projekts ist ein Zeitraum von 21 Wochen vorgesehen, von KW43/2016 bis zum Abgabetermin am 2. Juni 2017. Die Planung der einzelnen Arbeitsschritte wird in den nächsten Abschnitten beschrieben.

2.1 Projektphasen

Das Projekt ist grundsätzlich in fünf Phasen unterteilt:

1. Evaluierung der möglichen Technologien und Architekturentscheidung. Die Abschnitte 4 und 5 beschreiben die Ergebnisse dieser Phase.
2. Vorbereitung des Projekts. Dazu gehören das Aufsetzen des Backends, der Datenbank und des Frontends, sowie die Verbindung aller Komponenten.
3. Umsetzung der Benutzerschnittstelle des Shops und für den Admin-Bereich.
4. Umsetzung der serverseitigen Schnittstellen samt der darunterliegenden Logik.
5. Verbindung von Benutzerschnittstelle zu den verfügbaren Schnittstellen im Server.

Hierbei ist vorgesehen, dass Phasen 3 und 4 weitestgehend parallel ablaufen. Eine visuelle Darstellung der Phasen, sowie eine feinere Aufteilung der Aufgaben in jeder Phase können im Anhang A eingesehen werden.

2.2 Meilensteine

Mit der Planung jeder Phase wurden Meilensteine definiert, welche das Erreichen von bestimmten Zwischenzielen darstellen. Somit kann während des Projekts jederzeit der Fortschritt kontrolliert werden. Im Rahmen dieses Projekts sind diese Meilensteine jedoch nur als grobe Orientierung gedacht, da Abweichungen durch externe Variablen durchaus die Planung verändern können.

2. PLANUNG

Die festgelegten Meilensteine sind der Tabelle 1 zu entnehmen.

Meilenstein	Datum
Architektur festgelegt	02.12.2016
Projektgrundlage bereit	13.12.2016
Grundlegendes Datenmodell definiert	16.12.2016
Entitäts-Klassen des Datenmodells mit Datenbank-Mapping implementiert	22.02.2017
Grundlegende Shop-Benutzerschnittstelle implementiert	21.03.2017
Grundlegende Administrations-Benutzerschnittstelle implementiert	04.04.2017
Grundlegende Schnittstellen für den Shop implementiert	31.01.2017
Grundlegende Schnittstellen für den Admin-Bereich implementiert	14.04.2017
Services im Front-End implementiert	21.04.2017

Tabelle 1: Projektmeilensteine

2.3 Aufgabenverteilung

Die Aufgaben werden im Projekt so verteilt, dass es möglichst wenig Abhängigkeiten zwischen den Teammitgliedern gibt. Diese Trennung wird erreicht, indem die Aufgaben in Backend- und Frontend-Entwicklung unterteilt werden und die Teammitglieder jeweils eine Entwicklungsstelle übernehmen.

Dabei wird Jens Gerle hauptsächlich die Programmierung des Backends angehen und Yai-za Gonzalo Alt die des Frontends. Diese Aufteilung ist jedoch nicht zwingend einzuhalten und kann je nach Bedarf oder Interesse angepasst werden.

3 Spezifikation der Anwendung

In diesem Kapitel werden die Anforderungen an das System und die daraus resultierenden Anwendungsfälle beschrieben. Anschließend wird eine System-Abgrenzung vorgenommen, sowie auf die inhaltliche Gestaltung der Anwendung eingegangen.

3.1 Anforderungen

Die Kernanforderung an das System ist es, dem Benutzer zu ermöglichen, sich Artikel anzuschauen und diese bestellen zu können. Darüberhinaus soll es einem Administrator möglich sein, neue Artikel anzulegen oder bestehende zu löschen. Dafür ist die Implementierung eines Rollensystems nötig, um die Benutzergruppen unterscheiden zu können.

Für die Bestellung von Artikeln ist es nötig, dass sich ein Benutzer in der Anwendung registrieren und seine persönlichen Daten angeben kann. Um eine Bestellung aufzugeben muss sich der registrierte Benutzer am System anmelden können. Die detaillierte Beschreibung der für diese Funktionalitäten benötigten Anwendungsfälle findet sich in Abschnitt 3.3.

3.2 Nichtfunktionale Anforderungen

Die Anwendung sollte von allen gängigen Browsern in neueren Versionen sowie auf mobilen Geräten darstellbar sein. Sensible Daten, wie die persönlichen Daten der Anwender, müssen vor unbefugtem Zugriff geschützt sein. Der Admin-Bereich sollte getrennt vom Webshop sein.

Im Fehlerfall soll der Benutzer über Hinweise darauf aufmerksam gemacht werden. Dabei sollen interne Fehlermeldungen des Servers, sowie unverständliche Fehlercodes vor dem Anwender verborgen werden. Um die Benutzerfreundlichkeit der Anwendung zu sichern, sollte sich die Antwortzeit des Servers in einem vertretbaren Rahmen bewegen (Richtwert 3 Sekunden).

3. SPEZIFIKATION DER ANWENDUNG

3.3 Hauptanwendungsfälle

Das folgende Diagramm zeigt die Anwendungsfälle für den Benutzer und Administrator, sowie die jeweils beteiligten System-Komponenten. Die Anwendungsfälle werden im Anschluss im Einzelnen betrachtet.

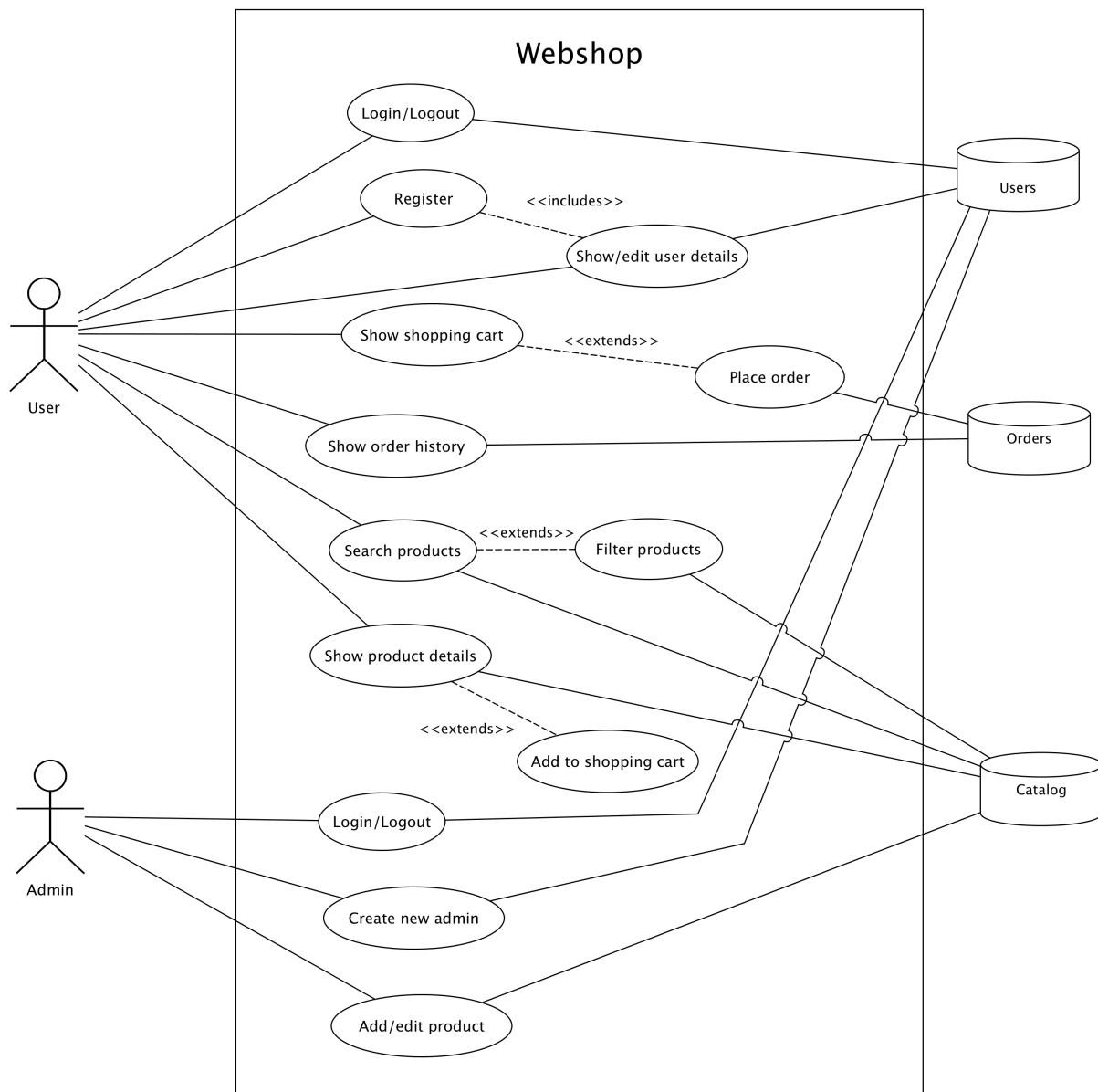


Abbildung 1: Anwendungsfalldiagramm mit den Hauptanwendungsfällen

3. SPEZIFIKATION DER ANWENDUNG

3.3.1 Benutzer- bzw. Kunden-Anwendungsfälle

Register (Registrierung)

Ein Benutzer soll sich durch Angabe seiner persönlichen Daten beim System registrieren können. Die dafür nötigen Pflichtangaben sind:

- Anrede
- Vorname
- Nachname
- Geburtsdatum
- E-Mail-Adresse
- Passwort
- Straße und Hausnummer
- Stadt
- Postleitzahl
- Land

Zusätzlich soll es bei der Registrierung möglich sein, eine abweichende Lieferadresse einzugeben. Nach der Registrierung muss es dem Benutzer möglich sein, sich am System anzumelden. Um Artikel anschauen zu können, muss ein Besucher der Seite nicht registriert oder angemeldet sein.

Login/Logout

Registrierte Nutzer können sich durch die Eingabe ihrer E-Mail-Adresse und ihrem Passwort im Shop einloggen. Bei einer fehlerhaften Eingabe soll eine Meldung ausgegeben werden. Zudem soll die Login-Seite neuen Benutzern die Möglichkeit bieten, sich zu registrieren. Der Login ist Voraussetzung für folgende weitere Use Cases: Anzeigen/Bearbeiten

3. SPEZIFIKATION DER ANWENDUNG

der persönlichen Informationen, Zum Warenkorb hinzufügen, Bestellung aufgeben und Bestellhistorie einsehen. Außerdem muss beim Einloggen auch der Warenkorb des jeweiligen Benutzers geladen werden, da dieser Artikel enthalten kann, die der User beim letzten Besuch des Shops hineingelegt hat. Beim Logout wird der Benutzer wieder abgemeldet und kann ohne einen erneuten Login nicht mehr auf seine Daten zugreifen oder die benutzerspezifischen Anwendungsfälle ausführen.

Show/edit user details (Anzeigen/Bearbeiten der persönlichen Informationen)

Einem eingeloggten Benutzer soll es möglich sein, seine persönlichen Daten einzusehen und zu ändern. Dafür soll ihm eine übersichtliche Seite mit allen Informationen und Funktionalitäten zur Verfügung stehen. Folgende Änderungen sollen möglich sein:

- Adressdaten (Liefer- und Rechnungsadresse)
- E-Mail-Adresse
- Passwort

Für eine Passwort-Änderung muss auch das bisherige Passwort eingegeben werden.

Show shopping cart (Warenkorb anzeigen)

Der Warenkorb soll jederzeit über einen Button am oberen Rand der Seite erreichbar sein. Ist der Benutzer noch nicht angemeldet, soll er beim Klick auf den Warenkorb auf die Login-Seite umgeleitet werden. Über einen Klick auf den Button wird eine detaillierte Auflistung der Artikel, die dem Warenkorb hinzugefügt wurden, sowie der Gesamtwert des Warenkorbs angezeigt. In der Detailansicht soll auch eine Anpassung der Menge für jeden Artikel und das Entfernen eines Artikels aus dem Warenkorb möglich sein. Der Warenkorb soll gespeichert werden, so dass der Einkauf zu einem späteren Zeitpunkt fortgesetzt werden kann.

Add to shopping cart (Zum Warenkorb hinzufügen)

Artikel sollen jederzeit zum Warenkorb hinzugefügt werden können, wenn ein Benutzer eingeloggt ist. Sowohl in einer Übersicht über mehrere Artikel als auch in der Detailansicht

3. SPEZIFIKATION DER ANWENDUNG

eines Artikels soll dies möglich sein. Das Hinzufügen eines Artikels zum Warenkorb soll dem Benutzer rückgemeldet werden, damit er erkennt, dass die Aktion erfolgreich war.

Place order (Bestellung aufgeben)

Über die Detailansicht des Warenkorbs sollen Bestellungen aufgegeben werden können. Vor der tatsächlichen Auslösung der Bestellung soll dem Benutzer eine Zusammenfassung gezeigt werden, die neben den einzelnen Artikeln auch seine Adressdaten und den Gesamtwert der Bestellung enthält. In dieser Übersicht soll auch die Menge für die einzelnen Artikel angepasst werden können. Nach der Bestätigung durch den Nutzer soll die Bestellung ausgelöst und eine entsprechende Meldung ausgegeben werden.

Show order history (Bestellhistorie einsehen)

Sämtliche getätigte Bestellungen sollen dem eingeloggten Benutzer jederzeit einsehbar sein. Dafür soll eine Liste mit allen Bestellungen aufgeführt werden, die dieser User aufgegeben hat. Über einen Button am Ende der Zeile sollen die Details der jeweiligen Bestellung eingesehen werden können.

Search products (Artikelsuche)

Die Suche nach Artikeln im Shop soll über eine hierarchische Unterteilung in Kategorien realisiert werden. Unter den Hauptkategorien kann der Benutzer in Unterkategorien auswählen, wonach er suchen möchte. Dadurch kann erreicht werden, dass ein Nutzer seinen Wunsch so weit präzisieren kann, dass er nur noch eine überschaubare Menge an Artikeln gezeigt bekommt, die seinen Kriterien entsprechen.

Filter products (Artikel filtern)

Um die Suche nach Artikeln weiter zu unterstützen, soll eine Filterung möglich sein, die eine Auswahl nach verschiedenen Artikelmerkmalen bietet. So soll der Benutzer nach der Marke, Farbe oder Material filtern können, um nur Produkte angezeigt zu bekommen, die den gewählten Filtern entsprechen.

Show product details (Detailansicht eines Artikels)

Durch einen Klick auf den Namen oder das Vorschaubild eines Artikels, soll dessen Detail-

3. SPEZIFIKATION DER ANWENDUNG

ansicht erscheinen, die alle verfügbaren Informationen zu diesem Artikel zeigt.

3.3.2 Administrator-Anwendungsfälle

Anwendungsfälle für Administratoren sind nur für Benutzer zugänglich, die über die entsprechende Rolle verfügen. Der Bereich für alle derartigen Operationen soll getrennt vom restlichen Shop sein und ist vor unbefugtem Zugriff zu schützen.

Login/Logout

Die Anmeldung für Administratoren soll über eine gesonderte URL erfolgen, die nicht im Shop verlinkt ist. Sie erfolgt ebenfalls über die E-Mail-Adresse und ein Passwort. Zusätzlich muss geprüft werden, ob der Benutzer über Administrator-Rechte verfügt. Erst nach erfolgreicher Authentifizierung und Verifikation der Rolle sollen die anderen Funktionen erreicht werden können.

Create new Admin (Neuen Admin hinzufügen)

Die Rolle des Admin kann nur von einem bestehenden Admin vergeben werden. Voraussetzung ist ein bestehendes Nutzerkonto (Anmeldung beim Shop) für den Benutzer, der künftig Administrator sein soll. Über eine Liste mit allen registrierten Nutzern, soll ausgewählt werden können, welcher Benutzer zu einem Admin gemacht werden soll. In gleicher Weise kann einem bestehenden Admin auch seine Rolle entzogen werden.

Add/edit product (Artikel hinzufügen/ändern)

Um neue Artikel für den Shop anzulegen oder bestehende zu ändern/löschen, soll es jedem Admin möglich sein, über eine Eingabemaske entsprechende Angaben zu machen. Für das Anlegen neuer Artikel muss das Hochladen von Bilddateien ermöglicht werden. Neue Artikel oder Änderungen sollten sofort im Shop integriert werden.

3.4 System-Abgrenzung

Das System beschränkt sich auf einen Web-Server mit der Anwendung, die in Shop und Admin-Bereich unterteilt ist und der Datenbank. Abb. 2 zeigt den System-Kontext der An-

3. SPEZIFIKATION DER ANWENDUNG

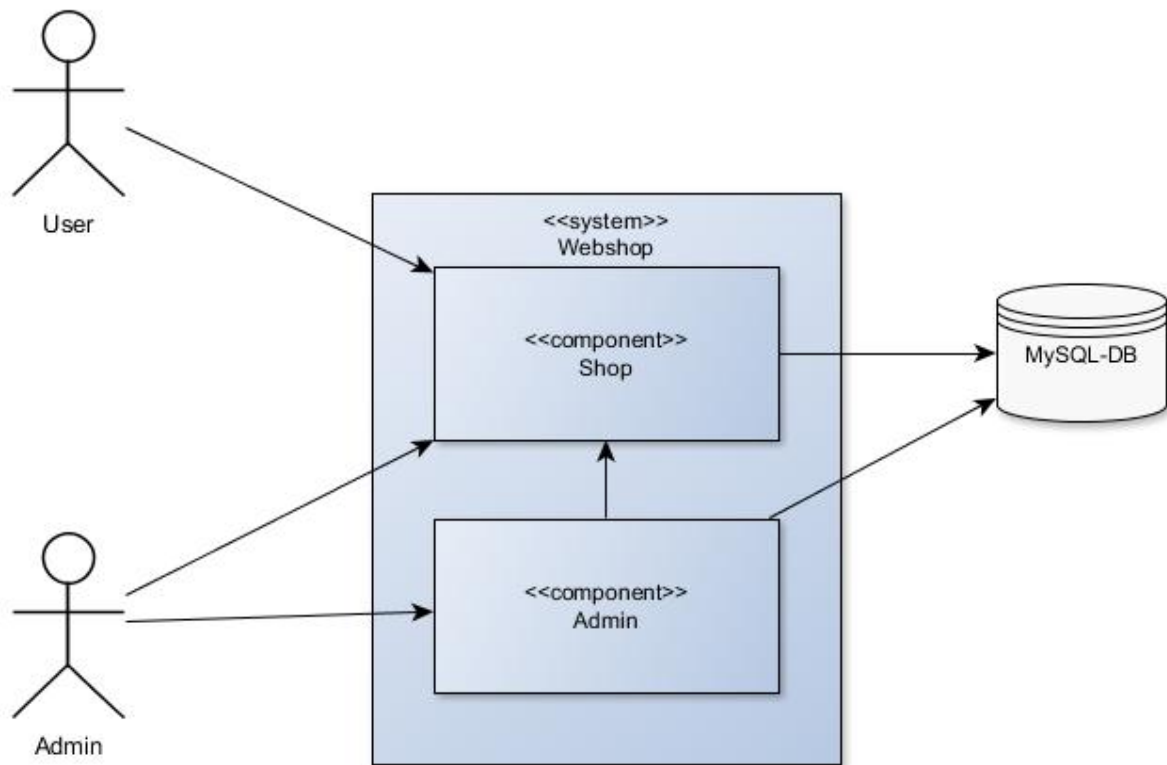


Abbildung 2: Systemkontext

wendung.

3.5 Inhaltliche Definition

Für den thematischen Inhalt des Webshops wurde das Thema Wolle und Handarbeit gewählt. Der Fokus liegt dabei auf der Wolle. Als Hauptkategorien wurden Wolle, Stricken, Häkeln, Zubehör und Bücher festgelegt. Um das Thema auch in der Gestaltung zu spiegeln, wurde viel Wert auf eine freundliche, ansprechende Oberfläche gelegt.

4 Architektur von Webanwendungen

Da es sich bei einem Online-Shop um eine Webanwendung handelt ist es notwendig, die allgemeine Architektur von solchen Anwendungen, sowie die unterschiedlichen Technologien, die eingesetzt werden können, zu analysieren. Diese Information werden später wichtig um eine Architekturentscheidung passend zu den Anforderungen und Zielen treffen zu können.

Moderne Webanwendungen werden heutzutage überwiegend nach der Client-Server-Architektur aufgebaut. Anders als bei den rein serverbasierten Ansätzen, wird im Client-Server-Ansatz nicht eine komplette Seite im Server generiert und dem Client übermittelt. Stattdessen bekommt der Client initial eine Seite mit wenig Daten geliefert, die per asynchronen Aufrufen vom Server geholt werden [2].

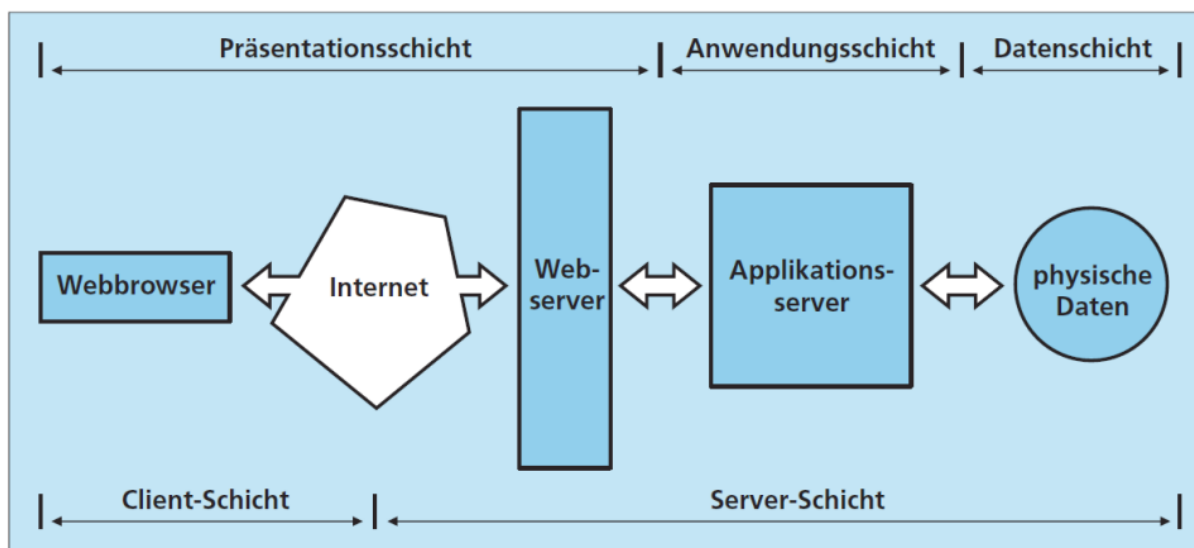


Abbildung 3: Drei-Schichten-Modell der Client-Server-Architektur [3]

Ein Modell dieser Architektur ist die Three-Tier-Architektur. Hierbei ist die Anwendung in drei logische Schichten unterteilt [4]:

- *Tier 1: Präsentationsschicht.* Ist für die Darstellung der Daten und allgemein der Benutzerschnittstelle verantwortlich.
- *Tier 2: Anwendungsschicht.* Auch Businesslogik-Schicht genannt, kontrolliert diese Schicht die eigentliche Funktionalität der Anwendung.

4. ARCHITEKTUR VON WEBANWENDUNGEN

- *Tier 3: Datenschicht.* Enthält die physischen Daten, üblicherweise in einer separaten Datenbank.

Abbildung 3 veranschaulicht diese Unterteilung der logischen Komponenten im Zusammenhang mit der Trennung von Client und Server.

4.1 Serverseite

Ein wichtiger Punkt der serverseitigen Architektur ist die Auswahl einer passenden Programmiersprache. Zu den Aufgaben solcher Programmiersprachen gehören hauptsächlich das Entgegennehmen und Beantworten von Anfragen des Clients, sowie die Kommunikation mit der Datenbank (falls vorhanden). Welche Sprache hierfür gewählt wird bestimmt auch die Frameworks und weitere Technologien, die entsprechend zur Verfügung stehen. Im Folgenden sind einige der beliebtesten Sprachen für die Server-Programmierung, sowie jeweils die Frameworks, welche oft im Rahmen der Web-Entwicklung damit verwendet werden aufgelistet [5]:

- *C#.* Objektorientiert, von Microsoft für die .NET Common Language Runtime entwickelt. *Framework: ASP.NET*
- *Java.* Objektorientiert, universell einsetzbar und robust. *Framework: Spring*
- *Node.js.* Entstand durch die wachsende Popularität von JavaScript in der Web-Programmierung. Hierbei wird die JavaScript-Syntax für Clients auch im Server benutzt. *Frameworks: Express und Hapi*
- *PHP.* Wurde von Anfang an für die Entwicklung dynamischer Webanwendungen konzipiert und ist die verbreitetste Programmiersprache in diesem Bereich. *Frameworks: Laravel und Symfony*
- *Ruby.* Gewann deutlich Popularität nach der Veröffentlichung des Frameworks Ruby on Rails. *Framework: Ruby on Rails*

4. ARCHITEKTUR VON WEBANWENDUNGEN

4.1.1 Webserver

Die Aufgabe eines Webserver ist es, die Inhalte einer Webseite per HTTP an den Client zu liefern. In der Regel handelt es sich hierbei um statische Inhalte, also HTML-Dateien, Bilder aber auch dynamisch generierte Dateien. Für die dynamische Erzeugung von Dateien können Webserver Skriptsprachen wie Perl, PHP, ASP oder JSP unterstützen.

Die verbreitetsten Webserver-Produkte sind Apache HTTP Server, Microsoft Internet Information Server (ISS) und nginx [6].

4.1.2 Applikationsserver

Ein Applikationsserver liefert, konkret im Bereich der Webanwendungen, die notwendige Businesslogik um dynamische Inhalte generieren zu können, sowie andere Funktionalitäten bereitzustellen. Dieser Server ist direkt an den Webserver angeschlossen und fängt die Anfragen nach dynamischem Inhalt ab. Diese werden dann zum Beispiel mit einer Kombination aus Templates, laufenden Programmen und Datenbankzugriffen erzeugt [7].

4.1.3 Datenbank

Für die Datenhaltung in einer Webanwendung werden fast immer Datenbanken verwendet, um Inhalte unabhängig von der Anwendung persistent speichern zu können. Dabei werden meist relationale oder NoSQL-Datenbanken eingesetzt.

Relationale Datenbanken bilden in ihrem Datenmodell die Beziehungen zwischen den einzelnen Datensätzen ab. Als Entitäten bezeichnet, werden logisch unabhängige Datensätze in jeweils eigenen Tabellen gespeichert. Jeder Datensatz ist dabei in einer eigenen Zeile gespeichert und die Tabellenspalten stellen die einzelnen Attribute dar. Verweise auf andere Entitäten für die Abbildung der Beziehungen werden durch sogenannte Fremdschlüssel-Spalten realisiert, die einen bestimmten Datensatz einer anderen oder auch derselben Entität referenzieren können. Die am meisten verwendeten Produkte relationaler Datenbanken sind Oracle Database, MySQL, Microsoft SQL Server, PostgreSQL und DB2 [8].

4. ARCHITEKTUR VON WEBANWENDUNGEN

NoSQL-Datenbanken sind die häufigste Alternative zu den relationalen Lösungen. Der Name ist ein Akronym für „Not only SQL“ und spielt damit auf die Beschränkungen der verbreitetsten Datenbanksprache SQL relationaler Modelle an. Unter dieser Kategorisierung gibt es verschiedene Modelle. Folgend sind die wichtigsten Datenmodelle mit je einem Beispiel [9]:

- *Dokumentenorientierte Datenbanken.* MongoDB
- *Key-Value-Datenbanken.* Redis
- *Graphdatenbanken.* Neo4j
- *Spaltenorientierte Datenbanken.* Apache Cassandra

4.1.4 ORM

Unter dem Object Relational Mapping, kurz ORM, versteht man die Zuordnung der Repräsentation von Objekten in der verwendeten Programmiersprache zu der dafür vorgesehenen Datenstruktur in der relationalen Datenbank. Diese Aufgabe benötigt sowohl eine technische Lösung zur Anbindung und Ansteuerung der Datenbank aus der Anwendung, als auch eine logische für die richtige Zuordnung von Objekten auf Entitäten-Tabellen und Objekt-Attributen auf deren Spalten. Für die Anbindung der Datenbank gibt es je nach eingesetzter Programmiersprache und Datenbank-Technologie spezielle Tools, auf die an dieser Stelle nicht näher eingegangen werden soll. Zum Zweck des Mappings, also der logischen Zuordnung, gibt es Hilfe in Form von Frameworks, die es ebenfalls für die verschiedenen zugrunde liegenden Technologien gibt. Die wichtigsten Vertreter sind hier das Entity Framework für .NET-Sprachen und Hibernate für Java.

4.2 Clientseite

Die wichtigsten clientseitigen Webtechnologien sind schon seit einigen Jahren HTML, CSS und JavaScript. Hierbei sind HTML und CSS Technologien für die Darstellung der Webseite und JavaScript allgemein für die clientseitige Programmierung zuständig [10].

4. ARCHITEKTUR VON WEBANWENDUNGEN

4.2.1 JavaScript Erweiterungen

Ein wichtiger Wendepunkt in der Geschichte von JavaScript war die Einführung von AJAX („Asynchronous JavaScript and XML“). Durch AJAX wurde es möglich, asynchrone Anfragen zum Server durchzuführen, um Teile einer Seite zu verändern ohne sie komplett neu laden zu müssen. Ein großer Vorteil davon ist die erhöhte Reaktionsgeschwindigkeit der Weboberfläche, da diese nicht durch die Anfragen blockiert wird.

Kurze Zeit später kamen die sogenannten JavaScript Document Object Model (DOM) Libraries wie *jQuery* dazu. Diese Bibliotheken fügten zahlreiche nützliche Funktionalitäten zur einfachen Manipulation der DOM und zur Implementierung von AJAX-Aufrufen mit deutlich weniger Code [11].

4.2.2 Arten von Webanwendungen

Die bereits erwähnten Verbesserungen an JavaScript, aber auch an HTML mit der Veröffentlichung von HTML5, brachten neue Techniken mit sich, mit denen neuartige Anwendungen aufgebaut werden konnten. Single Page Applications (SPAs) sind eine Art von Webanwendung, die in diesem Rahmen erschien. In den nächsten Abschnitten werden zum Vergleich die traditionellen Webanwendungen, sowie die modernen Single Page Applications näher beschrieben.

Traditionelle Webanwendungen

Eine traditionelle Webanwendung generiert die dargestellte Webseite immer wieder neu wenn eine beliebige HTTP-Anfrage eingeht. Das gilt demnach nicht nur für den initialen Aufruf der Webseite, sondern zum Beispiel auch für das Senden eines ausgefüllten Formulars. Die im Server erzeugte HTML-Seite wird als Antwort zum Client gesendet, was dort wiederum eine Aktualisierung bewirkt. Bei einer Aktualisierung lädt der Browser die Seite neu und ersetzt dabei die ursprüngliche HTML durch die neue [11]. Abb. 4 veranschaulicht diesen Ablauf.

4. ARCHITEKTUR VON WEBANWENDUNGEN

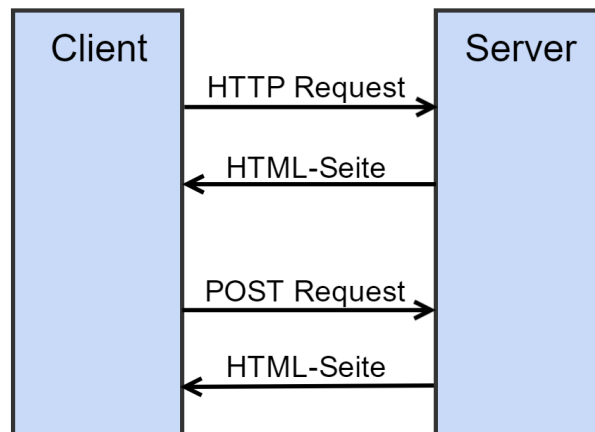


Abbildung 4: Lebenszyklus einer traditionellen Webanwendung

Single Page Applications

Wie in Abb. 5 zu erkennen ist, gibt es beim initialen Aufruf der Webseite keine Unterschiede zwischen traditionellen und Single Page Anwendungen. Bei allen weiteren Anfragen ist der Ablauf jedoch anders. Durch AJAX sind die Anwendungen in der Lage, Anfragen an den Server zu senden, bei denen nur die notwendigen Daten angefordert werden. Der Client erhält daraufhin eine Antwort, welche diese Daten enthält (üblicherweise im JSON-Format). Sobald die Daten ankommen kann der Client die HTML-Seite partiell anpassen, ohne ein Neuladen des Browser-Fensters. Dieser Ablauf ist für alle Benutzerinteraktionen in der Anwendung gleich, selbst für die Navigation. Das macht die Antwortzeiten solcher Anwendungen sehr kurz, was eine sehr positive Auswirkung auf die Benutzerzufriedenheit hat.

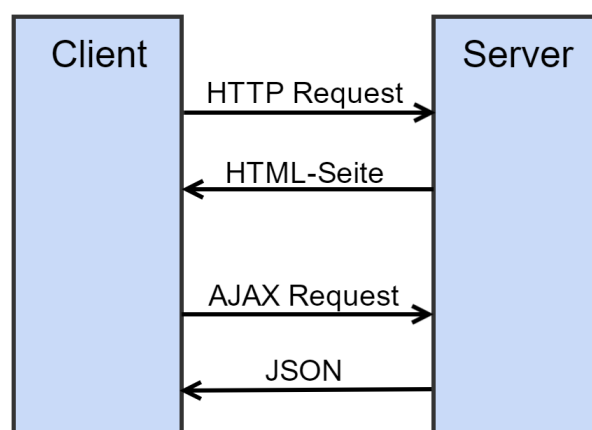


Abbildung 5: Lebenszyklus einer SPA

5 Architekturentscheidung

Nach einer sorgfältigen Analyse der unterschiedlichen Technologien wurde eine Architektur entworfen, welche in Abb. 6 dargestellt ist. Die Komponenten wurden so gewählt, dass sie untereinander mit möglichst wenig Konfigurationsaufwand reibungslos funktionieren.

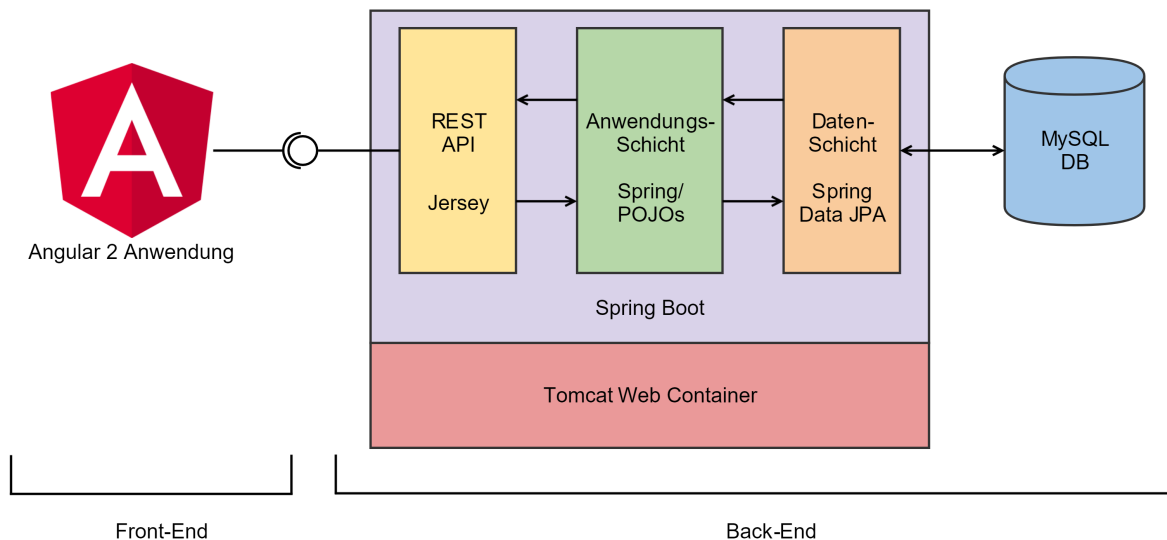


Abbildung 6: Grundlegende Architektur des Webshops

Im Folgenden werden die theoretischen Grundlagen zu den einzelnen Komponenten, sowie weitere Erläuterungen zu den Gründen der Architekturentscheidung beschrieben.

5.1 Spring Boot Anwendung

Spring ist ein Framework für die Entwicklung von Applikationen in der Programmiersprache Java. Das Framework kann für alle Arten von Java-Anwendungen genutzt werden, da eine große Vielfalt an Abhängigkeiten für die unterschiedlichsten Funktionalitäten zur Verfügung steht.

Ursprünglich entstand Spring als eine leichtgewichtige Alternative zur Java Enterprise Edition (JEE), sodass die Komponenten der Software nicht mehr als Enterprise JavaBeans (EJBs) sondern als einfache Java Objekte (POJOs) realisiert werden können. Die Eigenschaften von EJBs werden anhand von Techniken wie Dependency Injection und aspektori-

5. ARCHITEKTURENTSCHEIDUNG

entierter Programmierung erreicht. Ein großer Nachteil hierbei ist jedoch der große Konfigurationsaufwand für den Entwickler. Initial erfolgte diese Konfiguration über mehrere XML-Dateien. Ab Spring 2.5 wurde die XML-basierte Konfiguration teilweise durch sogenannte Annotationen in den Java-Klassen ersetzt, was den Aufwand jedoch nicht sonderlich reduziert [12].

Das Ziel des Spring Boot Projekts ist es genau diesen Nachteil zu beseitigen. Durch eine automatische Konfiguration können Entwickler schnell komplexere Spring-Anwendungen aufbauen, ohne auf die Konfiguration von allgemeinen Funktionalitäten achten zu müssen. Zudem ist auch das Einbinden von Spring-Abhängigkeiten stark vereinfacht [13].

Für die Verwendung von Spring Boot als serverseitiger Anwendung des Webshops spricht vieles. Zum einen ist im Team bereits einige Erfahrung aus anderen Projekten vorhanden, die hier wiederverwendet werden kann. Zum anderen erleichtert Spring Boot durch das einfache Aufsetzen der Anwendung, der umfangreichen Dokumentation, sowie der großen Auswahl an Abhängigkeiten deutlich den Start mit der Umsetzung der eigentlichen Anforderungen. Besonders die Abhängigkeit `spring-boot-starter-web` ist für die Entwicklung von Webanwendungen nützlich, denn damit wird automatisch ein Webserver bzw. ein Java Servlet Container (Apache Tomcat) in das Projekt eingebettet.

5.2 Angular 2 Frontend

Angular 2 ist die neue, komplett umgebaute Version des Angular 1.x Frameworks (auch bekannt als AngularJS). Diese Version liefert eine brandneue Architektur basierend auf Komponenten die in TypeScript implementiert werden. Bei TypeScript handelt es sich wiederum um ein von Microsoft entwickeltes Superset von JavaScript. TypeScript führt ein optionales Typsystem ein, sowie mehrere ECMAScript 6 Funktionalitäten, wie zum Beispiel Unterstützung von Interfaces, Klassen und Vererbung. Der Vorteil dabei ist, dass die Erweiterungen von TypeScript von einem Compiler in herkömmliches JavaScript umgewandelt werden, so dass man TypeScript mit jedem Browser oder auch mit Node.js benutzen kann, solange der Code vorher kompiliert wird [14].

5. ARCHITEKTURENTSCHEIDUNG

Bei Angular 2 handelt es sich um ein sehr junges Framework (die endgültige Version wurde im September 2016 veröffentlicht), sodass es noch zu früh ist um dessen Erfolg messen zu können. Fest steht jedoch, dass die aktuelle Beliebtheit von TypeScript ein ausschlaggebender Faktor für den Erfolg sein könnte. Da im Rahmen dieses Projekts allgemein keine bestimmten Technologien vorgegeben wurden, bietet es sich durchaus an, das Framework mit den neuen Features und der neuen Architektur näher kennenzulernen.

Neben dem eigenen Interesse an diesem Framework sprach noch ein weiterer Faktor maßgeblich für die Entscheidung, Angular als Technologie für die Implementierung des Frontends zu verwenden. Es ist geplant, den Webshop als eine Single Page Application zu realisieren, deren Vorteile bereits in Abschnitt 4.2.2 geschildert wurden. Angular gehört schon seit der ersten Version zu den beliebtesten Frameworks für diesen Anwendungsfall, da die mitgelieferten Tools und Funktionalitäten die Implementierung sehr vereinfachen. Besonders der Routing-Mechanismus für die Navigation in der SPA ist in Angular sehr ausgereift.

Auf den nächsten Seiten werden die Eigenschaften des Frameworks etwas ausführlicher behandelt, da das Thema im Laufe der Implementierung mitunter den größten Einarbeitungsaufwand mit sich brachte.

Eine Angular 2 Anwendung besteht im Grunde aus den folgenden Bausteinen [15]:

- *Komponenten*
- *Templates*
- *Metadaten*
- *Modulen*
- *Data Binding*
- *Direktiven*
- *Services*
- *Dependency Injection*

5. ARCHITEKTURENTSCHEIDUNG

5.2.1 Komponenten

Komponenten sind TypeScript-Klassen, die die Darstellung der Oberfläche, die sogenannte „View“ kontrollieren. Jede Klasse enthält Attribute und Methoden, mit denen die Darstellung verändert werden kann.

5.2.2 Templates

Templates definieren wie die Darstellung einer Komponente genau aussieht. Hierfür wird eine erweiterte Form von HTML verwendet. Templates können somit zum Beispiel auch Kinder einer Komponente durch besondere HTML-Tags einbinden.

5.2.3 Metadaten

Metadaten geben Angular Informationen darüber, wie eine Klasse verarbeitet werden soll. Komponenten können erst von Angular als solche erkannt werden, wenn die Klasse die entsprechenden Metadaten enthält. In TypeScript wird das mittels einem Decorator gemacht, eine Annotation, die über die Klassendeklaration gesetzt wird.

```
1 @Component({
2   selector: 'sample-component',
3   templateUrl: './sample.component.html',
4   providers: []
5 })
6 export class SampleComponent implements OnInit {
7   /* . . . */
8 }
```

Listing 1: Beispiel Metadaten-Annotation für Komponenten

In Listing 1 ist als Beispiel die Annotation für Komponenten dargestellt. Unter `@Component` können unterschiedliche Aspekte einer Komponente definiert werden, wie der HTML-Selektor, der Ablageort des Templates oder benötigte Abhängigkeiten.

5. ARCHITEKTURENTSCHEIDUNG

5.2.4 Module

Module fassen mehrere Komponenten und Services zu einem Funktionalitäts-Block zusammen. Jede Angular Anwendung besitzt mindestens ein Modul, das sogenannte „Root Module“.

```
1 @NgModule({
2   declarations: [ AppComponent ],
3   imports: [ ],
4   providers: [ ],
5   bootstrap: [AppComponent]
6 })
7 export class AppModule { }
```

Listing 2: Root Modul

Listing 2 zeigt den initialen Zustand eines Root-Moduls. Jedes Modul wird durch die Metadaten in der `@NgModule`-Annotation definiert. Diese teilt Angular mit, welche Komponenten im Modul deklariert sind, sowie welche Module und Services importiert bzw. eingebunden werden müssen. Das Root-Modul enthält zusätzlich auch die `bootstrap`-Definition, in der die Hauptkomponente festgelegt wird, die Angular beim Ausführen der Anwendung in die `index.html` der Webseite einfügt [16].

5.2.5 Data Binding

Als Data Binding (zu Deutsch Datenbindung) bezeichnet man den Mechanismus zur Zuordnung von bestimmten Elementen eines Templates zu Datenstrukturen der dazugehörigen Komponente. Abb. 7 zeigt die vier Arten von Data Binding Syntax in Angular. Jede Form von Data Binding hat auch eine Richtung: von der Komponente zum Document Object Model (DOM) im Template, vom DOM zur Komponente oder bidirektional.

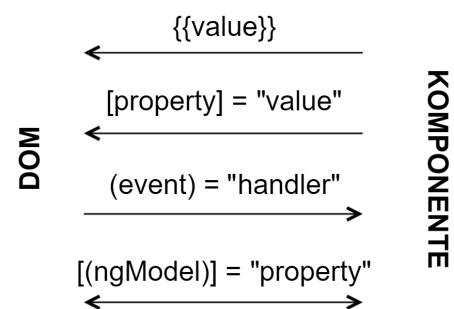


Abbildung 7: Arten des Data Bindings in Angular 2

5. ARCHITEKTURENTSCHEIDUNG

Um diese vier Data Binding Formen weiter zu veranschaulichen, ist in Listing 3 jeweils ein Beispiel dargestellt. In der ersten Zeile wird durch die doppelte geschweifte Klammern der Wert `name` vom Attribut `user` der Komponenten-Klasse in die DOM eingefügt. Zeile 2 zeigt eine eingebundene Kindkomponente `FoodComponent`, die als Input den Wert des Attributs `selectedFood` von der Komponenten-Klasse erhält. Die runden Klammern mit einem Event-Bezeichner in Zeile 3 binden eine Methode der Komponente zum angegebenen Event. Die letzte Zeile zeigt eine besondere Art von Data Binding, das *Two-Way Data Binding*. Hierbei fließt der Wert vom Attribut der Klasse nicht nur von der Komponente in die DOM ein, sondern wird auch in der Komponente aktualisiert wenn dieser sich beispielsweise durch eine Benutzereingabe verändert.

```
1 <h1>Hallo {{user.name}}</h1>
2 <food-component [user]="selectedFood"></food-component>
3 <li (click)="selectFood(food)"></li>
4 <input [(ngModel)]="user.name">
```

Listing 3: Beispiele zu den Data Binding Arten

5.2.6 Direktiven

Bei Direktiven handelt es sich um Klassen mit dem `@Directive`-Decorator, in denen beliebige Funktionalitäten zur dynamischen Transformation der DOM definiert werden können. Komponenten sind sozusagen Direktiven mit einem Template. Neben Komponenten gibt es aber auch zwei andere Arten von Direktiven: strukturelle und Attribut-Direktiven. Erstere verändern das Layout der View durch das Hinzufügen, Löschen oder Ersetzen von Elementen in der DOM. Attribut-Direktiven verändern hingegen das Aussehen oder das Verhalten eines bereits vorhandenen DOM-Elements. Ein Beispiel dafür ist die `ngModel`-Direktive die bereits in Listing 3 gezeigt wurde.

