Coding Logic

Overview:
The project "Hotel Room Booking Application " allows users to book rooms in a hotel. This project is a Microservice based project which uses Spring Framework and Maven.

The Project is divided into Three Microservices that are as follows:

Booking Service - This service is exposed to User to collect required information for booking.The user is asked to provide the information about his/her booking request such as number of rooms and number of days of booking with some other relevant information.
After this the user is provided with the Price for the booking. If the user wishes to proceed he is asked for the payment related details. Once a user enters the details, this service call the Payment Service.

Payment Service - The service is called by the Booking Service once the user wishes to confirm the room booking. This service is a dummy service which mocks the Payment for Room. Once the Payment is Complete, the transaction ID is updated.

Eureka Server - This is used as a communication between the other microservices.

Technologies Used:

Spring Framework: Spring framework is used to make sure the application follows Inversion Of Control and Dependency Injection.

Maven : It is used a Build tool. All the required Dependencies for the application are mentioned in the POM.xml file and Maven makes sure to load all the dependencies to local repository from the Central Repository for local use.

MySql: MySQL is configured to use as DBMS.

Workflow Detailed:

Booking Service:
The service requests the following information from the user and saves it to the database.

toDate, fromDate, aadharNumber and the number of rooms and returns the room number list (roomNumbers) and total roomPrice to the user.

The below code generates the random numbers in the booking service

// generate random numbers based on provided total number count

```
public ArrayList<String> getRoomNumbers(int noOfRooms) {
        Random random = new Random();
        int upperBound = 100;

        ArrayList<String> roomsList = new ArrayList<String>();

        for (int i= 0; i<noOfRooms; i++){
            roomsList.add(String.valueOf((random.nextInt(upperBound))));
        }
        return roomsList;
    }
```

Based on the schema definitions of the booking table, BookingInfoEntity class is created. The class is annotated with @Entity for the microservice to identify it as an entity and spring can create a table in the configured database for the same. The getter and setter methods along with toString are also created.

// booking table entity class to map to and from database

```
@Entity
public class BookingInfoEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int bookingId;
    @Column
    private LocalDate fromDate;
    @Column
    private LocalDate toDate;
    @Column
    private String aadharNumber;
    @Column
    private int numOfRooms;
    @Column
    private String roomNumbers;
    @Column(nullable = false)
    private int roomPrice;
    @Column(columnDefinition = "integer default 0")
    private int transactionId;
    @Column
    private LocalDate bookedOn;
```

```java
    public int getBookingId() {
        return bookingId;
    }

    public void setBookingId(int bookingId) {
        this.bookingId = bookingId;
    }

    public LocalDate getFromDate() {
        return fromDate;
    }

    public void setFromDate(LocalDate fromDate) {
        this.fromDate = fromDate;
    }

    public LocalDate getToDate() {
        return toDate;
    }

    public void setToDate(LocalDate toDate) {
        this.toDate = toDate;
    }

    public String getAadharNumber() {
        return aadharNumber;
    }

    public void setAadharNumber(String aadharNumber) {
        this.aadharNumber = aadharNumber;
    }

    public int getNumOfRooms() {
        return numOfRooms;
    }

    public void setNumOfRooms(int numOfRooms) {
        this.numOfRooms = numOfRooms;
    }
```

```java
    public String getRoomNumbers() {
        return roomNumbers;
    }

    public void setRoomNumbers(String roomNumbers) {
        this.roomNumbers = roomNumbers;
    }

    public int getRoomPrice() {
        return roomPrice;
    }

    public void setRoomPrice(int roomPrice) {
        this.roomPrice = roomPrice;
    }

    public int getTransactionId() {
        return transactionId;
    }

    public void setTransactionId(int transactionId) {
        this.transactionId = transactionId;
    }

    public LocalDate getBookedOn() {
        return bookedOn;
    }

    public void setBookedOn(LocalDate bookedOn) {
        this.bookedOn = bookedOn;
    }

    @Override
    public String toString() {
        return "Booking{" +
                "bookingId=" + bookingId +
                ", fromDate=" + fromDate +
                ", toDate=" + toDate +
                ", aadharNumber='" + aadharNumber + '\'' +
                ", numOfRooms=" + numOfRooms +
```

```
                ", roomNumbers='" + roomNumbers + '\'' +
                ", roomPrice=" + roomPrice +
                ", transactionId=" + transactionId +
                ", bookedOn=" + bookedOn +
                '}';
    }
}
```

Repository is an interface which extends JPA Repository, which persists Java Objects into relational databases.

```
public interface BookingDAO extends JpaRepository<BookingInfoEntity,
Integer> {
}
```

DTO class is independently worked as an interface among Repository and service layers. This is to keep the Database some portion of the back end isolated and unexposed. BookingDTO class is used to map the request/response for the /booking POST API.

