Entwurfsdokumentation
SWARM Composer
-
Softwareprojekt SoSe18
WSP31

Lorenz Boguhn
Lennart Brandt
Arved Hansen
Matthias Johannsen
Lars Jürgensen
Birger Thormählen
Björn Vonheiden
Thomas Zacher

9. September 2018

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Dokumentaufbau

In dieser Entwurfsdokumentation beschreiben wir die technische Ausführung, der im Pflichtenheft beschriebenen Funktionen, des SWARM Composers. Hierfür stellen wir erst die Komponenten dar und zeigen dann, wie diese später auf der Hardware verteilt werden. Danach beschreiben wir mithilfe mehrerer Klassendiagramme die innere Struktur unserer Software. Im fünften Kapitel, in den Sequenzdiagrammen, spezifizieren wir die detaillierte Ausführung komplexer Anwendungsfälle. Weiter haben wir unsere API-Schnittstelle designet auf die die Clients zugreifen. Zum Abschluss erklären wir im Glossar noch einige Abkürzungen und Begriffe.

1.2 Zweckbestimmung

Das Produkt soll einem die Möglichkeit bieten sich schnell über die Kompatibilität von verschiedenen Produkten zu informieren. Dabei soll das System besonders leicht und intuitiv bedienbar sein. Um dies zu verwirklichen, ist die Webseite speziell darauf ausgelegt, passende Kombinationen zu entwickeln und Inkompatibilitäten leicht durch Alternativen zu ersetzten. Im Gegensatz dazu ist die App für Präsentationen von Diensten und Kombinationen konzipiert. Zudem ist es ein Ziel die Verwaltung der Produkte leicht zu gestalten.

1.3 Entwicklungsumgebung

Software	Version	URL
Docker	17.03.2-ce	`https://www.docker.com`
Git	-	`https://git-scm.com`
GitLab	-	`https://git.informatik.uni-kiel.de`
Java Development Kit	8u144	`http://www.oracle.com/technetwork/java/javase/downloads/index.html`
Jenkins	-	`http://134.245.1.240:9002`
Jira	-	`http://maui.se.informatik.uni-kiel.de:58080`
Tomcat	-	`http://134.245.1.240:9001`

Tabelle 1.1: Enwicklungsumgebung und Tools allgemein

Software	Version	URL
Bootstrap	4.1.1	`https://getbootstrap.com/`
Fabric.js	2.3.6	`http://fabricjs.com/`
IntelliJ IDEA Ultimate	2018.2	`https://www.jetbrains.com/idea/`
jQuery	3.3.1	`https://jquery.com`
Maven	3.5.4	`https://maven.apache.org`
Spring Boot	2.0.4	`https://spring.io/projects/spring-boot`
Spring Framework	5.0.8	`https://spring.io/projects/spring-framework`
Thymeleaf	3.0.9	`https://www.thymeleaf.org`

Tabelle 1.2: Enwicklungsumgebung Webapplikation

Software	Version	URL
Android Studio	3.1.4	`https://developer.android.com/studio/`
Gradle	4.10	`https://gradle.org`

Tabelle 1.3: Enwicklungsumgebung App

Kapitel 2

Komponentendiagramme

2.1 Server

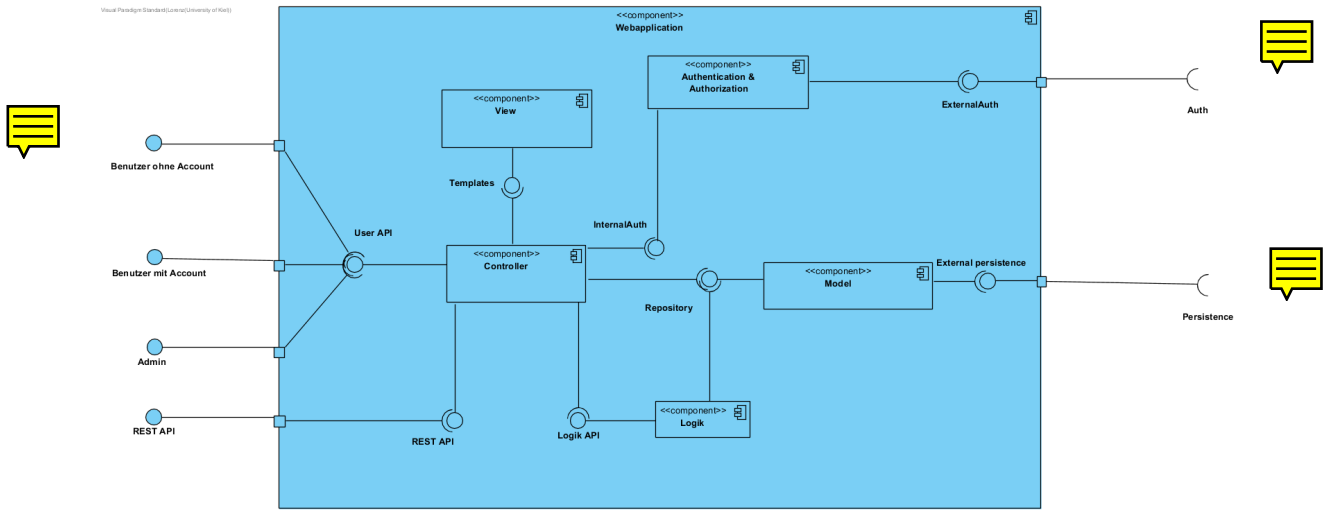

Abbildung 2.1: Komponentendiagramm - Server

Komponentenname	Aufgabe
Authentification & Authorization	Authentifizierung und Logins/Logouts
Controller	Beantwortet Anfragen von außen und übernimmt die interne Kommunikation
Logik	Berechnet Lösungen für Anfragen, die über die einfache Logik des Controllers hinaus geht
Model	Enthält alle Daten aus der DB und bietet sie über eine Schnittstelle an
View	View bereitet die Daten in einer HTML Seite auf und enthält Funktionalität für die Webseite

Tabelle 2.1: Subkomponentenbeschreibung - Server

Die meisten Schnittstellen aus dieser Abbildung werden bereits durch das Spring-Framework bereitgestellt. In diesem System stellen die Webseitennutzer über die User API Anfragen an den Controller. Die App-Nutzer nutzen dafür die REST API. Der Controller wertet die Anfrage aus und bearbeitet diese mit Hilfe der anderen Komponenten. Dabei ist die Authentification & Authorization-Komponente dafür zuständig, zu überprüfen ob eine Anfrage authentifiziert ist und auch für Logins und Logouts. Die View-Komponente ist hier bei für die Aufbereitung der Informationen in HTML zuständig. Weiter ist dort Funktionalität/Logik für die Browserwebseite mit enthalten. In der Logik-Komponente werden komplexere Berechnungen durchgeführt. So werden hier beispielsweise Alternativen für inkompatible Verbindungen. Die Model-Komponente ist lediglich für die Datenverwaltung und die Kommunikation mit der DB zuständig. Dafür stellt die Model-Komponente die Repository-Schnittstelle bereit.