5.2.7 Services

Der Begriff Service ist in Angular sehr weit umfassend, da grundsätzlich alles mögliche ein Service sein kann. Typischerweise handelt es sich jedoch einfach um eine Klasse, die eine konkrete Aufgabe erledigt, wie zum Beispiel Logging, Datenzugriff, Berechnungen, usw.

5. ARCHITEKTURENTSCHEIDUNG

Diese Logik sollte idealerweise nie in Komponenten selbst enthalten sein, da diese nur als Mediatoren zwischen der Darstellung (View) und der Logik zu verstehen sind.

Services zeigen an sich also keine besonderen Angular-Eigenschaften. Um die Trennung zwischen der in Services gekapselten Logik und den Komponenten einfach zu überwinden, wird jedoch ein charakteristischer Mechanismus von Angular verwendet, die Dependency Injection.

5.2.8 Dependency Injection

Dieser Mechanismus ermöglicht es, eine neue Instanz einer Klasse mit allen nötigen Abhängigkeiten (in der Regel Services) zu versorgen. Um Angular mitzuteilen, welche Services eine Komponente benötigt, müssen diese im Konstruktor oder alternativ im `@Component`-Decorator deklariert werden, wie in Listing 1 bereits gezeigt wurde.

5.3 REST Services

Für die Kommunikation zwischen Front- und Backend wird REST eingesetzt. REST steht für Representational State Transfer und ist ein Architekturstil für den Entwurf von verteilten Systemen. Die Grundlage von REST wird durch folgende sechs Prinzipien festgelegt [17]:

- *Client-Server*. Die Unterteilung zwischen Client und Server ermöglicht es den jeweiligen Komponenten sich unabhängig voneinander zu entwickeln und erlaubt dementsprechend die Skalierung des Systems.
- *Zustandslos*. Die Kommunikation zwischen Client und Server sollte zustandslos sein. Das bedeutet, dass der Server sich nicht den Zustand des Clients merken braucht. Stattdessen kümmert sich der Client darum, alle notwendigen Informationen in der Anfrage mitzuliefern, damit der Server diese verstehen und verarbeiten kann.
- *Mehrschichtige Systeme*. Es können mehrere Schichten wie zum Beispiel Gateways, Firewalls oder Proxies zwischen Client und Server existieren. Diese Schichten können transparent hinzugefügt, modifiziert, umgeordnet oder gelöscht werden.

5. ARCHITEKTURENTSCHEIDUNG

- *Cache.* Antworten vom Server müssen als „cacheable“ oder „noncacheable“ deklariert sein. Dadurch wissen Clients, ob sie die Antworten zwischenspeichern können, um sie später wiederverwenden zu können. Dieser Vorgang verringert die Last auf dem Server und verbessert die Performance.
- *Uniform Interface.* Alle Interaktionen zwischen Client, Server und dazwischenliegenden Komponenten basieren auf der Uniformität der Schnittstellen. Solche Schnittstellen definieren welche Ressourcen über eine eindeutige URI durch bestimmte HTTP-Methoden erreichbar sind. Ein Beispiel für eine REST URI könnte folgendermaßen aussehen:

`http://blog.example.com/posts/1`

Die URI repräsentiert hier den Zugriff auf einem Blog-Post mit Identifikator 1.

- *Code on demand.* Clients können ihre Funktionalitäten erweitern, indem sie Code (wie JavaScript Skripte oder Java Applets) bei Bedarf herunterladen und ausführen. Dies ist jedoch eine optionale Eigenschaft.

Anwendungen die diese Prinzipien befolgen, gelten als „RESTful“, also REST-konform. Es ist hierbei nicht vorgegeben, welche Technologie konkret für die Entwicklung solcher Anwendungen benutzt werden soll. Theoretisch ist es möglich eine REST-Anwendung mit einer beliebigen Netzwerk-Infrastruktur oder einem beliebigen Protokoll aufzubauen. In der Praxis nutzen diese Applikationen jedoch hauptsächlich die Eigenschaften der Web-Entwicklung und verwenden dementsprechend HTTP als Übertragungsprotokoll.

Von den sechs oben beschriebenen Punkten ist das Prinzip des Uniform Interfaces das Hauptmerkmal von REST-Anwendungen, welches diese Art von Applikationen von anderen netzwerkbasierten Anwendungen unterscheidet.

5.4 MySQL Datenbank

Den Ausschlag für die Wahl einer MySQL-Datenbank gaben mehrere Faktoren. Zum einen ist MySQL weit verbreitet und bietet als Open Source-Projekt kostenfrei eine Vielzahl an Funktionalitäten. Ein weiterer Punkt ist die bereits vorhandene Erfahrung im Umgang mit

5. ARCHITEKTURENTSCHEIDUNG

SQL aus vorigen Projekten. Unter dem Gesichtspunkt der bisherigen Entscheidung für Java als Implementierungssprache für das Backend ist MySQL eine gute Wahl, da es entsprechende Treiber für die Ansteuerung gibt und die Java Persistence API (JPA) eine gute Integration der Datenbank in die Anwendung ermöglicht.

5.5 Hibernate ORM

Mit Java und einer MySQL-Datenbank als Voraussetzung bietet sich die Verwendung von Hibernate für die Objektrelationale Abbildung an. Mit Hibernate stehen dem Entwickler viele Annotationen zur Verfügung, die eine einfache Kennzeichnung von Klassen und Attributen für ihre Rolle in der Datenbank ermöglichen. Auch bei komplexeren Strukturen wie einer Vererbungshierarchie von Klassen gewährleistet Hibernate eine Abbildung auf die Datenbankstruktur und bietet verschiedene Möglichkeiten an, wie das geschehen kann. Außerdem können die Tabellen der Datenbank auch von Hibernate erzeugt werden, wenn das Modell über die Java-Klassen zuvor definiert wird.

6 Entwurf

Dieses Kapitel beschreibt den konzeptionellen Entwurf des Datenmodells und der Oberfläche als Grundlage für die darauf aufbauende Umsetzung.

6.1 Datenmodell

Beim Entwurf des Datenmodells wurde darauf geachtet, es möglichst schlank zu halten und nur benötigte Informationen zu speichern. Die grundlegenden Entitäten sind:

- *User*. Die registrierten Benutzer
- *Address*. Adressdaten der User
- *BaseItem*. Artikeldaten
- *Category*. Kategorien
- *ShoppingCart*. Warenkorb
- *ShoppingOrder*. getätigte Bestellungen

Abbildung 8 zeigt das Datenmodell in der Gesamtsicht.

Die Beziehungen zwischen den Entitäten sind entweder direkt per Fremdschlüssel oder über Beziehungstabellen dargestellt und werden im Folgenden kurz beschrieben.

Ein User hat eine Lieferadresse (Shipping address) und eine Rechnungsadresse (Billing address). Um die bereits angesprochene Hierarchie von Kategorien abzubilden, können Kategorien eine Eltern-Kategorie haben (Parent category). Jeder Artikel ist genau einer Kategorie zugeordnet. Für die im Shop so genannten Sets mit mehreren Artikeln, gibt es eine zusätzliche Tabelle, die einem Set mehrere Einzel-Artikel zuordnen kann. Die Warenkörbe sind immer einem Benutzer zugeordnet. In gleicher Weise sind auch die Bestellungen Benutzer-spezifisch. Um der Tatsache gerecht zu werden, dass in einem Warenkorb oder

6. ENTWURF

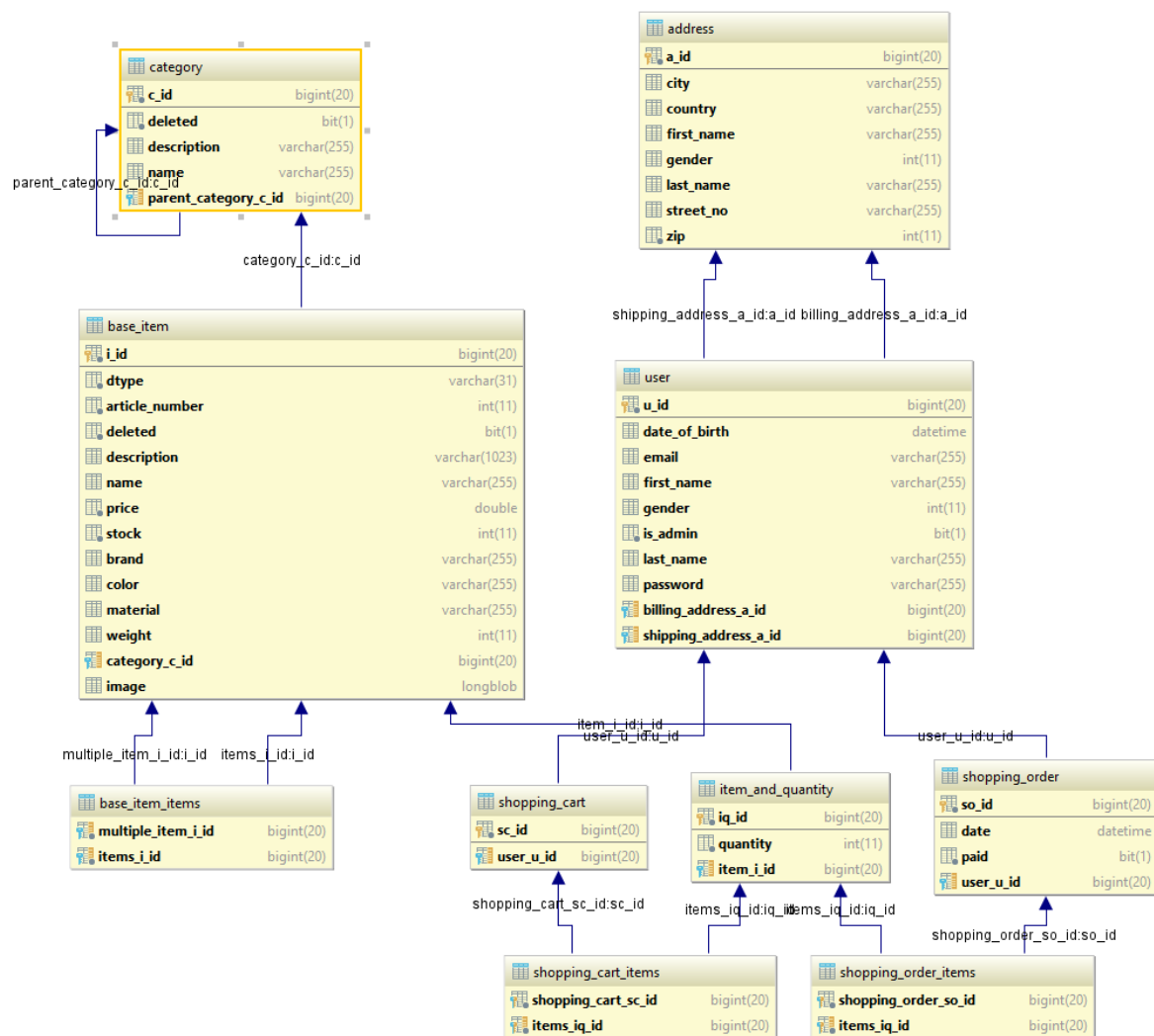


Abbildung 8: Entity Relationship Modell

in einer Bestellung Artikel mehrmals vorhanden sein können (wenn ein Benutzer von einem Artikel mehrere Exemplare bestellt), wurde eine zusätzliche Entität eingeführt. Diese *ItemAndQuantity* genannte Entität hält jeweils eine Referenz auf einen Artikel, sowie eine Anzahl für diesen Artikel. Warenkörbe und Bestellungen enthalten also immer Einträge aus dieser Tabelle. Auf diese Weise genügt es, für einen Artikel immer nur einen Eintrag zu speichern, auch wenn die bestellte Menge größer ist. Ohne diese Lösung müsste es für eine Bestellung von beispielsweise dreißig Exemplaren eines Artikels auch dreißig Einträge geben.

6.2 Design der Weboberfläche

Vor Beginn der Implementierung wurde ein Konzept für das Design der Weboberfläche erstellt. Der Fokus lag hierbei eher auf den strukturellen Aspekten der Anwendung, also die Platzierung der unterschiedlichen Elemente auf der Oberfläche, und nicht so sehr auf dem genauen Aussehen dieser Elemente. Der Grund dafür ist, dass die Struktur einen direkten Einfluss auf die Implementierung der Komponenten hat, während Farben, Schriftarten, Icons usw. auch im Verlauf der Entwicklung ohne einen größeren Umbau verändert werden können.

Abb. 9 zeigt die grundlegende Struktur der Shop-Oberfläche im Browser. Wie man sehen kann, ist der Entwurf sehr simpel und intuitiv gehalten. Eine Leiste im Kopfbereich enthält die Links zum Benutzerbereich und zum Warenkorb. Die größere Menüleiste unter dem Logo erlaubt die Navigation zwischen den Hauptkategorien des Webshops. Je nachdem in welchem Bereich der Anwendung sich der Benutzer befindet, variiert der Inhalt der Seite. Prinzipiell soll es fünf unterschiedliche Arten des Inhaltes geben: die Startseite, die Produktlisten zur gewählten Kategorie, die Kontoübersicht, der Warenkorb und die Bestellmaske. Die Mockups dazu sind im Anhang B zu finden.

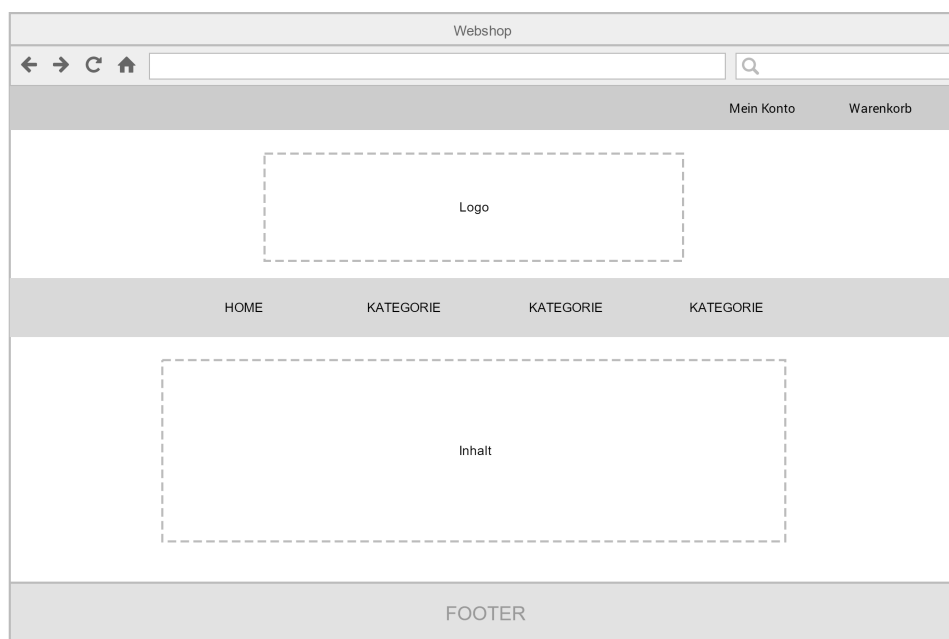


Abbildung 9: Mockup des Webshops

6. ENTWURF

Da eine wachsende Mehrheit der Nutzer mobile Geräte für ihre Internetzugriffe verwendet, ist es sehr wichtig eine Strategie für die Darstellung der Weboberfläche in solchen Geräten zu definieren. Die in 2016 von der Mediaagentur Zenithmedia veröffentlichte Vorhersage sagte aus, dass in 2017 weltweit 75% der Internetnutzung über mobile Geräte erfolgen wird, 79% in 2018 [18].

