Coding Logic

Overview:

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

The Project is divided into Four 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 relavanct information.
After this the user is provided with the Price for the booking. If the the user wishes to proceed he is asked for the payment related details. Once user enters the details, this service calls the Payment Service.

Payment Service - The service is called by the Booking Service once user wishes to confim 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.

Notification Service - After the payment is confirmed and the room are booked, this service is responsible to send the user a notification with required details. This Service used Asynchronous Comminication.

Eureka Server - This is used a registery for the other microservices.

Technologies Used:
following technologies have been used in the project:

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 Depenedencies 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.

Amazon RDS: Amazon RDS is used as the Database Service. On RDS, MySQL is configured to use as DBMS. Different Databases are created for different Services.

Namely PAYMENT_SERVICE" for Payment Service which host table "Payment" and "BOOKING_SERVICE" which hosts table "booking"

Hibernate is used as the ORM tool and for the integration with Spring framework, **Spring Data** is used.

Workflow Detailed:

Booking Service:
The service requests the following information from user and save 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 static
ArrayList<String> getRandomNumbers(int count){ Random rand = new Random();
int upperBound = 100;
ArrayList<String>numberList = new ArrayList<String>();
for (int i=0; i<count; i++){
numberList.add(String.valueOf(rand.nextInt(upperBound)));
}
return numberList;
}
```

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(name="booking") @JsonPropertyOrder({"id"}) public class
BookingInfoEntity {
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
@JsonProperty("id") private int bookingId ;
@Column(nullable = true) private Date fromDate;
@Column(nullable = true) private Date toDate;
@Column(nullable = true) private String aadharNumber ; @JsonProperty(access =
JsonProperty.Access.WRITE_ONLY
```

```
) private int numOfRooms ; private String roomNumbers ;
@Column(nullable = false) private int roomPrice ; private int
transactionId=0;
@Column(nullable = true) private Date bookedOn;
}
```

Repository is an interface which extends JPA Repository, which persists Java

Objects into relational databases.

```
@Repository
public interface BookingRepository 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
@JsonPropertyOrder({"id"}) public class BookingDTO {
@JsonProperty("id") private int bookingId ; private Date fromDate; private
Date toDate; private String aadharNumber ;
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY) private int numOfRooms
; private String roomNumbers ;
```

```
private int roomPrice ; private int transactionId=0; private Date bookedOn;
@JsonProperty("id") public int getBookingId() { return bookingId; }
}
```

The another DTO class named BookingTransactionDTO class is used to map for the */booking/{bookingId}/transaction* POST API

```
// DTO class which is used to map booking transaction public class
BookingTransactionDTO {
@NotNull private String paymentMode;
@NotNull private int bookingId; private int transactionId; private String
upiId ; private String cardNumber ;
@JsonProperty("id") public void setTransactionId(int transactionId) {
this.transactionId = transactionId;
}
}
```

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 ;
}
```

On the other hand, KafkaNotificationDTO class is used to map the request after booking successfully completed by the kafka producer which is written in config/KafkaConfig class.

```
// DTO class which is used to map kafka notifications public class
KafkaNotificationDTO { private String topic ; private String key ;
private String value ;
}
```

Config directory inside the Booking service contains the KafkaInfoProducer interface which defines the gerProducer method of apache kafka, following the implementation is written in the KafkaConfig class which overrides the getProducer method and returns the kafka producer instance to BookingService class for further use.

```
public interface KafkaInfoProducer {
Producer<String, String> getProducer() throws IOException;
}
// Kafka Implementation
@Component public class KafkaConfig implements KafkaInfoProducer {
@Override
public Producer<String, String> getProducer() throws IOException { Properties
properties = new Properties(); properties.put("bootstrap.servers",
"ec2-174-129-48-56.compute-1.amazonaws.com:9092"); properties.put("acks",
"all"); properties.put("retries", 0); properties.put("linger.ms", 0);
properties.put("partitioner.class",
"org.apache.kafka.clients.producer.internals.DefaultPartitioner");
properties.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
properties.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
properties.put("request.timeout.ms", 30000); properties.put("timeout.ms",
30000); properties.put("max.in.flight.requests.per.connection", 5);
properties.put("retry.backoff.ms", 5);
//Instantiate Producer Object return new KafkaProducer<>(properties);
}

