Entwurfsdokumentation
SWARM Composer
-
Softwareprojekt SoSe18
Gruppe HRS3 105b

Jeremia Böhmig
Felix Gröner
Janek Haberer
Robert Köhler
Johanna Menzel
Jette Petzold
Christian Richter
Connor Schönberner

9. September 2018

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Dokumentaufbau

1.1.1 Ziel des Dokuments

Ziel des Dokuments ist die Dokumentation der Entwicklung des SWARM Composers. Die hier aufgeführten Diagramme und Erklärungen sollen neuen Teammitgliedern den Einstieg in das Projekt und zukünftigen Entwicklern die Einarbeitung in die Software erleichtern. Die frühzeitige Planung der Struktur der Software führt zu einem reibungsloseren Entwicklungsprozess. Außerdem finden sich hier konkrete Absprachen der kleineren Teilprojekte, um die Entwicklung entkoppelter Komponenten zu ermöglichen.

1.1.2 Aufbau

In den Kapiteln 2 und 3 wird je ein plattformübergreifendes Komponenten- bzw. Verteilungsdiagramm dargestellt. Darauf folgt in Kapitel 4 eine Sammlung von Klassendiagrammen, die die Code-Struktur auf den jeweiligen Plattformen erklären. In Kapitel 5 werden die wichtigsten Methodenaufrufe und Abläufe, die in den Klassendiagrammen zu finden sind, als Sequenzdiagramme modelliert.

1.2 Zweckbestimmung

Der SWARM Composer dient dazu, unterschiedliche Software für Bauprojekte in Bezug auf ihre Kompatibilität bezüglich der Ein- und Ausgabeformate zu überprüfen. Die Software ist Teil des Forschungsprojektes SWARM des Unternehmens adesso. Der SWARM Composer besteht aus zwei Teilen: einem Webserver und einer App. Auf dem Webserver können Benutzer und Benutzerinnen Dienste zu Kompositionen zusammenfassen und auf Kompatibilität überprüfen. In der App können Kompositionen grafisch präsentiert und per PDF verschickt werden.

1.3 Entwicklungsumgebung

In der untenstehenden Tabelle finden sich grundlegende Frameworks, Bibliotheken, Tools und Sprachen, die für die Entwicklung im Rahmen dieses Projektes erforderlich sind.

Software	Version	URL
Maven	4.0.0	maven.apache.org/POM/4.0.0
Spring	2.0.4	spring.io
Android Studio	3.1.4	https://developer.android.com/studio/
Tomcat	8.0.53	http://tomcat.apache.org
git	2.17.1	https://git-scm.com/downloads
node.js	8.11.4	https://nodejs.org/en/download/
Visual Paradigm	15.1	https://www.visual-paradigm.com/download/
Thymeleaf	3.9.9	http://www.thymeleaf.org/download.html
vue.js	2.5.2	https://vuejs.org
Bootstrap	4.1.1	https://getbootstrap.com
Javascript	ES6	-
Java Development Kit	8u144	http://www.oracle.com/technetwork/java/javase/downloads/index.html

Tabelle 1.1: Enwicklungsumgebung

Kapitel 2

Komponentendiagramme

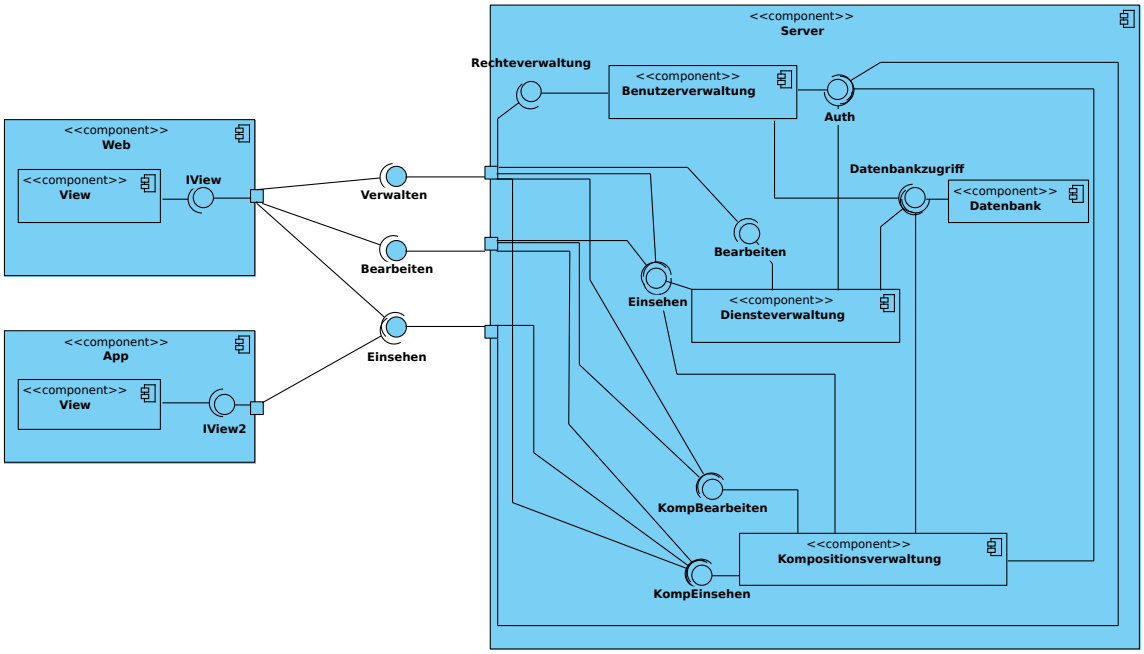

Abbildung 2.1: Komponentendiagramm

Die Serverkomponente bzw. das Backend beschreibt die internen Vorgänge der Software. Diese Komponente ist über drei verschiedene Schnittstellen, welche die unterschiedlichen Benutzerrechte repräsentieren, erreichbar.
In der Komponente “Benutzerverwaltung” werden die Daten der Benutzer aus der “Datenbank” ausgelesen und verarbeitet. Dadurch kann die “Benutzerverwaltung” eine Authentifizierungsschnittstelle für die anderen Subkomponenten bereitstellen. Zudem kann die Rechteverwaltung manipuliert werden, falls über die *Verwalten*-Schnittstelle zugegriffen wird. Die Komponente “Diensteverwaltung” liest die Daten der Dienste aus der “Datenbank” aus und verarbeitet diese. Von außen kann auf die “Diensteverwaltung” durch die Schnittstellen *Bearbeiten* und *Einsehen* zugegriffen werden. Die Schnittstelle *Bearbeiten* ist nur für Administratoren über die Schnittstelle *Verwalten* zugänglich. Es werden bei jedem Zugriff die Rechte des zugreifenden Benutzers durch die Schnittstelle *Auth* der

