

Spring Cloud로 개발하는 マイクロ서비스 アプリケイ션



Microservices

+



Spring
Cloud

```
class Book {
    def self, title, price, author;
    self.title = title
    self.price = price
    self.author = author
}

public static void main(String[] args)
{
    var fs = require('fs');
    fs.readFile('/JONE.txt' /* 1 */,
        function (err, data) {
            console.log(data); // 3
        });
}

<@interface NextInnovationDelegate : NSObject <UIApplicationDelegate> >

<@implementation NextInnovationDelegate >
<@end>
```



목차

Part II

- Section 0: Microservice와 Spring Cloud 소개
- Section 1: Service Discovery
- Section 2: API Gateway Service
- Section 3: E-commerce 애플리케이션
- Section 4: Users Microservice - ①
- **Section 5: Catalogs, Orders Microservice**
- Section 6: Users Microservice - ②
- Section 7: Configuration Service
- Section 8: Spring Cloud Bus

Section 5.

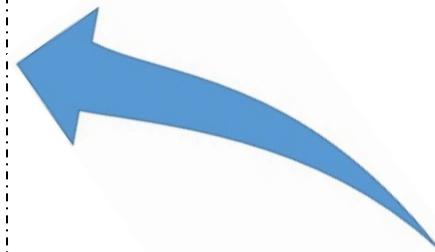
Catalogs, Orders Microservice

- Users Microservice 기능 추가
- Catalogs Microservice 프로젝트 생성
- Orders Microservice 프로젝트 생성

Users Microservice 기능 추가

- Features

- 신규 회원 등록
- 회원 로그인
- 상세 정보 확인
- 회원 정보 수정/삭제
- 상품 주문
- 주문 내역 확인



Users Microservice

Front-end

Business Logic

: ...
: ...
: ...
: ...



Database



Users Microservice 기능 추가

■ APIs

기능	URI (API Gateway 사용 시)	URI (API Gateway 미사용 시)	HTTP Method
사용자 정보 등록	/user-service/users	/users	POST
전체 사용자 조회	/user-service/users	/users	GET
사용자 정보, 주문 내역 조회	/user-service/users/{user_id}	/users/{user_id}	GET
작동 상태 확인	/user-service/users/health_check	/health_check	GET
환영 메시지	/user-service/users/welcome	/welcome	GET

```
1 {  
2   "name": "Dowon Lee",  
3   "email": "edowon0623@gmail.com",  
4   "pwd": "test1234"  
5 }
```

```
1 {  
2   "userId": "95be247c-7812-4943-b0c9-47eaad744534",  
3   "name": "Dowon Lee",  
4   "email": "edowon0623@gmail.com",  
5   "orders": []  
6 }
```



Users Microservice 기능 추가

- Root Request Mapping 정보 변경
 - UserController.java

```
@GetMapping("/health_check")
public String status(HttpServletRequest request) {
    return String.format("It's Working in User Service on Port %s", request.getServerPort());
}
```



Users Microservice 기능 추가

- API Gateway Service 변경 – application.yml (API Gateway service)

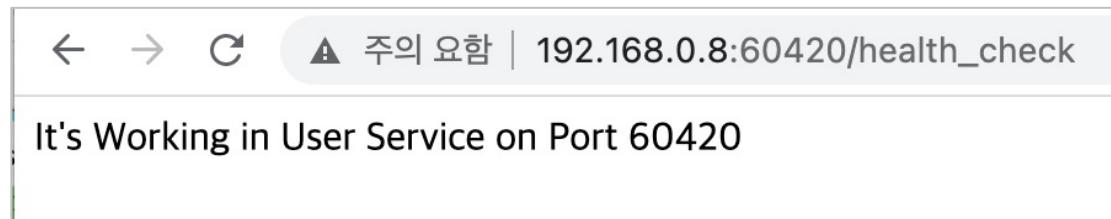
- User service route 추가

```
spring:  
  application:  
    name: gateway-service  
  cloud:  
    gateway:  
      default-filters:  
        - name: GlobalFilter  
          args:  
            baseMessage: Spring Cloud Gateway GlobalFilter  
            preLogger: true  
            postLogger: true  
      routes:  
        - id: user-service  
          uri: lb://USER-SERVICE  
          predicates:  
            - Path=/user-service/**
```

Users Microservice 기능 추가

■ 실행

Application	AMIs	Availability Zones	Status
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.8:gateway-service:8000
USER-SERVICE	n/a (1)	(1)	UP (1) - user-service:d8c9a5408b14584d2220d1d718f95392



