

# Apple Market

중고거래 사이트

## — 1.프로젝트 기획

- 프로젝트 기획 & 기능
- 개발 환경

## — 2.프로젝트 환경설정

- pom.xml
- Application
- Security

## — 3.DB구조

- EER Diagram 구조
- DB Table 설명

## — 4.프로젝트 구성

- MVC모델 구성

## — 5.코드 리뷰

- 연관관계 매핑
- 카테고리

## — 6.마무리

- 프로젝트 시연
- 업데이트 예정 사항

# 1. 프로젝트 기획

Apple Market

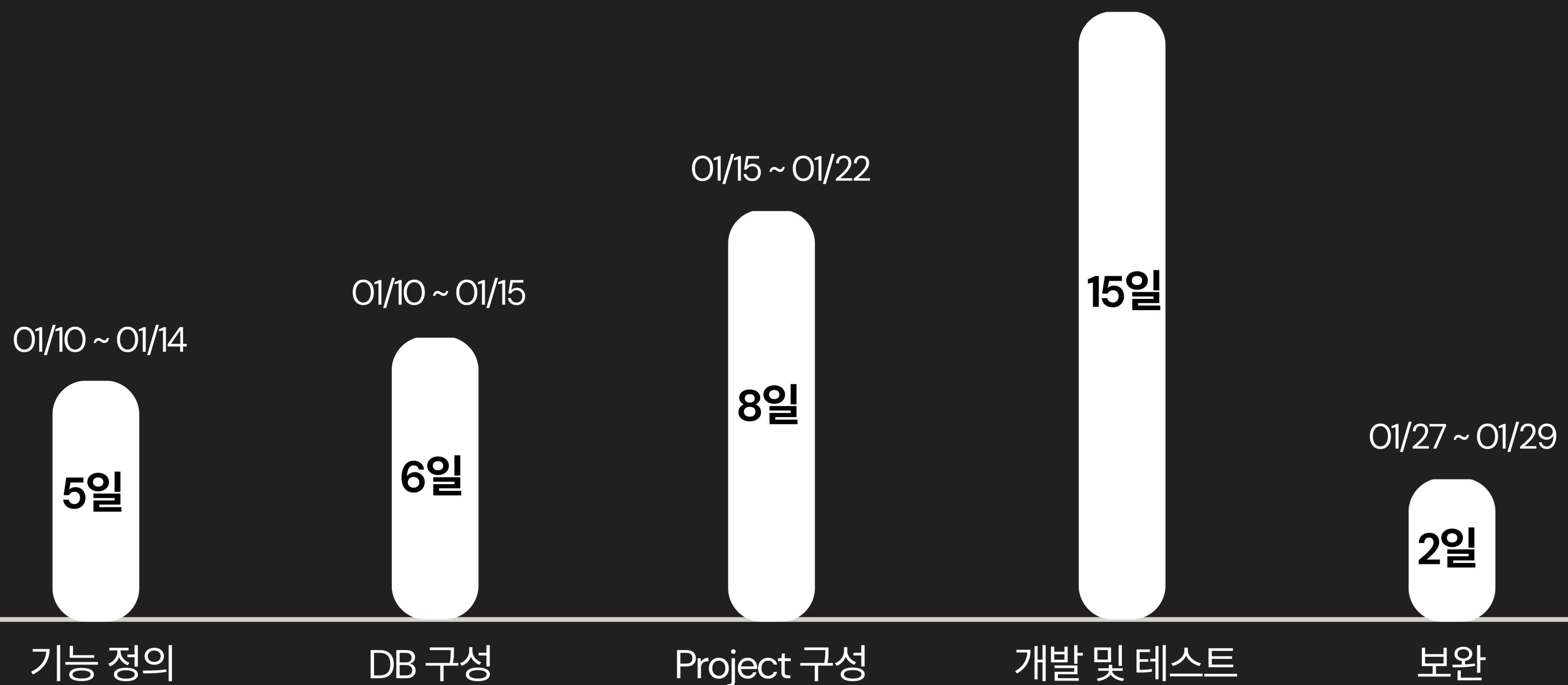
페이지 구성 (MVC 구성)

주요 기능 (카테고리 설정 및 사용자 주소 맵핑)

DB 구성 ( 테이블 및 컬럼 간의 관계 설정)

제작 및 테스트

# 개발 일정



## 개발 도구

Eclipse IDE 4.25.0  
My SQL Workbench 8.0.13

## 개발 언어

Model/Controller: Java, SpringFramework  
View: JQuery, Css, JavaScript  
DB: SQL, Hibernate  
프로젝트 구성: Maven  
오픈소스: 카카오 주소 API, 주소공공데이터API