“Benutzerverwaltung” überprüft.
Die Komponente “Kompositionsverwaltung” stellt die benötigten Funktionen für das Bearbeiten und Einsehen von Kompositionen zur Verfügung und bekommt die Daten dafür von den Komponenten “Datenbank” und “Diensteverwaltung”. Das Bearbeiten von Kompositionen ist nur über die Schnittstellen *Bearbeiten* und *Verwalten* möglich, das Einsehen auch über das *Einsehen*-Interface.
Die Komponente “Datenbank” speichert alle nötigen Daten für die Komponenten “Benutzer-”, “Dienste-” und “Kompositionsverwaltung” und stellt diese bei Bedarf zur Verfügung. Zusätzlich kann man über die *Einsehen*-Schnittstelle auch direkt auf die Authentifizierung zugreifen um eine initiale Registrierung und den Login zu ermöglichen.

Die Komponente “Web” mit der Subkomponente “View Verwaltung” ist für die Darstellung in der Webapplikation zuständig. Da man über die Weboberfläche verwalten, bearbeiten und einsehen können soll, ist die Web-Komponente auch mit all diesen Schnittstellen verbunden.
Die Komponente “App” mit der Subkomponente “View” dient zur Darstellung der angefragten Daten in der App. In der App können nur Kompositionen eingesehen werden, daher ist diese nur mit der *Einsehen*-Schnittstelle verbunden.

Kapitel 3

Verteilungsdiagramm

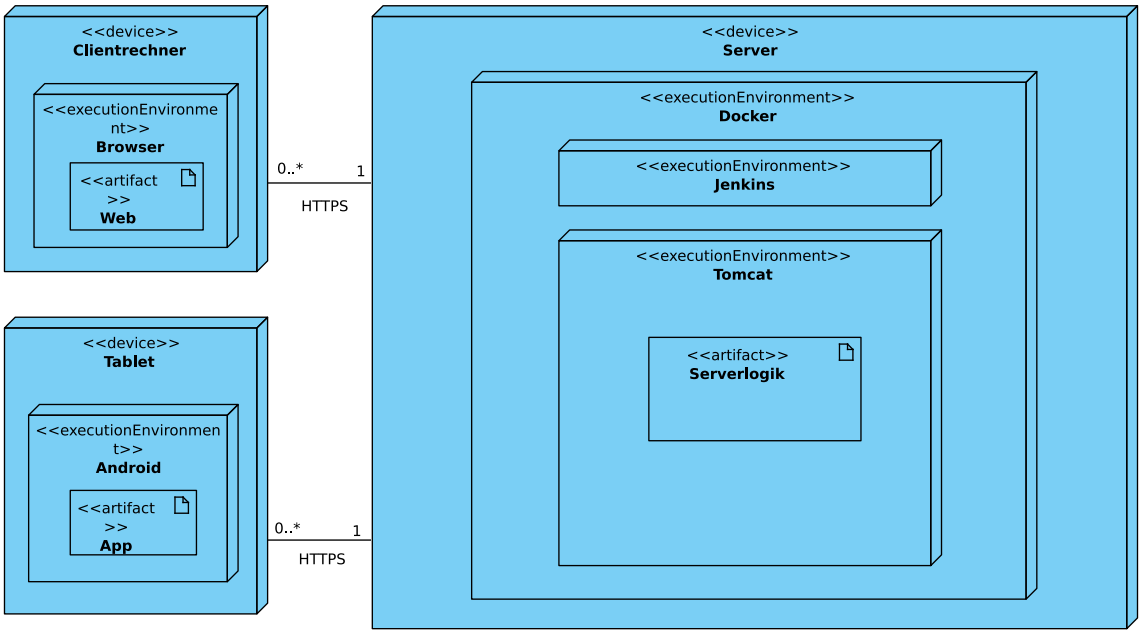

Abbildung 3.1: Verteilungsdiagramm

Die App-Komponente läuft auf einem Tablet Android 6.0 oder höher, während die Web-Komponente in den Browsern auf dem Rechnern des Nutzers läuft. Die Geschäftslogik in der Komponente Serverlogik wird auf einem physikalischen Server ausgeführt, auf dem in einem Docker-Image neben Apache Tomcat auch Jenkins betrieben wird. Sowohl die Clientrechner als auch die Tablets sind mit dem Server über HTTPS verbunden.

3.1 REST Schnitstelle

Aktion	HTTP Methode	Endpunkt
createService(s)	POST	/services
editService	PUT	/services/{$service_{id}$}
getServices	GET	/services
getServiceDetails	GET	/services/{$service_{id}$}
deleteService	DELETE	/services/{$service_{id}$}
checkCompatibility	GET	/services/{$user_{id1}$}/{$user_{id2}$}
getCompositions	GET	/compositions
getCompositionDetail	GET	/compositions/{$comp_{id}$}
createComposition	POST	/compositions
editComposition	PUT	/compositions/{$comp_{id}$}
getUserPermissions	GET	/compositions/{$comp_{id}$}/users
createUserPermission	POST	/compositions/{$comp_{id}$}/users/{email}
editUserPermission	PUT	/compositions/{$comp_{id}$}/users/{email}
deleteUserPermission	DELETE	/compositions/{$comp_{id}$}/users/{$user_{id}$}
getUsers	GET	/users
getUserDetails	GET	/users/{$user_{id}$}
editUser	PUT	/users/{$user_{id}$}
register	POST	/users

Tabelle 3.1: Art der Anfrage und zu kontaktierender Endpunkt der API

Aktion	**Inhalt der Anfrage**	**erwartete Antwort**
createService(s)	List of services	201 - CREATED
editService	single service	200 - OK
getServices	query: string	200 - OK + List of *Service*
getServiceDetails	-	200 - OK + *Service*
deleteService	-	200 - OK
checkCompatibility	-	200 - OK + *CompatibilityAnswer*
getCompositions	-	200 - OK + List of *SimpleComp*
getCompositionDetail	-	200 - OK + *DetailComp*
createComposition	name: string	201 - CREATED
editComposition	*Composition Object*	200 - OK
getUserPermissions	*userAuthorizations*	200 - OK + List of *SimpleUser*
createUserPermission	*userPermission Object*	201 - CREATED
editUserPermission	*userPermission Object*	200 - OK
deleteUserPermission	-	200 - OK
getUsers	query: string	200 - OK + List of *SimpleUser*
getUserDetails	-	200 - OK + *DetailUser*
editUser	*Detail User*	200 - OK
register	*User*	201 - CREATED

