Sweet Home

FSD C6 Course Project: Hotel Room Booking Application

Submitted by: Akash Kumar Sethy

upGrad-PG Diploma in Software Development - Feb '21

Coding Logic

➢ Database Configuration

I have used the in-memory database h2. By default, Spring Boot configures the application to connect to an in-memory store with the username "sa" and password "password". The in-memory database is volatile, and data will be lost when we restart the application.

Configuration in application.yml file

```yaml
spring:
  jpa:
    hibernate.ddl-auto: create
    generate-ddl: true
    show-sql: true

  datasource:
    url: jdbc:h2:mem:testdb
    driver-class-name: org.h2.Driver
    username: sa
    password: password
  h2:
    console:
      enabled: true
```

➢ Eureka Server Configuration

Port Number: **8761**

Dependencies-

- Spring Cloud Netflix Eureka Server

Eureka Server (in application.yml file)

```yaml
server:
  port: 8761

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
```

For Eureka Clients (BookingService, PaymentService, API Gateway in application.yml file)

```yaml
eureka:
  client:
    fetch-registry: true
    register-with-eureka: true
    service-url:
      defaultZone: http://localhost:8761/eureka

  instance:
    hostname: localhost
```

➢ API-Gateway Configuration

Port Number: **9191**

Dependencies-

- Spring Cloud Netflix Eureka Client
- Spring Boot Actuator

Configuration in application.yml file

```yaml
server:
  port: 9191

spring:
  application:
    name: API-GATEWAY

  cloud:
    gateway:
      routes:
        - id: BOOKING-SERVICE
          uri: lb://BOOKING-SERVICE
          predicates:
            - Path=/hotel/**

        - id: PAYMENT-SERVICE
          uri: lb://PAYMENT-SERVICE
          predicates:
            - Path=/payment/**

    discovery:
      enabled: true
```

➢ Booking Service Configuration

Port Number: **8081**

Dependencies-

- Spring Cloud Netflix Eureka Client
- Spring Boot Web
- Spring Boot Data JPA
- H2 Database

BookingServiceController `@RequestMapping(value = "/hotel")`

Has the following dependencies

```
@Autowired
private BookingService _bookingService;
```

This class has two methods for the two endpoints of Booking Service.

1. createBooking

```
@PostMapping(value = "/booking", produces = MediaType.APPLICATION_JSON_VALUE,
consumes = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<BookingInfoEntity> createBooking(@RequestBody BookingDTO
bookingDTO){
    BookingInfoEntity bookingInfoEntity=
POJOConverter.covertUserDTOToEntity(bookingDTO);
    BookingInfoEntity savedBooking =
_bookingService.createBooking(bookingInfoEntity);
    return new ResponseEntity(savedBooking, HttpStatus.CREATED);
}
```

- Create booking takes Request body BookingDTO which contains values from users for fromDate, todate, aadharNumber and numOfRooms.
- Converts BookingDTO into BookingDetailsEntity with the help of POJOConverter and sends it to service class, which returns BookingDetailsEntity after saving the details to database.

2. updateTransactionId

```
@PostMapping(value = "/booking/{bookingId}/transaction", produces =
MediaType.APPLICATION_JSON_VALUE, consumes = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<BookingInfoEntity>
updateTransactionId(@PathVariable(name="bookingId") int bookingId, @RequestBody
PaymentDTO paymentDTO){
    BookingInfoEntity bookingInfoEntity = _bookingService.updateTransaction(bookingId,
paymentDTO);
    return new ResponseEntity<>(bookingInfoEntity, HttpStatus.CREATED);
}
```

- The second Controller method allow users to do payment. It accepts PaymentDTO along with bookingId for which user wants to make payment.
- Afterwards it validates the paymentMode and calls the Payment Service Endpoint 1 via synchronous communication.
- The Paymnent service endpoint 1 then generates payment and returns the transactionId to BookingService.
- After receiving the transactionId from Payment service, this method then updates the transactionId for respective bookingId and returns the BookingDetailsEntity.

