Business Trips - Application

Aspects of a Business Trips App, Backend

Beschreibung
In diesem Dokument geht es um die Umsetzung von Konzepten für eine Business Trip App. Die Technologien, welche eingeführt werden, sind weiter unten beschrieben.
Folgende Anforderungen muss die App erfüllen:

- Die eigenen Funktionen aus dem Frontend werden umgesetzt.
- Architektur: Trennung der View von der Logik React
    - Backend: Java Spring Boot und Spring Data JPA / REST

Frontend: Web App mit React Zeitbedarf:	12 Lektionen
Hilfsmittel:	Java, Spring Boot, React
Methode/Sozialform:	PA (Backend, Frontend mit React)
Kompetenzen:	-
Abgaben	1. Frontend React 2. Backend Java Spring Boot/Spring REST

Legende:

Partnerarbeit Dokumentation Code

Inhaltsverzeichnis

1 START 3
1.1 ENTITYS FÜR BUSINESS-TRIP 3
1.1.1 Beispiel Datenmodell 3
1.1.2 Aufgabe Business-Trip-Entity 4
1.2 REPOSITORY FÜR BUSINESSTRIP 5
1.2.1 Aufgabe BusinessTripRepository 6
1.3 TEST-DATEN 6
1.3.1 Aufgabe Testdaten 7
1.4 AUFGABE – PRÜFT DIE GANZE BUSINESS-TRIP APP 7
1.5 AUFGABE – FÜHRT DIE TESTS MIT POSTMAN DURCH 7

1 Start

Übernehme die letzte Version der biztrips React App mit Spring Web/ React
Folgenden Stack werden wir benutzen:

1. Java, Maven als den Build Manager
2. Spring Boot mit Spring Web / Restful

Die Aufgabe verwendet dabei folgende Konzepte kennen lernen:

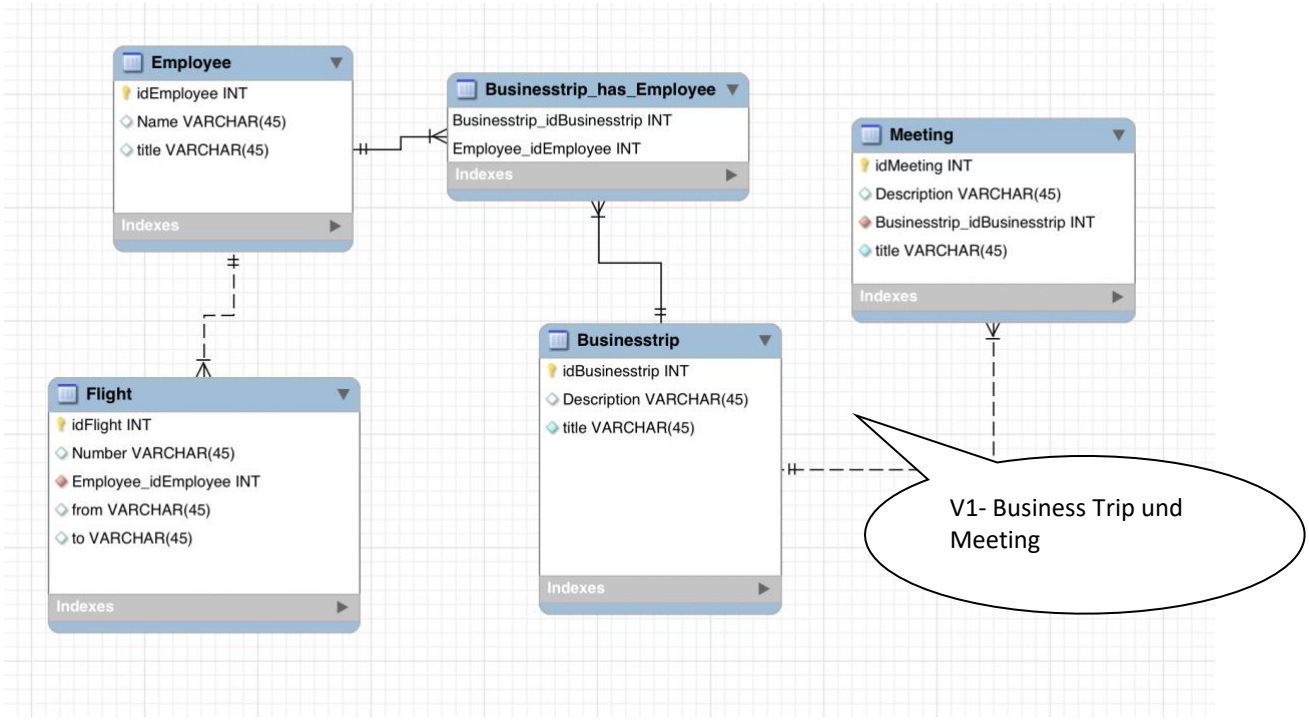
- Spring Web – RestController / View React
- Datenobjekte / Listen
- Zweischichtige Architektur / Repositories von Spring
- Dependency Injection mit @Autowired
- JPA Entities

