Department of Information Engineering and Computer Science
Fintech
University of Trento

Web Architectures
Simone Bianchin

Assignment 4
11 December 2022

Index

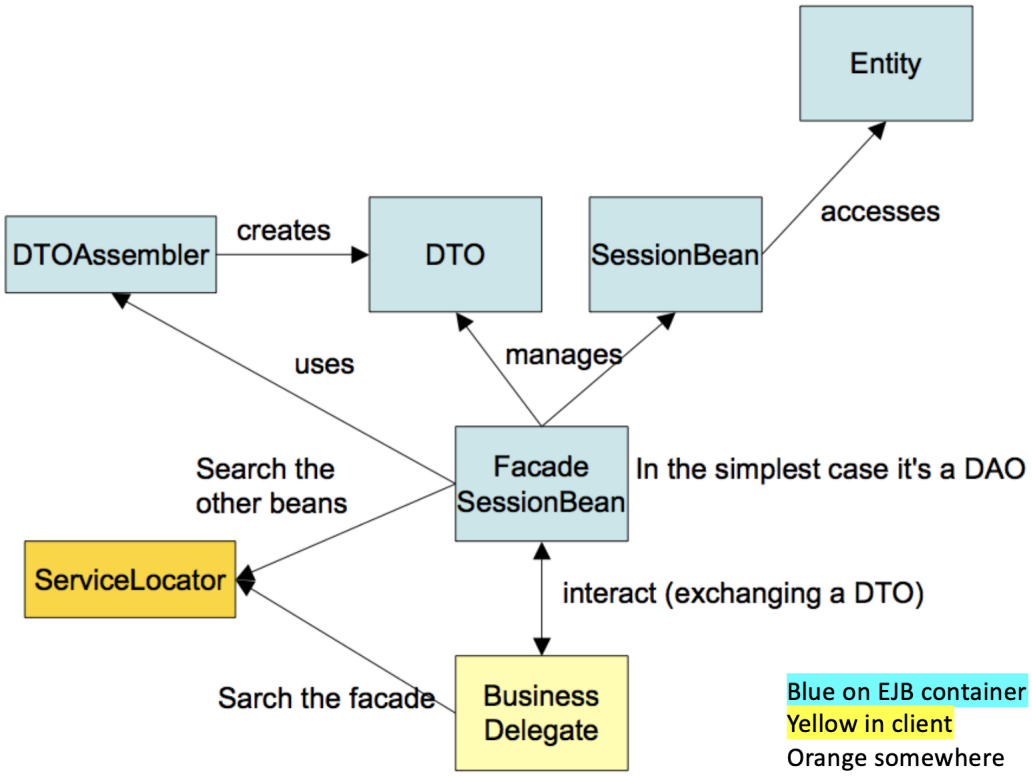
The case

A little sum-up of the assignment.

1. Implement a simple school database schema and two main operations that interact with it.
2. The Web App is composed of a Java JSP Frontend, naming service and EJB hosted with wildly and a H2 database.

My implementation

My implementation follows the architecture shown in class.



Entity All the entities implemented as JPA as shown in listing 1, and their integration with Hibernate really showed their potential. With the provided decorators, the domain of the entities and their relationship are defined very fast and clearly.

Listing 1: The implementation of Teacher Entity.

```
@Entity
@Table(name = "TEACHER")
public class Teacher {
    @Id
    @Column(name="ID", nullable = false)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "myseq")
    @SequenceGenerator(name = "myseq", sequenceName = "SEQUENCE_NAME", allocationSize = 1)
    private int id;

    @Column(name = "FIRSTNAME")
    private String firstName;

    @Column(name = "LASTNAME")
    private String lastName;

    public Teacher() {};

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public TeacherDTO toTDO() {
        return new TeacherDTO(this.id, this.firstName, this.lastName);
    }
}
```

Data source - EJB/JPA The data source definition has been included in the *persistence.xml*. By doing so the naming service of Wildfly is utilized to make the EJBs and Hibernate independent from hardcoded

endpoint. *persistence.xml* includes also some tweaking to automatically generate SQL schemas that will be executed into H2.