Tabelle 3.2: Argumente und erwartete Rückgabe eine API Anfrage

Kapitel 4

Klassendiagramme

Klassendiagramme des Servers

Klassendiagramme des Models

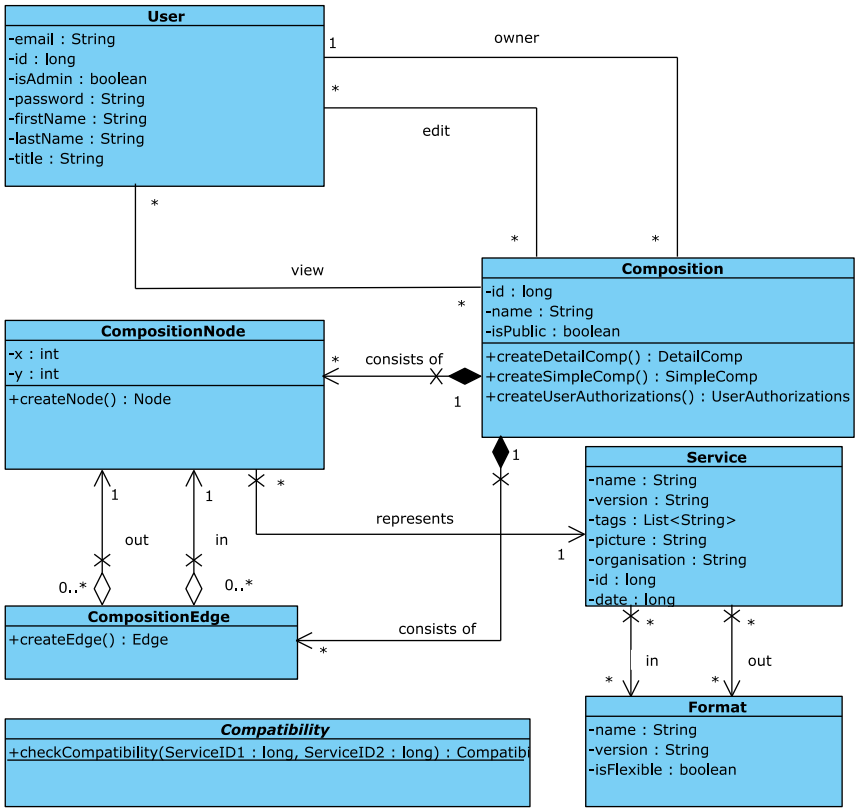

Abbildung 4.1: Klassendiagramm der Informationen, wie sie gespeichert werden

User

- Speicherung der Nutzerdaten sowie einer künstlichen Datenbank-ID
- Eine Flag, die den Adminstatus festlegt
- Listen von erstellten, einsehbaren und veränderbaren Kompositionen

Composition

- Speicherung einer künstlichen ID
- Speicherung der Kompositionen als Graphen durch Speicherung der Knoten und Kanten
- Speicherung des Urhebers und der Nutzenden mit Zugriffs- bzw. Bearbeitungsrechten
- Konvertermethode zum Erstellen eines detaillierten, versendbaren Objektes
- Konvertermethode zum Erstellen eines reduzierten Objektes
- Methode zum Erstellen eines UserAuthorizations-Objektes, um Nutzerrechte später einzusehen

CompositionNode

- Speicherung eines Dienstes, für dessen Verwendung im Kompositionsgraphen
- Konvertermethode zum Erstellen eines detaillierten, versendbaren Objektes
- Referenz auf CompatibilityAnswer, um Kompatibilität der verbundenen Dienste darzustellen

CompositionEdge

- Speicherung einer gerichteten Kompositionskante als Paar von Kompositionsknoten
- Konvertermethode zum Erstellen eines detaillierten, versendbaren Objektes

Service

- Speicherung der Diensteigenschaften
- Speicherung der zum Dienst gelisteten Tags
- Speicherung einer künstlichen ID
- Speicherung je einer Liste der passenden Ein- bzw. Ausgabeformate

Format

- Speicherung von Name, Version und Kompatibilitätsgrad

Compatibility

- Utilklasse
- Methode zum Bestimmen der Kompatibilität zweier Dienste
- Methode soll für Einzelanfragen und Kanten verwendet werden

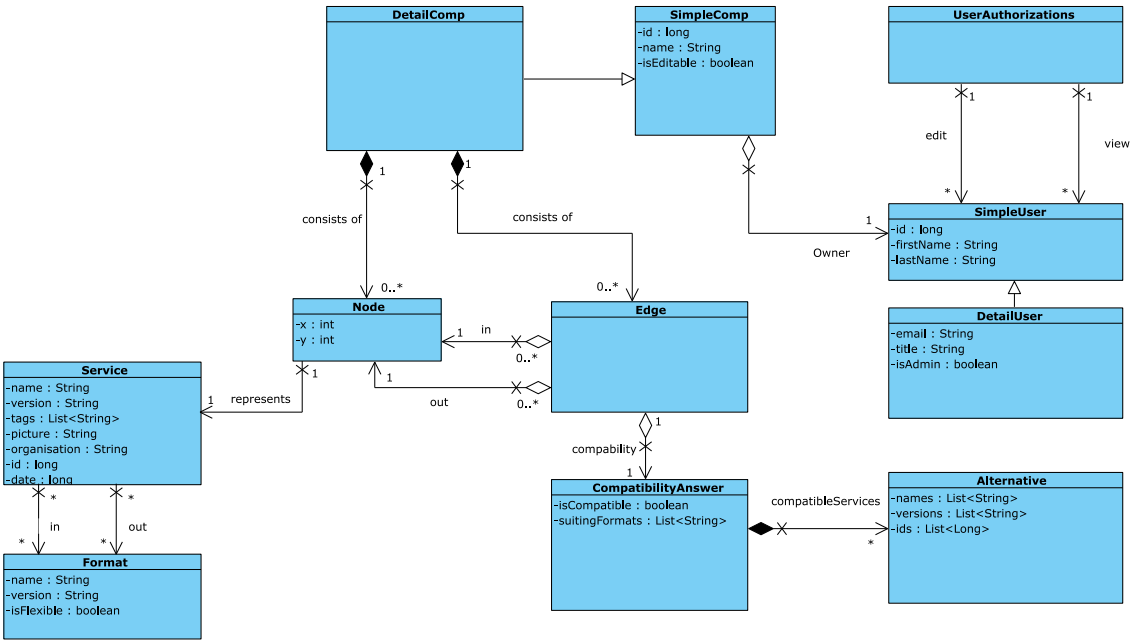

Abbildung 4.2: Klassendiagramm der Informationen, wie sie per JSON verschickt werden

