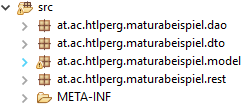
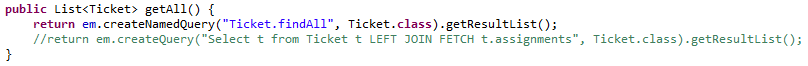
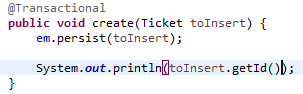
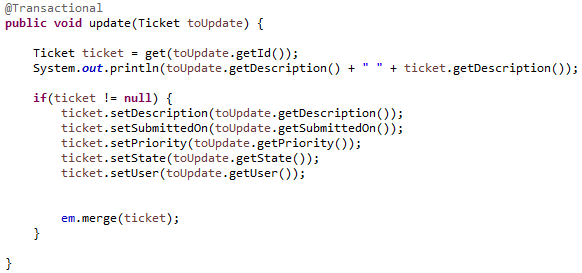
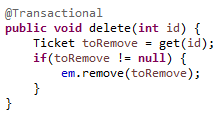
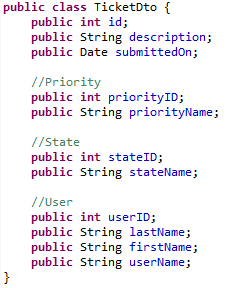
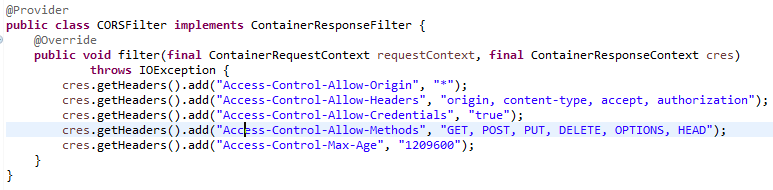
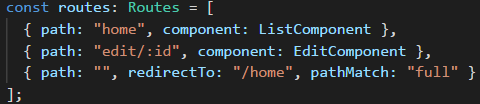
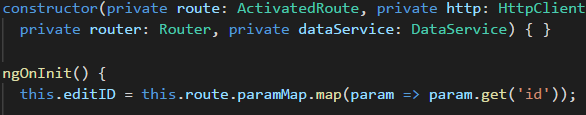
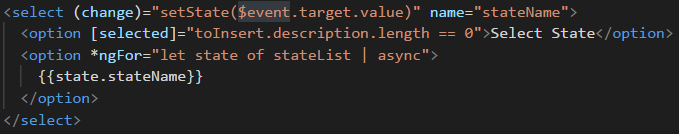
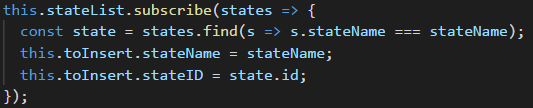
**JPA – Backend**

* **MySQL-Server starten** – mysqld.cmd
* **Wildfly** starten
* **JDBC Treiber** am Wildfly deployen
* **Schema** in **MySQL Workbench** erstellen
* **Datasource** im Wildfly anlegen
  + Localhost:9990
  + Configuration -> Start
  + Subsystems -> Datasources -> Non-XA -> Add
  + MySQL -> JNDI Name vergeben -> Detected Driver -> Nicht den Fabric/h2 auswählen!
  + Wichtig: Bei Connection URL statt mysqldb EUREN Schemanamen eintragen
  + Username: root, Password ist leer; Test Connection unbedingt ausführen
  + JNDI Name kopieren/merken
* **Dynamic Webproject** anlegen
  + Target Runtime = WildFly 11.0 Runtime; den Rest lassen
  + Bei web.xml Deployment Descriptor Hackerl setzen  
    