Listing 2: persistence.xml

```
<?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/persi
             version="2.2">
    <persistence-unit name="default">
        <jta-data-source>java:jboss/datasources/SchoolDS</jta-data-source>
        <properties>
            <property name="hibernate.connection.driver_class" value="org.h2.Driver"/>
            <property name="hibernate.archive.autodetection" value="class"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
            <property name="hbm2ddl.auto" value="create-drop"/>
            <!-- without below table was not created -->
            <property name="javax.persistence.schema-generation.database.action" value="drop-and-create" /
        </properties>
    </persistence-unit>
</persistence>
```

Local Beans - EJB/JPA

Two local beans (StudentBean, EnrollmentBean) have been defined to implement the atomic operations on the relative Entity. Here we can see how the PersistenceContext is used, related to the Data Source described before. LocalBeans are not remotely exposed since there are Facades to do the job.

Listing 3: StudentBean.java

```
@Stateless
@LocalBean
public class StudentBean implements Serializable {
    @PersistenceContext
    private EntityManager entityManager;

    public Student getStudent(int matriculation) {
        Query q = entityManager.createQuery("from␣Student␣where␣matriculation␣=␣:matriculation", Student.c
        List<Student> students = q.setParameter("matriculation", matriculation)
                .getResultList();
        return students.isEmpty() ? null : students.get(0);
    }

    public StudentBean () {}
}
```

Facades - EJB/JPA

Two Facades have been defined, one for each Use Case. They wrap up all the LocalBean operations to have enough data to compose the relative DTO. The local beans are defined as class fields, and they are pulled from the pool and invoked as simple instances. Since these Facades are remotely accessible, two

interfaces are shared with the clients with a library defined as a separate project. This gives maximums flexibility and maintainability through the development.

Listing 4: GetStudentInfoFacade.java

```
@Stateless
@Remote(GetStudentInfoIF.class)
public class GetStudentInfoFacade implements GetStudentInfoIF {
    @EJB
    StudentBean userBean;

    @EJB
    EnrollmentBean enrollmentBean;

    public GetStudentInfoDTO getStudentInfo(int matriculation) {
        Student student = userBean.getStudent(matriculation);
        if (student == null) return null;

        List<Enrollment> enrollments = enrollmentBean.getStudentEnrollment(student.getMatriculation());
        List<EnrollmentDTO> enrollmentDTOS = enrollments.stream()
                .map(Enrollment::toDTO)
                .collect(Collectors.toList());
        return new GetStudentInfoDTO(student.getMatriculation(), student.getFirstName(), student.getLastNa
    }
}
```

DTO - EJB/JPA DTOs classes become very useful to pass elaborated data to the front-end. They are generated from the corresponded Entity through a specific method (*toDTO()*).

Listing 5: EnrollmentDTO.java

```
public class EnrollmentDTO implements Serializable {
    int studentMatriculation;
    int courseId;
    String courseName;
    int rating;

    public EnrollmentDTO(int studentMatriculation, int courseId, String courseName, int rating) {
        this.studentMatriculation = studentMatriculation;
        this.courseId = courseId;
        this.courseName = courseName;
        this.rating = rating;
    }

    public int getStudentMatriculation() {
        return this.studentMatriculation;
    }

    public int getCourseId() {
        return this.courseId;
    }

    public int getRating() {
        return this.rating;
    }
```

```
    public String getCourseName() {
        return this.courseName;
    }
}
```

Service Locator - Web Client

As seen in class, the Service Locator pattern is useful to reduce CPU usage by avoiding doing multiple lookups on the same resource. Through a hashmap each service is cached with its URL as key.

