



# Web Architectures

## Assignment 4

Antonio Pio Padalino

December 11, 2022

### Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	The project goal . . . . .	2
1.2	The project structure . . . . .	2
<b>2</b>	<b>Methods</b>	<b>3</b>
2.1	The remote container . . . . .	3
2.2	The client logic . . . . .	5
<b>3</b>	<b>Comments</b>	<b>8</b>
3.1	Possible improvements . . . . .	8

# 1 Introduction

## 1.1 The project goal

The assignment concerns the development of a web application backed by Enterprise Java Beans. The application implements two simple pages: one which given a student matriculation number displays their anagraphic data and the list of courses they are enrolled in with grades (if available), and another which given - again - a student matriculation number displays all the teachers available as advisors, meaning all teachers from the courses the student is enrolled in. Keeping in mind that teacher and courses are related by a one-to-one relation, while student and courses are many-to-many. Data persistency is provided by an H2 Database to keep all teachers, students and courses information, and other patterns such as Business Delegates, Façade, Service Locator and DTO are used throughout the project as well. One more core specification for the project is that the web application must run on a Tomcat server outside of the Wildfly where the EJBs are deployed.

## 1.2 The project structure

To achieve the assignment goal, two projects have been created.

- **Assignment4\_remote** is the project containing all EJBs used, the integration with the H2 database, and all business logics. Will be also referenced throughout the report as the remote container.
- **Assignment4** is the project containing the Servlet for the student and advisor choice pages and the logics for such requests, and it accesses the stateless bean on the remote container using JNDI lookup.

To enter more in detail, **Assignment4\_remote** contains:

- **StudentHandlerBean** and its interface. A stateless bean which is responsible for the interactions with the database, which exposes methods for querying different pieces of information about a student;
- **StudentEntity**, **TeacherEntity** and **CourseEntity**. Entity classes for the tables *Student*, *Teacher* and *Courses* of the H2 database;
- **StudentDTO**. A Data Transfer Object[2] for a single student, containing fields for all the possible pieces of information which **StudentHandlerBean** can query.

while **Assignment4** contains:

- **HelloServlet**, **StudentServlet** and **AdvisorChoiceServlet**, which handle the 3 main pages of the web application, including a very simple home page;
- **ServiceLocator**, which given a JNDI address performs the lookup and invoke a remote stateless bean;
- **BusinessDelegate**, which handles the usage of the **ServiceLocator** and the remote bean **StudentHandlerBean**, exposing a method to *directly* obtain the student's requested information.

**StudentHandlerBeanIf**, interface of **StudentHandlerBean**, has been included inside the project **Assignment4** under the `it.unitn.padalino.assignment4remote.ejb` classpath, to simulate the one in the remote container. Also **StudentDTO** has been included to make it possible for the Business Delegate to handle students data.

More on each component and their specific functioning will be discussed in the Methods section.

## 2 Methods

Before discussing of each component, here is a brief overview of their interactions.

An H2 database containing information about students, courses and respective teachers is instantiated and populated with some random entries at first. The stateless bean **StudentHandlerBean**, which lives on the remote container, handles persistency through Hibernate, and exposes a number of methods to get data from the database. The `.getStudent()` method, in particular, pack all information gathered on a student inside a **StudentDTO** object, for an easy transfer and access. On an external Tomcat, **StudentHandlerBean** is invoked through JNDI lookup, and information about students are requested and displayed in the Student page and the Advisor choice page. Obtaining the requested student's info is a job for the Business Delegate, which handles the lookup through a Service Locator and also keeps a cache of the students which have already been queried to reduce the number of remote calls and provide a faster service.

