MyBatis 실습

▼ MyBatis?

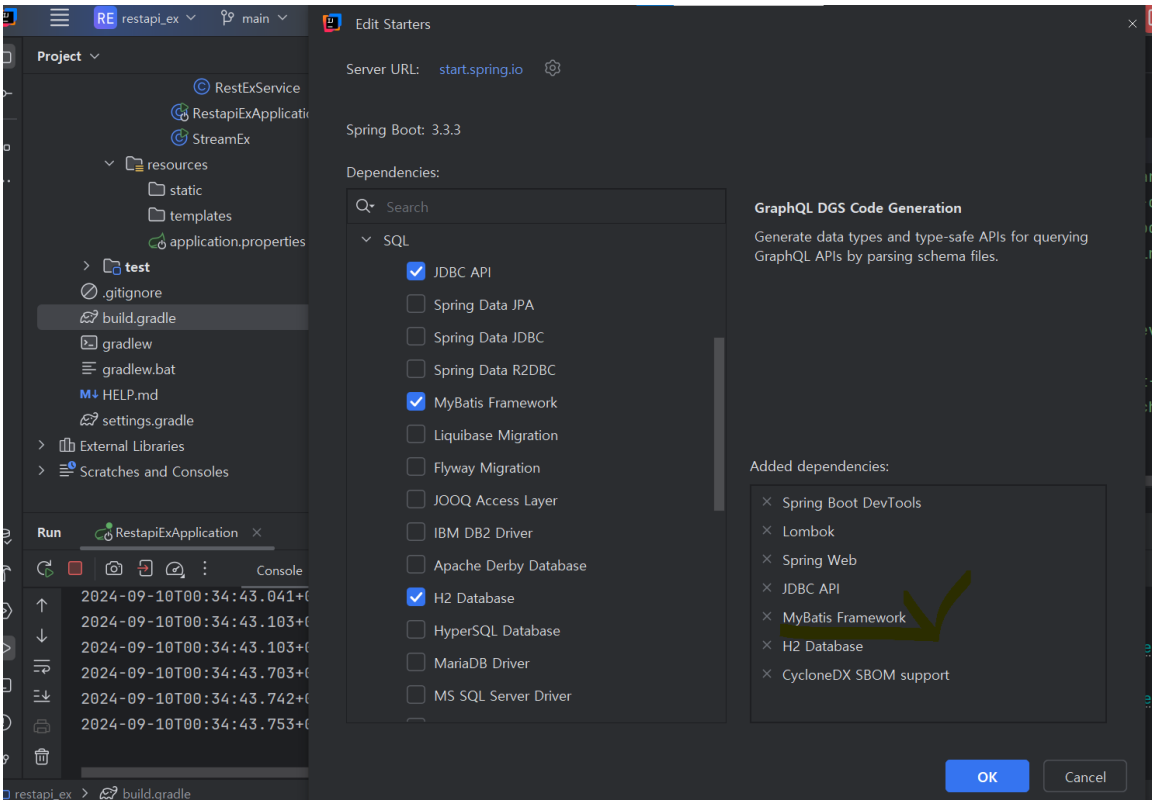
MyBatis는 위에서 설명한 기술 중 **JPA**와 비슷하게 객체와 관계형 데이터베이스 간의 매핑을 지원하는 **ORM 프레임워크**에 가까우면서도, JPA와는 다른 방식으로 SQL을 직접적으로 다루는 특성을 갖고 있습니다. 아래에서 MyBatis를 설명하고 위의 기술들과 비교하겠습니다.