Listing 6: ServiceLocator.java

```
public class ServiceLocator {
    public enum Service {
        GET_USER_INFO_EJB("ejb:/WildflyDemo-1.0-SNAPSHOT/GetStudentInfoFacade!facade.GetStudentInfoIF"),
        GET_USER_ADVISORS_EJB("ejb:/WildflyDemo-1.0-SNAPSHOT/GetStudentAdvisorsFacade!facade.GetStudentAdv

        private final String text;

        Service(final String text) {
            this.text = text;
        }

        @Override
        public String toString() {
            return text;
        }
    }
    private static HashMap<String, Object> cache;

    static {
        cache = new HashMap<String, Object>();
    }

    public static Object getService(Service serviceUrl) throws NamingException {
        Properties p = new Properties();
        p.put(InitialContext.PROVIDER_URL, "http-remoting://localhost:8888");
        p.put(InitialContext.INITIAL_CONTEXT_FACTORY, "org.wildfly.naming.client.WildFlyInitialContextFact
        Object service = cache.get(serviceUrl);
        if (service == null) {
            InitialContext context = new InitialContext(p);
            service = (Object) context.lookup(serviceUrl.toString());
            cache.put(serviceUrl.toString(), service);
        }
        return service;
    }
}
```

Business Delegates - Web Client

Business delegates are classes that encapsulate all the Use Case calls to the remote Facades. Since the Use Cases of this assignment need just a call each the Business Delegates are pretty simple.

Listing 7: GetStudentInfoBD.java

```
public class GetStudentInfoBD {
```

```
    private GetStudentInfoIF lookupService = null;

    public GetStudentInfoBD() {
        try {
            this.lookupService = (GetStudentInfoIF) ServiceLocator.getService(ServiceLocator.Service.GET_U
        } catch (NamingException e) {
            throw new RuntimeException(e);
        }
    }

    public GetStudentInfoDTO doTask(int matriculation) {
        return this.lookupService.getStudentInfo(matriculation);
    }
}
```

FrontEnd - Web Client

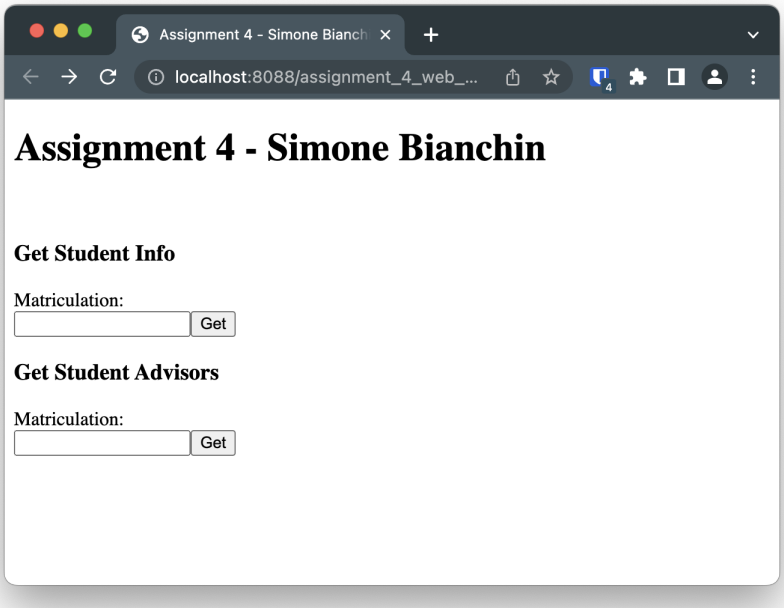
Finally, the front-end servlet calls the relative Business Delegates and shows the output. It also manages some edge cases like nonexisting students.