DetailComp

- Erbt von SimpleComp und speichert damit Metadaten zur Komposition (ob bearbeitbar, Name und ID)
- Listen der User (als SimpleUser) mit Bearbeitungs- bzw Einsichtrecht
- Verweis auf Autor

Edge

- Besteht aus einem geordneten Paar von Nodeobjekten, dem Eingang- und Ausgangsknoten.
- Speicherung der Kompatibilität der Dienste anhand einer CompatibilityAnswer

Node

- Speicherung der Position und Verweis auf den verwendeten Service

SimpleComp

- Speicherung der elementarsten Daten für die Listenansicht der Komposition

SimpleUser

- Speicherung der elementarsten Daten (ID und Name)

DetailUser

- Erbt von SimpleUser
- Dient der Bearbeitung der User
- Flag um User als Administratoren zu markieren
- Kein Passwortvermerk

Alternative

- Speicherung der möglichen Konverter eventuell Konverterketten (Name, Versionsnummer, ID), um Kompatibilität zu erzeugen

CompatibillityAnswer

- Speicherung der Kompatibilität zwischen zwei Diensten
- Speicherung der effektiv kompatiblen Formate
- Liste von Alternativen
- Zum Antworten auf Einzelanfragen zur Kompatibilitätsprüfung zweier Dienste

UserAuthorizations

- Speicherung je einer Liste für User mit Einsichts- bzw. Bearbeitungsrecht

Klassendiagramm des Controllers

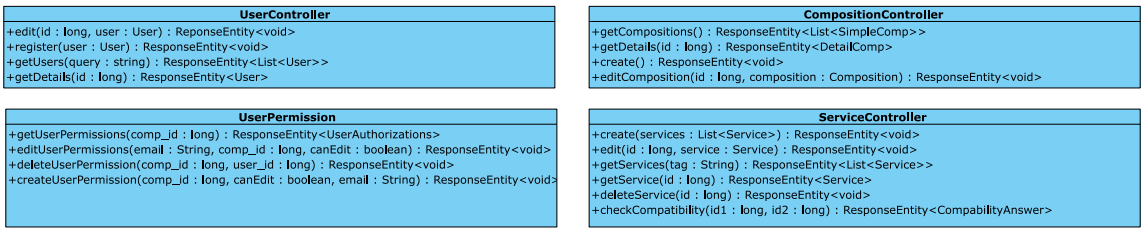

Abbildung 4.3: Klassendiagramm des Controllers

CompositionController

- Klasse stellt einen REST-Controller dar
- Verschiedene Methoden zum Erstellen, Anfragen und Bearbeiten von Compositions aus der Datenbank
- die Authorisierung des Nutzers wird in der Behandlung der Anfragen berücksichtigt
- Rückgabetyp ist stets ResponseEntity, da so ein entsprechender HTTP-Code als Antwort gegeben werden kann.

ServiceController

- Rest-Controller, analog zu CompositionController
- Verwaltet den Zugriff und Bearbeitung von Servicen.

UserController

- REST-Controller, analog zu den anderen Klassen
- Erlaubt es Daten von User zu verändern

UserPermission

- REST-Controller
- Erlaubt es dem Owner, einer Komposition die Zugriffsrechte auf selbige zu verändern.

Klassendiagramm zur Android-App

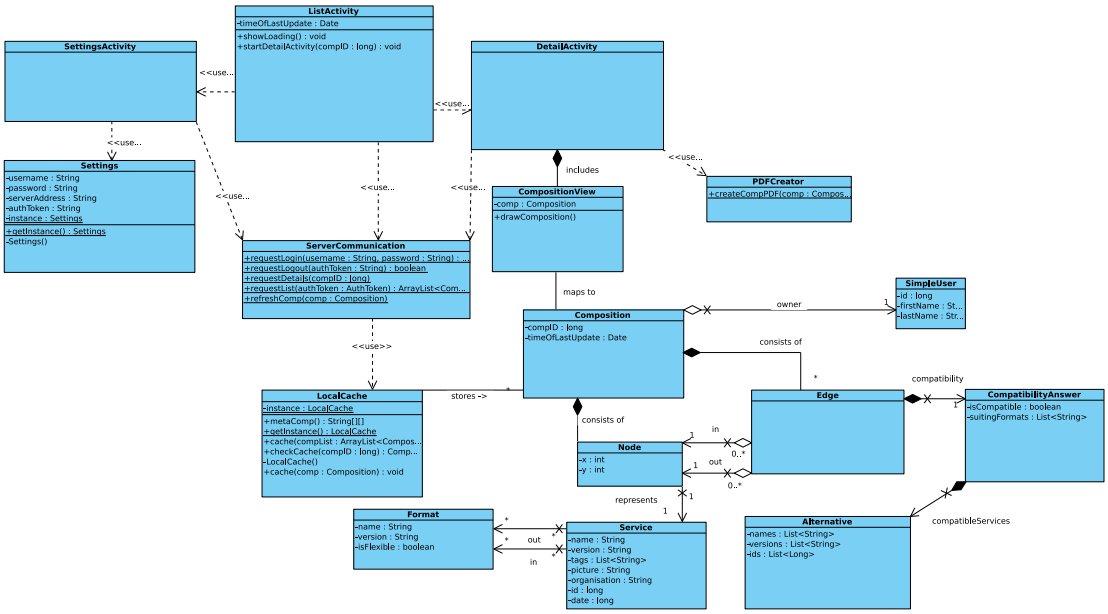

Abbildung 4.4: Klassendiagramm - App

ListActivity

- MainActivity und Übersicht über sichtbare Kompositionen

SettingsActivity

- Activity zum Festlegen von Einstellungen, zum Einloggen und Festlegen der Serveradresse

DetailActivity

- Detailansicht zur grafischen Darstellung einer Komposition

Settings

- Dient zur Kapselung der getroffenen Einstellungen

CompositionView

- Eigentliche View, in der das Zeichnen einer Komposition stattfindet.
- Jede DetailActivity verfügt über ein CompositionView.

Composition

- Model-Abstraktion einer Komposition

Node

- Interne Model-Abstraktion eines Diensts, der als Knoten in einer Komposition fungiert.

Edge

- Interne Model-Abstraktion einer Kante zwischen zwei Diensten in einer Komposition

PDFCreator

- Helper-Klasse zur Generierung von PDFs, die Kompositionsbilder beinhalten.

ServerCommunication

- Anlaufpunkt für sämtliche Kommunikation mit dem Backend.
- ServerCommunication übernimmt die Aufgabe, Verbindungen zum Server herzustellen, die Daten zu interpretieren und im richtigen Format weiterzugeben.

