Script: NVS 4

Version: 1.0

1. General

1.1. Create Project

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

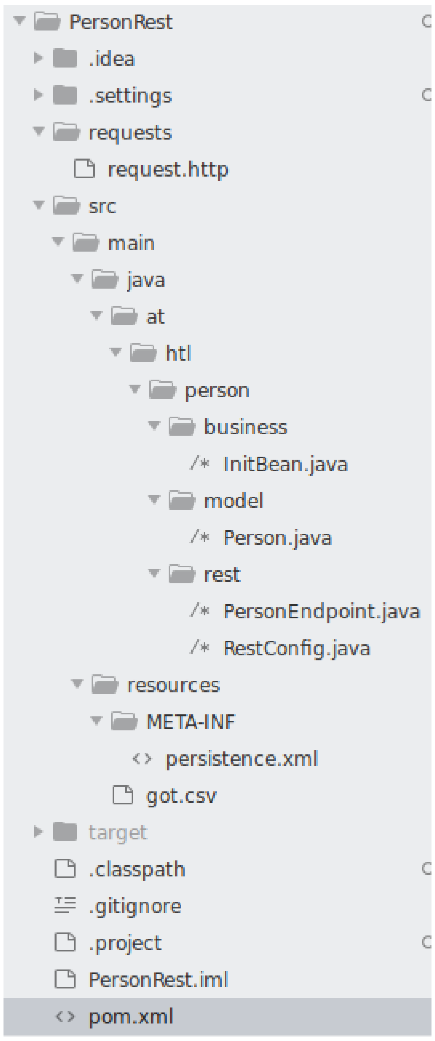
```
<groupId>at.htl</groupId>
<artifactId>PersonRest</artifactId>
```

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. Project Structure



```
PersonRest
  .idea
  .settings
  requests
    request.http
  src
    main
      java
        at
          htl
            person
              business
                InitBean.java
              model
                Person.java
              rest
                PersonEndpoint.java
                RestConfig.java
      resources
        META-INF
          persistence.xml
        got.csv
  target
  .classpath
  .gitignore
  .project
  PersonRest.iml
  pom.xml
```

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**.

1.4. 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 {

}
```

1.5. XML

For xml we have to declare the entity as:

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

@XmlRootElement
public class Vehicle {}
```

1.6. Pom

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>

    <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>

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

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


</project>
```

1.7. Request

Examples for request.html

###

```
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. Read data from csv

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

```
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

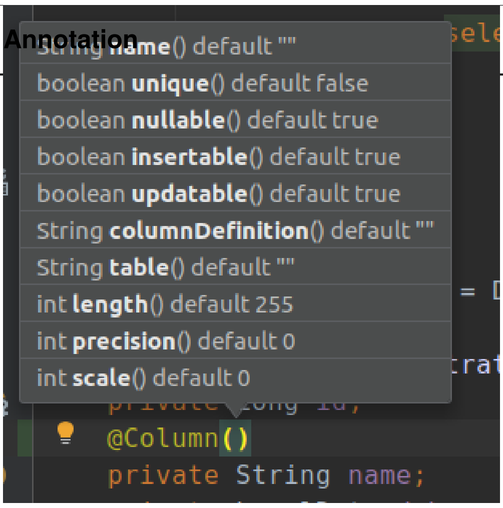
```
package at.htl.person.model;
import javax.persistence.*;

@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;
}
```

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
	options for fields / columns

Annotation	Description

 `String name() default ""`
`boolean unique() default false`
`boolean nullable() default true`
`boolean insertable() default true`
`boolean updatable() default true`
`String columnDefinition() default ""`
`String table() default ""`
`int length() default 255`
`int precision() default 0`
`int scale() default 0`
`@Column()`
`private String name;`	

`@GeneratedValue(strategy = GenerationType.IDENTITY)`	defines a auto generated key
`@Transient`	defines fields that should not be part of the entity

`/*  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

`/*  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. Named Query

Example for Queries

```
@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"
        )
})
```

Rest Example for 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();
}
```

2.3. JPQL

Java Persistance Query Language

More Complex 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());
    }
}
```

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

2.4. Enitiy Manager

```
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

Example for a Endpoint

```
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;

@Path("person")
public class PersonEndpoint {

    public PersonEndpoint() {
    }

    @PersistenceContext
    EntityManager em;

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

    @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();
    }
```

---

5. Technologies

5.1. Jakarta EE

good source: https://eclipse-ee4j.github.io/jakartaee-tutorial/

---

6. AsciiDoc

sdf

Last updated 2019-12-10 17:24:42 +0100