1.1 Entitys für Business-Trip

Für unsere Entitys, also die fachlichen Klassen unserer App, brauchen wir Spring Data JPA, hier ein Beispiel: https://spring.io/guides/gs/accessing-data-jpa/ dies erleichtert uns das Model in die Datenbank zu speichern!

1.1.1 Beispiel Datenmodell

Von der obersten Schicht, der View gehen wir nun zuunterst zur Datenbank, *kopfüber*!
Überlegen wir uns kurz, anhand des Datenmodell – Entwurfs welche Komponenten wir brauchen! Wir werden hier einfach starten und uns zunächst um die zwei Entitäten Businesstrip und Meeting kümmern! Den Rest machen wir dann später!



https://spring.io/guides/gs/accessing-data-jpa/
Das Modell: hier am Beispiel einer BusinessTrip - Entity:

```java
@Entity
public class BusinessTrip implements Serializable {

	private static final long serialVersionUID = 6702756380838250
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	private String title;
	private String description;
	private LocalDateTime startTrip;
	private LocalDateTime endTrip;

	@OneToMany(mappedBy = "businessTrip")
	private List<Meeting> meetings;

	// constructors / getter and setter omitted
}
```

Indicates that the persistence provider must assign primary keys for the entity using a database identity column.

Beziehungen im Datenmodell definiert

```java
@Entity
public class Meeting implements Serializable {

	private static final long serialVersionUID = 6705527563808382509L;
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	private String title;
	private String description;

	@ManyToOne
	@JoinColumn(name = "business_trip_idfs")
	private BusinessTrip businessTrip;
```

Beziehungen im Datenmodell definiert (Beidseitig)

1.1.2 Aufgabe Business-Trip-Entity

a) Mach euch mit dem code vertraut, code lesen und verstehen können!

1.2 Repository für BusinessTrip

Bei den Repositories oder im Java Jargon DAO's, also die Datenbank – nahe Schicht, nimmt uns Spring sehr viel Arbeit ab, denn die Klassen CrudRepository oder JpaRepository sind magically schon vorbereitet siehe https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories

4.1. Core concepts

The central interface in the Spring Data repository abstraction is `Repository`. It takes the domain class to manage as well as the ID type of the domain class as type arguments. This interface acts primarily as a marker interface to capture the types to work with and to help you to discover interfaces that extend this one. The `CrudRepository` provides sophisticated CRUD functionality for the entity class that is being managed.

Example 3. `CrudRepository` *interface*

```
public interface CrudRepository<T, ID extends Serializable>
  extends Repository<T, ID> {

  <S extends T> S save(S entity);      (1)

  Optional<T> findById(ID primaryKey); (2)

  Iterable<T> findAll();               (3)

  long count();                        (4)

  void delete(T entity);               (5)

  boolean existsById(ID primaryKey);   (6)

  // … more functionality omitted.
}
```

1. Saves the given entity.
2. Returns the entity identified by the given ID.
3. Returns all entities.
4. Returns the number of entities.
5. Deletes the given entity.
6. Indicates whether an entity with the given ID exists.

Dank den neuen Default Methoden in Interfaces, ist das Interface CrudRepository schon vorbereitet mit eben den CRUD-Methoden (Create, Read, Update und Delete), wie praktisch, dass man sich nicht mehr kümmern muss und die Methoden schon magisch da sind! Die Methoden welche zusätzlich verwendet werden können wir im interface hinzufügen, siehe das Beispiel vom CustomerRepository:

```
public interface CustomerRepository extends CrudRepository<Customer, Long> {

  List<Customer> findByLastName(String lastName);
}
```

