

Spring Cloud로 개발하는 마이크로서비스 애플리케이션



Microservices

+



Spring
Cloud

```
public static void main(String[] args)
```

```
</servlet>
```

```
<servlet-name>BoardController</servlet-name>
```

```
<servlet-class>com.joneconsulting.controller.BoardController</servlet-class>
```

```
<init-param>
```

```
<param-name>user_name</param-name>
```

```
<param-value>Kenneth Lee</param-value>
```

```
</init-param>
```

```
</servlet>
```

```
class Book
```

```
def self = self, title, price, author);
```

```
self.title = title
```

```
self.price = price
```

```
self.author = author
```

```
</servlet-class>
```

```
var fs = require('fs');
```

```
fs.readFile('/JONE.txt' /* 1 */,
```

```
function (err, data) {
```

```
console.log(data); // 3
```

```
});
```

```
@interface NextInnovationDelegate : NSObject <UIApplicationDelegate>
```

Part 1

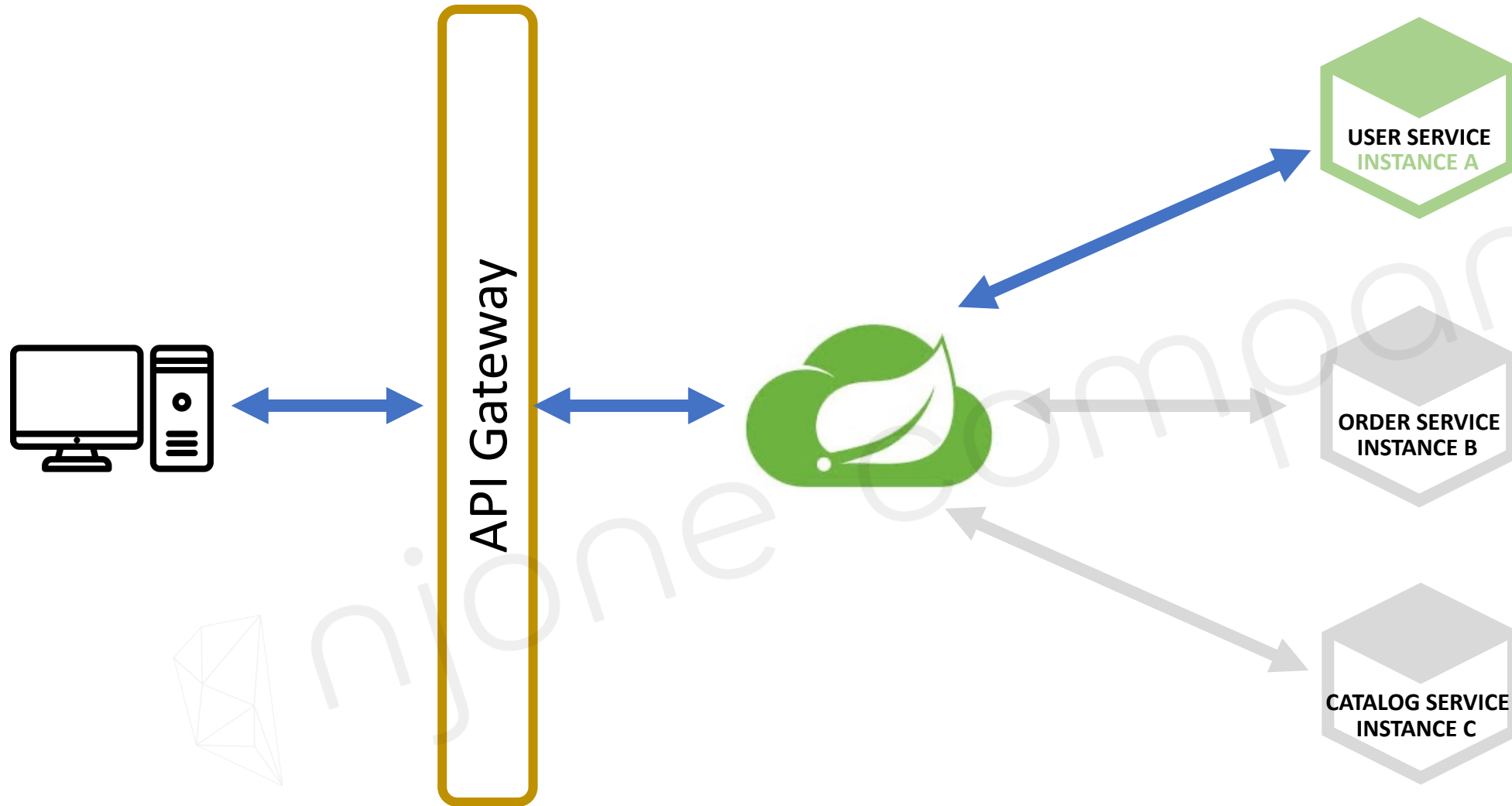
- Section 0: Microservice와 Sprig Cloud 소개
- Section 1: Service Discovery
- Section 2: API Gateway Service
- Section 3: E-commerce 애플리케이션
- **Section 4: Users Microservice - ①**
- Section 5: Catalogs, Orders Microservice
- Section 6: Users Microservice - ②
- Section 7: Configuration Service
- Section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

