

Web Architectures
assignment 5

Booking Platform - EJB

Sebastiano Chiari - 220527
`sebastiano.chiari@studenti.unitn.it`



January 6, 2022

1 Introduction

The aim of this assignment is to create a booking web application for a tourist location. Two accommodations categories are offered: apartments and hotels, both characterized by name, price and availability. Hotels have also information about half board price, number of stars and number of available places, while apartments have information extra price for final cleaning and maximum number of guests.

The server side is developed through EJB and deployed on a Wildfly server. The database (H2) is not embedded in the Wildfly server and it is added as a datasource in the Wildfly `standalone.xml` file. The client side is deployed outside the Wildfly server on a dedicated Tomcat server.

2 Implementation

2.1 Database

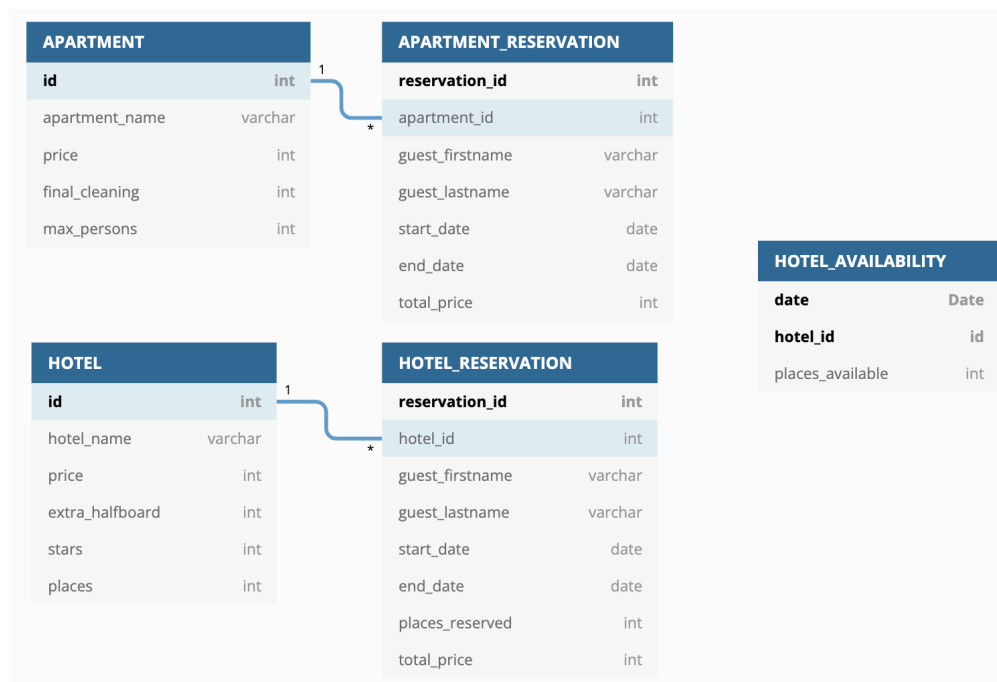


Figure 1: DB schema

APARTMENT and **HOTEL** tables model the two different accommodations type, replicating the same structure given in the guidelines pdf. I decided to model those two categories not as extensions of a common parent class but as single entities: they have different behaviours and there are no specific operations which could be synthesized in the same way on the server side for both classes worth a parent implementation with a more complex DB structure.

APARTMENT_RESERVATION and **HOTEL_RESERVATION** are two similar tables used to store reser-

vations: common fields are `start_date`, `end_date`, `guest_firstname`, `guest_lastname` and `total_price`. `HOTEL_RESERVATION` has an additional field, `places_reserved`, used to store how many places are associated with that reservation. Both tables have their respective `accommodation_id` field, which behaves as a Foreign Key to the respective table, linking the reservation with the chosen accommodation. Also in this case, since they both refer to different classes, I decided to model two different tables instead that a bigger unique one. More, for some queries (such for example a search narrowed down to one of the two different categories), having two tables is quicker and more efficient than having to scroll through a much bigger table.

