Assignment 4

Web Architectures - 11/12/2022

Sabin Andone

Introduction

For this fourth assignment it was necessary to create a web application which is backed by Enterprise Beans and this web application must run on a Tomcat (version 9.0.65) located outside of Wildfly (version 20.0.1). The web app must contain two pages which shows the following:

- a student page, where student's anagraphic data, his/her list of all the courses in which he/she is enrolled and the marks are shown given the student's matriculation number
- an advisor choice page, where students's anagraphic data and the names of the teachers of the courses the student is enrolled in is shown given the student's matriculation number

The data is provided by an H2 database and it must contain at least 3 students and 3 courses. For this assignment, the structure of the database is defined as follows: Student class has fields Name, Surname, Matriculation. Teacher class has fields Name, Surname. Course class has field Name. Teacher and courses are related by a 1:1 relation whereas student and courses are related by a N:M (many to many) relation, and each relation has an attribute vote which indicates the mark obtained by the student in that course (if he/she passed the exam for that course).

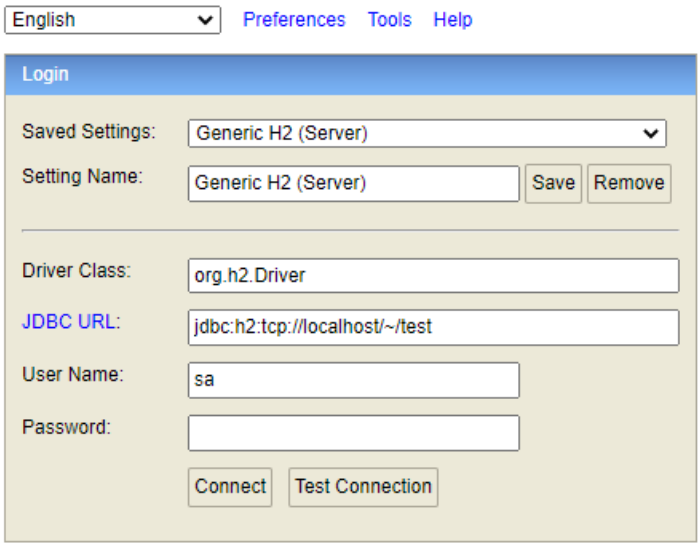
The web app must make use of business delegates, facade, service locator, data transfer objects. The implementation of the business logic must be located inside the EJB part while the web part is used for expressing the request and showing the results. This means that there must be two projects inside the zipped file: one for EJB that contains the beans deployed on Wildfly, one for the web app, which contains the servlets that generate the pages. Moreover, inside the zipped file of this assignment also the database file is provided.

Idea for the solution

The idea for the solution of this assignment is to create two projects: one EJB server project (deployed with Wildfly 20.0.1 Final) and one web app (on Tomcat 9.0.65) that remotely access the beans.

For the EJB project it is necessary to create and setup the database. In this case the database has been created using H2 console.

Figure 2.1: Login settings. The password for user "sa" is "sa".



The creation of the tables and the value insertion are done using SQL commands. For this project we need to create three tables: STUDENT, TEACHER, COURSE and STUDENT_COURSE. The last tables is needed for considering the Student-Course realtion. For each of them it is necessary to create the table and insert values into it as shown in the following Figure:

Figure 2.2: SQL commands for STUDENT table. We insert these commands in the H2 console.

```
CREATE TABLE STUDENT(ID INT PRIMARY KEY,
NAME VARCHAR(255),
SURNAME VARCHAR(255),
MATRICULATION INT);

INSERT INTO STUDENT VALUES(1,'Sabin','Andone',232098);
INSERT INTO STUDENT VALUES(2,'Mario','Rossi',231234);
INSERT INTO STUDENT VALUES(3,'Carlo','Bianchi',245678);
```

Now that the tables have been created on the H2 Database it is necessary to create the entity (JPA) classes in Java for each table created. In order to do this, it is necessary to create the EJB server project that must contains the beans and the class entities. Then it is necessary to add the libraries and the database inside the project, as in the following pictures:

Figure 2.3: Adding the jboss-client and h2 libraries, inside File → Project Structure.