2.2 App

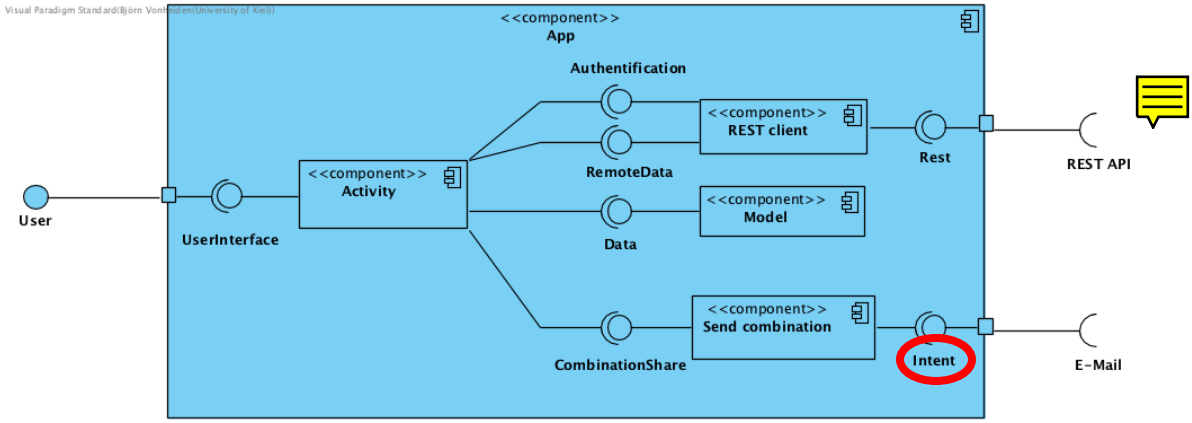

Abbildung 2.2: Komponentendiagramm - App

Komponentenname	Aufgabe
Activity	Darstellung der App Oberflächen und Schnittstelle für den Benutzer
Model	Modelliert die Daten, die von der REST API zur Verfügung gestellt werden
REST Client	Kommunikation zwischen REST API und der App
Send Combination	Generiert PDF und leitet dies weiter an Standard E-Mail Programm

Tabelle 2.2: Subkomponentenbeschreibung - App

Die App Komponente modelliert die zu entwickelnde App. Der User interagiert dabei über das User Interface mit der Anwendung. Die Activity Komponente nimmt dabei die Interaktionen entgegen und führt dementsprechend Aktionen aus. Dabei besteht die Activity Komponente aus der Darstellung, die bei Android über XML stattfindet und den eigentlichen Activities, die die Interaktionen entgegennehmen. Von den Activities aus werden die REST Client, Model und Send Combination Komponenten verwendet.

Der REST Client bietet einerseits eine Schnittstelle für die Authentifikation und für das Einloggen und anderseits eine, um Daten von dem Server zu holen.
Die Model Komponente modelliert die benötigten Daten in der App. Die Klassen im Model entsprechen den Daten von der REST API.
Die Send Combination Komponente ermöglicht das Teilen von Kombinationen. Dabei wird die Kombinationen und ein beschreibender Text zu einem PDF umgewandelt und dann über eine Schnittstelle an einen E-Mail Client weitergegeben.

Kapitel 3

Verteilungsdiagramm

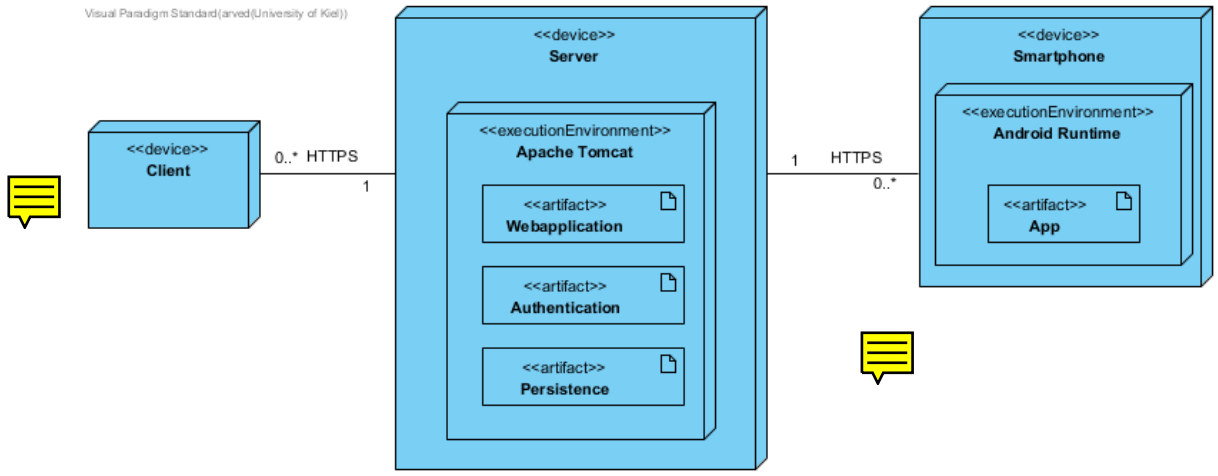

Abbildung 3.1: Verteilungsdiagramm

In dem Verteilungsdiagramm beschreiben wir welche Komponente auf welcher Hardware installiert wird.
Unsere Webapplikation kommuniziert via HTTPS mit den beiden Clients. Diese sind einmal ein Benutzer PC, auf dem die Webseite als .html Datei läuft und ein Smartphone mit der Ausführungsumgebung Android Runtime, auf dem die App als .apk installiert wird. Auf unserem Server läuft die Ausführungsumgebung Apache Tomcat. Hier läuft die Webapplikation als .war, die Authentifikation und die Persistenz des Spring Frameworks.

