

WODSS

Event Planner

**Autoren: Andreas Gassmann, Jonas Frehner,
Lukas Schönbächler**

n|w University of Applied Sciences and Arts
Northwestern Switzerland

FHNW

Schweiz

März 27, 2017

Abstract

Das vorliegende Projekt wurde im Rahmen des Moduls "wodss" der FHNW realisiert. Es wurde eine Plattform entwickelt, mit deren Hilfe auf einfache Art und Weise CS-Seminare und Anmeldungen zu den entsprechenden Anlässen verwaltet werden können. Die Lösung wurde mit Hilfe von Angular2 und Ionic im Frontend und SpringBoot im Backend realisiert. Architektonisch basiert das Projekt auf der Serverseite auf so genannten Microservices.

Contents

1	Mockups	4
1.1	Login	4
1.2	Übersicht	5
1.3	Detailansicht	7
1.4	Einschreiben	7
2	Architektur - Backend	8
2.1	Schichtenarchitektur	8
2.1.1	Webschicht	8
2.1.2	Businessschicht	8
2.1.3	Datenbankschicht	8
2.2	Microservices	8
2.2.1	Registry	8
2.2.2	Eventmanagement	9
2.2.3	Frontend	9
2.2.4	Mailer	9
2.2.5	Scheduler	9
2.3	Sicherheit	10
2.4	Datenbank	11
3	Architektur - Frontend	12
3.1	Ionic	12
4	Projektaufbau - Backend	13
4.1	Registry	13
4.2	Eventmanagement	13
4.2.1	Endpoints	14
4.2.2	Entities	15
4.2.3	Models	15
4.2.4	Security	15
4.3	Frontend	15
4.4	Mailer	16
4.5	Scheduler	16
5	Projektaufbau - Frontend	17
6	Technologien	18
6.1	Frontend	18
6.2	Backend	18
6.2.1	Authentifizierung und Autorisierung	18
7	Design-Entscheide	19

8 Lessons learned	20
Appendices	21

1 Mockups

1.1 Login

The image shows two hand-drawn mockups of login forms. The top mockup is titled 'Login Intern (FHNW)' and features a header bar with 'Titel' on the left and 'Login' on the right. Below the header is a form with two tabs: 'Intern' (selected) and 'Extern'. The form contains fields for 'Username' and 'Password', and a green 'Login' button. The bottom mockup is titled 'Login Extern' and features a header bar with 'Login' on the right. Below the header is a form with two tabs: 'Intern' and 'Extern' (selected). The form contains fields for 'Username' and 'Password', and a green 'Abmelden' button.