Section 4.

Users Microservice ①

- Users Microservice 개요
- Users Microservice - 프로젝트 생성
- Users Microservice - DB
- Users Microservice - 회원 가입
- Users Microservice - Security

Users Microservice



Users Microservice

■ 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

■ 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	/users/health_check	GET
환영 메시지	/user-service/users/welcome	/users/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 project

- Dependencies
 - DevTools, Lombok, Web, *Eureka Discovery Client*

The screenshot shows the Spring Boot Dependencies panel in an IDE. The left sidebar lists various dependency categories, with 'Spring Cloud Discovery' highlighted. The main panel shows a list of dependencies under the 'Spring Boot 2.4.1' version. 'Eureka Discovery Client' is checked. The right sidebar shows the 'Selected Dependencies' list, which includes 'Developer Tools' (Spring Boot DevTools, Lombok), 'Web' (Spring Web), and 'Spring Cloud Discovery' (Eureka Discovery Client). A red box highlights the 'Selected Dependencies' panel.

Dependencies	Spring Boot	Selected Dependencies
Developer Tools	2.4.1	Developer Tools
Web		Spring Boot DevTools
Template Engines		Lombok
Security		Web
SQL		Spring Web
NoSQL		Spring Cloud Discovery
Messaging		Eureka Discovery Client
I/O		
Ops		
Observability		
Testing		
Spring Cloud		
Spring Cloud Security		
Spring Cloud Tools		
Spring Cloud Config		
Spring Cloud Discovery		
Spring Cloud Routing		
Spring Cloud Circuit Breaker		

Users Microservice

- Application Class

```
@SpringBootApplication
@EnableDiscoveryClient
public class UserServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(UserServiceApplication.class, args);
    }

}
```


Users Microservice

- application.yml (or application.properties)

```
server:
```

```
  port: 0
```

```
spring:
```

```
  application:
```

```
    name: user-service
```

```
eureka:
```

```
  instance:
```

```
    instanceId: ${spring.application.name}:${spring.application.instance_id:${random.value}}
```

```
  client:
```

```
    register-with-eureka: true
```

```
    fetch-registry: true
```

```
    service-url:
```

```
      defaultZone: http://localhost:8761/eureka
```

Users Microservice

- RestController Class
 - 상태 체크 → */health_check*

```
@RestController
@RequestMapping("/")
public class UsersController {

    @GetMapping("/health_check")
    public String status() {
        return "It's Working in User Service";
    }
}
```

Users Microservice

- Eureka Discovery Service 등록

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
USER-SERVICE	n/a (1)	(1)	UP (1) - 10.90.1.91:a914ebf73f9f0946d0e675a04ba08ae2

← → ↺ ⚠ 주의 요함 10.90.1.91:56447/users/health_check
Working

Users Microservice

- Configuration 정보 추가
 - application.yml 파일에 Welcome message 추가
- UsersController Class
 - Environment 사용
 - @Value 사용

```
17 greeting:  
18 message: Welcome to the Simple E-Commerce.
```

```
18 private Environment env;  
19  
20 @Autowired  
21 public UsersController(Environment env) {  
22     this.env = env;  
23 }  
24  
25 @GetMapping("/health_check")  
26 public String status() { return "It's Working"; }  
29  
30 @GetMapping("/welcome")  
31 public String welcome() {  
32     return env.getProperty("greeting.message");  
33 }
```

Users Microservice

- Configuration 정보 추가

← → ↻ ⓘ 127.0.0.1:60143/welcome

Welcome to Simple E-Commerce.

