Projekt «Event Verwaltung»
Gruppe: Frauen
Datum: 14.01.2018

Inhaltsverzeichnis

Verwendete Open Source Frameworks

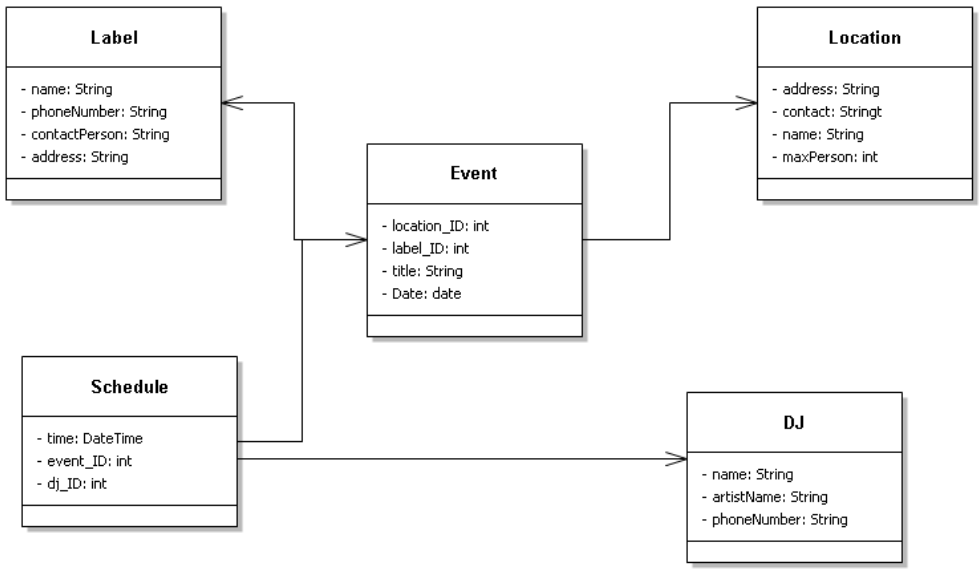
Wir haben in diesem Projekt keine anderen, als die vorgeschlagenen Frameworks verwendet. Folgende Frameworks sind im Einsatz:

- Maven
- JPA
- Spring Boot

Als Webserver wurde der embedded Tomcat vom Spring Boot verwendet. Auch bei der Datenbank haben wir auf die embedded Datenbank vom Spring Boot Framework zurückgegriffen. Für die Containerisierung der Microservices haben wir Docker verwendet.

Geplantes Domänenmodel

Das geplante Domänenmodel stimmt nicht mehr mit dem umgesetzten überein. Wir haben die Entität „Label“ weggelassen, da wir sie für unsere Eventverwaltung nicht zwingend benötigen und wir uns auf die restlichen Entitäten konzentrieren wollten.



Umgesetze Microservices

Wir haben unser Domänenmodell in folgende drei Microservices aufgeteilt:

1) **Event Service** (Entitäten: Event und Schedule)
   Da die Existenz eines Schedules ohne Event keinen Sinn macht, haben wir die beiden Entitäten zu einem Microservice zusammengefasst.
2) **DJ Service** (Entität: DJ)
   Für die DJ- Verwaltung haben wir uns für einen eigenständigen Microservice entschieden, da ein DJ auch ohne Event existieren kann. Ausserdem können so die DJs unabhängig von den Events verwaltet werden.
3) **Location Service** (Entität: Location)
   Eine Location könnte auch von anderen Anwendungen verwendet werden, deshalb ist dies für uns ein klarer Fall für einen eigenständigen Service. (Eine Location kann auch ohne Event existieren)

Aufgetauchte Schwierigkeiten

Problem 1: Die Felder für die jeweiligen Fremdschlüssel waren beim Erfassen im HAL-Browser readonly.

Problem 2: Wenn wir die erfassten DJS unter http://localhost:8080/djs aufrufen wollten bekamen wir einen http error 404- Not found zurück.

Problem 3: Wenn wir die Locations und die dazugehörigen Events anschauen wollten unter http://locahost:8080/eventLocation wurden zwar die Locations angezeigt, aber die Liste der Events war Leer:


```
{
  "_embedded" : {
    "eventLocations" : [ {
      "address" : "Testadresse 20",
      "contact" : "Tester",
      "name" : "Testlocation",
      "maxPerson" : 15,
      "events" : [ ],
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/eventLocation/{id}",
          "templated" : true
        },
        "eventLocation" : {
          "href" : "http://localhost:8080/eventLocation/{id}",
          "templated" : true
        }
      }
    }, {
```


Es waren aber Events vorhanden, welche die angezeigte Location referenziert hatten:


```
{
  "_embedded": {
    "events": [
      {
        "locationId": "402881a460f62a8b0160f649d9fd0000",
        "title": "testevent",
        "date": "2017-01-08T00:00:00.000+0000",
        "_links": {
          "self": {
            "href": "http://localhost:4444/events/402881a460f647aa0160f64bc6320000"
          },
          "event": {
            "href": "http://localhost:4444/events/402881a460f647aa0160f64bc6320000"
          }
        }
      },
```



```
    "locations": [
      {
        "address": "Testadresse 20",
        "contact": "Tester",
        "name": "Testlocation",
        "maxPerson": 15,
        "_links": {
          "self": {
            "href": "http://localhost:3333/locations/402881a460f62a8b0160f649d9fd0000"
          },
          "location": {
            "href": "http://localhost:3333/locations/402881a460f62a8b0160f649d9fd0000"
          }
        }
      },
```

Verwendete Lösungen

Lösung 1: Die Getter- und Setter- Namen der Fremdschlüsselattribute stimmten nicht mit den Namen der Attribute überein. Sie beinhalteten das Wort „Id“ nicht. Als wir dies korrigiert hatten, funktionierte es dann aber.

