hyungji 형지엘리트

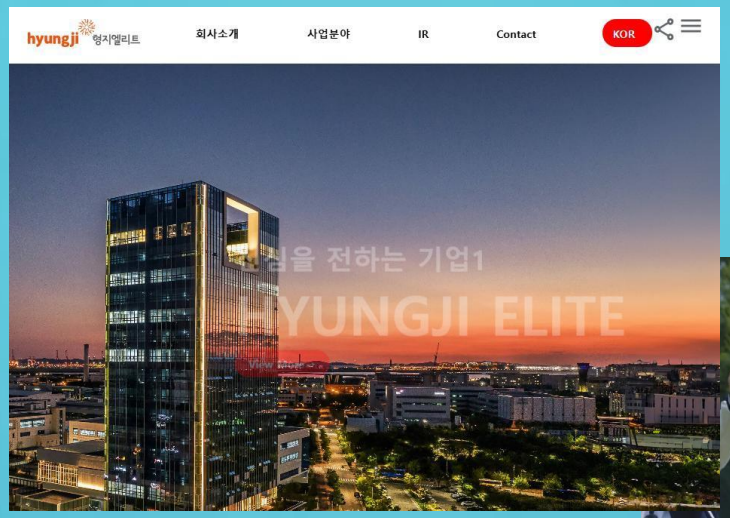
WEB DESIGN PROJECT

형지엘리트 웹 사이트
COPY

100vw WIDTH
100vh FLEX SYSTEM
개인 작업 100%
2024

Language : JaveScript

Makeup : HTML5 CSS3



INDEX

1. 프로젝트 개요
2. 기획 의도 및 목표
3. 디자인 콘셉트
4. 웹 사이트 상세 설명

OUR BUSINESS

형지엘리트는 다양한 분야에서 품질과 스타일을
검비한 다양한 솔루션을 제공합니다.

더 편한 교복 elite

엘리트 학생복

55년 교복 산업을 선도해온 대한민국 NO.1 엘리트

더 편한 교복은 물론 체육복, 학생 용품까지!
학생용품 토탈 솔루션을 제공합니다.

#초등학교 #국제학교 #중학교

#고등학교 #K교복 #상해엘리트

01

프로젝트 개요

프로젝트 명 : 형지엘리트 웹 사이트 COPY

사이트 주소 : https://lee-kunwoo.github.io/hyungji_pc/ (ｐｃ버전) https://lee-kunwoo.github.io/hyung_mobile/ (모바일버전)

부문 : 형지엘리트 웹 사이트

클라이언트 : 형지엘리트 제작자 : 이건우

제작일 : 2025.02.16

02

기획 의도 및 목표

형지엘리트는 '엘리트' 학생복 브랜드로 유명하며, 50년간 학생복을 주도해온 기업이다. 주 아이템은 학생복과 청소년 스포츠웨어, 유니폼을 전문으로 한다.

철저하게 학습을 위한 목적으로 접근한 사이트로 탑 다운 메뉴와 이미지변환, 버튼 이벤트의 동적인 슬라이드 구현, 텍스트 타이핑, 동영상, 스코롤 이벤트의 동적 구성을 감안하여 COPY 목적으로 선정하였다.

회사의 인트로인 메인페이지 만을 COPY 하여 전체 페이지 구성을 이루지 못하였으나, 동적인 웹페이지 구현에 많은 도움이 된 사이트다.

처음부터 반응형과 함께 구현하였으면 제작기간을 단축 할 수 있었으나, 시간을 소비한 아쉬움이 있다. 다만 학습을 목적으로 구현한 사이트인 만큼 향후 시간을 가지고 서브페이지를 포함한 전체 웹페이지를 구현하는 것이 목표이다.

03
디자인 콘셉트

브랜드 메인 컬러로 밝은
채도의 오렌지 컬러를 선택했다.

Color and Typography

Hyungji **영문체:** IBM Plex Sans

형지엘리트 **한글체:** IBM Plex Sans KR

#FF5E24 #FFFFFF #000000

FLEX SYSTEM

OUR SPECIAL FIELD

형지엘리트는 폭넓고 다채로운 분야에서
고객의 다양한 라이프스타일 지원을 위해 끊임없이 도전합니다.

각 분야별 클라이언트의 니즈 반영

현장조사 및 샘플제작으로 다양한 산업분야의 특성에 따른 맞춤형형 워크웨어 생산

학생용품 토탈 솔루션

교복매장에서 이너티, 스타킹, 벨트 등 학생 용품까지 원스톱으로 구매 가능

근무환경에 맞춘 라인업

유니폼, 단체복, 작업복, 일상복까지 다양한 근무환경에 맞춘 라인업 구성

스포츠 구단 스폰 및 매치데이 개최

스포츠 구단 메인 스폰 및 매치데이 행사 개최

04

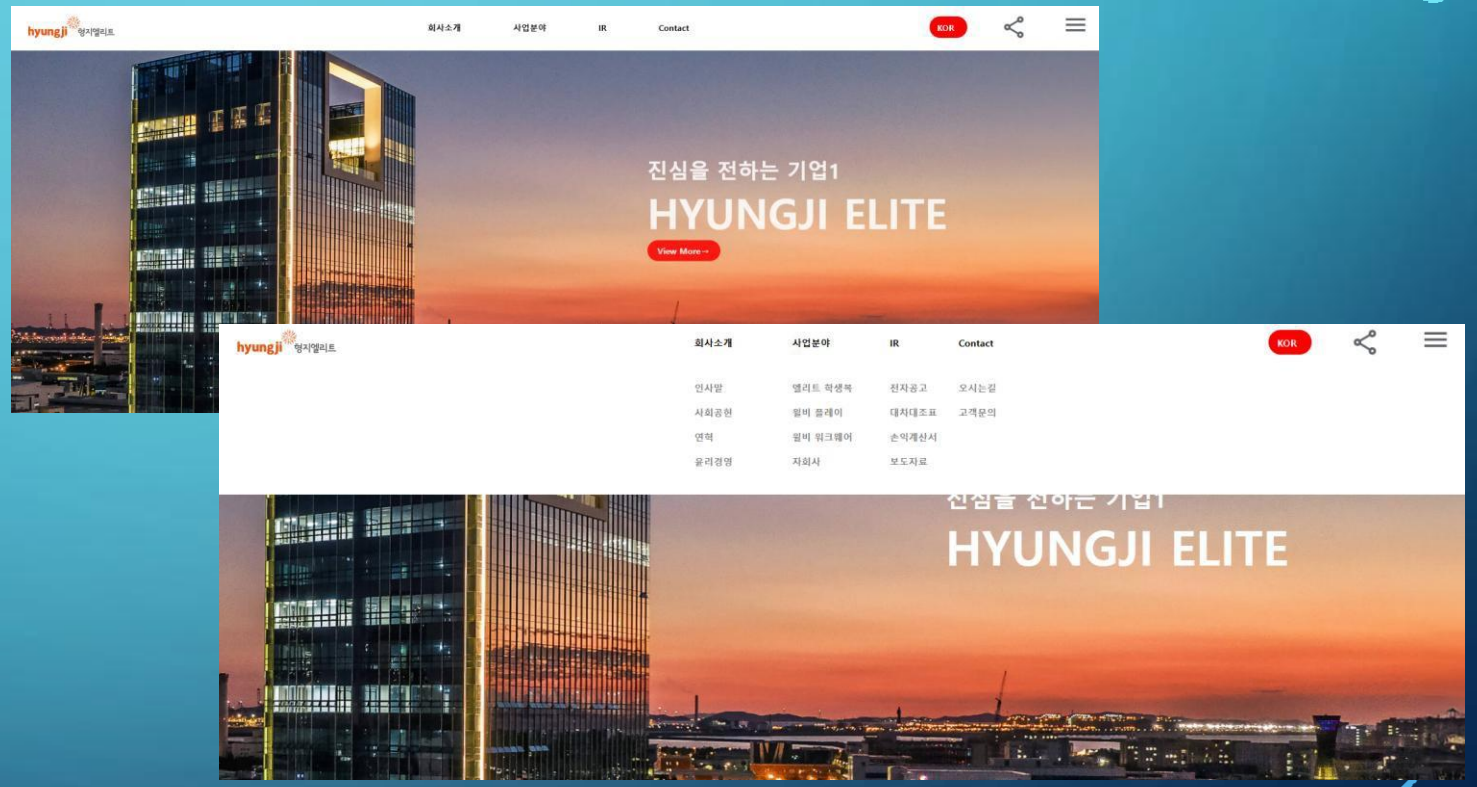
메인 페이지

메인페이지-1

각각의 메인 메뉴와 메인 로고가 있는 네비게이션 바를 마우스로 올리면 탑다운 메뉴가 슬라이드 되며 언제든지 상단 네비게이션 바를 이용하여 원하는 메뉴를 클릭하여 페이지를 이동할 수 있게 만들어 네비게이션 바를 넓게 사용하였다. 네비게이션 바 하단에는 높이가 700px 사이즈로 회사의 사옥이 자동으로 변환되는 슬라이드로 구현

형지엘리트는 사옥을 메인페이지에 배치함으로서 유저에게 규모가 있는 기업이라는 인상을 줌

기업가치 이념을 텍스트 변환로딩 처리하여 관심을 유도함



05

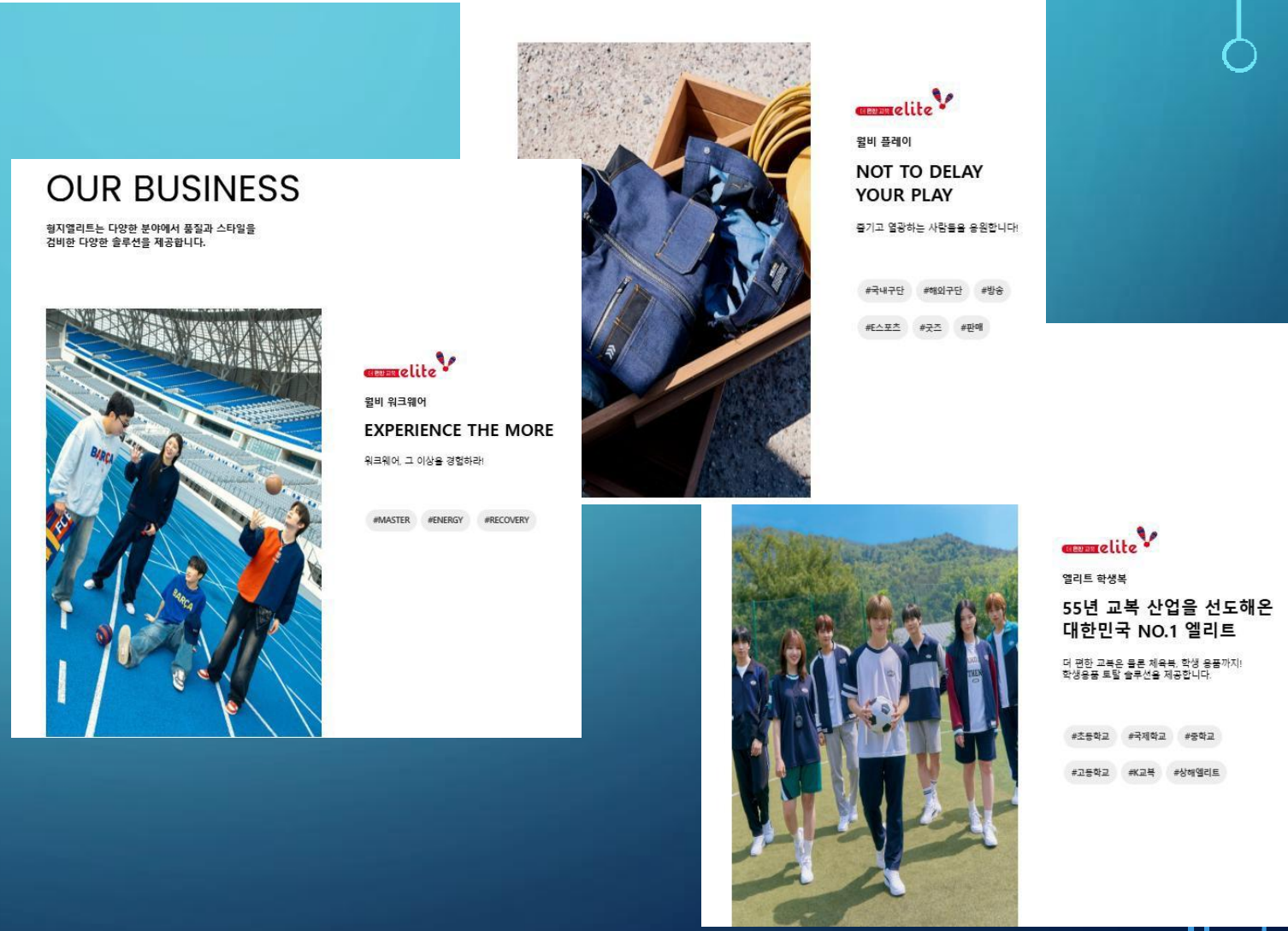
웹사이트 상세설명

메인페이지-2

상단 메인 이미지에서 스크롤을 내리면 브랜드의 아이덴티티가 적힌 문구가 스코롤 이벤트(아래로 위로 이동해도 타이핑이 자동으로 됨)

그 밑으로는 버튼을 클릭할 때 마다 제품설명과 모델사진등이 한장씩 총 3장이 좌우에서 슬라이드 되게 구성하였다,

많은 상품들을 한장씩 버튼에 따라 보여 줌으로써 유저에게 호기심을 자극하고 또한 동적인 에니메이션으로 화면에 계속 집중하고 유도하는 구성이다.