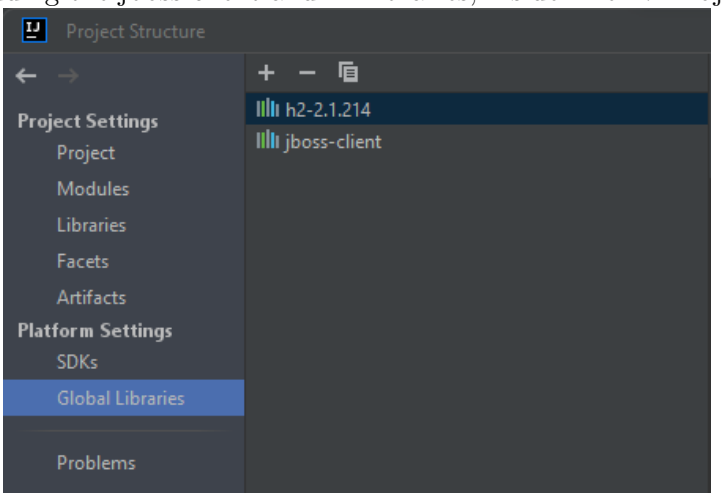
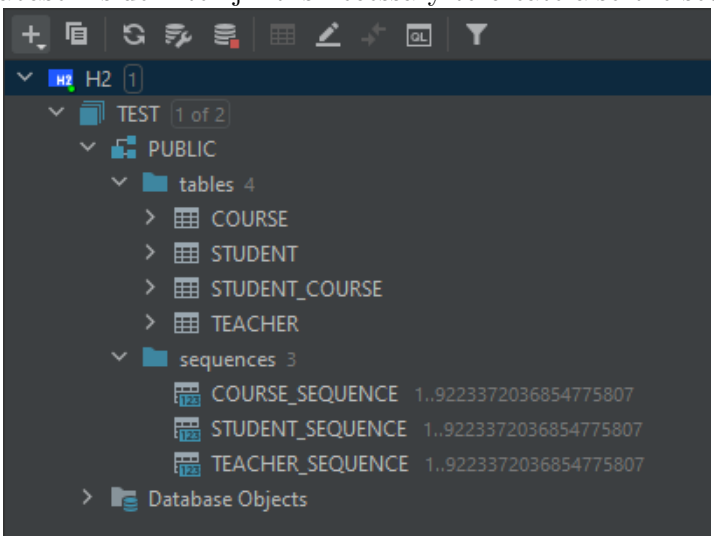
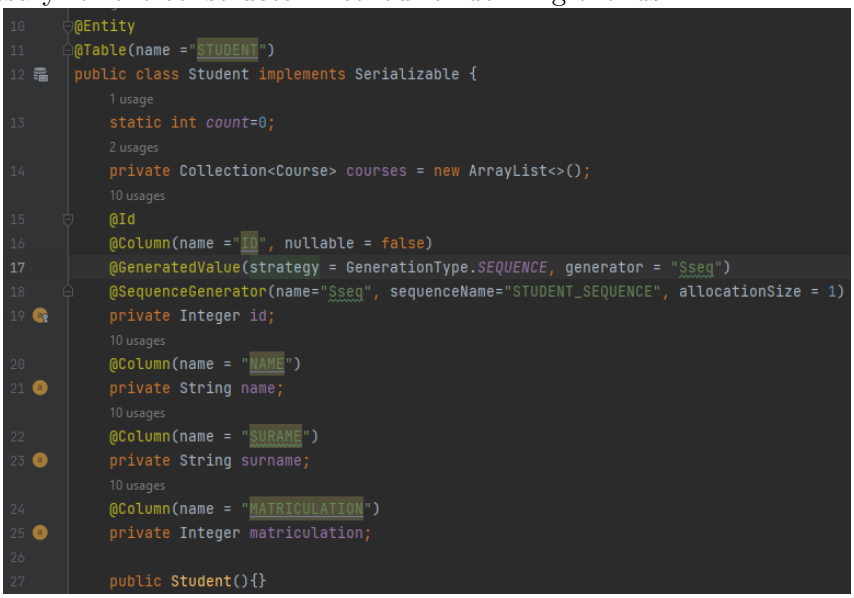