}
```

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 booking = modelMapper.map(bookingDTO,
BookingInfoEntity.class);
BookingInfoEntity savedBooking = bookingService.makeBooking(booking);
BookingDTO savedbookingDTO = modelMapper.map(savedBooking,
BookingDTO.class); return new ResponseEntity(savedbookingDTO,
HttpStatus.CREATED);
}

// POST API to confirm booking and store into Database as a booking
transaction
// It will return booking transaction detail data as a response
@PostMapping(value="/booking/{bookingId}/transaction", consumes =
MediaType.APPLICATION_JSON_VALUE, produces =
MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity confirmBooking(
@PathVariable(name = "bookingId") int bookingId,
@RequestBody BookingTransactionDTO bookingTransactionDTO) throws
IOException {
BookingInfoEntity savedBooking =
bookingService.updateBookingByCreatingTransaction(bookingId,
bookingTransactionDTO);
String message = "Booking confirmed for user with aadhaar number: " +
savedBooking.getAadharNumber() + " | " + "Here are the booking details: " +
savedBooking.toString();
// kafka notification send after successful transaction
bookingService.sendNotification("message", "notification", message); return
new ResponseEntity(savedBooking, 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 makeBooking(BookingInfoEntity booking); public
BookingInfoEntity getBookingBasedOnId(int id);
public BookingInfoEntity updateBookingByCreatingTransaction(int id,
BookingTransactionDTO bookingTransactionDTO);
public void sendNotification(String topic, String key, String value) throws
IOException;
}
```

These four methods are defined as follows.
The getBookingBasedOnID 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 getBookingBasedOnId(int id) {
// find booking from id into database raise error if id is not found if
(bookingRepository.findById(id).isPresent()){ return
bookingRepository.findById(id).get();
} throw new InvalidBooking( "Invalid Booking Id", 400 );
}
```

The validateTransaction method is used to validate the payment mode, if the payment mode is card/UPI then only API accepts the input otherwise it will throw an error that payment mode is not valid. And, if all validations are passed, it will find the Booking based on id in the database and returns the object.

```
public BookingInfoEntity validateTransaction(int id, BookingTransactionDTO
bookingTransactionDTO){
// validate if paymentMode is not card or not upi then raise an error //
return booking based on id from database if validation succeed
if(bookingTransactionDTO.getPaymentMode() != null){
String paymentMode =
bookingTransactionDTO.getPaymentMode().toLowerCase().strip();
if(!(paymentMode.equalsIgnoreCase("card") ||
```