Kapitel 4

Klassendiagramme

4.1 Server

4.1.1 Model

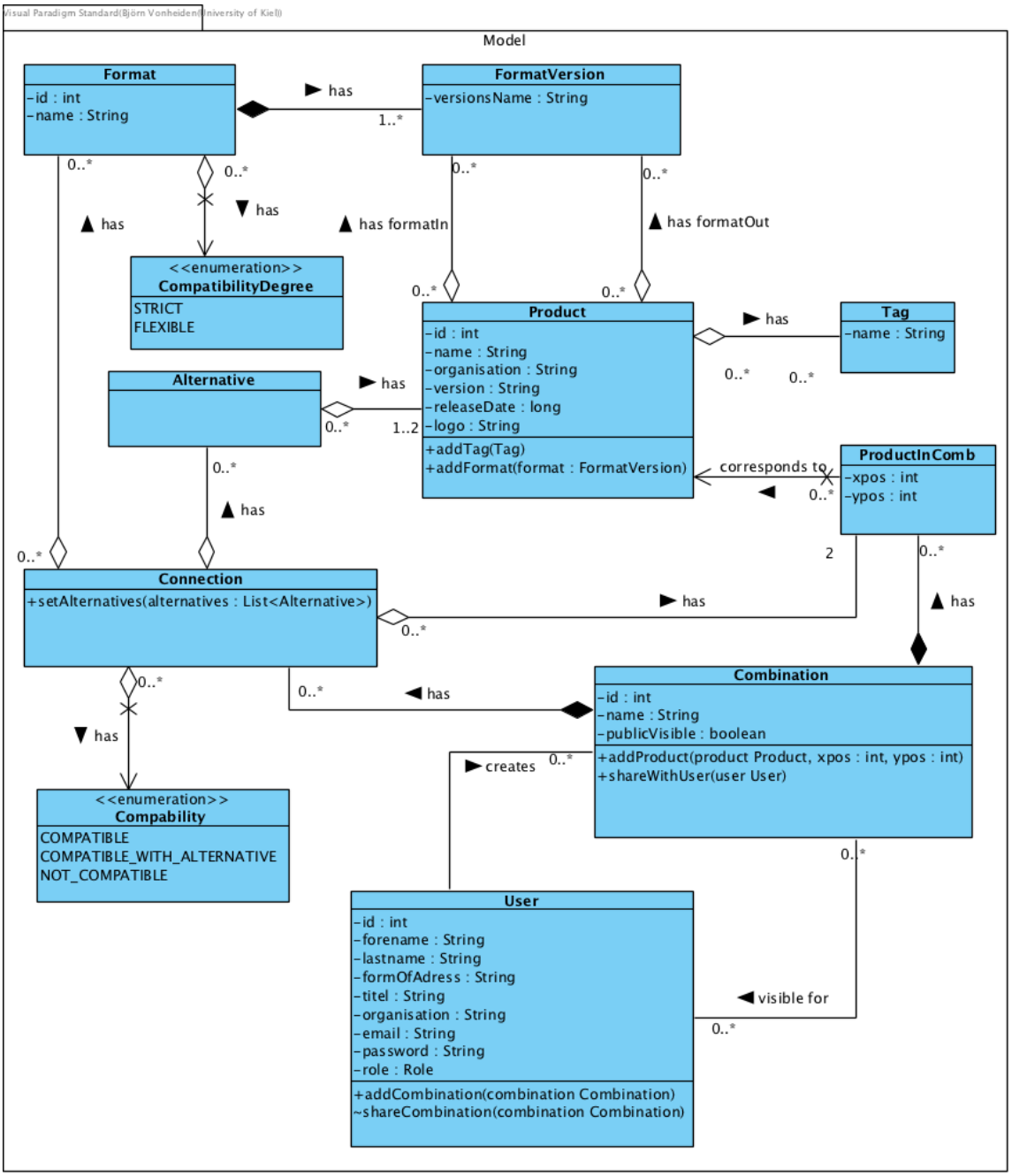

Abbildung 4.1: Klassendiagramm - Server - Model

Klassenname	Aufgabe
Model.Alternative	Eine Alternative zu einer Verbindung besteht entweder aus einem oder aus zwei Produkten, welche die Verbindung gültig machen.
Model.Combination	Informationen von einer Kombination und Personen, für die die Kombination freigegeben wurde.
Model.Compability	Ein Enum, welcher darstellt, ob eine Verbindung kompatibel ist, nicht kompatibel ist oder über eine alternative Kompatibel ist.
Model.CompatibilityDegree	Ein Enum, welcher entweder 'STRICT' oder 'FLEXIBLE' ist. Flexibel heißt, dass das Format abwärts kompatibel ist, strict heißt, dass nur das genau gleiche Format kompatibel ist.
Model.Connection	Informationen von einer Verbindung. Eine Verbindung hat zwei zugehörige Produkte und eine Richtung. Die Richtung wird durch die Reihenfolge der beiden Produkte definiert.
Model.Format	Ein Format hat eine oder mehrere Versionen und kann entweder strict oder flexible sein.
Model.FormatVersion	Informationen einer Format-Version. Jede Version ist einem Format zugeordnet und hat eine Liste der zugehörigen Produkte, da dies die Suche nach Alternativen effizienter macht.
Model.Product	Informationen von einem Dienst. Das Logo wird dabei durch einen Dateipfad definiert.
Model.ProductInComb	Speichert den Ort eines Produktes in einer bestimmten Kombination. In einer Kombination kann der selbe Dienst mehrmals vorkommen, dies sind dann aber unterschiedliche ProductInCombs.
Model.Tag	Ein Tag kann Produkten zugeordnet werden. Mithilfe dieser Tags kann nach Produkten gesucht werden.
Model.User	Speichert die Daten eines Benutzers, dessen Combinationen und seine Rolle

Tabelle 4.1: Klassenbeschreibung - Model

4.1.2 Logik

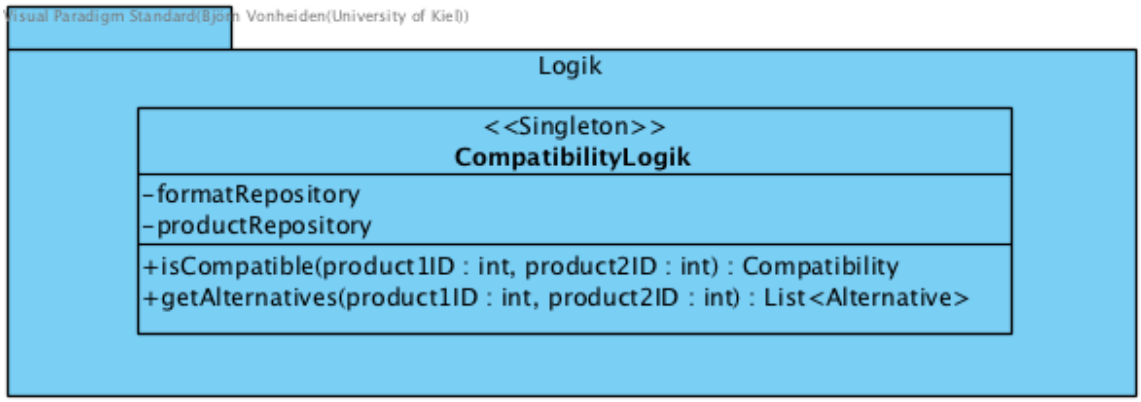

Abbildung 4.2: Klassendiagramm - Logik

Klassenname	Aufgabe
Logik.CompatibilityLogik	Diese Klasse bietet Methoden an, um die Kompatibilität zweier Dienste zu prüfen und eine Liste an Alternativen zu bekommen. Die Repository Variablen werden durch die Autowired-Annotation von Spring initialisiert.

Tabelle 4.2: Klassenbeschreibung - Logik

Im 'Logik' Package sollen alle komplexeren Berechnungen die über die einfache Logik eines Controller hinausgeht. Als Beispiel für eine solche Berechnung haben wir hier die Kompatibilitätsberechung dargestellt. Auch die Berechnung der Alternativen für eine inkompatible Verbindung wäre ein solches Beispiel. Wahrscheinlich werden in diesem Package später auch noch weitere Aufgaben erledigt, wie beispielsweise die Erstellung von JSON-Dateien beziehungsweise von Objekten, welche sich von den Objekten im Model unterscheiden und besser zum Verschicken geeignet sind. Zudem können noch Methoden dazukommen, welche Informationen aus JSON-Dateien auslesen und an das Modell weiterreichen.

