Script: NVS 4

Version: 1.0

1. General

1.1. Create Project

Create **Maven** Project with Intellij. For Example:

```
    <groupId>at.htl</groupId>
    <artifactId>PersonRest</artifactId>
```

1.2. Configure Data Source & and Drivers

Option	Input
Driver	Apache Derby (Remote)
Host	localhost
Port	1527
User	app
Password	app
Database	db
URL	jdbc:derby://localhost:1527/db

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

1.3. Start DerbyDB

Start DB:

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

1.4. Project Structure

- The source code is usually in 3 subfolders of the main folder **at.htl.project_Name** Folder. The subfolders 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 subfolder of the project's root directory.
- The **resources folder** which is also a subfolder of the project's root directory is for resources. Like: **csv files** or the folder **META-INF** which contains the **persistance.xml**.

1.5. XML

For xml we have to declare the entity as:

Example for Entity with XML

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

@XmlRootElement
public class Vehicle {}
```

1.6. Pom.xml

Pom.xml

```
    <?xml version="1.0" encoding="UTF-8"?>
    <project xmlns="http://maven.apache.org/POM/4.0.0"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>at.htl</groupId>
        <artifactId>vehicle</artifactId>
        <version>1.0-SNAPSHOT</version>
        <packaging>war</packaging>

                <properties>
            <maven.compiler.source>11</maven.compiler.source>
            <maven.compiler.target>11</maven.compiler.target>
        </properties>

        <dependencies>
            <dependency>
                <groupId>jakarta.platform</groupId>
                <artifactId>jakarta.jakartaee-api</artifactId>
                <version>8.0.0</version>
                <scope>provided</scope>
            </dependency>
            <dependency>
                <groupId>jakarta.xml.bind</groupId>
                <artifactId>jakarta.xml.bind-api</artifactId>
                <version>2.3.2</version>
                <scope>provided</scope>
            </dependency>
        </dependencies>

        <build>
            <finalName>vehicle</finalName>
        </build>


    </project>
```

Useful Package sources

```
                            <!-- Useful Sources -->
    <!-- https://mvnrepository.com/artifact/junit/junit -->
    <!-- https://mvnrepository.com/artifact/org.glassfish.jersey.core/jersey-client
-->
    <!-- https://mvnrepository.com/artifact/org.glassfish.jersey.media/jersey-media-
json-processing -->
    <!-- https://mvnrepository.com/artifact/org.glassfish/javax.json -->
    <!-- https://mvnrepository.com/artifact/org.glassfish.jersey.inject/jersey-hk2 -->
    <!-- https://mvnrepository.com/artifact/org.hamcrest/hamcrest -->
```

1.7. Request

Examples for 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"
    }
    ]

    ###

    GET http://localhost:8080/person/api/person/demo
    Accept: application/xml

    ###

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

1.8. 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 {

}
```

1.9. 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();
    }
}
```

2. JPA

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

2.1. Entity

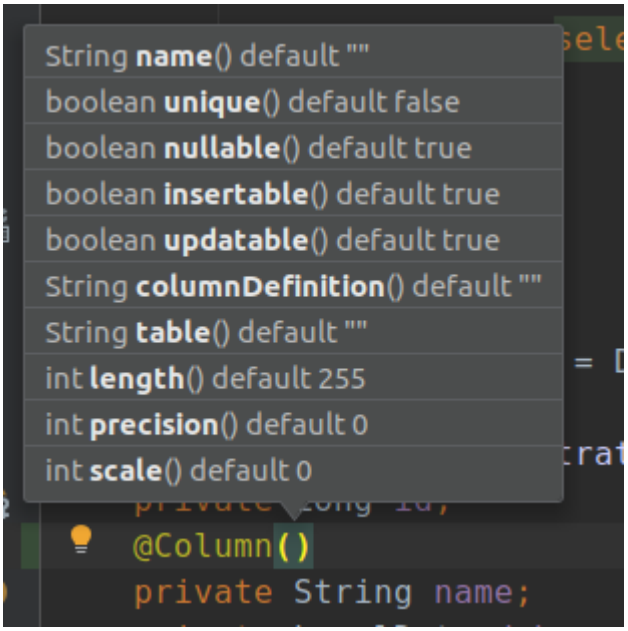
Example Person

```
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;
}
```

Source Package: import javax.persistence.*;

Table 1. Annotations:

Annotation	Description
`@Entity`	makes a class a entity
`@Entity(name = "Person")`	defines the table name of the entity
`@Id`	defines the Pk of a table entity
`@GeneratedValue(strategy = GenerationType.IDENTITY)`	defines a auto generated key

Annotation	Description
	options for fields / columns
`@GeneratedValue(strategy = GenerationType.IDENTITY)`	defines a auto generated key
`@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)

`/*  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-entites are orphans (=Waisen) then)

`/*  Bestelposition */`
`@ManyToOne`
`private Bestellung bestellung;`
	the inverse part of the relationship

Annotation	Description

`/*  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

2.2. 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 customaizability so if you want to ahv custom fields you have to create a new @Entity class and a new @Embaddable class for the Id.

2.2.1. 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.

2.2.2. 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;
    }
```

2.3. JPQL

Java Persistance Query Language

Query:

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());
        }
    }
```

Exmple for a Responde Obect:

Example for Query Responde 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
    }
```

Exmple for saving Respone in a Tuble:

Example for a Tuble Responde

```
    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);
    }
```

2.4. 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();
    }
```

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

2.5. Enitiy 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();
```

3. 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
```

4. REST

4.1. 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

4.2. Examples a RestEndpoint

Common Inports 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();
    }
```

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();
}
```

4.3. Examples for a RestClient

Exmaple 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);
```

5. Technologies

5.1. Jakarta EE

Good Source:
https://eclipse-ee4j.github.io/jakartaee-tutorial/

5.2. Junit

Table 2. Method Anotations

tag	Description

@Test	Turns a public method into a JUnit test case.
@Before	Method to run before every test case
@After	Method to run after every test case
@BeforeClass	Method to run once, before any test cases haverun
@AfterClass	Method to run once, after all test cases have run

Table 3. Assert Methods

Method	Description
assertTrue(test)	fails if the Boolean test is false
assertFalse(test)	fails if the Boolean test is true
assertEquals(expected, actual)	fails if the values are not equal
assertSame(expected, actual)	fails if the values are not the same (by ==)haverun
assertNotSame(expected, actual)	fails if the values are the same (by ==)
assertNull(value)	fails if the given value is notnull
assertNotNull(value)	fails if the given value is null
fail()	causes current test to immediately fail
assertEquals(“message”, expected, actual)	Each method can also be passed a string to display if it fails

Good Source:
https://www.javatpoint.com/
== AsciiDoc

Great

6. Table of Content

Here is my preamble paragraph, but I could really place the TOC anywhere! Lorem ipsum foo bar baz.