MyBatis

- **특징**:
  - **SQL을 직접 작성**하고 이를 자바 객체와 매핑하는 기능을 제공.
  - 자바의 객체와 데이터베이스 간의 매핑을 XML 파일 또는 어노테이션을 통해 명시적으로 정의할 수 있음.
  - SQL을 사용하여 세밀한 쿼리 조작이 가능하고, 복잡한 쿼리도 손쉽게 처리할 수 있음.
  - 단순한 CRUD 작업뿐만 아니라 복잡한 SQL 쿼리를 사용할 때 매우 유리함.
  - 완전한 ORM(예: JPA)과는 달리, SQL을 직접 다루기 때문에 SQL 튜닝이나 고도의 쿼리 최적화가 필요할 때 유리.
- **MyBatis의 위치**:
  - **JDBC와 JPA** 사이에 위치한다고 볼 수 있습니다.
    - **JDBC**처럼 SQL을 직접 작성해야 하지만, JDBC의 반복적인 작업을 템플릿 방식으로 줄여줍니다.
    - **JPA**처럼 객체와 테이블 간 매핑을 지원하지만, JPA처럼 완전한 ORM을 제공하지 않고 개발자가 SQL을 관리할 수 있는 유연성을 줍니다.
- **비교**:
  - **JDBC**: 순수 JDBC보다 사용하기 편리하며, XML 또는 어노테이션을 사용해 쿼리를 작성함으로써 코드의 가독성을 높일 수 있습니다. 커넥션 관리나 예외 처리도 보다 간편합니다.
  - **JPA**: JPA와 달리 SQL을 직접 작성해야 하지만, 이로 인해 복잡한 SQL 작업이나 성능 최적화가 필요한 경우 MyBatis가 더 유리합니다. JPA는 객체 중심으로 동작하고 SQL을 자동으로 생성하지만, MyBatis는 SQL을 개발자가 더 세밀하게 제어할 수 있습니다.

    - **스프링 데이터 JPA**: 스프링 데이터 JPA는 메서드 이름만으로 쿼리를 자동 생성하는 반면, MyBatis는 SQL을 직접 제어할 수 있어서 복잡한 비즈니스 로직에 필요한 쿼리 작성이 용이합니다.

- build.gradle 의존성 추가 : MyBatis Framework



- application.properties mybatis설정파일(xml) 위치 설정

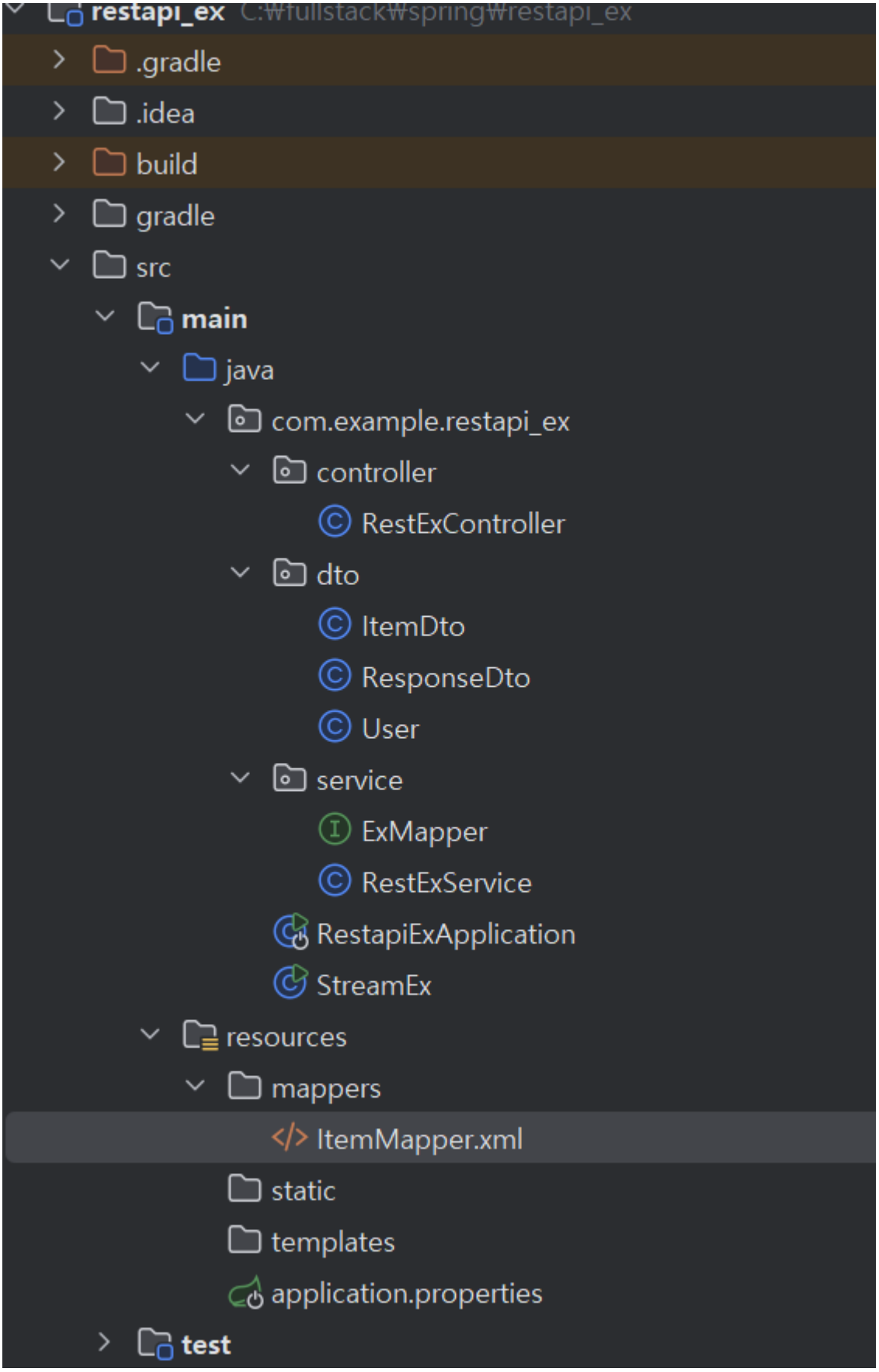
```
# spring.datasource.url=jdbc:h2:mem:testdb # 메모리에 testbd에 접속
spring.datasource.url=jdbc:h2:~/testdb  # 로컬에서 유저폴더/testbd.mv.db 파일
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
# H2 데이터베이스의 웹 콘솔 활성화
# http://localhost:8080/h2-console 경로에서 접근 가능
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update
```

```
mybatis.mapper-locations = mappers/*.xml
```