4.1.3 Controller

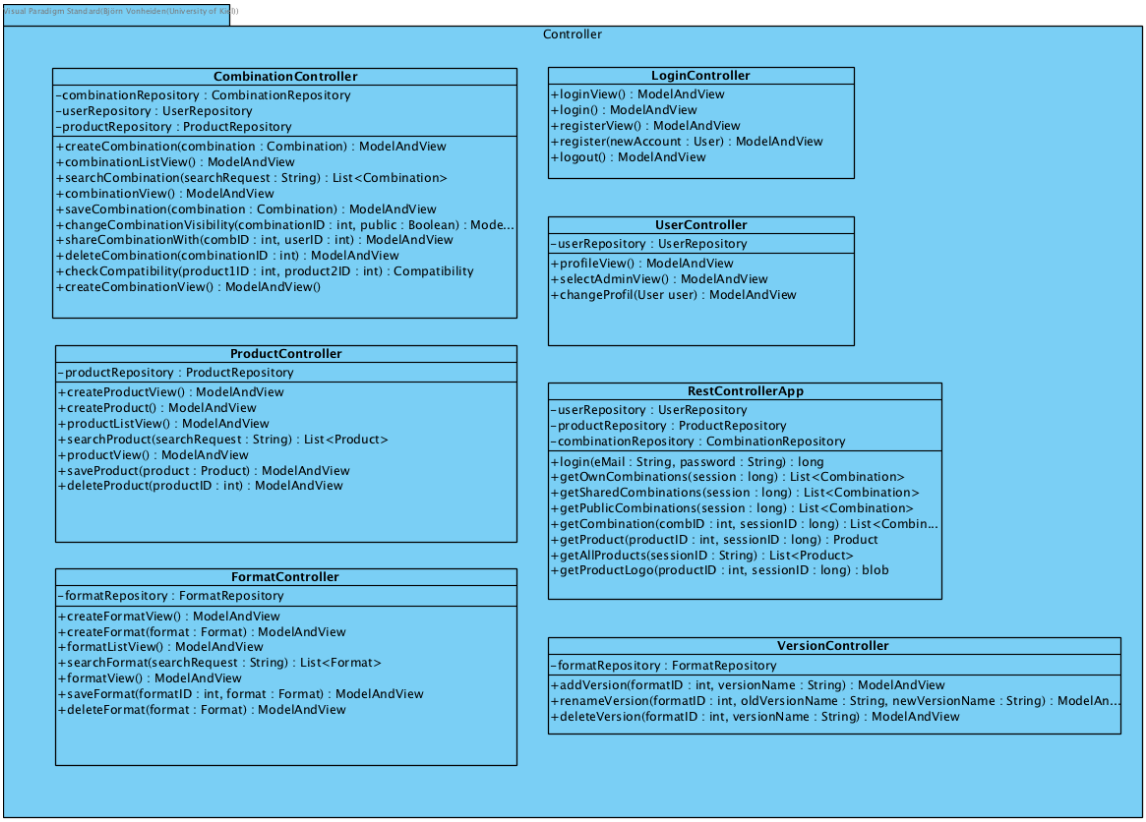

Abbildung 4.3: Klassendiagramm - Controller

Klassenname	Aufgabe
Controller.CombinationController	Hier sind alle Controller zur Webapplikation enthalten, welche etwas mit Kombinationen zu tun haben. Die Methoden, welche ModelAndView returnen, haben die Annotation '@Controller' und die anderen '@RestController'. Die Repositories sind autowired.
Controller.FormatController	Diese Klasse enthält alle Controller zur Webapplikation, welche etwas mit Formaten zu tun haben.
Controller.LoginController	Diese Klasse enthält alle Controller zur Webapplikation, welche etwas mit dem Login zu tun haben.
Controller.ProductController	Diese Klasse enthält analog zur oberen Klasse alle Controller zur Webapplikation, welche etwas mit Diensten zu tun haben.
Controller.RestControllerApp	Diese Klasse enthält alle Controller, welche mit der App kommunizieren. Die App schickt dabei GET Anfragen und die URLs fangen mit /App/ an. Die Controller sind mit '@RestController' annotiert, es werden also nicht wie bei der Webseite ModelAndViews returnt, sondern in JSON verpackte Objekte. Die Umwandlung wird von Spring übernommen.
Controller.UserController	Diese Klasse enthält alle Controller zur Webapplikation, welche etwas mit Benutzern zu tun haben.
Controller.VersionController	Diese Klasse enthält alle Controller zur Webapplikation, welche etwas mit Versionen von Formaten zu tun haben.

Tabelle 4.3: Klassenbeschreibung - Controller

4.1.4 Repository

Zusätzlich zu dem 'Model' Package haben wir ein 'Repository' Package, welches zu den Klassen des Models (z.B.: Product) jeweils ein Interface (z.B.: ProductRepository), welches CrudRepository extended. Die Funktion dieser Interfaces ist die Festlegung der Zugriffsarten auf das Model. Da ein Klassendiagramm dazu aber sehr redundant zu dem Model ist und die konkreten Methoden von der Implementierung der Controller abhängig ist, haben wir das hier weggelassen.

4.1.5 Security

In dem 'Security'-Package sollen die Funktionen der Authentification & Authorization umgesetzt werden. Da viele Funktionen und Klassen schon durch das Spring-Security-Framework gestellt werden und nur konfiguriert bzw. aufgerufen werden müssen und wir uns der Umsetzung noch nicht komplett klar sind, haben wir ein Diagramm an dieser Stelle weggelassen. So werden wir voraussichtlich beispielsweise für die Http-Requests vorkonfigurieren welche Rollen (siehe Model - User), welche Requests stellen darf. So unterscheiden wir die Rollen, wie im Pflichtenheft, in unangemeldete Nutzer, eingeloggte Nutzer und Administratoren. So werden die Seiten: Kombinationen(nur mit

den freigegebenen Kombinationen), Dienste, Login und Registrieren, für alle Nutzer freigeben. Die Seiten: Kombinationen(mit den zusätzlichen Optionen eigene und für einen Freigegebene), Profil und Logout sind für alle angemeldeten Nutzer freigegeben. Diese erhalten dabei auch Zugriff auf weitere Funktionen wie das Speichern einer Kombination. Für Administratoren werden zusätzlich noch die Seite Benutzer freigegeben. Außerdem wird ihm noch der Zugriff auf Administrator-Funktionen wie das Hinzufügen oder Löschen eines Dienstes. An den meisten Methoden werden wir den Sicherheitsaspekt durch die Security-Annotations des Spring Frameworks umsetzen.

