MSA 14~15주차

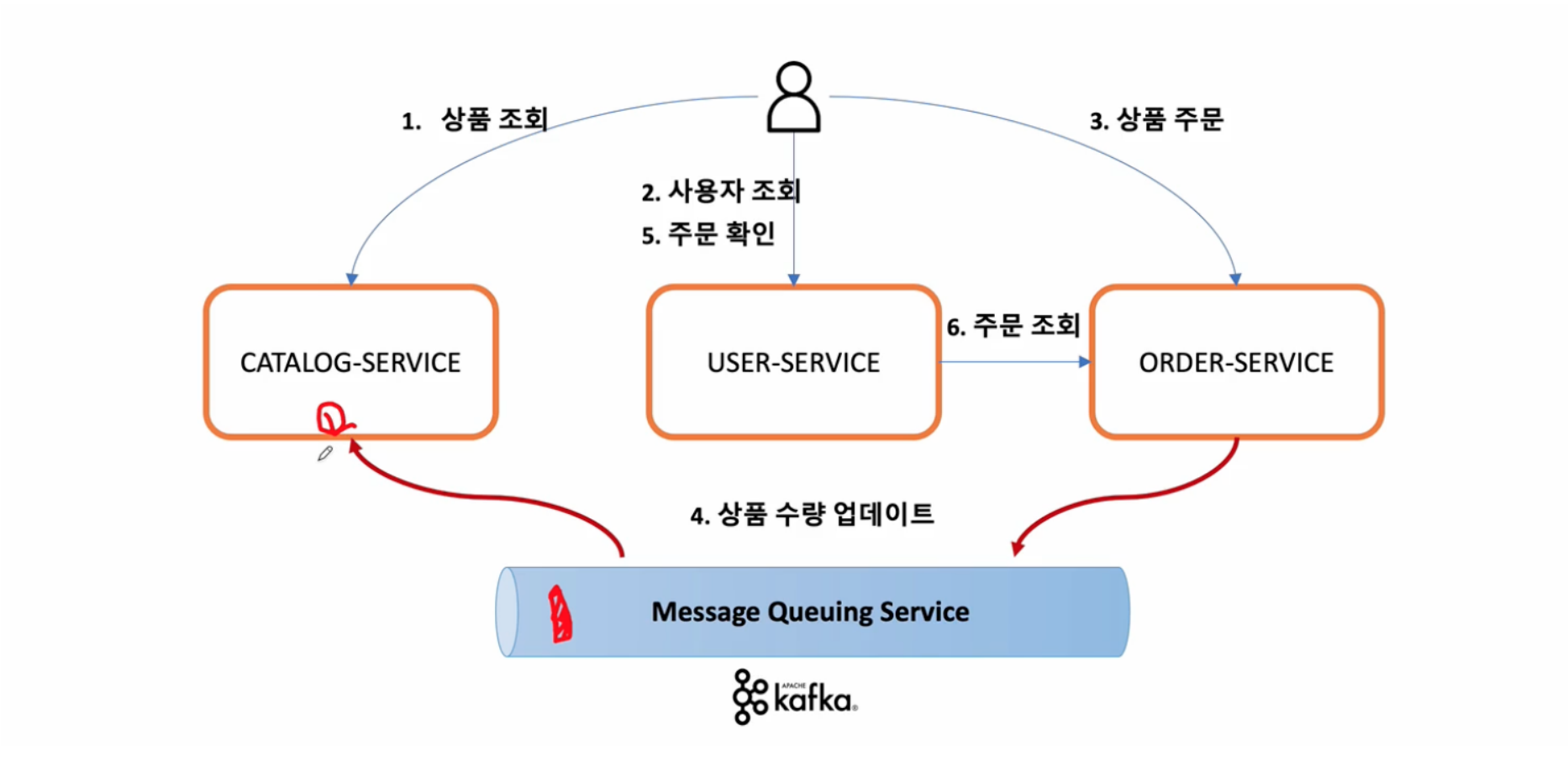
스터디원	소현
Status	Done
URL	https://github.com/SPRING-STUDY-2023/sohyeon-spring-cloud-msa/pull/9
비고	MSA Section12
제출 마감일	@December 20, 2023

데이터 동기화를 위한 Apache Kafka 활용(2)

Orders Microservice와 Catalogs Microservice에 Kafka Topic 적용

데이터 동기화(1) Orders → Catalogs

- Orders Service에 요청된 주문의 수량 정보를 Catalogs Service에 반영
- Orders Service에서 Kafka Topic으로 메시지 전송 → Producer
- Catalogs Service에서 Kafka Topic에 전송된 메시지 취득 → Consumer



Catalogs Microservice 수정

Dependency 추가

```
// https://mvnrepository.com/artifact/org.springframework.kafka/spring-kafka
implementation group: 'org.springframework.kafka', name: 'spring-kafka', version: '3.1.0'
```