## DB Connection

JDBC 드라이버: mysql-connector-java.8.0.17  
Connection Pool: c3p0 0.9.5.2

## 개발 서버

Apache Tomcat 9.0

## 운영체제

Windows 10

## 2. 프로젝트 환경설정

Apple Market

# Pom.Xml(1)

```
http://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation with catalog)
1<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3
4  <modelVersion>4.0.0</modelVersion><!--project 정보-->
5  <groupId>com.code.springdemo</groupId>
6  <artifactId>apple-market</artifactId>
7  <version>1.0.0</version>
8  <packaging>war</packaging>
9
10 <properties> <!--라이브러리 버전관리 (라이브러리의 버전을 따로 관리하고 싶을때 properties로 정의해 따로 관리 할 수 있다.)-->
11   <springframework.version>5.1.9.RELEASE</springframework.version> <!--라이브러리: springframework 5.1.9.RELEASE 버전-->
12   <springsecurity.version>5.0.0.RELEASE</springsecurity.version> <!--라이브러리: springsecurity 5.0.0.RELEASE 버전-->
13   <hibernate.version>5.4.4.Final</hibernate.version> <!--라이브러리: hibernate 5.4.4.Final 버전-->
14   <mysql.connector.version>8.0.17</mysql.connector.version> <!--라이브러리: mysql 8.0.17 버전-->
15   <c3po.version>0.9.5.2</c3po.version> <!--라이브러리: c3po 0.9.5.2 버전-->
16
17   <maven.compiler.source>1.8</maven.compiler.source> <!--라이브러리: maven JDK 1.8 버전-->
18   <maven.compiler.target>1.8</maven.compiler.target>
19 </properties>
20
21 <dependencies> <!--라이브러리 셋팅 사용할 라이브러리를 정의한다.-->
22   <dependency> <!-- -Spring MVC support (spring mvc에 사용되는 라이브러리) -->
23     <groupId>org.springframework</groupId> <!--Servlet API를 기반으로 구축된 독창적인 웹 프레임워크이며 처음부터 Spring Framework에 포함되어있다. -->
24     <artifactId>spring-webmvc</artifactId>
25     <version>${springframework.version}</version>
26   </dependency>
27   <dependency>
28     <groupId>org.springframework</groupId>
29     <artifactId>spring-tx</artifactId>
30     <version>${springframework.version}</version>
31   </dependency>
32   <dependency>
33     <groupId>org.springframework</groupId>
34     <artifactId>spring-orm</artifactId>
35     <version>${springframework.version}</version>
36   </dependency>
37
38   <dependency> <!--스프링 시큐리티는 사용자를 인증하고, 로그인 후 애플리케이션의 각 기능들에 대한 권한을 부여하는 기능을 구현하는데 사용되는 프레임워크로, 각 핸들러, 필터들을 거쳐 동작한다는 것이 포인트입니다. -->
39     <groupId>org.springframework.security</groupId>
40     <artifactId>spring-security-web</artifactId>
41     <version>${springsecurity.version}</version>
42   </dependency>
43
44   <dependency><!--스프링 시큐리티를 config하기 위한 라이브러리-->
45     <groupId>org.springframework.security</groupId>
46     <artifactId>spring-security-config</artifactId>
47     <version>${springsecurity.version}</version>
48   </dependency>
49
```

# Pom.Xml(2)

```
<dependency><!--:스프링 시큐리티를 taglibs를 사용하기 위한 라이브러리-->
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>${springsecurity.version}</version>
</dependency>

<dependency><!--하이버네이트 라이브러리-->
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>${hibernate.version}</version>
</dependency>

<dependency><!--JDBC드라이버:자바 프로그램에서 서로 다른 데이터베이스에 표준화된 방법으로 접근할 수 있는 방법을 하는 API-->
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql.connector.version}</version>
</dependency>

<dependency><!--c3p0 Connection과 StatementPool을 제공하는 라이브러리-->
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>${c3po.version}</version>
</dependency>

<dependency><!--Servlet, JSP and JSTL support (서블릿과 jsp와 jstl에 사용되는 라이브러리) -->
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>

<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.1</version>
</dependency>

<dependency><!--if문, 출력문, 변수선언 등의 기능들을 JSP에서 사용할 수 있도록 표준으로 모아둔 라이브러리-->
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```

# Pom.Xml(3)

```
<dependency><!--JAXB (Java Architecture for XML Binding)는 Java Object를 XML로 직렬화하고,  
XML을 Java Object로 역직렬화해주는 자바 API 입니다. JDK6 ~ 9 버전은 JAXB가 내장되어 있어 라이브러리를 추가 할 필요가 없습니다.-->  
    <groupId>javax.xml.bind</groupId>  
    <artifactId>jaxb-api</artifactId>  
    <version>2.3.0</version>  
</dependency>  
  
<dependency><!--파일 업로드에 필요한 라이브러리 . . .-->  
    <groupId>commons-fileupload</groupId>  
    <artifactId>commons-fileupload</artifactId>  
    <version>1.4</version>  
</dependency>  
  
</dependencies>  
  
<build><!--배포를 위한 빌드 -->  
    <finalName>apple-market</finalName>  
    <plugins>  
        <plugin>  
            <groupId>org.apache.maven.plugins</groupId>  
            <artifactId>maven-war-plugin</artifactId>  
            <version>3.2.0</version>  
        </plugin>  
    </plugins>  
</build>  
<name>apple-market</name>  
</project>
```

# ApplicationConfig(1)

```
public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
    /*AbstractAnootationConfigDispatcherServletInitializer는
    DispatcherServlet과 ContextLoaderListner를 생성한다
    ->Web.xml의 대안으로 아파치 톰캣7, 서블릿 3.0이상에서만 사용가능
    ※DispatcherServlet : 컨트롤러, 뷰리졸버, 핸들러 매핑 등 웹 컴포넌트가 포함된 빈을 로딩하는데 사용되는걸로 추측됨
    ※ContextLoaderListnet:애플리케이션 내의 빈을 로딩할 것으로 추측됨 (중간계층, 데이터 계층 컴포넌트) */
    @Override
    protected Class<?>[] getRootConfigClasses() {
        /*프로젝트에서 사용할 Bean들을 정의하기 위한 클래스를 지정
        →리턴된 @Configuration 클래스는 ContextLoaderListner*/
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { DemoAppConfig.class };
        //Spring Mvc 프로젝트 설정을 위한 클래스를 지정한다.
        //→리턴된 @Configuration 클래스는 DispatcherServlet에 매핑
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
        /*DispatcherServlet에 매핑할 요청 주소를 셋팅한다.
        DispatcherServlet을 '/'은 애플리케이션으로 오는 모든 요청을 처리한다.
        매핑 되기 위한, 하나 혹은 여러개의 패스를 지정할 수 있다.*/
    }
}
```

# Properties

```
1 #
2 # JDBC connection properties
3 #
4 jdbc.driver=com.mysql.cj.jdbc.Driver
5 jdbc.url=jdbc:mysql://localhost:3306/applemarket?useSSL=false&serverTimezone=UTC
6 jdbc.user=springstudent
7 jdbc.password=springstudent
8
9 #
10 # Connection pool properties
11 #
12 connection.pool.initialPoolSize=5
13 connection.pool.minPoolSize=5
14 connection.pool.maxPoolSize=20
15 connection.pool.maxIdleTime=3000
16
17 #
18 # Hibernate properties
19 #
20 hibernate.dialect=org.hibernate.dialect.MySQLDialect
21 hibernate.show_sql=true
22 hibernate.packagesToScan=com.code.springdemo.entity
```

# ApplicationConfig(2)