06

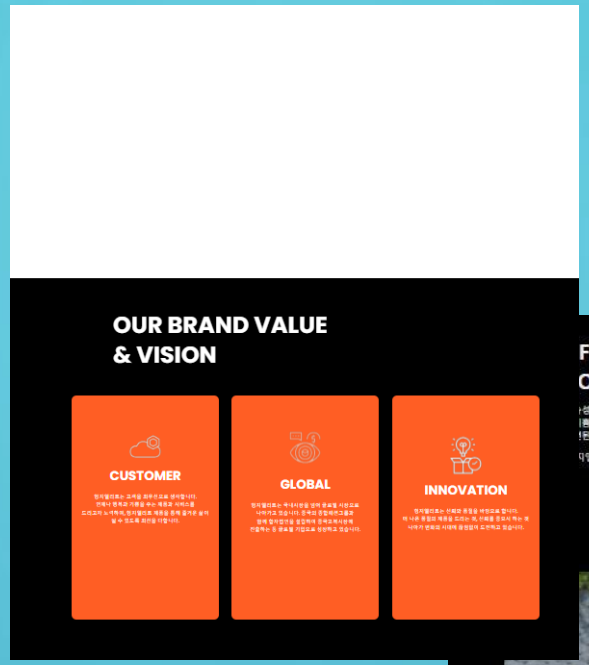
웹사이트 상세설명

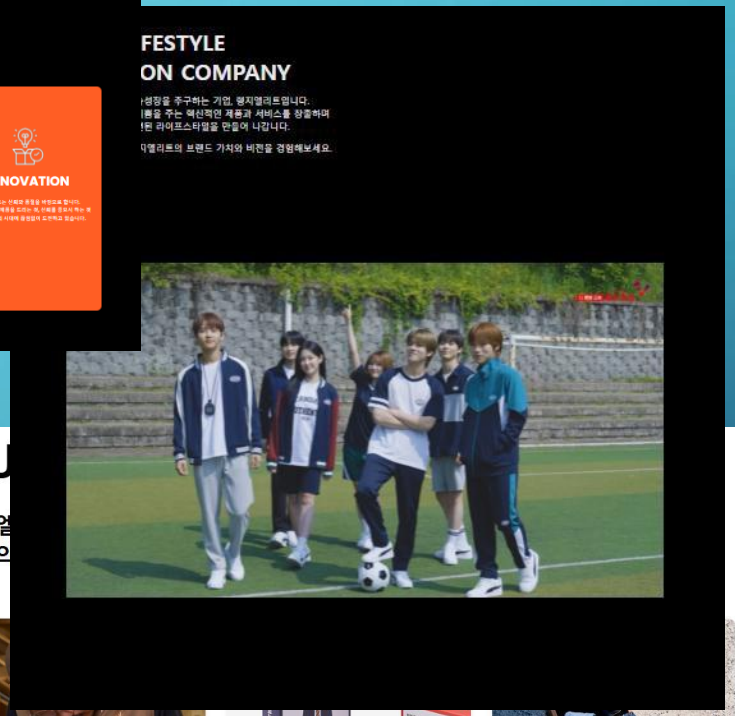
메인페이지-3

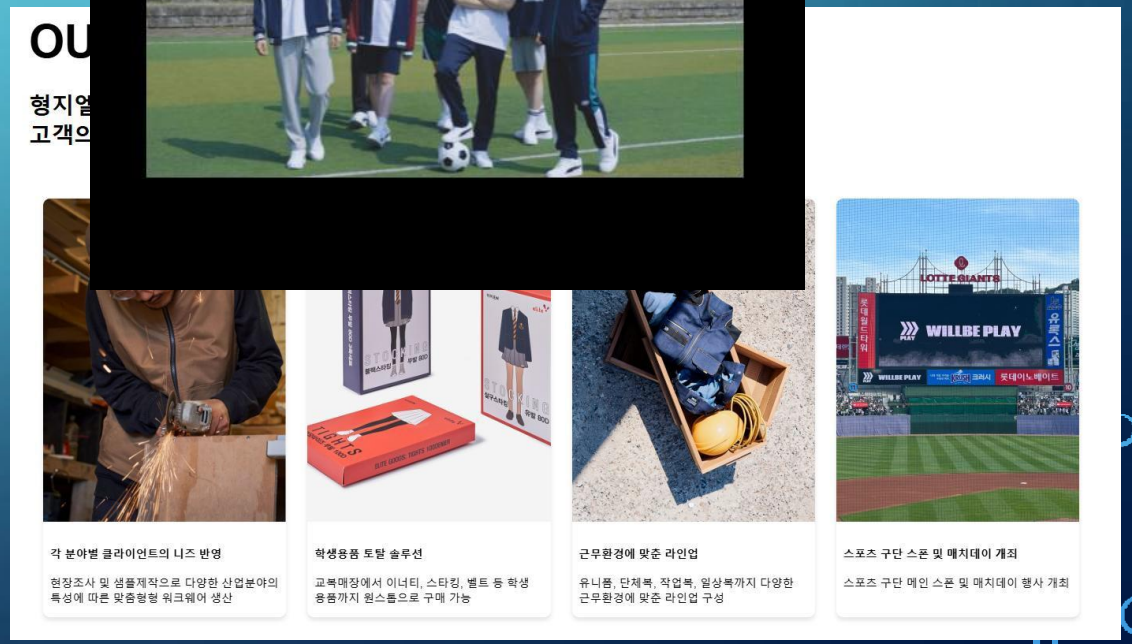
다음 부에는
형지엘리트 브랜드의 홍보영상을 Autoplay로
삽입 하였고, 배경색상과 텍스트, 동영상이 스크롤이
접근되면 자동으로 색상변환. 텍스트 크기조절, 영상이
커짐으로써 동적인 에니메이션 구현

다음페이지에는 상품설명이 상세하게 나타나 있는
카드가 스크롤이 아래 , 위로 이동됨에 따라 카드가
180도 회전되며,

그 다음페이지에는 상품설명이 되어 있는 플렉스
레이아웃으로 나열하는 구성을 하였다.







07

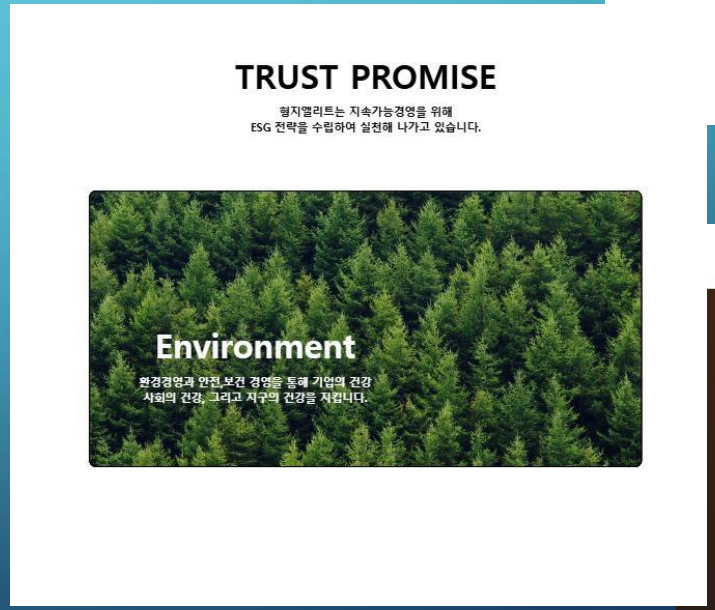
웹사이트 상세설명

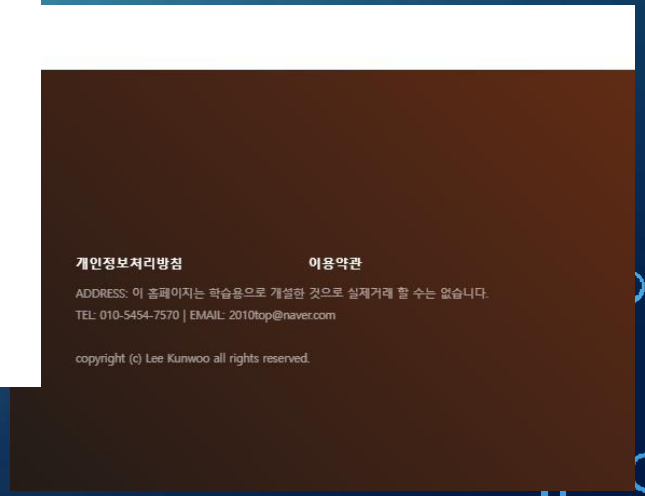
메인페이지-4

다음 부에는
형지엘리트 협력사들의 기업로고가
두즐로 좌에서 우로, 우에서 좌로 천천히 이동하며
기업협업이 활발하게 이루어 지고 있는 것을 홍보함.

다음페이지에는 기업가치의 환경보전과 상생의
이미지를 마우스와 스크롤이 위, 아래로 이동함에 따라
이미지와 텍스트가 자동으로 변환되게 구성함(이 영역은
형지엘리트를 COPY 한게 아니라 RE_DESIGN 하였음)

마지막부에는 형지엘리트를 COPY 하였다는 학습용 목적
임을 나타내어 유저에게 정보를 제공함





JungwoonNailArt

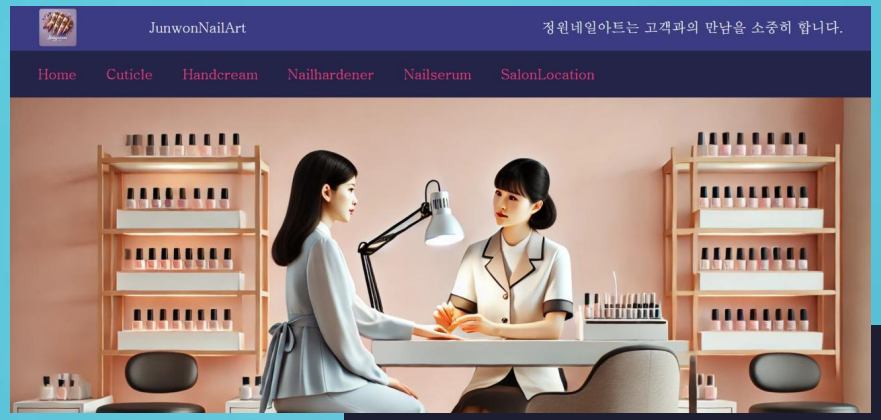
WEB DESIGN PROJECT

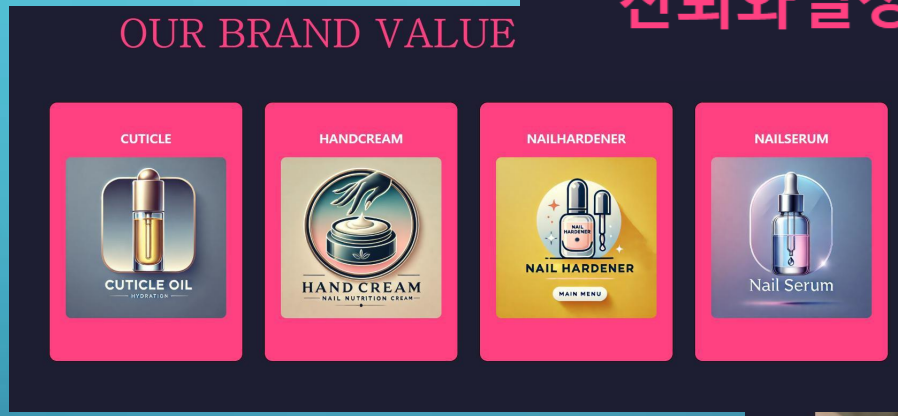
정원네일아트 웹 사이트 CREATE

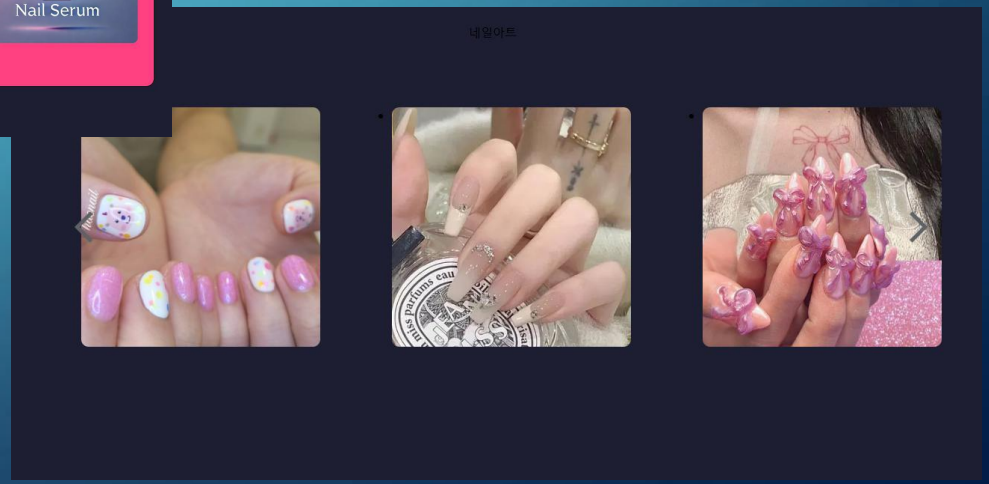
100vw WIDTH
100vh FLEX SYSTEM
개인 작업 100%
2025

Language : React Makeup : HTML

5 SCSS 3







INDEX

1. 프로젝트 개요
2. 기획 의도 및 목표
3. 디자인 콘셉트
4. 웹 사이트 상세 설명

01

프로젝트 개요

프로젝트 명 : 정원네일아트 웹 사이트 CREATE

사이트 주소 : https://lee-kunwoo.github.io/

부문 : 개인사업자(네일아트) 웹 사이트

클라이언트 : Any nail salon owner can become a client.

제작자 : 이건우

제작일 : 2025.02.20

02

기획 의도 및 목표

정원네이아트는 임의적으로 만든 네일아트 관련 사이트로 점차적으로 개인사업주도 전문성이 강화 되고 있어,
네일아트을 주사업으로 하고 있는 개인사업주에 제공하기 위한 의도로 제작 하였다.

리엑트 학습을 위한 사이트로 텍스트 메뉴와 이미지변환, 텍스트 타이핑, 동영상, 자동슬라이드 및 버튼이벤트 슬라이드로 구현된 사이트이다.

인트로인 메인페이지, 네일아트 관련 제품들의 서브페이지, 네일샵 위치등 모든 페이지를 제작하였으나.
제품의 이미지는 동일하게 적용하였으며 정보 또한 같다.

마지막으로 처음부터 반응형과 함께 구현하여, 제작기간을 단축하고 무엇 보다도 리엑트 웹사이트를 완전 구현 하였다는 것에 의의를 갖는다.

03
디자인 콘셉트

FLEX SYSTEM

브랜드 메인 컬러로 화려한
색상의 핑크컬러를 선택했다.