Bean 등록

```
@EnableKafka
@Configuration
public class KafkaConsumerConfig {
    @Bean
```

```
    public ConsumerFactory<String, String> consumerFactory() { // 접속하고자 하는 정보
        Map<String, Object> properties = new HashMap<>();
        properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092"); // kafka serv
        properties.put(ConsumerConfig.GROUP_ID_CONFIG, "consumerGroupId"); // consumer group id
        properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class)

        return new DefaultKafkaConsumerFactory<>(properties);
    }

    @Bean
    public ConcurrentKafkaListenerContainerFactory<String, String> kafkaListenerContainerFactory
        ConcurrentKafkaListenerContainerFactory<String, String> kafkaListenerContainerFactory
        kafkaListenerContainerFactory.setConsumerFactory(consumerFactory());

        return kafkaListenerContainerFactory;
    }
}
```

Consumer 추가

```
@Service
@Slf4j
@RequiredArgsConstructor
public class KafkaConsumer {
    private final CatalogRepository catalogRepository;

    @KafkaListener(topics = "example-catalog-topic")
    @Transactional
    public void updateQty(String kafkaMessage) {
        log.info("Kafka Message: ->" + kafkaMessage);

        Map<Object, Object> map = new HashMap<>();
        ObjectMapper mapper = new ObjectMapper();
        try {
            map = mapper.readValue(kafkaMessage, new TypeReference<>() {});
        } catch (JsonProcessingException ex) {
            ex.printStackTrace();
        }

        CatalogEntity entity = catalogRepository.findByProductId((String)map.get("productId"))
            .orElseThrow(EntityNotFoundException::new);
        entity.setStock(entity.getStock() - (Integer)map.get("qty"));
    }
}
```

Orders Microservice 수정

Dependency 추가

```
// https://mvnrepository.com/artifact/org.springframework.kafka/spring-kafka
implementation group: 'org.springframework.kafka', name: 'spring-kafka', version: '3.1.0'
```

Bean 등록

```
@EnableKafka
@Configuration
public class KafkaProducerConfig {
    @Bean
    public ProducerFactory<String, String> producerFactory () { // 접속 정보
        Map<String, Object> properties = new HashMap<>();
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092"); // kafka serv
        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer .class);

        return new DefaultKafkaProducerFactory<>(properties);
    }

    @Bean
    public KafkaTemplate<String, String> kafkaTemplate() {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

Producer 추가

```
@Service
@Slf4j
@RequiredArgsConstructor
public class KafkaProducer {
    private final KafkaTemplate<String, String> kafkaTemplate;

    public OrderDto send(String topic, OrderDto orderDto) {
        ObjectMapper mapper = new ObjectMapper();
        String jsonInString = "";
        try {
            jsonInString = mapper.writeValueAsString(orderDto);
        } catch (JsonProcessingException ex) {
            log.error(ex.getMessage(), ex);
        }

        kafkaTemplate.send(topic, jsonInString);
        log.info("Kafka Producer send data from the Order Microservice: " + orderDto);

        return orderDto;
    }
}
```

Controller 수정

Kafka Producer 전송 코드 추가

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/order-service")
public class OrderController {
    private final KafkaProducer kafkaProducer;

    @PostMapping("/{userId}/orders")
    public ResponseEntity<ResponseOrder> createOrder(@PathVariable String userId, @RequestBody R
        ...
```

```
        /* send this order to the kafka */
        kafkaProducer.send("example-catalog-topic", orderDto);

        ...
    }
}
```

Kafka를 활용한 데이터 동기화 테스트(1)

