



UNIVERSITY OF TRENTO

ASSIGNMENT 5

Vinci Nicolò 220229

nicolo.vinci@studenti.unitn.it

Web Architectures 2021/2022

06 January 2022

Contents

1	Introduction	2
2	Design	3
2.1	Entities	3
2.2	Database	4
2.3	Enterprise Java Beans	5
3	App	7
3.1	Deployment	7
3.2	App running	14
4	Useful links	17

1 Introduction

It needs to develop an application to handle reservations for a series of accommodations which are divided in two categories: hotels and apartments. An hotel is characterized by name, daily price, price for half board, stars and places. An apartment is characterized by name, daily price, price for final cleaning and the maximum number of persons. A user can check the available accommodations giving a start date, an end date and a number of persons. After that, he can pick one of them and check the price for the given period. The price for an hotel is calculated in the following way considering the half board or not:

$$Price = dailyPrice * nPersons * days \quad (1)$$

$$Price = (dailyPrice + halfBoard) * nPersons * days \quad (2)$$

The price for an apartment is:

$$Price = (dailyPrice * days) + finalCleaning \quad (3)$$

Then, the user can insert a name, a surname and a credit card number to confirm the reservation. Moreover, the user can monitor his reservations giving name and surname.

2 Design

The application relies on three main actors:

- A web application deployed in Apache Tomcat.
- JNDI and Enterprise Java Beans deployed in Wildfly.
- An H2 database.

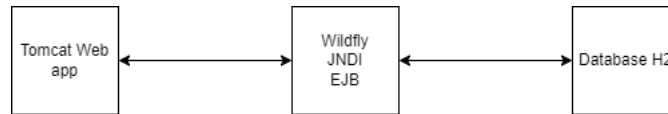


Figure 1: Infrastructure

2.1 Entities

The database structure has been generated from the entities. First of all, the entities and their relationship have been designed. There are seven entities:

- Guest: it needs to store the name and surname of the user who confirms a reservation.
- Accommodation: it is an abstract class to represent the general characteristics of hotel and apartment.
- Hotel: it is derived from the Accommodation entity. It specifies the half board price, places and stars.
- Apartment: it is derived from the Accommodation entity. It specifies the final cleaning price and the maximum number of persons.
- Reservation: it is an abstract class. It is linked with a Guest entity and an Accommodation entity. It defines the attributes for a reservation such as the credit card number, the start date, the end date and the number of persons.
- ReservationHotel: it is derived from the Reservation entity. It stores if the user has chosen the half board.
- ReservationApartment: it is derived from the Reservation entity. It is necessary for the inheritance type described later.

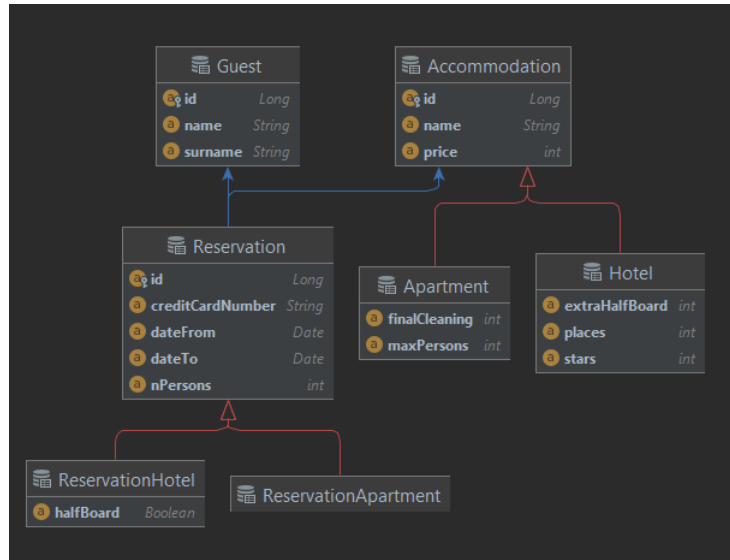


Figure 2: Entity schema

Two different methods have been chosen to map the inheritance relationships in the database. The abstract class `Accommodation` implements the join inheritance. Hence, in the database will be three different tables:

- Base `Accommodation` table.
- `Apartment` table with a foreign key to the `Accommodation` table.
- `Hotel` table with a foreign key to the `Accommodation` table.