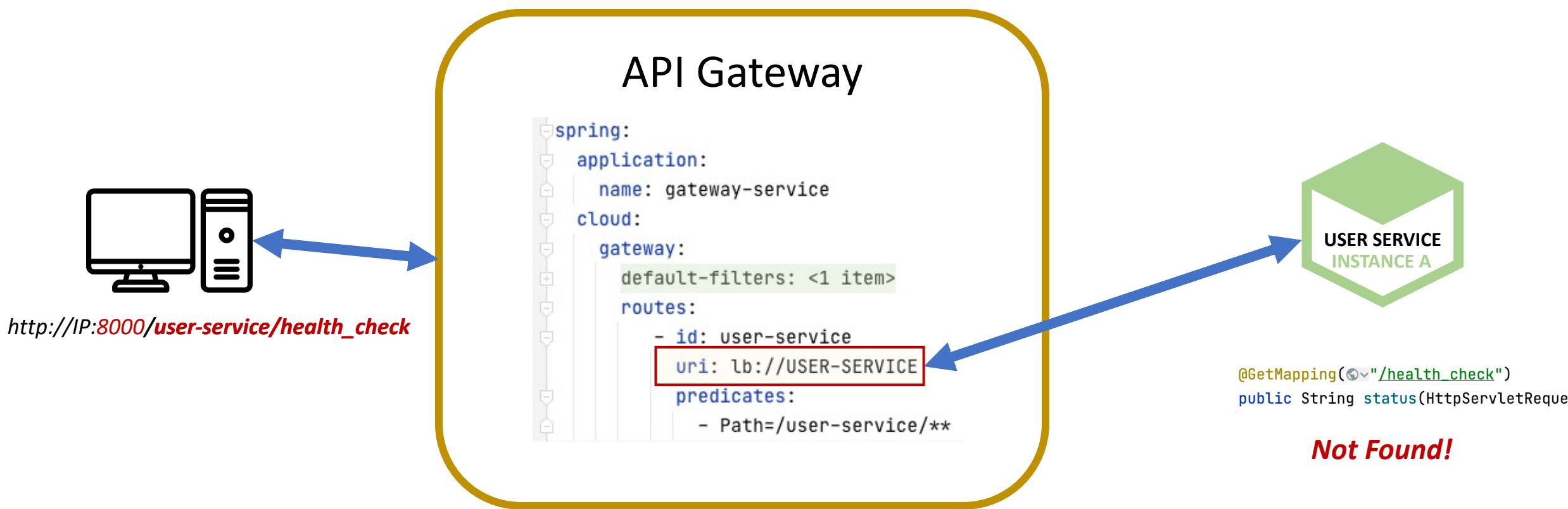
- user-service 직접 호출



- API Gateway 호출

Users Microservice 기능 추가

- Users Service의 URI와 API Gateway의 URI가 다름

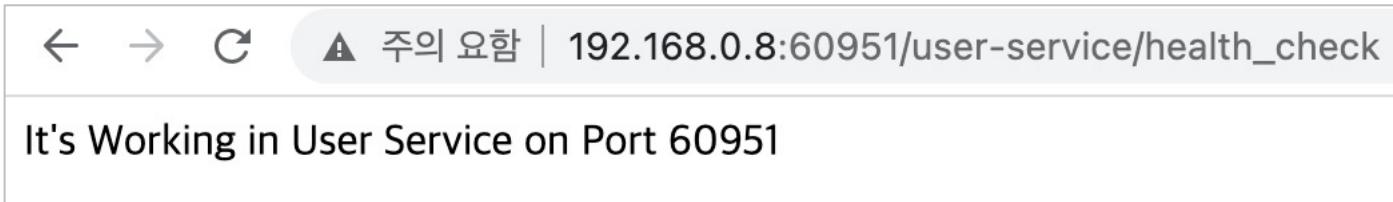


Users Microservice 기능 추가

- Root Request Mapping 정보 변경

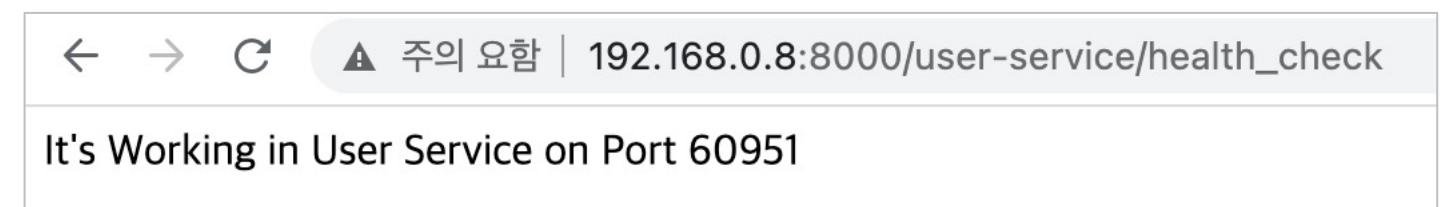
- UserController.java

```
@GetMapping("/user-service/health_check")
public String status(HttpServletRequest request) {
    return String.format("It's Working in User Service on Port %s", request.getServerPort());
}
```



- user-service 직접 호출

- API Gateway 호출



Users Microservice 기능 추가

- ResponseUser.java, ResponseOrder.java

```
@Data  
@JsonInclude(JsonInclude.Include.NON_NULL)  
public class ResponseUser {  
    private String email;  
    private String name;  
    private String userId;  
  
    private List<ResponseOrder> orders;  
}
```

```
@Data  
public class ResponseOrder {  
    private String productId;  
    private Integer qty;  
    private Integer unitPrice;  
    private Integer totalPrice;  
    private Date createdAt;  
  
    private String orderId;  
}
```

Users Microservice 기능 추가

- UserService.java, UserServiceImpl.java

```
public interface UserService {  
    UserDto createUser(UserDto userDto);  
    UserDto getUserByUserId(String userId);  
    Iterable<UserEntity> getUserByAll();  
}
```

```
@Override  
public UserDto getUserByUserId(String userId) {  
    UserEntity userEntity = userRepository.findById(userId);  
  
    if (userEntity == null)  
        throw new UsernameNotFoundException("User not found");  
  
    UserDto userDto = new ModelMapper().map(userEntity, UserDto.class);  
  
    List<ResponseOrder> ordersList = new ArrayList<>();  
    userDto.setOrders(ordersList);  
  
    return userDto;  
}  
  
@Override  
public Iterable<UserEntity> getUserByAll() {  
    return userRepository.findAll();  
}
```



Users Microservice 기능 추가