Titel Login

Intern Extern

Username

Password

Login

LOGIN: EXTERN

Login

Intern Extern

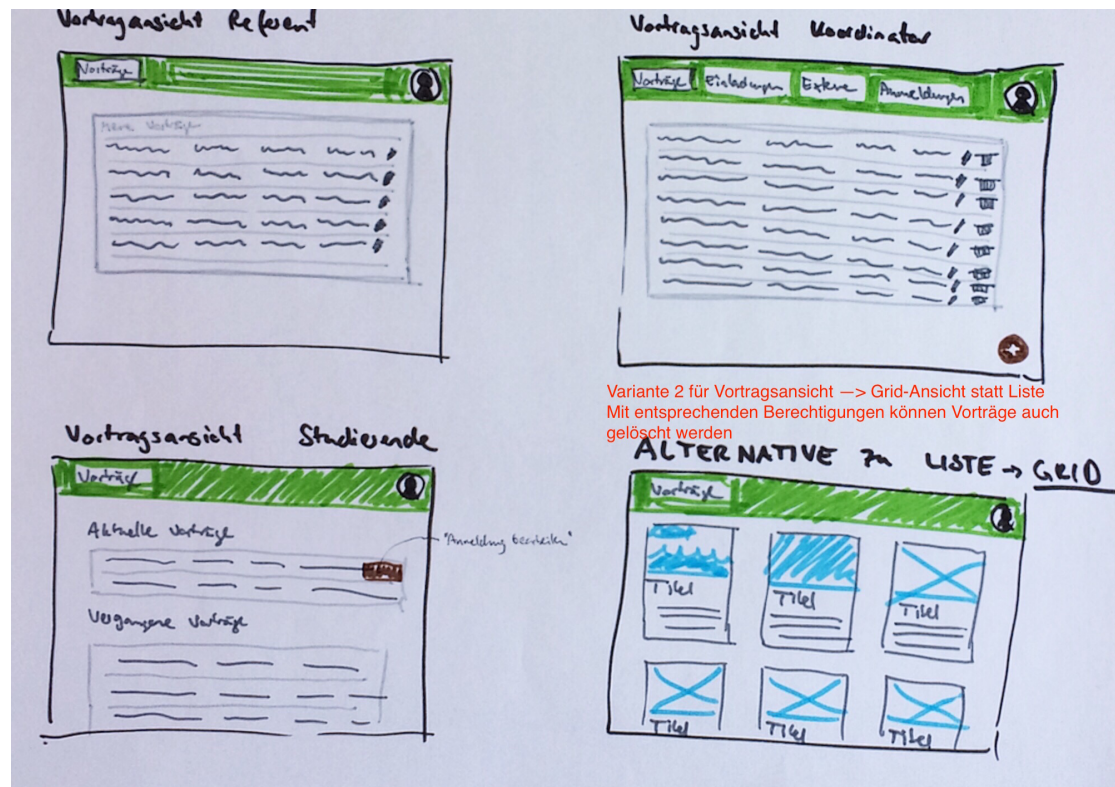
Username

Password

Abmelden

Login Extern

1.2 Übersicht



Externe Personen Übersicht

Externe Person, die sich abgemeldet hat

Externe Person erfassen

Externe

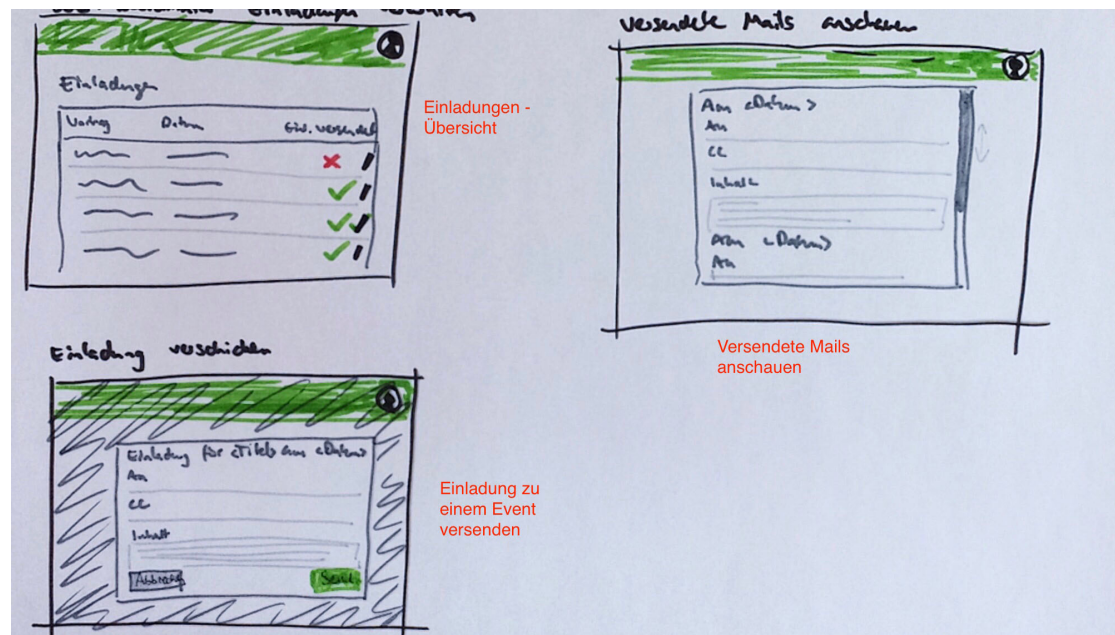
Name _____

Vorname _____

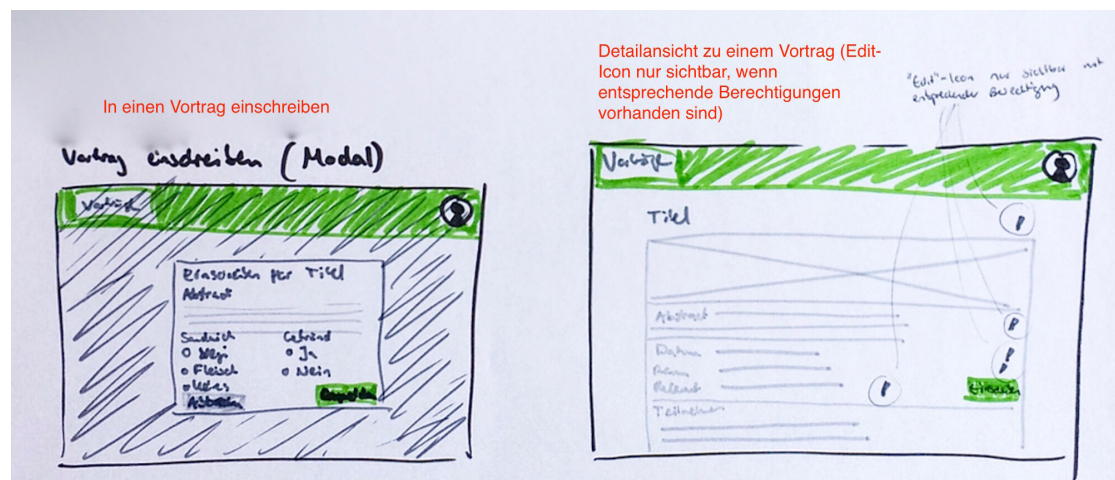
E-Mail _____

Abbrechen Erstellen

1.3 Detailansicht



1.4 Einschreiben



2 Architektur - Backend

Wie bereits in der Einleitung erwähnt, implementierten wir dieses Projekt als Microservices. Wir verwenden dazu "Eureka", eine von Netflix entwickelte Library (siehe auch: <https://github.com/netflix/eureka>). Die Applikation lässt sich in fünf Microservices unterteilen, welche im nachfolgenden Abschnitt genauer beschrieben werden. Die Microservices kommunizieren per REST untereinander.

2.1 Schichtenarchitektur

2.1.1 Webschicht

Die Webschicht wird automatisch durch den `RepositoryRestResource` bereitgestellt. Das heisst, die betreffenden `RepositoryRestResource`n stellen automatisch die so annotierten JPA-Entitäten via REST-Schnittstelle bereit.

2.1.2 Businesschicht