Es gibt verschiedene Ansätze, um Webseiten für unterschiedliche Displaygrößen zu optimieren. Eine Möglichkeit wäre es, eine zweite Version der Webseite zu erstellen, welche speziell für die Nutzung auf mobilen Endgeräten angepasst ist. Eine weitere, modernere Alternative sind Paradigmen wie *Responsive* und *Adaptive* Webdesign. In beiden Fällen werden sogenannte Breakpoints für die Größe des Displays gesetzt, bei denen die Webseite skaliert und sich an die neue Auflösung anpasst. Der Unterschied liegt darin, dass beim Adaptive Design eine feste Anzahl an Breakpoints gesetzt wird und es keine Übergänge dazwischen gibt, während beim Responsive Design eine fluide Anpassung auch zwischen den Breakpoints möglich ist.

Der für dieses Projekt gewählte Ansatz ist Responsive Webdesign, da es mithilfe der geplanten Technologien einfach zu realisieren ist. In Abb. 10 ist wieder die Grundstruktur der Webseite zu sehen, jedoch auf die Größe eines Smartphone-Displays skaliert. Eine wichtige Änderung bei dieser Skalierung betrifft die Menüleiste. Diese sollte nicht länger als Leiste dargestellt werden, sondern als auf- und zuklappbares Dropdown-Menü.

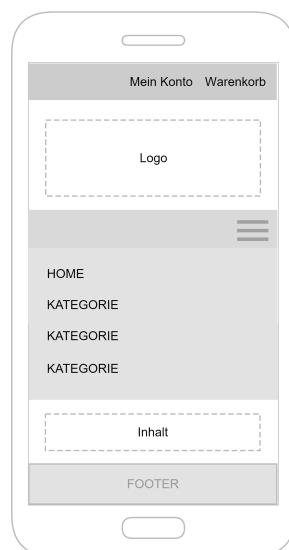


Abbildung 10: Mockup des Webshops Handy Version

7 Umsetzung

In diesem Kapitel werden die wichtigsten Aspekte der Umsetzung der in Abschnitt 3 beschriebenen Spezifikation behandelt, sowie exemplarisch einige Implementierungsdetails vorgestellt.

7.1 Initiales Aufsetzen der Anwendung

Bevor mit der Entwicklung begonnen werden kann, müssen die grundlegenden Bausteine der Anwendung angelegt, konfiguriert und miteinander verbunden werden.

7.1.1 Spring Boot Anwendung

Spring stellt eine Weboberfläche zur Verfügung, in der ein komplettes Spring Boot Projekt inklusive Abhängigkeiten einfach generiert werden kann. Die Oberfläche (namens Spring Initializr) erlaubt es Maven oder Gradle als Build-Management-Tool zu verwenden. In diesem Projekt fiel die Entscheidung für Maven, aufgrund der bereits vorhandenen Erfahrung mit dem Tool.

The screenshot shows the Spring Initializr web interface. At the top, it says "SPRING INITIALIZR bootstrap your application now". Below this, there's a form to generate a project. The form has two main sections: "Project Metadata" and "Dependencies".

Project Metadata:

- Artifact coordinates: Group (de.dhbw), Artifact (webshop)

Dependencies:

- Add Spring Boot Starters and dependencies to your application
- Search for dependencies: Web, Security, JPA, Actuator, Devtools...
- Selected Dependencies: Web, Jersey (JAX-RS), JPA, MySQL

At the bottom, there's a green button labeled "Generate Project alt + ⌘".

Abbildung 11: Spring Initializr Weboberfläche

Abb. 11 zeigt die gewählte Konfiguration für das Grundprojekt. Links sind die Metadaten für das Deployment definiert und rechts die Abhängigkeiten. Diese sind initial:

7. UMSETZUNG

- *Spring Web*. Wird benötigt um eine Webanwendung mit eingebettetem Tomcat-Server zu implementieren.
- *Jersey*. Hierbei handelt es sich um die Referenzimplementierung von JAX-RS. JAX-RS ermöglicht es, anhand von Annotationen, herkömmliche Java-Klassen per REST verfügbar zu machen [19].
- *Spring Data JPA*. Stellt die Zugriffstechnologie der Java Persistence API (JPA) zur Verfügung. Mit dieser Abhängigkeit können Datenzugriffsobjekte angelegt werden (sogenannte DAOs, Data Access Objects), die alle notwendige Datenbankoperationen implementieren [20].
- *Spring MySQL JDBC Driver*. Mit diesem Treiber kann die Spring Boot Anwendung automatisch eine Verbindung zur MySQL-Datenbank herstellen [20]. Alle weitere Konfigurationen für diese Verbindung werden später in dieser Arbeit beschrieben.

Die Ordnerstruktur des generierten Projekts kann der Abb. 12 entnommen werden. Es handelt sich hierbei um die übliche Struktur einer Java-Anwendung, mit einem `main` und einem `test` Verzeichnis. Initial sind im `main` Verzeichnis nur zwei Dateien: die `WebshopApplication` Klasse und die `application.properties`. Erstere enthält die Java `main`-Methode, welche die Anwendung lauffähig macht. Dort werden auch automatisch alle notwendige Konfigurationen für Spring Boot gesetzt. Die `application.properties` Datei ist zu diesem Zeitpunkt noch leer, aber sie wird später benutzt um unterschiedliche Aspekte der Anwendung (wie zum Beispiel Server-Port oder Verbindungsdaten zur Datenbank) auf einfache Art und Weise einzustellen.

Eine weitere wichtige Datei ist die `pom.xml` im Stammverzeichnis. Das Project Object Model oder POM ist die grundlegende Arbeitseinheit von Maven. Diese XML-Datei enthält Information über das Projekt, sowie Konfigurationsdetails, die von Maven verwendet werden um das Projekt zu bauen. Einige dieser Konfigurationen betreffen beispielsweise Projekt-Abhängigkeiten, Plugins und Build-Profile [21].

7. UMSETZUNG

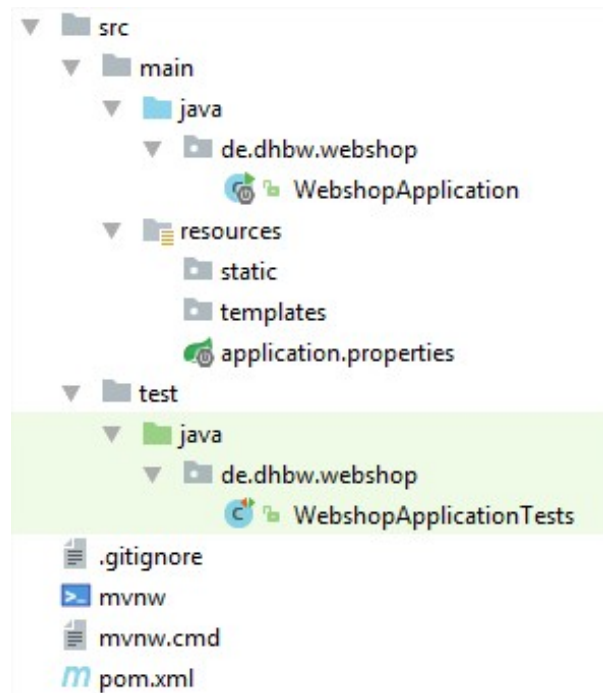


Abbildung 12: Ordnerstruktur des generierten Projekts

7.1.2 Angular 2 Anwendung

Auch hier wird ein Tool eingesetzt, um eine Basisanwendung zu generieren. Dabei handelt es sich um Angular CLI, ein Tool für die Initialisierung, Entwicklung und Wartung von Angular Anwendungen [22].

Angular CLI ist ein Open Source Projekt und wird anhand des Node Package Managers (NPM) installiert. Daraufhin stehen zahlreiche nützliche Befehle für die Kommandozeile zur Verfügung. Für das Generieren einer Angular-Anwendung wird `ng init` ausgeführt. Der Befehl erstellt alle notwendigen Ressourcen, sowie die komplette Konfiguration dazu. Die statischen Inhalte werden mit `ng build` gebaut.

Damit Spring Boot diese Inhalte verwenden kann, müssen sie jedoch im `resources/static` Verzeichnis (siehe Abb. 12) liegen. Das kann durch eine Anpassung an der `angular-cli.json` Datei erreicht werden (Zeile 4 von Listing 4). Des Weiteren enthält diese Datei auch weitere wichtige Konfigurationen, wie der Ablageort von anderen Ressourcen wie Bildern oder benötigten Stylesheets. An dieser Stelle werden demnach die CSS-Dateien für das Bootstrap Framework referenziert, sowie ein eigenes Stylesheet

7. UMSETZUNG

für weitere Anpassungen. Details zu Bootstrap und den Designentscheidungen werden später in dieser Arbeit beschrieben.

```
1  {
2    "root": "src",
3    "outDir": "src/main/resources/static",
4    "assets": [
```

Listing 4: Auszug der Datei angular-cli.json

7.2 Backend-Klassen

Wie bereits in Abschnitt 6.1 beschrieben, gibt es für jede Entität in der Datenbank eine Klasse, die sie repräsentiert. Um einzelne Datensätze aus der Datenbank zur Laufzeit in der Anwendung verwenden zu können, gibt es sogenannte DAO-Klassen. Diese gewährleisten den Zugriff auf die Datenbank und bieten außerdem Funktionen an, um bestimmte Datensätze über Abfragekriterien zu finden. Von den DAO-Klassen gelieferte Datensätze können als Instanzen der repräsentierenden Klasse weiterverwendet werden. Für die Realisierung dieser Zugriffs-Klassen wurden Spring Data JPA Repositories verwendet, die neben den grundlegenden CRUD-Operationen eine Vielzahl an möglichen Abfragen ermöglichen.

```
1  @Transactional
2  @Repository
3  public interface CategoryDao extends JpaRepository<Category, Long> {
4
5      Category findByName(String name);
6      List<Category> findByParentCategory(Category category);
7
8      @Query("SELECT c FROM Category c WHERE c.parentCategory IS NULL")
9      List<Category> getParentCategories();
10
11     @Query("SELECT c FROM Category c WHERE c.parentCategory = ?1")
12     List<Category> getChildCategories(Category category);
13
14     List<Category> findAll();
15 }
```

Listing 5: CategoryDao.java

Listing 5 zeigt eine solche DAO-Klasse. Zugriffs-Funktionen zu definieren kann auf un-

7. UMSETZUNG

terschiedliche Art und Weise passieren. Zum einen ermöglichen diese Repositories es, Suchkriterien über den Namen der jeweiligen Funktion zu definieren. So kann beispielsweise über eine Funktion `findByName(String name)` der Datensatz mit dem der Funktion übergebenen Namen gefunden werden, wenn die Tabelle eine Spalte mit der Bezeichnung „Name“ enthält. Die Zuordnung auf die entsprechende Spalte in der Tabelle und die Suchfunktionalität müssen dabei nicht vom Entwickler implementiert werden. Neben diesen semi-automatischen Funktionen können aber auch eigene Methoden definiert werden. Dafür kann über eine Annotation (`@Query`) eine SQL-Query für die Abfrage an die Datenbank definiert werden.

7.3 Objektrelationale Abbildung (ORM) mit Hibernate

Für die Zuordnung der Backend-Klassen zu den entsprechenden Tabellen in der Datenbank mit Hibernate stehen dem Entwickler eine Vielzahl von Annotationen zur Verfügung, von denen einige hier kurz beschrieben werden sollen. Die Hibernate Annotation um eine Klasse als Entität zu markieren ist denkbar einfach. Sie wird direkt über der Klassendeklaration mit `@Entity` gesetzt. Darüber hinaus können auch der Tabellen-Name sowie die Namen der einzelnen Tabellenspalten über der Deklarationen der Klasse beziehungsweise des Attributs gesetzt werden. Das ist allerdings nur für den Fall nötig, wenn diese Namen explizit vergeben werden sollen. Ansonsten werden sie von Hibernate gemäß dem Namen der Klasse in Kleinschreibung vergeben. Wenn man aber die Zuordnung für eine bereits bestehende Datenbank machen möchte, sind diese expliziten Namen sehr hilfreich. Ist eine entsprechende Tabelle beim Start der Anwendung nicht vorhanden, wird diese von Hibernate erzeugt.

Eine weitere wichtige Annotation betrifft die Primärschlüssel der Tabelle, die jeden Datensatz eindeutig identifizieren können. Mit `@Id` kann dieses Attribut gekennzeichnet werden. Empfehlenswert ist dabei die Verwendung eines künstlichen Schlüssels als Zahlenwert, der für jeden Eintrag hochgezählt wird. Auch dafür bietet Hibernate eine komfortable Lösung, indem über `@GeneratedValue` ein solcher generierter Wert definiert wird. In Kombination mit einer automatischen Inkrementierung, die MySQL für numerische Werte bietet, kann

7. UMSETZUNG

also automatisch ein eindeutiger Zahlenwert für jeden neuen Eintrag in der Tabelle gesetzt werden.

Für die Beziehungen zu anderen Entitäten können entsprechend der Kardinalität folgende Annotationen verwendet werden:

- @OneToOne. 1:1
- @OneToMany. 1:N
- @ManyToOne. N:1
- @ManyToMany. N:N

Über Parameter dieser Annotationen können neben der Ziel-Entität weitere Einstellungen (beispielsweise zum Cascading) gemacht werden, auf die hier nicht näher eingegangen werden soll.

7.3.1 Vererbung von Klassen

Eine Vererbungshierarchie von Klassen kann mit verschiedenen Strategien in der Datenbank abgebildet werden. Hibernate bietet dafür vier grundlegende Varianten an [23]:

Mapped Superclass

In dieser einfachsten Form der Abbildung einer Vererbung wird für jede konkrete Ausprägung der vererbenden Klasse eine eigene Tabelle angelegt, die neben ihren eigenen auch alle Spalten der Eltern-Klasse enthält. Die Vererbungshierarchie wird damit nicht explizit im Datenmodell ersichtlich.

Single table

Dabei werden alle erbenden Klassen in einer einzigen Tabelle in der Datenbank abgebildet. Dadurch enthält die Tabelle viele Spalten, da alle Attribute jeder Sub-Klasse enthalten sind. Für einzelne Datensätze einer bestimmten erbenden Klasse werden nicht benötigte Spalten mit „null“-Werten belegt. Um zu unterscheiden welche Datensätze zu welcher Sub-Klasse

7. UMSETZUNG

gehören, wird außerdem eine sogenannte „Discriminator Column“ definiert. Dieser Ansatz ist sehr performant, da für Abfragen immer nur eine Tabelle verwendet werden muss.

Table-per-subclass

Hier wird für die vererbende und jede erbende Klasse eine eigene Tabelle in der Datenbank erzeugt. Letzere enthalten dabei nur die Attribute, die sie zusätzlich zu den der Eltern-Klasse haben. Die Zuordnung zur Eltern-Klasse wird über JOINS hergestellt, was die Performanz bei häufigen Abfragen negativ beeinflussen kann.

Table-per-concrete-class

In dieser letzten Variante wird wie bei Table-per-subclass für alle Entitäten eine Tabelle erzeugt. Diese enthalten jedoch jeweils alle eigenen und die Attribute der Eltern-Klasse. Die Zuordnung wird über UNION-Operationen hergestellt, was ebenfalls zu Performance-Verlusten bei großen Klassen-Hierarchien führen kann.