Color and Typography

JungWoon **영문체 : Garamond**

정원네일아트 **한글체 : Batang**

#FF4081 #FFFFFF #1e1e33

04

메인 페이지

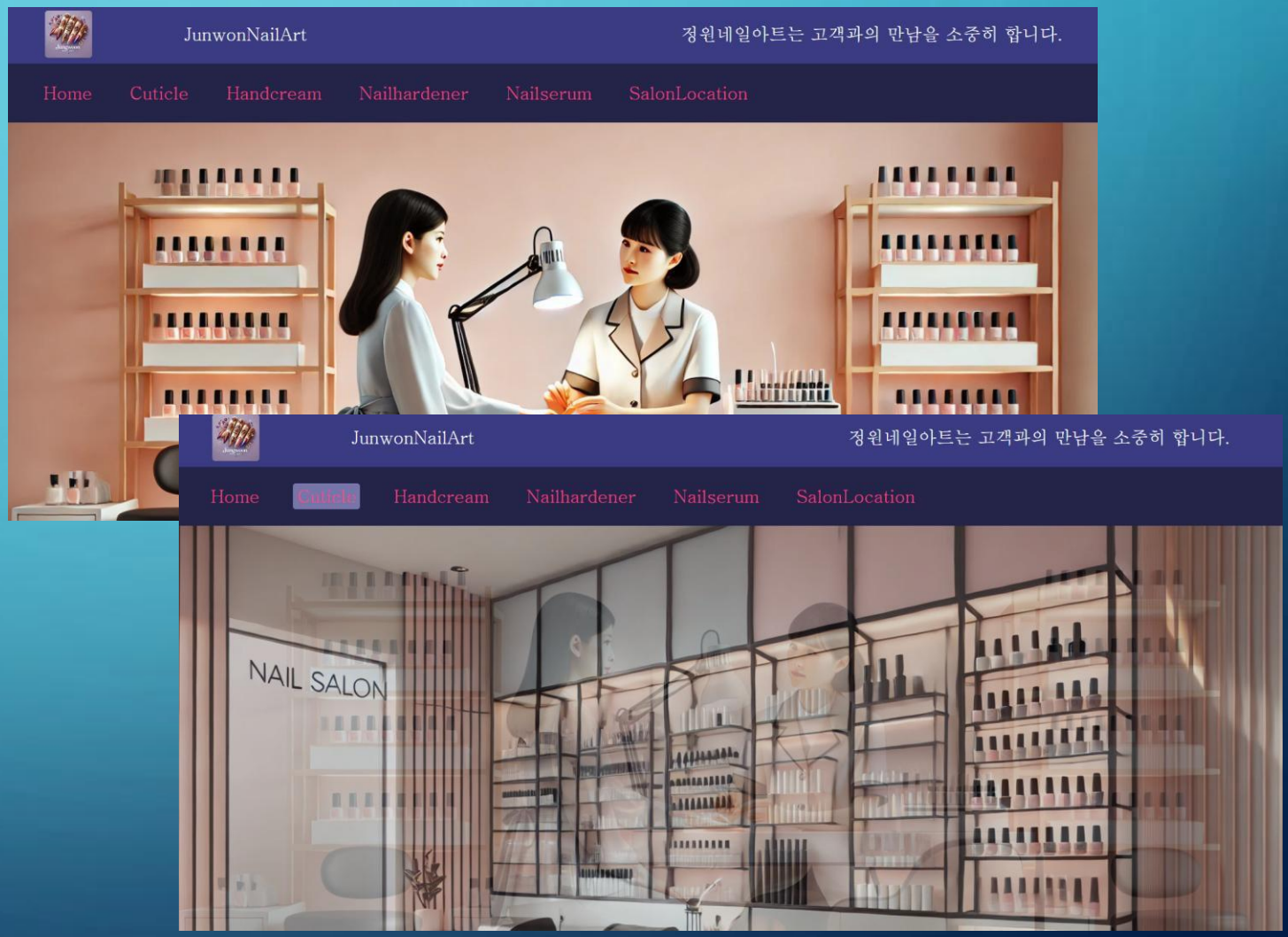
메인페이지-1

각각의 메인 메뉴와 메인 로고가 있는 네비게이션 바의 메뉴는 정지상태의 호버로 메뉴 주위가 색상이 변경되며, 언제든지 상단 네비게이션 바를 이용하여 원하는 메뉴를 클릭하여 페이지를 이동할 수 있게 만들었다.

네비게이션 바 하단에는 높이가 1000px 사이즈로 AI가 생성한 네일아트샵 이미지가 자동으로 변환되는 슬라이드로 구현

샵의 이미지를 전면에 배치함으로 유저에게 개인사업자임을 암시함.

가치 이념을 텍스트 변환로딩 처리하여 관심을 유도함



05

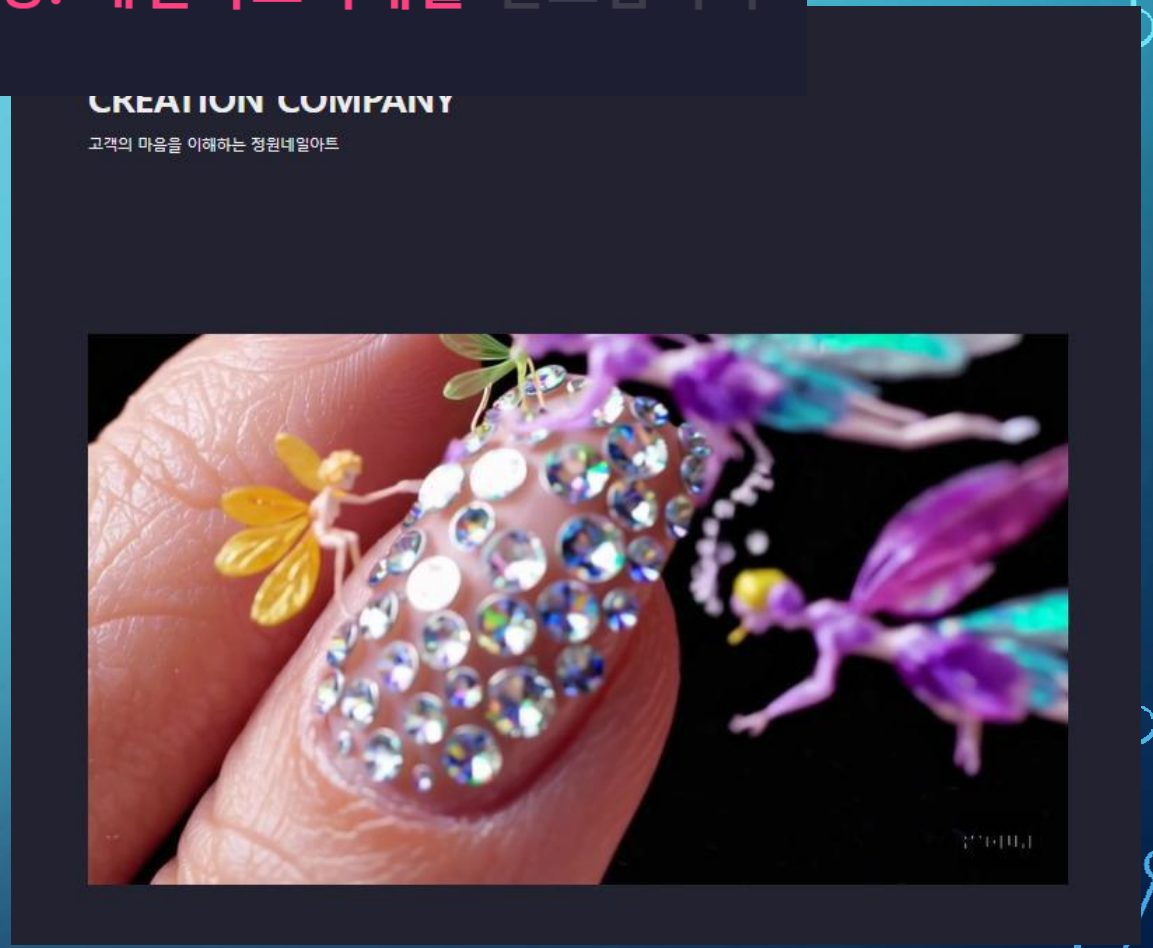
웹사이트 상세설명

메인페이지-2

상단 메인 이미지에서 스크롤을 내리면 브랜드의
아이덴티티가 적힌 문구가 스코롤
이벤트(아래로 위로 이동해도 타이핑이 자동으로 됨)

그 밑으로는 ChatGPT Sora 의 네일아트 가상 동영상이
자동으로 로딩되어 유저에게 웹페이지가 네일아트
웹사이트 라는 강한 인상을 남겨줌.

유저는 상상의 네일아트 영상을 감상함으로써 샵 방문에
대한 동기를 부여함



06

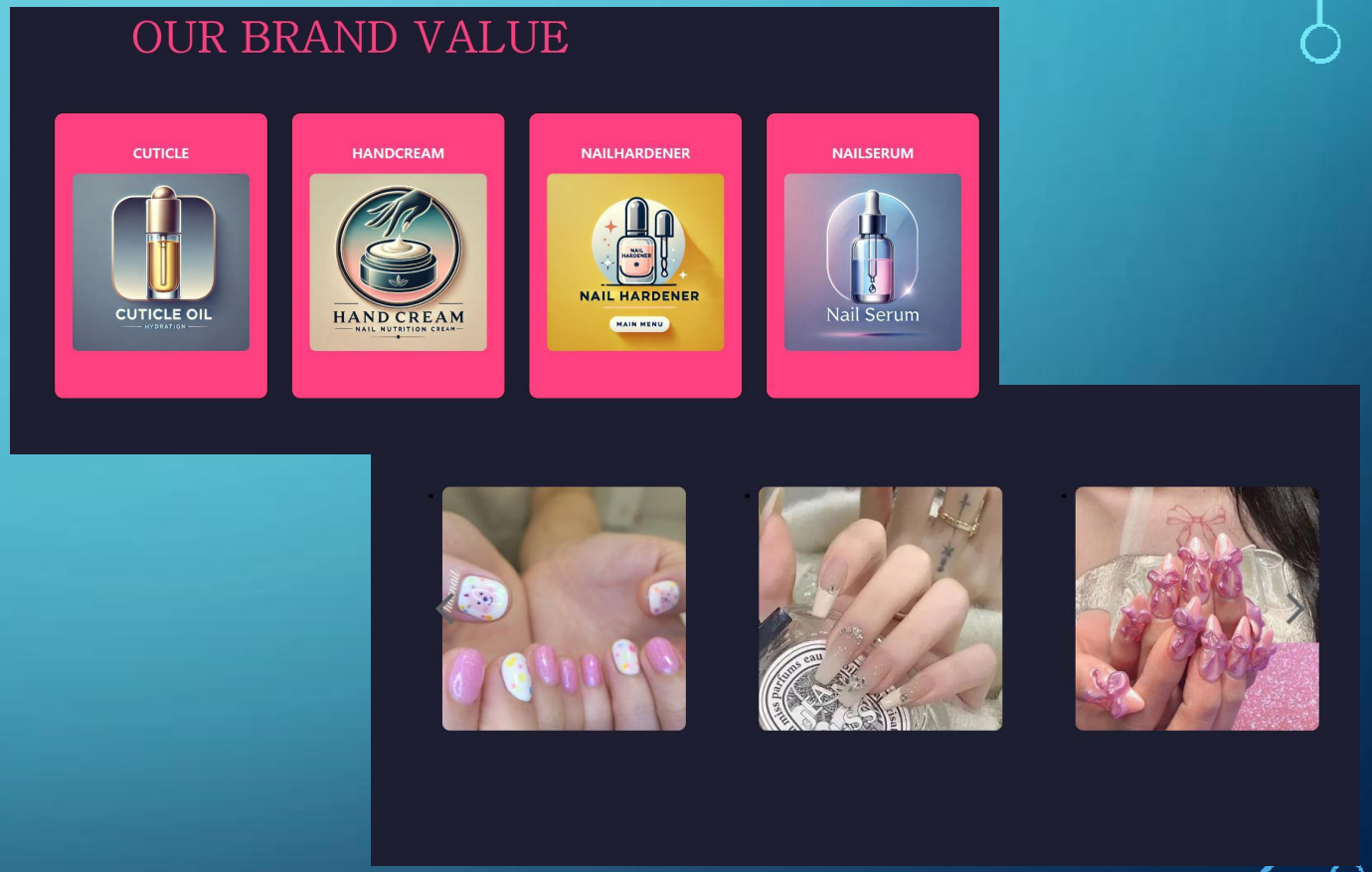
웹사이트 상세설명

메인페이지-3

다음 부에는
네일아트 관련 상품 이미지가 나타나 있는 카드가
스크롤이 아래 , 위로 이동됨에 따라 카드가 180도
회전되어 동적인 에니메이션 강화로 유저에게 지루함을
방지하도록 설계함

그 다음페이지에는 네일아트를 한 사람들의 이미지가
자동슬라이드 되며, 버튼에 따라 슬라이드가 이동되게 함.

다만, 인터넷의 이미지를 캡쳐하여 화질이 현저하게
떨어지며 이미지의 크기가 일정하지 않은 것이 아쉬움.



07

웹사이트 상세설명

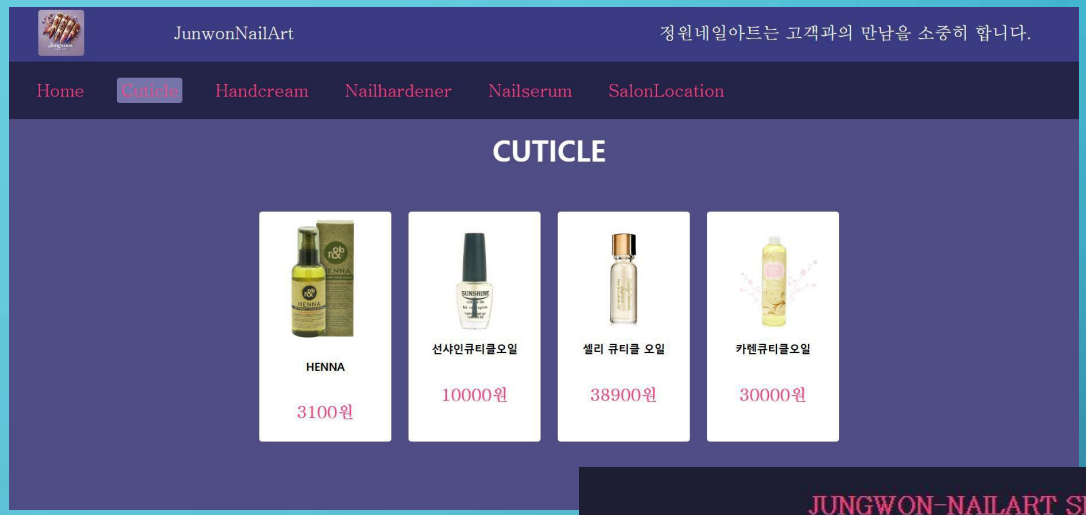
메인페이지-4

서브 페이지는 네일아트 관련상품을 쇼핑몰로 구성하여 유저에게 네일아트샵과 네일아트 전문가라는 인상을 주기 위한 다분히 유저에게 신뢰와 안정을 주기 위한 일종의 홍보 강화 페이지임

네일아트샵의 위치를 카카오맵으로 설계하여 유저 편리성을 강화 하였음

카카오맵은 다양한 기능은 없으나, 본래의 샵위치 정보를 알려주는 맵으론 손색이 없다고 생각함.