```
paymentMode.equalsIgnoreCase("upi"))){ throw new InvalidTransaction( "Invalid
mode of payment", 400 ); }
} else {
throw new InvalidTransaction( "Invalid mode of payment", 400 );
}
BookingInfoEntity booking = getBookingBasedOnId(id); return booking;
}
```

The sendNotification method takes the topic name, key and value as an argument and sends the message to kafka topic. The other service named notification service fetches the data from the same topic later.

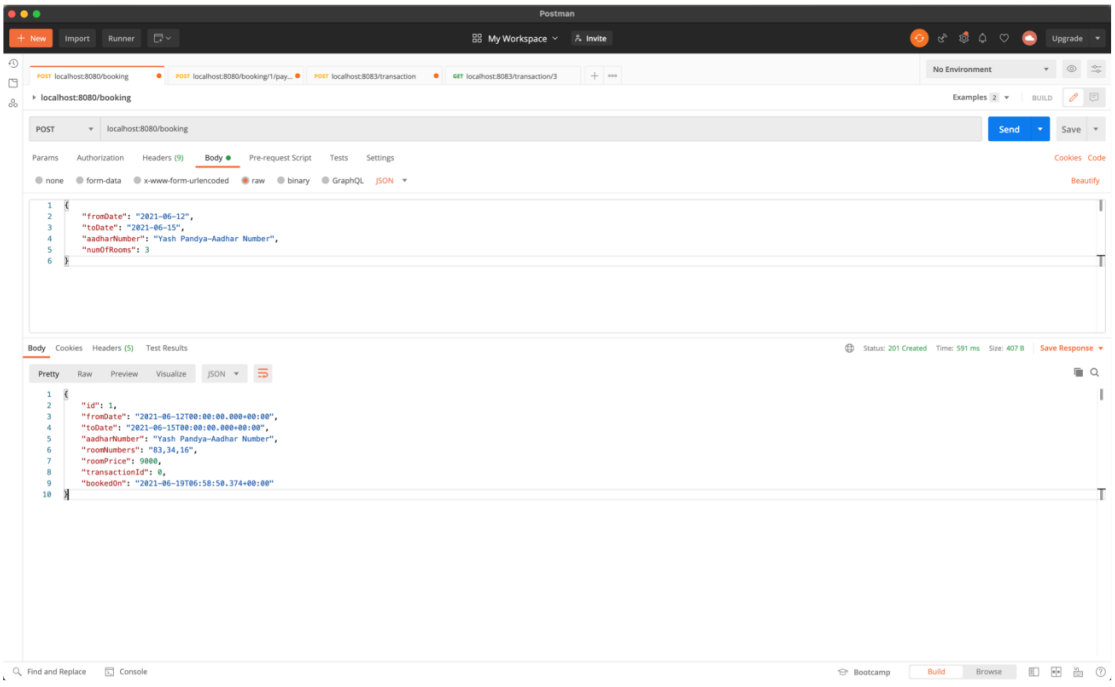
```
@Override
public void sendNotification(String topic, String key, String value) throws
IOException {
// send notification to topic using kafka
Producer<String, String> producer = kafkaConfig.getProducer();
System.out.println(producer.metrics());
// send single message producer.send(new ProducerRecord<String,
String>(topic, key, value));
producer.close();
System.out.println("Notification sent");
}
```

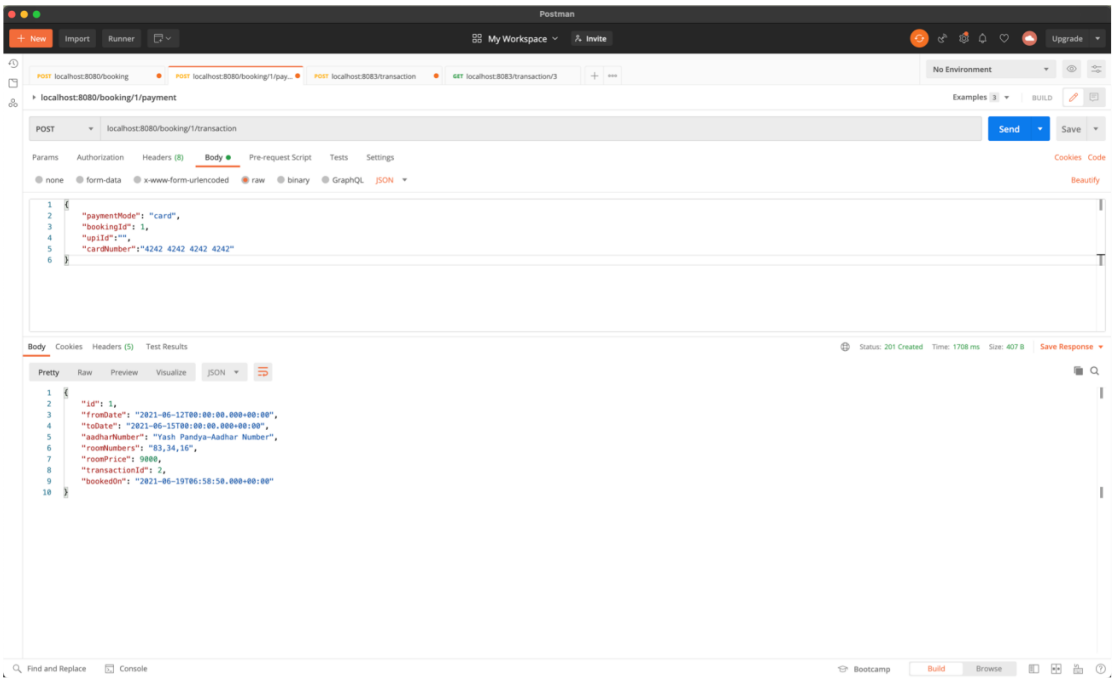
The last method within the class is updateBookingByCreatingTransaction, this method mapped with */booking/{bookingId}/transaction* POST API. This method generates the payment Map and sends the data to the payment service using rest template and updates the transactionId back into BookingInfoEntity (booking table). Method also invokes the validation method to verify the payment mode before proceeding further.

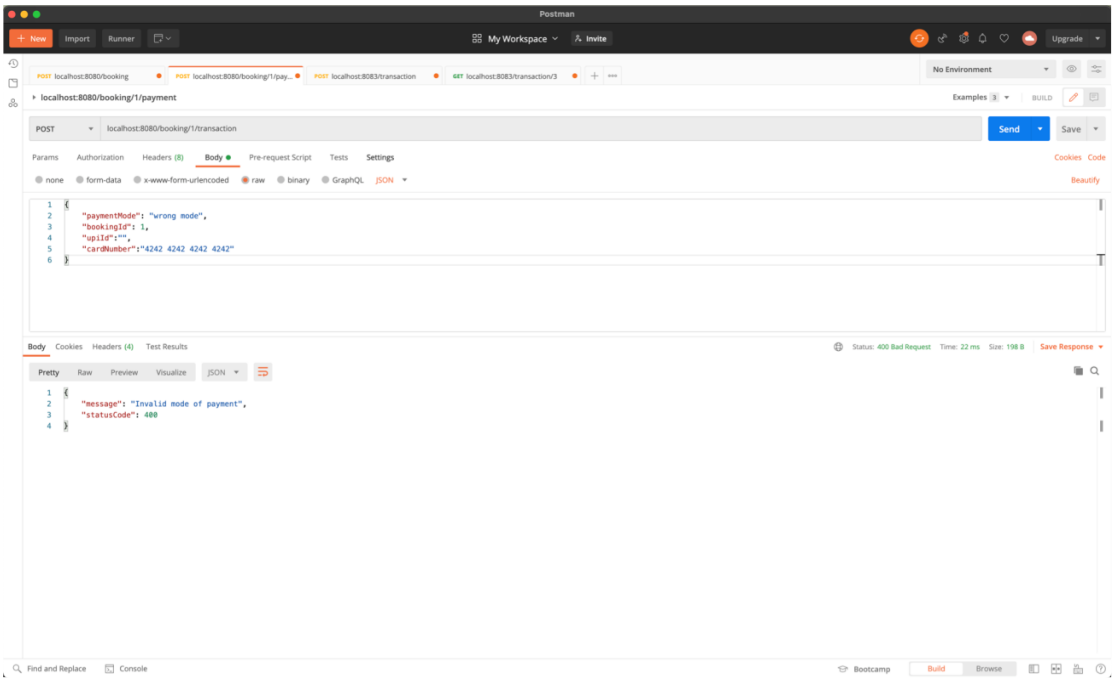
```
@Override