▼ application.properties이름을 application.yml로 변경후 아래 내용으로 설정해도 됨

```
spring:
    h2:
            console:
                    enabled: true

datasource:
    hikari:
            driver-class-name: org.h2.Driver
            jdbc-urlL jdbc:h2:~/testdb
            username: sa
            password:

mybatis:
    mapper-locations:
        - mappers/*xml
```

▼ src >main > r**esources > mappers** 폴더 만들고 **ItemMapper.xml** 파일 만들기



▼ com.example.restapi_ex.mapper 패키지에 ExMapper 인터페이스 추가

@Mapper 애너테이션 하고,
DB에서 ID로 값을 가져오는 findById 메서드 정의

```
package com.example.restapi_ex.mapper;

import org.apache.ibatis.annotations.Mapper;

import java.util.HashMap;

@Mapper
public interface ExMapper {

    HashMap<String, Object> findById(HashMap<String, Objec
    // 이 Mapper 인터페이스의 findById 메서드가 호출되면 xml의 쿼
}
```

▼ ExService에서 위 인터페이스의 메서드 사용

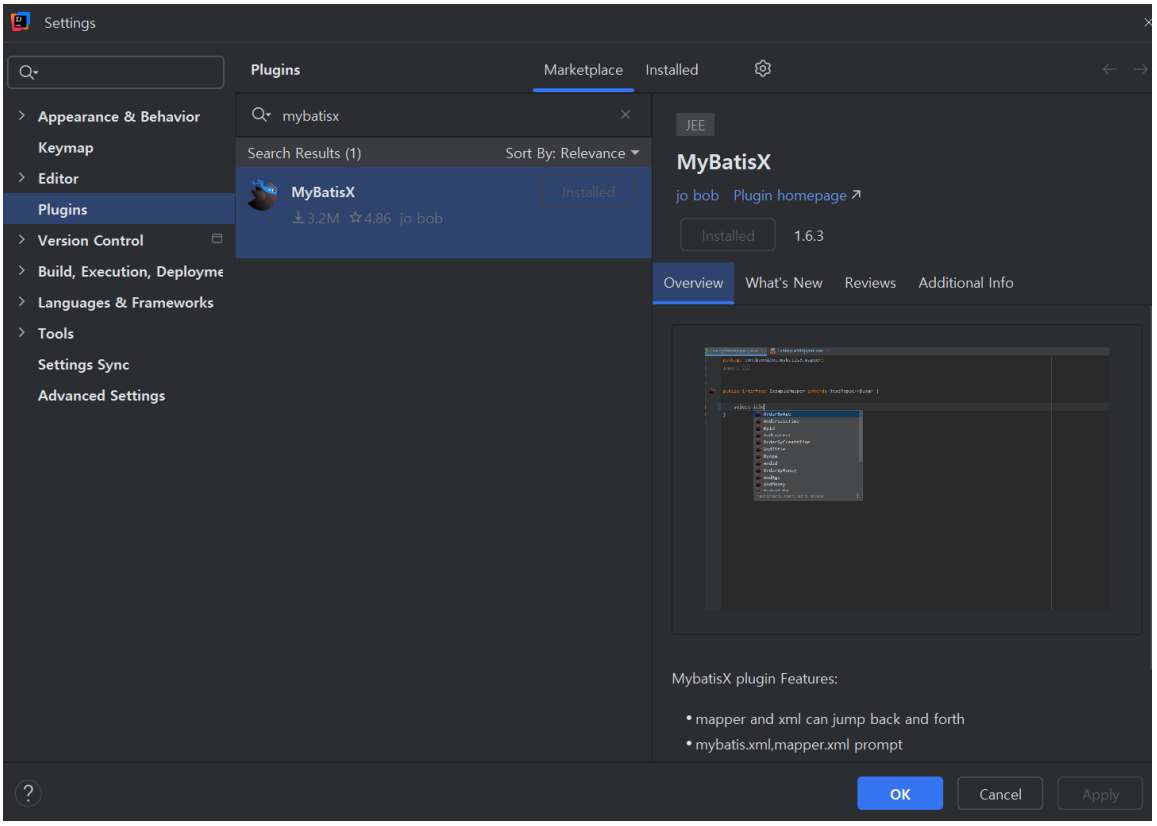
```
package com.example.restapi_ex.service;

import com.example.restapi_ex.dto.ItemDto;
import com.example.restapi_ex.mapper.ExMapper;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowi
import org.springframework.stereotype.Service;

import java.util.HashMap;

@Service
@Slf4j
public class RestExService {

    @Autowired
    private ExMapper exMapper;

    public boolean registerItem(ItemDto itemDto){
        // DB insert
```

```
        log.info("service...");

        return true; // DB insert가 성공했을 경우 true
    }

    public ItemDto getItemById(String id){
        HashMap<String, Object> paramMap = new HashMap<>()
        paramMap.put("id", id);