학습자입장에서는충분한 학습기회가 적은 아쉬움이 있음.

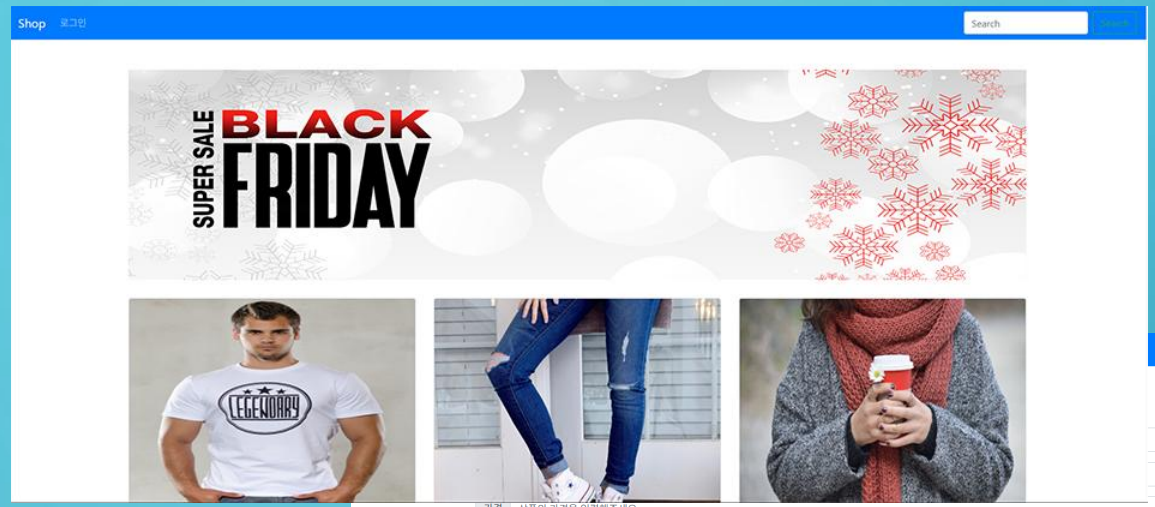


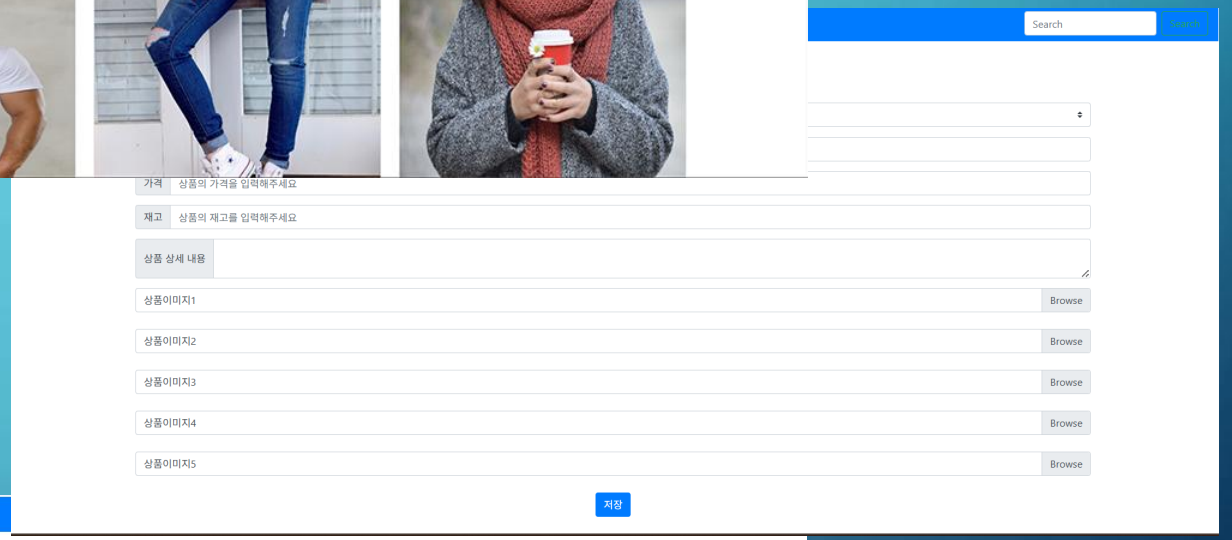
ShopTest PROJECT

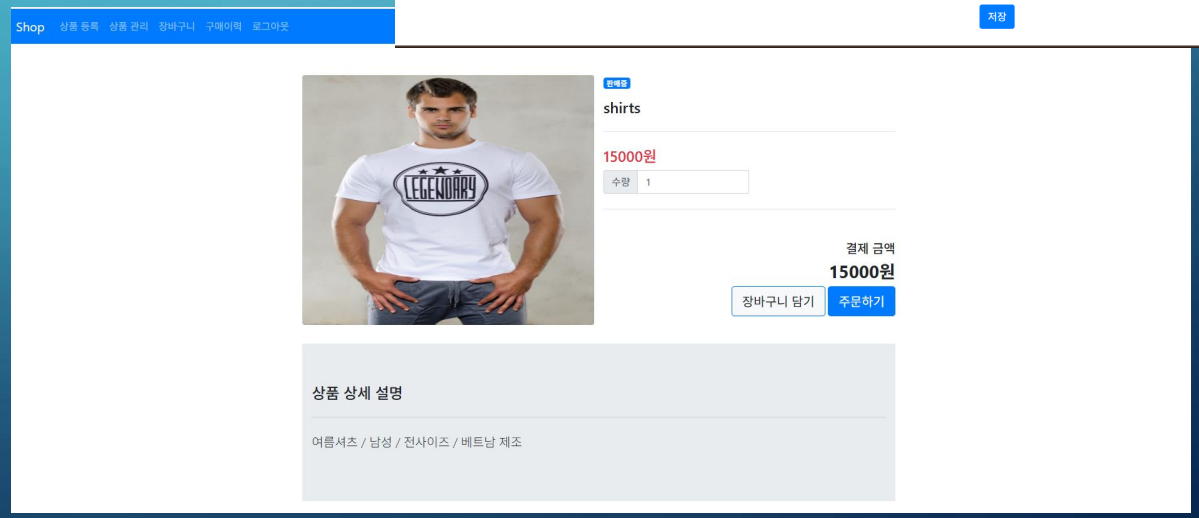
의류쇼핑몰 테스트 페이지

Spring Boot 기반 Full-Stack 웹 서비스 CREATE
개인작업 2025. 10.

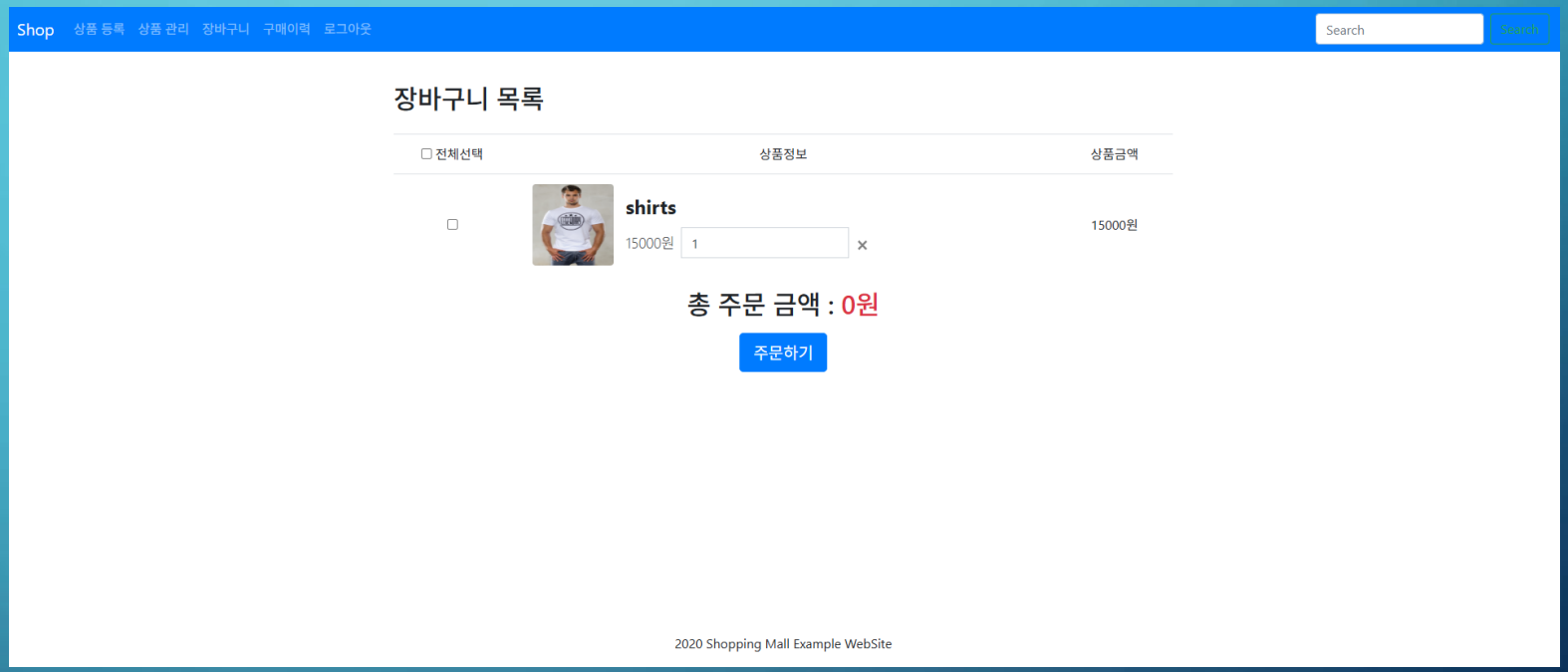
Language : Java, Spring Boot
Makeup : HTML5, CSS3, Thymeleaf used for template rendering







INDEX



01

프로젝트 개요

구분	내용
프로젝트 명	Spring Boot 기반 Full-Stack 웹 서비스 CREATE (예시: 개인 미니 쇼핑몰 또는 커뮤니티)
사이트 주소	http://springboot-leekunwoo-env.eba-8vnf42tm.ap-northeast-2.elasticbeanstalk.com/
부문	백엔드 중심 Full-Stack 웹 서비스 (Backend/Full-Stack Web Service)
클라이언트	개인 학습 및 백엔드 시스템 설계 역량 강화 목적
제작자	이건우
제작일	2025.12.10

02

기술 스택 (Key Tech Summary)

구분	내용
Language	Java, Spring Boot
Frontend	HTML5, CSS3, Thymeleaf used for template rendering
Database	PostgreSQL
Deployment	AWS Elastic Beanstalk

03

기획 의도 및 목표

Full-Stack 아키텍처 이해 :

프론트엔드(UI/UX)와 백엔드(데이터 처리/API) 간의 유기적인 연동을 직접 구현하여, 전체 웹 서비스의 구조와 흐름을 완전히 이해하는 것에 의의를 두었습니다.

2. 핵심 목표 (데이터, API, 클라우드)

본 프로젝트는 다음의 기술적 목표를 중심으로 기획 및 제작되었습니다.

1) 효율적인 데이터 처리 및 관리 경험 :

관계형 데이터베이스(RDB)를 설계하고, Spring Data JPA 등을 활용하여 데이터의 생성, 조회, 수정, 삭제(CRUD) 로직을 구현함

2) RESTful API 설계 및 구현 숙련 :

프론트엔드와의 통신 표준이 되는 RESTful API를 직접 설계하고 구현하는 경험을 통해, 요청과 응답의 구조를 체계화하고 효율적인 데이터 통신 방식을 구현

3) 클라우드 환경(AWS) 서비스 배포 경험 :

개발한 백엔드 서비스를 AWS Elastic Beanstalk와 같은 클라우드 환경에 배포하여 실제 사용자에게 접근 가능한 서비스로 운영하는
것을 목표로 하고 이는 개발 환경부터 배포 환경까지 전체 라이프 사이클을 이해하는 중요한 학습 과정을 완성

4) 정적 웹사이트와 동적 웹사이트의 분리로 운용비용의 효율적인 관리

- 포트폴리오 (정적 콘텐츠) : Netlify의 무료호스팅 서비스에 배포
- 쇼핑몰 (동적 콘텐츠) : AWS Elastic Beanstalk에 배포

04

디자인 콘셉트

1. 개발 효율성과 기능 구현 우선

1) 디자인 목표 : 본 프로젝트는 백엔드(데이터 처리/API) 간의 유기적인 연동 및 클라우드 배포 경험을 핵심 목표로 하였기 때문에, 디자인 요소는 기능 구현의 효율성을 최우선으로 고려했습니다.

2) 간결한 UI/UX : 복잡한 시각적 효과나 고급 애니메이션보다는, 사용자(관리자 및 일반 이용자)가 핵심 기능(장바구니, 주문, 회원가입, 상품 등록 등)을 직관적으로 사용할 수 있도록 간결하고 명료한 UI/UX를 채택했습니다.

3) 프론트엔드 최소화 전략 : 기존 포트폴리오의 웹 디자인 및 React 프로젝트에서 충분히 보여준 프론트엔드 역량을 과도하게 반복하지 않고, Spring Boot의 서버 템플릿 엔진(Thymeleaf)을 활용하여 기능 구현에 필요한 최소한의 화면만 구성했습니다.

2. 디자인 구성

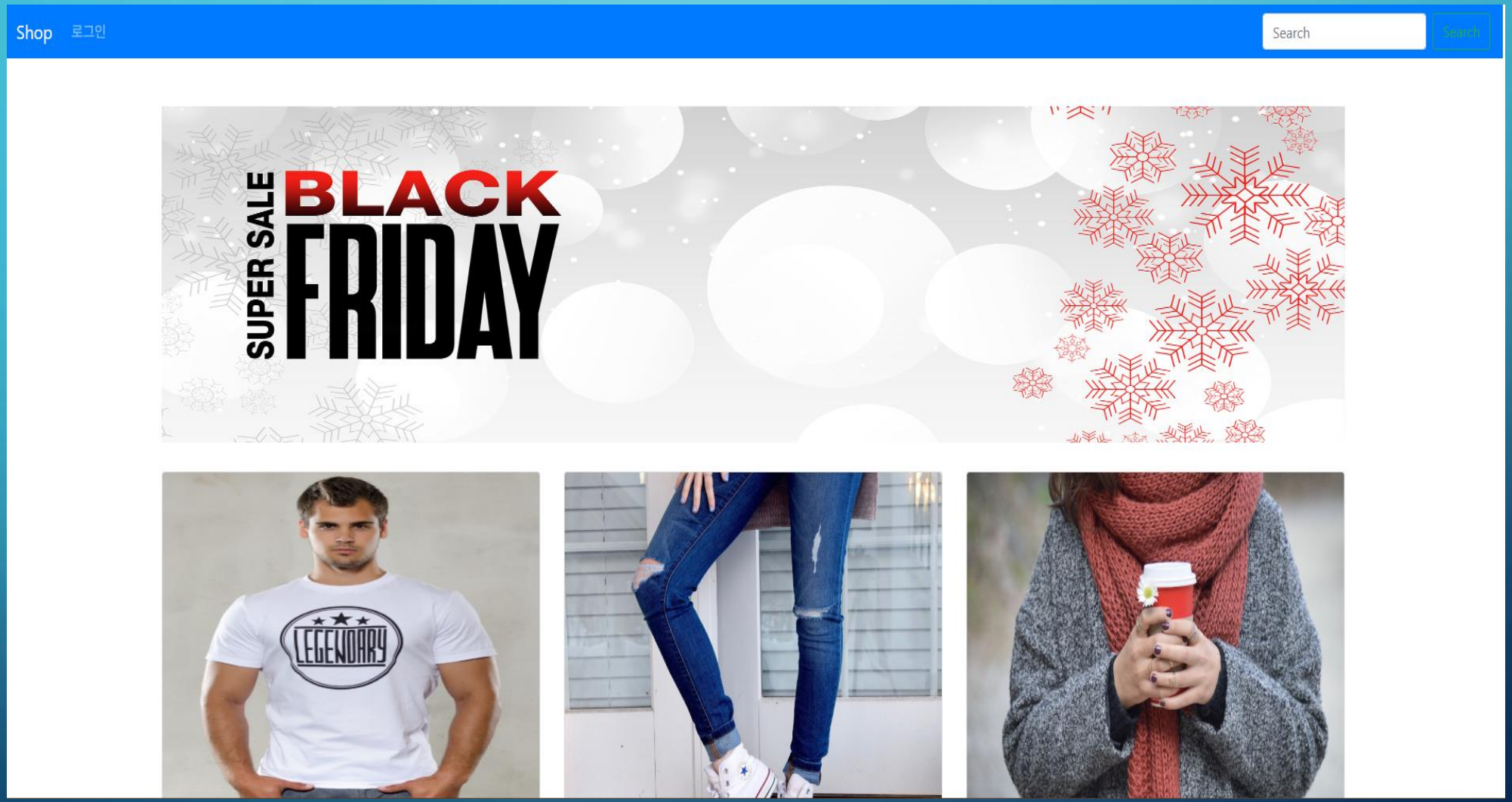
구분	내용
프론트엔드 마크업	HTML5, CSS3, JavaScript
템플릿 엔진	Thymeleaf를 사용하여 서버에서 화면을 동적으로 렌더링하도록 구성.
반응형 디자인	Bootstrap 등의 프레임워크를 활용하여 기본적인 반응형 레이아웃을 빠르게 구성함으로써, 디자인 리소스 투입을 최소화하고 개발 시간을 단축
시각적 특징	프로젝트의 목적(쇼핑몰)에 맞추어 상품 이미지와 정보 전달에 초점을 맞추었으며, 복잡한 커스터마이징 없이 표준적인 레이아웃을 활용하여 가독성을 높임

05

웹사이트 상세 설명 (메인 페이지 로직)

구분	설명	코드와의 연관성
URL 매핑	@GetMapping(value = "/")	이 어노테이션은 웹사이트의 **메인 페이지**가 로드되는 시작점
데이터 처리	itemService.getMainItemPage()	itemService를 호출하여 메인 페이지에 표시될 상품 리스트를 데이터베이스에서 조회
뷰 전달	return "main";	메소드가 최종적으로 **main이라는 이름의 뷰 템플릿**을 사용자에게 반환하도록 지시
의존성 주입	@RequiredArgsConstructor와 private final ItemService itemService;	ItemService 객체를 주입받아 사용함으로써, 상품 조회를 Controller와 분리하여 **Service 계층**에서 처리

1. Main 캡쳐 이미지



2. controller\MainController,java 캡쳐 이미지

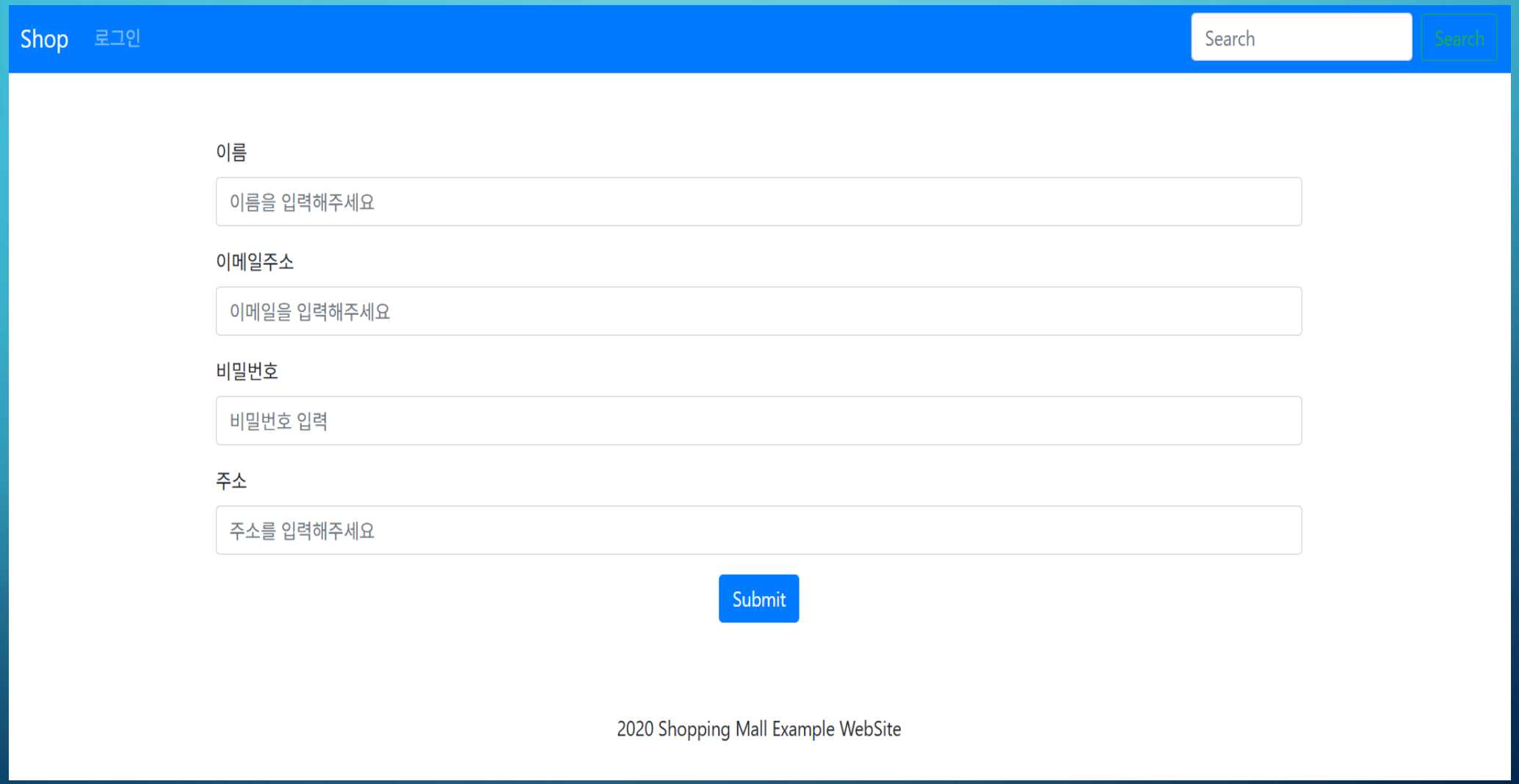
```java
package com.shop.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

import com.shop.dto.ItemSearchDto;
import com.shop.dto.MainItemDto;
import com.shop.service.ItemService;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.ui.Model;

import java.util.Optional;
no usages
@Controller
@RequiredArgsConstructor
public class MainController {

    private final ItemService itemService;

    @GetMapping(value = "/")
    public String main(ItemSearchDto itemSearchDto, Optional<Integer> page,
Model model){

        Pageable pageable = PageRequest.of(page.isPresent() ? page.get() : 0, pageSize: 6);
        Page<MainItemDto> items =
itemService.getMainItemPage(itemSearchDto, pageable);
        model.addAttribute( attributeName: "items", items);
        model.addAttribute( attributeName: "itemSearchDto", itemSearchDto);
        model.addAttribute( attributeName: "maxPage", attributeValue: 5);
        return "main";
    }
}
```

05

웹사이트 상세 설명 (회원관리 로직)

영역	내용 및 설명	증빙 자료
1. 회원가입	MemberController의 @GetMapping(value = "/new") 메소드는 **회원가입 폼을 사용자에게 제공**	
2. 입력값 검증 (Validation)	@PostMapping(value = "/new") 메소드에서 @Valid 및 BindingResult를 사용하여 사용자가 입력한 **데이터의 유효성을 검증** 에러가 발생하면 다시 폼으로 이동하여 사용자 경험을 개선	**MemberController code**
3. 중복 및 예외 처리	try-catch 구문을 통해 IllegalStateException 등 **중복 회원 가입 예외를 처리** 이는 비즈니스 로직에 대한 **방어적 프로그래밍**을 적용했음을 보여줌	MemberController code
4. 핵심 보안 기술	PasswordEncoder를 주입받아 **비밀번호를 암호화**하여 저장.	MemberController code
5. Entity/DTO 활용	MemberFormDto를 통해 데이터를 받고, Member Entity로 변환하여 memberService.saveMember()를 호출 이는 **DTO, Entity, Service 계층을 분리**하는 MVC 패턴의 모범 사례를 따랐음을 증명	MemberController code

1. 회원가입 캡쳐 이미지


Shop
로그인
Search
Search
이름
이름을 입력해주세요
이메일주소
이메일을 입력해주세요
비밀번호
비밀번호 입력
주소
주소를 입력해주세요
Submit
2020 Shopping Mall Example WebSite

2. controller\MemberController,java 캡쳐 이미지

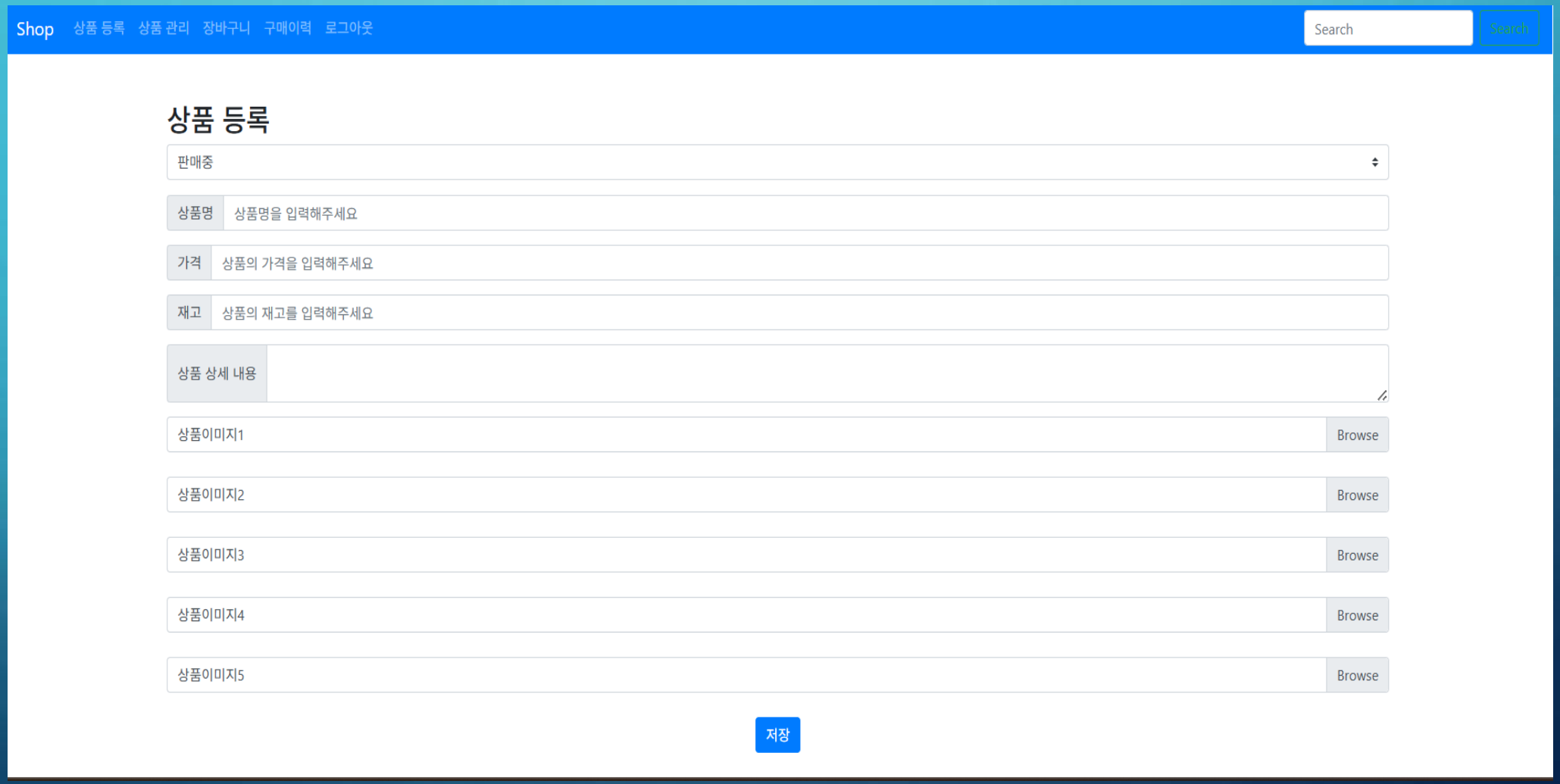
```java
package com.shop.controller;

import com.shop.dto.MemberFormDto;
import com.shop.service.MemberService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import com.shop.entity.Member;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.validation.BindingResult;
import jakarta.validation.Valid;

no usages
@RequestMapping("/members")
@Controller
@RequiredArgsConstructor
public class MemberController {

    private final MemberService memberService;
    private final PasswordEncoder passwordEncoder;
```