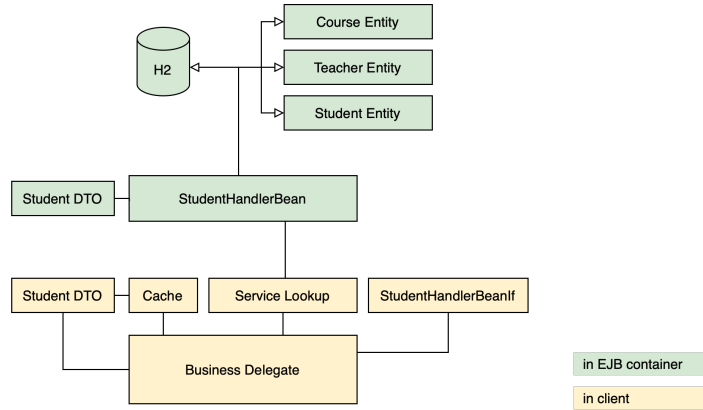


Figure 1: A schematic representation of all the components described and their interactions.

### 2.1 The remote container

**Persistency and entities** JPA (Java Persistence API) is a Java specification that bridges the gap between relational databases and object-oriented programming. It defines a common abstraction that one can use in their code to interact with relational databases. Hibernate is a standard implementation of the JPA specification, which helps in mapping Java data types to SQL data types.

In this framework, *Entities*[4] are nothing but POJOs representing data that can be persisted to the database, with each entity representing a table stored - in this case - in a H2[6][5] database called **unidb**. The relational H2 database is made up of 4 tables:

- **STUDENT**, containing students information, in particular about their name, surname, and matriculation (the id, primary key)
- **TEACHER**, containing teachers information, in particular about their name and surname, together with a unique id;
- **COURSE**, containing courses information, in particular the course name and a unique id;
- **ATTENDANCY**, a table mapping students to attended courses, together with an entry for the grade for each course (which is 0 if the exam is still to be attempted).

Note that, since the mapping between teachers and courses is one-to-one, it is not necessary to have another table to map a teachers to courses, but we can just assume that there will always be as many entries in the **COURSES** table as there will be in the **TEACHERS** table, and teacher with id N will always hold the course with id N.

After the database creation, all the tables described have been created with the following SQL syntax:

```
1 CREATE TABLE STUDENT
2 (
3     MATRICULATION INTEGER NOT NULL GENERATED always AS IDENTITY CONSTRAINT student_pk
4     PRIMARY KEY,
5     NAME          VARCHAR(30),
6     SURNAME       VARCHAR(30)
7 );
```

```
1 CREATE TABLE TEACHER
2 (
3     ID          INTEGER NOT NULL GENERATED always AS IDENTITY CONSTRAINT teacher_pk
4     PRIMARY KEY,
5     NAME        VARCHAR(30),
6     SURNAME     VARCHAR(30)
7 );
```

```
1 CREATE TABLE COURSE
2 (
3     ID          INTEGER NOT NULL GENERATED always AS IDENTITY CONSTRAINT course_pk PRIMARY
4     KEY,
5     NAME        VARCHAR(30)
6 );
```

```
1 CREATE TABLE ATTENDANCY
2 (
3     ID          INTEGER NOT NULL GENERATED always AS IDENTITY CONSTRAINT attend_pk
4     PRIMARY KEY,
5     STUDENT_ID  INTEGER,
6     COURSE_ID   INTEGER,
7     GRADE       INTEGER
8 );
```

Then some random names and surnames for 10 students, 4 teachers and some fantasy names for the respective courses have been inserted with the `INSERT INTO <TABLE> (<COL1>, <COL2>, etc.) VALUES` syntax.