        HashMap<String, Object> res = exMapper.findById(pa

        // 일단 여기까지 작성하고 mapper 쿼리 작성
    }
}
```

▼ resources > mappers > ItemMapper.xml 쿼리문 작성

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//E
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.example.restapi_ex.mapper.ExMapper"

    <select id="findById"  parameterType="hashmap" resultT
        SELECT ID, NAME FROM ITEM WHERE ID = #{id}
    </select>

</mapper>
```

▼ IntelliJ 메뉴 File > Setting > Plugin 에서 MyBatisX 라는 플러그인을 설치하면 mapper.xml의 쿼리문과 매핑된 mapper 인터페이스의 메서드를 바로 확인 가능



▼ ExService에서 호출하는 부분 마저 작업

```
package com.example.restapi_ex.service;

import com.example.restapi_ex.dto.ItemDto;
import com.example.restapi_ex.mapper.ExMapper;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowi
import org.springframework.stereotype.Service;

import java.util.HashMap;

@Service
@Slf4j
public class RestExService {

    @Autowired
    private ExMapper exMapper;

    public boolean registerItem(ItemDto itemDto){
        // DB insert
```

```
        log.info("service...");

        return true; // DB insert가 성공했을 경우 true
    }

    public ItemDto getItemById(String id){
        HashMap<String, Object> paramMap = new HashMap<>()
        paramMap.put("id", id);

        HashMap<String, Object> res = exMapper.findById(pa

                // ItemDto 타입 리턴할 것이므로 ItemDto하나 만들
        ItemDto itemDto = new ItemDto();
        itemDto.setId((String)res.get("ID"));
        itemDto.setName((String)res.get("NAME"));

        return itemDto;
    }
}
```

▼ 이제, 실제 Service를 호출해봐야 되니까, 컨트롤러에서 작업
마지막에 GetMapping 추가