4.2 App

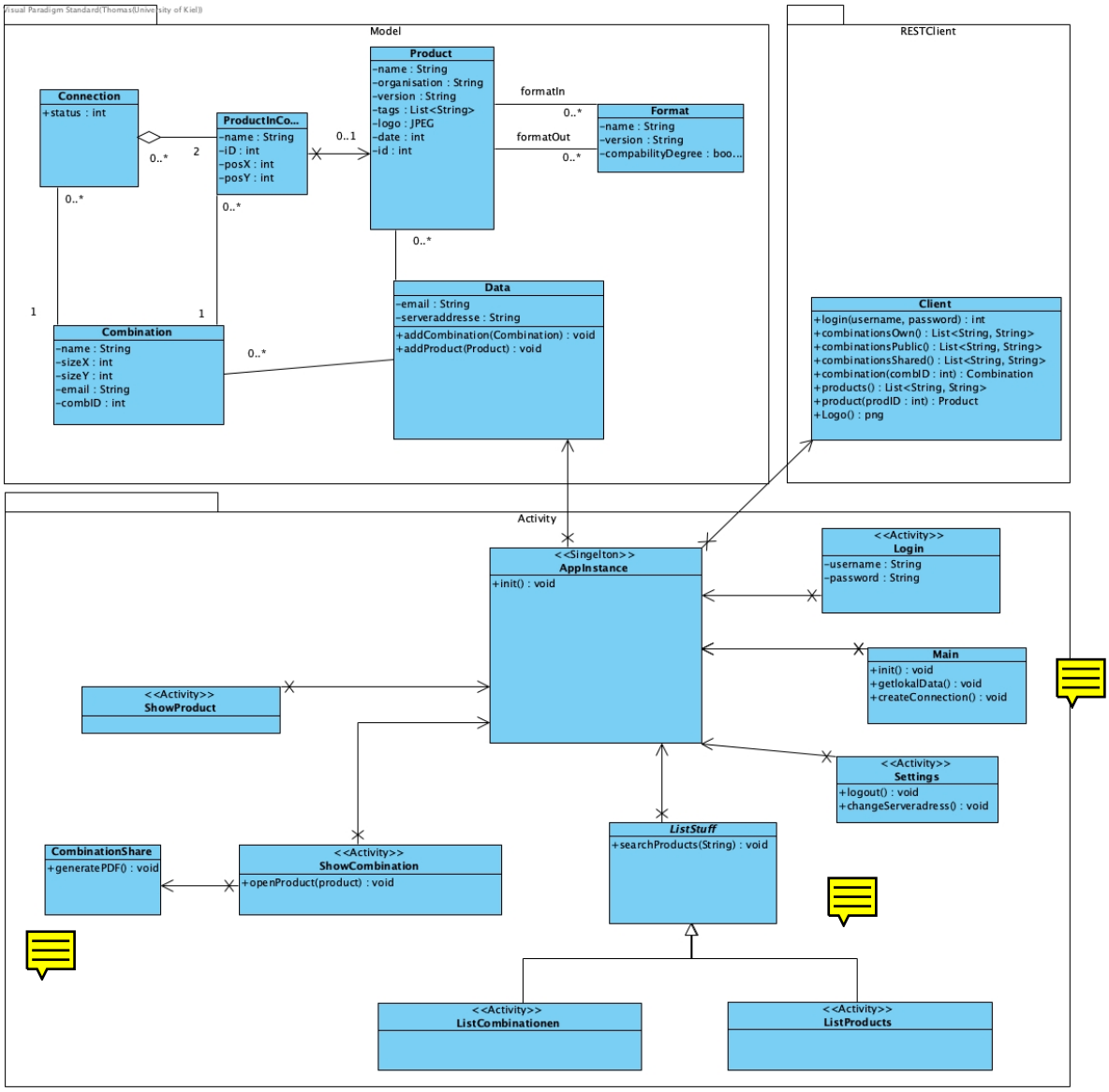

Abbildung 4.4: Klassendiagramm - App

Klassenname	Aufgabe
AppInstance	Singelton erlaubt Zugriff auf Data und Client
Client	Übernimmt alle Kommunikationen mit dem Server
CombinationShare	Leitet PDF an das Android System weiter
DataManagement.Combination	Informationen von einer Kombination
DataManagement.Connection	Informationen von einer Verbindung
DataManagement.Data	Speichert lokal relevante Daten
DataManagement.Format	Informationen eines Formats
DataManagement.Product	Informationen von einem Dienst
DataManagement.ProductInComb	Speichert ausschließlich die Informationen eines Produkts die zum Visualisieren der Kombination relevant sind.
ListCombination	Darstellung aller möglichen Kombinationen und diese suchbar machen
ListProducts	Darstellung aller möglichen Dienste und diese suchbar machen
ListStuff	Abstrakte Klasse für die Listen(Dienste und Kombinationen)
Login	Stellt den Login bereit
Main	Initialisiert die Applikation
Settings	Stellt Einstellungen für Benutzer bereit
ShowCombination	Generiert Canvas einer Kombination und visualisiert diese
ShowProduct	Zeigt einen einzelnen Dienst an

Tabelle 4.4: Klassenbeschreibung - App

Kapitel 5

Sequenzdiagramme

5.1 Webseite, Server

Bei den folgenden Sequenzdiagrammen stellt die RestAPI Klasse das Spring Framework dar, damit wir sowohl die HTML Requests, als auch die Controller Methoden zeigen können.

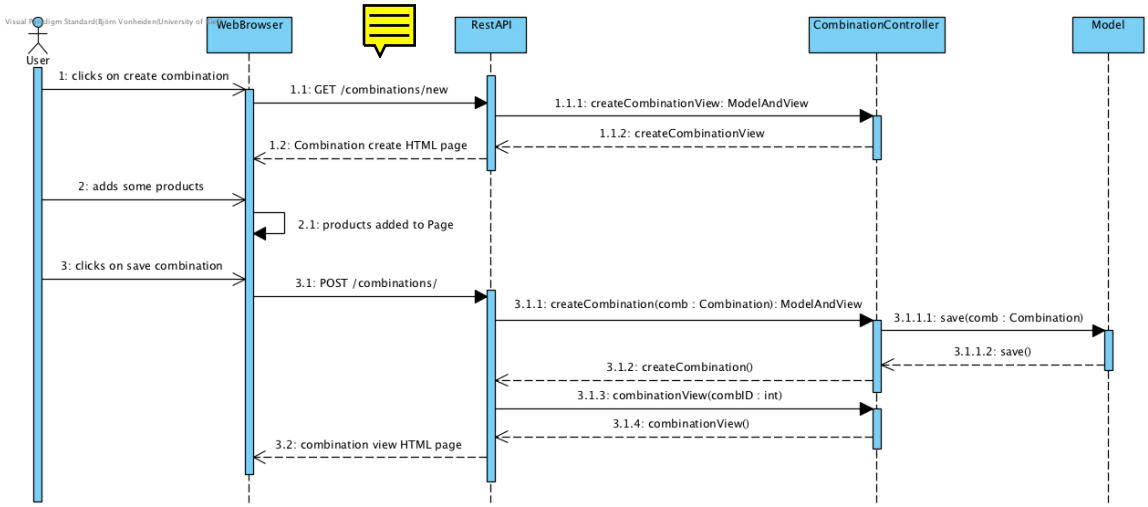

Abbildung 5.1: Sequenzdiagramm - Kombination erstellen Webseite

In der Abbildung 5.1 wird dargestellt, wie man auf der Webseite eine Kombination erstellen kann. Dabei wird in diesem Fall nur berücksichtigt, dass der Benutzer ein paar Dienste zu der Kombination hinzufügt, aber keine Kompatibilitätsprüfung ausführt. Eine Kompatibilitätsprüfung wird in dem Sequenzdiagramm 5.3 ausgeführt.
Der Benutzer klickt, um eine neue Kombination zu erstellen, auf einen Button. Dafür wird dann eine neue Webseite von dem Server angefragt. Im Browser kann der Benutzer dann einzelne Dienste der Kombination hinzufügen. Wenn er dann fertig ist, klickt er auf Kombination speichern. Dadurch schickt der Browser einen POST Request mit den entsprechenden Daten an die API. Auf dem Server wird daraufhin die Kombination gespeichert. Der CombinationController führt dann ein redirect aus, um auf die Webseite zu gelangen, wo die Kombination dann angezeigt wird.