05

웹사이트 상세 설명 (상품 관리 및 CRUD 구현)

영역	내용 및 설명	증빙 자료
1. 상품 등록 기능 구현	@GetMapping ()이 폼을 제공하고, @PostMapping()가 데이터를 받아 **상품 저장 로직**을 실행	ItemController code
2. 복수 이미지 업로드 처리	상품 등록시, @RequestParam() List<MultipartFile> itemImgFileList를 통해 **여러 개의 이미지 파일을 동시에** 전달받아 처리하는 로직을 구현	ItemController code
3. 데이터 검증 및 필수 값 처리	itemNew 메소드 내부에서 @Valid와 BindingResult를 사용하여 입력값 **유효성 검증**을 수행	ItemController code
4. 상품 조회 및 수정 로직	@GetMapping()를 통해 상품 상세 정보(itemDtl)를 조회하며, itemUpdate 메소드를 통해 수정 기능을 구현	ItemController code
5. 예외 및 에러 처리	EntityNotFoundException과 같은 **예외를 처리**.	ItemController code
6. 상품 관리 페이징	@GetMapping(value = {"/admin/items", "/admin/items/{page}"})를 통해 **관리자용 상품 목록** 조회 시에도 PageRequest.of()를 사용하여 **페이징 처리**를 적용	ItemController code
7. DTO/Entity 활용	ItemFormDto와 ItemImgDto를 활용하여 상품 정보와 이미지 정보를 구조화하고, Item Entity로 매핑하는 과정을 통해 **객체지향적인 데이터 관리**를 수행	

1. 상거래 서비스 캡쳐 이미지



2. controller\ItemController,java 캡쳐 이미지

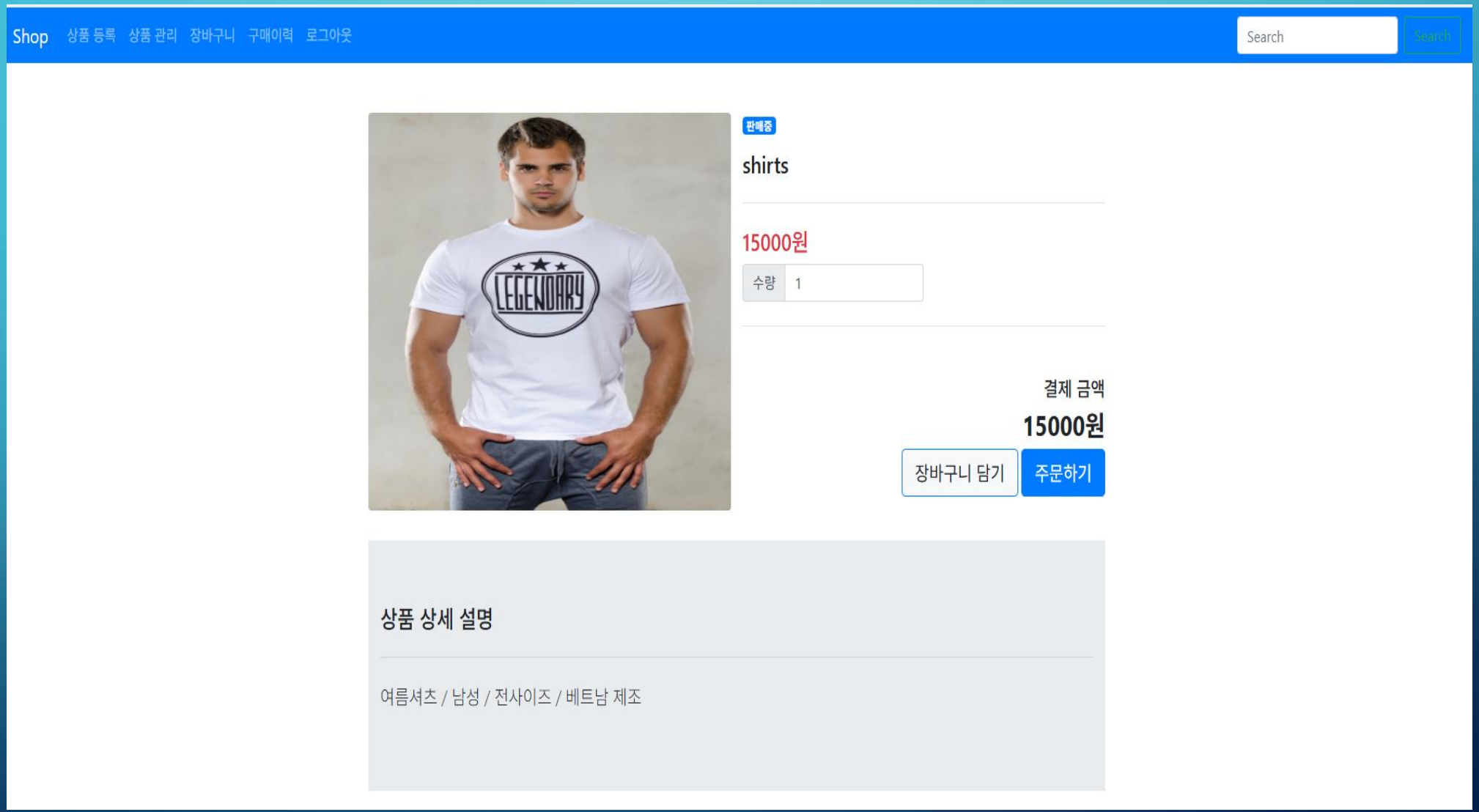
```java
package com.shop.controller;

import com.shop.dto.MainItemDto;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.ui.Model;
import com.shop.dto.ItemFormDto;

import com.shop.service.ItemService;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.PostMapping;
import jakarta.validation.Valid;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
import java.util.List;

import org.springframework.web.bind.annotation.PathVariable;
import jakarta.persistence.EntityNotFoundException;

import com.shop.dto.ItemSearchDto;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import java.util.Optional;

// 상품등록 페이지에 접근할 수 있도록 ItemController 클래스 작성
no usages
@Controller
@RequiredArgsConstructor
public class ItemController {

    private final ItemService itemService;

    @GetMapping(value = "/admin/item/new")
    public String itemForm(Model model) {
        model.addAttribute( attributeName: "itemFormDto" , new ItemFormDto());
        return "item/itemForm";
    }
```

05

웹사이트 상세 설명 (View 렌더링 방식)

영역	내용 및 설명	증빙 자료
1. 뷰 엔진 사용 증명	ThymeleafExController는 Spring Boot에 통합된 **Thymeleaf 템플릿 엔진**의 다양한 기능을 학습하고 구현하기 위해 작성	ThymeleafExController code
2. 뷰 로직 처리 방식	model.addAttribute()와 같이 **컨트롤러에서 데이터를 모델에 담아**, return "thymeleafEx/thymeleafEx01";와 같은 문자열을 반환하여 **templates 폴더** 내의 뷰 렌더링	ThymeleafExController code
3 . 동적 데이터 바인딩	**Controller에서 생성된 객체(ItemDto)** 또는 리스트(itemDtoList)를 뷰로 전달하고, **Thymeleaf 문법**을 사용하여 반복문(for loop) 등으로 화면에 출력하는 **동적 바인딩**을 구현	ThymeleafExController code
4. URL 파라미터 처리	@GetMapping 시 전달되는 URL 파라미터(param1, param2)를 **자동으로 바인딩**하여 모델에 담는 기능을 구현	ThymeleafExController code
5. 상품 상세 페이지 적용	**상품 상세 페이지**는 실제 코드에서는 ItemController가 상품 데이터를 조회한 뒤 (itemService.getItemDtl), Thymeleaf 템플릿(itemDtl.html 등)을 통해 해당 데이터를 받아 **동적으로 화면을 구성**	ItemController code
6. 풀스택 구현 역량 입증	**프론트엔드와 백엔드를 긴밀하게 결합**하여 서버 측에서 모든 뷰를 제어	

1. 상품상세 서비스 캡쳐 이미지



2. controller\ThymeleafExController.java 캡쳐 이미지

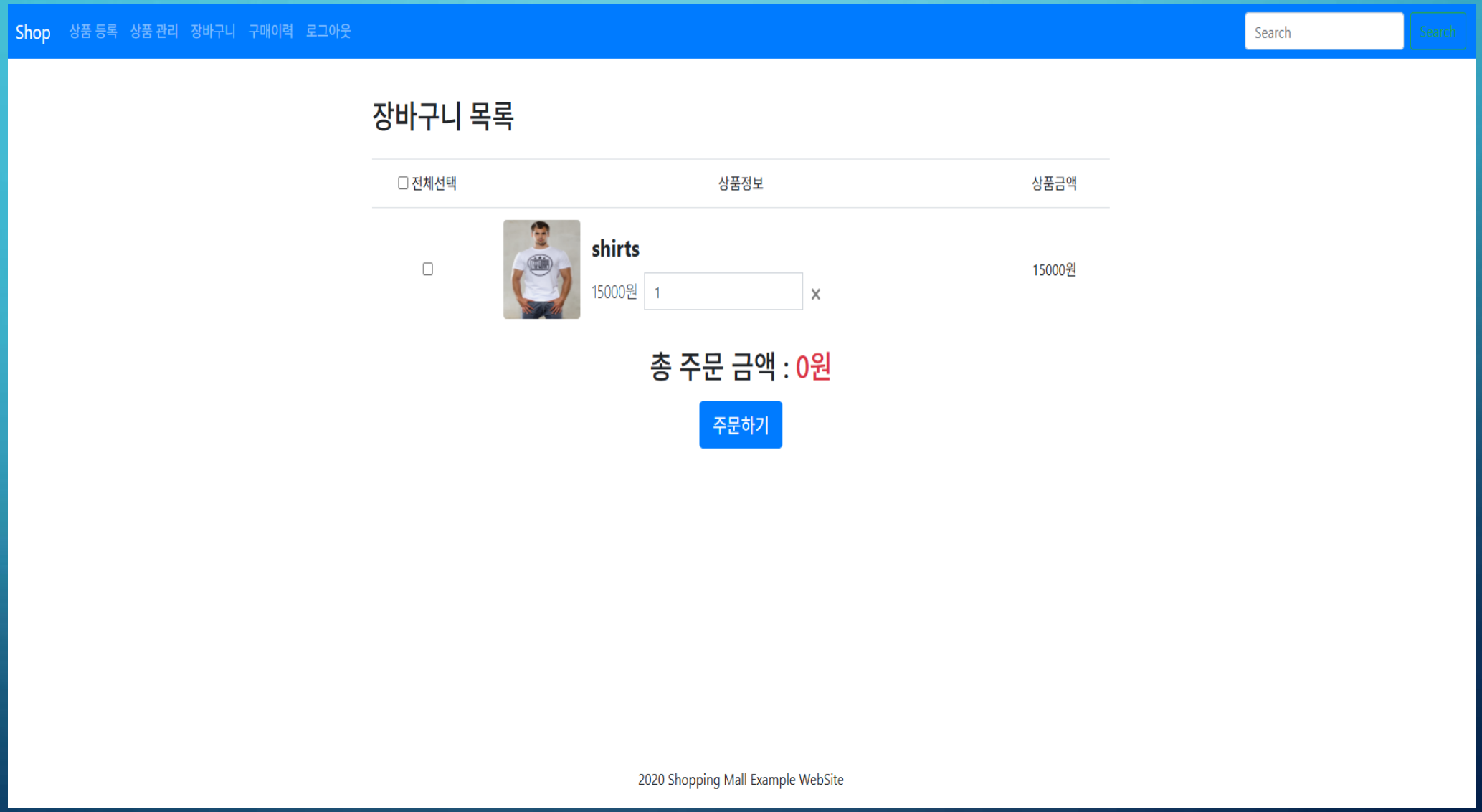
```java
package com.shop.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import com.shop.dto.ItemDto;
import java.time.LocalDateTime;

import java.util.ArrayList;
import java.util.List;

no usages
@Controller
@RequestMapping(value="/thymeleaf")
// 클라이언트의 요청에 대해서 어떤 컨트롤러가 처리할지 매핑하는 어노테이션이다.
// url에 "/thymeleaf" 경로로 오는 요청을 ThymeleafExController가 처리하도록 한다.
public class ThymeleafExController {

    no usages
    @GetMapping(value = "/ex01")
    public String thymeleafExample01(Model model) {
        model.addAttribute( attributeName: "data", attributeValue: "타임리프 예제 입니다.");
        // model 객체를 이용해 뷰에 전달한 데이터를 key, value 구조로 넣어 준다.
        return "thymeleafEx/thymeleafEx01";
        // templates 폴더를 기준으로 뷰의 위치와 이름(thymelealEx01.html)을 반환
    }

    no usages
    @GetMapping(value = "/ex02")
    public String thymeleafExample02(Model model){
        ItemDto itemDto = new ItemDto();
        itemDto.setItemDetail("상품 상세 설명");
        itemDto.setItemNm("테스트 상품1");
        itemDto.setPrice(10000);
        itemDto.setRegTime(LocalDateTime.now());

        model.addAttribute( attributeName: "itemDto", itemDto);
        return "thymeleafEx/thymeleafEx02";
    }
```

05

웹사이트 상세 설명 (장바구니 및 주문 로직)