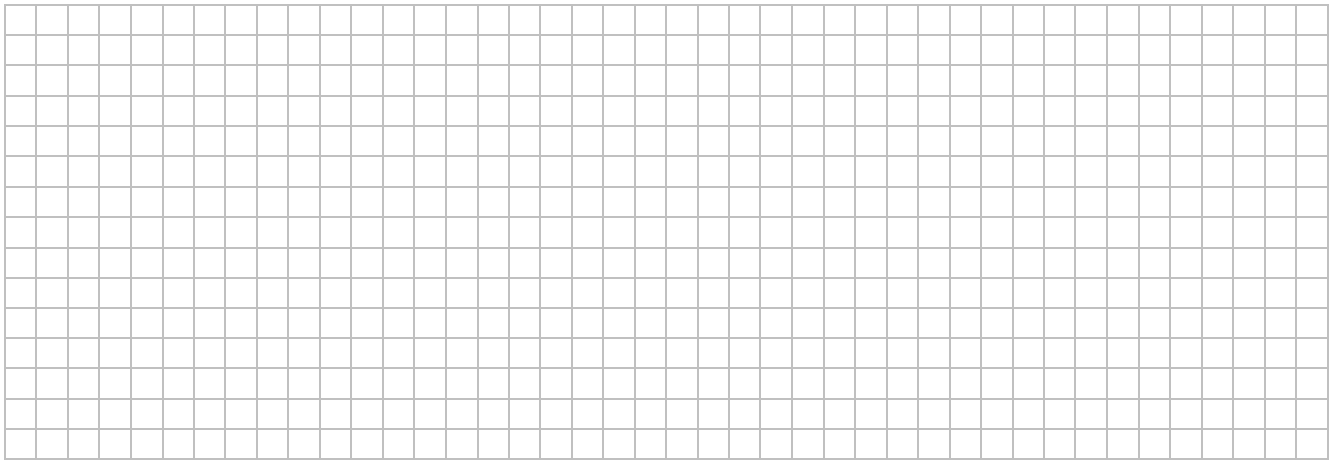
Hier fügen wir die zusätzlichen Methoden welche wir brauchen, wir halten uns an die Namenskonvention und Spring erstellt uns magisch die SQL-Abfragen im Hintergrund, wow!

CrudRepository Interface

CrudRepository is an interface and extends Spring data Repository interface. CrudRepository provides generic CRUD operation on a repository for a specific type. It has generic methods for CRUD operation. To use CrudRepository we have to create our interface and extend CrudRepository. We need not to implement our interface, **its implementation will be created automatically at runtime**. Find some of CrudRepository methods.

1.2.1 Aufgabe BusinessTripRepository

a) Code lesen und verstehen: BusinessTripRepository ähnlich dem CustomerRepository Beispiel!



1.3 Test-Daten

Um Testdaten einzufügen kann im Spring – Ecosystem ein CommandLineRunner-Bean-Methode erstellt werden welches beim Starten aufgerufen wird! Dies kann direkt in der Application Class erfolgen oder in einer separaten Initializer Class.
Für weitere Infos https://spring.io/guides/gs/accessing-data-jpa/

Bean in der Application Class:

```
@Bean
  public CommandLineRunner demo(CustomerRepository repository) {
        return (args) -> {
              // save a couple of BusinessTrips
              BusinessTrip bt01 = new BusinessTrip(1L, "BT01", "San Francisco World Trade Center on new Server/IOT/Client ",    LocalDateTime.of(2021, 2, 13, 9, 00),
LocalDateTime.of(2021, 2, 15, 16, 56));
              BusinessTrip bt02 = new BusinessTrip(2L, "BT02", "Santa Clara Halley on new Server/IOT/Client", LocalDateTime.of(2021, 6, 23, 9, 00),
LocalDateTime.of(2021, 6, 27, 16, 56));
              BusinessTrip bt03 = new BusinessTrip(3L, "BT03", "San Cose City Halley on Docker/IOT/Client", LocalDateTime.of(2021, 12, 13, 9, 00),
LocalDateTime.of(2021, 12, 15, 16, 56));

              businessTripRepository.save(bt01);
              businessTripRepository.save(bt02);
              businessTripRepository.save(bt03);

              // save a couple of meetings

              Meeting one = new Meeting(1L, "One Conference", "Keynote on One Conference", bt01);
              Meeting zero = new Meeting(2L, "Zero Conference", "Workshop Zero on One Conference", bt01);
Meeting handsOn = new Meeting(3L, "One Conference", "HandsOn on One Conference", bt02);

              Meeting workOn = new Meeting(4L, "One Conference", "Keynote on One Conference", bt02);
              Meeting stayTuned = new Meeting(5L, "One Conference", "Keynote on One Conference", bt03);
```

```
                meetingRepository.save(one);
                meetingRepository.save(zero);
                meetingRepository.save(handsOn);
                meetingRepository.save(workOn);
                meetingRepository.save(stayTuned);

                        }
    }
```