- @Value 사용

```
greeting:  
  message: Welcome to Simple E-Commerce.
```

```
@Data  
@Component  
public class Greeting {  
    @Value("${greeting.message}")  
    private String message;  
}
```

```
@Autowired  
private Greeting greeting;
```

```
@GetMapping("/welcome")  
public String welcome() {  
    //  
    return env.getProperty("greeting.message");  
    return greeting.getMessage();  
}
```

Users Microservice – H2 Database

■ H2 Database

- 자바로 작성된 오픈소스 RDBMS
- Embedded, Server-Client 가능
- JPA 연동 가능

The screenshot displays the H2 Database GUI. On the left, a tree view shows the database structure: jdbc:h2:mem:testdb, INFORMATION_SCHEMA, Sequences, and Users. The main area shows a SQL statement: `SELECT * FROM ORDERS`. Below the statement, the query result is displayed as a table with 8 columns: ID, CREATED_AT, ORDER_ID, PRODUCT_ID, QTY, TOTAL_PRICE, UNIT_PRICE, and USER_ID. The result shows 2 rows of data. On the right, a 'Login' dialog box is open, showing the 'Generic H2 (Embedded)' settings. The fields include Driver Class (org.h2.Driver), JDBC URL (jdbc:h2:mem:testdb), User Name (sa), and Password. The 'Connect' button is highlighted.

SQL statement:

```
SELECT * FROM ORDERS
```

ID	CREATED_AT	ORDER_ID	PRODUCT_ID	QTY	TOTAL_PRICE	UNIT_PRICE	USER_ID
1	2020-10-30 03:04:03.415	eff14ca4-3ec2-4615-a0b5-2583f9f9d4c6	CATALOG-0001	10	15000	1500	95be247c-7812-4943-b0c9-47eaad744534
2	2020-10-30 03:05:18.99	e48c9ee1-b926-4b9a-b226-9189d16d59f0	CATALOG-0002	7	6300	900	95be247c-7812-4943-b0c9-47eaad744534

(2 rows, 12 ms)

Login dialog box fields:

- Saved Settings: Generic H2 (Embedded)
- Setting Name: Generic H2 (Embedded) [Save] [Remove]
- Driver Class: org.h2.Driver
- JDBC URL: jdbc:h2:mem:testdb
- User Name: sa
- Password: []
- [Connect] [Test Connection]

Users Microservice – H2 Database

■ Dependency 추가

- Project 생성 시 선택
- <https://mvnrepository.com/artifact/com.h2database/h2>

```
<!-- https://mvnrepository.com/artifact/com.h2database/h2 -->  
<dependency>  
  <groupId>com.h2database</groupId>  
  <artifactId>h2</artifactId>  
  <version>1.4.200</version>  
  <scope>runtime</scope>  
</dependency>
```

Users Microservice – H2 Database

- application.yml
 - h2 설정 추가

```
spring:
  application:
    name: user-service
  h2:
    console:
      enabled: true
      settings:
        web-allow-others: true
    path: h2-console
```


Users Microservice – H2 Database

- h2 1.4.198 이후 버전부터는 보안 문제로 자동으로 데이터베이스 생성하지 않음

English ▼ Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded) ▼

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:testdb

User Name: sa

Password:

Connect Test Connection

- 1.4.198 이전 버전 사용
- 데이터베이스 직접 생성

Database "mem:testdb" not found, either pre-create it or allow remote database creation (not recommended in secure environments) [90149-200] 90149/90149 (Help)

Users Microservice – H2 Database

- h2 1.4.198 이후 버전부터는 보안 문제로 자동으로 데이터베이스 생성하지 않음

```
<!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
```

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

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded) ▼

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:testdb

User Name: sa

Password:

Connect Test Connection

Test successful

Users Microservice – 회원 가입

■ 회원 가입

- POST → /users/

```
@PostMapping("/users")
public String createUser(@RequestBody RequestUser user) {
    return "Create user method is called";
}
```

```
@Data
public class RequestUser {
    private String email;
    private String pwd;
    private String name;
}
```