Indeed, the abstract class `Reservation` implements the single table per class hierarchy method. Hence, in the database there will be only one table with a column used to distinguish the `ReservationHotel` and the `ReservationApartment`. The `ReservationApartment` is necessary to implement this type of inheritance, even if it does not add any attribute to the abstract class `Reservation`. It has been implemented the single table inheritance to have only one big table for storing the hotel and apartment reservations. Then, the `Reservation` entity is linked to the `Guest` and `Accommodation` entity with a many to one relation on the column `id`. In the database, the reservation table will have a foreign key to the guest table and another one to the accommodation table. After that, the SQL code to develop the database is generated from the entity thanks to JPA buddy.

2.2 Database

The file `db.sql` contains the SQL code to generate the database via the H2 console. The schema is called `WEBARCH`. At the end, the default hotels and apartments are inserted in the database.

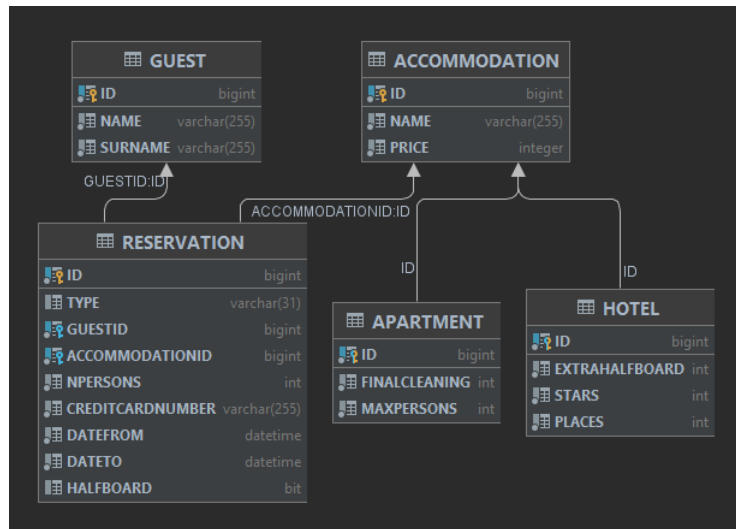


Figure 3: Database schema

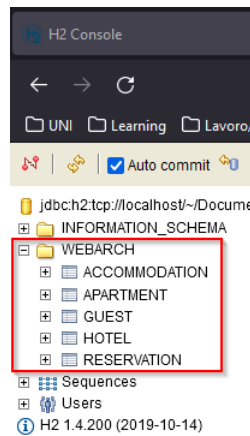


Figure 4: H2 console

Tables have been correctly deployed in the *WEBARCH* schema, as shown in figure 4.

2.3 Enterprise Java Beans

Different EJBs have been developed to encapsulate the business logic of the application. Each of them has the corresponding Java interface and is marked with three Java annotations:

- Remote: the EJB can be reached by a remote client.

- Stateless: the EJB is stateless.
- TransactionManagement(TransactionManagementType.CONTAINER): the container will manage the transactions to the database.

The queries implemented by EJBs follow the CRUD paradigm. The developed EJBs are:

- ApartmentServiceBean: mainly used to get the available apartments given a start date, an end date and a number of persons. It can be used to retrieve one or all apartments. It also computes the reservation price for an apartment.
- GuestServiceBean: mainly used to retrieve a guest given a name and surname. It can be used also to create and delete a guest.
- HotelServiceBean: mainly used to get the available hotels given a start date, an end date and a number of persons. It can be used to retrieve one or all hotels. It also computes the reservation price for an hotel.
- ReservationApartmentServiceBean: mainly used to store a reservation for an apartment. It can also be used to retrieve all apartment reservations given a guest. Moreover, it can return one or all apartment reservations and delete one apartment reservation.
- ReservationHotelServiceBean: mainly used to store a reservation for an hotel. It can also be used to retrieve all hotel reservations given a guest. Moreover, it can return one or all hotel reservations and delete one hotel reservation.
- RoutineServiceBean: it implements the starting routine to fill the hotel and apartment occupancy for the period February 1-28. It can be used to clean the occupancy data.