Innerhalb des Webshops wurde für die Hierarchie bei Artikeln die „Single Table“-Strategie gewählt. Listing 6 zeigt die Klassen-Definitionen mit den Annotationen für das ORM.

```
1  /**
2   * Base Class
3   */
4  @Entity
5  @Inheritance(strategy= InheritanceType.SINGLE_TABLE)
6  @XmlRootElement
7  public class BaseItem { /*...*/ }
8
9  /**
10   * Child Class "Single"
11   */
12  @Entity
13  @DiscriminatorValue("single")
14  public class SingleItem extends BaseItem { /*...*/ }
15
16  /**
17   * Child Class "Multiple"
18   */
19  @Entity
20  @DiscriminatorValue("multiple")
21  public class MultipleItem extends BaseItem { /*...*/ }
```

Listing 6: Vererbungshierarchie

7.4 REST-Schnittstellen

Wie in Abschnitt 7.1.1 bereits erwähnt, wurde für das Erstellen der REST-Services innerhalb der Spring Boot Anwendung Jersey als Implementierung der Java-Schnittstelle JAX-RS benutzt. Für die Darstellung der zu übertragenden Ressourcen verwendet JAX-RS standardmäßig die Implementierung von JAXB (Java Architecture for XML Binding). Der Vorteil daran ist, dass die Klassen mit der Annotation `@XmlElement` automatisch bei einem REST-Aufruf in die entsprechende XML- oder JSON-Darstellung umgewandelt werden. Die Unterstützung des JSON-Formates ist dadurch gewährleistet, dass JAXB die Jackson Bibliotheken benutzt. Die Jackson Bibliothek bietet einen mächtigen Parser für die Konvertierung von Java-Objekten in JSON-Strings [19].

Über Annotationen können normale Java-Klassen zu sogenannten Endpoints gemacht werden, die Anfragen über eine URI entgegen nehmen und beantworten können. Die wichtigsten sind [24]:

- `@Path`. Legt die URI für die Web-Services der Klasse oder einer Methode fest.
- `@GET`, `@POST`, `@PUT`, `@DELETE`, `@HEAD`. Definiert die HTTP-Methode für den Aufruf einer Service-Methode.
- `@Consumes`, `@Produces`. Spezifiziert den MIME-Typ, der von der Funktion konsumiert bzw. produziert werden kann.
- `@*Param`. Wird verwendet, um Parameter bei der Anfrage auszulesen (z.B. `@PathParam` für Parameter in der URI der Anfrage).

Für die vorliegende Anwendung wurden die Endpoints thematisch separiert. Abb. 13 zeigt die definierten Endpoint-Klassen und ihre Implementierung.

Zusätzlich zu den Benutzer- und Administrator-Funktionalitäten wurden zwei weitere Endpoints für die Authentifizierung und den File-Upload erstellt.

7. UMSETZUNG

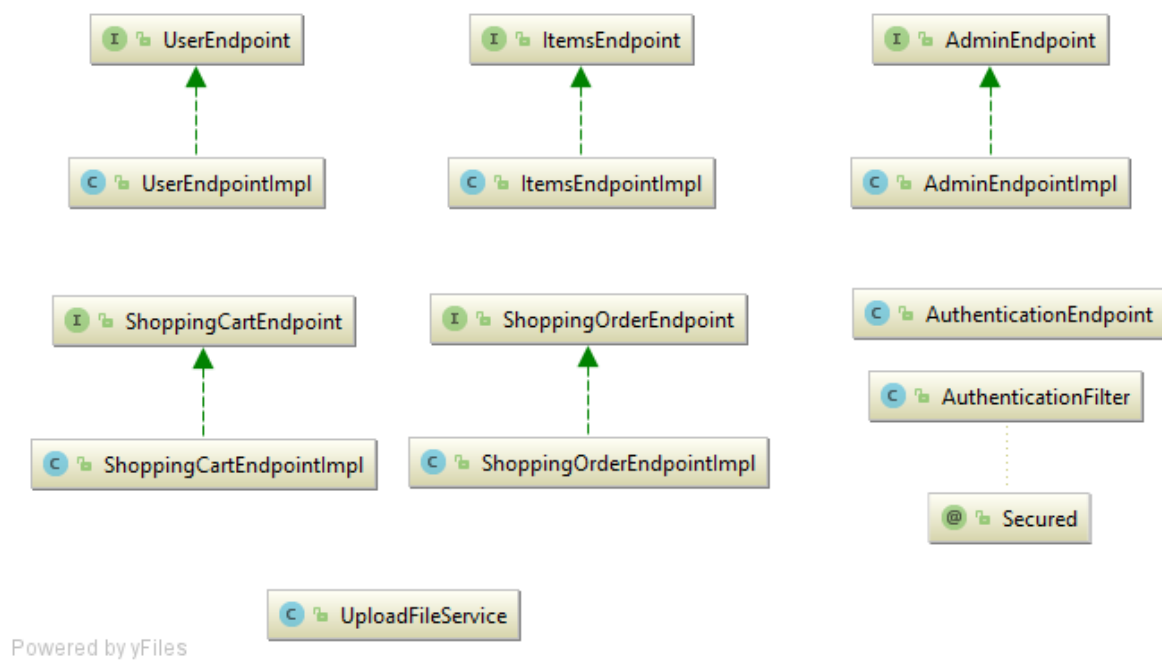


Abbildung 13: REST-Endpoints

```
1 /**
2  * Declaration of Register-method in the UserEndpoint
3  */
4 @PUT
5 @Path( "register" )
6 Response register(User user);
7
8 /**
9  * Implementation of defined Endpoint-method
10  * in UserEndpointImpl
11  */
12 @Override
13 public Response register(User user) {
14     if (userDao.findByEmail(user.getEmail()) == null) {
15         user.setPassword(BCrypt.hashpw(user.getPassword(), BCrypt.gensalt()));
16         ShoppingCart cart = new ShoppingCart();
17         cart.setUser(user);
18         userDao.save(user);
19         shoppingCartDao.save(cart);
20         return Response.ok()
21             .entity("User erfolgreich angelegt. Sie können sich jetzt mit
22                 Ihrer Email-Adresse und Ihrem Passwort anmelden").build();
23     } else return Response.status(Response.Status.BAD_REQUEST)
24         .entity("Benutzer existiert bereits").build();
25 }
```

Listing 7: REST-Beispiel

7. UMSETZUNG

Listing 7 zeigt die Deklaration der Registrierungs-Funktion für Benutzer sowie deren Implementierung als Beispiel für die Endpoint-Funktionalitäten.

7.5 Weitere Implementierungsdetails

In diesem Abschnitt werden einzelne Details zur Implementierung vorgestellt. Die dabei behandelten Themen resultierten aus der Notwendigkeit für eine bestimmte Funktionalität des Shops oder stellten ein Problem dar, für das eine Lösung gefunden werden musste.

7.5.1 Passwort-Verschlüsselung

Dass Passwörter nicht unverschlüsselt in einer Datenbank enthalten sein sollten, ist seit dem Diebstahl von persönlichen Nutzerdaten und -Zugängen bei großen Online-Anbietern nicht nur Fachleuten, sondern auch Laien bekannt. Das Bundesamt für Sicherheit in der Informationstechnik BSI gibt dafür offizielle Leitlinien heraus, in welcher Form das geschehen kann [25]. Auf die einzelnen Kryptographieverfahren und die Generierung solcher Schlüssel soll hier aber nicht weiter eingegangen werden, sondern lediglich gezeigt werden, wie die Verschlüsselung in der Anwendung umgesetzt wurde.

Dabei wurde auf das bestehende Framework jBCrypt zurückgegriffen, eine Java-Implementierung des OpenBSD Blowfish Passwort-Hashing-Algorithmus [26]. Dieser benutzt eine 64-Bit Block-Cipher, um einen Hash des Passworts zu generieren, der dann in der Datenbank gespeichert werden kann [27]. Beim Registrieren eines Nutzers wird sein Passwort über das Framework zu einem Hash-Wert verarbeitet und gespeichert. Wenn ein User sich am System anmeldet, wird das im Klartext eingegebene Passwort wiederum mit dem gespeicherten Hash-Wert verglichen, was erneut vom Framework bewerkstelligt wird.

7.5.2 Token-Authentifizierung

Um zu vermeiden, dass sich ein Benutzer für jede Anfrage an den Server immer wieder anmelden muss, wurde auf die Verwendung einer Token-Authentifizierung zurückgegriffen.

JSON Web Token (JWT) ist ein offener Standard für die sichere Übermittlung von

7. UMSETZUNG

JSON-Objekten zwischen zwei Partnern und wird häufig für die Authentifizierung bei Web-Anwendungen verwendet. Durch eine digitale Signierung mit einem Verschlüsselungs-Algorithmus sind die Tokens fälschungssicher. Außerdem können im Token selbst zusätzliche Informationen gehalten werden. Diese können beispielsweise verwendet werden, um den Benutzer zu identifizieren, wodurch bei einer erneuten Anfrage nicht zuerst wieder eine Datenbank-Abfrage nötig wird [28].

Für den Webshop wurde JWT integriert, indem beim Login ein Token erstellt und übermittelt wird. Für erneute Anfragen wird dieses mitgeschickt. Als zusätzliche Information enthält es die User-Id und die Rolle des Benutzers (User oder Admin). Bei sämtlichen REST-Endpoint-Methoden, die eine Anmeldung erfordern, wird das Token und für die Admin-Funktionalitäten zusätzlich die Rolle überprüft.

7.5.3 File Upload

Für das Hochladen und Speichern von Dateien oder Bildern durch Benutzer gibt es in Web-Anwendungen verschiedene Realisierungsmöglichkeiten. Wie bei allen Eingaben, die von Usern gemacht werden, muss dabei auch auf die Sicherheit geachtet werden, da durch solche Operationen Schaden an der Anwendung oder der Datenbank angerichtet werden können. Da in der vorliegenden Anwendungen der Upload von Bild-Daten nur von Administratoren durchgeführt werden kann, ist dieser Aspekt aber vernachlässigbar.

Um Bilder für Artikel persistent zu speichern und auch bei einer möglichen Neu-Installation der Anwendung verfügbar zu haben, wurde eine Lösung gewählt, die die Bilder mit den Artikeldaten in der Datenbank speichert. Der Datentyp Blob (Binary Large Object) ermöglicht das Speichern eines simplen binären Byte-Arrays, das beim Upload direkt als Stream geschrieben werden kann. Eine dafür erstellte REST-Endpoint-Funktion liefert dann für einen Artikel das entsprechende Bild zurück, das im Frontend angezeigt werden kann.

7.6 Frontend

Dieser Abschnitt behandelt die Details zur Implementierung der Angular 2 Anwendung. Die Gliederung ist so aufgebaut, dass einige der in Abschnitt 5.2 beschriebenen Architekturelemente mit der konkreten Umsetzung im Rahmen dieses Projekts vertieft werden.

7.6.1 Module

Es ist weitestgehend den Entwicklern überlassen, welche Funktionalitäten in welcher Form zu Angular-Modulen gekapselt werden. Prinzipiell wäre es auch möglich, eine Anwendung ganz ohne Module zu erstellen (abgesehen vom bereits erwähnten, obligatorischen Root-Modul). Im Webshop wurden jedoch der Übersichtlichkeit halber zwei Module definiert: ein Shop- und ein Admin-Modul. Somit ist die Trennung zwischen diesen Bereichen auch in der Implementierung gegeben.

Beide Module enthalten eine Modul-Klasse, eine Basiskomponente mit Template, sowie ein Routing-Modul, das importiert wird. Die Aufgabe der Routing-Module ist es, beim Aufruf einer bestimmten URL der Webseite eine dazu zugewiesene Komponente in einen Platzhalter des DOMs zu laden.

```
1  const routes: Routes = [  
2    { path: 'shop', component: WebshopComponent, children: [  
3      { path: 'category/:id', component: ItemsComponent },  
4      { path: 'cart', component: ShoppingCartComponent, canActivate: [  
5        AuthGuard]],  
6      { path: 'account', component: AccountComponent, canActivate: [  
7        AuthGuard]],  
8      { path: 'login', component: LoginComponent },  
9      { path: 'register', component: RegisterComponent }  
10   ]},  
11 ];  
12 @NgModule({  
13   imports: [RouterModule.forChild(routes)],  
14   exports: [RouterModule]  
15 })  
16 export class WebshopRoutingModule { }
```

Listing 8: Routing-Module

Listing 8 zeigt einige der Routen des Shop-Moduls. Wie in Zeile 2 zu sehen ist, sind alle

7. UMSETZUNG

URLs des Webshops unter dem Pfad „shop“ definiert, der die Hauptkomponente des Moduls repräsentiert. Im Template dieser Komponente ist der bereits erwähnte Platzhalter für die Kind-Komponenten enthalten. Die Eigenschaft `canActivate` der Pfade sagt aus, dass diese URL geschützt ist und ausschließlich angemeldete Benutzer darauf zugreifen können. Dafür wird eine Funktion ausgeführt die überprüft, ob bereits ein Benutzer angemeldet ist und im negativen Fall eine Weiterleitung zur Login-Komponente tätigt.

7.6.2 Komponenten

Die Anwendung ist so aufgebaut, dass für den Inhalt jeder Seite eine Komponente existiert. Diese kann wiederum aus weiteren Komponenten bestehen, die in das Template eingebunden werden. Ein Beispiel dafür ist die Produktaufstellung zu jedem Kategorie-Reiter. Alle Seiten dieser Art sehen wie auf Abb. 14 aus. Die roten Markierungen identifizieren die zwei Kind-Komponenten, die eingebunden sind.

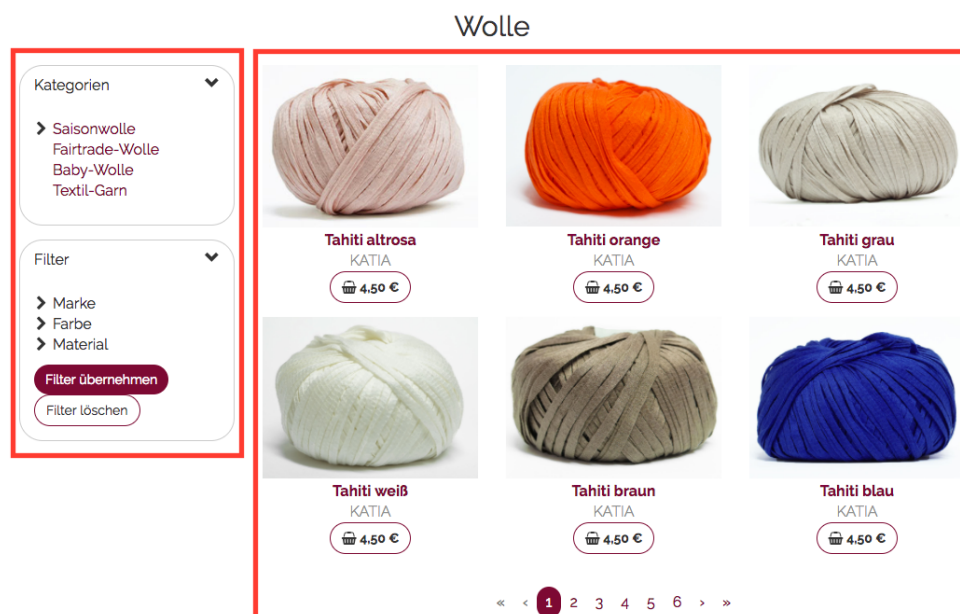


Abbildung 14: Inhalt der Seite für die Kategorie Wolle

Wie diese Seite realisiert wurde ist in Listing 9 veranschaulicht. Die Einbindung der Unterkomponenten erfolgt jeweils in Zeilen 4 und 8. Die Attribute in orangefarbener Schrift sind sogenannte *Inputs* der Komponente, eine Form des Data-Binding, die in Abschnitt 5.2 bereits beschrieben wurde.