- UserController.java

- 전체 사용자 목록 보기

```
@GetMapping(value="/users")  
public ResponseEntity<List<ResponseUser>> getUsers() {  
    Iterable<UserEntity> userList = userService.getUserByAll();  
  
    List<ResponseUser> result = new ArrayList<>();  
    userList.forEach(v -> {  
        result.add(new ModelMapper().map(v, ResponseUser.class));  
    });  
  
    return ResponseEntity.status(HttpStatus.OK).body(result);  
}
```



Users Microservice 기능 추가

■ UserController.java

- 사용자 정세보기 + 주문 목록 보기

```
@GetMapping(value="/users/{userId}")  
public ResponseEntity<ResponseUser> getUser(@PathVariable("userId") String userId) {  
    UserDto userDto = userService.getUserByUserId(userId);  
    ResponseUser returnValue = new ModelMapper().map(userDto, ResponseUser.class);  
  
    return ResponseEntity.status(HttpStatus.OK).body(returnValue);  
}
```

Users Microservice 기능 추가

■ 실행

POST http://127.0.0.1:8000/user-service/users

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

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

```
1 {
2   "name": "Dowon Lee",
3   "email": "test1@test.com",
4   "pwd": "test1111"
5 }
```

Body Cookies Headers (8) Test Results

Status: 201 Created

Pretty Raw Preview Visualize JSON

```
1 {
2   "email": "test1@test.com",
3   "name": "Dowon Lee",
4   "userId": "28d68e9e-b9fb-4514-b245-7515dcbe046f"
5 }
```

- 회원가입
 - POST → <http://127.0.0.1:8000/user-service/users>

Users Microservice 기능 추가

■ 실행

GET ▼ http://127.0.0.1:8000/user-service/users

Params Authorization **Headers (10)** Body ● Pre-request Script Tests Settings

Headers 👁 8 hidden

KEY	VALUE	DESCRIPTION

Body Cookies Headers (8) Test Results 🌐 Status: 200 OK Timi

Pretty Raw Preview Visualize JSON ▼ ⟳

```
1 [  
2 {  
3   "email": "test1@test.com",  
4   "name": "Dowon Lee",  
5   "userId": "28d68e9e-b9fb-4514-b245-7515dcbe046f"  
6 }]  
7 ]
```

