

Sweet-Home

Eureka

<http://localhost:8761/>

Instances currently registered with Eureka

Booking Service

<http://localhost:8080/swagger-ui/index.html>

Created a Booking Controller and exposed two Post Api's and one Get Api and registered booking service with Eureka server by using Eureka Client.

This service is responsible for taking input from users like- toDate, fromDate, aadharNumber and the number of rooms required (numOfRooms) and save it in its RDS database. This service also generates a random list of room numbers depending on 'numOfRooms' requested by the user and returns the room number list (roomNumbers) and total roomPrice to the user. The formulae to calculate room price is as follows:

- **Create Booking [POST]**

```
curl --location --request POST 'http://localhost:8080/booking' \
--header 'Content-Type: application/json' \
--data-raw '{
    "fromDate": "2018-01-05T11:59:11.332Z",
    "toDate": "2018-02-07T11:59:11.332Z",
    "aadharNumber": "Kolp Polp ",
    "numOfRooms": 11
}'
```

In this service we pass following parameters and get the success result set Json response
Create an **BookingEntityInfo** entity to generate the Data Table structure in RDS database with bookingID is primary and unique key which can has auto incremental bookingID
To handle the exceptions Created a ControllerAdviser to return the appropriate error messages no such unreadable errors.

```
/*
 *This Method create booking records
 * @param BookingInfoEntity
 * @return BookingInfoEntity
 * */
```

```

    public BookingInfoEntity createBooking(BookingInfoEntity
bookingInfoEntity) {
        System.out.println(bookingInfoEntity.getFromDate() + "\t" +
bookingInfoEntity.getToDate());

        long noOfDays =
ChronoUnit.DAYS.between(bookingInfoEntity.getFromDate(),
bookingInfoEntity.getToDate());
        bookingInfoEntity.setRoomPrice( 1000 *
bookingInfoEntity.getNumOfRooms() * ((int)noOfDays) );

        bookingInfoEntity.setRoomNumbers(getRandomNumber(bookingInfoEntity.getNum
OfRooms()));
        bookingInfoEntity = bookingRepository.save(bookingInfoEntity);
        return bookingInfoEntity;
    }

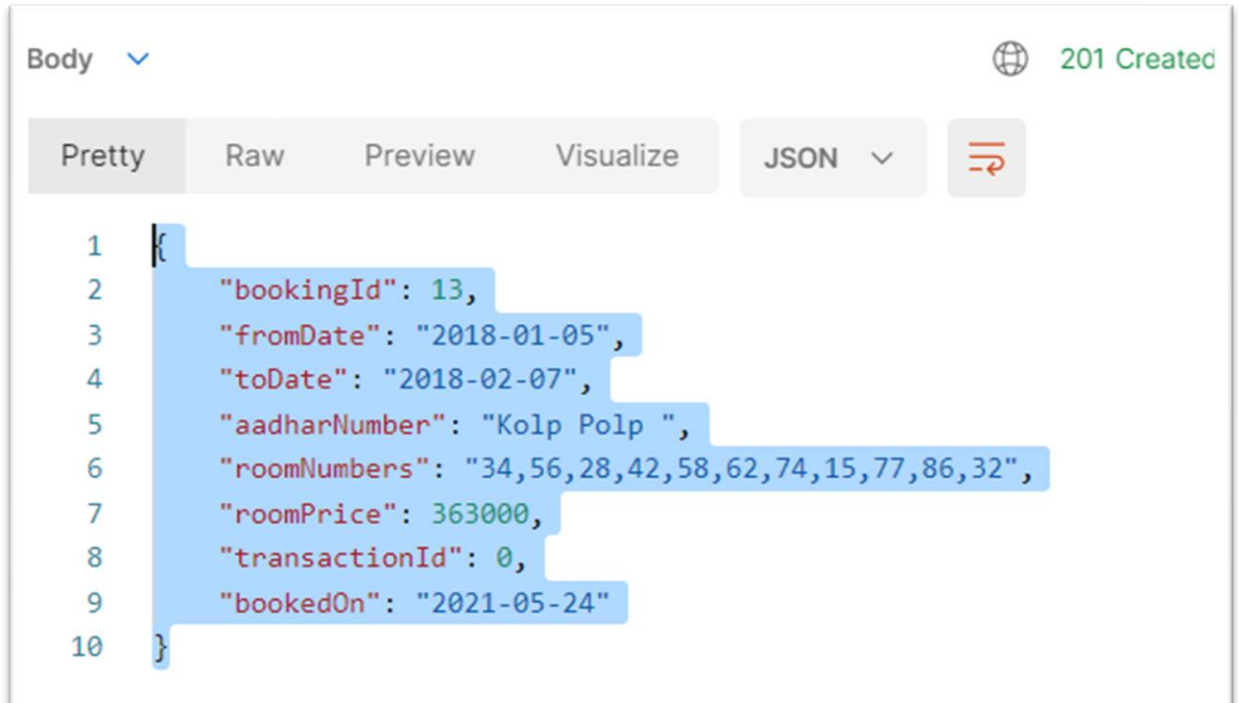
```

To generate a Room number used **getRandomNumber()** function to generate the random number according to the input user provided.

Room Price is currently fixed to 1000/day for a single room and on that basis we calculate price for number of room required and for how many days.

➤ $\text{roomPrice} = 1000 * \text{numOfRooms} * (\text{number of days})$

Then pushed the all records to the RDS database and return the complete booking entity with bookingID. The value of the transactionId returned is 0. It means that no transaction is made for this booking. Once the transaction is done, the transactionId field in the booking table will get replaced with the transactionId received from the Payment service



- **Update transaction id [POST]**

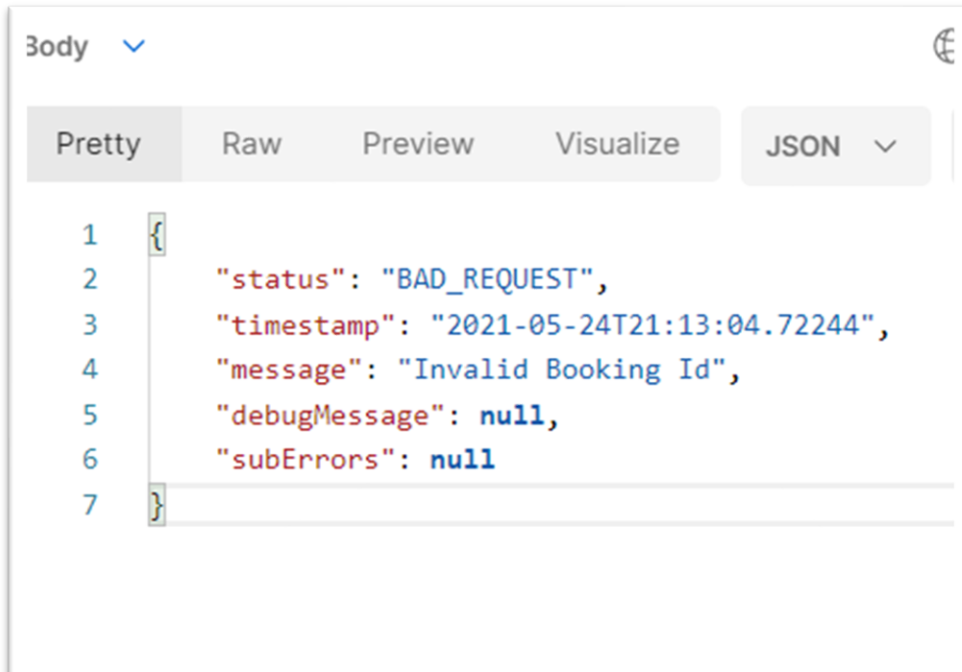
```

curl --location --request POST 'localhost:8080/booking/1022/transaction' \
--header 'Content-Type: application/json' \
--data-raw '{

```

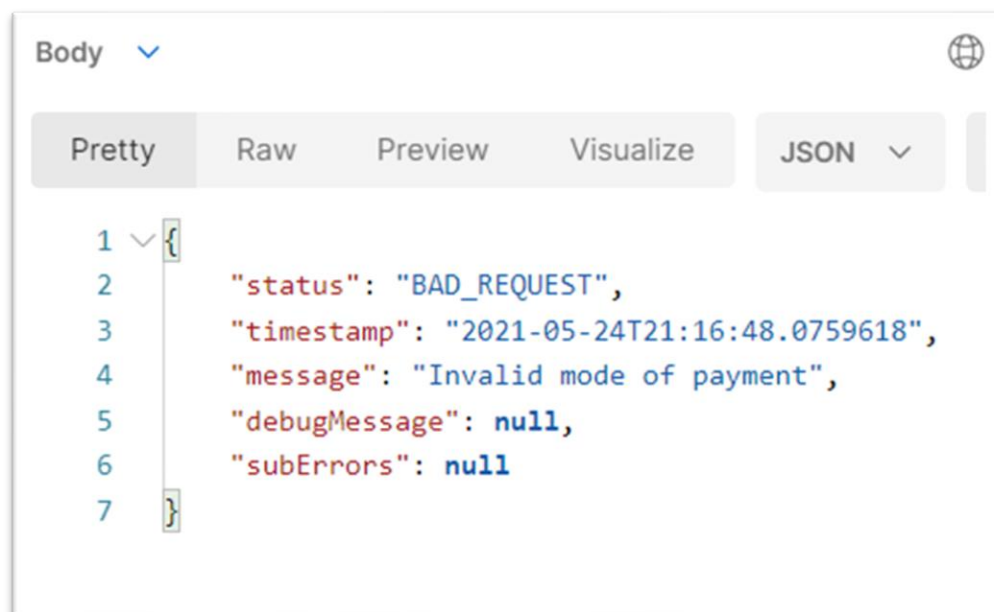
```
"paymentMode": "Card",  
"bookingId": 1022,  
"upiId": "MyUPIId",  
"cardNumber": "Test Card 2"  
}'
```

First it will check and validate the request the request data like if the bookingID is not exists then it will throw Bad_Request exception and return below response



The screenshot shows a web application interface with a 'Body' tab selected. Below the tab are four buttons: 'Pretty', 'Raw', 'Preview', and 'Visualize'. To the right of these buttons is a 'JSON' dropdown menu. The main area displays a JSON object in a pretty-printed format. The object has the following structure:
1 {
2 "status": "BAD_REQUEST",
3 "timestamp": "2021-05-24T21:13:04.72244",
4 "message": "Invalid Booking Id",
5 "debugMessage": null,
6 "subErrors": null
7 }

If bookingID found then it will check for the paymentMode if the paymentMode is not ("UPI" or "Card") then it will throw the Invalid payment mode exception



The screenshot shows a web application interface with a 'Body' tab selected. Below the tab are four buttons: 'Pretty', 'Raw', 'Preview', and 'Visualize'. To the right of these buttons is a 'JSON' dropdown menu. The main area displays a JSON object in a pretty-printed format. The object has the following structure:
1 {
2 "status": "BAD_REQUEST",
3 "timestamp": "2021-05-24T21:16:48.0759618",
4 "message": "Invalid mode of payment",
5 "debugMessage": null,
6 "subErrors": null
7 }

If all the above condition is fulfilled then it will call the Transaction API endpoint with transaction request in body request

```
curl --location --request POST 'localhost:8083/transaction' \
--header 'Content-Type: application/json' \
--data-raw '{
    "paymentMode": "Carad",
    "bookingId": 2,
    "upiId": "MyUPIId",
    "cardNumber": "Test Card 2"
}'
```

```
/**
 * callPaymentServiceApi method is used to pass the transaction json
raw data to the Payment api
 * then it will fetch the transaction api in response
 *
 * @param Transaction modal will be passed in raw body as json
 * **/
private Transaction callPaymentServiceApi(Transaction paymentDetails)
{
    HttpHeaders headers = new HttpHeaders();
    headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
    HttpEntity<Transaction> entity = new
HttpEntity<Transaction>(paymentDetails, headers);

    return restTemplate.exchange(
        "http://localhost:8083/transaction", HttpMethod.POST,
entity, Transaction.class).getBody();
}
```

And fetch the transactionID from the transaction api and then save it in Booking table after saving the record we call the kafka Producer method to create a Topic and push the message to the kafka Consumer, method to push the message

```
/**
 * This help us to produce the Kafka Messages on EC2 kafka server
 *
 * @param sendMessageCount is used to set the iteration for the
message, means how many time you wanted this message to be
 * repeated
 *
 * @param Message: bookingentityinfo toString details will be passed
in this as a successfull response after updating the
 * transaction id in Booking Table
 * **/
public static void runProducer(final int sendMessageCount, String
message) throws InterruptedException {

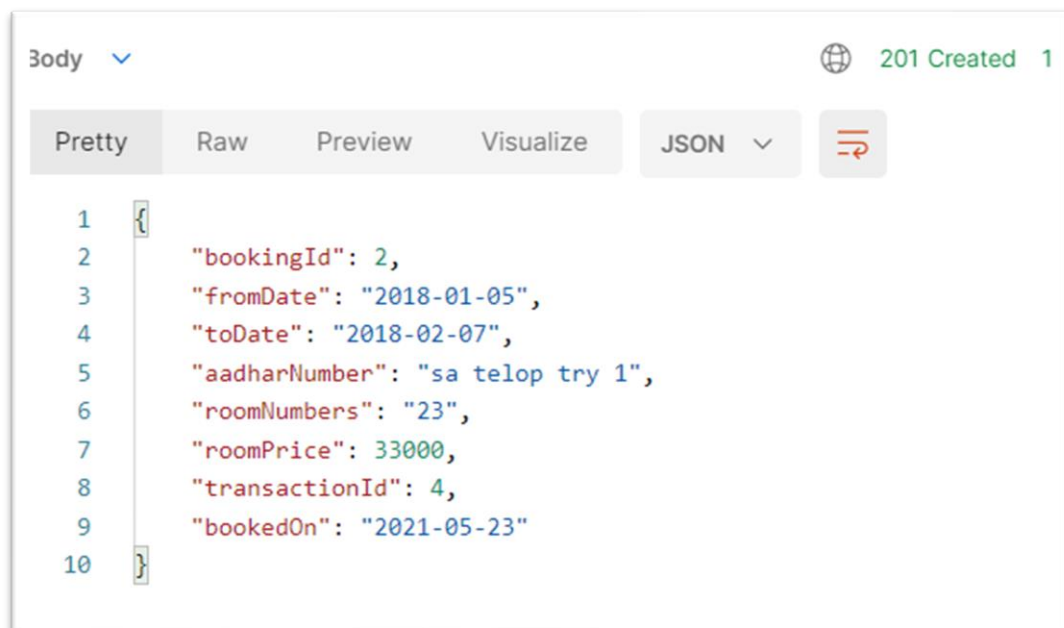
    final CountDownLatch countDownLatch = new
CountDownLatch(sendMessageCount);
    try (Producer<Long, String> producer =
Configuration.createProducer()) {
        long time = System.currentTimeMillis();
        for (long index = time; index < time + sendMessageCount;
index++) {
```

```

        final ProducerRecord<Long, String> record =
            new ProducerRecord<>(Configuration.TOPIC,
index, message);
        producer.send(record, (metadata, exception) -> {
            long elapsedTime = System.currentTimeMillis() -
time;
            if (metadata != null) {
                System.out.printf("sent record(key=%s
value=%s) " +
                                "meta(partition=%d, offset=%d)
time=%d\n",
                                record.key(), record.value(),
metadata.partition(),
                                metadata.offset(), elapsedTime);
            } else {
                exception.printStackTrace();
            }
            countDownLatch.countDown();
        });
        countDownLatch.await(25, TimeUnit.SECONDS);
    }
}

```

return the below successful response and also successfully pushed the message to the kafka topic



The screenshot shows a REST client interface with a 'Body' tab selected. The response is a JSON object with the following fields: bookingId (2), fromDate (2018-01-05), toDate (2018-02-07), aadharNumber (sa telop try 1), roomNumbers (23), roomPrice (33000), transactionId (4), and bookedOn (2021-05-23). The status bar at the top right indicates '201 Created' and '1' item.

```

1  {
2      "bookingId": 2,
3      "fromDate": "2018-01-05",
4      "toDate": "2018-02-07",
5      "aadharNumber": "sa telop try 1",
6      "roomNumbers": "23",
7      "roomPrice": 33000,
8      "transactionId": 4,
9      "bookedOn": "2021-05-23"
10 }

```

Payment Service

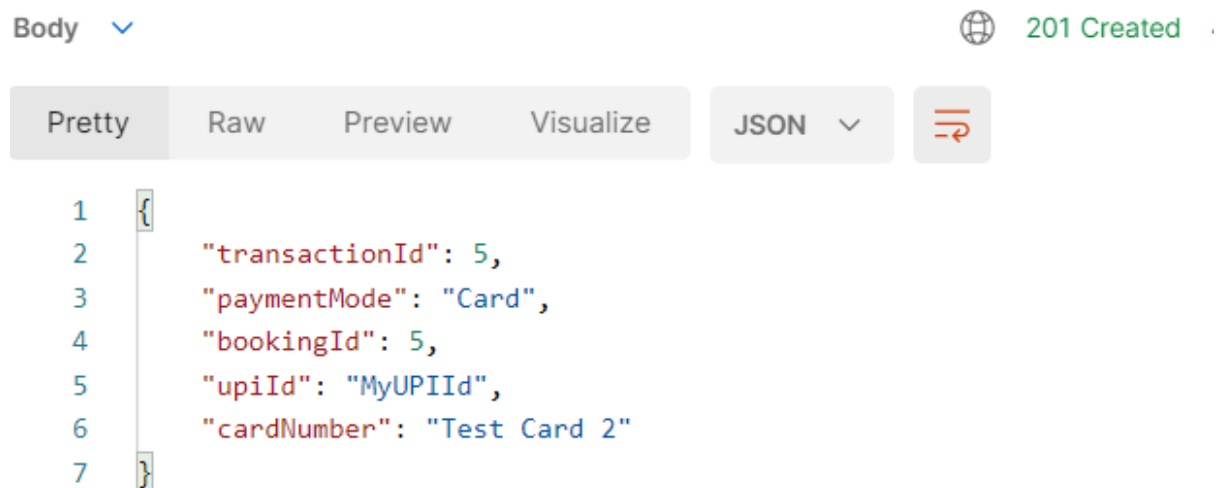
<http://localhost:8083/swagger-ui/index.html>

This service is responsible for taking payment-related information- paymentMode, upid or cardNumber, bookingId and returns a unique transactionId to the booking service. It saves the data in its RDS database and returns the transactionId as a response.

- Created payment controller to generate a transaction id with the requested json body payload

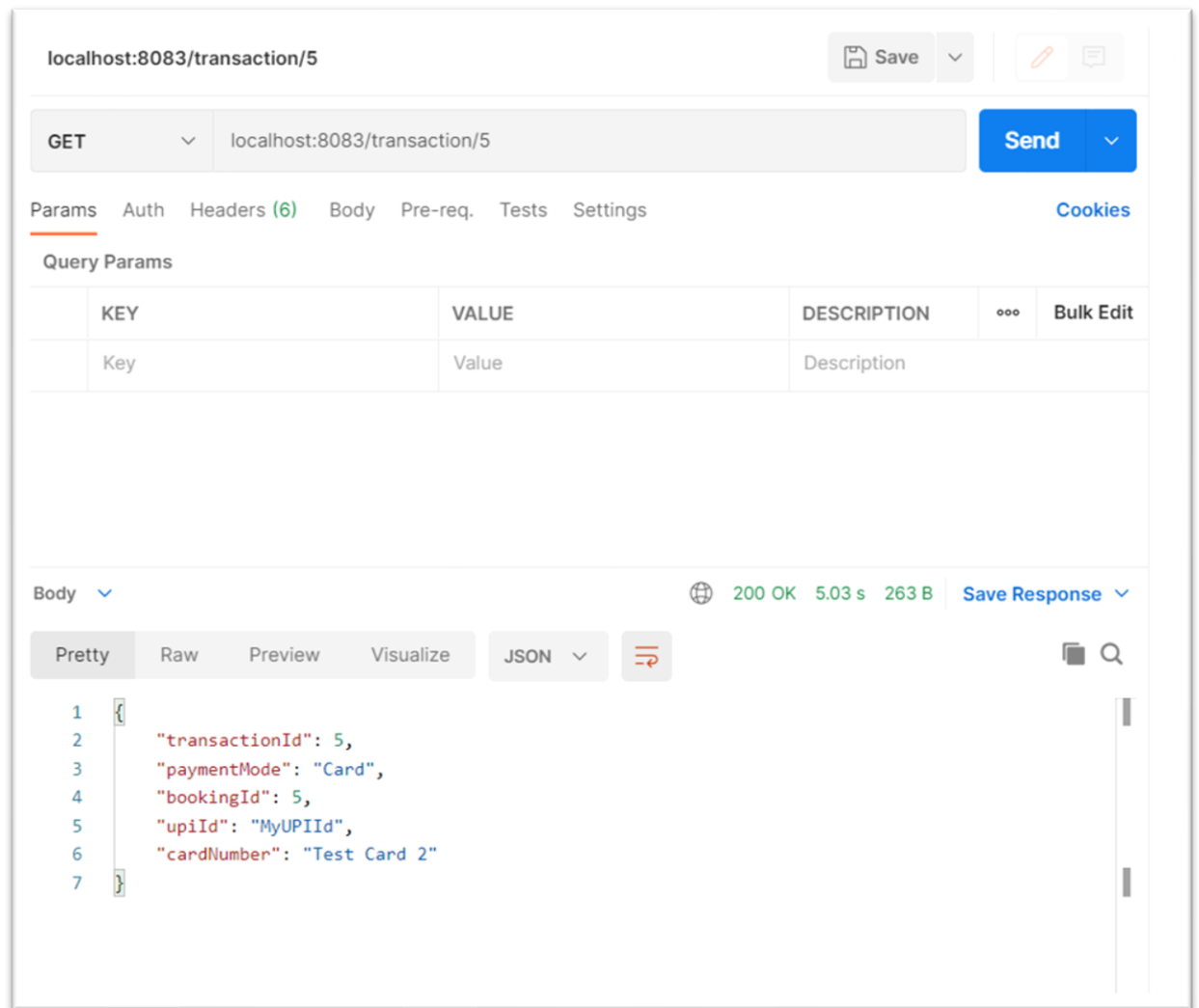
```
curl --location --request POST 'localhost:8083/transaction' \  
--header 'Content-Type: application/json' \  
--data-raw '{  
  "paymentMode": "Card",  
  "bookingId": 5,  
  "upiId": "MyUPIId",  
  "cardNumber": "Test Card 2"  
}'
```