Another support table, `HOTEL_AVAILABILITY`, modelling the most complex information to obtain otherwise: the number of remaining places within an hotel given a specific date or range of dates. This table has a composite primary key, with the `Date` field and the `hotel_id` one: this ensures that no duplicates can be found and allows for faster search queries. The `places_available` field stores the remaining available places for a given couple hotel and date. When a reservation is made, the corresponding row is updated. If the hotel has maximum capacity for a given day, no row is present within the DB, thus a new entry is added.

2.1.1 Entity mapping

Each table has its own mapping within a corresponding entity.

Both the relationships between the accommodation and the accommodation reservation tables have been modeled as a many-to-one, adding the corresponding `@JoinColumn` descriptor.

```
    @ManyToOne
    @JoinColumn(name = "APARTMENT_ID", referencedColumnName = "ID")
    private ApartmentEntity apartmentId;
```

Since there was no specification about the possibility of adding, updating or removing an accommodation, all the entries has been added manually and the generation strategy for their ID is set as `IDENTITY`.

On the other hand, the reservation tables has a different ID generation strategy: each table has its own sequence used to generate IDs, exploiting the `SEQUENCE` strategy.

```
public class ApartmentReservationEntity {
    @Id
    @Column(name = "RESERVATION_ID")
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "apartment_generator")
    @SequenceGenerator(name="apartment_generator", sequenceName = "APARTMENT_SEQ", allocationSize = 1)
```

The `HOTEL_AVAILABILITY` table, since it has a composite primary key, has the need for a supporting entity, `HotelAvailabilityEntityPK`, which is in charge of handling the primary key for that table.

2.2 Server side

The server side is developed using EJB and EJB patterns.

2.2.1 BEANS and SERVICES

Most of the implemented SessionBeans are Façades, since they implement complex interactions within the application workflow.

- **BookingService** and **BookingServiceWrapper** They are in charge of managing the booking flow. The **BusinessDelegate** client side calls the **BookingServiceWrapper**, passing together a **BookingDTO**. The **BookingServiceWrapper** then calls the **BookingService** respective method, handling the eventuality that the transaction fails and a rollback is performed. The **bookAccommodation** method within the **BookingService** is a transaction which handles all the logic needed to perform the booking operation: given the accommodation type, which can be hotel or apartment, it checks if there is still the availability for the request accommodation (otherwise, it is set the rollback option, intercepted then by the wrapper) and updates accordingly all the DB tables. This method has as **TransactionAttribute** value **REQUIRES_NEW**, in order to always have a new transaction to begin when the bean is called.
- **CheckoutBean** is responsible for retrieving all the needed information in order to build the checkout page and returns them as a **AccommodationDTO**.
- **DBManagerBean**, **@Singleton** class with two routines, called after the bean is constructed, which completely clean the occupancy data and generates random occupancy for apartments and hotels according to the specifications: random hotel occupancy between 90 and 100% for the total places for every single day and 4 random days occupancy for each apartment in the period between February, 1 and February, 28.
- **HotelAvailabilityBean** is a bean used to retrieve a specific hotel availability given a start and an end date and a number of places.
- **ReservationService**, another façade session bean used to retrieve all the reservations given firstname and lastname provided by the user. It returns a list of **ReservationDTO**, encapsulating both hotels and apartments reservations.
- **SearchService** handles the search workflow. Given search type (which could be **ANY**, **APARTMENT** or **HOTEL**, the **BusinessDelegate** calls the respective method to find all the available accommodations according to the dates and the number of guests provided.

Since no specific information regarding the current user needs to be retained server-side, no **@Stateful** beans have been implemented.

2.2.2 DAO

A DAO (Data Access Object) is a pattern that provide an abstract interface to some type of database. By mapping application calls to the persistency layer, the DAO provides some

specific operations without exposing details of the database. More, it helps separating what data access the application needs from the actual implementation.