```
@Configuration //설정파일을 만들기 위한 애노테이션 or Bean을 등록하기 위한 애노테이션이다. Bean을 등록할때 싱글톤(singleton)이 되도록 보장해준다. 스프링컨테이너에서 Bean을 관리할수있게 됨.
@EnableWebMvc
/*
 * @Controller 애노테이션을 붙인 컨트롤러를 위한 설정을 생성합니다. 추가로 @EnableWebMvc 애노테이션을 사용하면
 * WebMvcConfigurer 타입의 빈을 이용해 MVC 설정을 추가로 생성합니다.
 */
@EnableTransactionManagement
@ComponentScan("com.code.springdemo")
/*
 * @Component 애너테이션 및 stereotype{@Service,@Repository, @Controller @Entity}
 * 애너테이션에 부여된 Class들을 자동으로 scan하여 Bean으로 등록해주는 역할을 하는 애너테이션입니다. 이전 xml파일에
 * <context:component-scan base-package="패키지 경로" />을 이용해 지정해주었던 것을 java파일을 이용하여
 * bean을 scan하기 위해서 생겨났습니다.
 */
@PropertySource({ "classpath:persistence-mysql.properties" })
/*
 * @PropertySource("classpath:properties경로") :정적으로 하드코딩된 내용을 쓸 때 주로 사용한다(Ex. DB
 * 설정정보) PropertySource를 Spring에 추가하기 위해서는 Environment클래스와 @Configuration과 함께 사용
 * 된다.
 */
//WebMvcConfigurer를 사용하면 @EnableWebMvc가 자동적으로 세팅해주는 설정에 개발자가 원하는 설정을 추가할 수 있게 된다. 즉 Override가 가능하다.
public class DemoAppConfig implements WebMvcConfigurer {

    @Autowired
    private Environment env;

    private Logger logger = Logger.getLogger(getClass().getName());

    @Bean
    //JSP를 뷰 기술로 사용할 경우 다음과 같이 InternalResourceViewResolver 구현체를 빈으로 등록해주면 된다(Web.xml형식).
    /*
     * <bean id="viewResolver"
     * class="org.springframework.web.servlet.view.InternalResourceViewResolver">
     * <property name="prefix" value="WEB-INF/view"/>
     * <property name="suffix" value=".jsp"/>
     * </bean> */
    //JSP를 뷰 기술로 사용할 경우 다음과 같이 InternalResourceViewResolver 구현체를 빈으로 등록해주면 된다(Java형식).
```

# ApplicationConfig(3)

```
//JSP를 뷰 기술로 사용할 경우 다음과 같이 InternalResourceViewResolver 구현체를 빈으로 등록해주면 된다(Java형식).
public ViewResolver viewResolver() {

InternalResourceViewResolver viewResolver = new
InternalResourceViewResolver();

viewResolver.setPrefix("/WEB-INF/view/");
viewResolver.setSuffix(".jsp");
// 이는 ViewResolver가 "WEB-INF/view/뷰이름.jsp"를 뷰 JSP로 사용한다는 것을 의미한다.
return viewResolver; }

@Bean
//DB와 관계된 커넥션 정보를 담고있으며 빈으로 등록하여 인자로 넘겨준다.
//주요 기능 DB 서버와의 연결을 해준다. DB Connection pooling기능
public DataSource myDataSource() {

    //properties에 담긴 DB정보를 셋팅
    ComboPooledDataSource myDataSource = new ComboPooledDataSource();

    try {
        myDataSource.setDriverClass(env.getProperty("jdbc.driver"));
    }
    catch (PropertyVetoException exc) {
        throw new RuntimeException(exc);
    }

    logger.info(">>>>jdbc.url=" + env.getProperty("jdbc.url"));
    logger.info(">>>>jdbc.user=" + env.getProperty("jdbc.user"));

    myDataSource.setJdbcUrl(env.getProperty("jdbc.url"));
    myDataSource.setUser(env.getProperty("jdbc.user"));
    myDataSource.setPassword(env.getProperty("jdbc.password"));}
```

# ApplicationConfig(4)

```
//Hibernate 설정정보를 추가
private Properties getHibernateProperties() {
    Properties props = new Properties();
    //JPA의 주요 특징인 "데이터베이스에 종속적이지 않다"를 쓰기위해 hibernate.dialect 를 쓴다.
    props.setProperty("hibernate.dialect", env.getProperty("hibernate.dialect"));

    //Hibernate에서 실행하는 SQL 문을 출력할지 여부를 지정한다. true시 콘솔에 출력된다. 현재 true 상태
    props.setProperty("hibernate.show_sql", env.getProperty("hibernate.show_sql"));
    return props;
}

//DB 셋팅 부분 String형을 int형으로 캐스팅 하는 메서드
private int getIntProperty(String propName) {
    String propVal = env.getProperty(propName);

    int intPropVal = Integer.parseInt(propVal);

    return intPropVal;
}

@Bean
//SessionFactory를 생성하는 FactoryBean입니다.
public LocalSessionFactoryBean sessionFactory() {

    LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();

    sessionFactory.setDataSource(myDataSource());
    //Hibernate가 지정된 패키지 아래에 있는 ORM annotation이 붙은 도메인 객체를 스캔한다.
    sessionFactory.setPackagesToScan(env.getProperty("hibernate.packagesToScan"));
    sessionFactory.setHibernateProperties(getHibernateProperties());

    return sessionFactory;
}
```

## ApplicationConfig(5)

```
@Bean  
@Autowired  
  
// 하이버네이트 DAO에는 HibernateTransactionManager 를 사용한다.  
// SessionFactory 타입의 프로퍼티로 넣어주면 된다.  
public HibernateTransactionManager transactionManager(SessionFactory sessionFactory) {  
  
    HibernateTransactionManager txManager = new HibernateTransactionManager();  
    txManager.setSessionFactory(sessionFactory);  
  
    return txManager;  
}  
  
@Override  
// 해당 위치에 있는 파일을 Resources 파일로 사용한다.  
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
    registry  
        .addResourceHandler("/resources/**")  
        .addResourceLocations("/resources/");  
}  
}
```

# SecurityConfig(1)

```
5 public class SecurityWebApplicationInitializer extends AbstractSecurityWebApplicationInitializer {  
5  
7     //Spring Security 를 웹에서 사용하기 위해서는 Spring Security Filter 를 사용할 수 있게 스프링 프로젝트에 등록해줘야한다.  
3     //따라서 이를 가능하게 해줄, AbstractSecurityWebApplicationInitializer 라는 클래스를 상속 받게 해야한다.  
9 }  
9
```