```
@GetMapping("/item")
   public ItemDto getItem(@RequestParam("id") String id) {
       ItemDto res = restExService.getItemById(id);
       return res;
   }
```

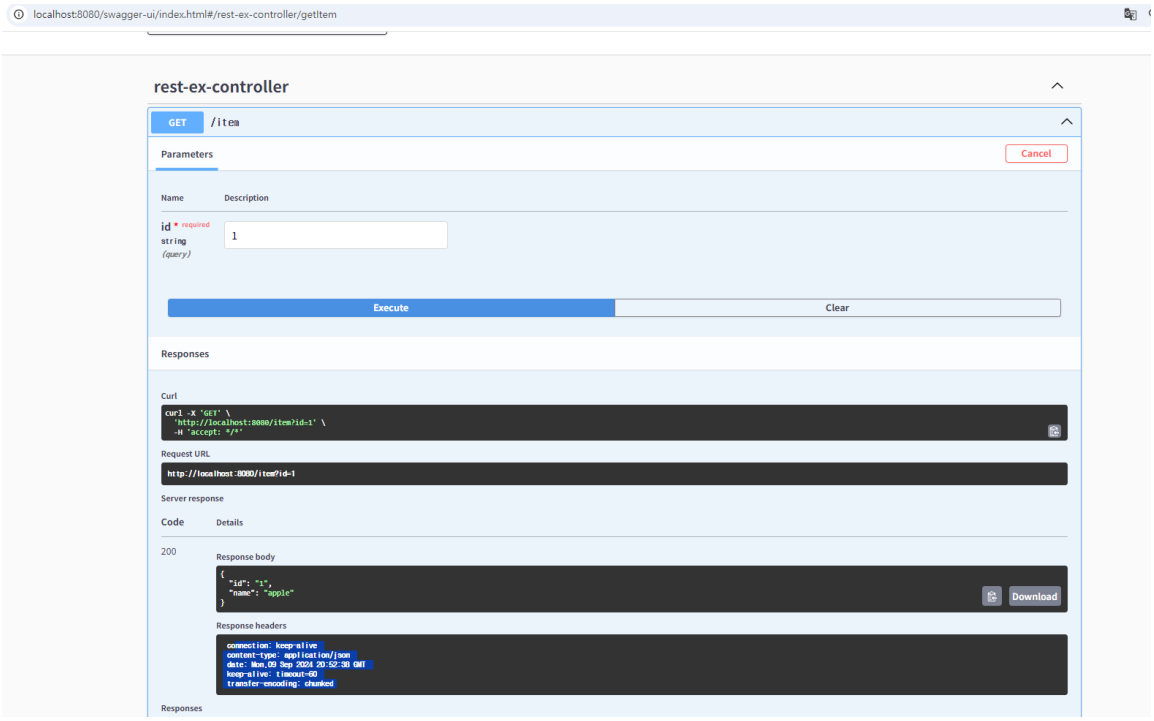
```
package com.example.restapi_ex.controller;

import com.example.restapi_ex.dto.ItemDto;
import com.example.restapi_ex.dto.ResponseDto;
import com.example.restapi_ex.dto.User;
import com.example.restapi_ex.service.RestExService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowi
import org.springframework.web.bind.annotation.*;
```

```
@RestController
@Slf4j
public class RestExController {
    //의존성 주입
    @Autowired
    private RestExService restExService;

    // http://localhost:8080/test
    @GetMapping("/test")
    public String test(){
        log.info("test");
        return "{}";
    }

    // http://localhost:8080/test2
    @GetMapping("/test2")
    public String test2(){
        log.info("test2");
        return "test2";
    }

    // http://localhost:8080/param?name=Spring
    // @RequestParam을 사용하여 쿼리 파라미터 값을 가져옴
    @GetMapping("/param")
    public String testRequestParam(@RequestParam String nam
        log.info("RequestParam: " + name);
        return "Hello, " + name;
    }

    // http://localhost:8080/path/Spring
    // @PathVariable을 사용하여 URL 경로에서 값을 가져옴
    @GetMapping("/path/{name}")
    public String testPathVariable(@PathVariable String nam
        log.info("PathVariable: " + name);
        return "Path Variable: " + name;
    }

    // http://localhost:8080/body
```

```
    // @RequestBody를 사용하여 요청 본문 데이터를 객체로 바인딩
    @PostMapping("/body")
    public String testRequestBody(@RequestBody User user)
        log.info("RequestBody: " + user);
        return "Request Body: " + user;
    }

    // http://localhost:8080/item
    // @RequestBody를 사용하여 요청 본문 데이터를 객체로 바인딩
    // JSON 타입에 해당하는 매핑이 되는 DTO 클래스를 먼저 만들어 준
    // 그 DTO 클래스에 데이터를 채워서 주고 받음
    @PostMapping("/item")
    public ResponseDto testRequestBody(@RequestBody ItemDt
        log.info("item: " + item);

        boolean b = restExService.registerItem(item);
        if( b ){
            ResponseDto responseDto = new ResponseDto();
            responseDto.setMessage("ok");
            return responseDto;
        }