Users Microservice – 회원 가입

■ 회원 가입

- Validation check

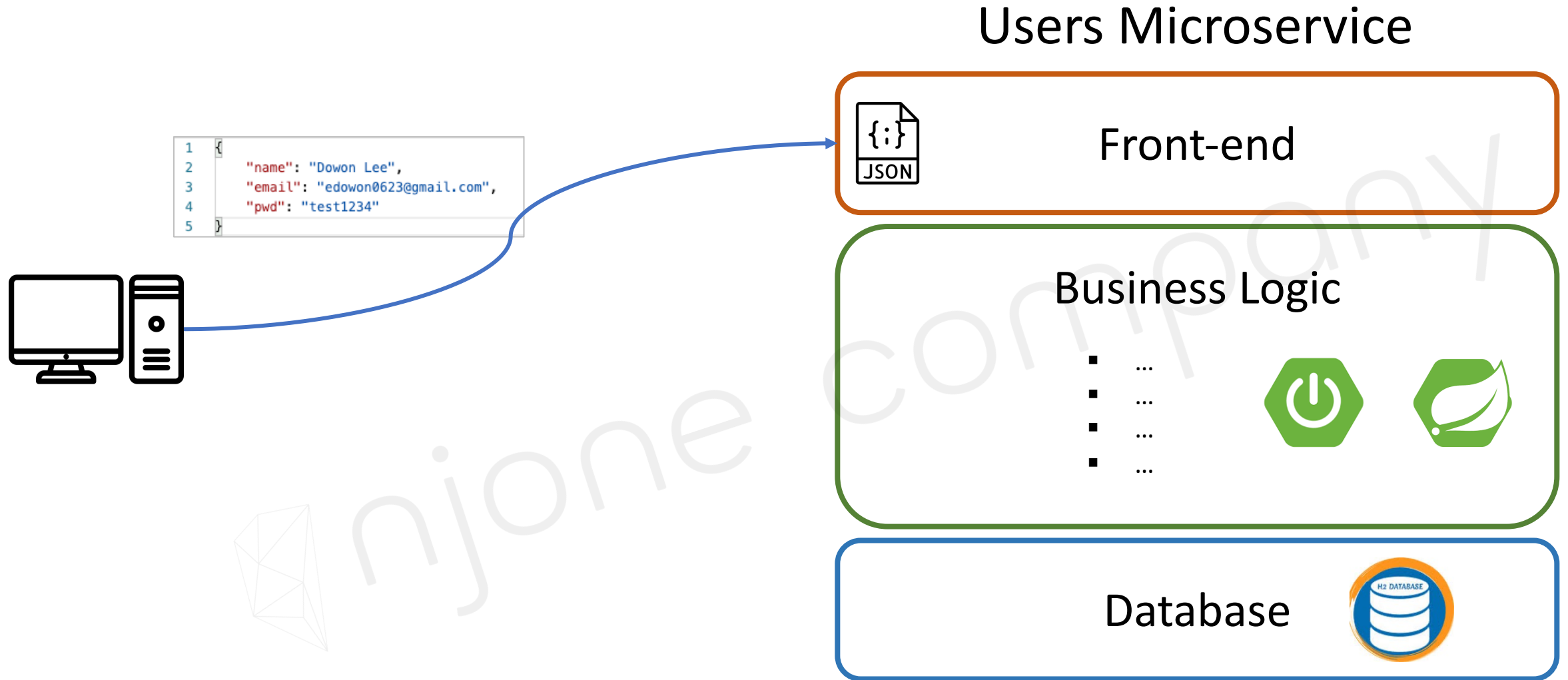
```
@Data
public class RequestUser {
    @NotNull(message = "Email cannot be null")
    @Size(min = 2, message = "Email not be less than two characters")
    @Email
    private String email;

    @NotNull(message = "Password cannot be null")
    @Size(min = 8, message = "Password must be equal or grater than 8 characters and less than 16 characters")
    private String pwd;

    @NotNull(message = "Name cannot be null")
    @Size(min = 2, message = "Name not be less than two characters")
    private String name;
}
```