public BookingInfoEntity updateBookingByCreatingTransaction(int id,
BookingTransactionDTO bookingTransactionDTO) {
// create transaction data into payment service
BookingInfoEntity booking = validateTransaction(id, bookingTransactionDTO);
Map<String,String> paymentUriMap = new HashMap<>();
paymentUriMap.put("paymentMode", bookingTransactionDTO.getPaymentMode());
paymentUriMap.put("bookingId",
String.valueOf(bookingTransactionDTO.getBookingId()));
paymentUriMap.put("upiId", bookingTransactionDTO.getUpiId());
paymentUriMap.put("cardNumber", bookingTransactionDTO.getCardNumber());
```

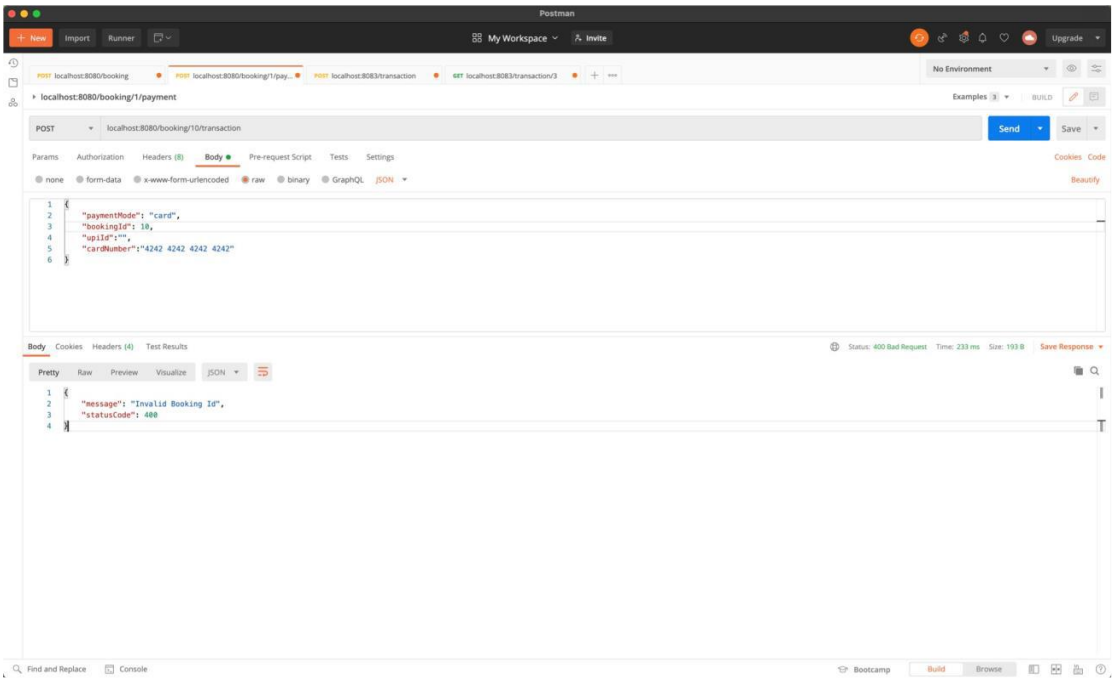
```
// calling payment service API using rest template
BookingTransactionDTO updateBookingTransactionDTO =
restTemplate.postForObject("http://PAYMENT-SERVICE/transaction",
paymentUriMap,
BookingTransactionDTO.class);
// return booking data with updated information
if(updateBookingTransactionDTO != null){
booking.setTransactionId(updateBookingTransactionDTO.getTransactionId());
bookingRepository.save(booking); return booking;
} return null;
}
```

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




Postman
New
Import
Runner
My Workspace
Invite
Upgrade
No Environment
localhost:8080/booking/1/payment
Examples 3
POST
localhost:8080/booking/1/transaction
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"paymentMode": "card",
"bookingId": 1,
"upiId":"",
"cardNumber":"4242 4242 4242 4242"
}
Body
Cookies
Headers (5)
Test Results
Status: 201 Created
