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

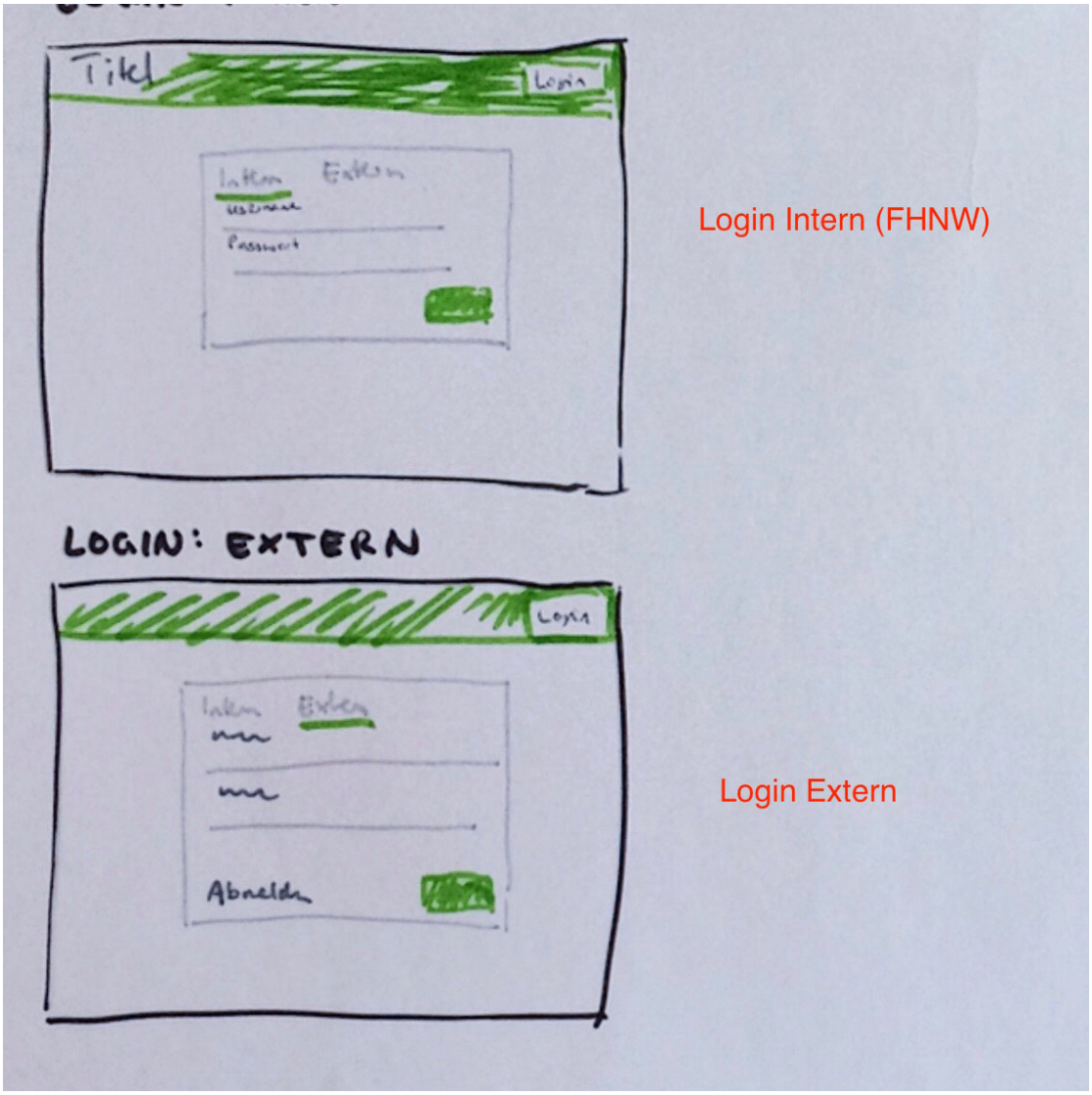
Abstract

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 einerseits CS-Seminare und Anmeldungen zu den entsprechenden Anlässen verwaltet werden können. Die Lösung wurde mit Hilfe von Angular2 im Frontend und SpringBoot im Backend realisiert. Architektonisch basiert das Projekt auf der Serverseite auf so genannten Microservices.