After the database has been populated, to connect it successfully to the `Assignment4_remote` project, the following line has been added to the `persistence.xml` inside `resources/META-INF`:

```
1 <jta-data-source>java:jboss/datasources/UniDB</jta-data-source>
```

and the `standalone.xml` inside `standalone/configuration` in the Wildfly home directory has been edited by inserting the following under `datasources`:

```
1 <datasource jndi-name="java:jboss/datasources/UniDB" pool-name="UniDB" enabled="
2 true" use-java-context="true" statistics-enabled="true">
3     <connection-url>jdbc:h2:tcp://localhost/~/unidb;DB_CLOSE_DELAY=- 1;
4     DB_CLOSE_ON_EXIT=FALSE</connection-url>
5     <driver>h2</driver>
6     <security>
7         <user-name>admin</user-name>
8         <password>admin</password>
9     </security>
10 </datasource>
```

SELECT * FROM STUDENT;			SELECT * FROM COURSE;			SELECT * FROM TEACHER;			SELECT * FROM ATTENDANCY;			
MATRICULATION	NAME	SURNAME	ID	NAME		ID	NAME	SURNAME	ID	STUDENT_ID	COURSE_ID	GRADE
1	Dalton	Burton	1	Practical Demonology		1	Ms. Virginia	Glass	1	1	1	26
2	Eliana	Fleming	2	Wave Theory		2	Mr. Bruno	Gibbs	2	1	2	28
3	Caitlin	Sears	3	Foundations of Necromancy		3	Ms. Lulu	Baldwin	3	1	4	30
4	Sasha	Wilkins	4	Time Travel Lab		4	Mr. Ivan	Phelps	4	2	2	0
5	Carlos	Zuniga							5	2	3	27
6	Paul	Pope							6	3	1	0
7	Nikita	Calhoun							7	4	1	25
8	Gary	Proctor							8	4	2	0
9	Alexandra	Rush							9	4	3	0
10	Alyssia	Colon							10	4	4	30
(10 righe, 6 ms)			(4 righe, 5 ms)			(4 righe, 6 ms)			11	5	1	20
									12	6	4	0
									13	6	2	31
									14	7	2	0
									15	8	1	0
									16	9	2	30
									17	9	3	30
									18	10	2	0
									19	10	4	18
									(19 righe, 6 ms)			

Figure 2: The schema of the H2 database, as it was described.

**The StudentHandlerBean** The stateless bean `StudentHandlerBean` manages the communication with the database, retrieving the desired information about a student. It exposes methods to query the name of a student, the surname, the list of courses attended, the list of grades for the courses attended, and the list of teachers of said courses. Also, another method `.getStudent()` simply put all those information together returning a `StudentDTO`.

Hiding all the complexity and queries needed to extract a student's data under simple methods (especially with the `.getStudent()` method), the `StudentHandlerBean` may be considered a Façade[3].

To retrieve a student's details, the method exposed by the bean relies on both Hibernate queries and native SQL queries. Hibernate Query Language (HQL)[7] is a query language similar to SQL, but working with persistent objects and their properties. A student's name and surname is easily queried with an HQL query, but for obtaining the lists of courses, teachers and grades - requiring a slightly more complex syntax - native SQL queries were preferred.

Of course, the bean is annotated with

```
1 @Stateless
2 @Remote(StudentHandlerBeanIf.class)
```

to be able to invoke the bean remotely.

Having completed the work on the remote container, `Assignment4remote` has been deployed on Wildfly-20.0.1. This has been done by building the artifact on IntelliJ and then copying the `assignment4remote-1.0-SNAPSHOT.war` file to `standalone/deployments` inside the Wildfly home directory.

## 2.2 The client logic

On the client side, the main actors are the servlets for the pages of the simple web application and the Business Delegate, which handles the lookup and invocation of the remote stateless bean and the request of students' information.

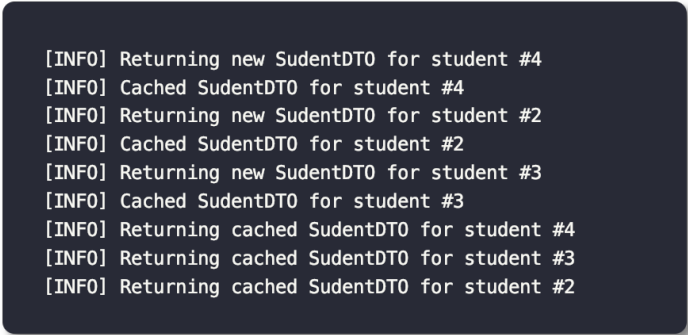
**The job of the Business Delegate** The Business Delegate[1] adds a layer of abstraction between presentation and business logics, helping in the further decoupling of the two. By using the BD pattern

we are also able to encapsulate knowledge about how to locate, connect to, and interact with the remote business object that is the Student Handler bean described earlier.