        ResponseDto responseDto = new ResponseDto();
        responseDto.setMessage("fail");
        return responseDto;
    }

    @GetMapping("/item")
    public ItemDto getItem(@RequestParam("id") String id)
        ItemDto res = restExService.getItemById(id);
        return res;
    }
}
```

▼ 테스트 해보기 : http:localhost:8080/swagger-ui/index.html

- DB에 값이 없을 경우는 예외처리 하기



▼ Service의 registerItem메서드도 마저 작성해 보기

```
package com.example.restapi_ex.service;

import com.example.restapi_ex.dto.ItemDto;
import com.example.restapi_ex.mapper.ExMapper;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowi
import org.springframework.stereotype.Service;

import java.util.HashMap;

@Service
@Slf4j
public class RestExService {

    @Autowired
    private ExMapper exMapper;

    public boolean registerItem(ItemDto itemDto){
        // DB insert
        HashMap<String, Object> paramMap = new HashMap<>()
```

```
        paramMap.put("id", itemDto.getId());
        paramMap.put("name", itemDto.getName());

        exMapper.registerItem(paramMap);
        // registerItem 에서 alt+enter 누르면 자동으로 mapper에

        log.info("service...");

        return true; // DB insert가 성공했을 경우 true
    }

    public ItemDto getItemById(String id){
        HashMap<String, Object> paramMap = new HashMap<>()
        paramMap.put("id", id);

        HashMap<String, Object> res = exMapper.findById(pa

        ItemDto itemDto = new ItemDto();
        itemDto.setId((String)res.get("ID"));
        itemDto.setName((String)res.get("NAME"));

        return itemDto;
    }
}
```

▼ RestExService의 registerItem메서드에서 alt+enter 눌러서 자동으로 만들어진 ExMapper 메서드 확인

```
package com.example.restapi_ex.mapper;

import org.apache.ibatis.annotations.Mapper;

import java.util.HashMap;

@Mapper
public interface ExMapper {
```

```
    HashMap<String, Object> findById(HashMap<String, Objec
    // 이 Mapper 인터페이스의 findById 메서드가 호출되면 xml의 쿼

    void registerItem(HashMap<String, Object> paramMap);

}
```

▼ ExMapper 인터페이스의 메서드 이름 그대로 copy해서 mapper 쿼리에 추가
resources > mappers > ItemMapper.xml