- 전체 회원 목록
 - GET → <http://127.0.0.1:8000/user-service/users>

Users Microservice 기능 추가

■ 실행

GET ▼ http://127.0.0.1:8000/user-service/users/28d68e9e-b9fb-4514-b245-7515dcbe046f

Params Authorization Headers (10) Body ● Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIP
Key	Value	Descrij

Body Cookies Headers (8) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON ↻

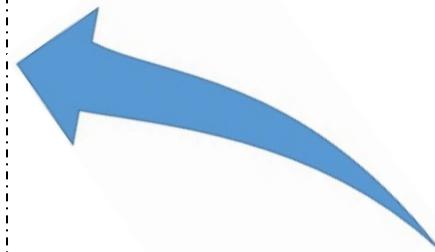
```
1 {  
2   "email": "test1@test.com",  
3   "name": "Dowon Lee",  
4   "userId": "28d68e9e-b9fb-4514-b245-7515dcbe046f",  
5   "orders": []  
6 }
```

- 회원 상세 정보 + 주문 정보
 - GET → <http://127.0.0.1:8000/user-service/users/{회원ID}>

Catalogs Microservice 추가

- Features

- 신규 회원 등록
- 회원 로그인
- 상세 정보 확인
- 회원 정보 수정/삭제
- 상품 주문
- 주문 내역 확인



Users Microservice

Front-end

Business Logic

: ...
: ...
: ...
: ...



Database





Catalogs Microservice 추가

■ APIs

기능	마이크로서비스	URI (API Gateway 사용 시)	HTTP Method
상품 목록 조회	Catalogs Microservice	/catalog-service/catalogs	GET
사용자 별 상품 주문	Orders Microservice	/order-service/{user_id}/orders	POST
사용자 별 주문 내역 조회	Orders Microservice	/order-service/{user_id}/orders	GET

Catalogs Microservice 추가

■ 프로젝트 생성

The screenshot shows the Spring Initializr web application interface. At the top, there's a search bar and a dropdown for 'Spring Boot' set to version 2.4.2. On the left, a sidebar lists various categories under 'Developer Tools': Web, Template Engines, Security, SQL, NoSQL, Messaging, I/O, Ops, Observability, Testing, Spring Cloud, Spring Cloud Security, Spring Cloud Tools, and Spring Cloud Config. The 'Developer Tools' category is currently selected, indicated by a blue background. In the main central area, three checkboxes are shown: 'Spring Boot DevTools' (checked), 'Lombok' (checked), and 'Spring Configuration Processor' (unchecked). To the right, a 'Selected Dependencies' panel lists the chosen items: 'Spring Boot DevTools', 'Lombok', 'Web' (with 'Spring Web'), 'SQL' (with 'Spring Data JPA'), and 'Spring Cloud Discovery' (with 'Eureka Discovery Client'). Each listed item has a close button ('X') to its right.

Dependencies

Spring Boot 2.4.2 ▾

Selected Dependencies

Developer Tools

- Web
- Template Engines
- Security
- SQL
- NoSQL
- Messaging
- I/O
- Ops
- Observability
- Testing
- Spring Cloud
- Spring Cloud Security
- Spring Cloud Tools
- Spring Cloud Config

Spring Boot DevTools

Lombok

Spring Configuration Processor

Developer Tools

- Spring Boot DevTools
- Lombok

Web

- Spring Web

SQL

- Spring Data JPA

Spring Cloud Discovery

- Eureka Discovery Client



Catalogs Microservice 추가

- 프로젝트 생성

- Dependencies 추가

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.3.176</version>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.modelmapper</groupId>
    <artifactId>modelmapper</artifactId>
    <version>2.3.8</version>
</dependency>
```

Catalogs Microservice 추가

- 프로젝트 생성

- application.yml 파일

```
server:  
  port: 0  
  
spring:  
  application:  
    name: catalog-service  
  
  h2:  
    console:  
      enabled: true  
      settings:  
        web-allow-others: true  
    path: /h2-console
```

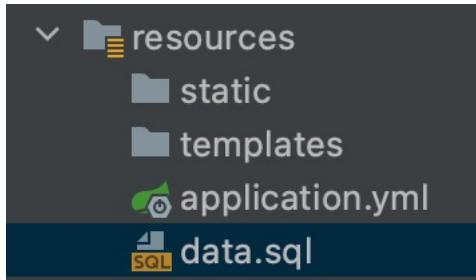
```
jpa:  
  hibernate:  
    ddl-auto: create-drop  
    show-sql: true  
    generate-ddl: true  
  
  datasource:  
    driver-class-name: org.h2.Driver  
    url: jdbc:h2:mem:testdb  
  
  eureka:  
    client:  
      service-url:  
        defaultZone: http://localhost:8761/eureka
```

