Script: NVS 4

Table of Contents

Version: 1.6

1. General 2

1.1. Configure Data Source & and Drivers

Start DerbyDB

Start DB:

```
demoTest101/db$ /opt/db-derby-10.14.2.0-bin/bin/startNetworkServer -noSecurityManager
```

Configure in IJ

Option	Input
Driver	Apache Derby (Remote)
Host	localhost

Option	Input
Port	1527
User	app
Password	app
Database	db
URL	jdbc:derby://localhost:1527/db;create=true

Good Source: https://www.tutorialspoint.com/intellij_idea/index.htm

1.2. Project Structure

Source/main/java/at/htl/[ProjectName]/

business/
model/
rest/

Source/main/java/at/htl/[ProjectName]/

control/
entity/
boundary/

Source/main/resources/

Files.csv
application.properties (Quarkus)

META-INF/ (WildFly)

persistence.xml

- The source code is usually in 3 subdirectory of the main folder **at.htl.project_Name** Folder. The subdirectory are **business, model, rest**.
- In the **business folder** is the **InitBean.java** which contains the init method for the Application server.
- In the **model folder** are the **Entities**.
- In the **rest folder** is the **Endpoints.java** and the **RestConfig.java** which configures the rest service.
- For testing the REST service a **request.http** can be created this file should be placed in the **requests folder** which is a subdirectory of the project's root directory.
- The **resources folder** which is also a subdirectory of the project's root directory is for resources. Like: **csv files** or the folder **META-INF** which contains the **persistance.xml**.

1.2.1. Repository

Example for a Repository

```
@Transactional
public class CourseRepository {

    @PersistenceContext
    EntityManager em;
}
```

1.2.2. Entity

Example Person

```
package at.htl.person.model;
import javax.persistence.*;

@Entity
//@Entity(name = "Person")
public class Person {
    @Transient
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd.MM.yyyy");

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "customer_name")
    private String name;
}
```

XML-Root

For xml we have to declare the entity as:

Example for Entity with XML+

```
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Vehicle {}
```

1.2.3. Rest Config

Rest Config File

```
package at.htl.vehicle.rest;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("api")
public class RestConfig extends Application {

}
```

1.2.4. InitBean (Read data from csv)

Good Source:

https://stuetzpunkt.wordpress.com/2016/12/28/how-to-access-file-in-resources-folder-javaee/

Example for read csv in InitBean

```
private void init(
    @Observes
    @Initialized(ApplicationScoped.class) Object object) {
    readCsv(FILE_NAME);
}

private void readCsv(String fileName) {
    URL url = Thread.currentThread().getContextClassLoader()
            .getResource(fileName);
    try (Stream<String> stream = Files.lines(Paths.get(url.getPath())
            , StandardCharsets.UTF_8)) {
        stream
                .skip(1)
                ...
                .forEach(em::merge);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

1.2.5. Request.http

Examples for a POST in request.http

```
POST http://localhost:8080/person/api/person
Content-Type: application/json

[
  {
    "dob": "2001-10-07",
    "name": "Chiara"
  },
  {
    "dob": "2002-03-23",
    "name": "Christoph"
  }
]
```

Examples for GET in request.http

```
### Get All as XML
GET http://localhost:8080/person/api/person/demo
Accept: application/xml

### Get Susi
GET http://localhost:8080/person/api/person?name=Susi
```

---

2. Quarkus

2.1. Create a project

Needed for a basic setup:

- Web:
  - RESTEasy Jackson
  - REST Client Jackson
  - SmallRye OpenAPI
- Data:
  - Hibernate ORM
  - Hibernate ORM with Panache
- Serialization:
  - RESTEasy Jackson
  - REST Client Jackson

Q: Getting Started

2.2. Start Quarkus

2.3. Persistance

- Quarkus - Datasources
- Quarkus - Simplified Hibernate ORM with Panache

2.4. Networking

- Quarkus - Writing JSON REST Services
- Quarkus - Using OpenAPI and Swagger UI
- Quarkus - Using WebSockets

2.5. Quarkus Cheat-Sheet

- Home of Quarkus Cheat-Sheet

2.6. Read CSV File

- How to access a file in the resources-Folder (JakartaEE)

---

3. Hints

3.1. Convert String to LocalDate

Example Problem