```
<insert id="registerItem" parameterType="hashmap">
    INSERT INTO ITEM(ID, NAME)    VALUES( #{id}, #{name} )
</insert>
```

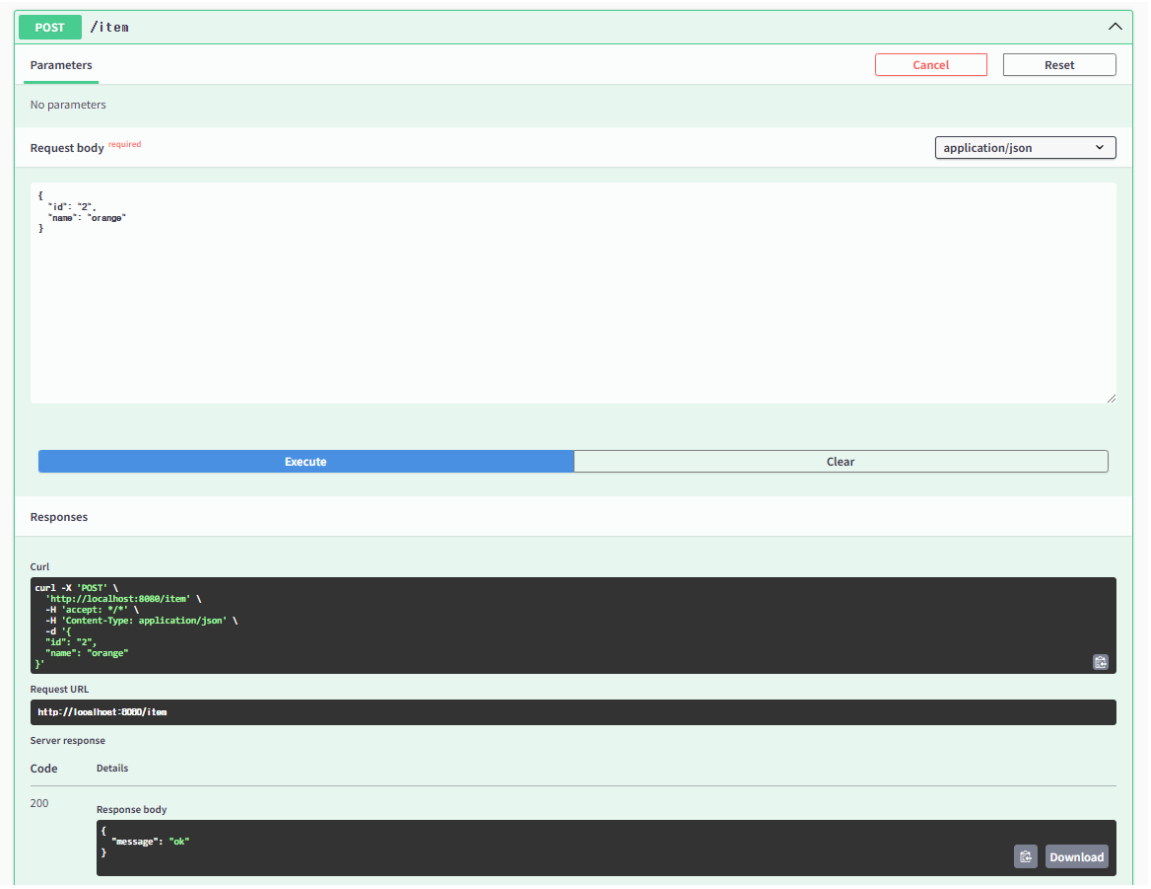
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//E
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.example.restapi_ex.mapper.ExMapper"

    <select id="findById"  parameterType="hashmap" resultT
        SELECT
            ID, NAME
        FROM ITEM
        WHERE ID = #{id}
    </select>

    <insert id="registerItem" parameterType="hashmap">
        INSERT INTO ITEM(ID, NAME)
        VALUES( #{id}, #{name} )
    </insert>

</mapper>
```

▼ swagger-ui/index.html에서 테스트

Post : /item
{
“id”:”2”,

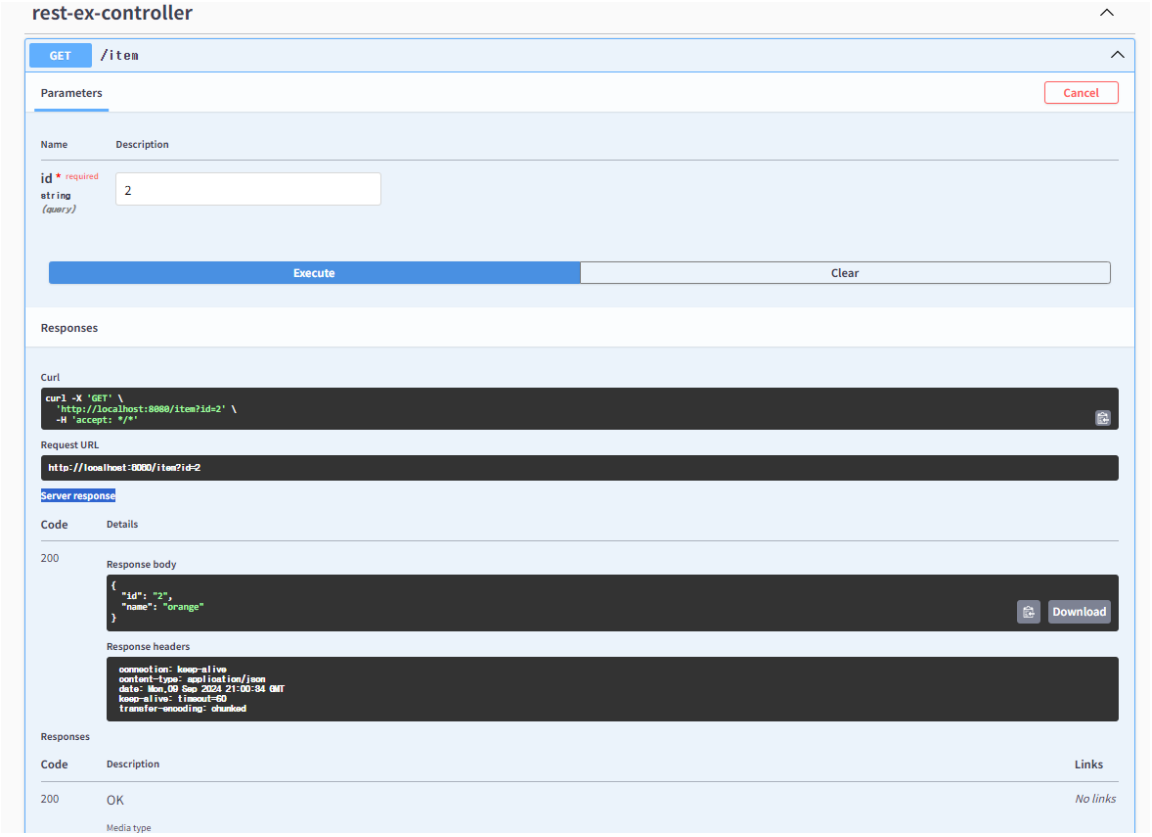
“name”:”orange

}



▼ Get: /item

id: 2 가져오는지도 확인



▼ [만약] registerItem의 파라미터 타입을 ItemDto로 변경한다면

▼ ItemMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.example.restapi_ex.mapper.ExMapp

    <select id="findById"  parameterType="hashmap" resu
        SELECT
            ID, NAME
        FROM ITEM
        WHERE ID = #{id}
    </select>

<!--    <insert id="registerItem" parameterType="hashma
<!--        INSERT INTO ITEM(ID, NAME)-->
<!--        VALUES( #{id}, #{name} )-->
```

```
<!--    </insert>-->
    <!-- parameterType을 ItemDto로 변경 -->
    <insert id="registerItem" parameterType="com.exampl
        INSERT INTO ITEM(ID, NAME)
        VALUES( #{id}, #{name} )
    </insert>

</mapper>
```

▼ ExMapper인터페이스

```
package com.example.restapi_ex.mapper;

import com.example.restapi_ex.dto.ItemDto;
import org.apache.ibatis.annotations.Mapper;

import java.util.HashMap;

@Mapper
public interface ExMapper {

    HashMap<String, Object> findById(HashMap<String, Ob
    // 이 Mapper 인터페이스의 findById 메서드가 호출되면 xml의

    void registerItem(ItemDto itemDto);

}
```

▼ RestExService

```
package com.example.restapi_ex.service;

import com.example.restapi_ex.dto.ItemDto;
import com.example.restapi_ex.mapper.ExMapper;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Aut
import org.springframework.stereotype.Service;
```

```
import java.util.HashMap;

@Service
@Slf4j
public class RestExService {

    @Autowired
    private ExMapper exMapper;

    public boolean registerItem(ItemDto itemDto){
        // DB insert
//        HashMap<String, Object> paramMap = new HashMa
//
//        paramMap.put("id", itemDto.getId());
//        paramMap.put("name", itemDto.getName());
//
//        exMapper.registerItem(paramMap);
        exMapper.registerItem(itemDto);
        // registerItem 에서 alt+enter 누르면 자동으로 mapp

        log.info("service...");

        return true; // DB insert가 성공했을 경우 true
    }

    public ItemDto getItemById(String id){
        HashMap<String, Object> paramMap = new HashMap<
        paramMap.put("id", id);

        HashMap<String, Object> res = exMapper.findById

        ItemDto itemDto = new ItemDto();
        itemDto.setId((String)res.get("ID"));
        itemDto.setName((String)res.get("NAME"));

        return itemDto;
    }
}
```

[확인]

▼ build.gradle 의존성

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-s
    implementation group: 'org.springdoc', name: 'springdo
    implementation 'org.mybatis.spring.boot:mybatis-spring
    testImplementation 'org.mybatis.spring.boot:mybatis-sp
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-
    runtimeOnly 'com.h2database:h2'
    testImplementation 'org.springframework.boot:spring-bo
    testRuntimeOnly 'org.junit.platform:junit-platform-lau
}
```

▼ application.properties

```
# spring.datasource.url=jdbc:h2:mem:testdb
# 기본 디렉터리 사용자디렉터리\testdb.mv.db
spring.datasource.url=jdbc:h2:~/testdb
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
# http://localhost:8080/h2-console 로 확인 가능
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update

mybatis.mapper-locations=mappers/*.xml
```