BookingServiceImpl

Has the following dependencies

```
@Autowired
private BookingServiceDao _bookingServiceDao;

@Autowired
private BookingServiceUtils bookingServiceUtils;

@Autowired
RestTemplate restTemplate;

@Value("${paymentApp.url}")
private String paymentServiceUrl;
```

The URL to do synchronous communication with PaymentService using RestTemplate

```
paymentApp:
  url: http://PAYMENT-SERVICE/payment/transaction
```

This class contains the following two **utility** methods from BookingServiceUtils

- getRandomNumbers generates random numbers between 1 to 100 for number of room requested by users via Endpoint1.
- validateDate validates the date format given by user.
- validateFromToDate validate if from date is not greater than to date.
- getNumberOfDays calculates number of days from fromDate to toDate.
- checkPaymentMethod checks if the payment method is correct or not(Payment should be “UPI” or “CARD”).

This class contains the following two **exceptions** from CustomExceptionHandler

- DateRangeException
- RecordNotFoundException
- InvalidPaymentException

This class contains the following two **service** methods

1. createBooking
   - Takes fromDate, toDate, aadharNumber, numOfRooms from BookingDetailsEntity from Controller class.
   - Afterwards it validates dates, calculates Room Price, gets random room numbers from getRandomNumbers and then updates the values to the received BookingDetailsEntity.
   - Sets bookedOn to now (), transactionId is set to 0 by default.
2. acceptPaymentDetails
   - Validates paymentMode and throws InvalidPaymentException if the paymentMode is anything except for "UPI" or "CARD".
   - Uses bookingId to find the BookingInfoEntity stored in the database. Throws RecordNotFoundException if not found.
   - Uses restTemplate to call Payment Service through API Gateway. Sets the transactionId. Prints booking confirmation message on console. And returns the BookingInfoEntity with updated transactionId.

➢ Payment Service Configuration

Port Number: **8083**

Dependencies-

- Spring Cloud Netflix Eureka Client
- Spring Boot Web
- Spring Boot Data JPA
- H2 Database

PaymentServiceController `@RequestMapping(value = "/payment")`

Has the following dependencies

```
@Autowired
PaymentService _paymentService;
```

This class has two methods for the two endpoints of Payment Service.

1. makePayment

```
@PostMapping(value = "/transaction", produces = MediaType.APPLICATION_JSON_VALUE,
consumes = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Integer> makePayment(@RequestBody TransactionDTO transactionDTO){
    TransactionDetailsEntity
transactionDetailsEntity=POJOConverter.transactionDtoToEntity(transactionDTO);
    Integer transactionId=_paymentService.makeATransaction(transactionDetailsEntity);
    return new ResponseEntity<>(transactionId, HttpStatus.CREATED);
}
```

- Takes Request body TransactionDTO(Same as PaymentDTO from BookingService) which contains values from users for paymentMode, bookingId, upiId and cardNumber.
- Converts TransactionDTO into TransactionDetailsEntity with the help of POJOConverter and sends it to service class, which returns transactionId after saving the details to database.
- This End point is used by BookingService Endpoint 2 to accept payment from user and returning the transactionId to the User.

2. getTransactionDetails

```
@GetMapping(value = "/transaction/{transactionId}", produces =
MediaType.APPLICATION_JSON_VALUE, consumes = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<TransactionDetailsDTO> getTransactionDetails(@PathVariable int
transactionId){
    TransactionDetailsEntity transactionDetailsEntity
=_paymentService.getTransactionById(transactionId);
    TransactionDetailsDTO
transactionDetailsDTO=POJOConverter.transactionDetailsEntitytoDTO(transactionDetailsEnt
ity);
    return new ResponseEntity<>(transactionDetailsDTO, HttpStatus.OK);
}
```

- Accepts transactionId from user and sends it to service class.
- Service class gets the transaction details from database by using the transactionId and returns the TransactionDetailsEntity.
- POJOConverter converts TransactionDetailsEntity into TransactionDetailsDTO and send it as Response Entity.

PaymentServiceImpl

Has the following dependencies

```
@Autowired
PaymentServiceDao _paymentServiceDao;
```

This class contains the following two **exceptions** from CustomExceptionHandler

- RecordNotFoundException

This class contains the following two **service** methods

1. makeATransaction
   - Accepts TransactionDetailsEntity from Controller class and returns transactionId after saving the details to database.
2. getTransactionById
   - Gets the transaction details from database by using the transactionId and returns the TransactionDetailsEntity
   - Throws RecordNotFoundException if not found the Id in database.