```
logging:  
  level:  
    com.example.catalogservice: DEBUG
```

Catalogs Microservice 추가

■ 프로젝트 생성

- data.sql



```
insert into catalog(product_id, product_name, stock, unit_price)
    values('CATALOG-0001', 'Berlin', 100, 1500);
insert into catalog(product_id, product_name, stock, unit_price)
    values('CATALOG-0002', 'Tokyo', 100, 900);
insert into catalog(product_id, product_name, stock, unit_price)
    values('CATALOG-0003', 'Stockholm', 100, 1200);
```

```
2021-01-27 09:32:49.733  INFO 70554 --- [ restartedMain] org.hibernate.Hibernate: : HHH000412: Hibernate ORM core version 5.4.27.
2021-01-27 09:32:49.799  INFO 70554 --- [ restartedMain] o.hibernate.common.annotations: HCANN000001: Hibernate Commons Annotations {5
2021-01-27 09:32:49.856  INFO 70554 --- [ restartedMain] org.hibernate.dialect.Dialect: : HHH000400: Using dialect: org.hibernate.dialect
Hibernate: drop table if exists catalog CASCADE
Hibernate: create table catalog (id bigint generated by default as : default CURRENT_TIMESTAMP not null, product_id v
Hibernate: alter table catalog add constraint UK_9gggyslu2usn0rxs32l
```

SELECT * FROM CATALOG;

ID	CREATED_AT	PRODUCT_ID	PRODUCT_NAME	STOCK	UNIT_PRICE
1	2021-01-27 09:32:50.207	CATALOG-0001	Berlin	100	1500
2	2021-01-27 09:32:50.208	CATALOG-0002	Tokyo	100	900
3	2021-01-27 09:32:50.208	CATALOG-0003	Stockholm	100	1200

(3 rows, 6 ms)



Catalogs Microservice 추가

- CatalogEntity.java, CatalogRepository.java

```
@Data  
@Entity  
@Table(name="catalog")  
public class CatalogEntity implements Serializable {  
    @Id  
    @GeneratedValue(strategy=GenerationType.IDENTITY)  
    private long id;  
  
    @Column(nullable = false, length = 120, unique = true)  
    private String productId;  
    @Column(nullable = false)  
    private String productName;  
    @Column(nullable = false)  
    private Integer stock;  
    @Column(nullable = false)  
    private Integer unitPrice;  
  
    @Column(nullable = false, updatable = false, insertable = false)  
    @ColumnDefault( value = "CURRENT_TIMESTAMP" )  
    private Date createdAt;  
}  
  
public interface CatalogRepository extends CrudRepository<CatalogEntity, Long> {  
    CatalogEntity findByProductId(String productId);  
}
```



Catalogs Microservice 추가

- CatalogDto.java, ResponseCatalog.java

```
@Data  
public class CatalogDto implements Serializable {  
    private String productId;  
    private Integer qty;  
    private Integer unitPrice;  
    private Integer totalPrice;  
  
    private String orderId;  
    private String userId;  
}  
  
@Data  
@JsonInclude(JsonInclude.Include.NON_NULL)  
public class ResponseCatalog {  
    private String productId;  
    private String productName;  
    private Integer stock;  
    private Integer unitPrice;  
    private Date createdAt;  
}
```



Catalogs Microservice 추가

- CatalogService.java, CatalogServiceImpl.java

```
public interface CatalogService {  
    Iterable<CatalogEntity> getAllCatalogs();  
}
```

```
@Service  
@Slf4j  
public class CatalogServiceImpl implements CatalogService {  
    CatalogRepository repository;  
  
    Environment env;  
  
    @Autowired  
    public CatalogServiceImpl(CatalogRepository repository,  
                             Environment env) {  
        this.repository = repository;  
        this.env = env;  
    }  
  
    @Override  
    public Iterable<CatalogEntity> getAllCatalogs() {  
        return repository.findAll();  
    }  
}
```



Catalogs Microservice 추가

- CatalogController.java