The transactional behaviour of any bean is the following:

- Create/Delete operation: *Required*.
- Select operation: *Supports*.

3 App

The deployment and the app running will be shown in this section.

3.1 Deployment

First of all, the H2 database has to be created and located. Then, the H2 database needs to be started. Open a shell, go in the *bin* folder of H2 and run:

```
$ java -jar h2-1.4.200.jar
```

A login page will compare, as shown in the figure 5.

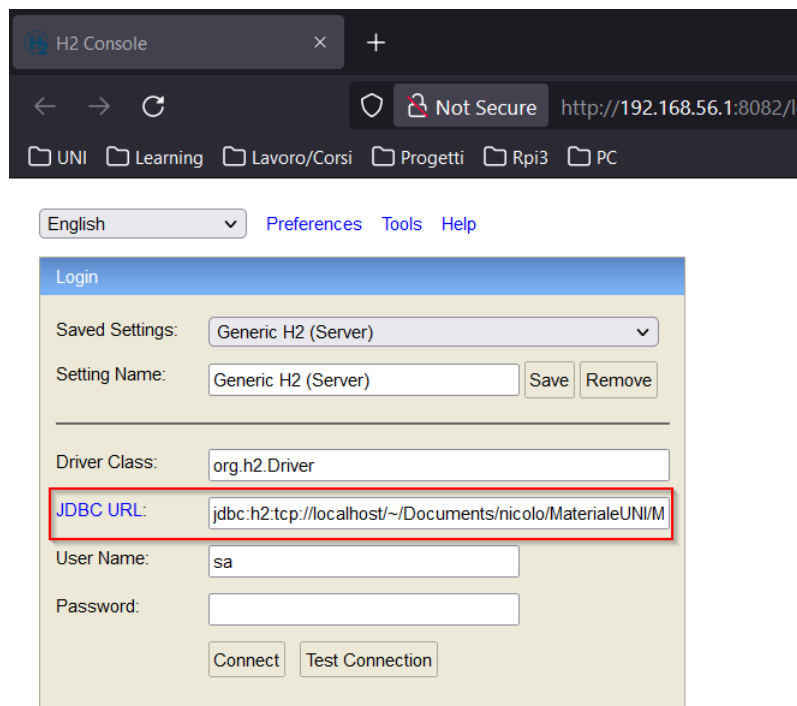


Figure 5: H2 login page

The SQL commands in the script *db.sql* has to be executed in H2 shell in order to create the tables and the initial accommodations for the project. However, the database *Assignment5_Vinci_Nicolo.mv.db* in the folder *database* has already the tables, the initial accommodations and the default guest Mock Guest. The database path has to be specified in the *JDBC URL* field in the login page. The default credentials are *sa* and empty password. After that, the business logic has to be deployed in Wildfly. It is used Wildfly 25.0.1.Final for this project. Firstly, the *standalone.xml* located in *wildfly-25.0.1.Final/standalone/configuration* has

to be modified. In the *datasources* tag, it is needed to add the path to the H2 database and delete the password tag as show in the figure 6.

```
<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="true">
      <connection-url>jdbc:h2:tcp://localhost/~Documents/nicola/MaterialeOIM/Registrazione/anno/semestre/webarchitectures/assignment/Assignments5/Assignments5_Vinci_Nicolo:/connection-url</connection-url>
      <driver>h2</driver>
      <security>
        <user-name>sa</user-name>
      </security>
    </datasource>
    <drivers>
      <driver name="h2" module="com.h2database.h2">
        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
      </driver>
    </drivers>
  </datasources>
</subsystem>
```

Figure 6: Datasources tag in standalone.xml

Then, the EJB_app project can be opened in IntelliJ. The IntelliJ driver for H2 needs to be specified in the file *persistence.xml* located in *resources/META-INF* with the name assigned in the *standalone.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/persistence/2.2">
  <persistence-unit name="Default" transaction-type="JTA">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>