7. UMSETZUNG

```
1 <h2 class="text-center">{{category.name}}</h2>
2 <div class="row">
3   <div class="col-lg-3 col-md-3 col-sm-4 col-xs-12">
4     <items-sidebar [categories]="categories" [filters]="filters" ></
      items-sidebar>
5   </div>
6   <div class="col-lg-9 col-md-9 col-sm-8 col-xs-12">
7     <items-grid [items]="items"></items-grid>
8   </div>
9 </div>
```

Listing 9: Template für die Produklisten-Komponente

7.6.3 Services

Alle Methoden zum Aufruf der REST-Schnittstellen sind in mehrere Services gekapselt. In der Angular Anwendung gibt es einen Service für jede auf Abb. 13 dargestellte Schnittstelle, sozusagen als Gegenstück davon. Listing 10 zeigt beispielsweise die Methode zum Aufruf des Warenkorbes eines Benutzers.

```
1 getShoppingCart() {
2   // add authorization header with jwt token
3   let headers = new Headers({ 'Authorization': 'Bearer ' + this.
      authenticationService.token });
4   let options = new RequestOptions({ headers: headers });
5
6   this.http.get('/api/v1/shopping-cart', options)
7     .map((response: Response) => response.json())
8     .subscribe(
9       data => {
10         this.shoppingCart = data;
11         this.shoppingCart.items = _.sortBy(this.shoppingCart
            .items, ['iq_id']);
12         this.shoppingCartUpdate.next(this.shoppingCart);
13       },
14       error => {
15         console.log(error);
16       }
17     );
18 }
```

Listing 10: Methode für den Aufruf des Warenkorbes

In den Zeilen 3 und 4 von Listing 10 wird der Authentifizierungs-Header für die HTTP-Anfrage vorbereitet. Der Header besteht aus einem JWT-Token (siehe Abschnitt 7.5.2), der beim Login in der Anwendung gespeichert wird. Die Anfrage selbst beginnt in Zeile 6, mit

7. UMSETZUNG

dem Aufruf der GET-Methode. Diese und alle weitere HTTP-Methoden sind in Angular durch ein eigenes Modul gewährleistet. Wie man sieht erhält die Methode als Parameter die URL zum gewünschten REST-Endpoint und die Header. Der Rückgabewert des asynchronen Aufrufs ist ein sogenanntes *Observable*-Objekt der RxJS-Bibliothek. Diese Bibliothek implementiert in JavaScript das Observer-Pattern, ein Verhaltensmuster das zur Weitergabe von Änderungen an einem Objekt an andere beobachtenden Objekte dient [29].

Zwei der wichtigsten Methoden eines RxJS Observable-Objekts sind in Listing 10 zu sehen. Zum einen die `map()`-Funktion in Zeile 7 und zum anderen `subscribe()` in der darunterliegenden Zeile. Erstere dient zur Transformation der empfangenen Daten, in diesem Fall das Parsen von den Daten zu einem JSON-Objekt. Die `subscribe()`-Funktion ermöglicht es, das Observable-Objekt im Hinblick auf jegliche Änderungen zu beobachten (im dargestellten Beispiel die Daten aus dem asynchronen Aufruf und eventuelle Fehlermeldungen). Sobald eine Änderung erkannt bzw. eine Antwort vom Server erhalten wird, können beliebige Aktionen mit den Daten durchgeführt werden.

Ein weiterer Anwendungsfall des Observer-Patterns in Angular-Anwendungen sind die Interaktionen zwischen Komponenten, die nicht direkt miteinander verwandt sind (keine Eltern-Kind-Beziehung). Auch davon ist in Listing 10 ein Beispiel zu sehen. Das `shoppingCartUpdate`-Objekt in Zeile 12 ist im Service als ein sogenannter *EventEmitter* deklariert. Andere Komponenten und Services der Anwendung können mit `subscribe()` dieses Objekt beobachten. Wird wie in Zeile 12 die Methode `next()` mit einem neuen Wert aufgerufen, bekommen alle Beobachter die Änderung mit.

7.6.4 Styling

In Abschnitt 7.1.2 wurde bereits erwähnt, dass im Frontend das Bootstrap Framework für das Design der Oberfläche verwendet wird. Grund dafür ist, dass Bootstrap besonders für die Entwicklung von Anwendungen mit Responsive Design konzipiert ist. Traditionell wurde die Aufteilung der Elemente auf einer Webseite mit Tabellen gemacht, die jedoch schwer zu skalieren sind. Die Lösung von Bootstrap ist das sogenannte *Grid System*. Hierbei ist die Oberfläche in Zeilen mit jeweils zwölf virtuellen Spalten mit skalierbarer Breite aufgeteilt.

7. UMSETZUNG

Listing 9 auf Seite 44 zeigt, wie das Grid System eingesetzt werden kann. Mit der CSS-Klasse `col` können beliebige Spalten-Kombinationen für unterschiedliche Display-Größen erstellt werden.

8 Deployment

Im Verlauf der Entwicklung stellte sich intern das Bedürfnis nach einer durchgehend lauffähigen Online-Demo des Webshops heraus. Dabei sollten alle Änderungen am Projekt automatisch kompiliert und in das Demo-System eingespielt werden. Um dies umsetzen zu können, wurde eine Deployment-Strategie definiert, die in diesem Abschnitt näher erläutert wird.

8.1 Versionsverwaltung mit Git

Für die Versionsverwaltung des Projekts wird Git verwendet, eines der beliebtesten Versionsverwaltungssysteme in der Softwareentwicklung. Als zentrale Ablage des Quellcodes wurde ein Repository in der GitHub-Plattform erstellt, sodass die Teammitglieder bequem ihre Änderungen an das Repository senden und auch die neuesten Änderungen erhalten können [30]. Neben den allgemeinen Vorteilen von Git und GitHub ist deren Einsatz auch eine Voraussetzung für die nächsten Schritte.

8.2 Travis Continuous Integration

Die Travis CI (Continuous Integration) Software ermöglicht es, automatisch bei einem Commit auf das GitHub-Repository die Anwendung zu bauen (inklusive Test-Suites falls solche konfiguriert sind). Zudem kann Travis auch so eingestellt werden, dass nach einem erfolgreichen Build der Code direkt in die gewünschte Umgebung deployed wird [30].

Die Einrichtung von Travis ist in wenigen Schritten erledigt:

1. Je nachdem, ob es sich um ein Open Source oder ein privates Repository handelt, ist eine Anmeldung mit dem GitHub-Konto über entweder *travis-ci.org* oder *travis-ci.com* nötig. In diesem Fall handelt es sich um ein Open Source Projekt.
2. In der Profilübersicht sind alle Repositories des angemeldeten Benutzerkontos aufge-

8. DEPLOYMENT

listet. Anhand eines Buttons kann Travis für das entsprechende Repository aktiviert werden. Dadurch wird ein sogenanntes „Webhook“ gesetzt, welches zukünftige Commits erkennt und automatisch einen Build ausführt.

3. Im Hauptverzeichnis des Repositories muss eine Datei mit dem Namen `.travis.yml` erstellt werden. Diese Datei teilt Travis mehrere Informationen mit, die für den Build benötigt werden. Das sind beispielsweise die verwendete Programmiersprache, weitere Befehle die ausgeführt werden sollen und die Deployment-Konfigurationen.

Der Inhalt der `.travis.yml`-Datei für dieses Projekt ist in Anhang C zu finden. Ein wichtiger Schritt des automatisierten Builds ist die Ausführung des `ng build` Befehls von Angular, der bereits in Abschnitt 7.1.2 erwähnt wurde. Darauf folgt ein automatischer Commit der generierten Ressourcen für die Angular-Anwendung und zuletzt das Deployment der gesamten Applikation auf die Server-Instanz (in diesem Fall von Heroku).

8.3 Heroku

Ursprünglich war es gedacht, einen eigenen privaten Webserver für das Deployment des Webshops einzurichten. Es wurde jedoch schnell klar, dass der Aufwand im Verhältnis zur sehr geringen erreichten Performanz viel zu hoch wurde. Dadurch fiel die Entscheidung darauf, eine Cloud-Lösung zu verwenden. Das geeignetste Service-Modell in diesem Anwendungsfall war *Platform-as-a-Service* (PaaS). Diese Dienstleistung stellt alle notwendige Infrastrukturkomponenten wie Server, Speicher und Netzwerkelemente zur Verfügung, sowie zusätzlich Middleware, Entwicklungstools, BI-Dienste (Business Intelligence), Datenbank-Verwaltungssysteme und mehr [31]. Zur Veranschaulichung sind in Abb. 15 die Unterschiede zwischen den drei Servicemodellen des Cloud Computings dargestellt: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) und Infrastructure-as-a-Service (IaaS).

Im PaaS-Umfeld ist Heroku einer der beliebtesten Anbieter. Das Erstellen eines virtuellen Webserver (in Heroku *Dyno* genannt) ist sehr einfach und auch das automatische Deployment von Travis nach Heroku funktioniert reibungslos. Zudem werden inzwischen einige Programmiersprachen und auch Frameworks wie Spring unterstützt [30].

8. DEPLOYMENT

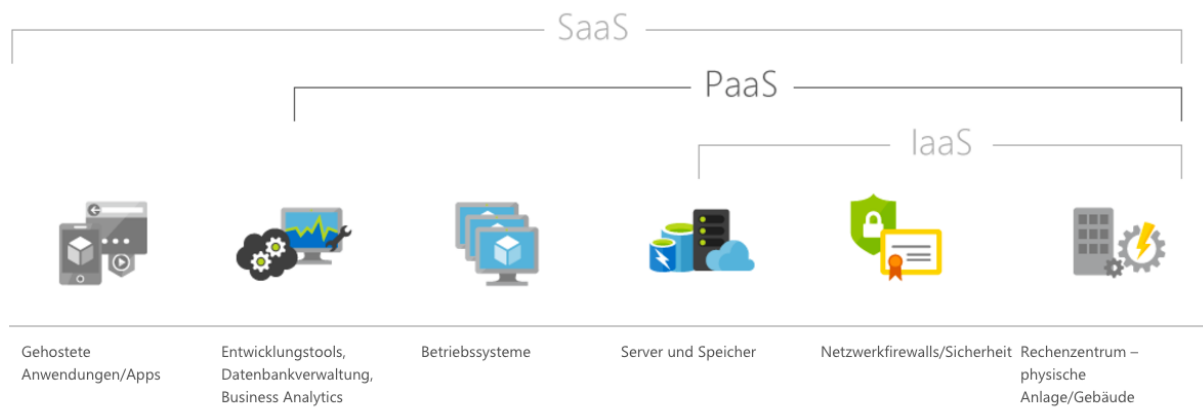


Abbildung 15: Bestandteile der Cloud-Service-Modelle [31]

Wie bei vielen anderen Cloud-Anbietern werden bei Heroku unterschiedliche Tarife vermarktet, die entsprechend mehr oder weniger Dienstleistungen umfassen. Für dieses Projekt war der kostenlose Tarif völlig ausreichend. Dieser enthält einen Dyno mit 512MB RAM, was für den Anwendungsfall genügt.

8.4 AWS Datenbank Instanz

Auch hier war es ursprünglich geplant, einen privaten Datenbank-Server einzurichten und verfügbar zu machen, jedoch waren letztendlich die Performanz und die Stabilität des Servers annehmbar. Eine interessante Alternative dazu bot Amazon Web Services (AWS), bzw. konkret deren Amazon Relational Database Service (Amazon RDS). Das Produkt ermöglicht es, relationale Datenbanken einfach zu erstellen, zu verwalten und zu skalieren [32]. Über die Weboberfläche von Amazon RDS konnte für das Projekt in wenigen Schritten eine kostenfreie MySQL Datenbank-Instanz mit 5GB Speicherplatz und 1GB RAM angelegt werden.

9 Fazit

Dieses Kapitel soll abschließend ein Fazit des Projekts und der entstandenen Anwendung ziehen. Dabei soll auf Probleme eingegangen werden, sowie in einem Ausblick gezeigt werden, was über den Projekt-Umfang hinaus weiter getan werden könnte. Zuletzt soll eine kritische Bewertung des Projekts erfolgen.

9.1 Hürden und Stolpersteine

Exemplarisch werden hier einige Schwierigkeiten, die während des Projekts auftraten und ihre jeweilige Lösung beschrieben.

9.1.1 Artikel-Bilder

Anfangs wurden die Bilder für die Artikel in ein Verzeichnis innerhalb der Anwendung gespeichert. Das stellt für die Artikel, die bereits bei Auslieferung der Anwendung vorhanden sind, auch kein Problem dar. Allerdings werden dann neue Bilder, die im Zuge des Anlegens von Artikeln in dieses Verzeichnis hochgeladen werden, nur so lange gespeichert, wie die Anwendung unverändert auf dem Server liegt. Sobald durch ein Deployment der Webshop neu auf den Server gelegt wird, sind diese Bilder nicht mehr vorhanden, da sie nicht Teil der verwalteten Anwendung sind.

Demnach wurde in einer Anpassung ein Verzeichnis außerhalb der Anwendung definiert, was diesen Umstand umging, allerdings neue Probleme mit sich brachte. Die Benutzung relativer und absoluter Pfade für eine Dateireferenzierung ist problematisch, da sie je nach der Umgebung der Anwendung variieren können und damit fehleranfällig sind und eine manuelle Anpassung nötig machen können.

Also wurde nach einiger Recherche die Variante gewählt, die Bilder direkt in der Datenbank als Blob zu speichern. Diese Lösung macht die Bilder unabhängig von der Anwendung und durch die Speicherung in der Datenbank genauso persistent wie alle anderen Ressourcen.

9. FAZIT

9.1.2 Kategorien-Hierarchie

Die Hierarchie der Kategorien ist eine elegante Möglichkeit, Artikel zu gruppieren, brachte aber auch einiges an Aufwand und Schwierigkeiten mit sich. Je nach Verschachtelungstiefe wird es zunehmend schwieriger, die Kategorien zu verwalten und herauszufinden, welche Artikel in einer Kategorie und allen ihren Kinder-Kategorien vorhanden sind. Da die Anzahl dieser Hierarchie-Ebenen bei der Planung nicht definiert wurde, wurden für die Abfrage der Artikel einer Kategorie rekursive Funktionen nötig, um dynamisch je nach Verschachtelungstiefe alle Artikel abzurufen. Dieser Umstand wirkt sich auf Abfragen der obersten Hierarchie negativ auf die Performanz aus. Eine Festlegung der Ebenen oder eine zusätzliche Abbildung der Hierarchie in der Datenbank hätte den Umgang damit erleichtert.

9.1.3 Filterung

Auch für die Filterung von Artikeln nach bestimmten Merkmalen musste während der Entwicklung eine Lösung gefunden werden. Nach einiger Überlegung, wie das bewerkstelligt werden kann, fiel die Entscheidung für eine Umsetzung im Frontend. Dank Angular und TypeScript konnte eine solche Logik dort implementiert werden, die man sonst eher im Backend ansiedeln würde. Dennoch war diese Realisierung mit einigem Aufwand verbunden, um die dafür benötigten Informationen jeweils dynamisch auszulesen und für die Filterung verfügbar zu machen.

9.2 Ausblick

In diesem Abschnitt werden einige Aspekte beschrieben, die für eine Weiterentwicklung des Webshop sinnvoll oder wünschenswert wären.

9.2.1 Suche

Eine Volltext-Suche zu implementieren, wäre für viele Aspekte des Shops und des Admin-Bereichs hilfreich. Diese sollte in der gesamten Anwendung verfügbar gemacht werden, um zum Beispiel nach Artikeln über Schlagworte zu suchen oder im Admin-Bereich nach Usern

9. FAZIT

über den Namen. Einige solcher Such-Möglichkeiten wurden jeweils in den DAO-Klassen als Funktionen definiert. Um diese Funktionalität aber nicht für alle Entitäten einzeln umzusetzen, wäre eine solche globale Suche wünschenswert und würde sich auch im Hinblick auf einen wachsenden Shop lohnen.