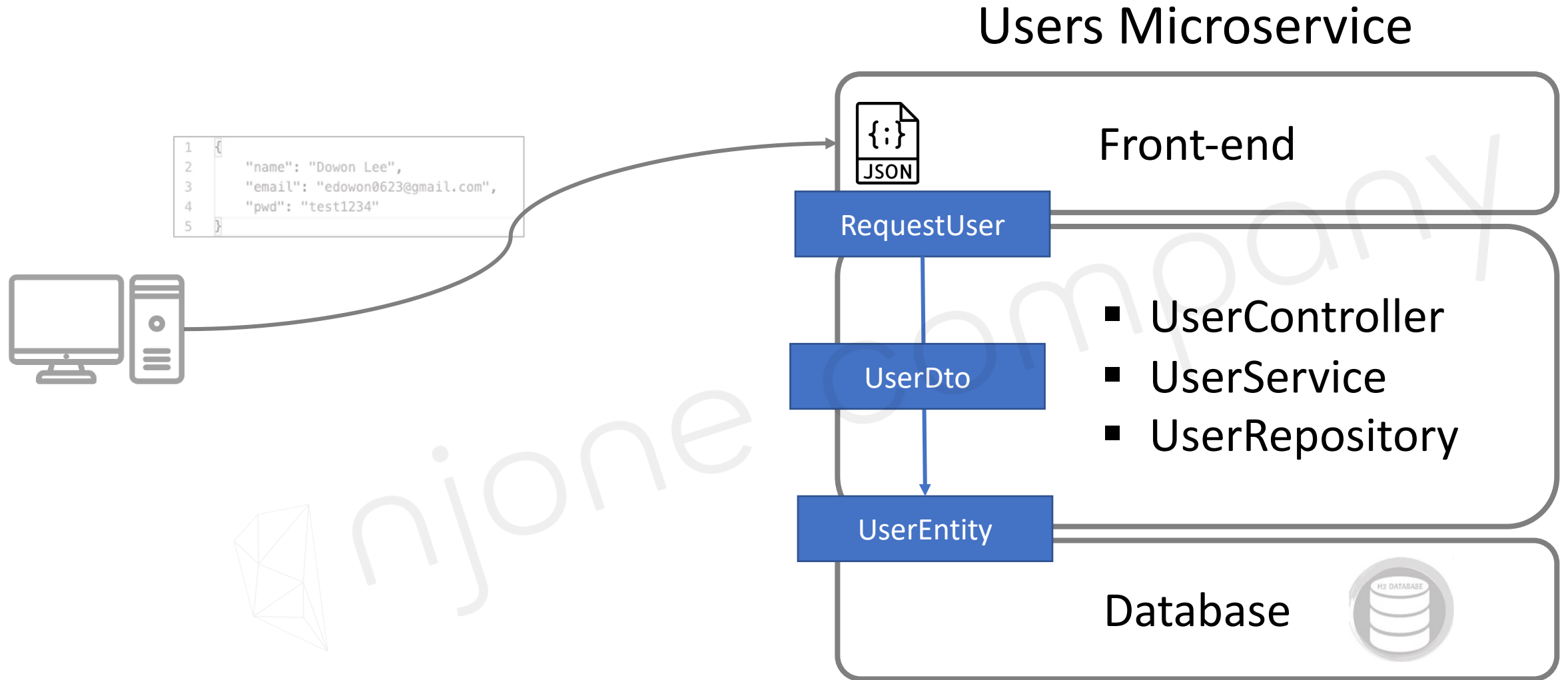
Users Microservice – 회원 가입

■ 회원 가입



Users Microservice – 회원 가입

- 회원 가입



Users Microservice – 회원 가입

- UserService.java, UserServiceImpl.java, UserDto.java

```
public interface UserService {  
    UserDto createUser(UserDto userDto);  
}
```