Figure 2.4: the H2 database inside Intellij. It is necessary to create also the sequences for each entity.



The first entity to create is the Student class. It has a many-to-many relation with Course class, thus it is necessary to also consider the list of all courses, defined as a Collection of Course.

Figure 2.5: Student entity class. In this case we define for each column an attribute for the class, The static count variable is necessary for the constructor method for defining the ids.



```java
@Entity
@Table(name ="STUDENT")
public class Student implements Serializable {
    1 usage
    static int count=0;
    2 usages
    private Collection<Course> courses = new ArrayList<>();
    10 usages
    @Id
    @Column(name ="ID", nullable = false)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Sseq")
    @SequenceGenerator(name="Sseq", sequenceName="STUDENT_SEQUENCE", allocationSize = 1)
    private Integer id;
    10 usages
    @Column(name = "NAME")
    private String name;
    10 usages
    @Column(name = "SURAME")
    private String surname;
    10 usages
    @Column(name = "MATRICULATION")
    private Integer matriculation;

    public Student(){}
```

After we define the attributes for the class, we also define the constructor methods, together with setter and getter methods and other methods like toString, equals and hasCode.

Figure 2.6: Constructor method, setter and getter methods for Student class.

```java
public Student(){}
public Student(String name, String surname, int matriculation){
    this.id = count++;
    this.name = name;
    this.surname = surname;
    this.matriculation = matriculation;
}
public int getId() { return id; }
public void setId(int id) { this.id = id; }
public String getName() { return name; }
public void setName(String name) {this.name = name; }
1 usage
public String getSurname() { return surname; }
public void setSurname(String surname) {this.surname = surname; }
1 usage
public int getMatriculation() { return matriculation; }
public void setMatriculation(int matriculation) {this.matriculation = matriculation; }

@Override
public String toString() { return "Student [id=" + id + ", name=" + name + ", surname=" + surname +
        ", matriculation=" + matriculation + "]";}
```

Figure 2.7: Since Student has a N:M relation with Course, we define this relation inside the Student entity class as in the Figure. Then the set and get methods for courses attribute are defined.

```
@ManyToMany(cascade={CascadeType.ALL},fetch=FetchType.EAGER, mappedBy="students")
@JoinTable(name="STUDENT_COURSE")
public Collection<Course> getCourses() {
    return courses;
}
public void setCourses(Collection<Course> courses) {
    this.courses = courses;
}
```

For the Course class we define the attributes and the methods as we did in the Student class, but since Course has also a 1:1 relation with Teacher, it is necessary to consider also the teacher attribute and the set and get methods for teacher.

Figure 2.8: Since Course has a 1:1 relation with Teacher, we define this relation inside the Course entity class as in the Figure. Then the set and get methods for teacher attribute are defined.

```
2 usages
private Teacher teacher;

@OneToOne(cascade={CascadeType.PERSIST}, mappedBy="course")
public Teacher getTeacher() {
    return teacher;
}
public void setTeacher(Teacher teacher) {
    this.teacher = teacher;
}
```

For the Teacher class we define again attributes and methods as we did for the other two classes, but in this case we only have a 1:1 relation with Course class.