```
@Configuration  
@EnableWebSecurity  
public class DemoSecurityConfig extends WebSecurityConfigurerAdapter {  
  
    @Autowired  
    private DataSource securityDataSource;  
  
    @Override  
    protected void configure(AuthenticationManagerBuilder auth) throws Exception{  
        final String usernameQuery ="SELECT customer_id, customer_pw, customer_activated as enabled FROM customer WHERE customer_id = ? ";  
        final String authQuery ="SELECT customer_id, customer_role FROM customer WHERE customer_id = ?";  
  
        auth.jdbcAuthentication()  
            .dataSource(securityDataSource)  
            .usersByUsernameQuery(usernameQuery)  
            .authoritiesByUsernameQuery(authQuery)  
            .passwordEncoder(passwordEncoder());  
    }  
  
    @Bean  
    public PasswordEncoder passwordEncoder() {  
        return new BCryptPasswordEncoder();  
    }  
}
```

## SecurityConfig(2)

```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
    http.authorizeRequests()  
        .antMatchers("/css/**").permitAll() // css 적용 경로 지정  
        .antMatchers("/writePost/**").hasRole("CUSTOMER")  
        /* .antMatchers("/priceInquiry/**").hasRole("manager") */  
        .anyRequest().permitAll() // anyRequest().authenticated() 로그인을 한 사용자만 이용할 수 있게 설정  
    .and()  
    .formLogin()  
    .loginPage("/showMyLoginPage") //로그인 페이지 경로 지정  
    .failureUrl("/showMyLoginPage?error=true") //로그인 실패 후 경로지정  
    .permitAll()  
    .and()  
    .logout()  
    .logoutSuccessUrl("/")  
    .permitAll()  
    .and()  
    .exceptionHandling().accessDeniedPage("/acess-denied"); // 접근제한 경로 설정  
  
    // POST로 받아온 데이터 한글 인코딩  
    CharacterEncodingFilter filter = new CharacterEncodingFilter();  
    filter.setEncoding("UTF-8");  
    filter.setForceEncoding(true);  
    http.addFilterBefore(filter,CsrfFilter.class);  
}
```

# 3.DB구조

Apple Market

# DB Table 설명

category	
no	INT(11)
category_name	VARCHAR(30)

sub_category	
no	INT(11)
category_no	INT(11)
sub_category_name	VARCHAR(50)

activity_areas	
no	INT(11)
customer_no	INT(11)
areasinfo_no	INT(11)

announcement	
no	INT(11)
title	VARCHAR(45)
text	TEXT
created	TIMESTAMP

customer	
no	INT(11)
customer_id	VARCHAR(100)
customer_pw	VARCHAR(600)
customer_name	VARCHAR(30)
customer_role	VARCHAR(30)
customer_address	VARCHAR(100)
customer_address_detail	VARCHAR(100)
customer_phone	VARCHAR(30)
customer_activated	TINYINT(1)

areasinfo	
ZIP_NO	INT(11)
SIDO	VARCHAR(20)
SIGUNGU	VARCHAR(20)
DORO	VARCHAR(80)
BUILD_NO1	DECIMAL(5,0)
BUILD_NO2	DECIMAL(5,0)
BUILD_NM	VARCHAR(200)
DONG_NM	VARCHAR(20)
H_DONG_NM	VARCHAR(40)
ZIBUN1	DECIMAL(4,0)
ZIBUN2	DECIMAL(4,0)

review	
no	INT(11)
buyer_no	INT(11)
goods_no	INT(11)
review	TEXT
created	TIMESTAMP

goods	
no	INT(11)
seller_no	INT(11)
selling_area_no	INT(11)
category_no	INT(11)
subcategory_no	INT(11)
title	VARCHAR(50)
price	INT(11)
description	TEXT
created	TIMESTAMP
status	VARCHAR(45)

# 4. 프로젝트 구성

Apple Market

# MVC모델 구성(1)

- 📁 com.code.springdemo.controller
  - > 📄 categoryController.java
  - > 📄 custoemrController.java
  - > 📄 goodsController.java
  - > 📄 securityController.java

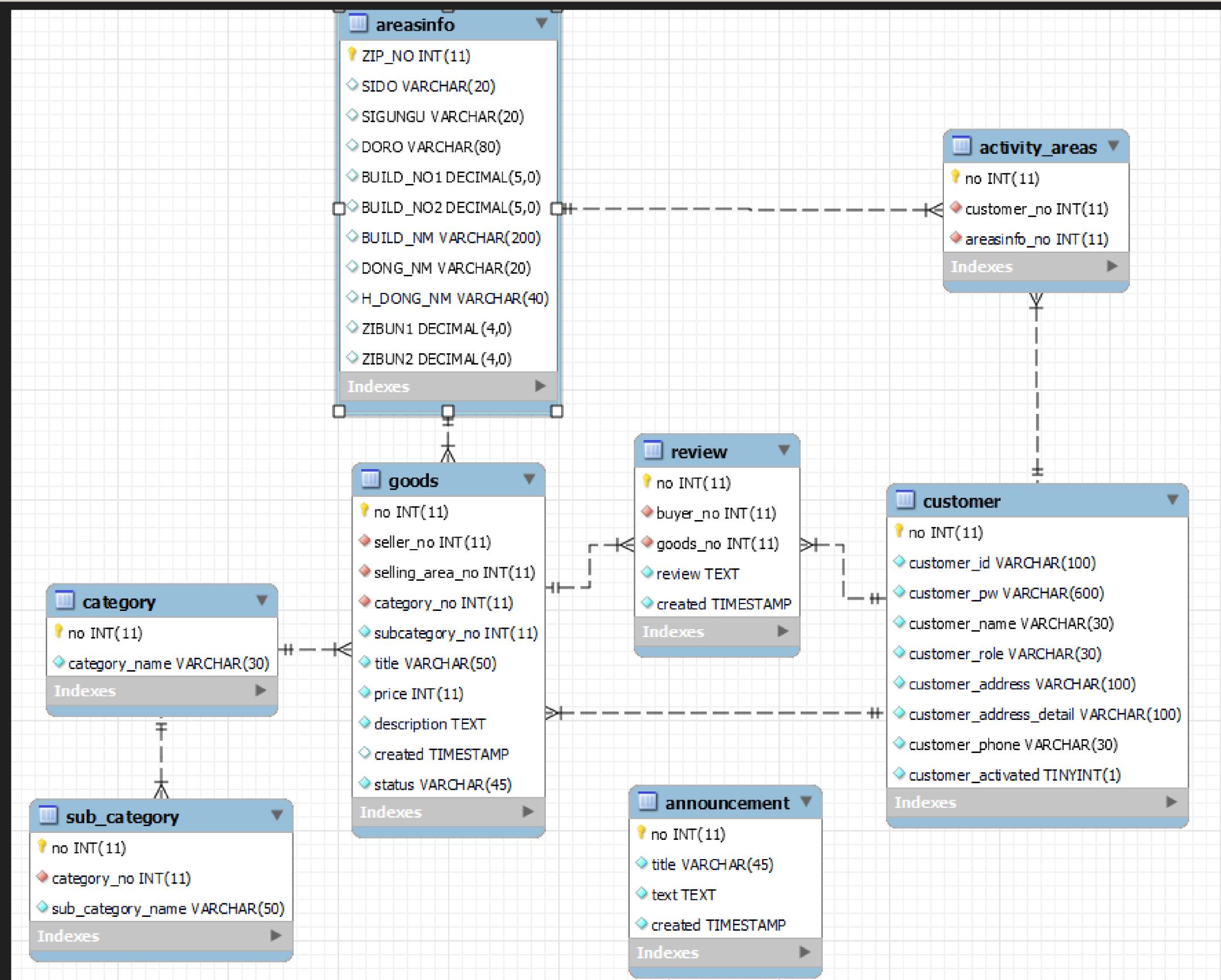
- 📁 com.code.springdemo.dao
  - > 📄 ActivityAreasDAO.java
  - > 📄 ActivityAreasDAOImpl.java
  - > 📄 AnnouncementDAO.java
  - > 📄 AnnouncementDAOImpl.java
  - > 📄 CategoryDAO.java
  - > 📄 CategoryDAOImpl.java
  - > 📄 CustomerDAO.java
  - > 📄 CustomerDAOImpl.java
  - > 📄 GoodsDAO.java
  - > 📄 GoodsDAOImpl.java

- 📁 com.code.springdemo.dto
  - > 📄 AreasInfoDTO.java
  - > 📄 CategoryDetail.java
  - > 📄 Reviews.java
  - > 📄 SubCategoryDetail.java

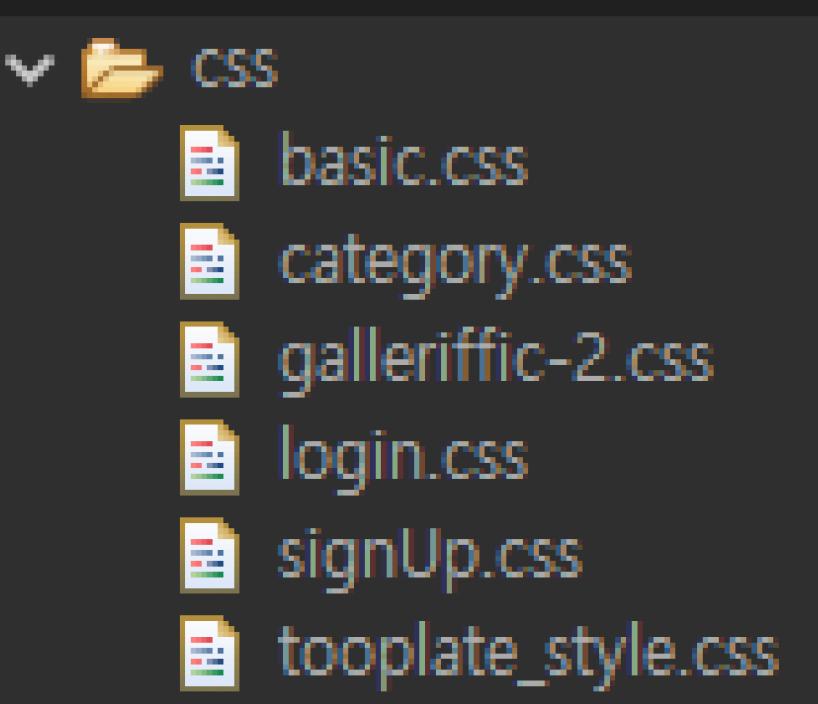
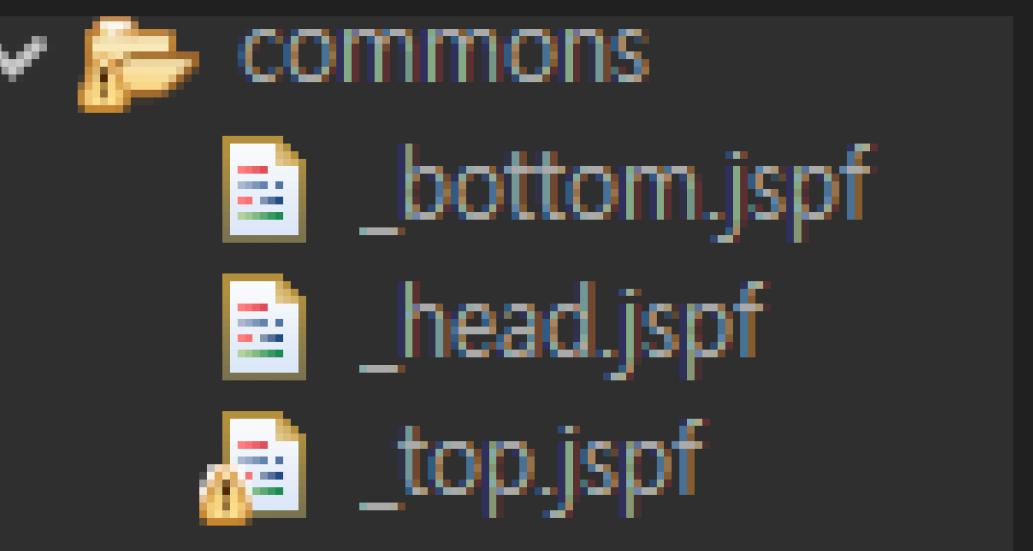
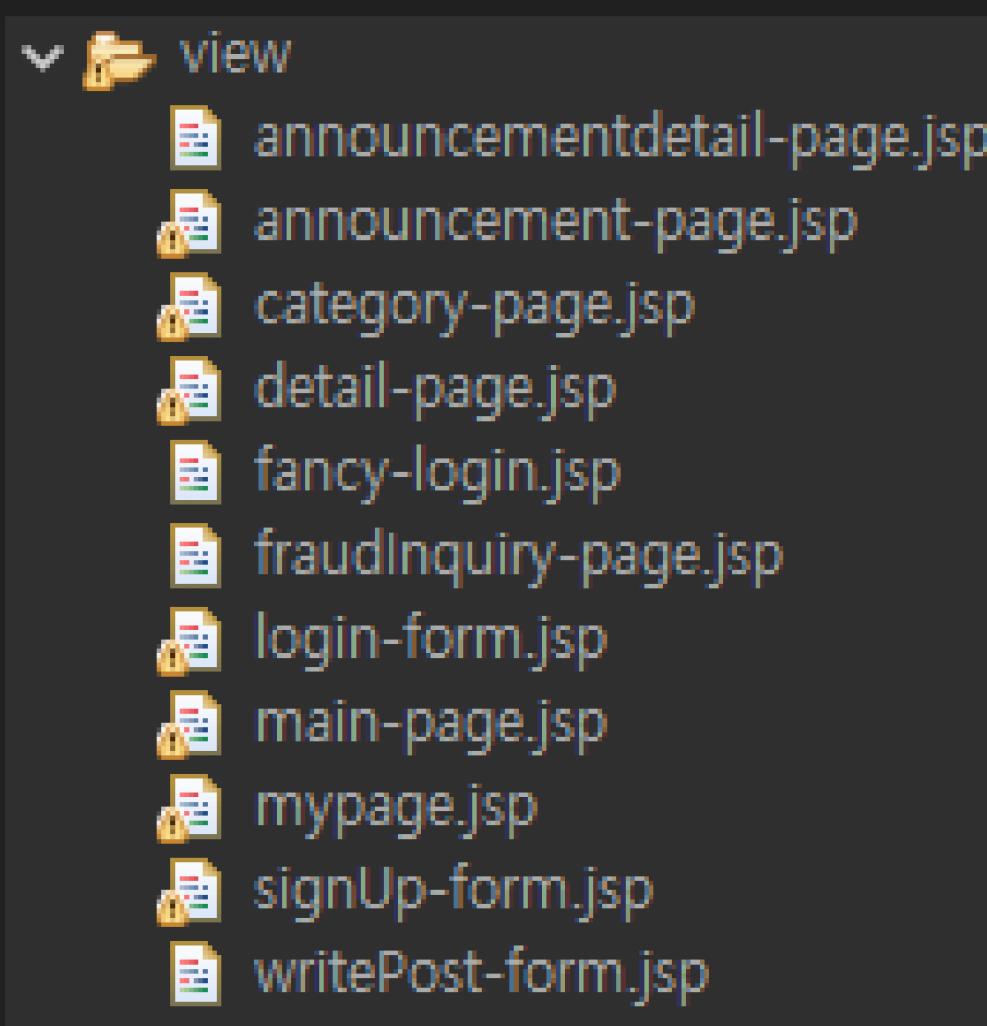
- 📁 com.code.springdemo.entity
  - > 📄 ActivityAreas.java
  - > 📄 Announcement.java
  - > 📄 AreasInfoVO.java
  - > 📄 Category.java
  - > 📄 Customer.java
  - > 📄 Goods.java
  - > 📄 Review.java
  - > 📄 SubCategory.java

- 📁 com.code.springdemo.service
  - > 📄 CustomerService.java
  - > 📄 CustomerServiceImpl.java

# EER Diagram



## MVC모델 구성(2)



# 5. 코드 리뷰

Apple Market

```
@Entity  
@Table(name="category")  
public class Category {  
  
    • @Id  
    @GeneratedValue(strategy=GenerationType.IDENTITY)  
    @Column(name="no")  
    private int no;  
  
    • @Column(name="category_name")  
    private String categoryName;  
  
    • @OneToMany(fetch = FetchType.LAZY) //지연로딩(FetchType.LAZY) : 데이터를 실질적으로 요구하는 코드를 만나면 그 때 쿼리문이 발생한다.  
        //즉시로딩(FetchType.EAGER) : 해당 Entity를 조회 시 연관관계에 있는 나머지 데이터도 조회해준다.  
    @JoinColumn(name="category_no", insertable = false, updatable = false)  
    //insertable=false , updatable=false 조인된 컬럼을 insert문과 update문에 포함할지 여부 묻는다. ↴  
    private List<SubCategory> subCategory;
```

```
@Override  
public List<Category> getCategory() {  
    Session currentSession = sessionFactory.getCurrentSession();  
    Query<Category> theQuery = currentSession.createQuery("from Category",Category.class);  
    List<Category> categorys = theQuery.getResultList();  
  
    for(int j=0;categorys.size()>j;j++) {  
        System.out.println("CategoryNo:::::"+categorys.get(j).getNo());  
        System.out.println("CategoryName:::::"+categorys.get(j).getCategoryName());  
  
        for(int i=0;categorys.get(j).getSubCategory().size()>i;i++)  
            System.out.println("CategorySubCategoryName:::::"+categorys.get(j).getSubCategory().get(i).getSubCate  
    }  
}  
  
return categorys;  
}
```

# 연관관계매핑(3)