Time: 1708 ms
Size: 407 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"fromDate": "2021-06-12T00:00:00.000+00:00",
"toDate": "2021-06-15T00:00:00.000+00:00",
"aadharNumber": "Yash Pandya-Aadhar Number",
"roomNumbers": "83,34,16",
"roomPrice": 9000,
"transactionId": 2,
"bookedOn": "2021-06-19T06:58:50.000+00:00"
}
Find and Replace
Console
Bootcamp
Build
Browse


Postman
New
Import
Runner
My Workspace
Invite
Upgrade
No Environment
localhost:8080/booking/1/payment
Examples 3
POST
localhost:8080/booking/1/transaction
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"paymentMode": "wrong mode",
"bookingId": 1,
"upiId":"",
"cardNumber":"4242 4242 4242 4242"
}
Body
Cookies
Headers (4)
Test Results
Status: 400 Bad Request
Time: 22 ms
Size: 198 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"message": "Invalid mode of payment",
"statusCode": 400
}
Find and Replace
Console
Bootcamp
Build
Browse


Postman
New
Import
Runner
My Workspace
Invite
Upgrade
No Environment
localhost:8080/booking/1/payment
Examples 3
BUILD
POST
localhost:8080/booking/10/transaction
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"paymentMode": "card",
"bookingId": 10,
"upiId":"",
"cardNumber":"4242 4242 4242 4242"
}
Body
Cookies
Headers (4)
Test Results
Status: 400 Bad Request
Time: 233 ms
Size: 193 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"message": "Invalid Booking Id",
"statusCode": 400
}
Find and Replace
Console
Bootcamp
Build
Browse