    <class>it.unitn.disi.vinci.entities.Guest</class>
    <class>it.unitn.disi.vinci.entities.Accommodation</class>
    <class>it.unitn.disi.vinci.entities.Hotel</class>
    <class>it.unitn.disi.vinci.entities.Apartment</class>
    <class>it.unitn.disi.vinci.entities.Reservation</class>

    <properties>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.use_sql_comments" value="true"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" />
    </properties>
  </persistence-unit>
</persistence>
```

Figure 7: H2 driver in persistence.xml

An artifact has to be created in order to deploy the EJB_app project in Wildfly. Go in File → Project Structure and create a new *Web Application Exploded* artifact.

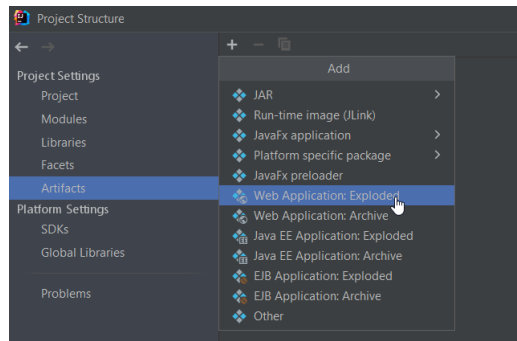


Figure 8: Web Application Exploded artifact

Add to the artifact the project and all the dependencies, except for the descriptors.

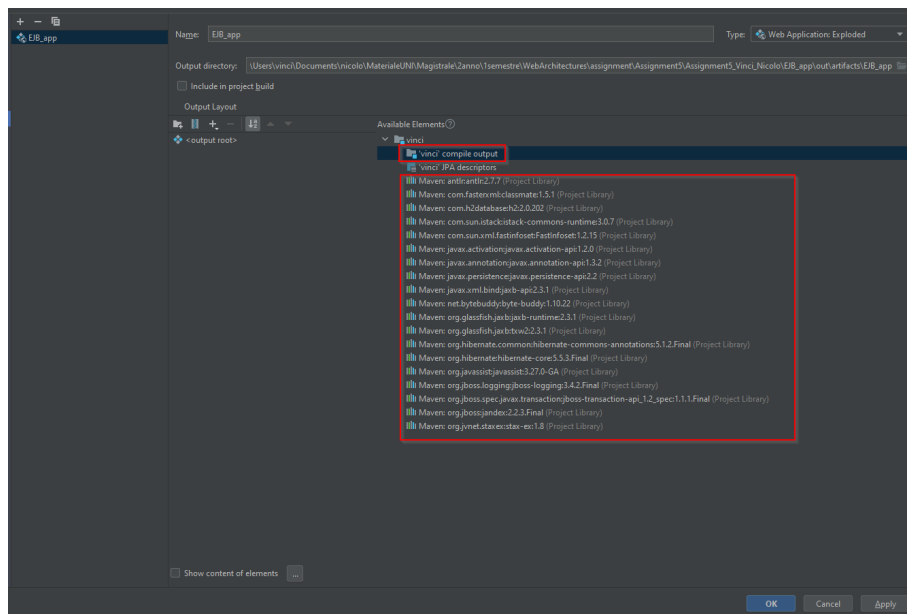


Figure 9: Artifact creation

Then, create a new local Wildfly configuration. The Wildfly path needs to be specified in *configure*.

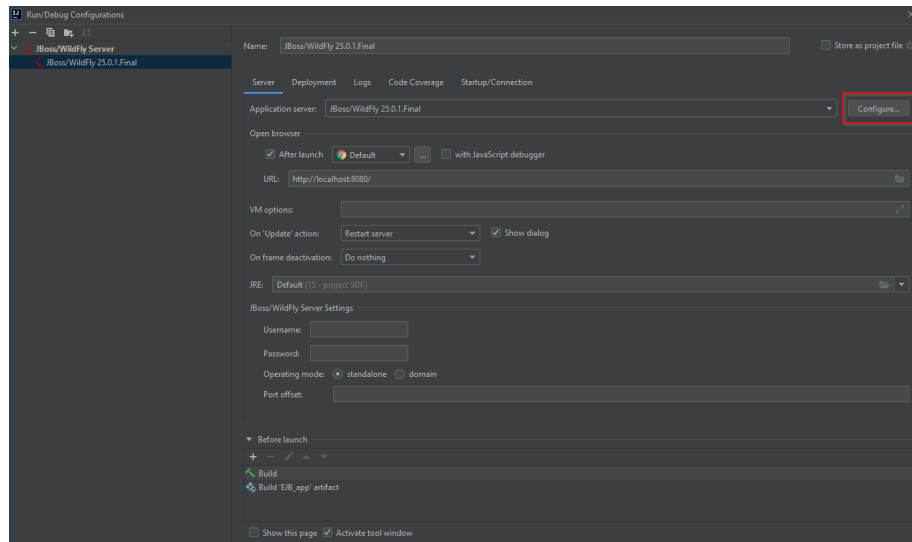


Figure 10: Wildfly configuration

The previous artifact needs to be added in the deploy tab.

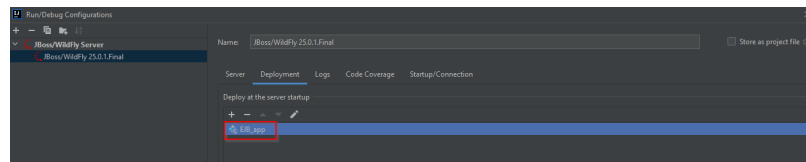