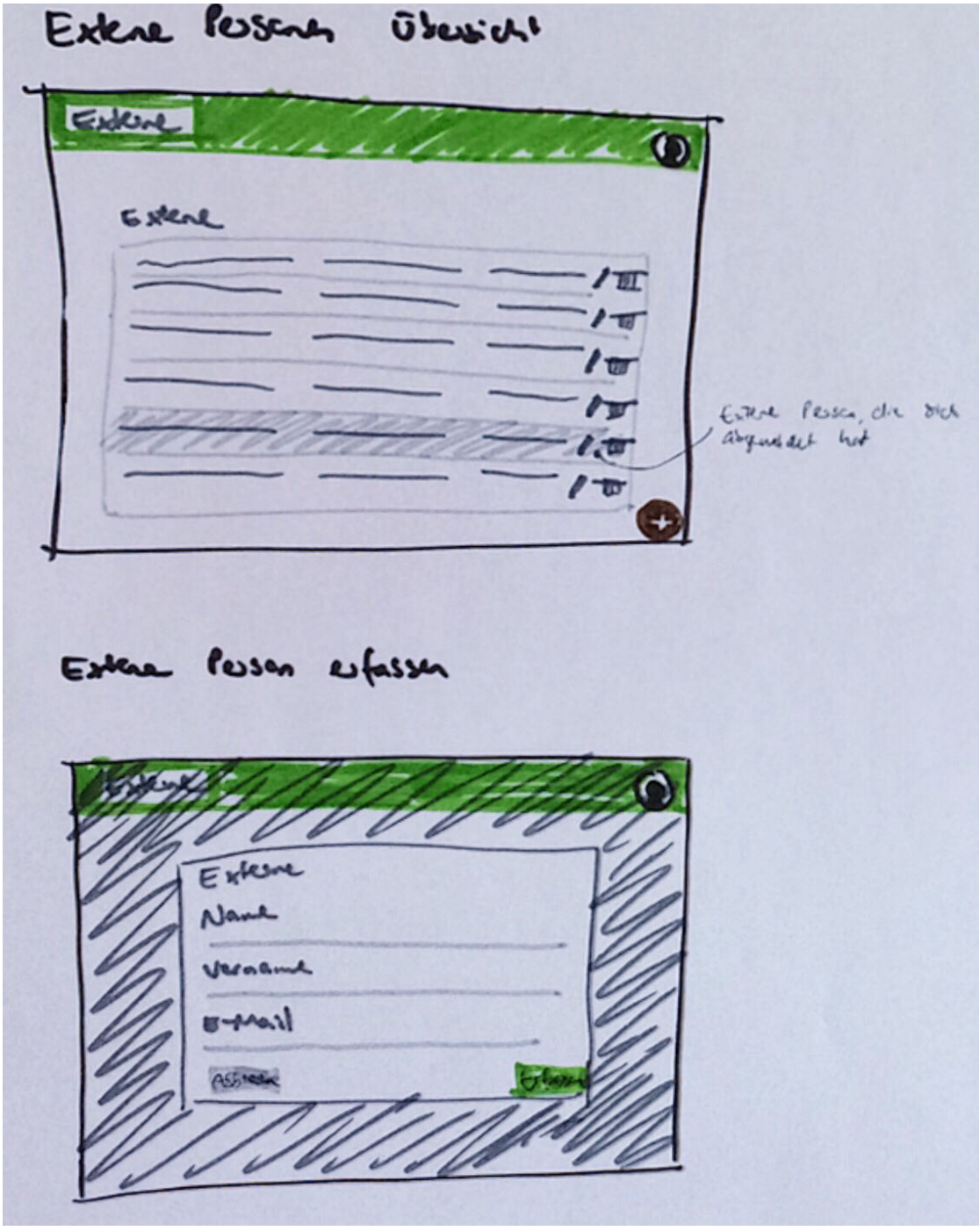
Contents

1 Mockups

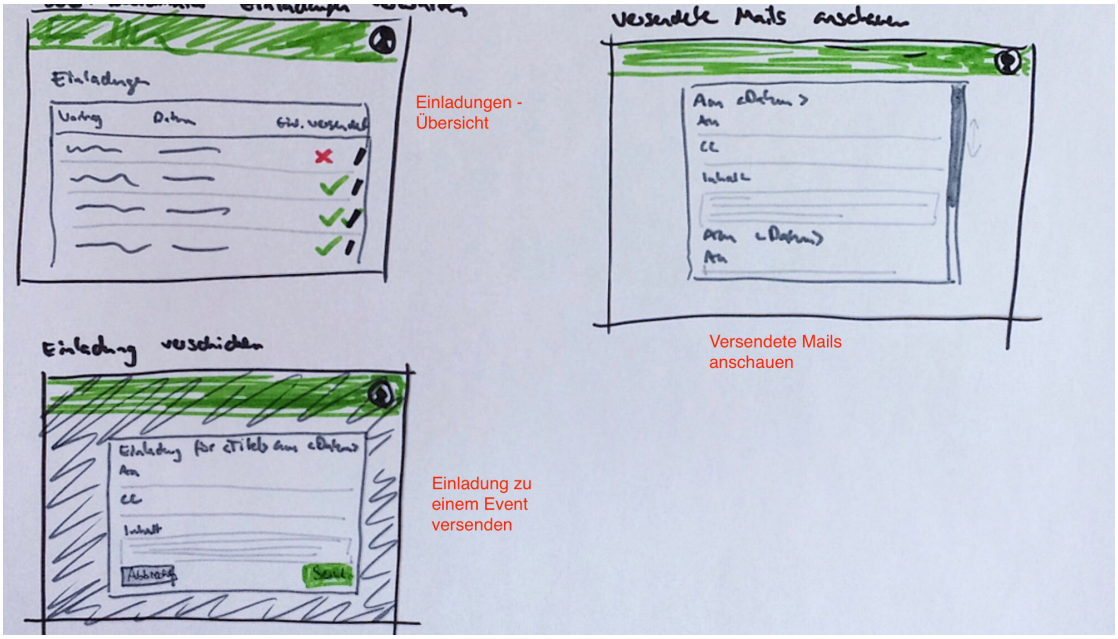
1.1 Login



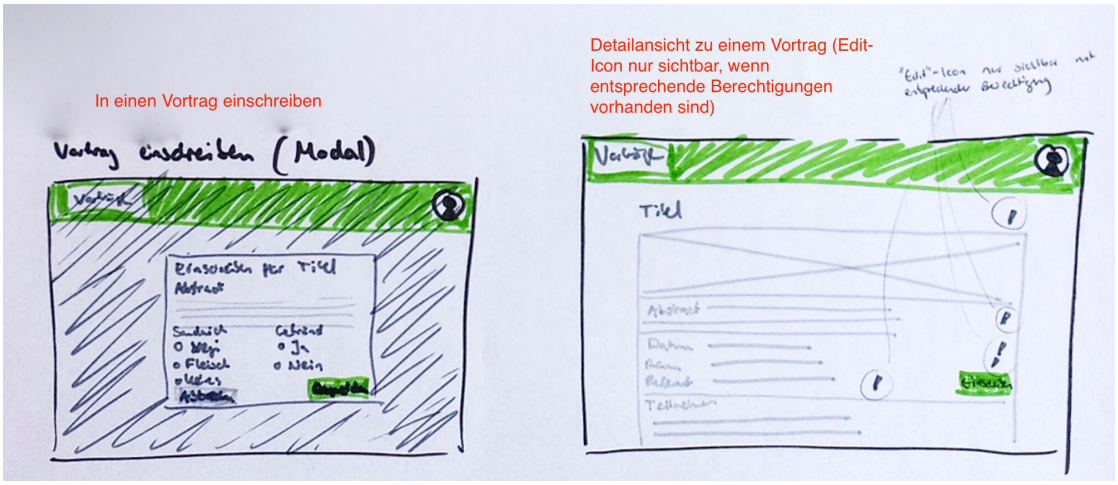
1.2 Übersicht


Externe Personen Übersicht
Externe
Externe
Externe Person, die sich abgemeldet hat
Externe Person erfassen
Externe
Externe
Name
Vorname
E-Mail

1.3 Detailansicht



1.4 Einschreiben



2 Architektur

Unsere Implementation des "Event-Planners" wird als Microservices realisiert. Wir verwenden dazu "Eureka", eine Library entwickelt von Netflix 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

Wahrgenommen durch RepositoryRestController

2.1.2 Businesschicht

Wahrgenommen durch RepositoryRestController

2.1.3 Datenbankschicht

Siehe unterkapitel Datenbank

2.2 Microservices

2.2.1 Registry

Das "Bindeglied" zwischen den verschiedenen Microservices. 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 Eurika Installation/Architektur muss dieser Microservice nicht nochmals zusätzlich gestartet werden (bestehende Registry kann verwendet werden).