```
DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd-MM-yy");
String dateString = "03-07-88"; // 3rd of July 1988

LocalDate date = LocalDate.parse(dateString, dtf);

System.out.println(date); // --> 2088-07-03
```

When converting a String with a two-digit-year to a LocalDate variable, the base of the convertion is 2000 so you get 2088 as result.

Example Solution

```
To prevent this, you can subtract 100 years in the DateTimeFormatter-Object

DateTimeFormatter dtf = new DateTimeFormatterBuilder()
      .appendPattern("dd-MM-")
      .appendValueReduced(ChronoField.YEAR, 2, 2, 1900)
      .toFormatter();

String dateString = "03-07-88"; // 3rd of July 1988

LocalDate date = LocalDate.parse(dateString, dtf);

System.out.println(date); // --> 1988-07-03
```

Now the correct date is displayed

Source:

- https://stackoverflow.com/a/38354449
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/format/DateTimeFormatter.html

---

4. JPA

JPA is a concept that can be implemented like a interface, the current reference implementation is EclipseLink.

The majority of imports is located in the fallowing package:
Source Package: import javax.persistence.*;

Table 1. ***Common JPA Annotations***

Annotation	**Description**
@Entity	makes a class a entity
@Entity(name = "Person")	defines the table name of the entity

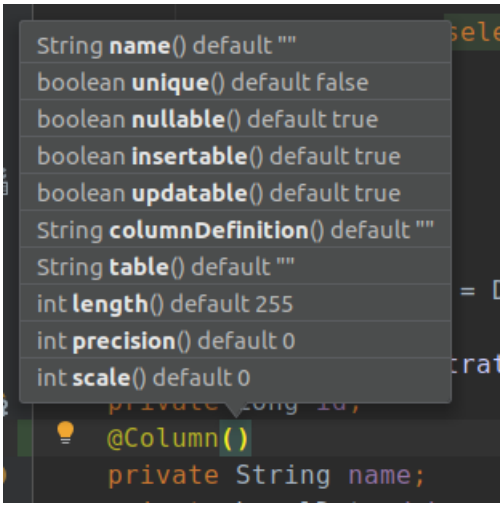
Annotation	Description
@Id	defines the Pk of a table entity
@GeneratedValue(strategy = GenerationType.IDENTITY)	defines a auto generated key
	options for fields / columns
@Transient	defines fields that should not be part of the entity
@Enumerated(EnumType.STRING) private EmploymentType empType;	defines what kind of datatype of a enum get stored in the db (by default int)

Table 2. ***JPA Relationship Annotations***

Annotation	Description
/* Bestellung */ @OneToMany(mappedBy="bestellung", cascade = CascadeType.Persist, orphanRemoval=true) private List<Bestellungsposition> bestellungspositionListe;	delete dependent children, when the parent is going to be deleted (child-entities are orphans (=Waisen) then)
/* Bestelposition */ @ManyToOne @JoinColumn(name = "bestellung_id") private Bestellung bestellung;	the inverse part of the relationship

/* Person */
@ManyToOne()
@JoinColumns({
@JoinColumn(name = "Address_No"),
@JoinColumn(name = "ssn")
})
private Address address;
 	
/* Address */
@OneToMany(mappedBy = "id.person", cascade = CascadeType.PERSIST)
private List<Address> addresses = new ArrayList<>();
	when address has a composition key

/* Person */ @OneToOne @JoinColumn(unique = true) private Address address;	defines a OneToOne relationship and adds a Fk to the Address in the Person
@OneToOne(cascade = {CascadeType.PERSIST, CascadeType.REMOVE}) private Address address;	the Address would get added the same moment as the parent object and removed

4.1. ManyToMany Relationship

There are two ways to make a many to many relationship in JPA. You can decide between a auto generate

association table or you can make one yourself. The auto generated on has a down side due to a leg of customizability so if you want to ahv custom fields you have to create a new @Entity class and a new @Embaddable class for the Id.

4.1.1. Auto Generated Table

Example Auto Generated Association Table