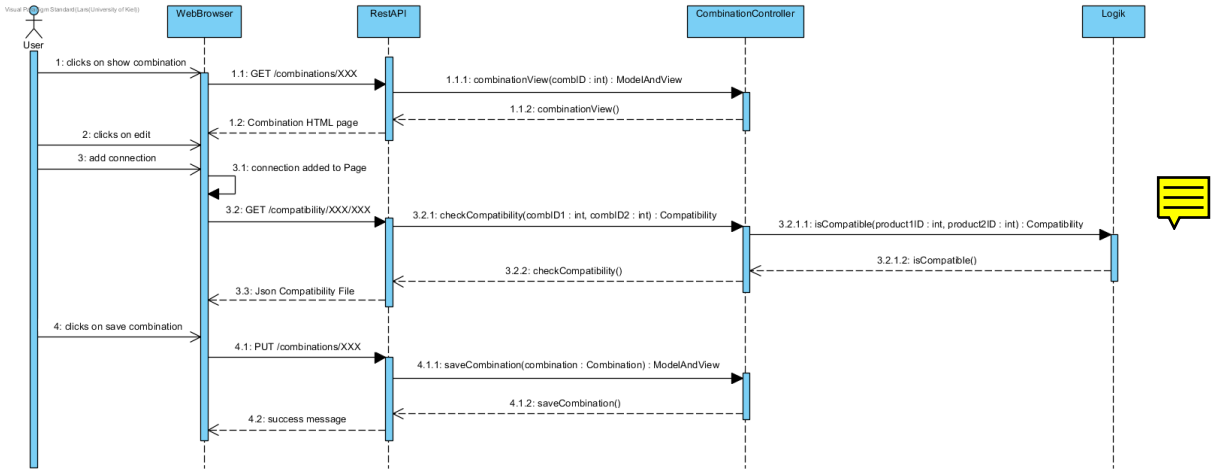

Abbildung 5.2: Sequenzdiagramm - Kombination bearbeiten Webseite

In der Abbildung 5.2 lässt sich der Benutzer eine seiner Kombinationen anzeigen und bearbeitet dann diese Kombination. Die Kombination soll bereits mindestens zwei nicht verbundene Dienste enthalten. Dann fügt er dort eine Verbindung zwischen zwei Diensten ein.
Als erstes öffnet der Benutzer eine seiner Kombinationen. Dafür wird eine Anfrage an den Server geschickt und dieser gibt eine HTML-Seite zurück. Der Benutzer, klickt dann auf Kombination editieren. Im Browser hat er dann die Möglichkeit seine Kombination zu bearbeiten. Dort fügt er dann eine Verbindung zwischen zwei Kombinationen ein. Der Browser schickt dadurch dann eine Anfrage an den Server. Der Controller, der die Anfrage entgegennimmt, leitet die Überprüfung an die Logik weiter. Wenn der Controller dann eine Antwort bekommt, wird diese als JSON zurück an den Browser gegeben.

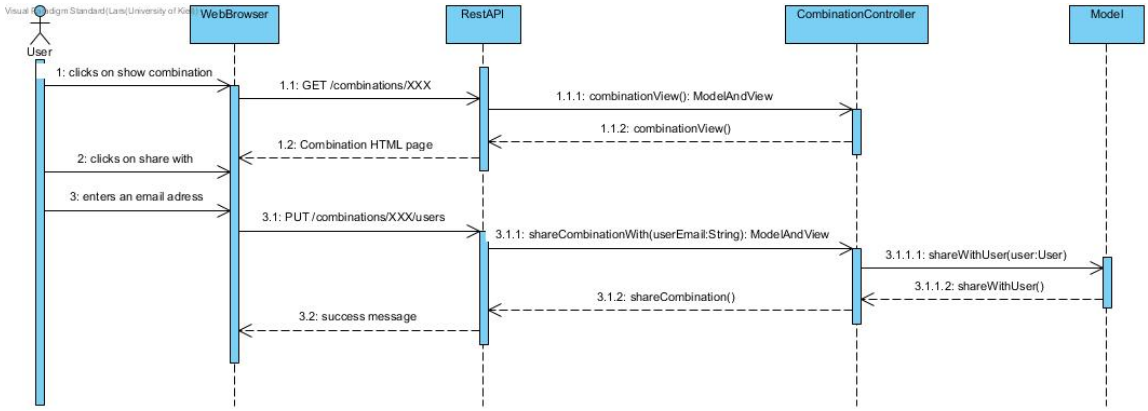

Abbildung 5.3: Sequenzdiagramm - Kombination freigeben Webseite

In der Abbildung 5.3 wird das Freigeben einer eigenen Kombination an einen anderen Nutzer dargestellt. Anschließend kann der andere Nutzer diese Kombination ebenfalls ansehen, allerdings nicht bearbeiten. Zunächst wird wie im Diagramm 5.2 eine Kombination angezeigt. Im Browser kann der Nutzer nun auf freigeben klicken und eine E-Mail Adresse eingeben. Diese wird über einen HTTP PUT Request an den Server geschickt. Dort macht der entsprechende Controller eine

Anfrage an das Model, wodurch der Nutzer zu den Personen mit Zugriffsrechten auf die Kombination hinzugefügt wird. Falls der andere Nutzer nun auf der Webseite eine GET /combinations oder auf der App eine GET /app/combinations/shared Anfrage macht, wird diese Kombination zusätzlich angezeigt.

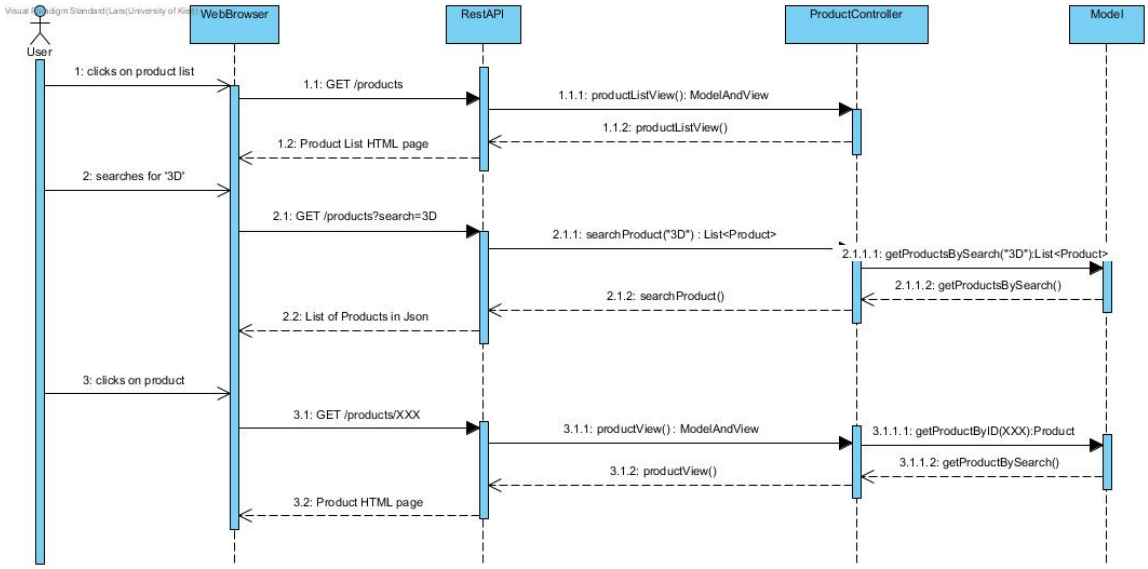

Abbildung 5.4: Sequenzdiagramm - Dienst Suchen Webseite

In der Abbildung 5.4 wird die Suche und das Anzeigen eines Dienstes gezeigt. Zunächst werden alle Produkte geladen, für eine Übersicht. Bei der anschließenden Suche wird der Suchbegriff in der URL weitergegeben. Der Controller macht dann eine Anfrage, welche im Repository definiert ist. Dabei wird sowohl nach einem Tag, als auch dem Dienstnamen gesucht und die jeweiligen Dienste werden aus dem Model geholt. Da die Methoden aber über Spring laufen, haben wir es im Diagramm direkt an das Model geschickt. Anschließend erhält der Nutzer die Dienste und kann über eine weitere Anfrage einen Dienst öffnen.