// DTO class which is used to map Booking detail

```
public class BookingDTO {

    private int bookingId;

    private LocalDate fromDate;

    private LocalDate toDate;

    private String aadharNumber;

    private int numOfRooms;

    private String roomNumbers;

    private int roomPrice;

    private int transactionId;

    private LocalDate bookedOn;

    public int getBookingId() {
        return bookingId;
```

```
    }

    public void setBookingId(int bookingId) {
        this.bookingId = bookingId;
    }

    public LocalDate getFromDate() {
        return fromDate;
    }

    public void setFromDate(LocalDate fromDate) {
        this.fromDate = fromDate;
    }

    public LocalDate getToDate() {
        return toDate;
    }

    public void setToDate(LocalDate toDate) {
        this.toDate = toDate;
    }

    public String getAadharNumber() {
        return aadharNumber;
    }

    public void setAadharNumber(String aadharNumber) {
        this.aadharNumber = aadharNumber;
    }

    public int getNumOfRooms() {
        return numOfRooms;
    }

    public void setNumOfRooms(int numOfRooms) {
        this.numOfRooms = numOfRooms;
    }

    public String getRoomNumbers() {
        return roomNumbers;
```

```
    }

    public void setRoomNumbers(String roomNumbers) {
        this.roomNumbers = roomNumbers;
    }

    public int getRoomPrice() {
        return roomPrice;
    }

    public void setRoomPrice(int roomPrice) {
        this.roomPrice = roomPrice;
    }

    public int getTransactionId() {
        return transactionId;
    }

    public void setTransactionId(int transactionId) {
        this.transactionId = transactionId;
    }

    public LocalDate getBookedOn() {
        return bookedOn;
    }



    public void setBookedOn(LocalDate bookedOn) {
        this.bookedOn = bookedOn;
    }
    @Override
    public String toString() {
        return "BookingInfoEntity{" +
                "bookingId=" + bookingId +
                ", fromDate=" + fromDate +
                ", toDate=" + toDate +
                ", aadharNumber='" + aadharNumber + '\'' +
                ", numOfRooms=" + numOfRooms +
                ", roomNumbers='" + roomNumbers + '\'' +
```

```
                ", roomPrice=" + roomPrice +
                ", transactionId=" + transactionId +
                ", bookedOn=" + bookedOn +
                '}';
    }

}
```

The ErrorDTO class is used to map the error/validation response.
// DTO class which is used to map exception/error

```
public class ErrorDTO {
    private String message;

    private int statusCode ;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public int getStatusCode() {
        return statusCode;
    }

    public void setStatusCode(int statusCode) {
        this.statusCode = statusCode;
    }
}
```

Booking controller is the interface between the presentation layer and the service class. This class has the POST methods which take the request input and send it to the service class and reverse. This class is also responsible for sending the response back in the HTTP form. it converts DTO to Entity and Entity back to DTO and sends either response or error back with appropriate status code.

// POST API to create booking and store into Database
// It will return booking detail data as a response

```
@PostMapping(value = "/booking" , consumes =
MediaType.APPLICATION_JSON_VALUE, produces =
MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity createBooking(@RequestBody BookingDTO
bookingDTO){
        BookingInfoEntity bookedRooms =
bookingService.saveBookingDetails(bookingDTO);
        System.out.println("Booking created successfully!");
        return new ResponseEntity(bookedRooms, HttpStatus.CREATED);
    }
```

At the last, in our Booking service, BookingService interface defined with all declaration of methods which are going to be implemented in BookingServiceImpl class which communicates with Database to store and retrieve data, also communicates to another service called as Payment Service to call it's APi in a synchronous way using restTemplate.

// Booking service Interface

```
public interface BookingService {
    public BookingInfoEntity saveBookingDetails(BookingDTO bookingInfo);

    public BookingInfoEntity getBookingDetailsById(int id);

    public List<BookingInfoEntity> getAllBookings();

    public int calculatePrice(int noOfRooms, int noOfDays);

    public ArrayList<String> getRoomNumbers(int noOfRooms);
}
```

The getBookingDetailsById method takes id as an argument and finds the data into the database, if the data is present with the requested id then it returns the BookingInfoEntity instance, otherwise it throws an error that the booking id is not valid.

```
@Override
    public BookingInfoEntity getBookingDetailsById(int id) {

        if (bookingDAO.findById(id).isPresent()){
```

```
            return bookingDAO.findById(id).get();
        }

        throw new InvalidBooking( "Invalid Payment Id", 400 );

    }
```

All the validation errors / exceptions are mapped in the exception directory and specific exception classes are implemented. InvalidBooking takes care of invalid booking id and InvalidTransaction takes care of payment mode invalidation errors