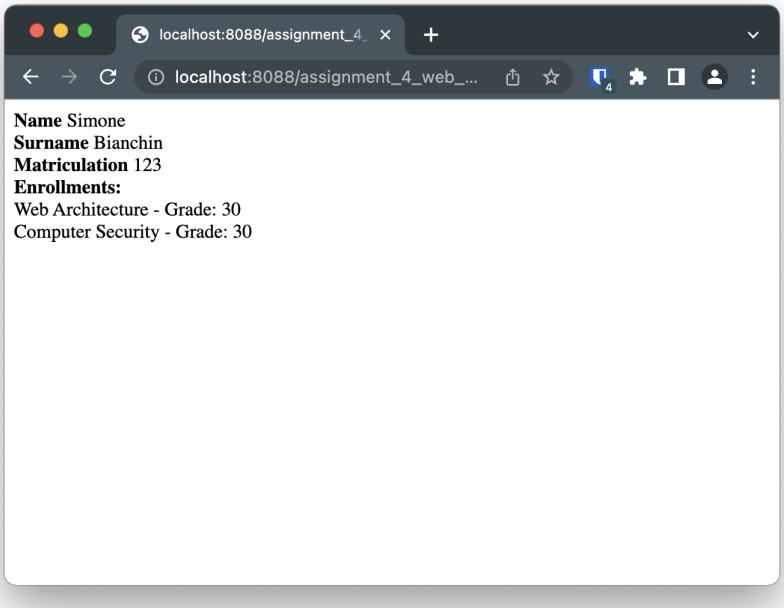
Listing 8: GetStudentInfoServlet.java

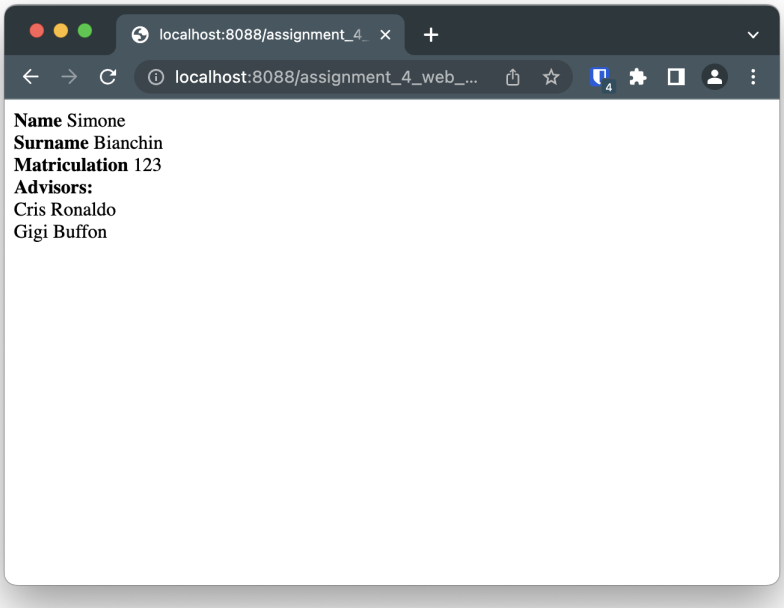
```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        response.setContentType("text/html");

        GetStudentInfoBD getStudentInfoBD = new GetStudentInfoBD();
        GetStudentInfoDTO studentInfoDTO = getStudentInfoBD.doTask(Integer.parseInt(request.getParameter("

        PrintWriter out = response.getWriter();
        out.println("<html><body>");

        if (studentInfoDTO == null) {
            out.println("No␣student␣with␣such␣matriculation␣found.");
            out.println("</body></html>");
            return;
        }

        out.println("<b>Name</b>␣"+studentInfoDTO.getStudentFirstName()+"<br␣/>");
        out.println("<b>Surname</b>␣"+studentInfoDTO.getStudentLastName()+"<br␣/>");
        out.println("<b>Matriculation</b>␣"+studentInfoDTO.getMatriculation()+"<br␣/>");
        out.println("<b>Enrollments:</b>"+"<br␣/>");
        for (EnrollmentDTO enrollment: studentInfoDTO.getEnrollments()) {
            out.println(""+enrollment.getCourseName() + "␣-␣Grade:␣"+enrollment.getRating()+"<br␣/>");
        }
        out.println("</body></html>");
    }
```

App running

Here some screenshots of the app running.


Assignment 4 - Simone Bianch
localhost:8088/assignment_4_web_...
Assignment 4 - Simone Bianchin
Get Student Info
Matriculation:
Get
Get Student Advisors
Matriculation:
Get


localhost:8088/assignment_4_
localhost:8088/assignment_4_web_...
Name Simone
Surname Bianchin
Matriculation 123
Enrollments:
Web Architecture - Grade: 30
Computer Security - Grade: 30



Comments

This assignment helped me to understand the great potential of EJB and made me go over the JPAs I had seen previously. Also, I learns some new patterns that were found to be useful. The configuration of the whole project has been a bit frustrating, I first tried to structure a submodule folder pattern with Gradle, but the dependencies never worked. I'm gonna look into that in the future.