```
@Entity
class Student {

    @Id
    Long id;

    @ManyToMany
    @JoinTable(
        name = "course_like",
        joinColumns = @JoinColumn(name = "student_id"),
        inverseJoinColumns = @JoinColumn(name = "course_id"))
    Set<Course> likedCourses;
}

@Entity
class Course {

    @Id
    Long id;

    @ManyToMany(mappedBy = "likedCourses")
    Set<Student> likes;
}
```

The new association is in this case owned by the student.

4.1.2. Composite Key

Example Composite Key

```
@Embeddable
class CourseRatingKey implements Serializable {

    @Column(name = "student_id")
    Long studentId;

    @Column(name = "course_id")
    Long courseId;

    // standard constructors, getters, and setters
    // hashcode and equals implementation
}
```

Example Using a Composite Key

```
@Entity
class CourseRating {

    @EmbeddedId             //Could be a normal @Id
    CourseRatingKey id;     //Long id;

    @ManyToOne
    @MapsId("student_id")   //This would then bin unnecessary
    @JoinColumn(name = "student_id")
    Student student;

    @ManyToOne
    @MapsId("course_id")    //This would then bin unnecessary
    @JoinColumn(name = "course_id")
    Course course;

    int rating;
}

class Student {
    @OneToMany(mappedBy = "student")
    Set<CourseRating> ratings;
}

class Course {
    @OneToMany(mappedBy = "course")
    Set<CourseRating> ratings;
}
```

4.2. Sequence as Key

Example Sequence as Primary Key

```
@Entity
@Table(name = "XY_MY_OBJECT")
@SequenceGenerator(name="xy_my_object_seq", initialValue=500, allocationSize=1)
public class MyObject {

    @GeneratedValue(strategy= GenerationType.SEQUENCE, generator="xy_my_object_seq")
    @Id Long id;
}
```

---

5. JPQL

Java Persistance Query Language

5.1. CreateQuery

Example for More Advanced Example

```
public void getStuff(){
    System.out.println("\n JPA_1 | Query2:");
    Query query2 = em.createQuery(
            "SELECT NEW demo.AwesomePeopleDetail(p.isAwesome, count(p.SSN)) from Person p group by p.isAwesome");
    List<AwesomePeopleDetail> result2 = query2.getResultList();
    for (AwesomePeopleDetail apc : result2) {
        System.out.println(apc.isAwesome() + ": " + apc.getCount());
    }
}
```

5.2. Response Object

Example for Query Response Class

```
public class AwesomePeopleDetail {

    private boolean isAwesome;
    private long count;

    public AwesomePeopleDetail(boolean isAwesome, long count) {
        this.isAwesome = isAwesome;
        this.count = count;
    }
    //region Properties
    ...
    //endregion
}
```

5.3. Response Tuple

Example for saving Response in a Tuple:

Example for a Tuple Response

```
private static void secondQuery(EntityManager em) {
    TypedQuery<Tuple> query = em.createQuery("select o.id, p.firstName || ' ' || p.lastName, a.country
        || ' ' || a.city || ' ' || a.street || ' ' || a.streetNo as name,
        sum(oi.amount * p2.price) as totalCost, sum(oi.amount) as pieces " +
            "from Person p join p.addresses a join Order o on o.customer = p join o.orderItems oi " +
            "join oi.id.product p2 where a.id.addressNo = o.shipmentAddress.id.addressNo group by o, p, a", Tuple.class);
    Tuple result = query.getResultList().get(0);
    var shipment = new OrderShipment((int) result.get(0), (String) result.get(1), (String) result.get(2),
        (BigDecimal) result.get(3), Math.toIntExact((long) result.get(4)));
    printShipmentInfo(shipment);
}
```

5.4. Named Query

Example for NamedQueries