LocalCache

- Cache zur Speicherung von durch Anfragen erhaltenden Daten, damit diese nicht erneut angefragt werden müssen.

Service

- Interne Abbildung eines Diensts
- Kapselt sämtlich relevante Informationen eines Services
- Verfügt über Verweis auf In- und Out-Formate

Format

- Format eines Services mit Version, Flexible-Flag und Name.

CompatibilityAnswer

- Kapselt die Auswertung, ob ein Dienstpaar beziehungsweise eine Kante kompatibel ist.
- Gleichzeitig speichert es die passenden Formate des Dienstpaars beziehungsweise einer Kante
- Gibt es Alternativen, finden sich diese in einer Liste, die als Instanzvariable abgelegt ist.

Alternative

- Speichert die Alternativen eines inkompatiblen Dienstpaares.

Klassendiagramm zum Web-Frontend

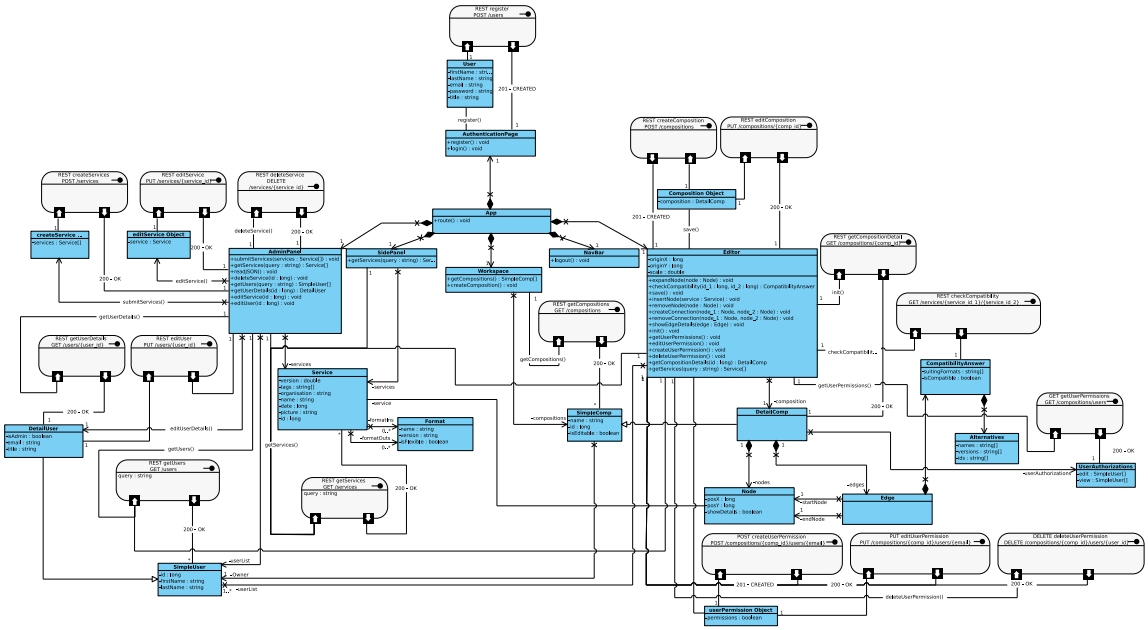

Abbildung 4.5: Klassendiagramm - Web

App
Alle Anfragen, die nicht an die REST-Schnittstelle vom Server gehen, werden hierhin zurück geleitet und dann entsprechend von hier aus geroutet.

AuthenticationPage
Bietet die Möglichkeit, über eine Maske sich einzuloggen oder zu registrieren.

AdminPanel
Bietet die Möglichkeiten, Services zu erstellen, bearbeiten und löschen und Rechte von Nutzern zu bearbeiten.

SidePanel
Ruft Services ab, stellt sie dar und lässt die angezeigten Services mit einer Suche einschränken.

Workspace
Zeigt eine Liste von allen einsehbaren und bearbeitbaren Kompositionen.

NavBar
Stellt einige Informationen zum Login-Status dar und ermöglicht den Logout.

Editor
Hier wird die Detailansicht einer Komposition angezeigt mit der Möglichkeit, diese durch Hinzufügen und Entfernen von Diensten und Kanten zu verändern, sofern die nötigen Rechte vorhanden sind. Es kann auch vom Autor der Komposition festgelegt werden, welche Nutzer die Komposition einsehen und bearbeiten können.

restlichen Klassen
Dienen zur Modellierung der Objekte, die über die REST-Schnittstelle ausgetauscht werden.

Kapitel 5

Sequenzdiagramme

Das dynamische Verhalten des Systems wird mittels Sequenzdiagrammen modelliert. Hier wird zunächst ein grobes, geräteübergreifendes Diagramm vorgestellt. Die restlichen Sequenzdiagramme beziehen sich nur auf wichtige Methoden in den gekapselten Ökosystemen APP, Website und Backend. Im Weiteren werden Lost und Found Messages verwendet, um die Kommunikation der App und Website mit dem Server zu visualisieren.

Geräteübergreifendes Sequenzdiagramm

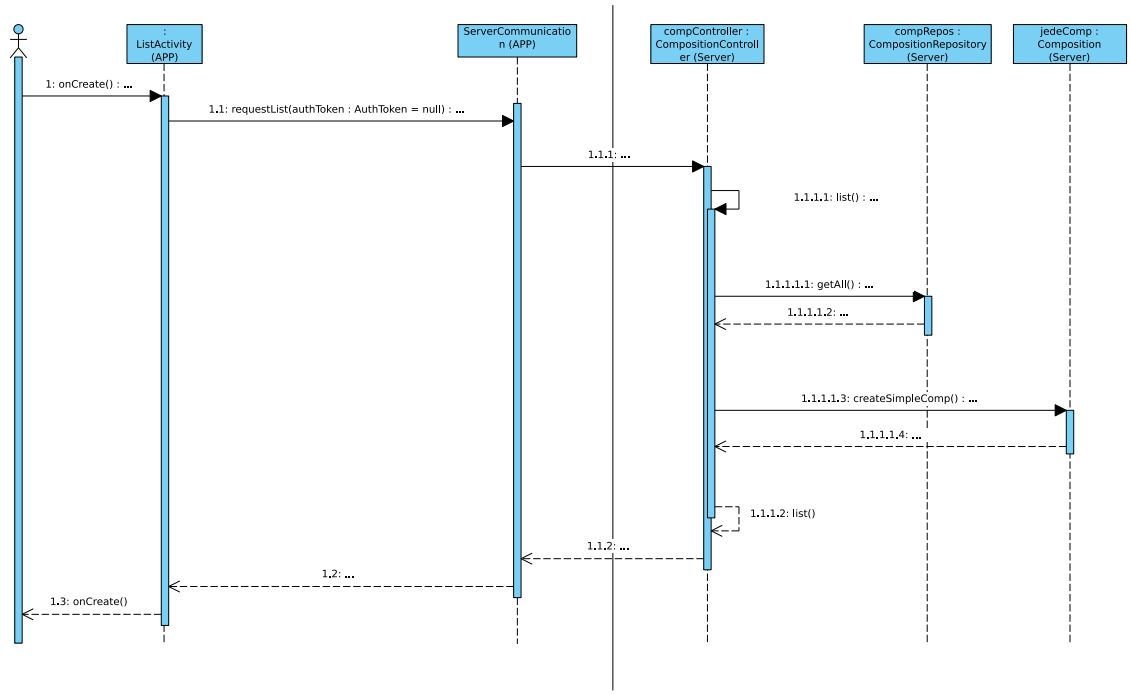

Abbildung 5.1: Sequenzdiagramm

Zusammengefasst beschreibt dieses Diagramm das grobe Verhalten des Systems, das abläuft, wenn

ein Nutzer über die App sich die für ihn anzeigbaren Kompositionen auflisten lässt. Dabei bezieht es die Kommunikation zwischen Android App und Backend mit ein.

Der Benutzer öffnet die App, wodurch die Methode onCreate() der ListActivity aufgerufen wird. Beim Start sendet die App ein HTTPS-Request an den Server. Daraufhin fragt der Server die Daten aus der Datenbank ab und erstellt aus diesen ein versendbares Objekt in Form einer Liste von SimpleComp-Objekten. Diese schickt das Backend im Rahmen der Antwort auf den GET-Request an die App, welche die erhaltenen Daten verarbeitet, worauf sie diese dem Benutzer anzeigen kann.

Sequenzdiagramme der App

Öffnen der ListActivity

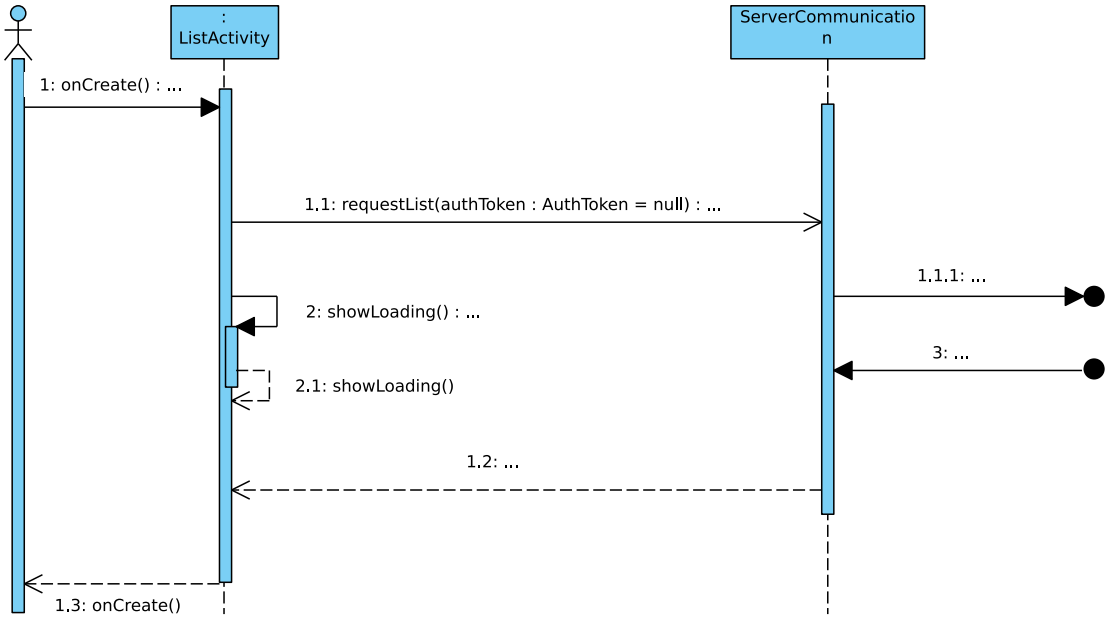

Abbildung 5.2: Sequenzdiagramm - Öffnen der ListActivity

Dieses Sequenzdiagramm zeigt den Vorgang, der abläuft, wenn die ListActivity initial gestartet wird und mit einer Liste aus Kompositionseinträgen zu füllen ist.

Der Ablauf der Aktivitäten beginnt mit dem Start der ListActivity, bei dem ein Rufen der Standard-Methode onCreate() erfolgt. Da der Nutzer noch nicht eingeloggt ist, startet die App eine requestList()-Anfrage ohne Token. Diese läuft in einem eigenen Thread, damit der UI-Thread nicht blockiert. Die Methode requestList() führt einen HTTPS-Request an das Backend aus. In der Zeit, in der die ListActivity die Antwort noch nicht erhalten hat, zeigt sie mithilfe von showLoading() eine Ladeanimation an.

Klicken auf Kompositionseintrag

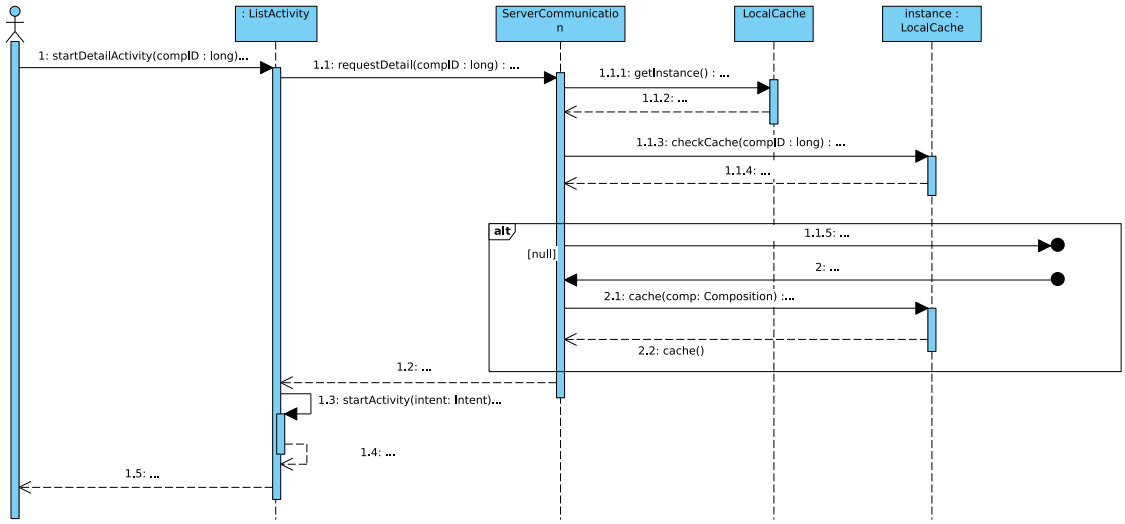

Abbildung 5.3: Sequenzdiagramm - Klick auf Kompositionseintrag

Fortführend schließt sich dieses Sequenzdiagramm an den Usage-Flow des obigen an. Ist die Liste geladen, will der Nutzer sich früher oder später eine der Kompositionen detailliert als Grafik anzeigen lassen. Hierfür reicht ein Tippen auf das jeweilige Listenelement aus, um die Detailansicht in Form der DetailActivity aufzurufen.

Durch die genannte UI-Aktion wird die Methode startDetailActivity() aufgerufen, was die statische Methode requestDetail() veranlasst, im LokalCache nach der gewünschten Komposition zu suchen. Falls diese noch nicht im Cache ist, sendet die App eine HTTPS-Request an den Server. In der Antwort auf diesen Request sind alle nötigen Details der Komposition enthalten, wozu unter anderem ihre Nodes zählen. Die Details werden in der Komposition gespeichert, die wiederum im LokalCache abgelegt ist, und so auch an die ListActivity zurückgegeben. Diese übergibt die nun mit Details ausgestattete Komposition an einen Intent, der zum Starten der DetailsActivity dient.

Da LocalCache ein Singleton ist, muss man zu Beginn die Instanz abrufen. Wir planen, die Instanz vorzudefinieren (eager instantiation), wodurch der Fall, dass man die Instanz erst erzeugen müsste, nie eintritt.

Die Methode httpsRequest() wird nicht implementiert, sondern dient hier zur Abstraktion. In der Implementation wird diese Abfrage asynchron erfolgen und in einem anderen Thread laufen, so dass sie den UI-Thread nicht blockiert.

Sequenzdiagramme des Servers

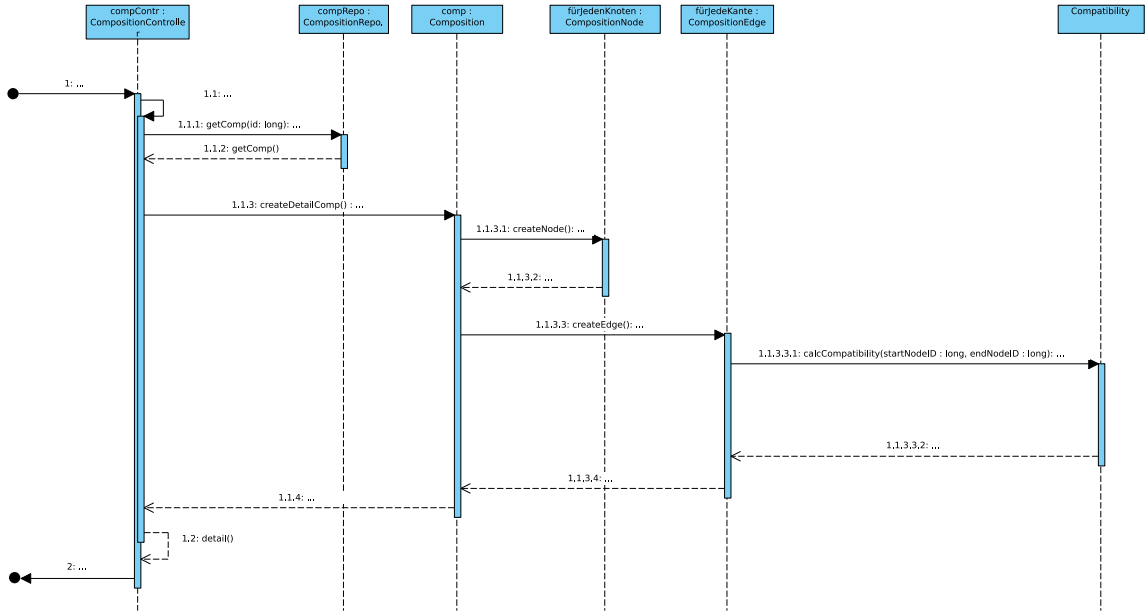

Abbildung 5.4: Sequenzdiagramm - Kompositionsanfrage

Nach der Betrachtung der Android-App folgt nun eine detailliertere Darstellung, wie sich die Server-Seite verhält, wenn eine Komposition in Detailansicht angefragt wird.

Eine solche Anfrage wird vom CompositionsController entgegengenommen. Die angefragte Komposition wird in der Datenbank nachgeschlagen (über das CompositionRepository) und als Composition-Objekt zurückgegeben. Anschließend wird eine DetailComp erstellt, welche die benötigten Daten für das Frontend enthält. Um diese zu erstellen, müssen zunächst alle CompositionNodes und danach alle CompositionEdges in versendbare Objekte (Nodes und Edges) umgewandelt werden. Dabei wird für jede Kante noch die Kompatibilität überprüft und mögliche Alternativen werden gesucht und ggf. gespeichert. Zurückgeliefert wird also eine DetailComp, die in allen Kanten eine CompatibilityAnswer enthält.

Wir haben uns dafür entschieden, eine Util-Klasse zur Berechnung der Compatibility zwischen zwei verschiedenen Diensten (repräsentiert durch ihre IDs) zu verwenden. Da auch HTTPS-Requests erwartet werden, die nur zu zwei einzelnen Diensten die Kompatibilität erfahren wollen, wäre eine Implementierung dieser Funktion in der Edge-Klasse nicht praktikabel.

Da es sich bei dem durch dieses Sequenzdiagramm modellierten dynamischen Verhalten um das komplexeste unseres Backends handelt und sich viele andere Abläufe in ihm widerspiegeln, sei nur dieses als Repräsentant der Arbeitsweise aufgeführt.