The next step to do after we define the three entity classes is to create the three bean classes, one for each entity class. Each bean consists in a interface and the class that implements it. The following figures show the code for Student bean and its interface (the same must be done with Course and Teacher)

Figure 2.9: Student bean interface. The bean class must define the methods declared in the interface. These methods are getters for each attribute plus a getter method for obtaining the instance of that entity class.

```
package it.unitn.andone.assignment_4;

import java.util.Collection;

6 usages  1 implementation
public interface StudentBeanIF {
    1 implementation
    String getName(String matriculation);
    1 implementation
    String getSurname(String matriculation);
    1 implementation
    Integer getMatriculation(int i);
    1 implementation
    Student getStudent(String matriculation);
    1 implementation
    Collection<Course> getCourses(String matriculation);
}
```

Figure 2.10: Student bean class. It implements the interface defined in the previous Figure. It is a stateless and remote bean. Stateless because for the purpose of this assignment we need to visualize data of students and their courses among with the teacher of each course, thus we just need to fetch pieces of data for the website, which can be done with asynchronous methods. Remote because then it is necessary to access the bean with Tomcat.

```
public class StudentBean implements StudentBeanIF{
    5 usages
    @PersistenceContext(unitName="default")
    private EntityManager entityManager;
    @Override
    public String getName(String matriculation) {
        Query q=entityManager.createQuery( s: "Select * From STUDENT where MATRICULATION = "+matriculation);
        Student s=(Student)(q.getSingleResult());
        return s.getName();
    }
    @Override
    public String getSurname(String matriculation) {...}
    @Override
    public Integer getMatriculation(int i) {...}
    @Override
    public Student getStudent(String matriculation) {
        Query q=entityManager.createQuery( s: "Select * From STUDENT where MATRICULATION = "+matriculation);
        Student s=(Student)(q.getSingleResult());
        return s;
    }
    @Override
    public Collection<Course> getCourses(String matriculation) {
        Query q=entityManager.createQuery( s: "Select * From STUDENT where MATRICULATION = "+matriculation);
        Student s=(Student)(q.getSingleResult());
        return s.getCourses();
    }
}
```

In order to be able to access the database, it is necessary to define the EntityManager. This can be done by first defining the @PersistenceContext to have unitName = "default", then in the persistence.xml file located in resources/META-INF directory it is necessary to add the data-source property as in the following Figure:

Figure 2.11: The persistence.xml file.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
             http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
             version="2.2">
    <persistence-unit name="default">
        <jta-data-source>java:jboss/datasources/AssignmentDS</jta-data-source>
    </persistence-unit>
</persistence>
```

Now, before finishing the EJB project, we need to create the client. It consists of the main function, a remoteEJB function that invokes remotely the bean and a lookup function that defines the jndi properties, defines the context and does the lookup on the bean. To do the lookup, it is necessary to look at the correct JNDI name, which in this case is "ejb:/Assignment_4-1.0-SNAPSHOT/StudentBean!it.unitn.andone.assignment_4.StudentBeanIF". To choose the right JNDI name we need to lookup at the start of the server log and choose the ejb one among the JNDI names.

Figure 2.12: JNDI names for the StudentBean. Note that the "java:module/StudentBean!it.unitn.andone.assignment_4.StudentBeanIF" will be needed later for the web app.

```
java:global/Assignment_4-1.0-SNAPSHOT/StudentBean!it.unitn.andone.assignment_4.StudentBeanIF
java:app/Assignment_4-1.0-SNAPSHOT/StudentBean!it.unitn.andone.assignment_4.StudentBeanIF
java:module/StudentBean!it.unitn.andone.assignment_4.StudentBeanIF
java:jboss/exported/Assignment_4-1.0-SNAPSHOT/StudentBean!it.unitn.andone.assignment_4.StudentBeanIF
ejb:/Assignment_4-1.0-SNAPSHOT/StudentBean!it.unitn.andone.assignment_4.StudentBeanIF
java:global/Assignment_4-1.0-SNAPSHOT/StudentBean
java:app/Assignment_4-1.0-SNAPSHOT/StudentBean
java:module/StudentBean
```

Figure 2.13: The client.