1.3.1 Aufgabe Testdaten

Erstellen Sie genügend realistische Testdaten, um das Backend zu testen (Min. 10 **realistische** Datensätze pro Entity)

b) Beschreibung der Testdaten in der Dokumentation

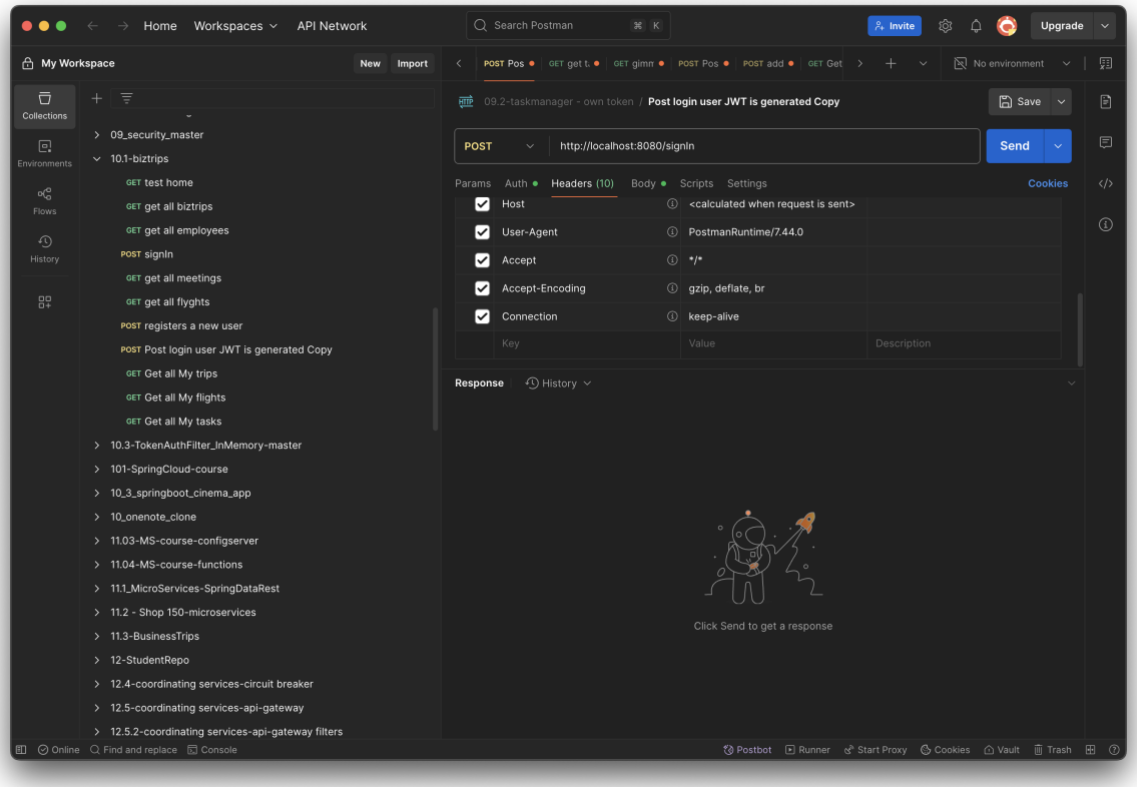
1.4 Aufgabe – Prüft die ganze Business-Trip App

a) Prüft die Entitys und deren Beziehungen kritisch
b) Prüft die Package Struktur, ändert Sie je nach Anforderungen
c) bereinigt den code und verwendet Spring Boot v3 :

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.6</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

1.5 Aufgabe – Führt die Tests mit Postman durch

d) Prüft alle Endpoints mit Postman


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Collections
Environments
Flows
History
09_security_master
10.1-biztrips
GET test home
GET get all biztrips
GET get all employees
POST signIn
GET get all meetings
GET get all flyghts
POST registers a new user
POST Post login user JWT is generated Copy
GET Get all My trips
GET Get all My flights
GET Get all My tasks
10.3-TokenAuthFilter_InMemory-master
101-SpringCloud-course
10_3_springboot_cinema_app
10_onenote_clone
11.03-MS-course-configserver
11.04-MS-course-functions
11.1_MicroServices-SpringDataRest
11.2 - Shop 150-microservices
11.3-BusinessTrips
12-StudentRepo
12.4-coordinating services-circuit breaker
12.5-coordinating services-api-gateway
12.5.2-coordinating services-api-gateway filters
09.2-taskmanager - own token / Post login user JWT is generated Copy
Save
POST
http://localhost:8080/signIn
Send
Params
Auth
Headers (10)
Body
Scripts
Settings
Cookies
Host
<calculated when request is sent>
User-Agent
PostmanRuntime/7.44.0
Accept
/
Accept-Encoding
gzip, deflate, br
Connection
keep-alive
Key
Value
Description
Response
History
Click Send to get a response
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash