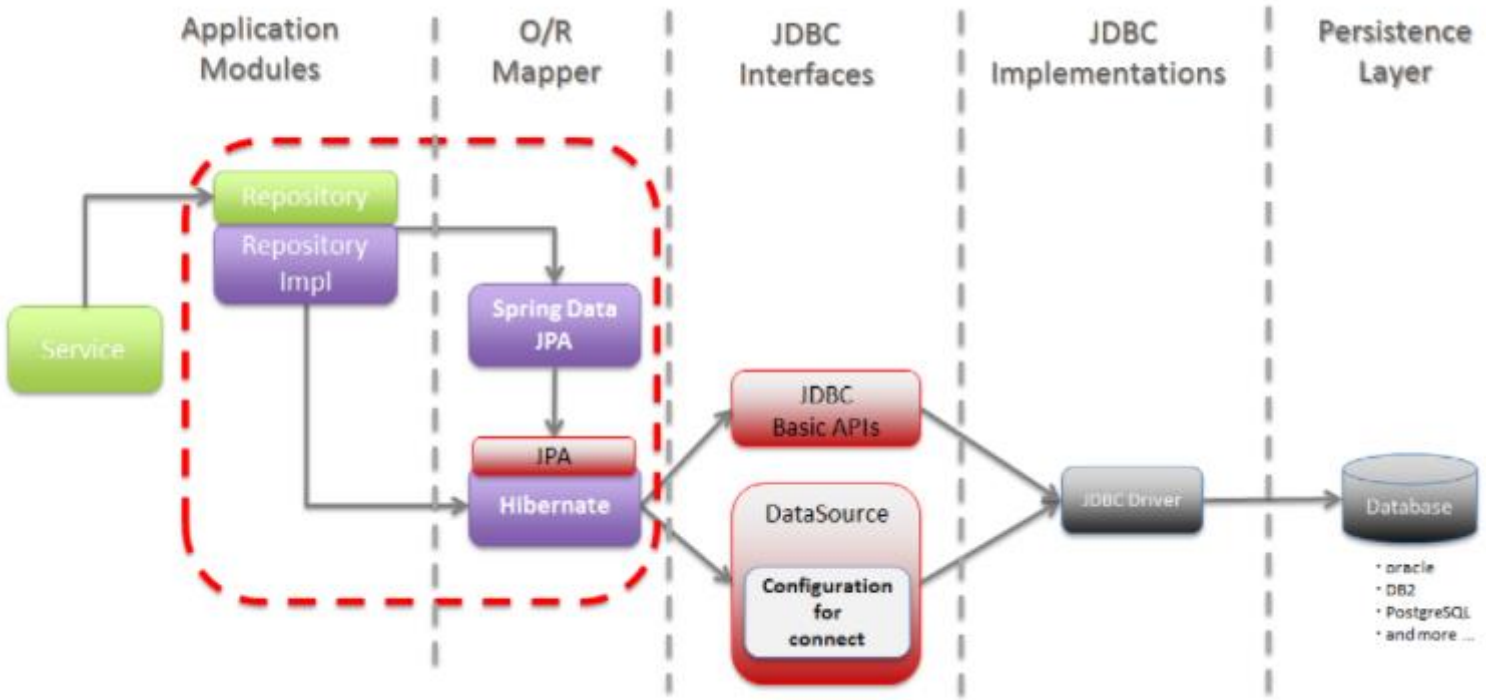
4. 스프링 JPA

목차

1. Spring Data JPA

❖ JPA(Java Persistence API)

- 자바 어플리케이션에서 관계형 데이터베이스를 사용하는 방식을 정의한 인터페이스
- 자바 ORM 기술에 대한 표준 명세로, JAVA에서 제공하는 API, 스프링에서 제공(X)
- 자바 클래스와 DB테이블을 매핑(sql을 매핑하지 않음)



1. Spring Data JPA

❖ SQL Mapper와 ORM

- ORM은 DB 테이블을 자바 객체로 매핑함으로써 객체간의 관계를 바탕으로 SQL을 자동으로 생성하지만 Mapper는 SQL을 명시해주어야 한다.
- ORM은 RDB의 관계를 Object에 반영하는 것이 목적이라면, Mapper는 단순히 필드를 매핑시키는 것이 목적이라는 점에서 지향점의 차이가 있음.

SQL Mapper

•SQL ←mapping→ Object 필드

•SQL 문으로 직접 데이터베이스를 조작

•Mybatis, jdbcTemplate

ORM(Object-Relation Mapping/객체-관계 매핑)

•DB 데이터 ←mapping→ Object 필드

•객체를 통해 간접적으로 디비 데이터를 다룬다.

•객체와 디비의 데이터를 자동으로 매핑해줌.

•SQL 쿼리가 아니라 **메서드로 데이터를 조작**.

•객체간 관계를 바탕으로 **sql을 자동으로 생성**

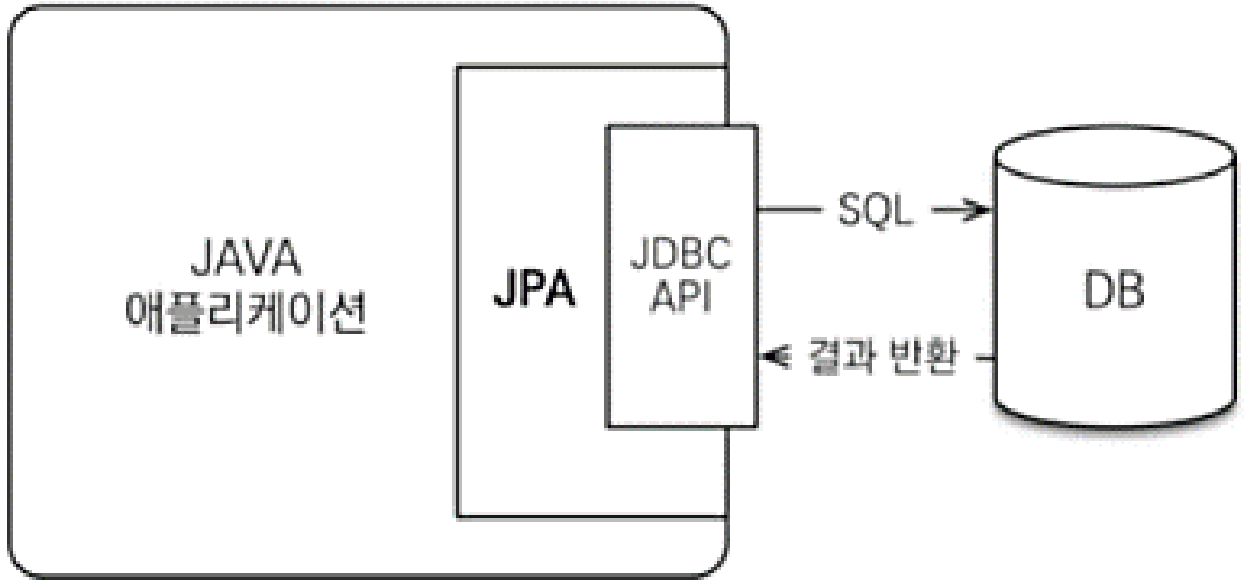
•Persistant API라고 할 수 있다.