The **BusinessDelegate** mainly exposes two methods:

- **.getStudentHandler()** is used to perform JNDI lookup through the **ServiceLocator** class and invoke the remote stateless bean **StudentHandlerBean**. The JNDI address of the bean and the properties used to connect to the Wildfly container where the bean is deployed are contained inside this method.
- **.getStudent()** can be considered the main task executed by the Business Delegate: first it instantiates a **StudentHandlerBeanIf** - only if it has not been instantiated before - and then uses *its interface* **.getStudent()** method. In such way, Servlets can simply interrogate the Delegate without having to deal directly with the bean invocation or the lookup.

The **BusinessDelegate** also uses a simple caching system, with an **HashMap** mapping the queried students's numbers and their **StudentDTO**. At each request, the given matriculation number is first checked against said Map, to check if the requested students has already been queried, to reduce the number of calls of the **StudentHandler**.



```
[INFO] Returning new SudentDTO for student #4
[INFO] Cached SudentDTO for student #4
[INFO] Returning new SudentDTO for student #2
[INFO] Cached SudentDTO for student #2
[INFO] Returning new SudentDTO for student #3
[INFO] Cached SudentDTO for student #3
[INFO] Returning cached SudentDTO for student #4
[INFO] Returning cached SudentDTO for student #3
[INFO] Returning cached SudentDTO for student #2
```

Figure 3: An excerpt from the server logs showing the functioning of the described cache systems. At their first call, DTOs for students 2, 3 and 4 are created and then cached, and upon re-call of the same matriculation number cached DTO are returned.

**The Servlets** The logic of the web application is implemented with 3 simple Servlets:

- **HelloServlet** serves as a simple home page with links to the *Student page* and to the *Advisor choice page*.
- **StudentPageServlet** dispatches the **student.jsp** page containing the form for selecting a matriculation number, and calls the Business Delegate to retrieve the DTO for the corresponding student. Then passes the DTO as an attribute so the **.jsp** can display the information needed, being the student name, surname, matriculation, and list of exams with grade, if any.
- **AdvisorChoiceServlet** does the same job as **StudentPageServlet**, but it simply displays the list of teachers from all attended courses.

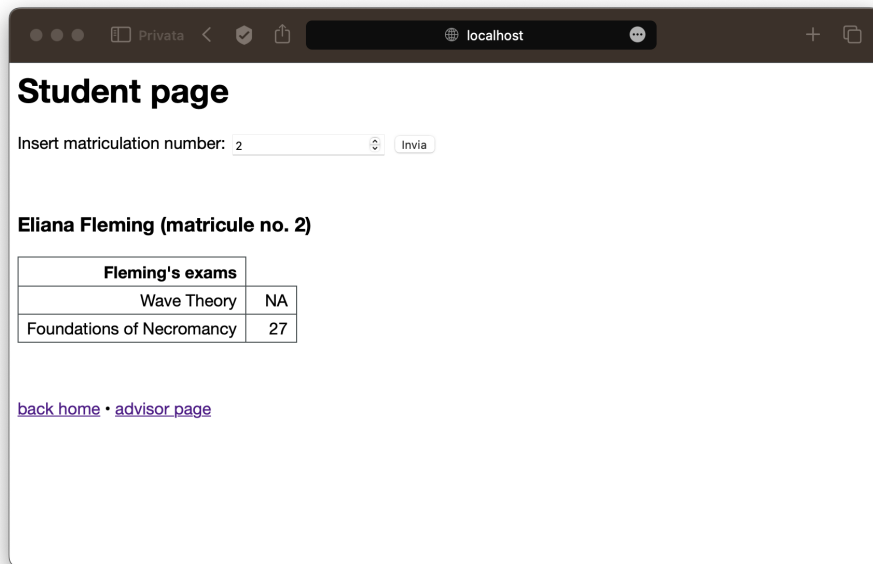


Figure 4: The student page in action. Name and surname of the student with matriculation number 2 are being shown, together with the attended exams and respective grades.