```
CategoryNo::::1
CategoryName::::수입명품
Hibernate: select subcategor0_.category_no as category2_7_0_, subcategor0_.no as nol_7_0_, subcategor0_.no as nol_7_1_, subcategor0_.category_no as categor
CategorySubCategoryName::::전체
CategorySubCategoryName::::여성신발
CategorySubCategoryName::::남성신발
CategorySubCategoryName::::가방 / 핸드백
CategorySubCategoryName::::지갑 / 벨트
CategorySubCategoryName::::여성의류
CategorySubCategoryName::::남성의류
CategorySubCategoryName::::패션잡화
CategorySubCategoryName::::시계 / 쥬얼리
CategorySubCategoryName::::유아동
CategorySubCategoryName::::기타 수입명품
CategoryNo::::2
CategoryName::::패션의류
Hibernate: select subcategor0_.category_no as category2_7_0_, subcategor0_.no as nol_7_0_, subcategor0_.no as nol_7_1_, subcategor0_.category_no as categor
CategorySubCategoryName::::전체
CategorySubCategoryName::::여성의류
CategorySubCategoryName::::남성의류
CategorySubCategoryName::::교복 / 체육복 / 단복
CategorySubCategoryName::::클로젯세어
CategoryNo::::3
CategoryName::::패션잡화
Hibernate: select subcategor0_.category_no as category2_7_0_, subcategor0_.no as nol_7_0_, subcategor0_.no as nol_7_1_, subcategor0_.category_no as categor
CategorySubCategoryName::::전체
CategorySubCategoryName::::운동화
CategorySubCategoryName::::여성신발
CategorySubCategoryName::::남성신발
CategorySubCategoryName::::가방 / 핸드백
CategorySubCategoryName::::지갑 / 벨트
CategorySubCategoryName::::악세서리 / 귀금속
CategorySubCategoryName::::시계
CategorySubCategoryName::::선글라스 / 안경
CategorySubCategoryName::::모자
CategorySubCategoryName::::기타잡화 / 관련용품
CategoryNo::::4
CategoryName::::뷰티
Hibernate: select subcategor0_.category_no as category2_7_0_, subcategor0_.no as nol_7_0_, subcategor0_.no as nol_7_1_, subcategor0_.category_no as categor
CategorySubCategoryName::::전체
CategorySubCategoryName::::스킨케어
CategorySubCategoryName::::메이크업
CategorySubCategoryName::::팩 / 클렌징 / 필링
CategorySubCategoryName::::헤어 / 바디
```

```
@GetMapping("/category")
public String category(@RequestParam("subCategoryNo") int subCategoryNo,
                      Authentication username,
                      Model theModel) {
    Customer customer = null;
    ActivityAreas activityAreas = null;
    
        <c:param name="subCategoryNo" value="${0}"/>
    
    <li><a href="${categoryLink}"><span></span>카테고리</a>
    </li>
    if(username !=null) {
        customer = customerService.getCustomersNo(username);
        activityAreas = customerService.getActivityAreas(customer.getNo());
    }
    List<Goods> goods = null;
    List<Category> categorys = customerService.getcategory();
    List<CategoryDetail> categoryDetail = customerService.getcategoryDetail(categorys);
    List<SubCategoryDetail> subCategoryDetail = customerService.getSubCategoryDetail(categorys);

    if(subCategoryNo==0 && activityAreas==null) {
        goods = customerService.getgoods();
    }else if(subCategoryNo!=0 && activityAreas==null) {
        goods = customerService.getgoods(subCategoryNo);
    }else if(subCategoryNo==0 && activityAreas!=null) {
        goods = customerService.getgoods(activityAreas);
    }else if(subCategoryNo!=0 && activityAreas!=null) {
        goods = customerService.getgoods(subCategoryNo,activityAreas);
    }

    theModel.addAttribute("customer", customer);
    theModel.addAttribute("category", categoryDetail);
    theModel.addAttribute("subCategory", subCategoryDetail);
    theModel.addAttribute("goods",goods);
}

return "category-page";
}
```

```
@Override
public void addActivityAreas(String customerAddress, int customerNo) {
    Session currentSession = sessionFactory.getCurrentSession();
    Query theQuery = currentSession.createQuery("SELECT no from AreasInfoVO WHERE "
        + "CONCAT(sido,sigungu,doro,buildNo1) =:customerAddress");
    theQuery.setParameter("customerAddress", customerAddress.trim().replaceAll(" ", ""));
    theQuery.setCacheable(true);

    List<AreasInfo> areasInfo = theQuery.getResultList();

    ActivityAreas activityAreas = new ActivityAreas();
    activityAreas.setCustoemr_No(customerNo);
    activityAreas.setAreasInfo_No(Integer.parseInt(areasInfo.get(0).toString()));

    currentSession.saveOrUpdate(activityAreas);
}

@Override
public ActivityAreas getActivityAreas(int customer_no) {
    Session currentSession = sessionFactory.getCurrentSession();

    Query theQuery = currentSession.createQuery("from ActivityAreas WHERE customer_No = :customer_no", ActivityAreas.class);
    theQuery.setParameter("customer_no", customer_no);

    List<ActivityAreas> areasInfo = theQuery.getResultList();

    ActivityAreas activityAreas = new ActivityAreas();
    activityAreas.setNo(areasInfo.get(0).getNo());
    activityAreas.setCustoemr_No(areasInfo.get(0).getCustoemr_No());
    activityAreas.setAreasInfo_No(areasInfo.get(0).getAreasInfo_No());

    return activityAreas;
}
```

```
@GetMapping("/category")
public String category(@RequestParam("subCategoryNo") int subCategoryNo,
                      Authentication username,
                      Model theModel) {
    Customer customer = null;
    ActivityAreas activityAreas = null;
    
        <c:param name="subCategoryNo" value="${0}"/>
    
    <li><a href="${categoryLink}"><span></span>카테고리</a>
    </li>
    if(username !=null) {
        customer = customerService.getCustomersNo(username);
        activityAreas = customerService.getActivityAreas(customer.getNo());
    }
    List<Goods> goods = null;
    List<Category> categorys = customerService.getcategory();
    List<CategoryDetail> categoryDetail = customerService.getcategoryDetail(categorys);
    List<SubCategoryDetail> subCategoryDetail = customerService.getSubCategoryDetail(categorys);

    if(subCategoryNo==0 && activityAreas==null) {
        goods = customerService.getgoods();
    }else if(subCategoryNo!=0 && activityAreas==null) {
        goods = customerService.getgoods(subCategoryNo);
    }else if(subCategoryNo==0 && activityAreas!=null) {
        goods = customerService.getgoods(activityAreas);
    }else if(subCategoryNo!=0 && activityAreas!=null) {
        goods = customerService.getgoods(subCategoryNo,activityAreas);
    }

    theModel.addAttribute("customer", customer);
    theModel.addAttribute("category", categoryDetail);
    theModel.addAttribute("subCategory", subCategoryDetail);
    theModel.addAttribute("goods",goods);
}