```
public class Client {
    public static void main(String[] args) throws Exception {
        remoteEJB();
    }
    1 usage
    private static void remoteEJB() throws NamingException {
        //lookup the remote stateless bean
        final StudentBeanIF student = lookupStudent();
        String name = student.getName( matriculation: "231234");
        System.out.println(name);
    }
    1 usage
    private static StudentBeanIF lookupStudent() throws NamingException {
        final Hashtable jndiProperties = new Hashtable();
        jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY, "org.wildfly.naming.client.WildFlyInitialContextFactory");
        if(Boolean.getBoolean( name: "http")) {
            //use HTTP based invocation. Each invocation will be a HTTP request
            jndiProperties.put(Context.PROVIDER_URL,"http://localhost:8080/wildfly-services");
        } else {
            //use HTTP upgrade, an initial upgrade requests is sent to upgrade to the remoting protocol
            jndiProperties.put(Context.PROVIDER_URL,"remote+http://localhost:8080");
        }
        final Context context = new InitialContext(jndiProperties);
        //do the lookup
        return (StudentBeanIF) context.lookup( name: "ejb:/Assignment_4-1.0-SNAPSHOT/StudentBean!it.unitn.andone.assignment_4.StudentBeanIF");
    }
}
```

Since the database must be accessible using Wildfly, inside the JBOSS_HOME/standalone/configuration it is necessary to modify the standalone.xml file as in the following Figure:

Figure 2.14: The standalone.xml file. It is necessary to add inside the tag subsystem xmlns="urn:jboss:domain:datasources:6.0" the datasource we created.

```
<subsystem xmlns="urn:jboss:domain:datasources:6.0">
    <datasources>
        <datasource jndi-name="java:jboss/datasources/ExampleDS" pool-name="ExampleDS" enabled="true" use-java-context="true" statistics-enabled="${wildfly.da
            <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE</connection-url>
            <driver>h2</driver>
            <security>
                <user-name>sa</user-name>
                <password>sa</password>
            </security>
        </datasource>
        <datasource jndi-name="java:jboss/datasources/AssignmentDS" pool-name="AssignmentDS" enabled="true" use-java-context="true" statistics-enabled="true">
            <connection-url>jdbc:h2:tcp://localhost/~/test;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE</connection-url>
            <driver>h2</driver>
            <security>
                <user-name>sa</user-name>
                <password>sa</password>
            </security>
        </datasource>
        <drivers>
            <driver name="h2" module="com.h2database.h2">
                <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
            </driver>
        </drivers>
    </datasources>
</subsystem>
```

Once the EJB server project is done, it is necessary to deploy it and to create the .jar file, which in this case it has been calles "Assignment_4.jar".

Now it is necessary to create the web application. The web app must show two pages: the one with the student's data and the courses that the student is enrolled in, and the one with the student's data and the list with the teachers of the courses the student is enrolled in. It has been decided to create a start page, that greets the user and explaining that the s/he must insert the matriculation number of the student. After the matriculation number has been inserted, the user must choose if s/he wants to observe the student page or the advisor page, by pressing the corresponding button. When one of the two button is pressed, the client sends the request with the inserted matriculation number to the choosen servlet and shows the results to the client. Note that it is necessary to include the jar file that contains the beans created in the EJB project into the web app project.

Figure 2.15: The start page. It consists of a paragraph and a form.

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Start Page</title>
</head>
<body>
<h1><%= "Hello! Insert the matriculation number of a student to see data about that student." %>
</h1>
<br/>
<form action="/StudentServlet" method="get">
    <label for="matr">Matriculation number:</label><br>
    <input type="text" id="matr" name="matr" placeholder="xxxxxx"><br><br>
    <input type="submit" value="Student page">
    <input type="submit" value="Advisor page" formaction="/AdvisorPageServlet">
</form>
</body>
</html>
```