영역	내용 및 설명	증빙 자료
1. 장바구니 상품 추가 (POST)	@PostMapping(value = "/cart") 메소드를 호출. 이 메소드는 @RequestBody를 통해 **비동기(AJAX)** 방식으로 상품 정보(CartItemDto)를 전달받아 처리	CartController code
2. RESTful API 활용	상품 수량 변경에 @PatchMapping을, 상품 삭제에 @DeleteMapping을 사용하여 **RESTful API 설계 원칙**을 준수.이는 자원에 대한 행위를 HTTP 메서드에 정확히 매핑하는 **백엔드 설계.**	CartController code
3. 비동기 응답 처리	장바구니 추가(order), 수량 변경(updateCartItem), 삭제(deleteCartItem) 등 모든 API 요청에 대해 @ResponseBody ResponseEntity를 반환하여, **JSON 형태의 데이터**와 함께 HTTP 응답 상태 코드(Status Code)를 클라이언트에게 전송. 이는 비동기 통신(AJAX)의 핵심 로직	CartController code
4. 장바구니 목록 조회	principal.getName()을 통해 **현재 로그인된 사용자**의 정보를 받아, 해당 사용자의 장바구니 목록(CartDetailDto)을 조회	CartController code
5. 권한 및 예외 처리	validateCartItem 서비스를 호출하여 **수정 및 주문 권한을 체크**하며, if(count <= 0) 등 비즈니스 규칙에 따른 에러 발생 시 HttpStatus.BAD_REQUEST나 HttpStatus.FORBIDDEN을 반환하여 **견고한 서버 응답**을 구현	CartController code
6. 장바구니 상품 주문 로직	@PostMapping(value = "/cart/orders")를 통해 장바구니 상품을 주문(orderCartItem)으로 전환하는 최종 로직을 구현	CartController code

1. 장바구니 및 주문 서비스 캡쳐 이미지



2. controller\CartController.java 캡쳐 이미지

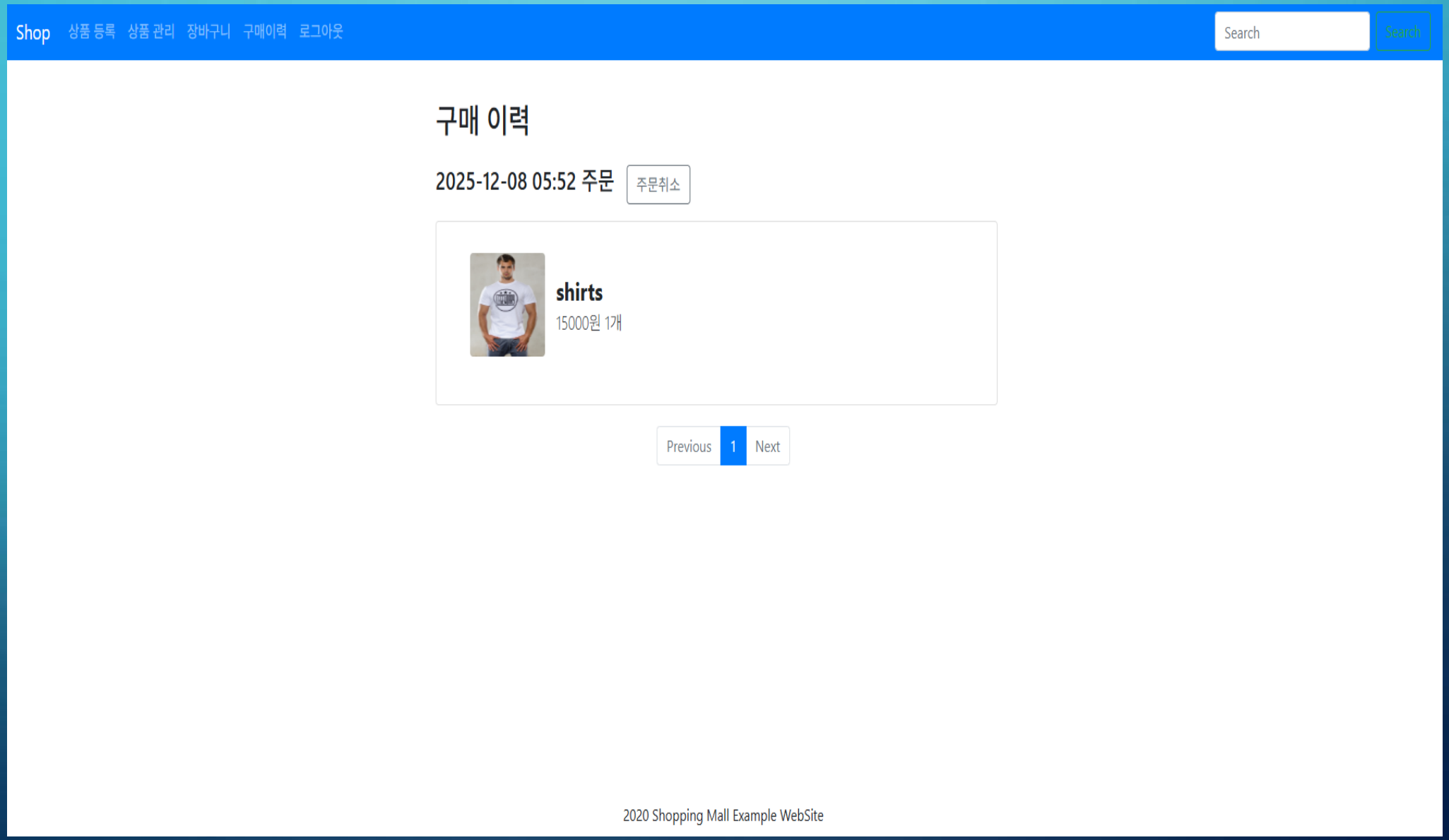
```java
no usages
@Controller
@RequiredArgsConstructor
public class CartController {

    private final CartService cartService;

    @PostMapping(value = "/cart")
    public @ResponseBody ResponseEntity order(@RequestBody @Valid CartItemDto cartItemDto,
                                              BindingResult bindingResult, Principal principal){

        if(bindingResult.hasErrors()){
            // 장바구니에 담을 상품 정보를 받는 cartItemDto 객체에 데이터 바인딩 시
            // 에러가 있는지 검사한다.
            StringBuilder sb = new StringBuilder();
            List<FieldError> fieldErrors = bindingResult.getFieldErrors();

            for (FieldError fieldError : fieldErrors) {
                sb.append(fieldError.getDefaultMessage());
            }

            return new ResponseEntity<String>(sb.toString(), HttpStatus.BAD_REQUEST);
        }

        String email = principal.getName();
        // 현재 로그인한 회원의 이메일 정보를 변수에 저장한다.
        Long cartItemId;

        try {
            cartItemId = cartService.addCart(cartItemDto, email);
            // 화면으로부터 넘어온 장바구니에 담을 상품 정보와 현재 로그인한 회원의 이메일
            // 정보를 이용하여 장바구니에 상품을 담는 로직을 호출한다.
        } catch(Exception e){
            return new ResponseEntity<String>(e.getMessage(), HttpStatus.BAD_REQUEST);
        }

        return new ResponseEntity<Long>(cartItemId, HttpStatus.OK);
        // 결과값으로 생성된 장바구니 상품 아이디와 요청이 성공하였다는 HTTP 응답 상태 코드를
        // 반환한다.
    }
```

05

웹사이트 상세 설명 (주문 및 거래 로직)

영역	내용 및 설명	증빙 자료
1. 비동기 주문 생성 (POST)	장바구니에서 '주문하기' 버튼 클릭 시, @PostMapping("/o@RequestBody와 @ResponseBody를 통해 비동기(AJAX)로 주문 정보(OrderDto)를 받아 처리하고, 주문 성공 시 HttpStatus.OK를 반환.	OrderController code
2. 사용자 인증 및 로직 분리	주문 생성 시 Principal 객체를 통해 **현재 로그인한 사용자의 이메일**을 조회. 이 정보를 orderService.order()에 전달하여 **회원별 주문 로직**을 실행. Controller는 데이터 수신/응답만 담당하고, 비즈니스 로직은 **Service 계층**(OrderService)에 위임.	OrderController code
3. 주문 기록 조회 및 페이징	@GetMapping(value = {"/orders", "/orders/{page}"})를 통해 **구매 이력**을 제공. PageRequest.of(...)를 사용하여 **주문 기록에도 페이징**을 적용하여, 대량의 거래 기록도 효율적으로 관리할 수 있도록 구현.	OrderController code
4. 주문 취소 기능 및 권한 검사	주문 기록 화면에서 '주문 취소' 버튼 클릭 시 @PostMapping("/order/{orderId}/cancel")를 호출. 이때 orderService.validateOrder()를 통해 **다른 사용자의 주문을 취소하지 못하도록 주문 취소 권한 검사**를 철저히 수행.	OrderController code
5. DTO 및 Entity 활용	OrderDto, OrderHistDto와 같은 DTO를 사용하여 계층 간 데이터 전달을 구조화하고, Order 및 OrderItem Entity를 통해 **트랜잭션 관계**를 명확하게 관리.	
6. 서버 응답 및 에러 처리	주문 로직 실행 중 발생하는 예외는 try-catch 구문으로 처리하며, 클라이언트에게 HttpStatus.BAD_REQUEST와 함께 **명확한 에러 메시지**를 반환.	OrderController code

1. 주문 생성, 이력 관리 및 취소 로직 서비스 캡쳐 이미지



2. controller\OrderController.java 캡쳐 이미지

```java
@Controller
@RequiredArgsConstructor
public class OrderController {

    private final OrderService orderService;

    @PostMapping(value = "/order")
    public @ResponseBody ResponseEntity order(@RequestBody @Valid OrderDto orderDto
            , BindingResult bindingResult, Principal principal){
        // 스프링에서 비동기 처리를 할 때 @RequestBody와 @ReponseBody 어노테이션을 사용한다.

        if(bindingResult.hasErrors()){
            //주문 정보를 받는 orderDto 객체에 데이터 바인딩 에러가 있는지 검사한다.
            StringBuilder sb = new StringBuilder();
            List<FieldError> fieldErrors = bindingResult.getFieldErrors();

            for (FieldError fieldError : fieldErrors) {
                sb.append(fieldError.getDefaultMessage());
            }

            return new ResponseEntity<String>(sb.toString(),
                    HttpStatus.BAD_REQUEST); // 에러 정보를 ResponseEntity
            // 객체에 담아서 반환한다.
        }

        String email = principal.getName(); // 현재 로그인 유저의 정보를 얻기 위해서
        // @Controller 어노테이션의 선언된 클래스에서 메소드 인자로 principal 객체를
        // 넘겨 줄 경우 해당 객체에 직접 접근할 수 있다. principal 객체에서 현재 로그인한
        // 회원의 이메일 정보를 조회한다.
        Long orderId;

        try {
            orderId = orderService.order(orderDto, email);
            // 화면으로부터 넘어오는 주문 정보와 회원의 이메일 정보를 이용하여 주문 로직을 호출한다.
        } catch(Exception e){
            return new ResponseEntity<String>(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
        return new ResponseEntity<Long>(orderId, HttpStatus.OK);
        // 결과값으로 생성된 주문 번호와 요청이 성공했다는 HTTP 응답 상태 코드를 반환한다.
    }
```

06

배포 및 환경 구성

영역	강조할 핵심 아이디어	시각 자료 추천
1. 환경 분리 전략	**개발(local) 환경과 운영(prod) 환경을 명확히 분리**했음을 강조. 이는 실무에서 필수적인 요소	application-*.properties
2. AWS 배포 과정	**AWS Elastic Beanstalk**를 사용한 **자동 배포(CI/CD)** 경험을 강조. JAR 파일 생성부터 클라우드에 배포되어 서비스가 실행되는 과정을 설명.	
3. 데이터베이스 구성	**DB 환경을 분리**했음을 강조. 로컬 개발 시에는 H2/MySQL 로컬 환경을 사용하고, 운영 환경에서는 AWS RDS(관계형 데이터베이스 서비스)를 연결.	
4. 보안 및 환경 설정	application-prod.properties 파일 등에 **민감 정보**를 따로 관리하고, 환경 변수를 통해 DB 접속 정보를 주입받는 등의 **보안적 접근**을 간략하게 언급	application-*.properties

1. 클라우드 기반 서비스 배포 (AWS Elastic Beanstalk)
CI/CD : 개발 환경에서 작성된 코드를 AWS Elastic Beanstalk를 통해 클라우드 환경에 배포하고, 서비스가 실제 URL(http://springboot-leekunwoo-env.eba-8vnf42tm.ap-northeast-2.elasticbeanstalk.com/)에서 구동되는
전체 CI/CD 라이프사이클을 구현.

목표 : 서버 관리의 복잡성을 줄이고, 애플리케이션의 확장성과 안정성을 클라우드 인프라에 의존하여 확보.

2. 환경 분리 및 설정 관리
Properties 파일 분리: application-local.properties, application-prod.properties, application-test.properties와 같이 환경별 설정 파일()을 분리하여 관리

목적: 로컬 개발 시에는 개발용 DB를 사용하고, 운영 환경에서는 별도의 DB(AWS RDS)에 연결되도록 설정하여,
환경에 따라 유연하게 대응하고 민감 정보 노출을 방지.

3. 보안 환경 구성
운영 DB 활용: 운영 환경에서는 PostgreSQL 등 관계형 데이터베이스를 AWS RDS로 구성하여 백엔드 서비스(Elastic Beanstalk)와 연결.

결과: 이는 영구적인 데이터 저장소를 안정적으로 관리하고, 클라우드 서비스 간의 연동 역량을 입증.

