



YouTube

NAVER 카페

구멍가게 코딩단

# 코드로 배우는 리액트

## 5. 상품 API 서버 구성하기

# 5장. 상품 API 서버 구성하기

- 복잡한 상품 데이터 처리 예제
- 상품 데이터에 여러 첨부 파일 포함
- 썸네일 이미지 생성 및 데이터 구조 변경.

## 개발목표

1. 파일 업로드와 다운 로드 처리
2. 하나의 상품에 여러 이미지를 가지는 데이터의 처리
3. 첨부 파일이 있는 API의 개발과 테스트

# Index

5.1 파일 업로드를 위한 설정

5.2 컨트롤러에서의 파일 처리

5.3 엔티티의 처리

5.4 서비스 계층과 컨트롤러 연동

## Spring boot 프로젝트 설정

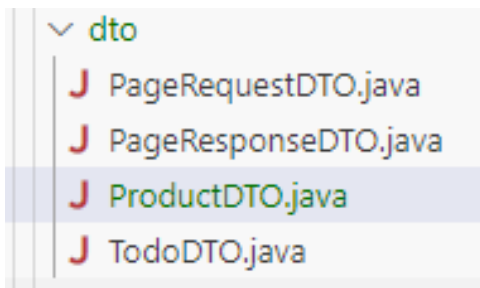
→ application.properties



```
spring.servlet.multipart.max-request-size=30MB  
spring.servlet.multipart.max-file-size=10MB  
  
org.zerock.upload.path=upload
```

## 상품 정보 처리를 위한 DTO

→ ProductDTO.java



```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class ProductDTO {

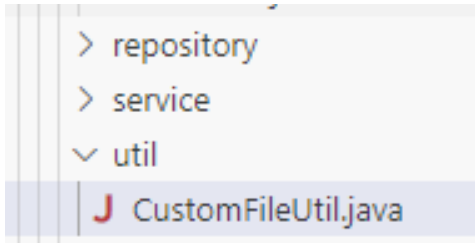
    private Long pno;
    private String pname;
    private int price;
    private String pdesc;
    private boolean delFlag;

    @Builder.Default
    private List<MultipartFile> files = new ArrayList<>();

    @Builder.Default
    private List<String> uploadFileNames = new ArrayList<>();
}
```

## 컨트롤러에서의 파일 처리

→ util 패키지를 구성 > CustomFileUtil 클래스를 추가



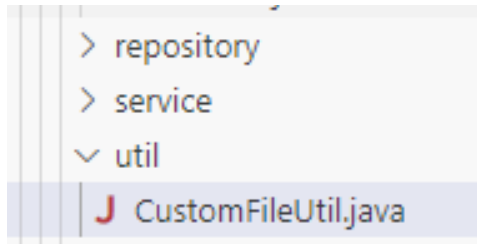
```
@Component
@Log4j2
@RequiredArgsConstructor
public class CustomFileUtil {

    @Value("${org.zerock.upload.path}")
    private String uploadPath;

    @PostConstruct
    public void init() {
        File tempFolder = new File(uploadPath);
        if(tempFolder.exists() == false) {
            tempFolder.mkdir();
        }
        uploadPath = tempFolder.getAbsolutePath();
        log.info("-----");
        log.info(uploadPath);
    }
}
```

## 컨트롤러에서의 파일 처리

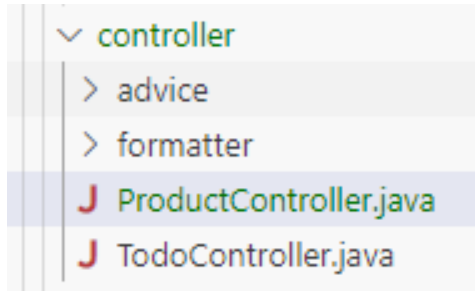
→ util 패키지를 구성 > CustomFileUtil 클래스를 추가



```
public List<String> saveFiles(List<MultipartFile> files) throws RuntimeException{
    if(files == null || files.size() == 0){
        return null;
    }
    List<String> uploadNames = new ArrayList<>();
    for (MultipartFile multipartFile : files) {
        String savedName = UUID.randomUUID().toString() + "_"
            + multipartFile.getOriginalFilename();
        Path savePath = Paths.get(uploadPath, savedName);
        try {
            Files.copy(multipartFile.getInputStream(), savePath);
            uploadNames.add(savedName);
        } catch (IOException e) {
            throw new RuntimeException(e.getMessage());
        }
    }
    return uploadNames;
}
```

## 컨트롤러에서의 파일 처리

→ ProductController.java



```
@RestController
@RequiredArgsConstructor
@Log4j2
@RequestMapping("/api/products")
public class ProductController {

    private final CustomFileUtil fileUtil;

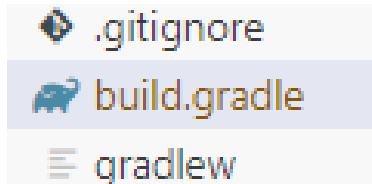
    @PostMapping("/")
    public Map<String, String> register(ProductDTO productDTO){

        log.info("register: " + productDTO);
        List<MultipartFile> files = productDTO.GetFiles();
        List<String> uploadFileNames = fileUtil.saveFiles(files);
        productDTO.setUploadFileNames(uploadFileNames);
        log.info(uploadFileNames);
        return Map.of("RESULT", "SUCCESS");
    }
}
```



## 썸네일 이미지 처리

→ build.gradle에 Thumbnailator라이브러리 추가

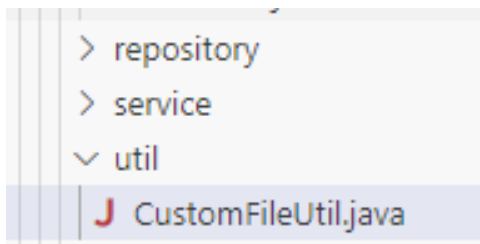


```
dependencies {  
    ... .생략  
    implementation 'net.coobird:thumbnailator:0.4.19'  
}
```

→ 라이브러리를 추가한 후에는 'Clean Java Server Workspaces'를 실행

## 썸네일 이미지 처리

→ CustomFileUtil 클래스



```
package org.zerock.mallapi.util;

...생략

import net.coobird.thumbnailator.Thumbnails;

@Component
public class CustomFileUtil {

    ...생략

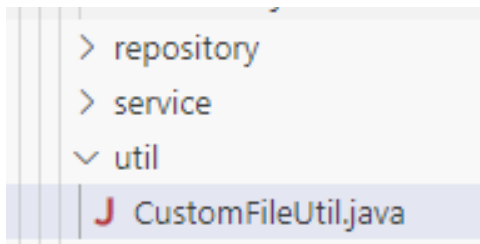
    public List<String> saveFiles(List<MultipartFile> files) throws RuntimeException{

        ...생략

        for (MultipartFile multipartFile : files) {
            String savedName = UUID.randomUUID().toString() + "_" + multipartFile.getOriginalFilename();
            Path savePath = Paths.get(uploadPath, savedName);
```

## 썸네일 이미지 처리

→ CustomFileUtil 클래스



```
try {
    Files.copy(multipartFile.getInputStream(), savePath);

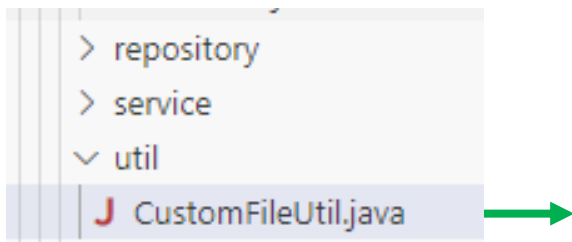
    String contentType = multipartFile.getContentType();
    if(contentType != null && contentType.startsWith("image")){ //이미지여부 확인

        Path thumbnailPath = Paths.get(uploadPath, "s_"+savedName);

        Thumbnails.of(savePath.toFile())
            .size(200,200)
            .toFile(thumbnailPath.toFile());
    }

    uploadNames.add(savedName);
} catch (IOException e) {
    throw new RuntimeException(e.getMessage());
}
} //end for
return uploadNames;
}
```

## 업로드 파일 보여주기



```
@Component
@Log4j2
@RequiredArgsConstructor
public class CustomFileUtil {

    @Value("${org.zerock.upload.path}")
    private String uploadPath;

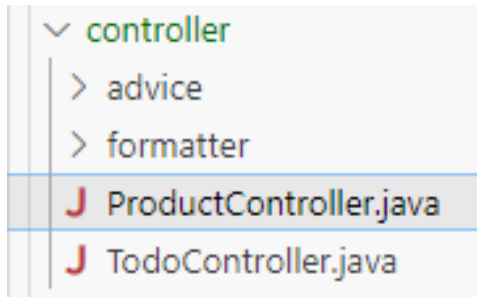
    @PostConstruct
    public void init() { ...생략 }
    public List<String> saveFiles(List<MultipartFile> files) throws RuntimeException { ...생략 }

    public ResponseEntity<Resource> getFile(String fileName) {
        Resource resource = new FileSystemResource(uploadPath+File.separator+fileName);
        if(!resource.exists()) {
            resource = new FileSystemResource(uploadPath+File.separator+"default.jpeg");
        }
        HttpHeaders headers = new HttpHeaders();
        try{
            headers.add("Content-Type", Files.probeContentType(resource.getFile().toPath()));
        } catch (Exception e){
            return ResponseEntity.internalServerError().build();
        }
        return ResponseEntity.ok().headers(headers).body(resource);
    }
}
```

## 업로드 파일 보여주기

→ ProductController.java

viewFileGET( ) 추가



```
@RestController
@RequiredArgsConstructor
@Log4j2
@RequestMapping("/api/products")
public class ProductController {

    private final CustomFileUtil fileUtil;

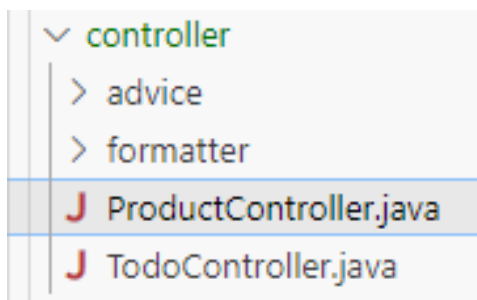
    @PostMapping("/")
    public Map<String, String> register(ProductDTO productDTO){

        log.info("register: " + productDTO);
        List<MultipartFile> files = productDTO.GetFiles();
        List<String> uploadFileNames = fileUtil.saveFiles(files);
        productDTO.setUploadFileNames(uploadFileNames);
        log.info(uploadFileNames);
        return Map.of("RESULT", "SUCCESS");
    }
}
```

## 업로드 파일 보여주기

→ ProductController.java

viewFileGET( ) 추가

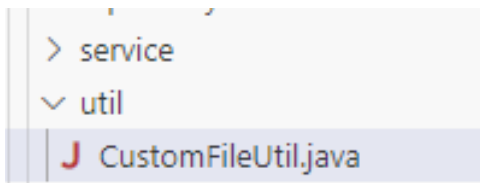


```
@GetMapping("/view/{fileName}")  
public ResponseEntity<Resource> viewFileGET(@PathVariable String fileName){  
  
    return fileUtil.getFile(fileName);  
  
}
```

## 서버 내부에서 파일의 삭제

→ ProductController.java

deleteFiles( ) 추가

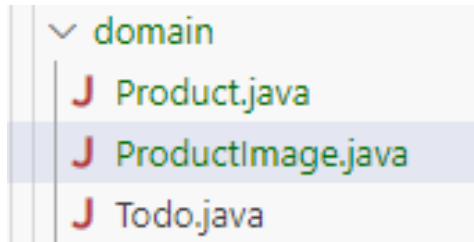


```
public void deleteFiles(List<String> fileNames) {
    if(fileNames == null || fileNames.size() == 0){
        return;
    }

    fileNames.forEach(fileName -> {
        //썸네일이 있는지 확인하고 삭제
        String thumbnailFileName = "s_" + fileName;
        Path thumbnailPath = Paths.get(uploadPath, thumbnailFileName);
        Path filePath = Paths.get(uploadPath, fileName);
        try {
            Files.deleteIfExists(filePath);
            Files.deleteIfExists(thumbnailPath);
        } catch (IOException e) {
            throw new RuntimeException(e.getMessage());
        }
    });
}
```

## Product 엔티티 클래스

→ domain 패키지에 ProductImage 클래스 생성



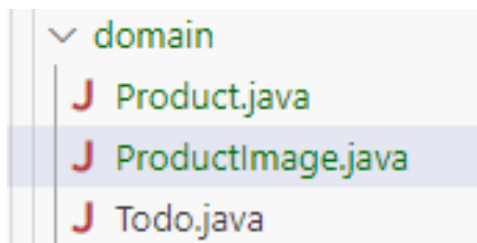
```
@Embeddable
@Getter
@ToString
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class ProductImage {

    private String fileName;
    private int ord;
    public void setOrd(int ord){
        this.ord = ord;
    }
}
```



## Product 엔티티 클래스

→ Product 엔티티 클래스 생성



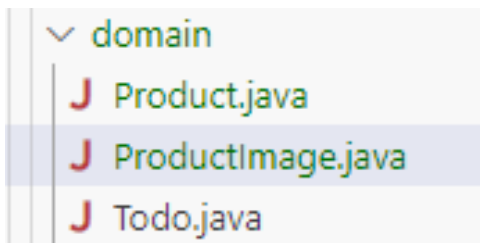
```
@Entity
@Table(name = "tbl_product")
@Getter
@ToString(exclude = "imageList")
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long pno;
    private String pname;
    private int price;
    private String pdesc;
    private boolean delFlag;

    @ElementCollection
    @Builder.Default
    private List<ProductImage> imageList = new ArrayList<>();
```

## Product 엔티티 클래스

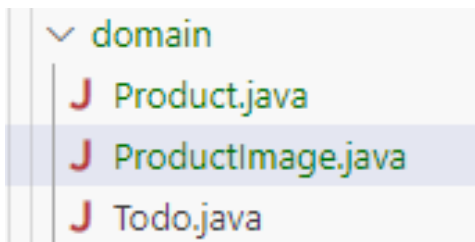
→ Product 엔티티 클래스 생성



```
public void changePrice(int price) {
    this.price = price;
}
public void changeDesc(String desc){
    this.pdesc = desc;
}
public void changeName(String name){
    this.pname = name;
}
public void addImage(ProductImage image) {
    image.setOrd(this.imageList.size());
    imageList.add(image);
}
public void addImageString(String fileName){
    ProductImage productImage = ProductImage.builder()
        .fileName(fileName)
        .build();
    addImage(productImage);
}
public void clearList() {
    this.imageList.clear();
}
}
```

## 레퍼지토리 처리

→ Product



```
package org.zerock.mallapi.domain;

...생략
public class Product {

    ...생략

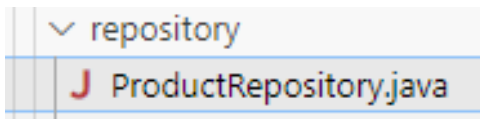
    private boolean delFlag;

    public void changeDel(boolean delFlag) {
        this.delFlag = delFlag;
    }

    ...생략
}
```

## 레퍼지토리 처리

→ JpaRepository



```
package org.zerock.mallapi.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.EntityGraph;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.zerock.mallapi.domain.Product;

public interface ProductRepository extends JpaRepository<Product, Long>{

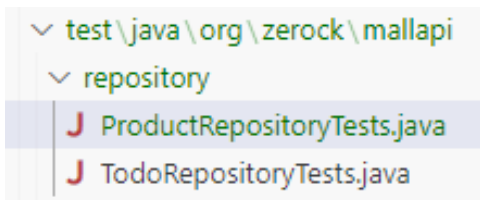
    @EntityGraph(attributePaths = "imageList")
    @Query("select p from Product p where p.pno = :pno")
    Optional<Product> selectOne(@Param("pno") Long pno);

    @Modifying
    @Query("update Product p set p.delFlag = :flag where p.pno = :pno")
    void updateToDelete(@Param("pno") Long pno , @Param("flag") boolean flag);

}
```

## 상품 등록 테스트

→ ProductRepositoryTests.java



```
@SpringBootTest
@Log4j2
public class ProductRepositoryTests {

    @Autowired
    ProductRepository productRepository;

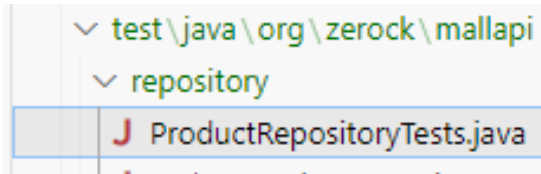
    @Test
    public void testInsert() {
        for (int i = 0; i < 10; i++) {
            Product product = Product.builder()
                .pname("상품"+i)
                .price(100*i)
                .pdesc("상품설명 " + i)
                .build();

            //2개의 이미지 파일 추가
            product.addImageString("IMAGE1.jpg");
            product.addImageString("IMAGE2.jpg");

            productRepository.save(product);
            log.info("-----");
        }
    }
}
```

## 상품 조회와 Lazy loading

→ ProductRepositoryTests.java



```
@Commit
@Transactional
@Test
public void testDelte() {

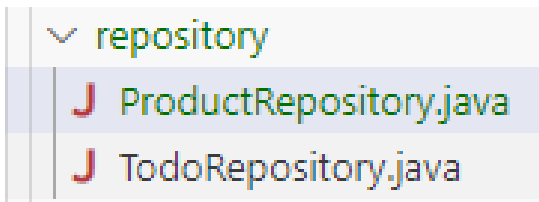
    Long pno = 2L;

    productRepository.updateToDelete(pno, true);

}
```

## 상품 조회와 Lazy loading

→ @EntityGraph



```
package org.zerock.mallapi.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.EntityGraph;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.zerock.mallapi.domain.Product;

public interface ProductRepository extends JpaRepository<Product, Long>{


    @EntityGraph(attributePaths = "imageList")
    @Query("select p from Product p where p.pno = :pno")
    Optional<Product> selectOne(@Param("pno") Long pno);

}
```

## 상품 조회와 Lazy loading

→ @EntityGraph

```
test\java\org\zerock\mallapi
  repository
    ProductRepositoryTests.java
    TodoRepositoryTests.java
```



```
@Test
public void testRead2() {

    Long pno = 1L;

    Optional<Product> result = productRepository.selectOne(pno);

    Product product = result.orElseThrow();

    log.info(product);
    log.info(product.getImageList());


}
```



## 상품의 삭제

→ @Id 값을 이용해서 삭제

```
test\java\org\zerock\mallapi
  repository
    ProductRepositoryTests.java
    TodoRepositoryTests.java
```



```
@Test
public void testDelete() {

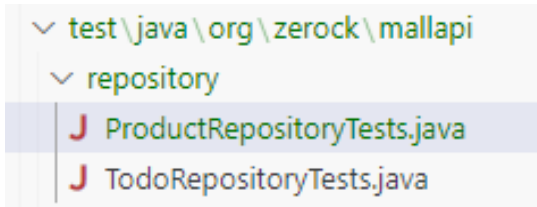
    Long pno = 1L;

    productRepository.deleteById(pno);

}
```