Sequenzdiagramme des Web Frontends

Erstellen einer Komposition

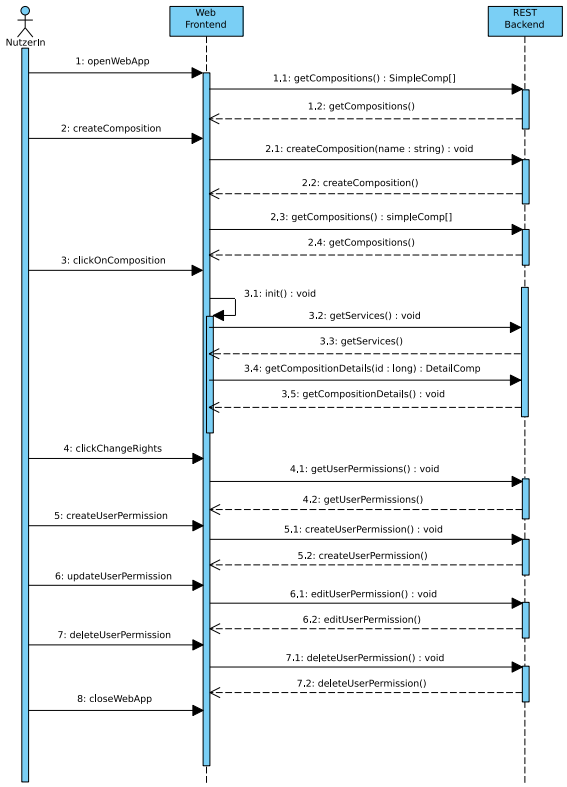

Abbildung 5.5: Sequenzdiagramm - Erstellen einer Komposition

Um eine neue Komposition zu erstellen, fragt das Web Frontend zunächst die dem Nutzer verfügbaren Kompositionen vom Backend an und stellt diese bei Erhalt der Antwort dar. Über ein Eingabefeld lässt sich ein Name für eine neue Komposition festlegen und mit einem zusätzlichen Knopf erstellen. Hiernach wird die Liste der verfügbaren Kompositionen aktualisiert. Ein Klick auf die Komposition löst das Abrufen der bisher nicht erhaltenden Kompositionsdetails vom Backend aus. Erhält das Frontend diese, zeigt es sie in der Bearbeitungsansicht an. Als Autor der Komposition lassen sich nun die Listen mit den Nutzenden einsehen und bearbeiten, die diese Komposition betrachten oder bearbeiten dürfen. Hierbei fordert das Web Frontend alle Informationen immer aus dem Backend an, beziehungsweise schickt Aktualisierungen vom Nutzenden an das Backend.

Bearbeiten einer Komposition

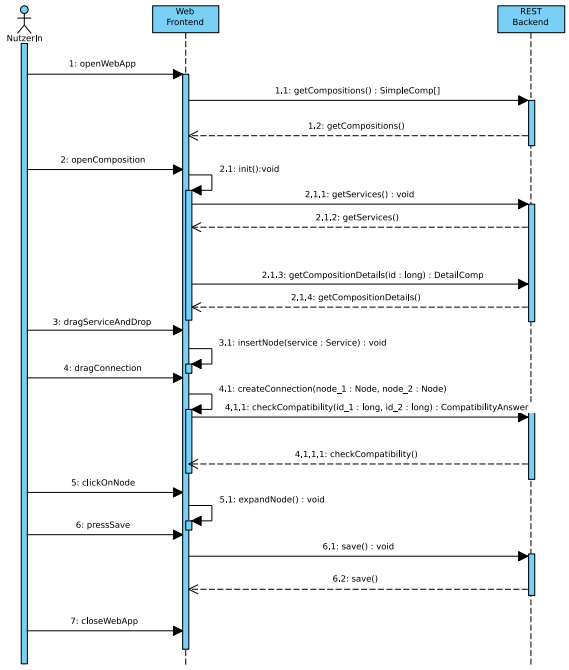

Abbildung 5.6: Sequenzdiagramm - Bearbeiten einer Komposition

Um eine Komposition zu bearbeiten, fragt das Frontend zunächst wieder die dem Nutzer verfügbaren Kompositionen vom Backend an und stellt diese dar. Ein Klick auf die Komposition lässt die Details sowie Services vom Backend abrufen und in der Bearbeitungsansicht anzeigen. Indem man einen Service aus dem SidePanel in das Canvas zieht, erstellt man lokal einen Knoten. Das Verbinden von zwei Knoten erstellt zunächst lokal eine Verbindung, wobei eine Anfrage an das Backend geht, die das Überprüfen der Kompatibilität überprüfen lässt. Beim Klicken auf einen Knoten wird eine Detailansicht für den verwendeten Dienst angezeigt. Wiederum sendet ein Klick auf Speichern die aktualisierte Komposition an den Server.

Bedienung des Adminpanels

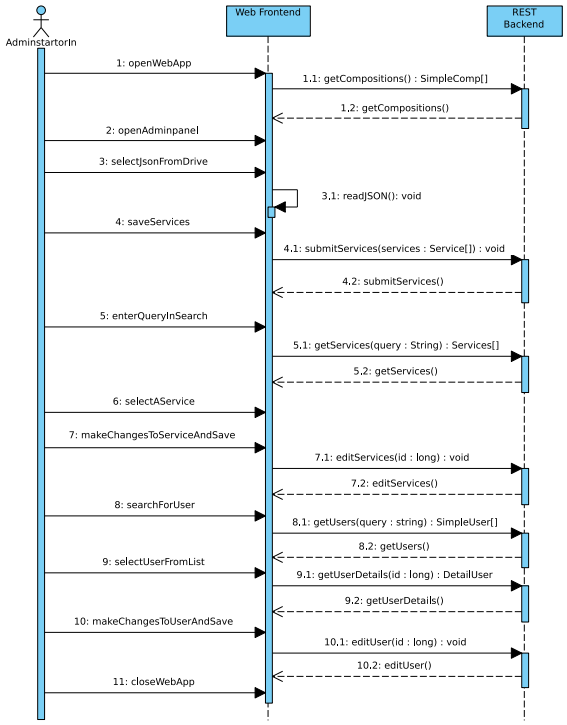

Abbildung 5.7: Sequenzdiagramm - Bedienung des Adminpanels

Das Adminpanel lässt sich aus der Kompositionsübersicht erreichen. Über eine Maske lässt sich eine JSON-Datei mit Services einlesen, die zunächst nur lokal zwischengespeichert und dann bei Bestätigung an das Backend übermittelt werden. Ein Suchfeld für Dienste erlaubt, eine Suchanfrage an das Backend zu schicken und durch Auswählen eines Listeneintrags die Details eines Services zu bearbeiten, die das Frontend bei Bestätigung der Eingabe an das Backend übermittelt. Weiterhin ermöglicht das Frontend die Suche nach registrierten Nutzenden. Wiederum fragt das Auswählen eines Nutzers die Details vom Backend ab, die ein Administrierender bearbeiten kann. Nach Bestätigung der Änderungen sendet das Frontend die angepassten Daten an das Backend, welches sie speichert.