9.2.2 Artikel und Angebote

Für einen Webshop im Produktiv-Betrieb wäre es attraktiv, wenn man bestimmte Inhalte dynamisch anpassen könnte. So könnten beispielsweise Artikel im Preis reduziert und hervorgehoben oder bestimmte Verkaufs-Aktionen durchgeführt werden. Solche Aktionen könnten jeweils auf der Startseite platziert werden, um eine Umsatz-Steigerung zu erzielen. Das würde auch bei der Kunden-Bindung helfen, da regelmäßige Kunden bei einer solchen dynamischen Gestaltung des Shops auch öfter die Seite besuchen.

9.2.3 Admin-Bereich

Im Admin-Bereich sind viele Erweiterungen denkbar. Eine Verwaltung der Kategorien wäre eine sinnvolle Erweiterung der Anpassungs-Möglichkeiten für Administratoren. Ebenso könnten die eben angesprochenen Aktionen oder Rabatte über den Admin-Bereich eingespielt werden, damit dafür nicht immer Änderungen am statischen Source-Code der Oberfläche gemacht werden müssen.

9.3 Kritische Würdigung

Zuletzt soll der Gesamtablauf des Projekts bewertet werden.

9.3.1 Spezifikation

Die Definition der Anforderungen und die Planung der Umsetzung hatte zwar gut funktioniert, brachte aber im Laufe des Projekts einige Lücken ans Licht, sodass nach-spezifiziert werden musste. Fehlende Aspekte bei der Spezifikation bringen häufig später bei der Entwicklung einen Mehr-Aufwand mit sich oder bringen den Ablauf ins Stocken. Trotz dieser

9. FAZIT

Lücken lief die Implementierung aber ohne schwerwiegende Probleme und auch die Zeitplanung konnte größtenteils eingehalten werden.

9.3.2 Arbeitsaufteilung und Tools

Die generelle Unterteilung in Front- und Backend-Entwicklung machte die Verantwortlichkeiten klar. Trotzdem waren beide Projektmitglieder über die Arbeit des anderen informiert und halfen sich gegenseitig aus. Gegen Ende der Umsetzung waren dann beide mit Themen im Frontend und Anpassungen oder Bugfixes beschäftigt. Die Verwendung von Git als Versionsverwaltungs-System hat einen reibungslosen Ablauf beim parallelen Arbeiten ermöglicht. Durch das Continuous Deployment konnten Änderungen quasi in Echtzeit auf dem Server getestet werden.

9.3.3 Lerneffekt

Die Verwendung von Angular 2 brachte einen großen Lerneffekt mit sich und hat nach einer gewissen Eingewöhnungsphase sehr gut funktioniert. Auch der Umgang mit diversen Tools rund um den gesamten Entwicklungsprozess haben einige gute Erfahrungen mit sich gebracht. Zusammenfassend kann der Lerneffekt als sehr hoch bezeichnet werden und als gute Erfahrung für zukünftige Projekte gesehen werden.

Literatur

- [1] Ulrich Riehm. *E-Commerce - Begriff, Geschichte, aktueller Stand und Ausblick*. URL: <http://www.itas.kit.edu/pub/v/2004/rieh04b.pdf> (besucht am 30.05.2017).
- [2] Casimir Saternos. *Client-Server Web Apps with JavaScript and Java*. O'Reilly Media, 2014.
- [3] Jim Conallen. *Building Web applications with UML*. Addison-Wesley Object Technology, 2000.
- [4] Techopedia. *Three-Tier Architecture*. 2017. URL: <https://www.techopedia.com/definition/24649/three-tier-architecture> (besucht am 08.05.2017).
- [5] CodeSchool, Hrsg. *Beginner's Guide to Web Development*. 2016.
- [6] Margaret Rouse. *Web server*. 2012. URL: <http://whatistechtarget.com/definition/Web-server> (besucht am 15.05.2017).
- [7] ITWissen.info, Hrsg. *Web Application Server*. 2013. URL: <http://www.itwissen.info/Web-Application-Server-web-application-server.html> (besucht am 15.05.2017).
- [8] solid IT GmbH. *DB-Engines Ranking von Relational DBMS*. URL: <https://db-engines.com/de/ranking/relational+dbms> (besucht am 30.05.2017).
- [9] solid IT GmbH. *Datenbankmanagementsysteme*. URL: <https://db-engines.com/de/systems> (besucht am 30.05.2017).
- [10] Sascha Thattil. *Die wichtigsten Technologien für die Entwicklung von Webanwendungen*. 2016. URL: <http://www.yuhiro.de/technologien-fuer-die-entwicklung-von-webanwendungen/> (besucht am 24.05.2017).
- [11] Gil Fink und Ido Flatow. *Pro Single Page Application Development*. Apress, 2014.
- [12] Craig Walls. *Spring Boot in Action*. Manning, 2015.
- [13] Phil Webb. *Spring Boot – Simplifying Spring for Everyone*. 6. Aug. 2013. URL: <https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone> (besucht am 15.05.2017).

LITERATUR

- [14] Pablo Deeleman. *Learning Angular 2*. PACKT Publishing, 2016.
- [15] Angular.io. *Architecture overview documentation*. 2017. URL: <https://angular.io/docs/ts/latest/guide/architecture.html> (besucht am 22.05.2017).
- [16] Angular.io. *NgModules Documentation*. 2017. URL: <https://angular.io/docs/ts/latest/guide/ngmodule.html> (besucht am 28.05.2017).
- [17] Balaji Varanasi und Sudha Belida. *Spring REST*. Apress, 2015.
- [18] Zenithmedia. *Zenith forecasts 75 percent of internet use will be mobile in 2017*. 2016. URL: <https://www.zenithmedia.com/mobile-forecasts-75-internet-use-will-mobile-2017> (besucht am 26.05.2017).
- [19] Oracle. *Jersey 2.22.1 User Guide*. 2015. URL: <https://jersey.java.net/documentation/latest/user-guide.html> (besucht am 19.05.2017).
- [20] Phillip Webb u. a. *Spring Boot Reference Guide*. 2017. URL: <http://docs.spring.io/spring-boot/docs/current/reference/html/> (besucht am 19.05.2017).
- [21] The Apache Software Foundation, Hrsg. *Introduction to the POM*. 2017. URL: <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html> (besucht am 19.05.2017).
- [22] Sumit Arora. *Angular CLI Project*. 2017. URL: <https://github.com/angular/angular-cli/wiki> (besucht am 19.05.2017).
- [23] Vlad Mihalcea u. a. *Hibernate ORM 5.2.10.Final User Guide*. 2017. URL: http://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html (besucht am 26.05.2017).
- [24] Jersey. *JAX-RS Application, Resources and Sub-Resources*. URL: <https://jersey.github.io/documentation/latest/jaxrs-resources.html> (besucht am 29.05.2017).

LITERATUR

- [25] Bundesamt für Sicherheit in der Informationstechnik. *Technische Richtlinie - Kryptographische Algorithmen und Schlüssellängen*. 2016. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile (besucht am 28.05.2017).
- [26] Damien Miller. *bcrypt*. 2015. URL: <http://www.mindrot.org/projects/jBCrypt/> (besucht am 28.05.2017).
- [27] Niels Provos und David Mazières. *A Future-Adaptable Password Scheme*. URL: <https://www.openbsd.org/papers/bcrypt-paper.pdf> (besucht am 28.05.2017).
- [28] Auth0 Inc. *JSON Web Tokens (JWT) in Auth0*. URL: <https://auth0.com/docs/jwt> (besucht am 29.05.2017).
- [29] Angular.io. *Angular HTTP documentation*. 2017. URL: <https://angular.io/docs/ts/latest/guide/server-communication.html> (besucht am 31.05.2017).
- [30] Mike Coutermarsh. *Heroku Cookbook*. PACKT Publishing, 2014.
- [31] Microsoft Azure. *Was ist PaaS?* 2017. URL: <https://azure.microsoft.com/de-de/overview/what-is-paas/> (besucht am 30.05.2017).
- [32] Biff Gaut u. a. *AWS Certified Solutions Architect Official Study Guide*. Amazon Web Services, 2016.

Appendices

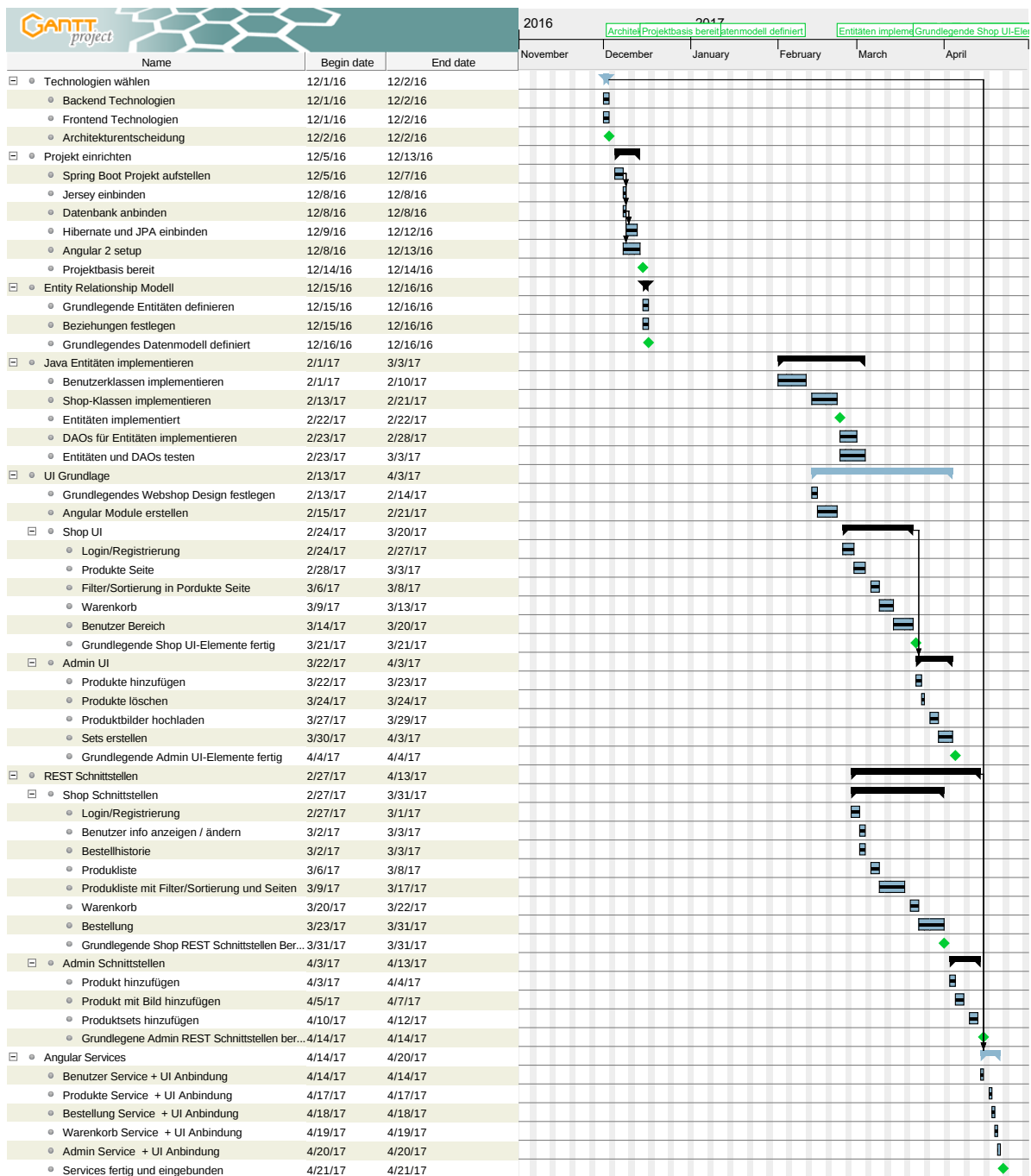
A Meilensteine

Erstellen eines Webshops

May 30, 2017

Gantt Chart

4



1. Juni 2017

57

B Design-Mockups

B.1 Produktliste einer Kategorie

Kategorie

Kategorien ▾
› Subkategorie 1
Subkategorie 2

Filter ▾
☐ Filter1
☐ Filter2
☐ Filter3

Sortierung ▾

Produktname Marke Preis	Produktname Marke Preis	Produktname Marke Preis
Produktname Marke Preis	Produktname Marke Preis	Produktname Marke Preis

<< < 1 2 3 4 > >>

B.2 Kontoübersicht

Kontoübersicht

Benutzerinformationen

Max Mustermann
mmuster@mail.de

E-Mail
ändern

Passwort
ändern

Rechnungsadresse

Max Mustermann
Musterstraße 1
12345 Musterstadt
Deutschland

Adresse ändern

Lieferadresse

Max Mustermann
Musterstraße 1
12345 Musterstadt
Deutschland

Adresse ändern

Bestellhistorie

Datum	Bestellungs-Nr.	
01.01.2017	1	<div>Details</div>
01.01.2017	2	<div>Details</div>
01.01.2017	3	<div>Details</div>

<< < **1** 2 3 4 > >>

B.3 Warenkorb

Warenkorb



Produkt A
Artnr.: 123456

- 1 + 30€ x



Produkt B
Artnr.: 123456

- 1 + 30€ x



Produkt C
Artnr.: 123456

- 1 + 30€ x

Zusammenfassung

Summe: 60,00€

Versandkosten: 0,00€

Gesamtsumme: 60,00€

Zur Kasse

B.4 Bestellmaske

Bestellung prüfen


Rechnungsadresse
Max Mustermann
Musterstraße 1
12345 Musterstadt
Deutschland

Adresse ändern

Lieferadresse
Max Mustermann
Musterstraße 1
12345 Musterstadt
Deutschland


Adresse ändern

Zahlungsart
Rechnung




Produkt A
Artnr.: 123456

- 1 + 30€ x



Produkt B
Artnr.: 123456

- 1 + 30€ x



Produkt C
Artnr.: 123456

- 1 + 30€ x

Zusammenfassung

Summe: 60,00€
Versandkosten: 0,00€
Gesamtsumme: 60,00€

Kostenpflichtig bestellen

1. Juni 2017

61

C Travis

YAML-Datei

```
1 sudo: true
2 language: java
3 jdk:
4   - oraclejdk8
5
6 env:
7   - TRAVIS_NODE_VERSION="7"
8
9 install:
10  - rm -rf ~/.nvm && git clone https://github.com/creationix/nvm.git ~/.nvm && (cd ~/.nvm && git checkout `git describe --abbrev=0 --tags`)&& source ~/.nvm/nvm.sh && nvm install $TRAVIS_NODE_VERSION
11  - npm install -g @angular/cli
12  - npm install
13
14 script: ng build
15
16 before_deploy:
17   - git config --global user.email "webshop_dev@mail.com"
18   - git config --global user.name "Travis CI"
19   - STASH_SAVE=$(git stash save)
20   - git checkout master
21   - if [[ "$STASH_SAVE" != "No local changes to save" ]]; then git stash pop; fi
22   - git add -A
23   - git diff-index --quiet --cached HEAD || git commit -m "Travis build [ci skip]"
24   - git push origin master
25
26 deploy:
27   provider: heroku
28   api-key:
29     secure: c8e184c6-1cae-4029-bdad-27b1a4c566ea
30   strategy: git
31   app: studienarbeit-webshop
```

Listing 11: Travis CI YAML-Datei