사전 실행

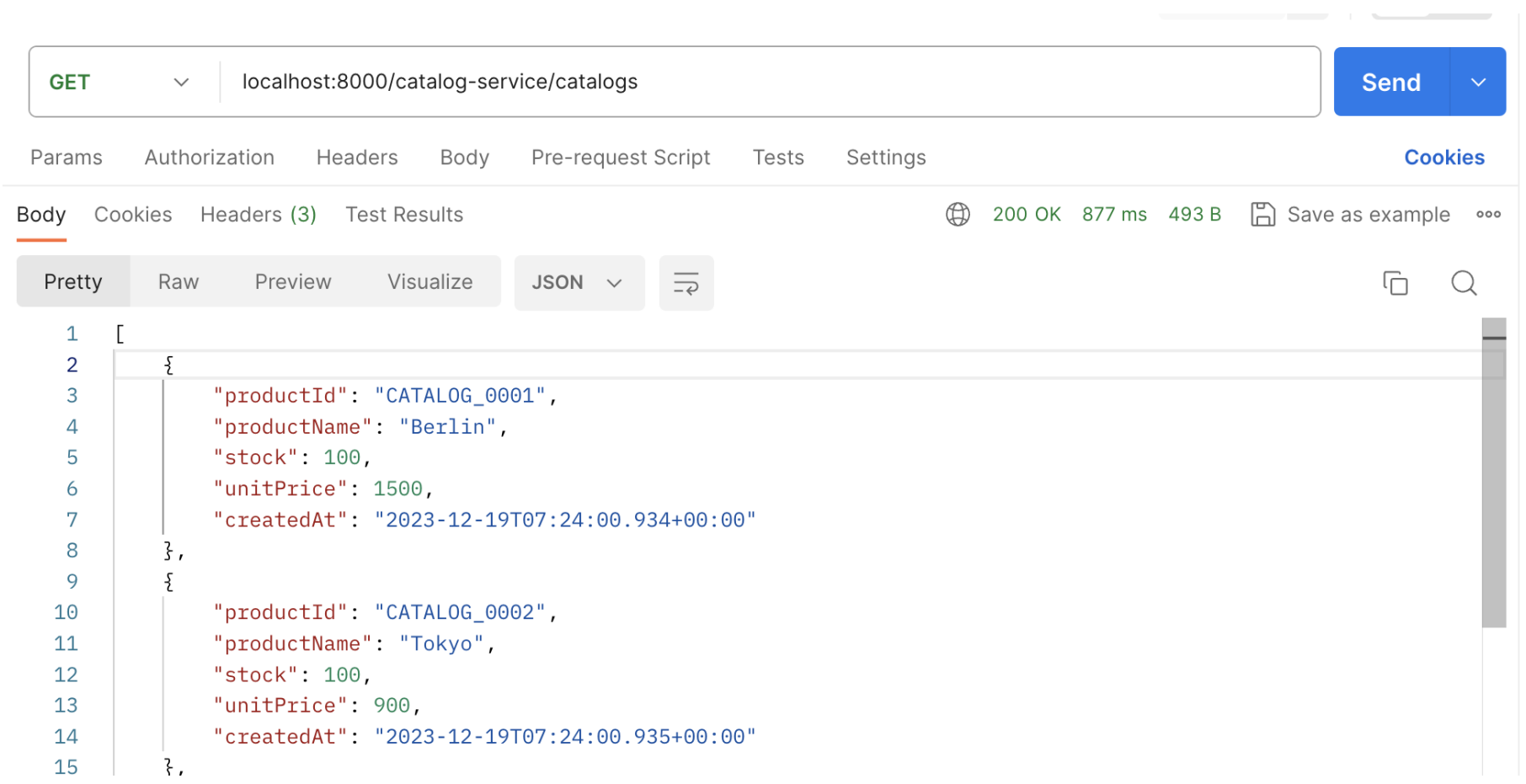
- Eureka Server
- Zookeeper Server

- Kafka Server

- Config Service
- Api-Gateway Service
- Catalog Service
- Order Service

주문 API 호출

- C15ea (CATALOG_0001)



Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM CATALOG
```

SELECT * FROM CATALOG;

STOCK	UNIT_PRICE	CREATED_AT	ID	PRODUCT_ID	PRODUCT_NAME
100	1500	2023-12-19 16:24:00.934832	1	CATALOG_0001	Berlin
100	900	2023-12-19 16:24:00.935891	2	CATALOG_0002	Tokyo
100	1200	2023-12-19 16:24:00.936078	3	CATALOG_0003	Stockholm

(3 rows, 6 ms)

Edit

- 주문 API 호출 (성공)

POST | localhost:8000/order-service/cf3071de-89f2-44f9-9e1d-12c834f2ad72/orders | Send

Params | Authorization | Headers | Body | Pre-request Script | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON | Beautify

```
{
    "productId": "CATALOG_0001",
    "qty": 5,
    "unitPrice": 1500
}
```

Body | Cookies | Headers (3) | Test Results | 201 Created | 385 ms | 243 B | Save as example

Pretty | Raw | Preview | Visualize | JSON

```
{
    "productId": "CATALOG_0001",
    "qty": "5",
    "unitPrice": 1500,
    "totalPrice": 7500,
    "orderId": "119ec1f2-db2d-4450-b836-561f6ed90336"
}
```

Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM ORDERS
```

SELECT * FROM ORDERS;

ID	CREATED_AT	ORDER_ID	PRODUCT_ID	QTY	TOTAL_PRICE	UNIT_PRICE	USER_ID
1	2023-12-19 16:28:38.349551	119ec1f2-db2d-4450-b836-561f6ed90336	CATALOG_0001	5	7500	1500	cf3071de-89f2-44f9-9e1d-12c834f2ad72

(1 row, 3 ms)

Edit

- 주문 후 Catalog 데이터 (CATALOG_0001)

GET localhost:8000/catalog-service/catalogs Send

Params Authorization Headers Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (3) Test Results 200 OK 60 ms 492 B Save as example

Pretty Raw Preview Visualize JSON


```
[
    {
        "productId": "CATALOG_0001",
        "productName": "Berlin",
        "stock": 95,
        "unitPrice": 1500,
        "createdAt": "2023-12-19T07:24:00.934+00:00"
    },
    {
        "productId": "CATALOG_0002",
        "productName": "Tokyo",
        "stock": 100,
        "unitPrice": 900,
        "createdAt": "2023-12-19T07:24:00.935+00:00"
    },
```


Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM CATALOG
```

SELECT * FROM CATALOG;

STOCK	UNIT_PRICE	CREATED_AT	ID	PRODUCT_ID	PRODUCT_NAME
95	1500	2023-12-19 16:24:00.934832	1	CATALOG_0001	Berlin
100	900	2023-12-19 16:24:00.935891	2	CATALOG_0002	Tokyo
100	1200	2023-12-19 16:24:00.936078	3	CATALOG_0003	Stockholm

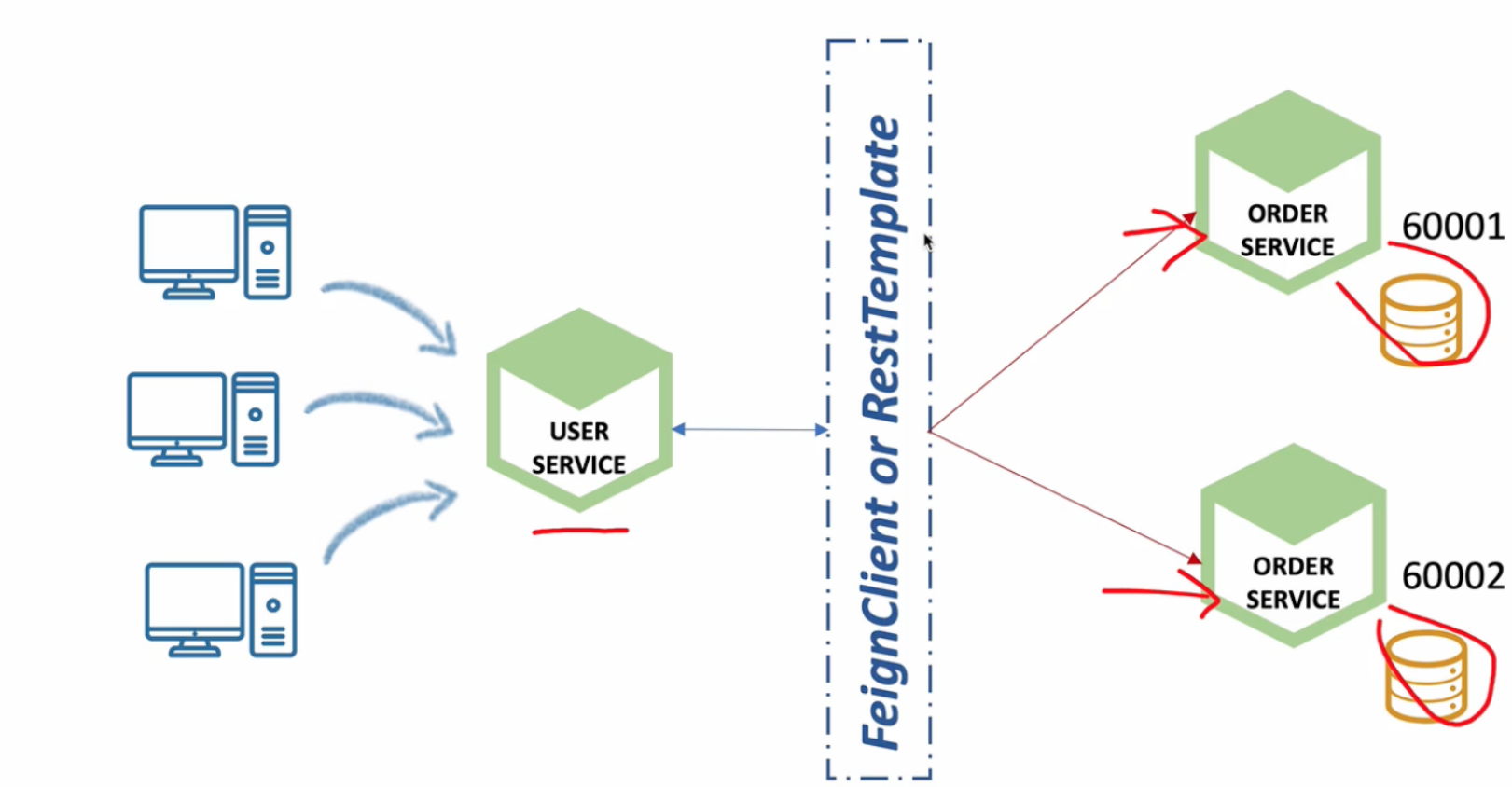
(3 rows, 6 ms)

Edit

Multi Orders Microservice 사용에 대한 데이터 동기화 문제

Orders Service 2개 기동

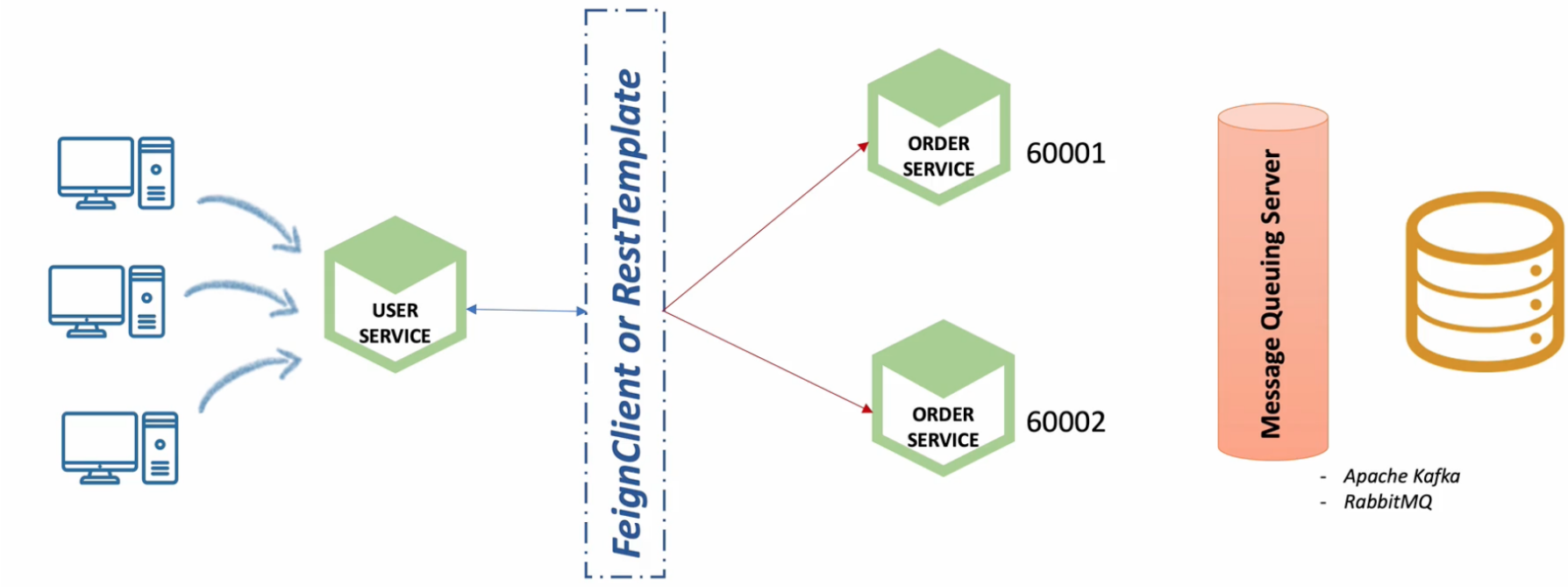
- Users의 요청 분산 처리
- **Orders 데이터도 분산 저장 → 동기화 문제**



Kafka Connect를 활용한 단일 데이터베이스 사용

데이터 동기화(2) - Multiple Orders Service

- Orders Service에 요청된 주문 정보를 DB가 아니라 Kafka Topic으로 전송
- Kafka Topic에 설정된 Kafka Sink Connect를 사용해 단일 DB에 저장 → 데이터 동기화



- Orders Service의 JPA 데이터베이스 교체 (H2 DB → MariaDB)

Orders Microservice 수정 - MariaDB

mydb 설정

- `mysql -u root -p` 진입
- orders 테이블 추가

```
create table orders (
  id int auto_increment primary key,
  user_id varchar(50) not null,
```

```
   product_id varchar(20) not null,
   order_id varchar(50) not null,
   qty int default 0,
   unit_price int default 0,
   total_price int default 0,
   created_at datetime default now()
 );
```

Order Service 변경 (db 설정)

접속 DB 정보 수정

```
spring:
    ...
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://localhost:3306/mydb
    username: test # 접속 계정 이름
    password: test1357 # 접속 계정 비밀번호
...
```

Trouble Shooting

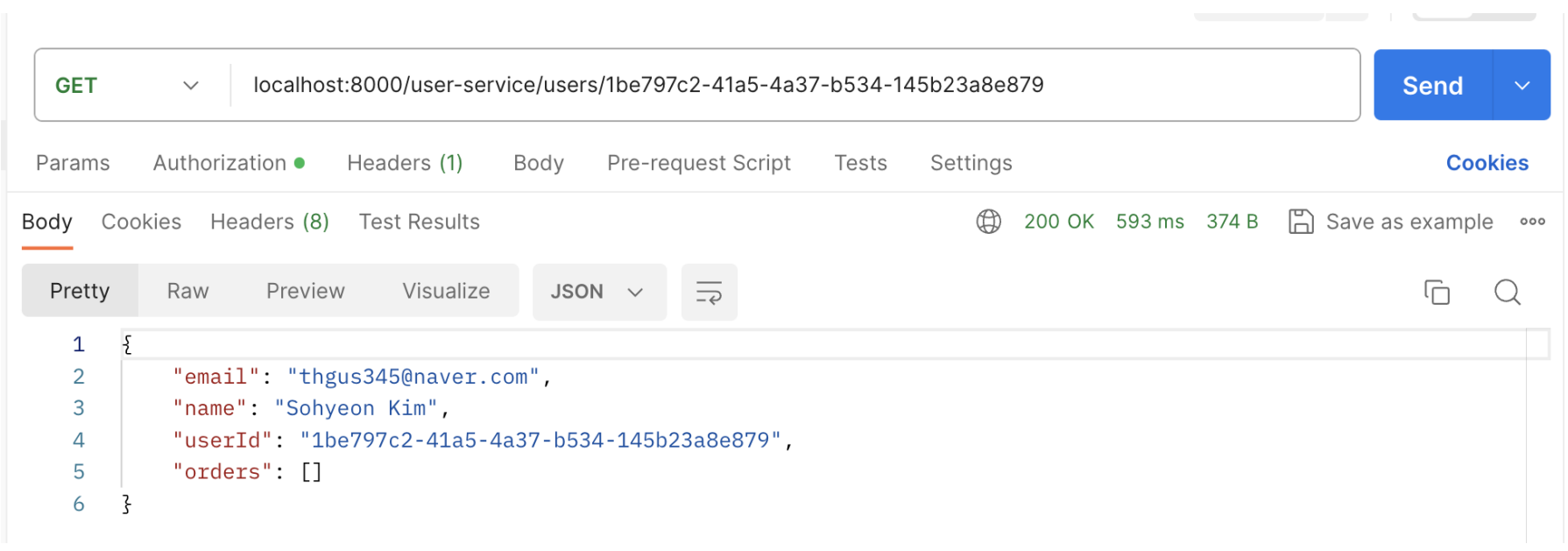
- MySQL, MariaDB 버전 호환성 이슈

> 🚨 **Error Log**
>
> java.sql.SQLException: (conn=19) Unknown system variable 'tx_isolation'

```
// https://mvnrepository.com/artifact/org.mariadb.jdbc/mariadb-java-client
implementation group: 'org.mariadb.jdbc', name: 'mariadb-java-client', version: '3.3.1'
```

API 테스트

- 주문 전 사용자 조회



- 상품 주문

POST localhost:8000/order-service/1be797c2-41a5-4a37-b534-145b23a8e879/orders Send

Params Authorization Headers Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "productId": "CATALOG_0001",
    "qty": 10,
    "unitPrice": 1500
}
```

Body Cookies Headers (3) Test Results 201 Created 542 ms 245 B Save as example

Pretty Raw Preview Visualize JSON

```
{
    "productId": "CATALOG_0001",
    "qty": "10",
    "unitPrice": 1500,
    "totalPrice": 15000,
    "orderId": "d7d7fceb-53c3-4024-86cc-730e6136a52b"
}
```

```
mysql> select * from orders;
+----+--------------------------------------+--------------+--------------------------------------+------+------------+-------------+---------------------+
| id | user_id                              | product_id   | order_id                             | qty  | unit_price | total_price | created_at          |
+----+--------------------------------------+--------------+--------------------------------------+------+------------+-------------+---------------------+
|  1 | 1be797c2-41a5-4a37-b534-145b23a8e879 | CATALOG_0001 | d7d7fceb-53c3-4024-86cc-730e6136a52b |   10 |       1500 |       15000 | 2023-12-19 08:07:12 |
+----+--------------------------------------+--------------+--------------------------------------+------+------------+-------------+---------------------+
1 row in set (0.02 sec)

mysql>
```

Orders Microservice 수정 - Order Kafka Topic

Orders Service의 Controller 수정

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/order-service")
public class OrderController {
    ...
    private final OrderProducer orderProducer;

    ...

    @PostMapping("/{userId}/orders")
    public ResponseEntity<ResponseOrder> createOrder(@PathVariable String userId, @RequestBody R
        ModelMapper mapper = new ModelMapper();
        mapper.getConfiguration().setMatchingStrategy(MatchingStrategies.STRICT);

        OrderDto orderDto = mapper.map(orderDetails, OrderDto.class);
        orderDto.setUserId(userId);

        /* JPA */
        // OrderDto createdOrder = orderService.createOrder(orderDto);
        // ResponseOrder responseOrder = mapper.map(createdOrder, ResponseOrder.class);

        /* Kafka */
        orderDto.setOrderId(UUID.randomUUID().toString());
        orderDto.setTotalPrice(orderDetails.getQty() * orderDetails.getUnitPrice());
```

```
        /* send this order to the kafka */
        kafkaProducer.send("example-catalog-topic", orderDto);
        orderProducer.send("orders", orderDto);

        ResponseOrder responseOrder = mapper.map(orderDto, ResponseOrder.class);

        return ResponseEntity.status(HttpStatus.CREATED).body(responseOrder);
    }
    ...
}
```

Orders Microservice 수정 - Order Kafka Producer

Orders Service의 Producer에서 발생하기 위한 메시지 등록

```
@Data
@AllArgsConstructor
public class KafkaOrderDto implements Serializable {
    private Schema schema;
    private Payload payload;
}
```

```
@Data
@Builder
public class Schema {
    private String type;
    private List<Field> fields;
    private boolean optional;
    private String name;
}
```