Da das vorliegende Projekt eine relativ einfache Business-Schicht aufweist (vielfach handelt es sich um einfache GET/POST/PUT/DELETE auf bestehenden Entitäten), wird diese ebenfalls von den `RepositoryRestResource`n wahrgenommen. Wo eine etwas ausgefeilterte Logik notwendig war (z.B. beim Login oder bei der Passwort-Zurücksetzen-Funktionalität) wurden eigene `RestController`s erstellt (wie beispielsweise das `LoginRepository`).

2.1.3 Datenbankschicht

Die Datenbankschicht wurde mit Hilfe von JPA-Repositories realisiert. Für eine genauere Diskussion zum Aufbau der Persistenzschicht siehe unterkapitel Datenbank.

2.2 Microservices

2.2.1 Registry

Die Registry stellt das "Bindeglied" zwischen den verschiedenen Microservices dar. Die Microservices verbinden sich auf die Registry um dort die URLs der anderen Microservices zu bekommen. Dadurch wird es möglich die Applikation auf mehreren Servern verteilt laufen zu lassen. Im Falle einer bestehenden Eureka Installation/Architektur muss dieser Microservice nicht nochmals zusätzlich gestartet werden (bestehende Registry kann verwendet werden). Durch die Verwendung des "Loadbalanced" `RestTemplate`s lösen Microservices die logischen Namen anderer Microservices selbstständig mit Rückgriff auf die Registry zu korrekten IP-Adressen auf.