And save the payload into the RDS database Transaction database and in response it will give the transaction entity with transaction id.



- Get api to fetch the transaction details from transaction table with below request

```
curl --location --request GET 'localhost:8083/transaction/5'
```



Notification Service

This service consumes the messages published by the Booking service on Kafka and prints the same on the console. This service will be created while establishing asynchronous communication using Kafka.

- It will work as consumer for the kafka topic can consume the message and display it to console log Async manor

- Below is configuration for kafka properties

```

public class Consumerloop implements Runnable {
    private final KafkaConsumer<String, String> consumer;
    private final List<String> topics;
    private final int id;

    public Consumerloop(int id, String groupId, List<String> topics) {
        this.id = id;
        this.topics = topics;
        Properties props = new Properties();
        props.put("bootstrap.servers", "34.229.70.228:9092");
        props.put("group.id", groupId);
        props.put("key.deserializer", StringDeserializer.class.getName());
        props.put("value.deserializer", StringDeserializer.class.getName());
        this.consumer = new KafkaConsumer<>(props);
    }

    @Override
    public void run() {
        try {
            consumer.subscribe(topics);

            while (true) {
                ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(10));
                for (ConsumerRecord<String, String> record : records) {
                    Map<String, Object> data = new HashMap<>();
                    data.put("partition", record.partition());
                    data.put("offset", record.offset());
                    data.put("value", record.value());
                    System.out.println(record.value());
                    consumer.commitAsync();
                }
            }
        } catch (WakeupException e) {
            // ignore for shutdown
        } finally {
            consumer.close();
        }
    }
}

```


- Main function to consume the kafka topics

```

public static void main(String[] args) {
    int numConsumers = 1;
    String groupId = "upgrad-group";
    List<String> topics = Arrays.asList("message");
    ExecutorService executor = Executors.newFixedThreadPool(numConsumers);

    final List<Consumerloop> consumers = new ArrayList<>();
    for (int i = 0; i < numConsumers; i++) {
        Consumerloop consumer = new Consumerloop(i, groupId, topics);
        consumers.add(consumer);
        executor.submit(consumer);
    }

    Runtime.getRuntime().addShutdownHook(new Thread() {
        @Override
        public void run() {
            for (Consumerloop consumer : consumers) {
                consumer.shutdown();
            }
            executor.shutdown();
            try {
                executor.awaitTermination(5000, TimeUnit.MILLISECONDS);
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
                e.getMessage();
            }
        }
    });
}

```