5.2 App

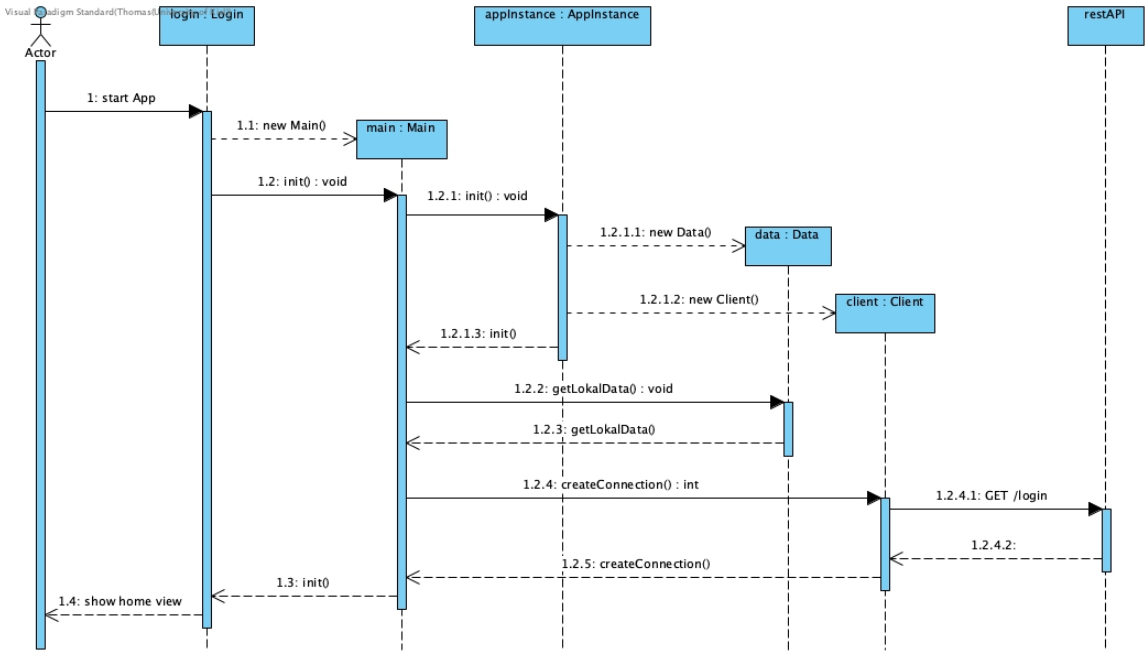

Abbildung 5.5: Sequenzdiagramm - App Start-Up

In der Abbildung 5.5 wird der Ablauf des Start-up der App abgebildet. Es wird dargestellt wie die wichtigen Klassen initialisiert werden und wie die Verbindung zum Server aufgebaut bevor der Login Bildschirm angezeigt wird.

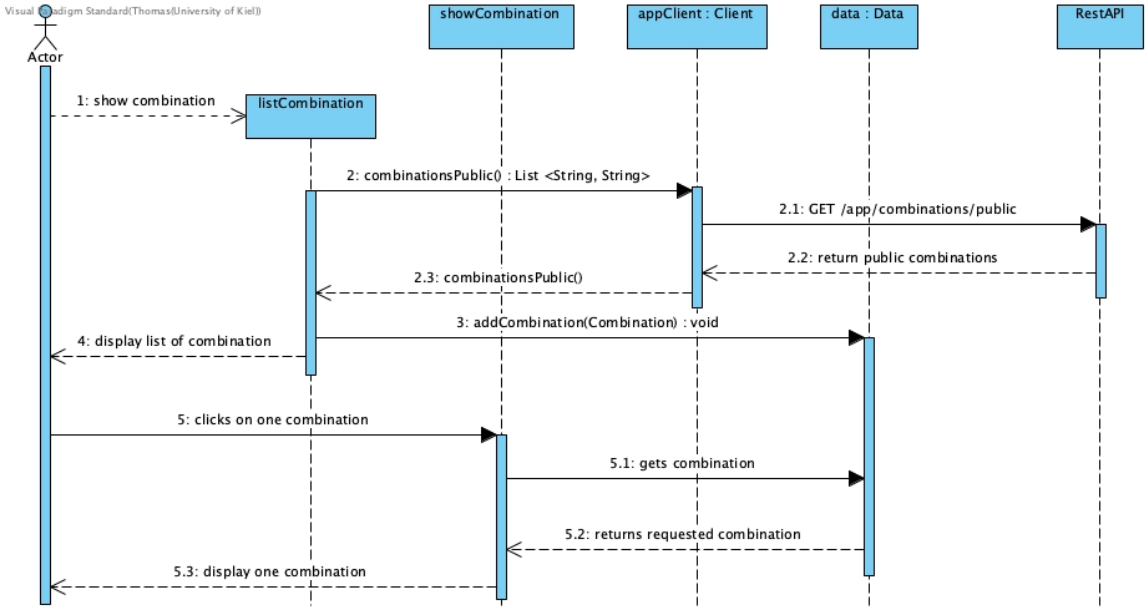

Abbildung 5.6: Sequenzdiagramm - Kombinationen anzeigen ohne Account

In der Abbildung 5.6 wird das Anzeigen von öffentlichen Kombinationen modelliert. Benutzer ohne eigenen Account haben nur die Option öffentlich geteilte Kombinationen anzusehen. Der Client der App holt sich mit der GET combinations/public Anfrage alle öffentlich verfügbaren Kombinationen. Diese werden zurück an die App übertragen, damit sie dem Nutzer zugänglich gemacht werden können.

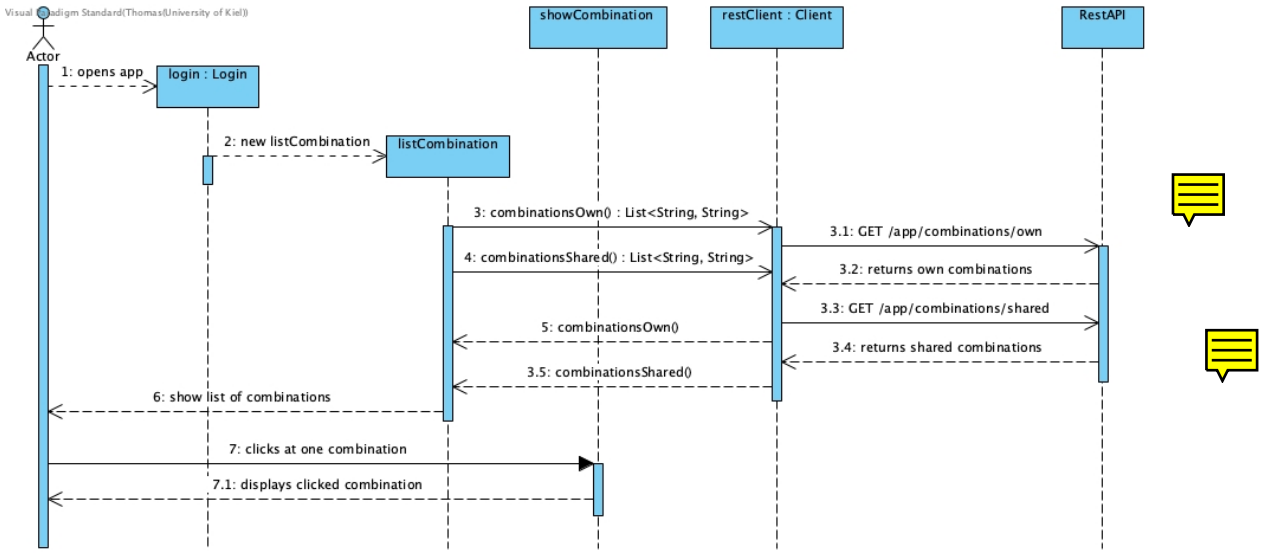

Abbildung 5.7: Sequenzdiagramm - Kombinationen anzeigen mit Account

In Abbildung 5.7 wird der Ablauf dargestellt, wenn ein eingeloggter User seine eigenen Kombinationen ansehen will. Zusätzlich wird die Liste auch mit den Kombinationen erweitert, die für den User freigegeben worden sind. Hier stellt der Client neben der GET combinations/own Anfrage auch eine combinations/shared Anfrage, da Nutzer mit eigenen Account auch die Kombinationen der eigenen Organisation oder von anderen Nutzern freigegebene Kombinationen ansehen kann. Die übertragenen Kombinationen tauchen auch wieder in einer Liste von möglichen Kombinationen auf.