Figure 2.16: The body of get method of the StudentServlet. The AdvisorPageServlet is similar, except for the fact that we fetch data about teachers and not courses.

```
//getting the matriculation number through the request
String matriculation = request.getParameter( s: "matr");

response.setContentType("text/html");
Context ctx = null;
StudentBeanIF student=null;
Collection<Course> courses = null;
try {
    ctx = new InitialContext();
    String name="java:module/StudentBean!it.unitn.andone.assignment_4.StudentBeanIF";
    student= (StudentBeanIF) ctx.lookup(name);
    courses = student.getCourses(matriculation);
} catch (NamingException e) {
    e.printStackTrace();
}
PrintWriter out = response.getWriter();
out.println("<html><head><title>" + matriculation + "</title></head><body>");
out.println("<h1>Name: " + student.getName(matriculation) + "</h1><br>");
out.println("<h1>Surname: " + student.getSurname(matriculation) + "</h1><br>");
out.println("<h1>Teachers: </h1><br>");

for (Course c : courses) {
    out.println("<p>"+c.getTeacher()+" </p><br>");
}
out.println("</body></html>");
```

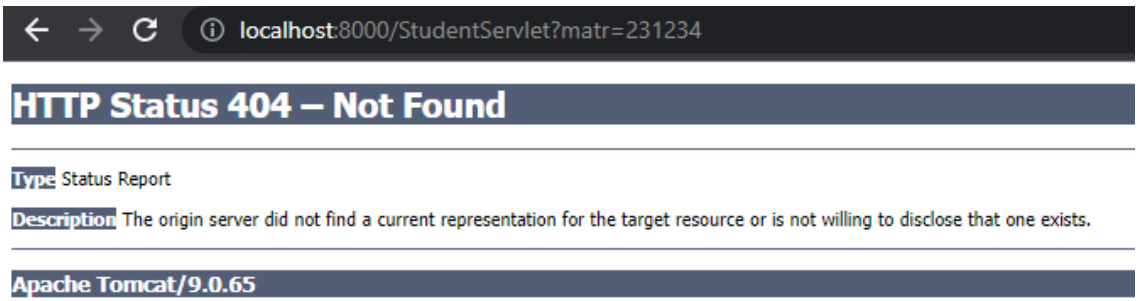
Comments

Problems: when executing the client inside the EJB server project the following error shows:
Exception in thread "main" javax.ejb.EJBException: java.lang.NullPointerException:
Cannot invoke "jakarta.persistence.EntityManager.createQuery(String)" because "this.entityManager" is null
at org.jboss.as.ejb3@20.0.1.Final//org.jboss.as.ejb3.tx.CMTTxInterceptor.invokeInOurTx(CMTTxInterceptor.java:266
at org.jboss.as.ejb3@20.0.1.Final//org.jboss.as.ejb3.tx.CMTTxInterceptor.required(CMTTxInterceptor.java:388)
at org.jboss.as.ejb3@20.0.1.Final//org.jboss.as.ejb3.tx.CMTTxInterceptor.processInvocation(CMTTxInterceptor.java:1
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext$Invocation.proceed(InterceptorContext.j
at org.jboss.weld.core@3.1.4.Final//org.jboss.weld.module.ejb.AbstractEJBRequestScopeActivationInterceptor.around
(AbstractEJBRequestScopeActivationInterceptor.java:81)
at org.jboss.as.weld.common@20.0.1.Final//
org.jboss.as.weld.ejb.EjbRequestScopeActivationInterceptor.processInvocation(EjbRequestScopeActivationInterceptor
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ejb3@20.0.1.Final//
org.jboss.as.ejb3.component.interceptors.CurrentInvocationContextInterceptor.processInvocation
(CurrentInvocationContextInterceptor.java:41)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ejb3@20.0.1.Final//
org.jboss.as.ejb3.component.invocationmetrics.WaitTimeInterceptor.processInvocation(WaitTimeInterceptor.java:47)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ejb3@20.0.1.Final//
org.jboss.as.ejb3.security.SecurityContextInterceptor.processInvocation(SecurityContextInterceptor.java:100)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ejb3@20.0.1.Final//
org.jboss.as.ejb3.deployment.processors.StartupAwaitInterceptor.processInvocation(StartupAwaitInterceptor.java:22)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ejb3@20.0.1.Final//org.jboss.as.ejb3.component.interceptors.ShutDownInterceptorFactory
$1.processInvocation(ShutDownInterceptorFactory.java:64)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ejb3@20.0.1.Final//
org.jboss.as.ejb3.deployment.processors.EjbSuspendInterceptor.processInvocation(EjbSuspendInterceptor.java:57)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ejb3@20.0.1.Final//
org.jboss.as.ejb3.component.interceptors.LoggingInterceptor.processInvocation(LoggingInterceptor.java:67)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ee@20.0.1.Final//
org.jboss.as.ee.component.NamespaceContextInterceptor.processInvocation(NamespaceContextInterceptor.java:50)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.invocation@1.5.2.Final//
org.jboss.invocation.ContextClassLoaderInterceptor.processInvocation(ContextClassLoaderInterceptor.java:60)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.run(InterceptorContext.java:438)