•JPA, Hibernate

1. Spring Data JPA

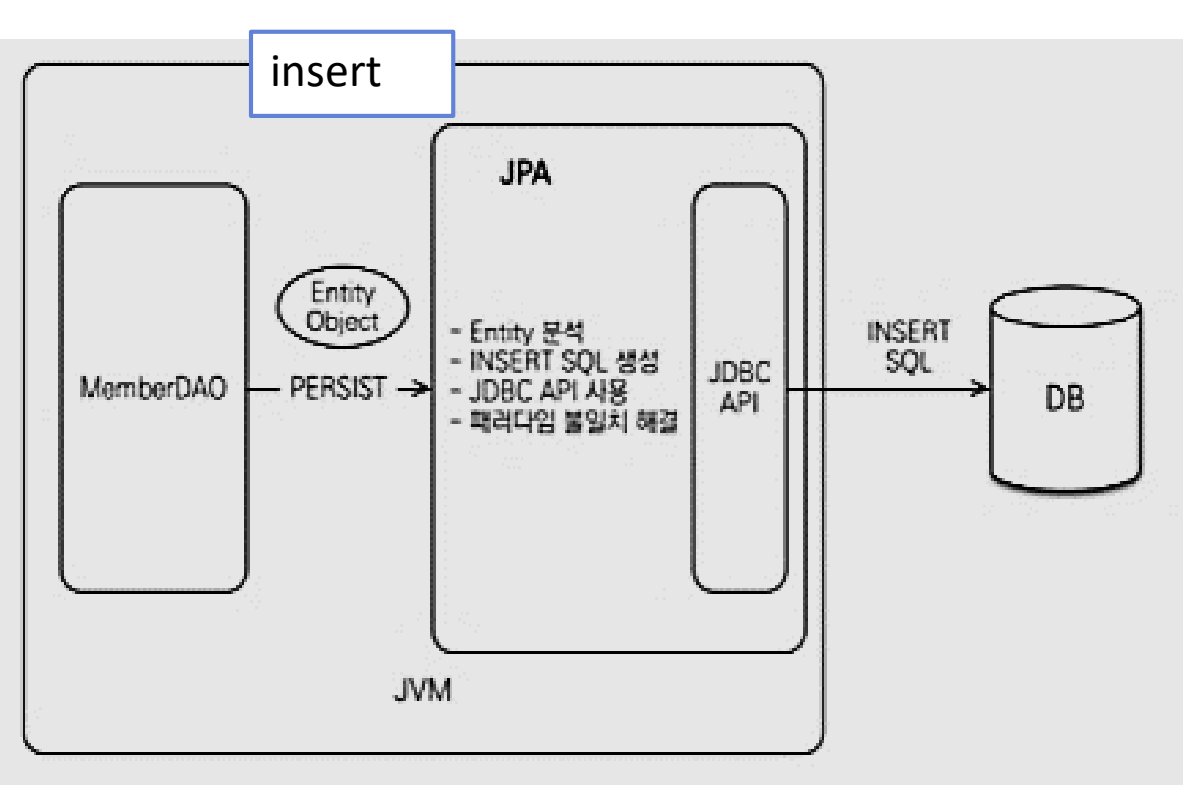
❖ spring-data-jpa

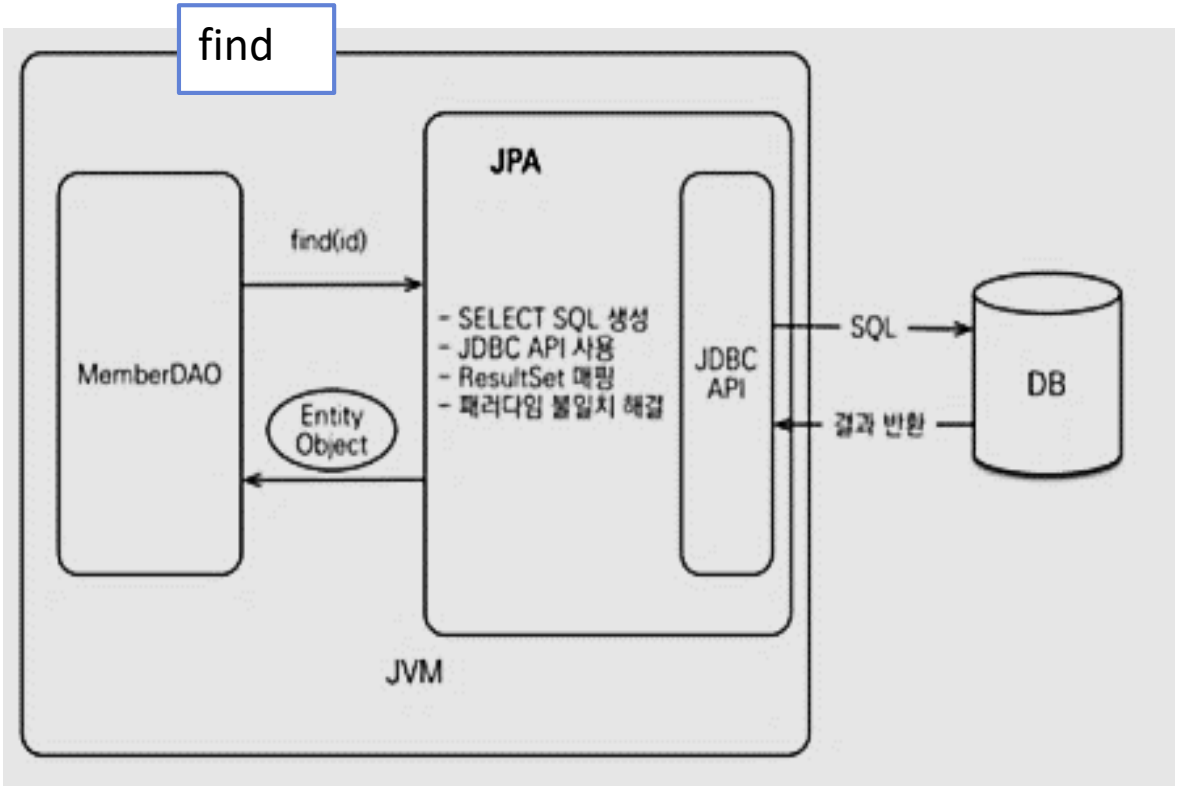
- JPA는 ORM을 위한 자바 EE 표준이며 Spring-Data-JPA는 JPA를 쉽게 사용하기 위해 스프링에서 제공하는 프레임워크
- 추상화 정도 : Spring-Data-JPA -> Hibernate -> JPA
- Spring Data JPA 장점



1. Spring Data JPA

❖ JPA 동작 과정

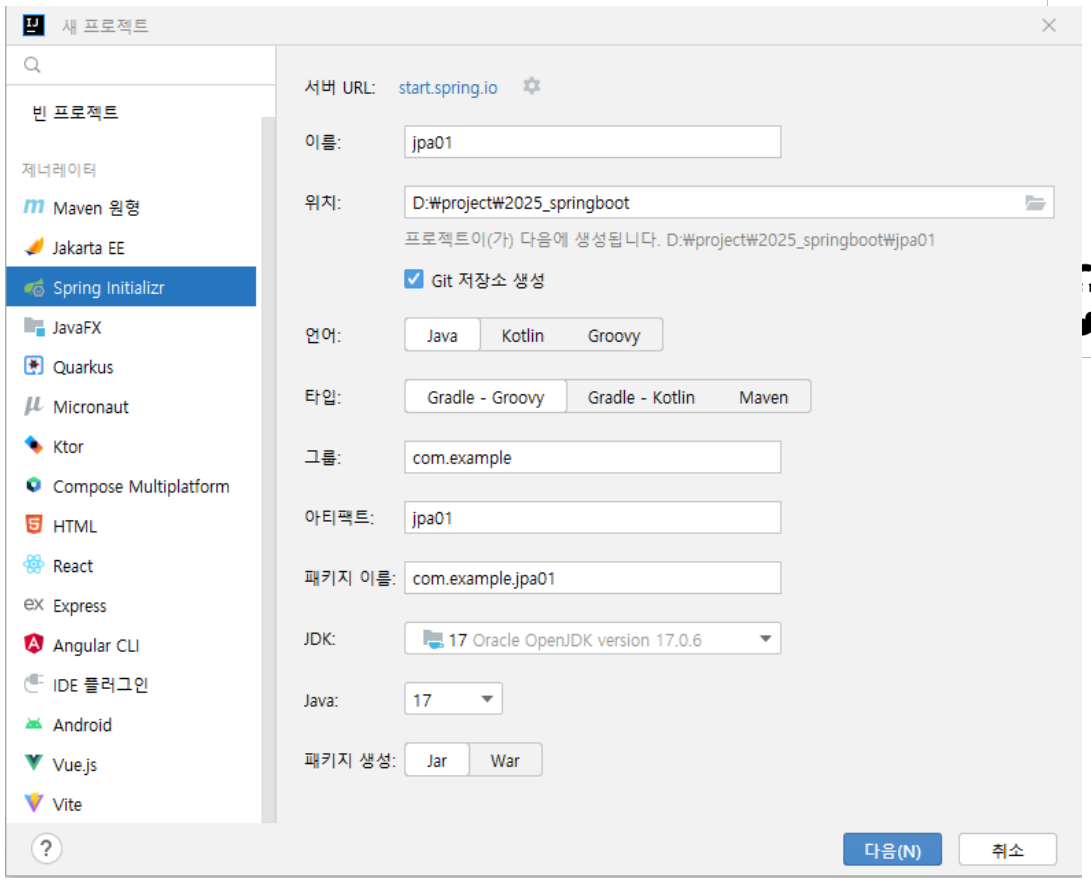


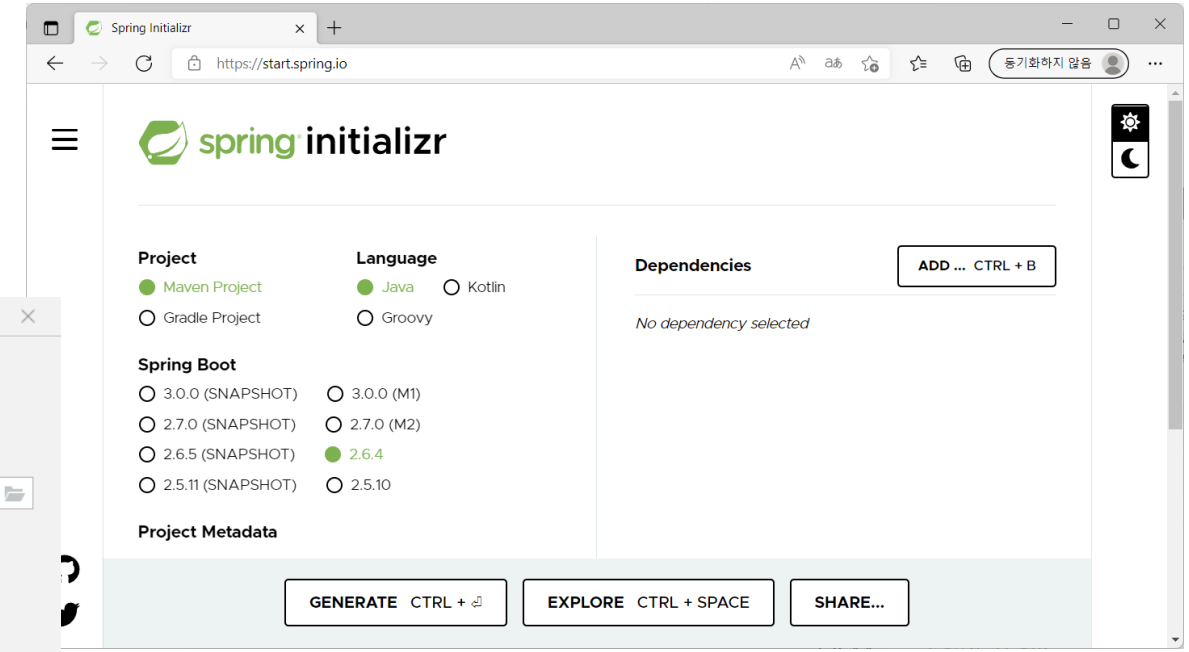


2. JPA 프로젝트 생성

❖ Spring boot

- 개발도구 지원
  - eclipse, Intellij, VSCode
  - Intellij 예:

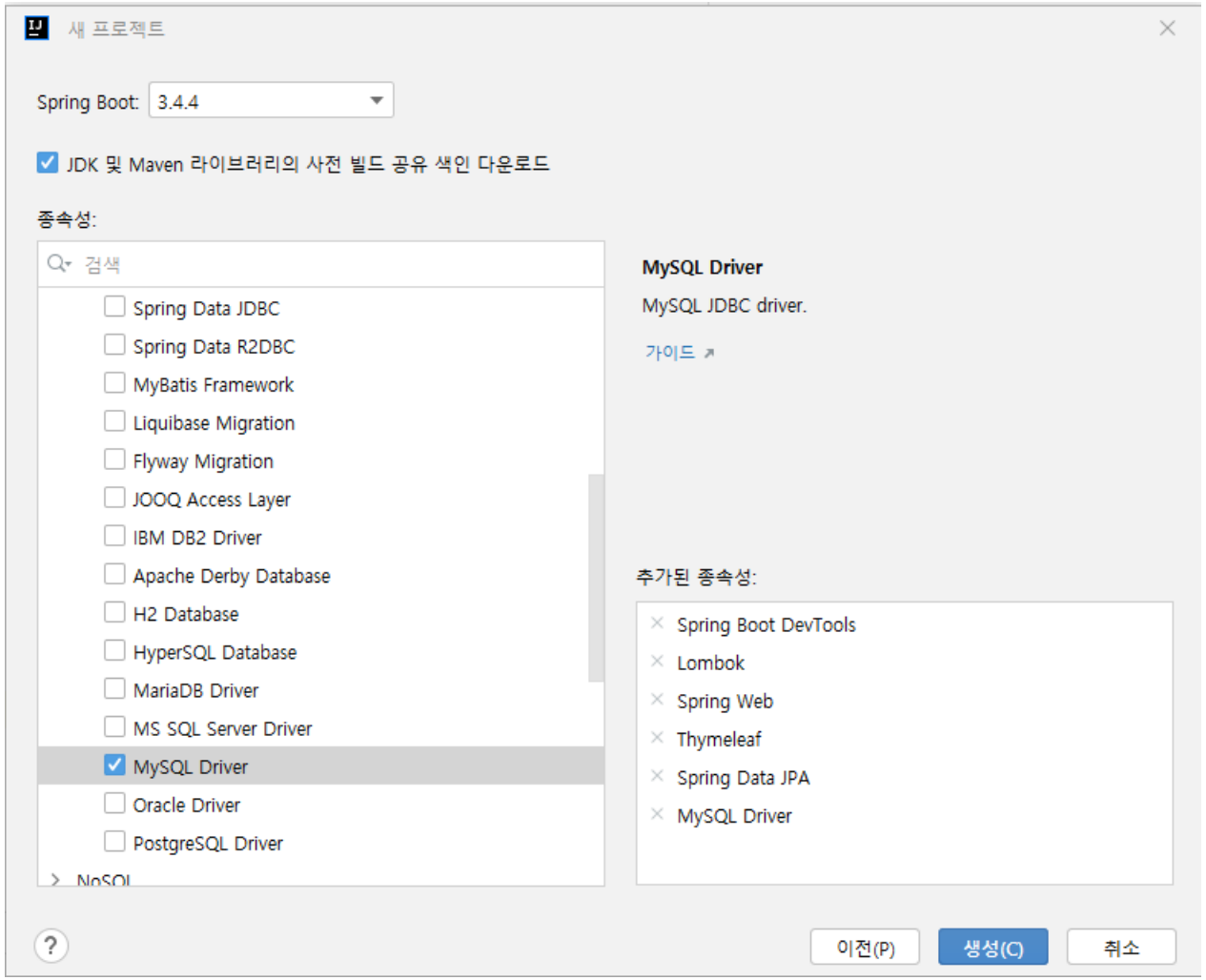




2. JPA 프로젝트 생성

❖ 프로젝트 생성시 추가하는 의존성 라이브러리

- Spring Boot DevTools
- Lombok
- Spring Web
- Thymeleaf
- Spring Data JPA
- MysqlDB Driver



2. JPA 프로젝트 생성

❖ DataSource 설정

- 자동 설정기능으로 인해 의존성 라이브러리를 추가한 것 만으로 자동으로 관련 설정을 사용하게 됨
- HikariCP를 기본으로 사용
- Spring Data JPA에서 사용할 데이터베이스 관련 설정 필요
- 설정 파일 : application.properties 혹은 application.yml 형식

```
***************************
APPLICATION FAILED TO START
***************************

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.

Reason: Failed to determine a suitable driver class
```

2. JPA 프로젝트 생성

❖ 속성 설정: application.properties

```
server.port=8081

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/sbdb
spring.datasource.username=pgm
spring.datasource.password=1234

#실행되는 쿼리 콘솔 출력
spring.jpa.show-sql=true

#콘솔창에 출력되는 쿼리를 가독성이 좋게 포맷팅
spring.jpa.properties.hibernate.format_sql=true
#쿼리에 물음표로 출력되는 바인드 파라미터 출력
logging.level.org.hibernate.type.descriptor.sql=trace

spring.jpa.hibernate.ddl-auto=update
#어떤 데이터베이스 방언(Dialect)**을 사용할지 설정
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
```

- **ddl-auto 옵션 종류**
  - ✓ create: 기존테이블 삭제 후 다시 생성 (DROP + CREATE)
  - ✓ create-drop: create와 같으나 종료시점에 테이블 DROP
  - ✓ update: 변경분만 반영(운영DB에서는 사용하면 안됨)
  - ✓ validate : 엔티티와 테이블이 정상 매핑되었는지 확인
  - ✓ none : 사용하지 않음(사실상 없는 값이지만 관례상 none이라고 함)

- **주의할 점**
  - ✓ 운영 장비에서는 절대 crate, create-drop, update 사용하면 안된다.
  - ✓ 개발 초기 단계는 create 또는 update
  - ✓ 테스트 서버는 update 또는 validate
  - ✓ 스테이징과 운영 서버는 validate 또는 none

2. JPA 프로젝트 생성 :

❖ Spring Data JPA를 위한 설정

- spring.jpa.hibernate.ddl-auto=update // ddl 처리방법

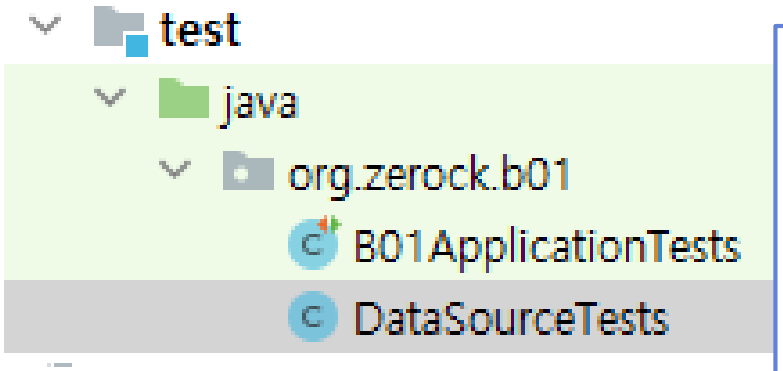
속성값	의미
none	DDL을 하지 않음
create-drop	실행할때 DDL을 실행하고 종료시에 만들어진 테이블등을 모두 삭제
create	실행할때마다 새롭게 테이블등을 생성
update	기존과 다르게 변경된 부분이 있을때는 새로 생성
validate	변경된 부분만 알려주고 종료

2. JPA 프로젝트 생성 : 전체설정

❖ application.yml

```
server:
  port: 8083
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/springdb
    username: pgm
    password: 1234
  devtools:
    livereload:
      enabled: true    # 코드수정 실기간 반영
  jpa:
    hibernate:
      ddl-auto: update   #ddl-auto 옵션
    show-sql: true    #실행되는 쿼리 콘솔 출력
```

2. JPA 프로젝트 생성 : 테스트 환경과 의존성 주입

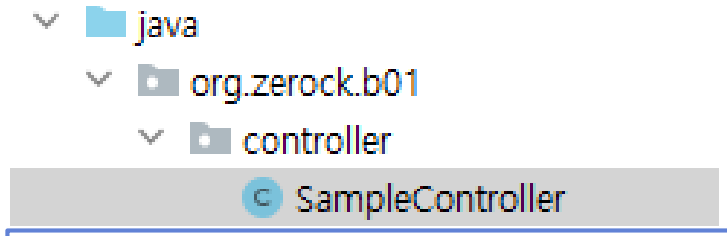



```java
@SpringBootTest
@Log4j2
public class DataSourceTest {
    @Autowired
    private DataSource dataSource;
    @Test
    public void testConnenction() throws SQLException{
        Connection conn= dataSource.getConnection();
        log.info(conn);
    }
}
```

```
JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed
org.zerock.b01.DataSourceTests           : Started DataSourceTests in 2.249 seconds (JVM running for 3.271)
org.zerock.b01.DataSourceTests           : HikariProxyConnection@2086483651 wrapping org.mariadb.jdbc.MariaDbConnection@31de27c
j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

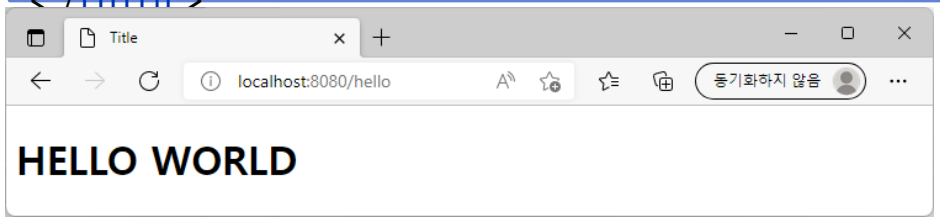
3. 스프링부트에서의 웹 개발

- web.xml이나 servlet-context.xml이 없는 환경에서 개발
- 설정을 위한 @Configuration이나 상속등을 사용
- 스프링부트는 기본적으로 JSP를 지원하지 않음

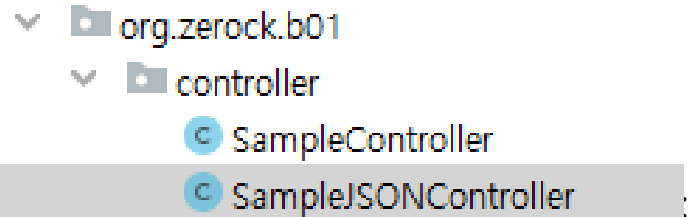



```
package org.zerock.b01.controller;
...
@Controller
@Log4j2
public class SampleController {

    @GetMapping("/hello")
    public void hello(Model model) {
        log.info("hello.....");
        model.addAttribute("msg", "HELLO WORLD");
    }
}
```


```
<!DOCTYPE html>
<html
xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
  <h1 th:text="${msg}"></h1>
</body>
</html>
```

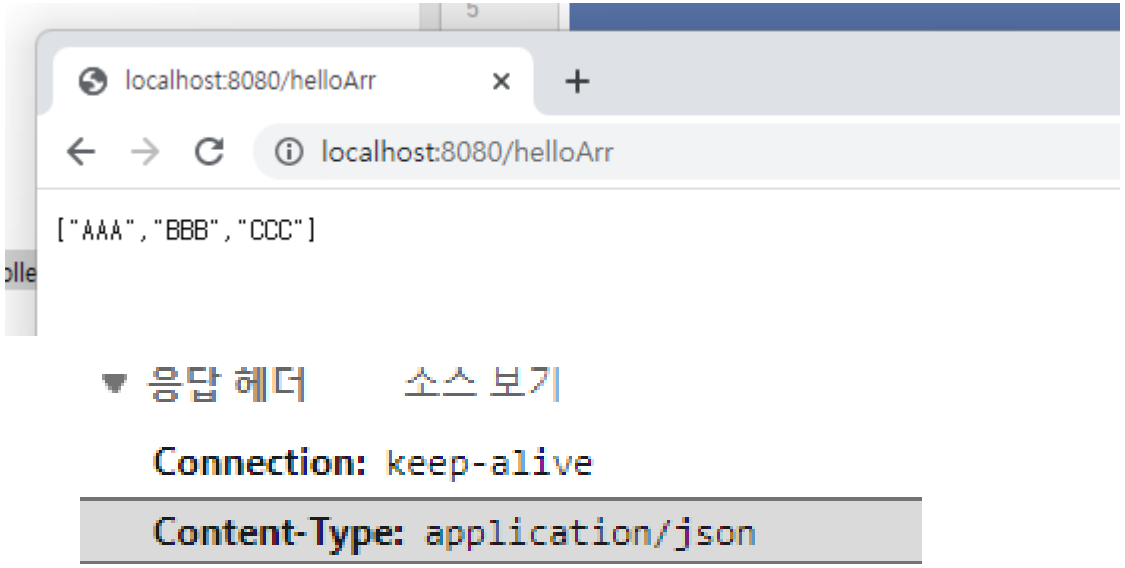


3. 스프링부트에서의 웹 개발 : JSON 데이터 만들기




```
import lombok.extern.log4j.Log4j2;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RestController;

@RestController
@Log4j2
public class SampleJSONController {
    @GetMapping("/helloArr")
    public String[] helloArr() {
        log.info("helloArr.....");
        return new String[]{"AAA", "BBB", "CCC"};
    }
}
```

별도의 라이브러리 없이 스프링부트는 기본적으로 json 처리를 지원



4. Entity

❖ Entity란

- Spring boot에서 데이터베이스 테이블과 매핑되는 클래스
- JPA(Java Persistence API)와 함께 사용되며, 객체 지향적으로 데이터를 다룰 수 있게 도와 줌.

- 데이터베이스의 테이블을 자바 클래스 형태로 표현한 것.
- JPA를 통해 이 클래스는 DB와 자동으로 매핑
- @Entity 어노테이션을 붙여서 정의
- @Id 필드를 정의하여 함.

4. Entity

❖ 엔티티 설계

```
@Data
@Entity
public class Member {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
        private String name;
        private String password;
        private String email;
        private String memo;
        @Column(name="address")
        private String addr;

}
```

```
@Entity
@Table(name="item")
@Getter @Setter @ToString
public class Item {
@Id
@Column(name="item_id")
@GeneratedValue(strategy=GenerationType.AUTO)
private Long id;      //상품 코드

@Column(nullable=false, length=50)
private String itemNm; //상품명

@Column(name="price", nullable=false)
private int price; //가격

@Column(nullable=false)
private int stockNumber; //재고수량

@Lob
@Column(nullable=false)
private String itemDetail; //상품 상세 설명

@Enumerated(EnumType.STRING)
private ItemSellStatus itemSellStatus;//상품 판매 상태

private LocalDateTime regTime;

private LocalDateTime updateTime;
```

4. Entity

❖ 엔티티 매핑관련 어노테이션

어노테이션	설명
@Entity	데이터베이스 테이블과 1:1로 매칭되는 객체, Entity 객체의 인스턴스 하나가 테이블에서 하나의 레코드 값을 의미
@Table	엔티티와 매핑할 테이블 이름 명시적으로 설정, 설정하지 않으면 table명 자동설정
@Id	테이블의 기본 키에 사용할 속성 지정
@GeneratedValue	키 값 생성 전략 명시
@Column	필드와 컬럼 매핑 및 필드 속성 지정(필드명, not null, length 등)
@Lob	BLOB, CLOB 타입 매핑
@CreationTimestamp	데이터 insert 시 시간 자동 저장
@UpdateTimestamp	데이터 update 시 시간 자동 저장
@Enumerated	Enum 타입 매핑
@Transient	해당 필드 데이터베이스 매핑 무시
@Temporal	날짜 타입 매핑
@CreateDate	엔티티가 생성되어 저장될 때 시간 자동 저장
@LastModifedDate	조회한 엔티티의 값을 변경할 때 시간 자동 저장
@DateTimeFormat(pattern = "yyyy-MM-dd'T'HH:mm:ss")	날짜 시간 포맷 설정

4. Entity

❖ @Column 어노테이션의 속성

속성	설명	기본값
name	필드와 매핑할 컬럼 이름 설정	객체이름 필드
unique(DDL)	유니크 제약조건 설정	
insertable	insert 가능 여부	true
updatable	update 가능 여부	true
length	String 타입의 문자 길이 제약조건 설정	255
nullable(DDL)	null 값의 허용 여부 설정, false 설정 시 DDL 생성시에 not null 제약조건 설정	
columnDefinition	데이터베이스 컬럼 정보 기술 예) @Column(columnDefinition="varchar(5) default '10' not null")	
precision, scale(DDL)	BigDecimal 타입에서 사용(BigInteger 가능) precisio은 소수점을 포함한 전체 자리수이고 , scale은 소수점 자리수, Double과 float 타입에 적용되지 않음	

4. Entity

❖ @GeneratedValue 어노테이션 4가지 전략

생성전략	설명
GenerationType.AUTO(default)	JPA 구현체가 자동으로 생성 전략 결정
GenerationType.IDENTIFY	기본키 생성을 데이터베이스 위임 MySql 데이터베이스의 경우 AUTO_INCREMENT 를 사용하여 기본 키 생성 ex) @GeneratedValue(strategy=GenerationType.IDENTITY) private Long id;
GenerationType.SEQUECE	데이터베이스 시퀀스 오브젝트를 사용하여 기본 키 생성 @SequenceGenerator를 사용하여 시퀀스를 등록이 필요 ex) @SequenceGenerator(name='seq", sequencename="jpa_sequence") @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="seq") private Long id;
GenerationType.TABLE	키 생성용 테이블 사용 @TableGenerator 필요

5. Repository

❖ Repository 설계