```
@RestController
@RequestMapping("/catalog-service")
public class CatalogController {
    private Environment env;
    CatalogService catalogService;

    @Autowired
    public CatalogController(Environment env, CatalogService catalogService) {
        this.env = env;
        this.catalogService = catalogService;
    }

    @GetMapping("/health_check")
    public String status(HttpServletRequest request) {
        return String.format("It's Working in Catalog Service on Port %s", request.getServerPort());
    }
}
```



Catalogs Microservice 추가

- CatalogController.java

```
@GetMapping(value="/catalogs")
public ResponseEntity<List<ResponseCatalog>> getCatalogs() {
    Iterable<CatalogEntity> orderList = catalogService.getAllCatalogs();

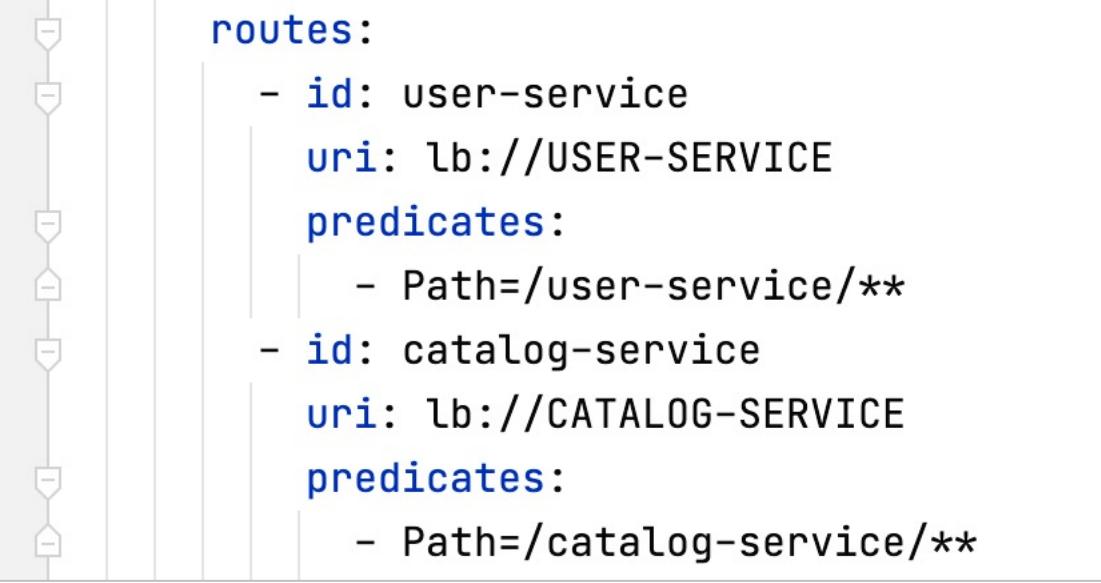
    List<ResponseCatalog> result = new ArrayList<>();
    orderList.forEach(v -> {
        result.add(new ModelMapper().map(v, ResponseCatalog.class));
    });

    return ResponseEntity.status(HttpStatus.OK).body(result);
}
```



Catalogs Microservice 추가

- API Gateway Service에 Service 등록



```
routes:
  - id: user-service
    uri: lb://USER-SERVICE
    predicates:
      - Path=/user-service/**
  - id: catalog-service
    uri: lb://CATALOG-SERVICE
    predicates:
      - Path=/catalog-service/**
```

- Test

Application	AMIs	Availability Zones	Status
CATALOG-SERVICE	n/a (1)	(1)	UP (1) - catalog-service:d859e8691c06622627ee783714b33d88
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.8:gateway-service:8000



Catalogs Microservice 추가

■ Test

GET

http://127.0.0.1:8000/catalog-service/catalogs