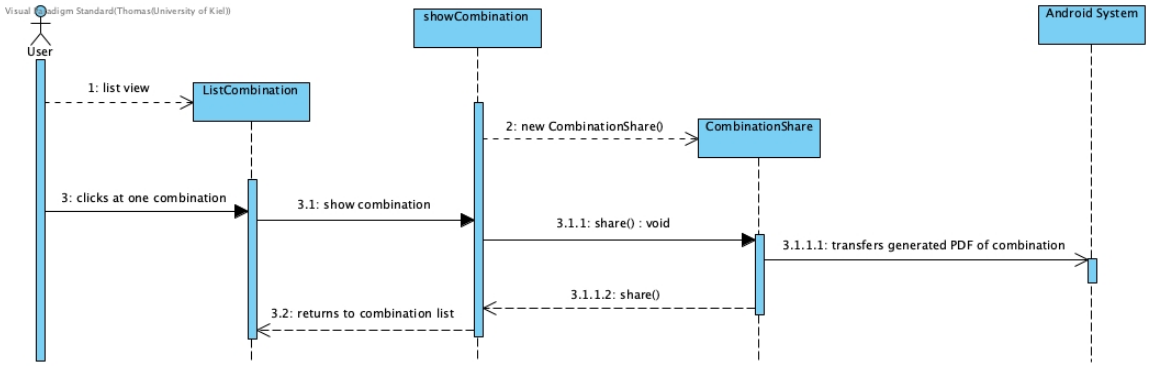

Abbildung 5.8: Sequenzdiagramm - Kombination teilen

In der Abbildung 5.8 beschreiben wir die Möglichkeit eine vorhandene Kombination mit dem Standard Mail-Programm auf dem Handy zu versenden. Der Nutzer öffnet die Kombination, die er per Mail versenden möchte. Dort besteht die Möglichkeit die Kombination zu teilen, wodurch sich die Standard Weiterleitung von Android öffnet. Dort werden die Daten an das Mail Programm weitergeleitet. Nach dem Versenden werden dem Nutzer wieder die Liste aller möglichen Kombinationen angezeigt.

Kapitel 6

Rest Api

6.1 Webseite

Methode	URL	Server Klasse	Server Methode
		Seiten	
GET	/register	LoginController	registerView
GET	/login	LoginController	loginView
GET	/combinations	CombinationController	combinationListView
GET	/combinations/XXX	CombinationController	combinationView
GET	/products	ProductController	productListView
GET	/products/XXX	ProductController	productView
GET	/formats/	FormatController	formatListView
GET	/formats/XXX	FormatController	formatView
GET	/users/XXX	UserController	profileView
GET	/admin	UserController	selectAdminView
GET	/combinations/new	CombinationController	createCombinationView
GET	/formats/new	FormatController	createFormatView
GET	/products/new	ProductController	createProductView
		Suchen	
GET	/products?search=XXX	ProductController	searchProduct
GET	/combinations?search=XXX	CombinationController	searchCombination
GET	/formats?search=XXX	FormatController	searchFormat
		Erstellen	
POST	/users	UserController	register
POST	/products	ProductController	createProduct
POST	/formats	FormatController	createFormat
POST	/combinations	CombinationController	createCombination
POST	/formats/XXX/version	VersionController	addVersion
		Update	
PUT	/users/XXX	UserController	changeUser
PUT	/combinations/XXX	CombinationsController	saveCombination
PUT	/combinations/XXX/visibility	CombinationsController	changeCombinationsVisibility
PUT	/combinations/XXX/users	CombinationsController	shareCombinationWith
PUT	/products/XXX	ProductController	saveProduct
PUT	/formats/XXX	FormatController	saveFormat
PUT	/users/XXX/version/XXX	VersionController	renameVersion
		Löschen	
DELETE	/combinations/XXX	CombinationsController	deleteCombination
DELETE	/products/XXX	ProductController	deleteProduct
DELETE	/formats/XXX	FormatController	deleteFormat
DELETE	/formats/XXX/version/XXX	VersionController	deleteVersion
		Sonstiges	
GET	/compatibility/XXX/XXX	CombinationsController	checkCompatibility
GET	/login/XXX	LoginController	login
GET	/logout/XXX	LoginController	logout

Tabelle 6.1: Rest - Webseite

6.2 App

Methode	URL	Server Klasse	Server Methode
GET	/app/combinations/own	RestControllerApp	getOwnCombinations
GET	/app/combinations/shared	RestControllerApp	getSharedCombinations
GET	/app/combinations/public	RestControllerApp	getPublicCombinations
GET	/app/combinations/XXX	RestControllerApp	getCombination
GET	/app/products	RestControllerApp	getAllProducts
GET	/app/products/XXX	RestControllerApp	getProduct
GET	/app/products/XXX/Logo	RestControllerApp	getProductLogo

Tabelle 6.2: Rest - App

6.3 Erklärung

Bei der Webseite wird grundsätzlich ein ModelAndView zurückgegeben, die View ist dabei die neue HTML Seite und das Model enthält die benötigten Daten. Es gibt aber auch Ausnahmen: Beispielsweise bei der Suche nach Diensten wird nur eine Liste von Diensten (über JSON) zuückgegeben, da sich die View nicht ändert und man sowohl in der Dienst-Liste, als auch bei der Komponentenbearbeitung nach Diensten suchen kann. Die Methoden, die ModelAndView zurückgeben, haben '@Controller' als Annotation und die anderen '@RestController'. Die jeweiligen Rückgabearten der Methoden kann man im Klassendiagramm finden. Bei den Requests von der App ist die URL immer mit '/app/' markiert, die jeweiligen Controller sind mit '@RestController' annotiert. Die HTTP Methode ist dort immer GET und es werden JSON Objekte im Response Body zuückgegeben. Allgemein sind die GET Methode für Datenabfrage und neue Views, POST für das erstellen von Objekten, PUT für das ändern von Daten und DELETE zum Löschen von Daten. Da User nicht gelöscht werden können, haben wir dafür keine Methode. Der Unterschied zwischen loginView (bzw. registerView) und login (register) ist, dass mit der View Methode nur die Seite zum anmelden oder registrieren angezeigt wird und mit der anderen die Aktion durchgeführt wird.

Kapitel 7

Glossar

Abkürzung	Beschreibung
Alternative	Einzelner Dienst oder eine Kette von Diensten, die in eine inkompatible Verbindung zwischengeschaltet werden können und dadurch eine Kompatibilität herstellen
API	Application Programming Interface
Client	Internetbrowser oder Android-App, je nach Benutzung
DB	Datenbank
Dienst	Ein Dienst besteht aus Name, Hersteller, Version, Datum, Tags und einem In und Out Format. Dienste können über die Web Anwendung erstellt werden oder über eine REST- Schnittstelle hinzugefügt werden
HTML	Hypertext Markup Language
HTTP-Request	Anfrage vom Client an den Server
HTTPS	Hypertext Transfer Protocol Secure
Kette	Mehrere Dienste die in Verbindung stehen
Kombinationen	Mit einer Kombination ist die Verknüpfung verschiedener Dienste gemeint.
Kompatibilität	Eigenschaft die Verbindungen zwischen Diensten aufweisen, wenn ein Ausgabeformat des ausgehenden Dienstes mit einem Eingabeformat des eingehenden Dienstes aufweist
PDF	Portable Document Format
Product	Dienst
Redirect	Weiterleiten von einer URL auf eine andere URL
REST	Representational State Transfer
User	Benutzer
XML	Extensible Markup Language

Tabelle 7.1: Glossar