```
public interface MemberRepository extends JpaRepository<Member, Long>{

}
```

- 쿼리 메소드 : 메소드 이름으로 쿼리 생성
- 스프링 데이터 JPA가 제공하는 쿼리 메소드 기능
  - 메소드 이름으로 쿼리 생성 : 메소드 이름을 분석해서 JPQL 쿼리 실행하는 것
  - 메소드 이름으로 JPA NamedQuery 호출
- @Query 어노테이션을 이용한 쿼리 직접 정의
- Spring Data JPA Querydsl

5. Repository

❖ 쿼리메소드 세부 기능

- 조회: find…By ,read…By ,query…By get…By : <T> 타입 반환
- COUNT: count…By : long 타입 반환
- EXISTS: exists…By : Boolean 타입 반환
- 삭제: delete…By, remove…By : long 타입 반환
- DISTINCT: findDistinct, findMemberDistinctBy
- LIMIT: findFirst3, findFirst, findTop, findTop3

5. Repository

❖ 쿼리 메소드

- Repository에서 지원하는 기본 메소드

메소드	기능
save(S entity)	엔티티 저장 및 수정
delete(T entity)	엔티티 삭제
count()	엔티티 총 개수 반환
findAll()	모든 엔티티 조회
findOne()	primary key로 한건의 레코드 찾기

- findBy, countBy Entity필드명 메소드

메소드	기능
findBy_____	쿼리를 요청하는 메서드 임을 알림
countBy_____	쿼리 결과 레코드 수를 요청하는 메서드임을 알림

5. Repository

❖ Query 메소드에 **포함할 수 있는 키워드**

메서드 이름 키워드	샘플	설명
And	findByEmailAndUserId(String email, String userId)	여러필드를 and 로 검색
Or	findByEmailOrUserId(String email, String userId)	여러필드를 or 로 검색
Between	findByCreatedAtBetween(Date fromDate, Date toDate)	필드의 두 값 사이에 있는 항목 검색
LessThan	findByAgeGraterThanEqual(int age)	작은 항목 검색
GreaterThanEqual	findByAgeGraterThanEqual(int age)	크거나 같은 항목 검색
Like	findByNameLike(String name)	like 검색
IsNull	findByJobIsNull()	null 인 항목 검색
In	findByJob(String … jobs)	여러 값중에 하나인 항목 검색
OrderBy	findByEmailOrderByNameAsc(String email)	검색 결과를 정렬하여 전달

5. Repository

❖ Pageable

- Query 메소드의 입력변수로 **Pageable 변수를 추가하면 Page타입을 반환형으로 사용**
- Pageable 객체를 통해 **페이징과 정렬을 위한 파라미터를 전달**

```
@RestController
@RequestMapping("/member")
public class MemberController {

   @Autowired
   MemberService memberService;

   @RequestMapping("")
   Page<Member> getMembers(Pageable pageable){
      return memberService.getList(pageable)
   }
}
```

query parameter 명	설명
page	몇번째 페이지 인지를 전달
size	한 페이지에 몇개의 항목을 보여줄것인지 전달
sort	정렬정보를 전달. 정렬정보는 필드이름,정렬방향 의 포맷으로 전달한다. 여러 필드로 순차적으로 정렬도 가능하다. 예: sort=createdAt,desc&sort=userId,asc

GET /users?page=1&size=10&sort=createdAt,desc&sort=userId,asc

5. Repository

❖ Spring DATA JPA @Query 어노테이션 사용

- SQL화 유사한 JPQL(Java Persistance Query Language)라는 객체지향 쿼리 언어 사용

```
public interface BoardRepository extends JpaRepository<Board, Long>{

   Page<Board> findByTitleContainingOrderByBnoDesc(String keyword, Pageable pageable);

   @Query("select  b from Board b where b.title like concat('%', :keyword,'%') order by b.bno desc ")
   Page<Board> findByKeyword(String keyword,Pageable pageable);

}
```

5. Repository

❖ Spring Data JPA Querydsl

- 동적쿼리 생성
- 쿼리 재사용할 수 있어 제약조건 조립 및 가독성 향상
- 문자열이 아닌 자바 소스코드로 작성하기 때문에 컴파일 시점 오류 발견
- IDE 도움을 받아서 자동 완성 기능 사용가능

5. Repository

❖ Spring Data JPA Querydsl : 환경설정

- build.gradle 맨위 상단에 오른쪽 내용 추가
- 종속성 추가