2.2.2 Eventmanagement

Der Microservice Eventmanagement ist der zentrale Service unserer Applikation. Er beinhaltet die Persistenzschicht mit allen Entitäten, sowie die Businesslogik der Anwendung. Zudem werden alle Zugriffe auf die Daten und die Logik via diesem Service getätigt. Hierfür werden die CRUD Operationen über eine REST-API zur Verfügung gestellt (mittels den oben erwähnten RepositoryRestResource-Controllern). Bestimmte Endpunkte stehen dabei nur nach erfolgreicher Authentifizierung und Autorisierung zur Verfügung. Die komplette Schnittstellendokumentation ist im Anhang (Swagger Schnittstellendokumentation) aufgeführt.

2.2.3 Frontend

Dieser Microservice stellt nur einen Container bereit, der dazu dient, statische Inhalte wie HTML-/CSS-Files, Bilder und so weiter zur Verfügung zu stellen. Er enthält keinerlei Business-Logik und speichert keine Benutzerdaten. Konkret wird der Build Folder unserer Ionic-App in den assets/static Order kopiert. Sollte bereits eine bestehende Webserver Infrastruktur zur Verfügung stehen (z.B. Apache oder Nginx), können diese Assets auch ohne zusätzlichen Microservice gehostet werden.

2.2.4 Mailer

Der Mailer ist ein simpler Mailservice, welcher eine API zum Versenden von Mails bietet. Diese API wird durch ein statisches Token geschützt, das alle Aufrufer mitschicken müssen, um Mails versenden zu können. Da der Token nur intern (via application.properties-Files der entsprechenden Microservices) bekannt ist, werden ausschliesslich interne (von anderen Microservices) Anfragen entgegengenommen.