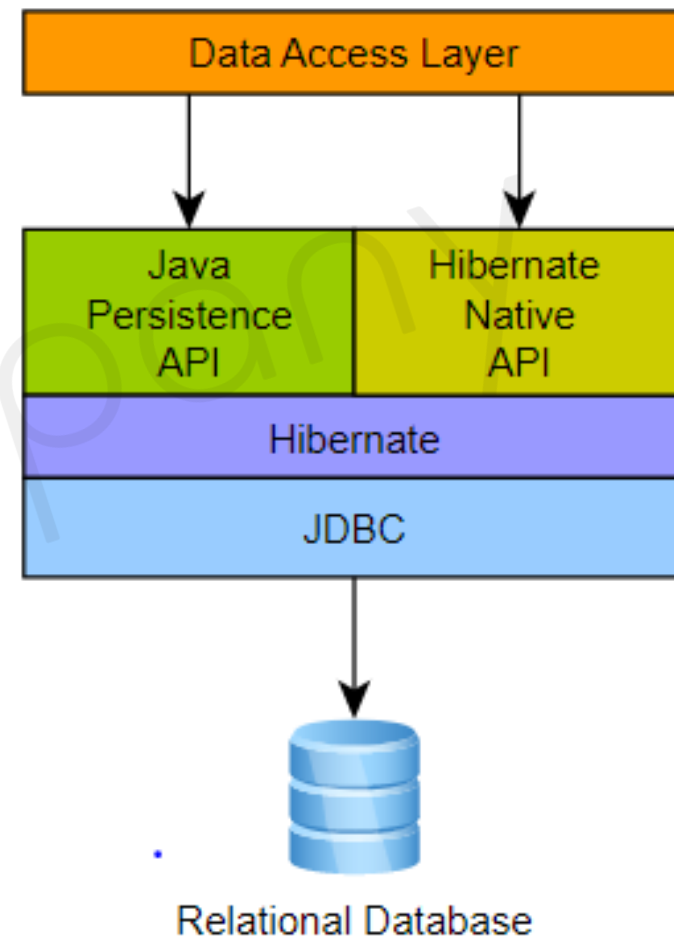
```
public class UsersServiceImpl implements UsersService {  
    @Override  
    public UserDto createUser(UserDto userDto) {  
        userDto.setUserId(UUID.randomUUID().toString());  
        return null;  
    }  
}
```

```
@Data  
public class UserDto {  
    private String email;  
    private String pwd;  
    private String name;  
    private String userId;  
    private Date createdAt;  
  
    private String encryptedPwd;  
}
```

Users Microservice – 회원 가입

■ JPA 설정

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```



Users Microservice – 회원 가입

■ UserEntity, UserRepository

```
8  @Data
9  @Entity
10 @Table(name="users")
11 public class UserEntity {
12     @Id
13     @GeneratedValue
14     private Long id;
15
16     @Column(nullable = false, length = 50, unique = true)
17     private String email;
18     @Column(nullable = false, length = 50)
19     private String name;
20     @Column(nullable = false, unique = true)
21     private String userId;
22     @Column(nullable = false, unique = true)
23     private String encryptedPwd;
```

```
public interface UserRepository extends CrudRepository<UserEntity, Long> {
}
```

Users Microservice – 회원 가입

- UserServiceImpl class

```
@Override
public UserDto createUser(UserDto userDto) {
    userDto.setUserId(UUID.randomUUID().toString());

    ModelMapper mapper = new ModelMapper();
    mapper.getConfiguration().setMatchingStrategy(MatchingStrategies.STRICT);
    UserEntity userEntity = mapper.map(userDto, UserEntity.class);
    userEntity.setEncryptedPwd("encrypted_password");
    userRepository.save(userEntity);

    return null;
}
```

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

Users Microservice – 회원 가입

- UserController class

```
@PostMapping
public String createUser(@RequestBody RequestUser user) {
    ModelMapper mapper = new ModelMapper();
    mapper.getConfiguration().setMatchingStrategy(MatchingStrategies.STRICT);

    UserDto userDto = mapper.map(user, UserDto.class);
    userService.createUser(userDto);
    return "Create user method is called";
}
```

Users Microservice – 회원 가입

■ 실행

The screenshot displays a REST client interface for a POST request to `http://192.168.0.8:63186/users`. The request body is a JSON object containing user details. Below the request, the response body shows a confirmation message. To the right, an H2 database console window shows the execution of a SQL query to retrieve all users from the `USERS` table, displaying the newly created user record.