1. application.properties 코드

```
spring.datasource.url=${DB_URL:jdbc:postgresql://localhost:5432/shop}
spring.datasource.username=${DB_USERNAME:postgres}

spring.datasource.password=${DB_PASSWORD}
server.port=5000

spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

logging.level.org.hibernate.SQL=debug
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=trace

spring.servlet.multipart.max-file-size=20MB
spring.servlet.multipart.max-request-size=20MB

itemImgLocation=${ITEM_IMG_LOCATION:C:/shop/item}
uploadPath=${UPLOAD_PATH:file:///C:/shop/}
```

2. application-local.properties 코드

```
spring.datasource.url=jdbc:postgresql://blog-database.ct6q0uqmchhf.ap-northeast-2.rds.amazonaws.com:5432/shop
spring.datasource.username=postgres
spring.datasource.password=new_password

itemImgLocation=C:/shop/item
uploadPath=file:///C:/shop/
```

3. application-prod.properties 코드

```
#AWS
spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
server.port=5000

spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

logging.level.org.hibernate.SQL=debug

spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
spring.thymeleaf.cache=false

server.tomcat.uri-encoding=UTF-8
spring.http.encoding.charset=UTF-8
spring.http.encoding.enabled=true
spring.http.encoding.force=true
#AWS
itemImgLocation=${ITEM_IMG_LOCATION:/tmp/shop/item}
uploadPath=${UPLOAD_PATH:file:///tmp/SHOP}
```

4. application-test.properties 코드

```
# PostgreSQL
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:test
spring.datasource.username=sa
spring.datasource.password=

# JPA
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# Thymeleaf
spring.thymeleaf.cache=false
```

07 기술적 도전 및 해결 과정

1. 동시성 문제 해결: 재고 관리의 안정성 확보

구분	내용	증빙 자료
도전 과제	여러 명의 사용자가 동일한 상품을 동시에 주문할 경우, **재고 차감 로직**에서 데이터의 **동시성(Concurrency) 문제**가 발생하여 재고 수량이 음수(-)가 되는 등 데이터 불일치(Inconsistency) 오류가 발생할 수 있다.	OutOfStockException
해결 전략	**트랜잭션(Transaction) 관리**를 통해 문제 발생을 방지. 주문(Order) 생성 및 재고 차감 과정 전체를 하나의 원자적(Atomic) 작업으로 묶어, 해당 트랜잭션이 **성공 또는 실패** 중 하나로만 완료되도록 보장.	service 계층에서 @Transactional 어노테이션 활용
결과	OutOfStockException과 같은 예외 처리 코드를 명시적으로 구현하여, 재고가 부족할 경우 **주문 요청을 거부**하고 사용자에게 정확한 메시지를 전달하는 **안정적인 재고 관리 시스템**을 구축.	repository, exception 패키지

2. 유지보수성 향상: 이미지 파일 관리의 효율화

구분	내용	증빙 자료
도전 과제	상품 이미지 파일의 **업로드, 저장, 수정, 삭제**와 같은 I/O(Input/Output) 작업을 컨트롤러에 직접 구현할 경우, ItemController의 코드가 복잡해지고 **이미지 처리 로직**의 재활용이 불가능해져 유지보수가 어려워진다.	ItemController 코드 복잡성
해결 전략	이미지 처리와 관련된 모든 비즈니스 로직을 FileService와 ItemImgService로 **분리.**	service 패키지 (FileService)
결과	I temController는 오직 **사용자 요청 처리**와 **Service 호출** 역할만 수행하며, 이미지 관련 로직은 FileService에서 담당하여 코드의 **관심사 분리(Separation of Concerns)** 원칙과 이를 통해 코드의 가독성 및 재사용성이 크게 향상.	ItemController의 Service 호출 구조

3. service\FileService.java 캡쳐 이미지

```java
package com.shop.service;

import lombok.extern.java.Log;
import org.springframework.stereotype.Service;
import java.io.File;
import java.io.FileOutputStream;
import java.util.UUID;

1 usage
@Service
@Log
public class FileService {

    2 usages
    public String uploadFile(String uploadPath, String originalFileName, byte[] fileData) throws Exception{
        UUID uuid = UUID.randomUUID();
        // UUID(Universally Unique Identifier)는 서로 다른 개체들을 구별하기 위해서 이름을
        // 부여할 때 사용한다. 실제 사용 시 중복될 가능성이 거의 없기 때문에 파일의 이름으로 사용하면
        // 파일명 중복 문제를 해결 할 수 있다.
        String extension = originalFileName.substring(originalFileName.lastIndexOf( str: "."));
        String savedFileName = uuid.toString() + extension;
        //UUID로 받은 값과 원래 파일의 이름의 확장자를 조합에서 저장될 파일 이름을 만든다.
        String fileUploadFullUrl = uploadPath + "/" + savedFileName;
        FileOutputStream fos = new FileOutputStream(fileUploadFullUrl);
        // FileOutputStream 클래스는 바이트 단위의 출력을 내보내는 클래스이다. 생성자로 파일이
        // 저장될 위치와 파일의 이름을 넘겨 파일에 쓸 파일 출력 스트림을 만든다.
        fos.write(fileData);
        //fileData를 파일 출력 스트림에 입력한다.
        fos.close();
        return savedFileName;
        //업로드된 파일의 이름을 반환한다.
    }
```

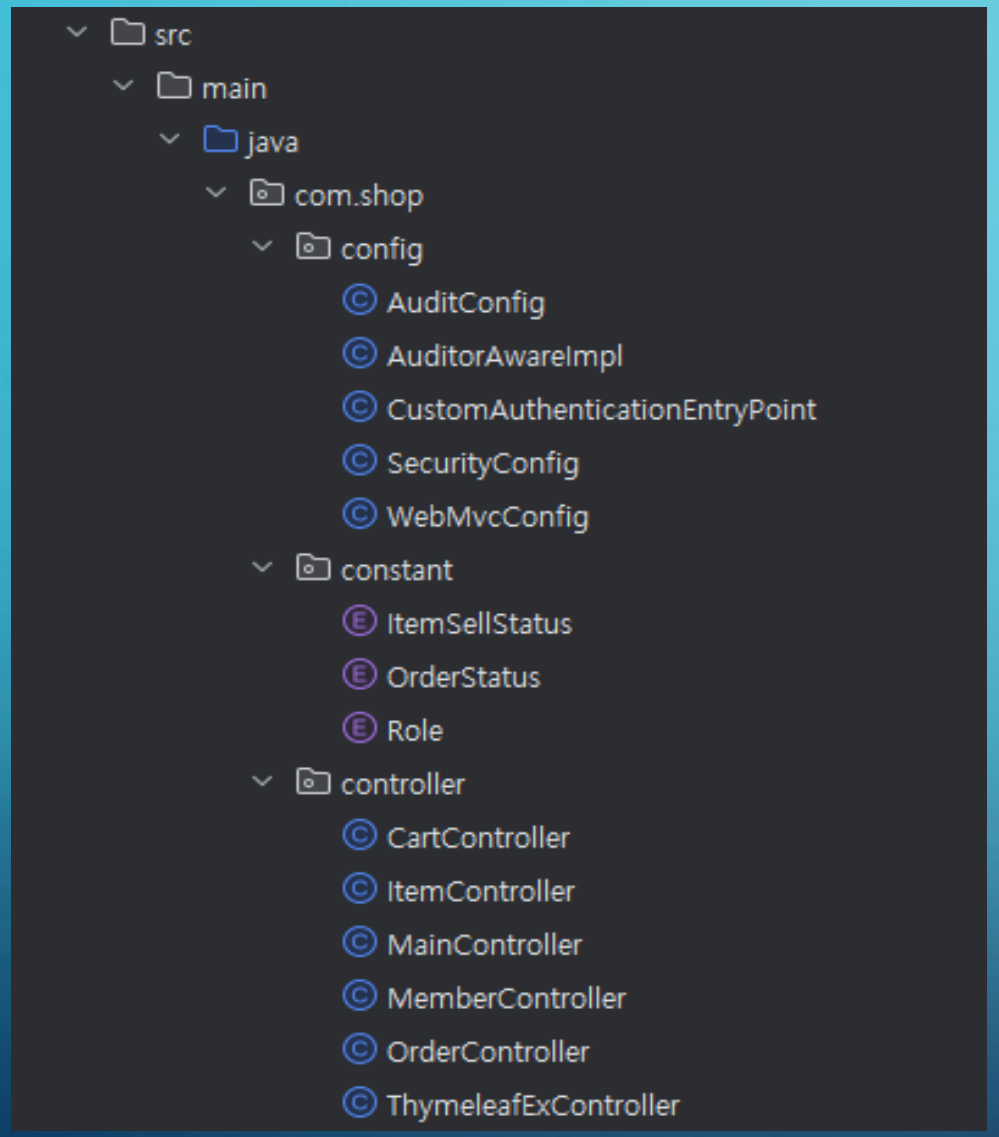
08

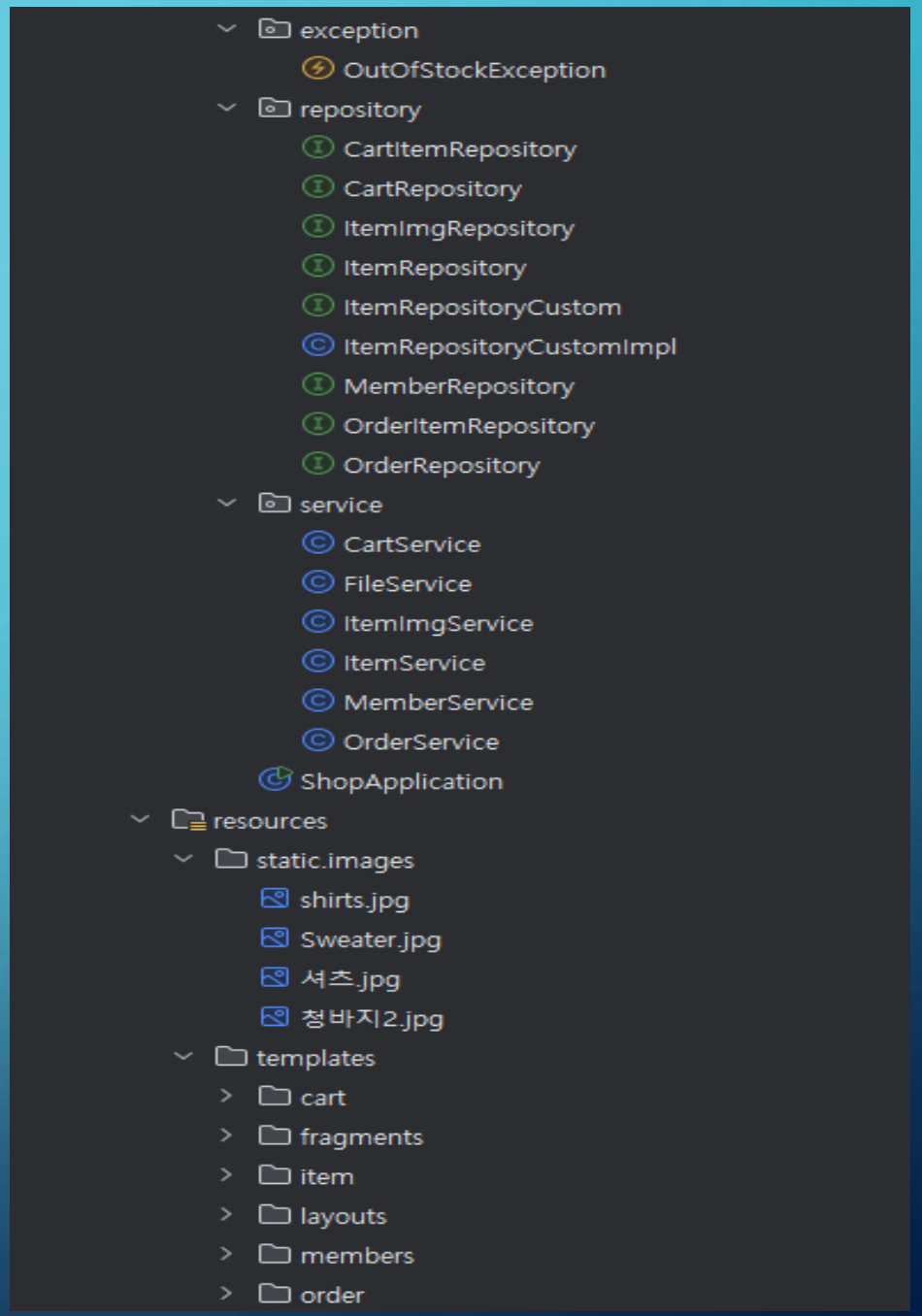
프로젝트 요약 및 기여도

1. 프로젝트 성과 및 핵심 기여 기술

영역	핵심 기여 내용	증빙
백엔드 아키텍처	**Spring Boot MVC 기반의 계층형 아키텍처**를 설계하고 구현. Controller-Service-Repository의 명확한 역할 분리로 **유지보수성이 높은** 코드를 작성.	코드 구조 이미지
거래 로직 완성	**장바구니, 주문, 주문 취소** 등 실제 상거래에 필요한 **핵심 비즈니스 로직**을 구현. @Valid를 활용한 **데이터 검증**과 **예외 처리**를 적용하여 서비스의 안정성을 확보.	CartController code, OrderController code
보안 및 인증	**Spring Security**를 도입하여 **회원가입/로그인, 비밀번호 암호화, 권한(Role) 관리** 등 필수적인 보안 기능을 직접 구현.	MemberController code
풀스택 환경 구축	**Thymeleaf**를 활용한 **서버 주도적 뷰(View) 렌더링**을 경험하며, 프론트엔드와 백엔드를 통합적으로 이해하는 **풀스택 개발 능력**을 입증.	ThymeleafExController code

2. 증빙자료 : 코드-루트 캡쳐 이미지





2. 개인적 기여 및 성장 경험

실무적 경험 습득: 로컬 환경에서의 개발부터 AWS Elastic Beanstalk를 활용한 클라우드 배포까지, 실제 서비스 개발 과정을 Full-Cycle로 경험하며 서비스 운영 역량을 길렀습니다.

문제 해결 능력 : 재고 동시성 문제 해결을 위한 트랜잭션 관리와 이미지 관리 로직 분리를 통해, 단순히 기능을 구현하는 것을 넘어 시스템의 안정성과 효율성을 개선하는 능력을 체득했습니다.

객체지향 설계: DTO와 Entity를 명확히 분리하고, 핵심 클래스들을 구조화함으로써 객체지향 설계 원칙을 실제 프로젝트에 적용하는 연습을 했습니다.

3. 향후 목표 및 비전

테스트 코드 작성 : 현재 구현된 로직의 안정성을 더욱 높이기 위해, 향후 JUnit 및 Mockito를 활용한 자동화된 테스트 코드를 추가하여 테스트 커버리지를 확보할 계획입니다.

결제 시스템 연동 : 실제 상업 서비스를 목표로 간편 결제 모듈(예: 카카오페이, 토스페이) 연동을 시도하여, 트랜잭션 복잡성을 다루는 심화 학습에 도전할 예정입니다.

읽어 주셔서

대단히 감사드립니다.!!!