return "category-page";
}
```

```
@Override  
@Transactional  
public List<Goods> getgoods() {  
  
    return goodsDAO.getgoods();  
}  
  
@Override  
@Transactional  
public List<Goods> getgoods(int subCategoryNo) {  
  
    return goodsDAO.getgoods(subCategoryNo);  
}  
|  
  
@Override  
@Transactional  
public List<Goods> getgoods(ActivityAreas activityAreas) {  
  
    return goodsDAO.getgoods(activityAreas);  
}  
  
@Override  
@Transactional  
public List<Goods> getgoods(int subCategoryNo, ActivityAreas activityAreas) {  
  
    return goodsDAO.getgoods(subCategoryNo, activityAreas);  
}
```

```
@Override  
public List<Goods> getgoods(ActivityAreas activityAreas) {  
    Session currentSession = sessionFactory.getCurrentSession();  
    Query<Goods> theQuery = currentSession.createQuery("from Goods WHERE sellingAreaNo= :sellingArea_No ORDER BY created DESC",Goods.class);  
    int i = activityAreas.getAreasInfo_No();  
    theQuery.setParameter("sellingArea_No",i);  
    theQuery.setMaxResults(30);  
    List<Goods> goods = theQuery.getResultList();  
  
    return goods;  
}
```

# 카테고리(6)

```
        break;
    case 28:
        theQuery = currentSession.createQuery("from Goods WHERE categoryNo = :category_No ORDER BY created DESC",Goods.class);
        theQuery.setParameter("category_No", 4);
        theQuery.setMaxResults(30);
        goods = theQuery.getResultList();
        break;
    case 37:
        theQuery = currentSession.createQuery("from Goods WHERE categoryNo = :category_No ORDER BY created DESC",Goods.class);
        theQuery.setParameter("category_No", 5);
        theQuery.setMaxResults(30);
        goods = theQuery.getResultList();
        break;
    case 46:
        theQuery = currentSession.createQuery("from Goods WHERE categoryNo = :category_No ORDER BY created DESC",Goods.class);
        theQuery.setParameter("category_No", 6);
        theQuery.setMaxResults(30);
        goods = theQuery.getResultList();
        break;
    case 54:
        theQuery = currentSession.createQuery("from Goods WHERE categoryNo = :category_No ORDER BY created DESC",Goods.class);
        theQuery.setParameter("category_No", 7);
        theQuery.setMaxResults(30);
        goods = theQuery.getResultList();
        break;
    case 66:
        theQuery = currentSession.createQuery("from Goods WHERE categoryNo = :category_No ORDER BY created DESC",Goods.class);
        theQuery.setParameter("category_No", 8);
        theQuery.setMaxResults(30);
        goods = theQuery.getResultList();
        break;
    case 80:
        theQuery = currentSession.createQuery("from Goods WHERE categoryNo = :category_No ORDER BY created DESC",Goods.class);
        theQuery.setParameter("category_No", 9);
        theQuery.setMaxResults(30);
        goods = theQuery.getResultList();
        break;
    default:
        theQuery = currentSession.createQuery("from Goods WHERE subCategory_no = :category_No ORDER BY created DESC",Goods.class);
        theQuery.setParameter("category_No", subCategoryNo);
        theQuery.setMaxResults(30);
        goods = theQuery.getResultList();
```

```
<body id="subpage">
    <div id="tooplate_wrapper">
        <div id="tooplate_header">
            <%@ include file="../../commons/_top.jspf"%>
        <div id="tooplate_main">
            <div id="tooplate_sidebar">
                <div class="sidebar_box">
                    <c:choose>
                        <c:when test="${customer.customerAddress eq null}">
                            <h2>카테고리</h2>
                        </c:when>
                        <c:otherwise>
                            <h2>카테고리</h2>
                            <h5>${customer.customerAddress}</h5>
                        </c:otherwise>
                    </c:choose>
                    <ul class="sidebar_nav">
                        <c:forEach var="tempCategory" items="${category}">
                            <details>
                                <summary><h5>${tempCategory.categoryName}</h5></summary>
                                <c:forEach var="tempSubCategory" items="${subCategory}">
                                    <c:if test="${tempCategory.no == tempSubCategory.categoryNo}">
                                        <c:url var="categoryLink" value="category">
                                            <c:param name="subCategoryNo" value="${tempSubCategory.no}" />
                                        </c:url>
                                        <a href="${categoryLink}">${tempSubCategory.subCategoryName}</a>
                                        <br>
                                    </c:if>
                                </c:forEach>
                            </details>
                            <hr>
                        </c:forEach>
                    </ul>
                </div>
            </div>
```

```
<div id="tooplate_content">
    <h2>판매상품타이틀</h2>
    <a href="writePost" class="more float_r"><strong>상품등록</strong></a>
    <c:forEach var="tempGoods" items="${goods}">
        <c:url var="detailpageLink" value="detailPage">
            <c:param name="goodsNo" value="${tempGoods.no}" />
        </c:url>
        <div class="post_box">
            <c:forEach var="tempCategory" items="${category}">
                <c:if test="${tempCategory.no == tempGoods.categoryNo}">
                    [&${tempCategory.categoryName}]
                </c:if>
                </c:forEach>
                <c:forEach var="tempSubCategory" items="${subCategory}">
                    <c:if test="${tempSubCategory.no == tempGoods.subCategory_no}">
                        [&${tempSubCategory.subCategoryName}]
                    </c:if>
                    </c:forEach>
                    <h5>제목:&${tempGoods.title}</h5>
                    
                    <p>가격:&${tempGoods.price}원</p>
                    <p>게시글등록일:&${tempGoods.created}</p>
                    <security:authorize access="isAuthenticated()">
                        <a href="${detailpageLink}" class="more float_r" style="font-size: 12px">자세히보기</a>
                    </security:authorize>
                    <div class="cleaner"></div>
                </div>
            </c:forEach>
```

# 6. 마무리

Apple Market

Home 카테고리 공지사항 로그인 회원가입

# Apple Market



Copyright © 2020 Company Name

## 업데이트 예정

- 해당 품목 검색 기능
- 관심상품 등록 기능
- 사용자 메시지 기능
- 사용자간 거리측정 기능
- 리뷰 누적기능(매너온도)
- 이미지업로드 기능(다중)
- 등등...

## 아쉬웠던 점

- 예정에 없던 프로젝트

## 후기

---

# Thank You

Apple Market