Figure 11: Artifact in the deployment tab

Now, the configuration can be run and out EJB_app project will be deployed in Wildfly. Indeed, the deployment can be checked in the administration console.

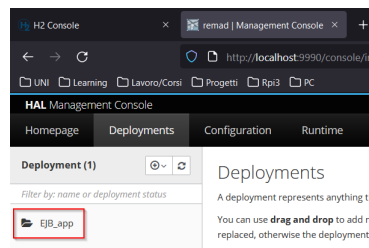


Figure 12: Deployment in Wildfly administration console

The H2 connection can be checked as shown in the figure 13.

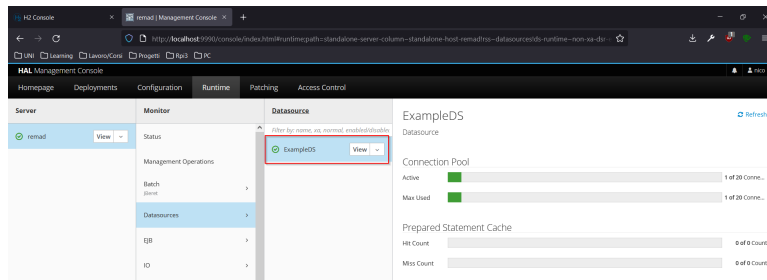


Figure 13: H2 connection in Wildfly administration console

The EJBs and JNDI mapping can also be checked.

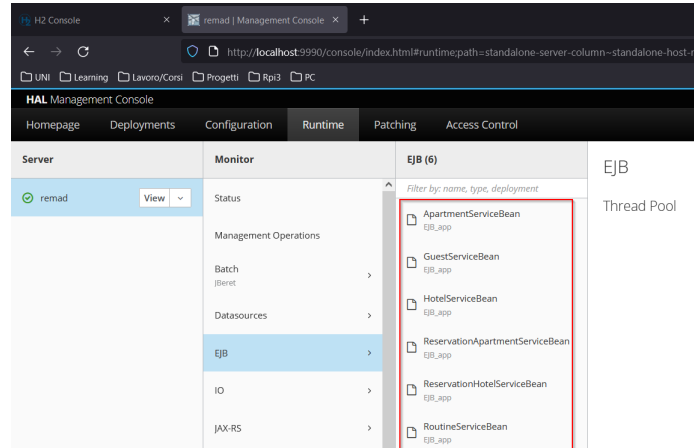


Figure 14: EJBs in Wildfly administration console

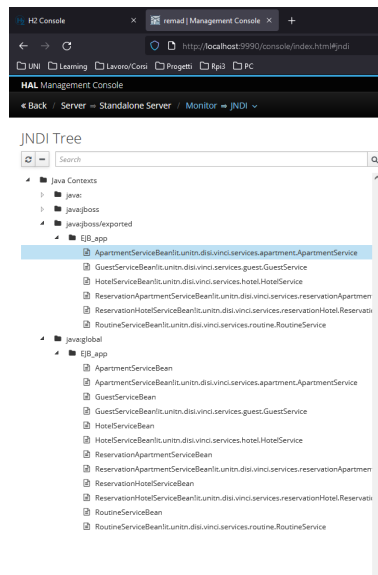


Figure 15: JNDI tree in Wildfly administration console

As last thing, the Tomcat_app needs to be deployed in Tomcat. Open the Tomcat_app project in IntelliJ. The HTTP port of Tomcat needs to be modified, because on port 8080 there is already the Wildfly embedded Tomcat application. Go in the configuration and change the HTTP port to 8000.

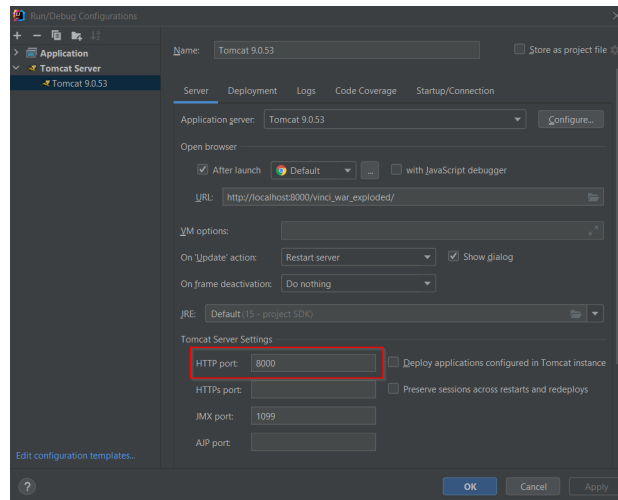


Figure 16: New HTTP Tomcat port

Then, the Wildfly driver needs to be declared in the *pom.xml* in order to communicate to Wildfly from the external Tomcat application.