Lösung 2: Zuerst haben wir im application.yaml file des frontend-services nachgeschaut, ob wir den Service überhaupt eingetragen haben. Da er eingetragen war haben wir im Log auf der Konsole beim starten des frontend-services nachgeschaut, ob das Mapping des DJ-Services ausgeführt wird:

```
: Initializing Spring FrameworkServlet 'dispatcherServlet'
: FrameworkServlet 'dispatcherServlet': initialization started
: FrameworkServlet 'dispatcherServlet': initialization completed in 60 ms
: Mapped URL path [/locations/**] onto handler of type [class org.springframework.cloud.netflix.zuul.web.ZuulController]
: Mapped URL path [/diskjokeys/**] onto handler of type [class org.springframework.cloud.netflix.zuul.web.ZuulController]
: Mapped URL path [/events/**] onto handler of type [class org.springframework.cloud.netflix.zuul.web.ZuulController]
: Mapped URL path [/eventLocation/**] onto handler of type [class org.springframework.cloud.netflix.zuul.web.ZuulController]
: Mapped URL path [/dj-service/**] onto handler of type [class org.springframework.cloud.netflix.zuul.web.ZuulController]
: Mapped URL path [/location-service/**] onto handler of type [class org.springframework.cloud.netflix.zuul.web.ZuulController]
: Mapped URL path [/event-service/**] onto handler of type [class org.springframework.cloud.netflix.zuul.web.ZuulController]
: Mapped URL path [/event-location-service/**] onto handler of type [class org.springframework.cloud.netflix.zuul.web.ZuulController
```

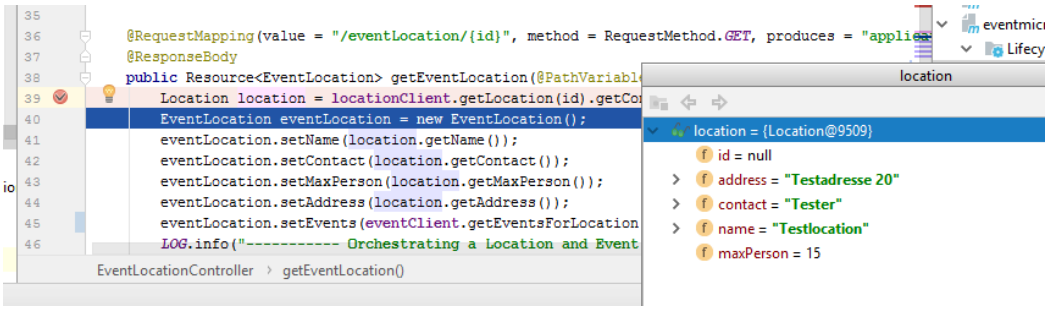
Wie man dem Logausschnitt entnehmen kann, sah hier alles gut aus. Als nächstes haben wir die application.yaml Datei des dj-services mit einem funktionierenden verglichen. Diese waren identisch bis auf den Namen. Beim Erfassen der DJs unter dem Port localhost:2222 ist uns dann schliesslich aufgefallen, dass die URL dort dJs hiess und nicht djs. Wir haben dann in der application.yaml Datei des frontend-service die Route angepasst:

Vorher:	Nachher:

`dj-service:`
`  path: /djs/**`
`  serviceId: dj-service`
`  stripPrefix: false`
	`dj-service:`
	`  path: /dJs/**`
	`  serviceId: dj-service`
	`  stripPrefix: false`

Nach dieser Anpassung funktionierte es dann.

Lösung 3: Wir haben herausgefunden, dass die Id, der Location nicht gesetzt wurde beim Laden. Nachdem wir den Parameter aus der Methode zum Laden der dazugehörigen Events nahmen, wurden auch die Events zu den Locations angezeigt.




```
{
  "address" : "Testadresse 20",
  "contact" : "Tester",
  "name" : "Testlocation",
  "maxPerson" : 15,
  "events" : [ {
    "id" : null,
    "locationId" : "402881a460f62a8b0160f649d9fd0000",
    "title" : "testevent",
    "date" : "2017-01-08T00:00:00.000+0000"
  } ]
}
```

Stand des Projektes

Es wurden alle Models sowie Repositories zu den drei Microservices erstellt. Ausserdem haben wir noch einen vierten Service implementiert, in welchem wir zu jeder Location die dazugehörigen Events ermitteln.

Installations- und Betriebsanleitung

Anforderungen

- IDE: Eclipse oder IntelliJ
- Java 8
- Maven
- Docker

Anleitung

1. Projekt clonen:
    a. Git Repository: https://github.com/schne5/frauen.parent
2. Maven „clean install“ ausführen auf dem parent Projekt
3. Über das Terminal die Dockercontainer starten
    a. Terminal starten
    b. In das Projektverzeichnis wechseln
    c. „docker-compose up“ Befehl eingeben und ausführen