```
@Data
@AllArgsConstructor
public class Field {
    private String type;
    private boolean optional;
    private String field;
}
```

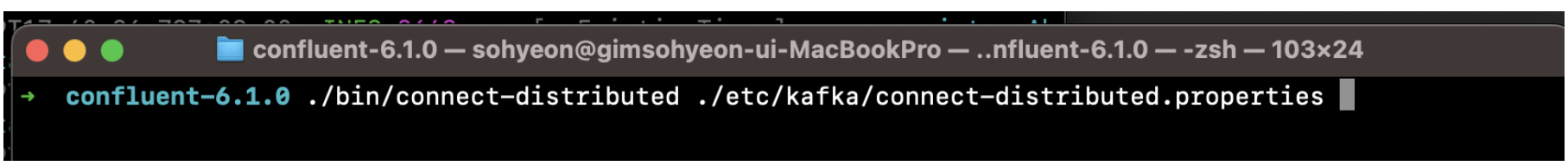
```
@Data
@Builder
public class Payload {
    private String order_id;
    private String user_id;
    private String product_id;
    private int qty;
    private int unit_price;
    private int total_price;
}
```

Orders Service의 OrderProducer 생성

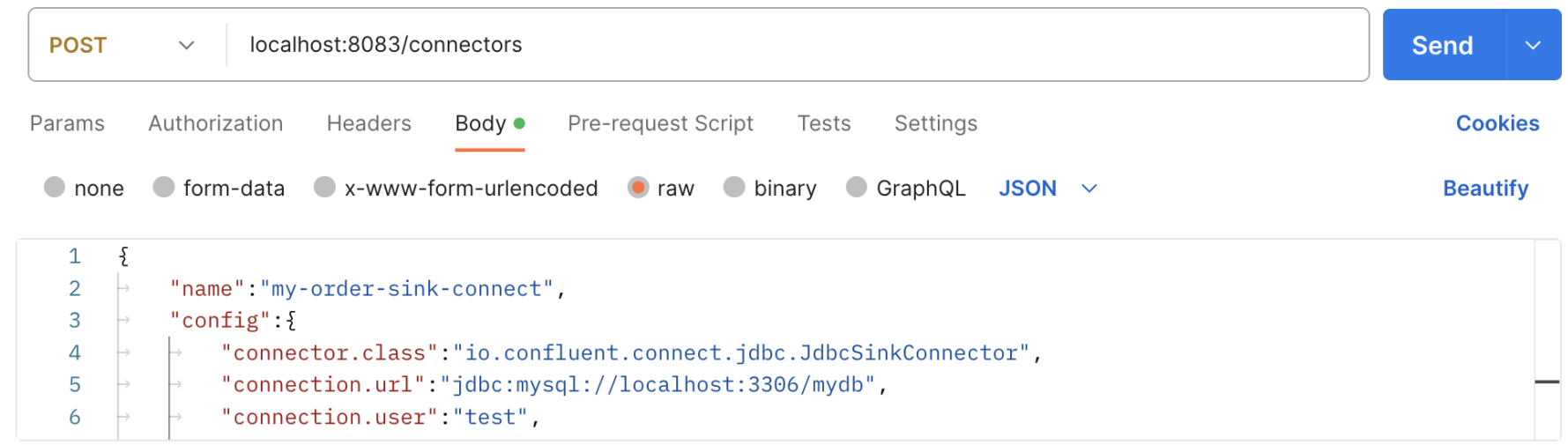
```
@Service
@Slf4j
@RequiredArgsConstructor
public class OrderProducer {
    private final KafkaTemplate<String, String> kafkaTemplate;

    List<Field> fields = List.of(
        new Field("string", true, "order_id"),
        new Field("string", true, "user_id"),
        new Field("int32", true, "product_id"),
        new Field("int32", true, "qty"),
        new Field("int32", true, "unit_price"),
        new Field("int32", true, "total_price")
    );
    Schema schema = Schema.builder()
        .type("struct")
        .fields(fields)
        .optional(false)
        .name("orders")
        .build();

    public void send(String topic, OrderDto orderDto) {
        Payload payload = Payload.builder()
            .order_id(orderDto.getOrderId())
            .user_id(orderDto.getUserId())
            .product_id(orderDto.getProductId())
            .qty(orderDto.getQty())
            .unit_price(orderDto.getUnitPrice())
            .total_price(orderDto.getTotalPrice())
            .build();

        KafkaOrderDto kafkaOrderDto = new KafkaOrderDto(schema, payload);

        ObjectMapper mapper = new ObjectMapper();
        String jsonInString = "";
        try {
            jsonInString = mapper.writeValueAsString(kafkaOrderDto);
        } catch (JsonProcessingException ex) {
            log.error(ex.getMessage(), ex);
        }

        kafkaTemplate.send(topic, jsonInString);
        log.info("Order Producer send data from the Order Microservice: " + kafkaOrderDto);
    }
}
```

Orders Service를 위한 Kafka Sink Connector 추가

- kafka connect 실행

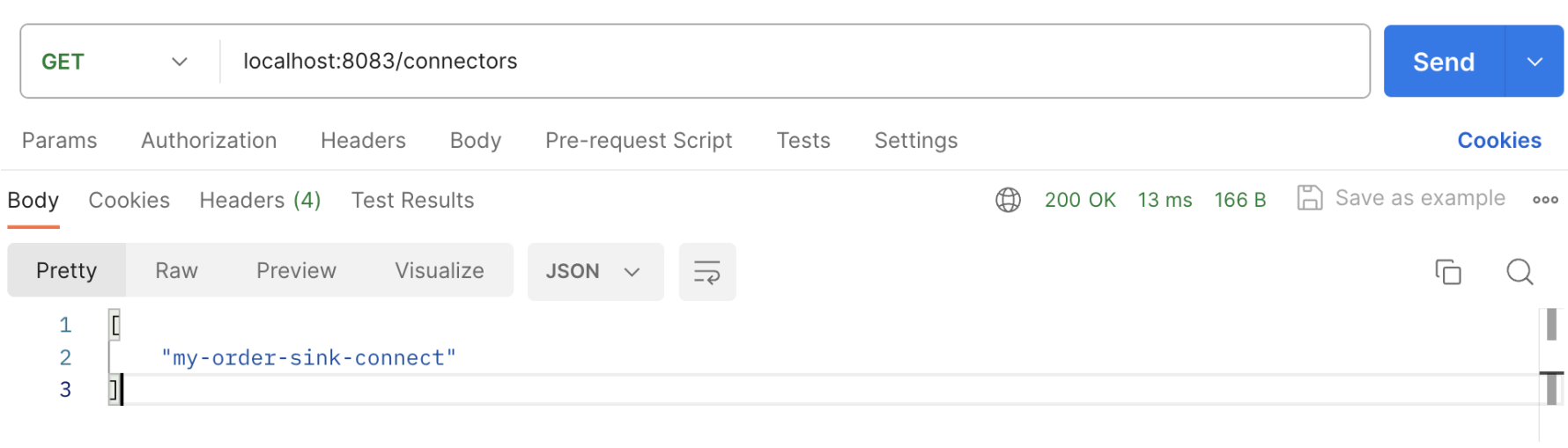


- Sink Connector 추가