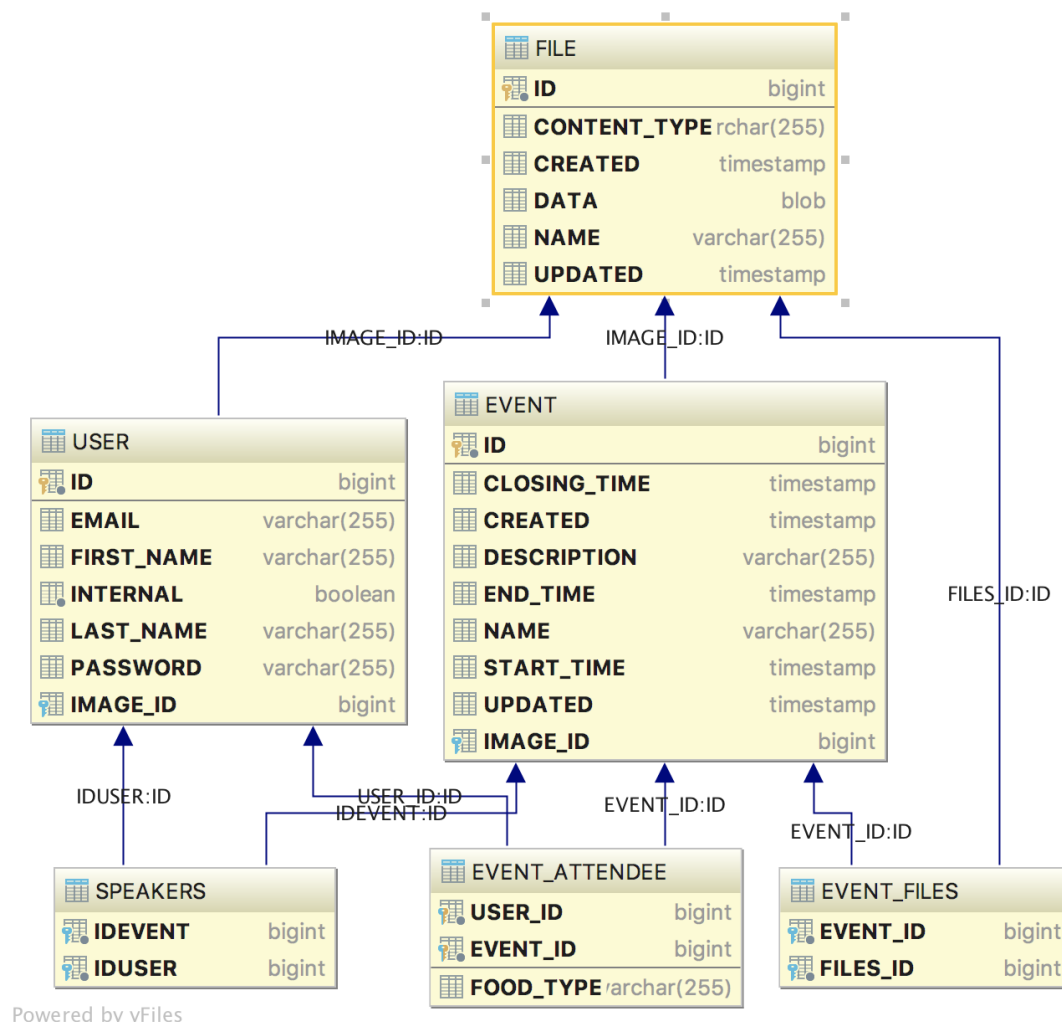
2.2.5 Scheduler

Der Scheduler Service führt zu bestimmten Zeiten verschiedene Tasks aus. Ein Beispiel hierfür wäre, unreferenzierte Mediendateien in der Nacht zu löschen, Erinnerungsmails zu versenden oder Events aufgrund verschiedener Kriterien zu archivieren.

2.3 Sicherheit

2.4 Datenbank

Um für die Datenintegrität garantieren zu können, werden alle Entitäten wie oben durch einen einzigen Microservice verwaltet (<https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/>).



3 Architektur - Frontend

3.1 Ionic

Pages usw.

4 Projektaufbau - Backend

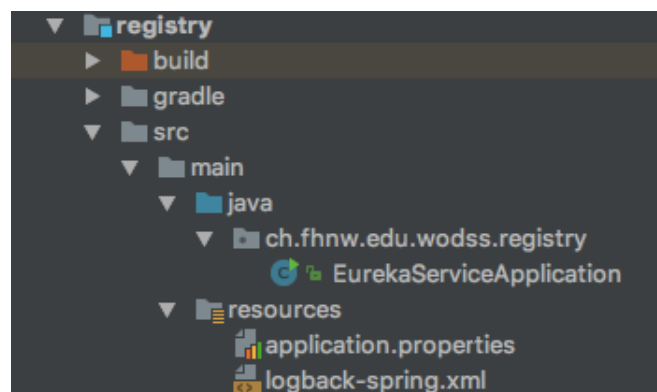
Für alle Microservices wurde jeweils ein eigenes Sub-Projekt erstellt, die jeweils aus einem build-, gradle- und src-Ordner bestehen. Für jeden Microservice existiert ein eigenes build.gradle, das die Projektspezifischen Abhängigkeiten enthält. Der Root-Folder enthält alle Microservices, sowie ein Ordner "ionic", der allen Frontend-Code enthält.

Zudem haben wir vier verschiedene .sh-Files erstellt, mit denen die vier Microservices Eventmanagement, Registry, Mailer und Scheduler je einzeln gestartet werden können. Dazu ein .sh-File mit dem alle erwähnten Services zusammen gestartet (run.sh) werden können, gestoppt werden können (kill.sh) oder gestoppt und gleich neu gestartet werden können (restart.sh).

Nachfolgend gehen wir kurz auf den internen Aufbau der einzelnen Microservices ein. Dabei werden wir nur den Aufbau des src-Ordners präsentieren, da sich der Build- und Gradle-Ordner jeweils sehr ähnlich zeigen. Zusätzlich werden auch die relevanten Klassen-Diagramme (in UML) pro Microservice präsentiert.