```
<dependency>
  <groupId>org.wildfly</groupId>
  <artifactId>wildfly-client-all</artifactId>
  <version>25.0.1.Final</version>
</dependency>
```

Figure 17: Driver to communicate to Wildfly from external Tomcat application

Now, the Tomcat app can be deployed. To sum up, the external Tomcat is able to perform JNDI lookup in order to retrieve any EJB deployed in Wildfly and Wildfly is able to persist data in the external H2 database.

3.2 App running

A Listener has been implemented in the Tomcat application. So, when the application starts, it calls the routine to occupy the database with the default guest called Mock Guest exploiting the EJB *RoutineServiceBean*. Instead, the routine to clean the database is called when the Tomcat application will be destroyed.

```
public class Listener implements ServletContextListener {

    public void contextInitialized(ServletContextEvent sce) {

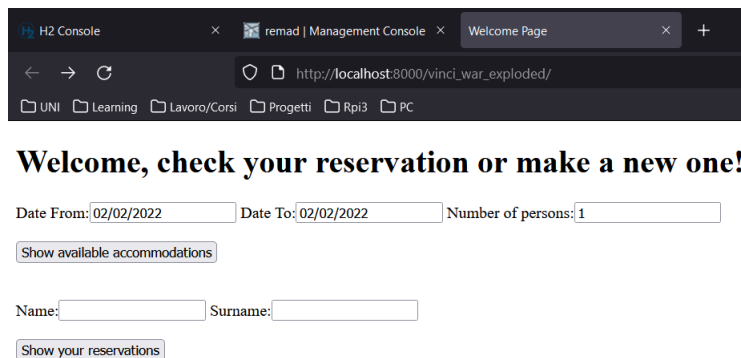
        try {
            ServiceLocator.getInstance().ejbLookup(RoutineService.class).routineHotel();
            ServiceLocator.getInstance().ejbLookup(RoutineService.class).routineApartment();
        } catch (final EntityNotFoundException | ParseException | EntityCRUDEException | EntityInputException | EJBNotFoundException e) {
            System.exit( status: 1);
        }
    }