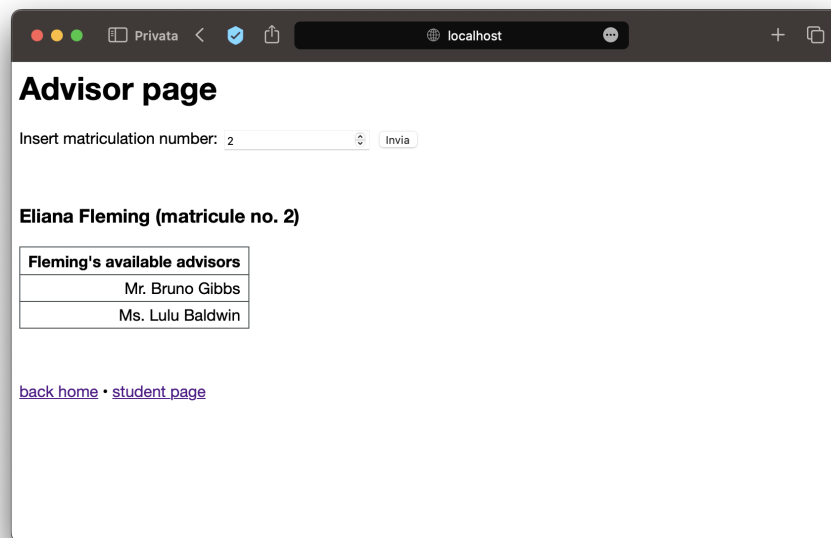


Figure 5: The advisor choice page in action. All available teachers for student with matriculation number 2 are being shown.

## 3 Comments

### 3.1 Possible improvements

**Methods to populate the db** Sticking to the requests of the assignment, the simple web application should simply retrieve data about students, courses and teachers and display them to the user. Methods to populate the database adding new entries to the tables were not implemented. However, the `StudentHandlerBean` could be simply expanded to include new methods which uses transactions to add a new teacher, course or student, to update the student-course bindings in the `ATTENDANCY` table, or maybe to simply add a new grade for a student.

**Actual advisor choice** The idea behind the Advisor choice page should be for each student to select an advisor among those available. This actual choice, as per assignment instructions, has not been implemented, but that could be done by maybe adding a new column inside the `STUDENT` table, to contain the id of the chosen advisor teacher, so that it could also be retrieved at a later time.



## References

- [1] java-design-patterns.com. *Java Business Delegate pattern*.  
<https://java-design-patterns.com/patterns/business-delegate/>.
- [2] www.baeldung.com. *Java DTO pattern*.  
<https://www.baeldung.com/java-dto-pattern>.
- [3] www.baeldung.com. *Java Façade pattern*.  
<https://www.baeldung.com/java-facade-pattern>.
- [4] www.baeldung.com. *JPA Entities*.  
<https://www.baeldung.com/jpa-entities>.
- [5] www.mastertheboss.com. *H2 Database tutorial*.  
<http://www.mastertheboss.com/jbossas/jboss-datasource/h2-database-tutorial/>.
- [6] www.tutorialspoint.com. *H2 Database introduction*.  
[https://www.tutorialspoint.com/h2\\_database/h2\\_database\\_introduction.htm](https://www.tutorialspoint.com/h2_database/h2_database_introduction.htm).
- [7] www.tutorialspoint.com. *Hibernate Query Language*.  
[https://www.tutorialspoint.com/hibernate/hibernate\\_query\\_language.htm](https://www.tutorialspoint.com/hibernate/hibernate_query_language.htm).