```
{
    "name":"my-order-sink-connect",
    "config":{
        "connector.class":"io.confluent.connect.jdbc.JdbcSinkConnector",
        "connection.url":"jdbc:mysql://localhost:3306/mydb",
        "connection.user":"test",
        "connection.password":"test1357",
        "auto.create":"true",
        "auto.evolve":"true",
        "delete.enabled":"false",
        "tasks.max":"1",
        "topics":"orders"
    }
}
```

- Connector 확인



GET localhost:8083/connectors/my-order-sink-connect/status Send

Params Authorization Headers Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (4) Test Results 200 OK 37 ms 315 B Save as example

Pretty Raw Preview Visualize JSON


```
{
    "name": "my-order-sink-connect",
    "connector": {
        "state": "RUNNING",
        "worker_id": "127.0.0.1:8083"
    },
    "tasks": [
        {
            "id": 0,
            "state": "RUNNING",
            "worker_id": "127.0.0.1:8083"
        }
    ],
    "type": "sink"
}
```


Kafka를 활용한 데이터 동기화 테스트(2)

2개 Order Service 실행

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CATALOG-SERVICE	n/a (1)	(1)	UP (1) - catalog-service:11087352addb086554ff63d4af089216
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.109:gateway-service:8000
ORDER-SERVICE	n/a (2)	(2)	UP (2) - order-service:6b8be15268a7096067f1116a96058c91 , order-service:0a8091fa5750430587fac0eb0c21feb7
USER-SERVICE	n/a (1)	(1)	UP (1) - user-service:57b1bf555e72bd9098ac62968fdf1524

주문 테스트 (상품 주문 API 3번 반복 호출)

단일 DB에 주문 데이터 저장

```
mysql> select * from orders;
+----+--------------------------------------+-------------+--------------------------------------+------+------------+-------------+---------------------+
| id | user_id                              | product_id  | order_id                             | qty  | unit_price | total_price | created_at          |
+----+--------------------------------------+-------------+--------------------------------------+------+------------+-------------+---------------------+
|  1 | 1be797c2-41a5-4a37-b534-145b23a8e879 | CATALOG_0001 | d7d7fceb-53c3-4024-86cc-730e6136a52b |   10 |       1500 |       15000 | 2023-12-19 08:07:12 |
|  2 | 1be797c2-41a5-4a37-b534-145b23a8e879 | 0           | 775d4f9c-ee06-4a15-ba65-1f344e3decd3 |   10 |       1500 |       15000 | 2023-12-19 09:06:04 |
|  3 | 1be797c2-41a5-4a37-b534-145b23a8e879 | 0           | c321a5e4-e708-45c4-abe3-b2fe7fa8491b |    5 |       1500 |        7500 | 2023-12-19 09:10:09 |
|  4 | 1be797c2-41a5-4a37-b534-145b23a8e879 | 0           | 562840ca-638a-44bb-9229-13685b232143 |   10 |       1500 |       15000 | 2023-12-19 09:10:41 |
|  5 | 1be797c2-41a5-4a37-b534-145b23a8e879 | 0           | 08742dac-7206-495b-bb55-d2709aac45ee |   10 |       1500 |       15000 | 2023-12-19 09:16:04 |
|  6 | 1be797c2-41a5-4a37-b534-145b23a8e879 | 0           | 82b20d35-e219-4b3d-b72e-030141676c22 |   10 |       1500 |       15000 | 2023-12-19 09:16:12 |
+----+--------------------------------------+-------------+--------------------------------------+------+------------+-------------+---------------------+
6 rows in set (0.01 sec)

mysql>
```

GET localhost:8000/user-service/users/1be797c2-41a5-4a37-b534-145b23a8e879 Send

Params Authorization Headers (1) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (8) Test Results 200 OK 146 ms 1.44 KB Save as example

Pretty Raw Preview Visualize JSON


```
{
    "email": "thgus345@naver.com",
    "name": "Sohyeon Kim",
    "userId": "1be797c2-41a5-4a37-b534-145b23a8e879",
    "orders": [
        {
            "productId": "CATALOG_0001",
            "qty": 10,
            "unitPrice": 1500,
            "totalPrice": 15000,
            "createdAt": "2023-12-18T23:07:12.000+00:00",
            "orderId": "d7d7fceb-53c3-4024-86cc-730e6136a52b"
        },
        {
            "productId": "0",
            "qty": 10,
            "unitPrice": 1500,
            "totalPrice": 15000,
            "createdAt": "2023-12-19T00:06:04.000+00:00",
```


Pull Request

https://github.com/SPRING-STUDY-2023/sohyeon-spring-cloud-msa/pull/9