```
1 [  
2 {  
3     "productId": "CATALOG-0001",  
4     "productName": "Berlin",  
5     "stock": 100,  
6     "unitPrice": 1500,  
7     "createdAt": "2021-01-27T00:32:50.207+00:00"  
8 },  
9 {  
10    "productId": "CATALOG-0002",  
11    "productName": "Tokyo",  
12    "stock": 100,  
13    "unitPrice": 900,  
14    "createdAt": "2021-01-27T00:32:50.208+00:00"  
15 }
```

- 상품 목록

- GET → <http://127.0.0.1:8000/catalog-service/catalogs>



Orders Microservice 추가

■ APIs

기능	마이크로서비스	URI (API Gateway 사용 시)	HTTP Method
상품 목록 조회	Catalogs Microservice	/catalog-service/catalogs	GET
사용자 별 상품 주문	Orders Microservice	/order-service/{user_id}/orders	POST
사용자 별 주문 내역 조회	Orders Microservice	/order-service/{user_id}/orders	GET

Orders Microservice 추가

■ 프로젝트 생성

The screenshot shows the Spring Initializr web application interface for creating a new Spring Boot project. The project is titled "Orders Microservice" and is being configured for "Spring Boot 2.4.2".

Dependencies: A search bar is at the top left. The sidebar on the left lists categories: Developer Tools, Web, Template Engines, Security, SQL, NoSQL, Messaging, I/O, Ops, Observability, Testing, Spring Cloud, Spring Cloud Security, and Spring Cloud Tools. The "Developer Tools" category is currently selected.

Selected Dependencies: A list of checked dependencies is shown on the right, grouped by category:

- Developer Tools:** Spring Boot DevTools (checked), Lombok (checked), Spring Configuration Processor (unchecked).
- Web:** Spring Web.
- SQL:** Spring Data JPA, H2 Database.
- Spring Cloud Discovery:** Eureka Discovery Client.

Each dependency item has a close button ("X") to its right.



Orders Microservice 추가

- 프로젝트 생성

- Dependencies 추가

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.3.176</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.modelmapper</groupId>
    <artifactId>modelmapper</artifactId>
    <version>2.3.8</version>
</dependency>
```

Orders Microservice 추가

■ 프로젝트 생성

- application.yml 파일

```
server:  
  port: 0  
  
spring:  
  application:  
    name: order-service  
  
  h2:  
    console:  
      enabled: true  
      settings:  
        web-allow-others: true  
      path: /h2-console  
  
  jpa:  
    hibernate:  
      ddl-auto: update  
  
datasource:  
  driver-class-name: org.h2.Driver  
  url: jdbc:h2:mem:testdb  
  
eureka:  
  client:  
    service-url:  
      defaultZone: http://localhost:8761/eureka  
  instance:  
    instanceId: ${spring.application.name}:${spring.application.instance_id:${random.value}}  
  
logging:  
  level:  
    com.example.orderservice: DEBUG
```

Orders Microservice 추가

- OrderEntity.java, OrderRepository.java

```
@Data  
@Entity  
@Table(name="orders")  
public class OrderEntity implements Serializable {  
    @Id  
    @GeneratedValue  
    private long id;  
  
    @Column(nullable = false, length = 120)  
    private String productId;  
    @Column(nullable = false)  
    private Integer qty;  
    @Column(nullable = false)  
    private Integer unitPrice;  
    @Column(nullable = false)  
    private Integer totalPrice;  
  
    @Column(nullable = false)  
    private String userId;  
    @Column(nullable = false, unique = true)  
    private String orderId;  
  
    @Column(nullable = false, updatable = false, insertable = false)  
    @ColumnDefault( value = "CURRENT_TIMESTAMP" )  
    private Date createdAt;  
}
```

```
public interface OrdersRepository extends CrudRepository<OrderEntity, Long> {  
    OrderEntity findByOrderId(String orderId);  
    Iterable<OrderEntity> findByUserId(String userId);  
}
```



Orders Microservice 추가

- OrderDto.java, ResponseOrder.java

```
@Data  
public class OrderDto implements Serializable {  
    private String productId;  
    private Integer qty;  
    private Integer unitPrice;  
    private Integer totalPrice;  
  
    private String orderId;  
    private String userId;  
}
```

```
@Data  
@JsonInclude(JsonInclude.Include.NON_NULL)  
public class ResponseOrder {  
    private String productId;  
    private Integer qty;  
    private Integer unitPrice;  
    private Integer totalPrice;  
    private Date createdAt;  
  
    private String orderId;  
}
```