at org.wildfly.security.elytron-private@1.12.1.Final//
org.wildfly.security.manager.WildFlySecurityManager.doChecked(WildFlySecurityManager.java:627)
at org.jboss.invocation@1.5.2.Final//
org.jboss.invocation.AccessCheckingInterceptor.processInvocation(AccessCheckingInterceptor.java:57)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.ChainedInterceptor.processInvocation(ChainedInterceptor.ja
at org.jboss.as.ee@20.0.1.Final//org.jboss.as.ee.component.ViewService$View.invoke(ViewService.java:198)
at org.wildfly.security.elytron-private@1.12.1.Final//
org.wildfly.security.auth.server.SecurityIdentity.runAsFunctionEx(SecurityIdentity.java:421)
at org.jboss.as.ejb3@20.0.1.Final//org.jboss.as.ejb3.remote.AssociationImpl.invokeWithIdentity(AssociationImpl.java:5
at org.jboss.as.ejb3@20.0.1.Final//org.jboss.as.ejb3.remote.AssociationImpl.invokeMethod(AssociationImpl.java:575)
at org.jboss.as.ejb3@20.0.1.Final//org.jboss.as.ejb3.remote.AssociationImpl.lambda
$receiveInvocationRequest$0(AssociationImpl.java:205)
at org.jboss.as.ejb3@20.0.1.Final//org.jboss.as.ejb3.remote.AssociationImpl.execute(AssociationImpl.java:298)
at org.jboss.as.ejb3@20.0.1.Final//org.jboss.as.ejb3.remote.AssociationImpl.receiveInvocationRequest
(AssociationImpl.java:251)
at org.jboss.ejb-client@4.0.33.Final//org.jboss.ejb.protocol.remote.EJBServerChannel
$ReceiverImpl.handleInvocationRequest(EJBServerChannel.java:473)
at org.jboss.ejb-client@4.0.33.Final//org.jboss.ejb.protocol.remote.EJBServerChannel
$ReceiverImpl.handleMessage(EJBServerChannel.java:208)
at org.jboss.remoting@5.0.18.Final//org.jboss.remoting3.remote.RemoteConnectionChannel.lambda
$handleMessageData$3(RemoteConnectionChannel.java:430)
at org.jboss.remoting@5.0.18.Final//org.jboss.remoting3.EndpointImpl$TrackingExecutor.lambda$execute
$0(EndpointImpl.java:991)
at org.jboss.threads@2.3.3.Final//
org.jboss.threads.ContextClassLoaderSavingRunnable.run(ContextClassLoaderSavingRunnable.java:35)
at org.jboss.threads@2.3.3.Final//org.jboss.threads.EnhancedQueueExecutor.safeRun(EnhancedQueueExecutor.java:1
at org.jboss.threads@2.3.3.Final//org.jboss.threads.EnhancedQueueExecutor
$ThreadBody.doRunTask(EnhancedQueueExecutor.java:1486)
at org.jboss.threads@2.3.3.Final//org.jboss.threads.EnhancedQueueExecutor$ThreadBody.run
(EnhancedQueueExecutor.java:1377)
at java.base/java.lang.Thread.run(Thread.java:832)
Caused by: java.lang.NullPointerException at deployment.Assignment_4-1.0-SNAPSHOT.war
//it.unitn.andone.assignment_4.StudentBean.getName(StudentBean.java:18)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:64)
at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.base/java.lang.reflect.Method.invoke(Method.java:564)
at org.jboss.as.ee@20.0.1.Final//
org.jboss.as.ee.component.ManagedReferenceMethodInterceptor.processInvocation
(ManagedReferenceMethodInterceptor.java:52)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext$Invocation.proceed(InterceptorContext.j
at org.jboss.as.weld.common@20.0.1.Final//
org.jboss.as.weld.interceptors.Jsr299BindingsInterceptor.delegateInterception(Jsr299BindingsInterceptor.java:79)
at org.jboss.as.weld.common@20.0.1.Final//
org.jboss.as.weld.interceptors.Jsr299BindingsInterceptor.doMethodInterception(Jsr299BindingsInterceptor.java:89)
at org.jboss.as.weld.common@20.0.1.Final//
org.jboss.as.weld.interceptors.Jsr299BindingsInterceptor.processInvocation(Jsr299BindingsInterceptor.java:102)
at org.jboss.as.ee@20.0.1.Final//org.jboss.as.ee.component.interceptors.UserInterceptorFactory
$1.processInvocation(UserInterceptorFactory.java:63)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)