* Rechtsklick aufs Projekt -> Convert to **JPA Project**
  + JNDI Name in persistence.xml unter Connection eintragen!!! 
  + Package Model anlegen -> Rechtsklick -> New -> Other -> JPA-Entities from Table
    - Gewünschte Tabellen auswählen
    - Key generator = identity
  + In den generierten Klassen können im Normalfall OneToMany Beziehungen einfach gelöscht werden, da diese LazyInit Exceptions verursachen; Man kann diese jedoch auch im Dao mittels einer JPQL Query beseitigen 🡪 dann nicht löschen
* **Dao** – ein Dao pro Entität, wenn man diese für Rest benötigt
  + Named und Scope Annotations festlegen   
    
  + EntityManager als PersistenceContext festlegen  
    
  + GetAll  
    
  + GetOne  
    
  + Create 🡪 Transactional Annotation wichtig!  
    
  + Update 🡪 Transactional  
      
    Objekt vorher vom EntityManager holen, um das ManagedObject zu erhalten und bei diesem dann die neuen Werte setzen – nicht das übergebene Mergen!
  + Delete 🡪 Transactional  
    
* **DTO**
  + Um ManyToOne Beziehung nach außen besser darzustellen
  + Client muss z.B. dann nur die ID des abhängigen Elements kennen und nicht das genaue Objekt  
      
    Attribute der abhängigen Objekte (Priority, State und User) als public Attribute in DTO einfügen
* **Rest**
  + Configure -> Add JAX-RS Support
  + Rechtsklick in Package Rest -> New JAX-RS Resource
    - Target Entities auswählen, die nach außen repräsentiert werden sollen
    - XML entfernen und Hakerl bei den gewünschten Operationen setzen
  + Benötigte Daos injecten
  + Nach außen nur DTOs schicken und von außen nur DTOs entgegennehmen  
    
  + DTO erzeugen/aus DTO Objekt erzeugen

|  |  |
| --- | --- |
|  |  |

* + Create
    - Dto in richtiges Objekt mittels der obigen Methoden umwandeln
    - Wenn null 🡪 Bad-Request
    - Ansonsten den langen, auskommentierten Header verwenden
  + ListAll
    - Returnwert: List<*EntityNameEinfügen*Dto>  
      
    - Alle Objekte in Dto umwandeln
  + Update
    - Dto in richtiges Objekt umwandeln und überprüfen, ob nicht null und in DB vorhanden 🡪 OK
    - Sonst NOT\_FOUND
  + Delete
    - Überprüfen, ob in DB
    - OK oder NOT\_FOUND
* CORS
  + Filter am Server, der die Requests intercepted  
    Unbedingt @Provider Annotation einfügen!  
    

**Angular Client**

* Ng new projektname –routing
* Ng generate component name
* Ng generate interface IName
* Ng generate service Data
* Für Two-Way Binding FormsModule aus @angular/forms in app.module.ts
* Services in app.module.ts in providers einfügen
* Routing
  + Routen definieren  
    
  + Parameter aus Route 🡪 Activated Route und paramMap.map 🡪 Observable!  
    
  + Routerevent manuell auslösen  
    
  + Routerevent über <a> auslösen  
    
* Daten aus Observable in Tabelle anzeigen  
    
  Mittels „async“ kann man die Daten ohne Subscribe direkt anzeigen  
  Zugriff auf Attribute mit {{}}
* Dropdown  
    
  $event ist das ausgelöste Event  
  So kann man mittels event.preventDefault() im Code z.B. einen Pagereload vermeiden
* Element aus Observable<irgendwos[]> finden  
  
* Array von Objekten von API abfragen
  + Angular httpclient
    - In app.module.ts HttpClientModule aus@angular/common/http
    - In Komponente HttpClient aus @angular/common/http
  + 
* Objekt einfügen/updaten
  + Einfügen = POST 🡪 KEINE ID ANGEBEN 🡪 am besten ID = undefined im Objekt  
     in toInsert  
    
  + Update = PUT  
    
* Wichtig ist, für alle Entities Interfaces zu erstellen, um Types nutzen zu können
* Bei POST, PUT und DELETE muss bei den Observables subscribed werden!!!