Orders Microservice 추가

- OrderService.java, OrderServiceImpl.java

```
public interface OrdersService {  
    OrderDto createOrder(OrderDto orderDetails);  
    OrderDto getOrderByOrderId(String orderId);  
    Iterable<OrderEntity> getOrdersByUserId(String userId);  
}
```

```
@Override  
public OrderDto createOrder(OrderDto orderDetails) {  
    orderDetails.setOrderId(UUID.randomUUID().toString());  
    orderDetails.setTotalPrice(orderDetails.getQty() * orderDetails.getUnitPrice());  
    ModelMapper modelMapper = new ModelMapper();  
    modelMapper.getConfiguration().setMatchingStrategy(MatchingStrategies.STRICT);  
  
    OrderEntity orderEntity = modelMapper.map(orderDetails, OrderEntity.class);  
  
    repository.save(orderEntity);  
  
    OrderDto returnValue = modelMapper.map(orderEntity, OrderDto.class);  
    return returnValue;  
}  
  
@Override  
public OrderDto getOrderByOrderId(String orderId) {  
    OrderEntity orderEntity = repository.findById(orderId);  
    OrderDto orderDto = new ModelMapper().map(orderEntity, OrderDto.class);  
  
    return orderDto;  
}  
  
@Override  
public Iterable<OrderEntity> getOrdersByUserId(String userId) {  
    return repository.findByUserId(userId);  
}
```



Orders Microservice 추가

- OrderController.java

```
@PostMapping(value="/{userId}/orders")
public ResponseEntity<ResponseOrder> createOrder(@PathVariable("userId") String userId,
                                                 @RequestBody RequestOrder orderDetails) {

    ModelMapper modelMapper = new ModelMapper();
    modelMapper.getConfiguration().setMatchingStrategy(MatchingStrategies.STRICT);

    OrderDto orderDto = modelMapper.map(orderDetails, OrderDto.class);
    orderDto.setUserId(userId);
    OrderDto createDto = ordersService.createOrder(orderDto);
    ResponseOrder returnValue = modelMapper.map(createDto, ResponseOrder.class);

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



Orders Microservice 추가

- OrderController.java

```
@GetMapping(value="/{userId}/orders")
public ResponseEntity<List<ResponseOrder>> getOrder(@PathVariable("userId") String userId) {
    Iterable<OrderEntity> orderList = ordersService.getOrdersByUserId(userId);

    List<ResponseOrder> result = new ArrayList<>();
    orderList.forEach(v -> {
        result.add(new ModelMapper().map(v, ResponseOrder.class));
    });

    return ResponseEntity.status(HttpStatus.OK).body(result);
}
```



Orders Microservice 추가

- API Gateway Service에 Service 등록

```
routes:  
  - id: order-service  
    uri: lb://ORDER-SERVICE  
    predicates:  
      - Path=/order-service/**
```

- Test

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.8:gateway-service:8000
ORDER-SERVICE	n/a (1)	(1)	UP (1) - order-service:3fc09ea7148e4c25f2c2ed68ce346282
USER-SERVICE	n/a (1)	(1)	UP (1) - user-service:68799d7ffb36e34a55751447d3553b12

Orders Microservice 추가

■ Test

The screenshot shows the Postman application interface for testing a POST request to the Orders Microservice.

Request URL: `127.0.0.1:8000/order-service/bbbe4241-943d-48f7-988e-809a55781496/orders`

Body (JSON):

```
1 {
2     "productId": "CATALOG-0002",
3     "qty": 10,
4     "unitPrice": 900
5 }
```

Response (Pretty JSON):

```
1 {
2     "productId": "CATALOG-0002",
3     "qty": 10,
4     "unitPrice": 900,
5     "totalPrice": 9000,
6     "orderId": "67419444-be51-40dc-a3e5-dcdb728d8780"
7 }
```

Notes:

- 상품 주문
- POST → `http://127.0.0.1:8000/order-service/{user_id}/orders`

Orders Microservice 추가

■ Test

The screenshot shows the Postman application interface with the following details:

- Method:** GET
- URL:** http://127.0.0.1:8000/order-service/c58d6b65-7271-4b62-9d42-6a8351262bb5/orders
- Headers:** (6) (highlighted in green)
- Body:** This request does not have a body
- Body Content:** JSON response (Pretty, Raw, Preview, Visualize buttons available)
- Headers Content:** (3) (highlighted in green)
- Test Results:** (highlighted in green)
- Status:** 200 OK
- Notes:** 상품 주문 조회 (Product Order Inquiry)
- Request URL:** GET → http://127.0.0.1:8000/order-service/{user_id}/orders

The JSON response body is displayed as:

```
1 [  
2 {  
3   "productId": "CATALOG-001",  
4   "qty": 10,  
5   "unitPrice": 1500,  
6   "totalPrice": 15000,  
7   "createdAt": "2021-02-04T16:23:19.842+00:00",  
8   "orderId": "f0bce669-fee4-43fa-a539-e15bf5351190"  
9 },
```