REST Client Request:

```
POST http://192.168.0.8:63186/users
```

Request Body (JSON):

```
{  "name": "Dowon Lee",  "email": "edowon0623@gmail.com",  "pwd": "test1234"}
```

Response Body:

```
1 create user method is called
```

H2 Database Console:

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM USERS
```

Query Results:

ID	EMAIL	ENCRYPTED_PWD	NAME	USER_ID
1	edowon0623@gmail.com	encrypted_password	Dowon Lee	6816e80c-6e22-40a4-af6d-7f6f3f07af50

(1 row, 5 ms)

Edit

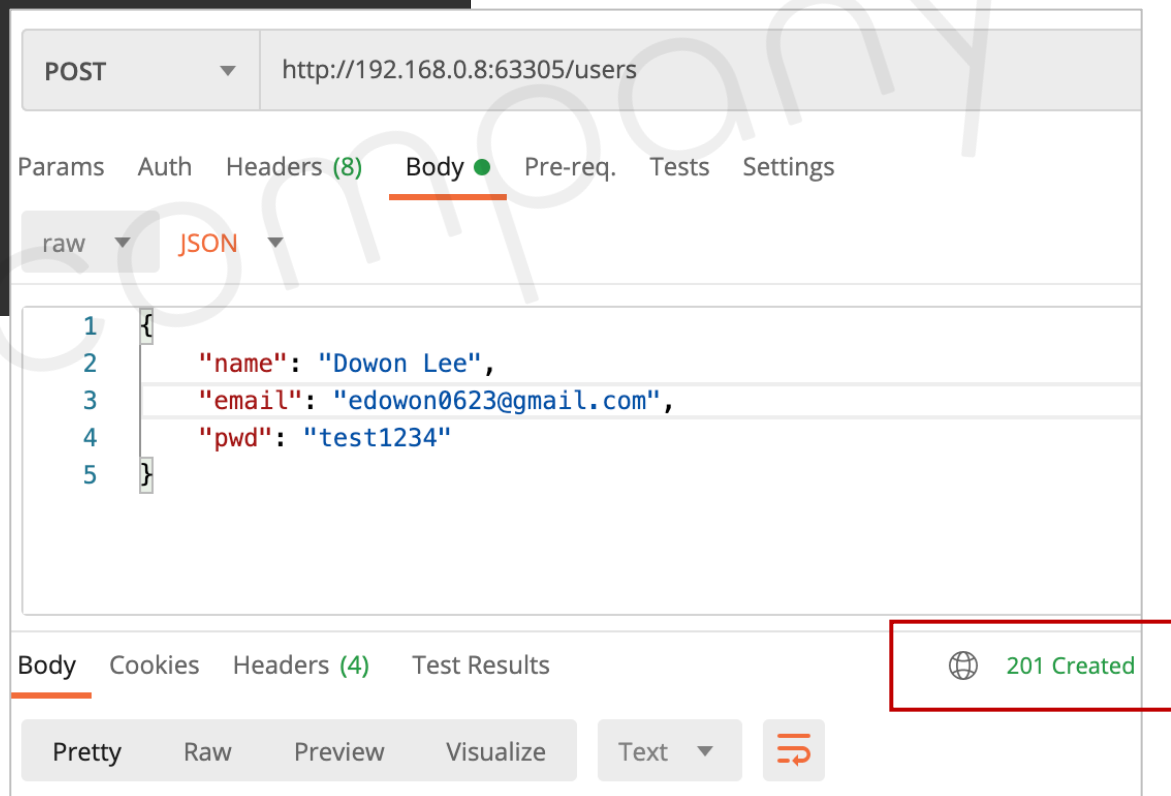
Users Microservice – 회원 가입

■ Http Status Code 변경

```
@PostMapping
public ResponseEntity createUser(@RequestBody RequestUser user) {
    IMapper mapper = new IMapper();
    mapper.getConfiguration().setMatchingStrategy(MatchingStrategies.STRICT);

    UserDto userDto = mapper.map(user, UserDto.class);
    userService.createUser(userDto);

    return new ResponseEntity(HttpStatus.CREATED);
}
```



The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://192.168.0.8:63305/users
- Body:** JSON format with the following content:

```
1 {
2   "name": "Dowon Lee",
3   "email": "edowon0623@gmail.com",
4   "pwd": "test1234"
5 }
```
- Status:** 201 Created (highlighted in a red box)
- Response Body:** Pretty, Raw, Preview, Visualize, Text (dropdown), and a refresh icon.

Users Microservice – 회원 가입

- ResponseUser.java, UserController.java