Each DB entity has its own DAO, which implements basic operations, such as **save** or **getByID**. Each DAO is annotated as a **@Singleton** and bean. Each operation which can not be considered a "fetch query" (but actively edit the DB, for example saving, updating or deleting something) has the **TransactionAttribute** value set to **REQUIRED**: this ensures that these operations will run in a transaction.

More, each specific DAO has some more operations given the kind of data is needed from the application: **getApartmentReservationByIDAndDates** from the **ApartmentReservationDAO**, for example, retrieves all the reservations from the **APARTMENT_RESERVATION** table given a specific apartment ID which may overlap with the provided dates.

2.2.3 DTO

A **DTO** (Data Transfer Object) is used to transfer data between software application subsystems.

The **DTOAssembler** is a singleton bean, instantiated when the application starts up, used as a common factory to create all the different DTOs.

The **AccommodationDTO** is used in order to export from the server side to the client side accommodations, regardless if they are hotels or apartments. This allows to reduce the business logic on the client side needed to handle two different types of entities.

The **ReservationDTO** is another DTO used from the server to the client to export reservations objects, regardless if they are hotels or apartments. Also in this case, it allows to reduce the business logic on the client side.

The **BookingDTO** is used to transfer the necessary information from the client to the server when we have to perform a booking operations. Also in this case, it allows to pack into a single object both hotels and apartments, leaving to the facade within the server to handle the different behaviour.

2.3 Client side

The client side is developed as a Java Web Application deployed on a Tomcat server, with a **ServiceLocator** and **BusinessDelegate** pattern.

The **ServiceLocator** is used to obtain services from the server side and to cache the ones already known. The Service Locator abstracts the API lookup services, vendor dependencies, lookup complexities, and business object creation, and provides a simple interface to clients. It is invoked by the **BusinessDelegate**.

The **BusinessDelegate** is the one in charge of calling the corresponding methods on the server side given the request of data from the client.

2.3.1 Presentation Layer

The **homepage** has a banner on top where the user can input its firstname and lastname to find all the reservation. There's a form where the user can input accommodation type, start date, end date and number of guests, and perform a search. The section below is updated given the results gained, showing a list of available options, with price information and a button for each accommodation to continue to the checkout page.

The **checkout** page is shown to the user after he chooses an accommodation. It shows a recap of the chosen accommodation and presents a form in which the user has to input his firstname, lastname and a credit card to perform the booking operation. In case of an hotel, the user can also choose to tick or untick the extra half board. Once the booking operations is completed, the user is redirected to the homepage, where a pop up is displayed showing the status of the operation (green if the booking has succeeded, red if something went wrong).

The **reservation** page shows the list of all the reservations in the DB given the couple firstname and lastname provided by the user. For each reservation, it is shown the name of the accommodation, the accommodation type, the period and the total price; for hotels it also displayed the number of places reserved.

2.3.2 Servlets

Since the focus of this project was on the business logic implementation of the server side, no validation has been performed over the forms.