```
buildscript {
    ext {
        queryDslVersion = "5.0.0"
    }
}
```

```
implementation "com.querydsl:querydsl-jpa:${queryDslVersion}:jakarta"
annotationProcessor(
        "jakarta.persistence:jakarta.persistence-api",
        "jakarta.annotation:jakarta.annotation-api",
        "com.querydsl:querydsl-apt:${queryDslVersion}:jakarta")
```

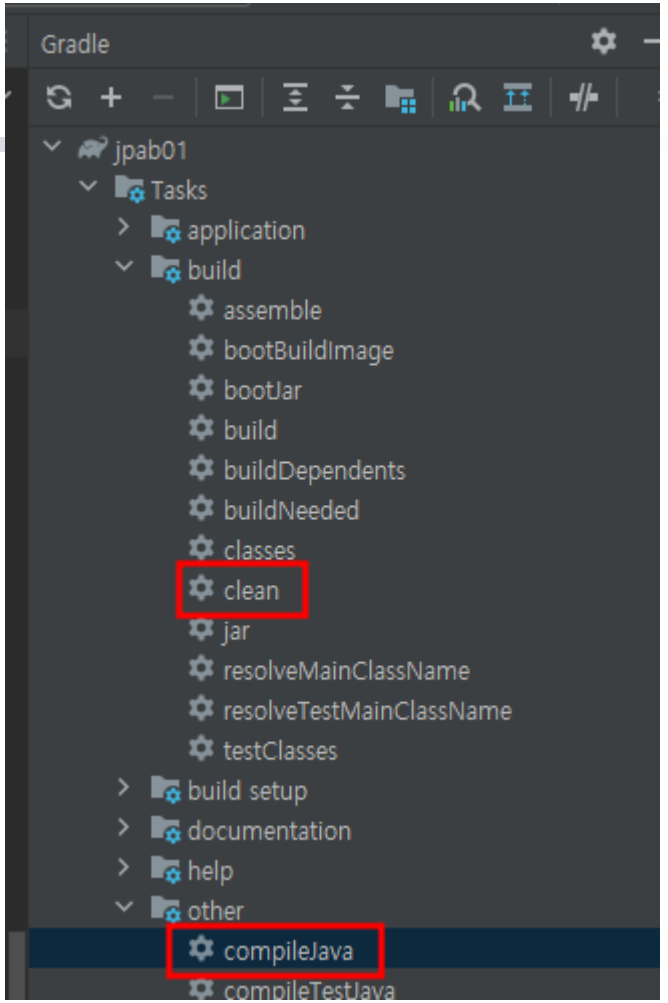
- 끝부분에 다음 내용 추가

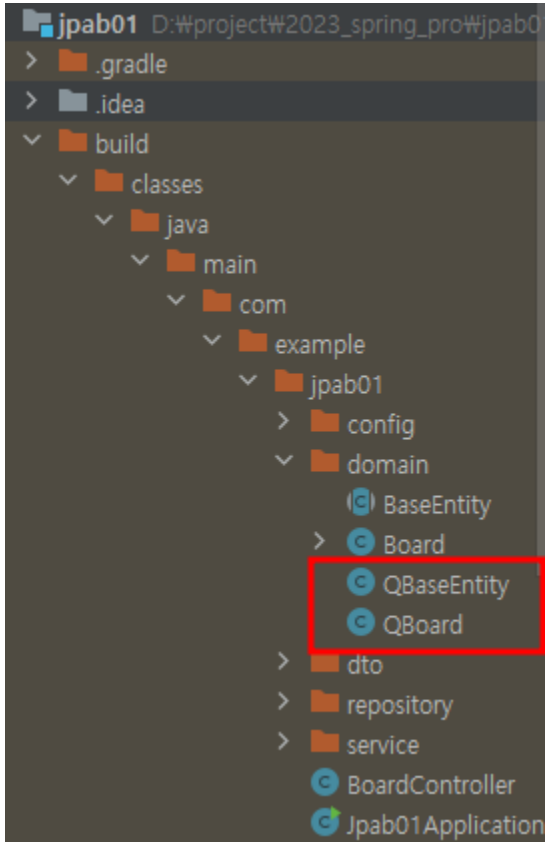
```
sourceSets {

    main {
        java {
            srcDirs = ["$projectDir/src/main/java", "$projectDir/build/generated"]
        }
    }
}

compileJava.dependsOn('clean')
```

5. Repository

❖ Spring Data JPA Querydsl : querydsl 설정 확인

- Gradle 메뉴 -> Tasks -> bulid -> clean 실행 후 ,
- Gradle 메뉴 -> Tasks -> other -> compilejava 실행
- 프로젝트 탐색기 창 -> build -> class 아래 QBoard, QBaseEnt





5. Repository

❖ Spring Data JPA Queryds

- 기존의 Repository와 Query

```
package com.example.jpab01.repository.search;

…
public interface BoardSearch {
    Page<Board> search1(Pageable pageable);

    Page<Board> searchAll(String[] types, String keyword, Pageable pageable);
}
```

```
package com.example.jpab01.repository.search;

…
public class BoardSearchImpl extends QuerydslRepositorySupport implements BoardSearch {
    public BoardSearchImpl() {
        super(Board.class);
    }
    …
}
```

```
public interface BoardRepository extends JpaRepository<Board, Long>, BoardSearch {

    …
}
```

5. Repository

```
public class BoardSearchImpl extends QuerydslRepositorySupport implements BoardSearch {
    public BoardSearchImpl() {
        super(Board.class);
    }

    public Page<Board> search1(Pageable pageable) {
        QBoard board = QBoard.board;

        JPQLQuery<Board> query = this.from(board);
        BooleanBuilder booleanBuilder = new BooleanBuilder();

        booleanBuilder.or(board.title.contains("1"));
        booleanBuilder.or(board.content.contains("1"));

        query.where(new Predicate[]{booleanBuilder});
        query.where(new Predicate[]{board.bno.gt(0L)});

        this.getQuerydsl().applyPagination(pageable, query);

        List<Board> list = query.fetch();
        long count = query.fetchCount();
        return new PageImpl(list, pageable, count);
    }
```

```java
@Override
public Page<Board> searchAll(String[] types, String keyword, Pageable pageable) {
    QBoard board = QBoard.board;
    JPQLQuery<Board> query = this.from(board);

    if( (types != null && types.length > 0) && keyword != null ){ //검색 조건과 키워드가 있다면
        BooleanBuilder booleanBuilder = new BooleanBuilder(); // (
        for(String type: types){
            switch (type){
                case "t":
                    booleanBuilder.or(board.title.contains(keyword));
                    break;
                case "c":
                    booleanBuilder.or(board.content.contains(keyword));
                    break;
                case "w":
                    booleanBuilder.or(board.writer.contains(keyword));
                    break;
            }
        }//end for
        query.where(booleanBuilder);
    }//end if
    //bno > 0
    query.where(board.bno.gt(0L));
    //paging
    this.getQuerydsl().applyPagination(pageable, query);
    List<Board> list = query.fetch();
    long count = query.fetchCount();
    return new PageImpl<>(list, pageable, count);
}
```