```
@Entity
@NamedQueries({
        @NamedQuery(
                name = "Person.findAll",
                query = "select p from Person p"
        ),
        @NamedQuery(
                name = "Person.findByName",
                query = "select p from Person p where p.name = :NAME"
        )
})
```

Example for a Rest using a NamedQuery

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public Person findByName(@QueryParam("name") String name) {
    return em
    .createNamedQuery("Person.findByName",Person.class)
    .setParameter("NAME", name)
    .getSingleResult();
}
```

5.5. Entity Manager

Example for creating a Entity Manager

Eample for Creating a EntityManager

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("my−persistence−unit");
EntityManager em = emf.createEntityManager();

em.getTransaction().begin();
// perform insert/update/delete/query
em.getTransaction().commit();
// or em.getTransaction().rollback();
em.close();
```

Good Sources:

https://www.tutorialspoint.com/de/jpa/jpa_jpql.htm

---

6. CRUD

- Create: persist entity

```
em.persist(person);
```

- Read: find entity by id

```
Person person = em.find(Person.class, "1234010190");
```

- Update: update entity fields

```
Person person = em.find(Person.class, "1234010190");
person.setName("Jane Doe");
// optional: other operations
em.merge();
//em.getTransaction().commit();
// executes update for the name of the person
```

- Delete: remove entity

```
Person person = em.find(Person.class, "1234010190");
em.remove(person);
// optional: other operations
em.getTransaction().commit();
// executes delete for the person
```

---

7. REST

7.1. HTTP Methods

- Get (Read: all or a specific resource)
- Post (Create or Update: without a specific ID)
- HEAD
- PUT (Create or Update: with a specific ID)
- DELETE (delete a specific resource)
- TRACE
- OPTIONS
- CONNECT

Good Source:
https://wiki.selfhtml.org/wiki/HTTP/Anfragemethoden

7.2. Examples a RestEndpoint

Common Imports for a RestEndpoint

```
import javax.annotation.PostConstruct;
import javax.json.*;
import javax.persistence.*;
import javax.transaction.Transactional;
import javax.ws.rs.*;
import javax.ws.rs.core.*;
import java.net.URI;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.List;
```

Example for a Endpoint

```
@Path("person")
public class PersonEndpoint {

    public PersonEndpoint() {
    }

    @PersistenceContext
    EntityManager em;

    @GET
    @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XM})
    public List<Person> findAll() {
        return em
                .createNamedQuery("Person.findAll", Person.class)
                .getResultList();
    }
```

7.3. POST

Example for a Post

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Transactional
public Response createPerson(
        final @Context UriInfo uriInfo,
        JsonValue jsonValue) {

    if (jsonValue.getValueType() == JsonValue.ValueType.ARRAY) {
        JsonArray jsonArray = jsonValue.asJsonArray();
        for (JsonValue value : jsonArray) {
            String name = value.asJsonObject().getString("name");
            ...
            p = em.merge(p);
        }
    } else {
        System.out.println("Ich bin ein Object");
    }
    return Response.ok().build();
}
```

7.4. Examples for a RestClient

Example for a get in a Java SE client

```
//import javax.ws.rs.* //core or client;

Client client = ClientBuilder.newClient();
WebTarget tut = client.target("http://localhost:8080/restprimer/api/time");

Response response = tut.request(MediaType.TEXT_PLAIN).get();
String payload = response.readEntity(String.class);
System.out.println("Request: " + payload);
```

Example Returning a URI in the Request

```
public Response foo(@Context UriInfo uri){
    URI uri = info.getAbsolutePathBuilder().path("/"
        newCourseType.getId()).build();
    return Response.created(uri).build();
}
```