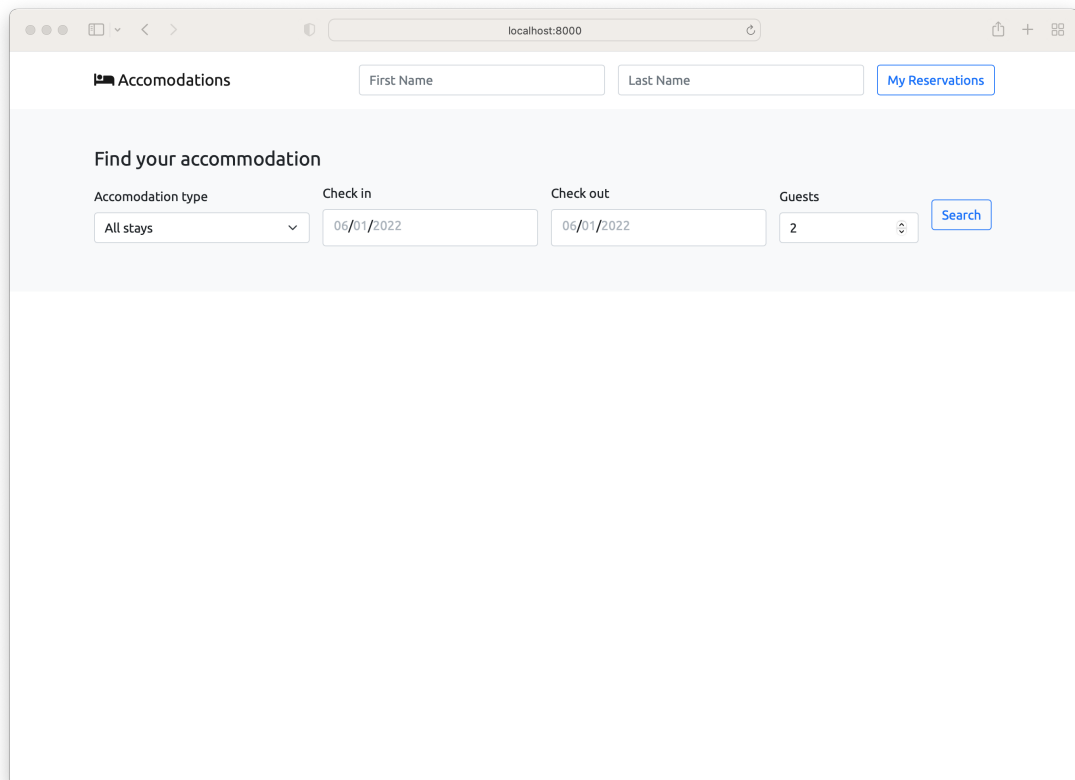
Each servlet runs the corresponding server method through the **BusinessDelegate** to retrieve the needed data to be presented to the user.

- **BookingServlet**
doGet() method is called via Ajax in the `index.jsp` javascript in order to remove the session attribute `transactionStatus` used to display the popup after a booking operation has been performed
doPost() method gathers all the parameters from the checkout form, calls the business delegate to perform the booking operation, sets the `transactionStatus` session variable according to the result of that operation and redirect the user to the index page.
- **CheckoutServlet**
doPost() method gathers all the parameters from the accommodation selected, calls the business delegate to retrieve all the needed information to be displayed in the `checkout.jsp` page.
- **IndexServlet**
doPost() method gathers all the parameters from the "Find your accommodation" form, calls the business delegate to find all the available accommodations and displays them in the `index.jsp` page
- **ReservationServlet**

`doPost()` method gathers all the parameters from the top form, calls the business delegate to retrieve all the reservations given the given firstname and lastname and display the information on the `reservation.jsp` page.

3 Results

In the following section, there are screenshots of the running application with descriptions that documents the various steps and scenarios.



The screenshot shows a web browser window with the address bar displaying `localhost:8000`. The page has a header with the title "Accommodations" and a navigation bar with two input fields labeled "First Name" and "Last Name", and a button labeled "My Reservations". Below the header, there is a section titled "Find your accommodation". This section contains four input fields: "Accommodation type" with a dropdown menu showing "All stays", "Check in" with the date "06/01/2022", "Check out" with the date "06/01/2022", and "Guests" with the number "2". A "Search" button is located to the right of these fields. The main content area below the search form is currently empty.

Accommodations

First Name

Last Name

My Reservations

Find your accommodation

Accommodation type

All stays

Check in

06/01/2022

Check out

06/01/2022

Guests

2

Search

Your results

Apartment

Pietra Bianca

40€ per night

+15€ final cleaning

135 €

Book now

Apartment

Tenuta Di Aritmino


60€ per night

+12€ final cleaning

192 €

Book now

Hotel

Majestic  Hotel

65€ per night (per person)

80€ with half board


starting at 585 €


Book now


Confirm booking

Your reservation

Majestic

 Hotel

 2022-02-04 - 2022-02-06

 3 guests

☒ Extra half board

720€

Billing address

First name

Sebastiano

Last name

Chiari

Credit card number

123412341234

Book your accommodation