2.2.2 Eventmanagement

Beinhaltet alle Entitäten (Persistance Layer) sowie die Businesslogik unserer Applikation und ermöglicht den Zugriff von aussen. Hierfür werden die CRUD Operationen über eine REST-API zur Verfügung gestellt. 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

Container welcher nur dazu dient, statische Inhalte zur Verfügung zu stellen. Enthält keinerlei Business Logik. Konkret wird der Build Order 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 gehosted werden.

2.2.4 Mailer

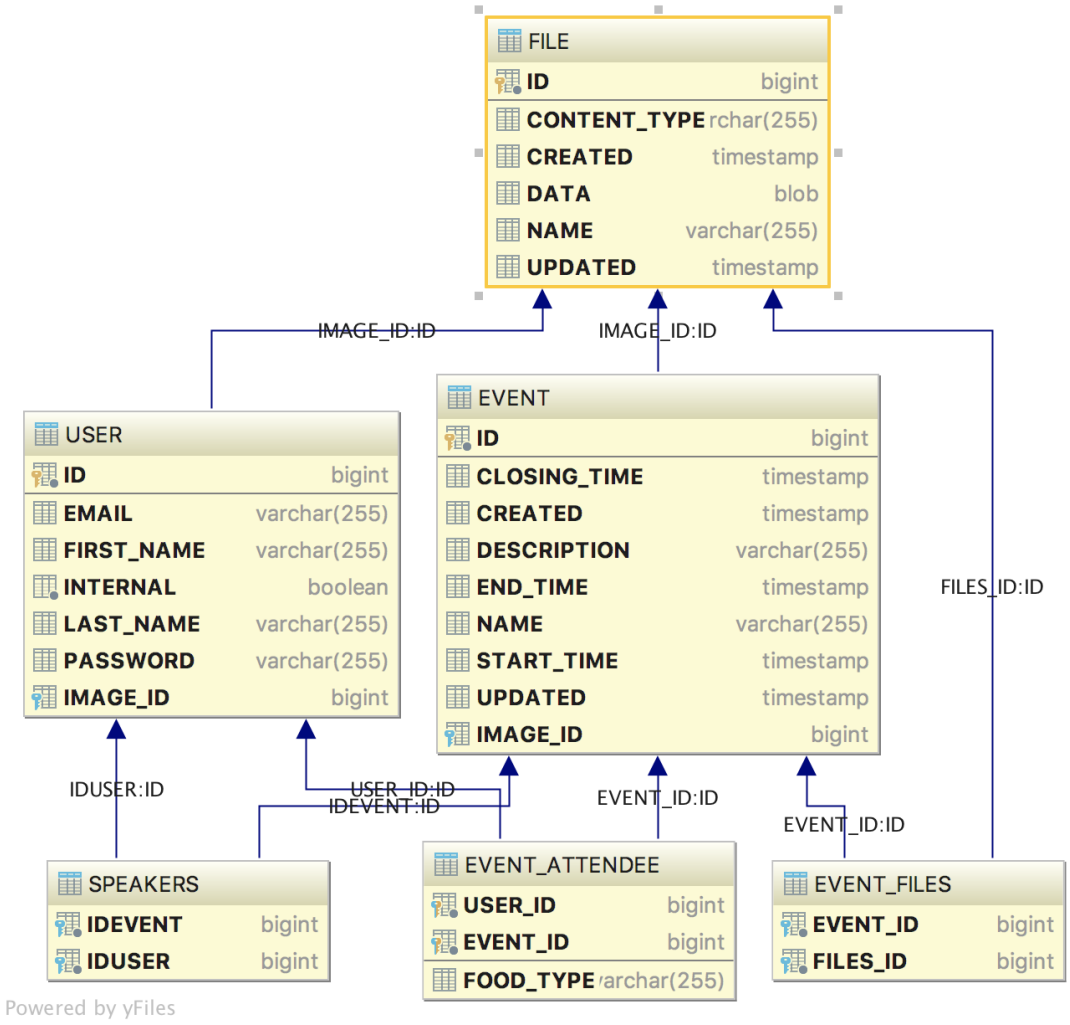
Simpler Mailservice, welcher eine API zum Versenden von Mails bietet. Diese API wird durch ein statische Token geschützt, es werden ausschließlich interne (von anderen Microservices) Anfragen entgegengenommen.

2.2.5 Scheduler

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

2.3 Datenbank

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



3 Technology

3.1 Frontend

Nach dem Start mit Angular 2 merkten wir schnell, das 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.

3.2 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.

3.2.1 Autentifizierung 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)

4 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.

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/