at org.jboss.as.ejb3@20.0.1.Final//
org.jboss.as.ejb3.component.invocationmetrics.ExecutionTimeInterceptor.processInvocation
(ExecutionTimeInterceptor.java:43)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.jpa@20.0.1.Final//
org.jboss.as.jpa.interceptor.SBInvocationInterceptor.processInvocation(SBInvocationInterceptor.java:47)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ee@20.0.1.Final//
org.jboss.as.ee.concurrent.ConcurrentContextInterceptor.processInvocation(ConcurrentContextInterceptor.java:45)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InitialInterceptor.processInvocation(InitialInterceptor.java:40
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.ChainedInterceptor.processInvocation(ChainedInterceptor.ja
at org.jboss.as.ee@20.0.1.Final//
org.jboss.as.ee.component.interceptors.ComponentDispatcherInterceptor.processInvocation
(ComponentDispatcherInterceptor.java:52)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ejb3@20.0.1.Final//
org.jboss.as.ejb3.component.pool.PooledInstanceInterceptor.processInvocation(PooledInstanceInterceptor.java:51)
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ejb3@20.0.1.Final//
org.jboss.as.ejb3.component.interceptors.AdditionalSetupInterceptor.processInvocation(AdditionalSetupInterceptor.jav
at org.jboss.invocation@1.5.2.Final//org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:422)
at org.jboss.as.ejb3@20.0.1.Final//org.jboss.as.ejb3.tx.CMTTxInterceptor.invokeInOurTx(CMTTxInterceptor.java:252
... 46 more
Process finished with exit code 1

Instead, when trying to execute the web app, after inserting the matriculation number into the form, it shows an error 404.



During the development of this assignment I made sure that everything has been set properly, by adding the libraries and the database, deploying the beans and configuring the file as shown in the previous section, but still these errors would show up.