4.1 Registry

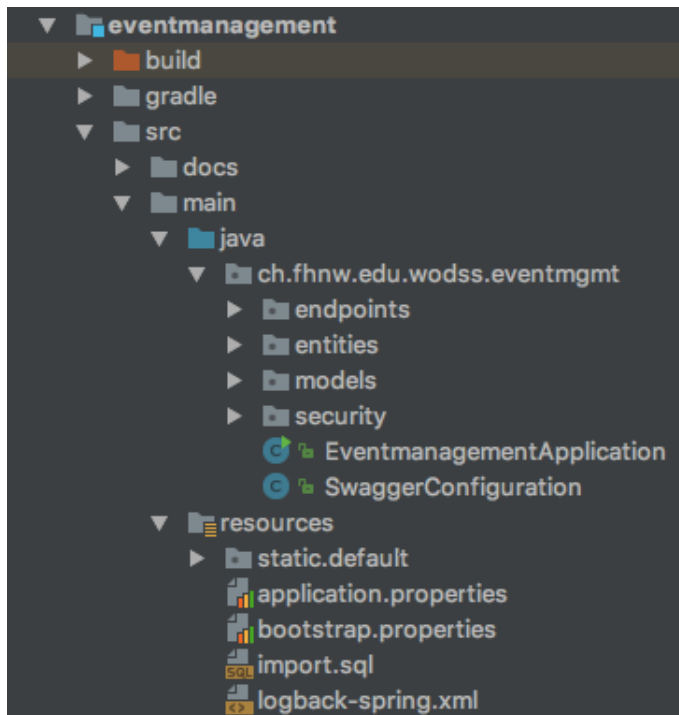
Die Registry hat folgende sehr simple Projektstruktur:



Der Grund für die schlanke Struktur liegt auf der Hand: da sich die Verwendung der Eureka-Library in diesem Projekt grösstenteils auf Annotationen beschränkt besteht die Registry nur aus einer einzigen Klasse, die mittels "EnableEurekaServer" und "SpringBootApplication" annotiert wurde. Die restlichen Microservices verwenden als Klienten die Annotation "EnableDiscoveryClient".

4.2 Eventmanagement

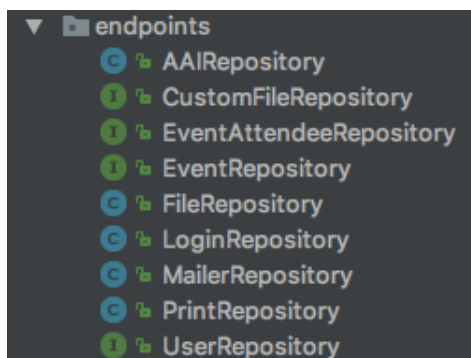
Dieser Microservice ist der komplexeste der fünf. Die grobe Struktur präsentiert sich wie folgt:



Die Swagger-Configuration-Klasse übernimmt, wie der Name erahnen lässt, die Konfiguration der Swagger-Dokumentation der REST-Schnittstelle. Die Klasse EventmanagementApplication dient der Initialisierung des Microservices.

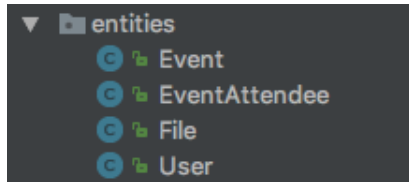
4.2.1 Endpoints

Im Folder "endpoints" sind alle Klassen enthalten, die entweder mittels RepositoryRestResource (sofern die Klasse auf einer konkreten JPA-Entität basiert) oder RestController (sofern die Klasse zusätzliche Geschäftslogik bereitstellt) annotiert sind. Die Struktur sieht folgendermassen aus:



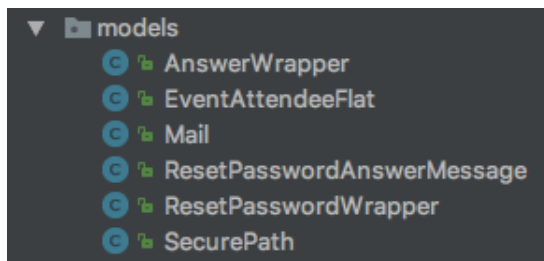
4.2.2 Entities

Der Ordner "entities" präsentiert sich wie folgt:



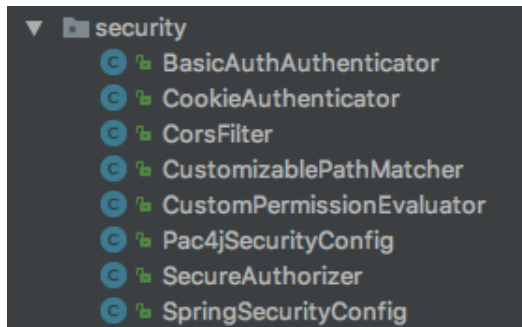
4.2.3 Models

Der Folder "model" enthält folgendes:



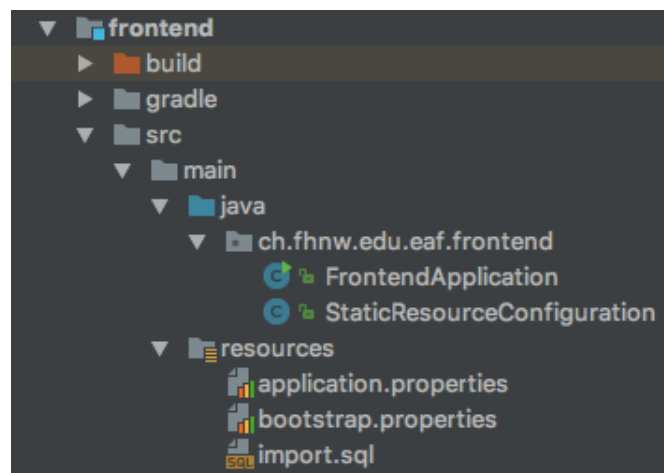
4.2.4 Security

Der Ordner "security" schlussendlich hat folgenden Inhalt:



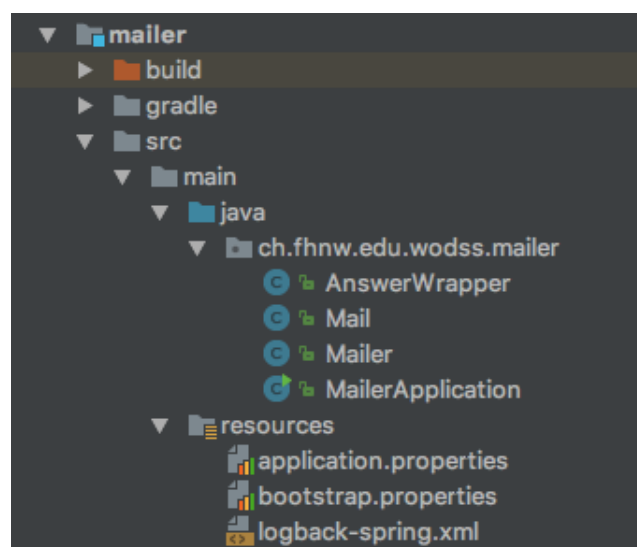
4.3 Frontend

Frontent:



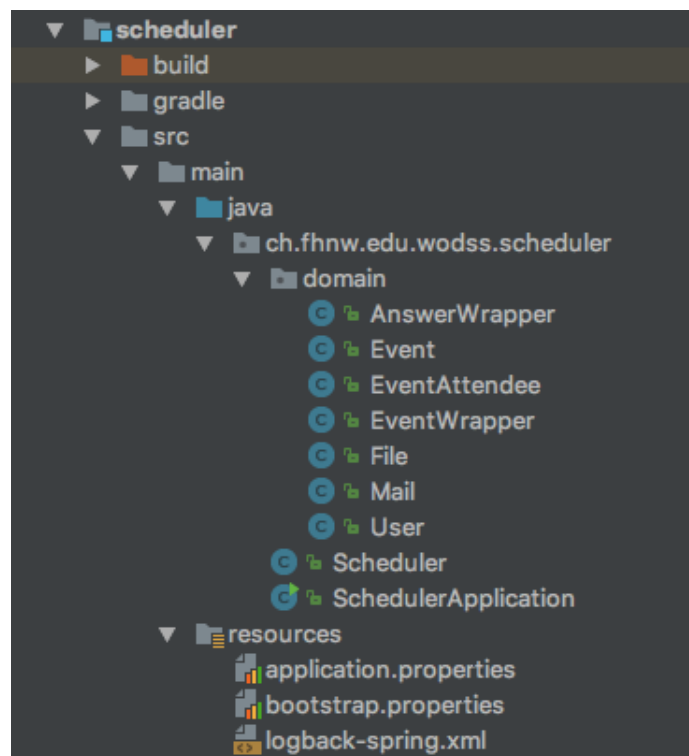
4.4 Mailer

Mailer:



4.5 Scheduler

Scheduler:



5 Projektaufbau - Frontend

6 Technologien

6.1 Frontend

Nach dem Start mit Angular 2 merkten wir schnell, dass uns einige Standardkomponenten fehlten. Außerdem war der Aufwand die Seite Mobile tauglich zu machen größer als erwartet. Aus diesem Grund wechselten wir nach kurzer Zeit auf Ionic 2. Da Ionic 2 auf Angular 2 aufbaut, war der Wechsel schnell vonstatten gegangen. Mithilfe der neuen Sidemenu Komponente kann die Webseite ohne weitere Probleme für Desktops optimiert werden.

6.2 Backend

Wie von der Projektbeschreibung vorgeschrieben, verwenden wir Java mit Spring Boot im Backend. Zusätzlich verwenden wir die spring-boot-data Erweiterung, welche uns das einfache erstellen von Rest Repositories ermöglicht.

6.2.1 Authentifizierung und Autorisierung

Für die Authentifizierung verwenden wir PAC4J und verwenden das JWT (Json Web Token). Zusätzlich können mithilfe von PAC4J Permissions und Rollen definiert werden. Abhängig von der Rolle, erhält der Benutzer zusätzliche Privilegien. Ein normaler Besucher kann lediglich Events ansehen und nichts editieren. Wir haben folgende Rollen identifiziert:

1. Besucher
2. Angemeldeter Gast (kann sich zusätzlich einschreiben)
3. Referent (kann zusätzlich den eigenen Event bearbeiten und Dateien anhängen)
4. Koordinator (kann alle Entitäten bearbeiten und löschen)

7 Design-Entscheide

Anfangs wollten verschiedene neue Technologien wie z.B. Elm, GraphQL und Microservices testen. Es stellte sich aber schnell heraus, dass es dabei einige Probleme gibt, welche im folgenden Abschnitt erläutert werden.

1. Elm

Elm ist eine relativ neue funktionale Sprache, die zu Javascript kompiliert. Der grosse Vorteil von Elm sind die Typsicherheit sowie der funktionale Aspekt. Es sollte dadurch beispielsweise keine Runtime-Fehler, die in Javascript an der tagesordnung sind, mehr geben. Es hat sich aber herausgestellt, dass das Ökosystem zwar bereits viele Funktionen bietet, wenn man aber genaue Anforderungen hat muss man teilweise Kompormisse eingehen. Weiter wird user Projekt durch die Wahl einer "neuen" Programmiersprache weniger wartbar.

2. GraphQL

Durch die JPA-GraphQL ist das initiale Aufsetzen eines GraphQL Endpoints nach einigen Versuchen relativ gut gegangen. Leider fehlen der Library aber noch einige zentrale Features, weshalb man nicht komplett auf die REST-API hätte verzichten können. Anstelle von zwei verschiedenen Endpunkten haben wir uns schlussendlich aus gründen der Wartbarkeit für REST entschieden.

8 Lessons learned

Im Laufe unseres Projekts wurden wir mit einigen Problemen konfrontiert, die insgesamt einen grossen Teil unserer Zeit in Anspruch genommen haben.

1. Microservices
2. Pac4J
3. Angular
Angular Material2 noch nicht ready, Ionic mehrheitlich ohne Probleme
4. ResourceRestController

Appendices

Swagger Schnittstellendokumentation

https://htmlpreview.github.io/?https://raw.githubusercontent.com/lukeisontheroad/simple_event_planner/master/docs/doc.html

Github Repository

https://github.com/lukeisontheroad/simple_event_planner

Microservices best practices

<https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/>

PAC4J

<http://www.pac4j.org/>