    public void contextDestroyed(ServletContextEvent sce) {

        try {
            ServiceLocator.getInstance().ejbLookup(RoutineService.class).cleanAll();
        } catch (final EntityNotFoundException | EntityCRUDEException | EJBNotFoundException e) {
            System.exit( status: 1);
        }
    }
}
```

Figure 18: Listener to populate and clean occupancy data

The welcome page is shown in the figure 19.



H2 Console x remad | Management Console x Welcome Page x +

← → ↻ http://localhost:3000/vinci_war_explored/ UN1 Learning Lavoro/Corsi Progetti Rpi3 PC

Welcome, check your reservation or make a new one!

Date From: 02/02/2022 Date To: 02/02/2022 Number of persons: 1

Show available accommodations

Name: Surname:

Show your reservations

Figure 19: Welcome page

Then, the user can insert a start date, an end date and a number of persons to check the available hotels and apartments.

Available Accommodations

Hotels

Pick	Name	Price	Extra Half Board	Stars	Places	
<input type="checkbox"/>	Artemide	100	20	4	60	<input type="checkbox"/> Extra Half Board
<input type="checkbox"/>	Majestic	65	15	3	50	<input type="checkbox"/> Extra Half Board
<input type="checkbox"/>	Zenith	70	18	3	40	<input type="checkbox"/> Extra Half Board

Apartments

Pick	Name	Price	Final Cleaning	Max Persons
<input type="checkbox"/>	Pietra Bianca	40	15	4
<input type="checkbox"/>	Sapore Di Sale	80	20	8
<input type="checkbox"/>	Tenuta Di Artimino	60	12	6

Figure 20: Accommodations page

After that, the user can pick an hotel and the extra half board for example. The resume of the reservation and the price are shown.

Price for accommodation

Date From: Wed Feb 02 00:00:00 CET 2022

Date To: Wed Feb 02 00:00:00 CET 2022

Number of Persons: 1

Price: 120

Credit Card Number: Name: Surname:

Figure 21: Single accommodation page

The user can insert the credit card number, the name and the surname to confirm the reservation.

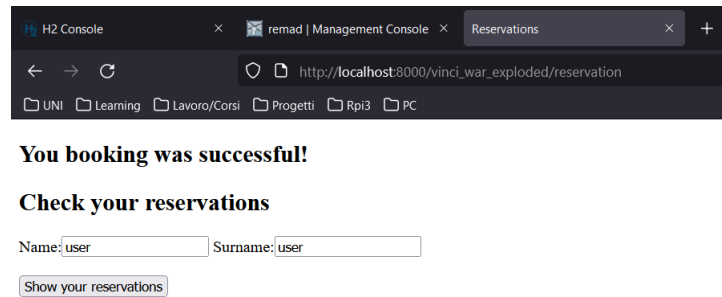


Figure 22: Successful reservation page

Now, the user can insert the previous name and surname to check his reservations.

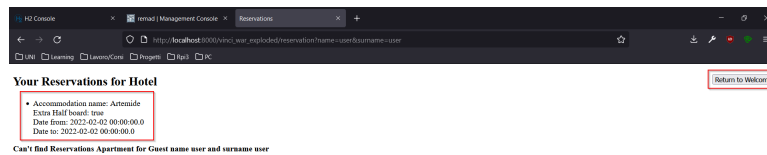


Figure 23: Reservation page

At any moment the user can return back to the welcome page with button *Return to Welcome*. The reservations for the Mock Guest user are the default created by the routine.

4 Useful links

Some links followed to develop the project are reported in this section.

- JPA buddy: <https://www.jpa-buddy.com/documentation/entity-designer/>.
- Hibernate prepared statement: <https://www.baeldung.com/jpa-query-parameters>.
- Join type: <https://www.baeldung.com/jpa-join-types>.