```
@Data
public class ResponseUser {
    private String email;
    private String name;
    private String userId;
}
```

```
@PostMapping
public ResponseEntity<ResponseUser> createUser(@RequestBody RequestUser user) {
    ModelMapper mapper = new ModelMapper();
    mapper.getConfiguration().setMatchingStrategy(MatchingStrategies.STRICT);

    UserDto userDto = mapper.map(user, UserDto.class);
    userService.createUser(userDto);

    ResponseUser responseUser = mapper.map(userDto, ResponseUser.class);

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

Users Microservice – 회원 가입

■ UserServiceImpl.java

```
@Override
public UserDto createUser(UserDto userDto) {
    userDto.setUserId(UUID.randomUUID().toString());

    ModelMapper mapper = new ModelMapper();
    mapper.getConfiguration().setMatchingStrategy(MatchingStrategies.STRICT);
    UserEntity userEntity = mapper.map(userDto, UserEntity.class);
    userEntity.setEncryptedPwd("encrypted_password");
    userRepository.save(userEntity);

    UserDto userVo = mapper.map(userEntity, UserDto.class);

    return userVo;
}
```

Body Cookies Headers (5) Test Results 201 Created

Pretty Raw Preview Visualize JSON

```
1 {
2   "email": "edowon0623@gmail.com",
3   "name": "Dowon Lee",
4   "userId": "2adcf98f-f44b-46ce-9eb3-564e0507df0e"
5 }
```

Users Microservice – Security

- Spring Security

- ***Authentication + Authorization***

- Step 1: 애플리케이션에 ***spring security jar***을 Dependency에 추가
- Step 2: ***WebSecurityConfigurerAdapter***를 상속받는 ***Security Configuration*** 클래스 생성
- Step 3: Security Configuration 클래스에 ***@EnableWebSecurity*** 추가
- Step 4: Authentication → ***configure(AuthenticationManagerBuilder auth)*** 메서드를 재정의
- Step 5: Password encode를 위한 ***BCryptPasswordEncoder*** 빈 정의
- Step 6: Authorization → ***configure(HttpSecurity http)*** 메서드를 재정의

Users Microservice – Security

■ Dependency 추가

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

■ WebSecurity class

```
@Configuration
@EnableWebSecurity
public class WebSecurity extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable();
        http.authorizeRequests().antMatchers(...antPatterns: "/users/**").permitAll();
        http.headers().frameOptions().disable();
    }
}
```



!! `http.header().frameOptions().disable()`
추가하지 않으면, h2-console 접근 안됨

Users Microservice – Security

■ 실행

```
2021-01-16 09:10:31.309 WARN 31652 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration :  
2021-01-16 09:10:31.374 INFO 31652 --- [ restartedMain] .s.s.UserDetailsServiceAutoConfiguration :
```

Using generated security password: cdd051b6-2692-4655-9967-1b02bc4d0363



Users Microservice – Security

- BCryptPasswordEncoder

- Password를 해싱하기 위해 Bcrypt 알고리즘 사용
- 랜덤 Salt를 부여하여 여러번 Hash를 적용한 암호화 방식

- UserServiceImpl.java

```
@Autowired
BCryptPasswordEncoder passwordEncoder;

@Override
public UserDto createUser(UserDto userDto) {
    userDto.setUserId(UUID.randomUUID().toString());
    userDto.setEncryptedPwd(passwordEncoder.encode(userDto.getPwd()));
}
```

- UserServiceApplication.java

```
@Bean
public BCryptPasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Users Microservice – Security

■ 실행

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

Body Cookies Headers (10) Test Results 201 Created

Pretty Raw Preview Visualize JSON

```
1 {  
2   "email": "edowon0623@gmail.com",  
3   "name": "Dowon Lee",  
4   "userId": "36c3914a-b664-49be-8801-4ea5dc1b657e"  
5 }
```

Wrap Line

SELECT * FROM USERS;

ID	EMAIL	ENCRYPTED_PWD	NAME	USER_ID
1	edowon0623@gmail.com	\$2a\$10\$X.8l32R6Y3N5t9kz8AOtJOVpNINzUjILhvJyAtI/E/cLalsE6M/L.	Dowon Lee	36c3914a-b664-49be-8801-4ea5dc1b657e

(1 row, 4 ms)