## 상품의 수정

→ 상품의 수정은 Product의 changeXXX( )를 활용



```
@Test
public void testUpdate(){

    Long pno = 10L;
    Product product = productRepository.selectOne(pno).get();
    product.changeName("10번 상품");
    product.changeDesc("10번 상품 설명입니다.");
    product.changePrice(5000);

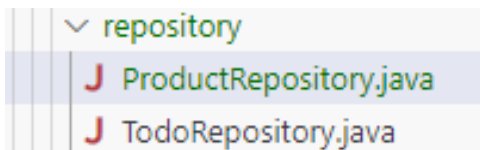
    //첨부파일 수정
    product.clearList();
    product.addImageString(UUID.randomUUID().toString()+"_"+"NEWIMAGE1.jpg");
    product.addImageString(UUID.randomUUID().toString()+"_"+"NEWIMAGE2.jpg");
    product.addImageString(UUID.randomUUID().toString()+"_"+"NEWIMAGE3.jpg");

    productRepository.save(product);

}
```

## 이미지가 포함된 목록 처리

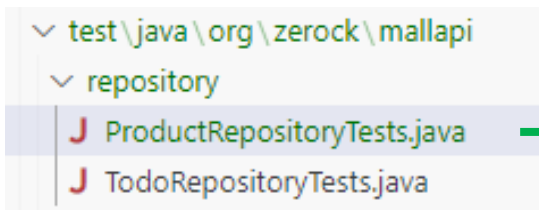
→ ProductRepository에 selectList( ) 추가



```
public interface ProductRepository extends JpaRepository<Product, Long>{  
  
    ...생략  
  
    @Query( "select p, pi  from Product p left join p.imageList pi  where  
pi.ord = 0 and p.delFlag = false " )  
    Page<Object[]> selectList(Pageable pageable);  
  
}
```

## 이미지가 포함된 목록 처리

→ Test



```
@SpringBootTest
@Log4j2
public class ProductRepositoryTests {

    @Autowired
    ProductRepository productRepository;

    ...생략

    @Test
    public void testList() {

        Pageable pageable = PageRequest.of(0, 10, Sort.by("pno").descending());
        Page<Object[]> result = productRepository.selectList(pageable);

        result.getContent().forEach(arr -> log.info(Arrays.toString(arr)));

    }
}
```

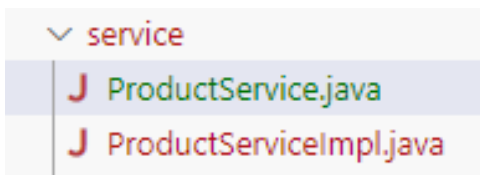
## 서비스 계층

→ DTO와 엔티티 간 변환 주의하여 기능 구현.

→ service 패키지 : ProductService 인터페이스와 ProductServiceImpl 클래스 추가.

## 목록 기능의 처리

→ ProductService인터페이스에 목록을 PageResponseDTO로 처리하는 getList( )를 추가



```
package org.zerock.mallapi.service;

import org.springframework.transaction.annotation.Transactional;
import org.zerock.mallapi.dto.PageRequestDTO;
import org.zerock.mallapi.dto.PageResponseDTO;
import org.zerock.mallapi.dto.ProductDTO;

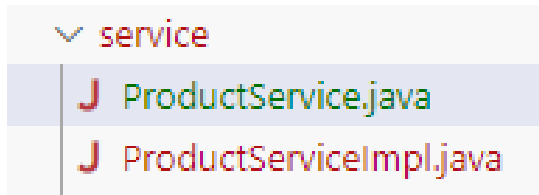
@Transactional
public interface ProductService {

    public PageResponseDTO<ProductDTO> getList(PageRequestDTO pageRequestDTO);

}
```

## 목록 기능의 처리

→ ProductService인터페이스에 목록을 PageResponseDTO로 처리하는 getList( )를 추가



```
@Service
@Log4j2
@RequiredArgsConstructor
@Transactional
public class ProductServiceImpl implements ProductService{

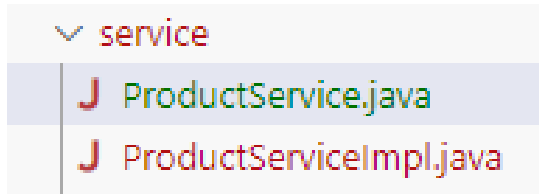
    private final ProductRepository productRepository;

    @Override
    public PageResponseDTO<ProductDTO> getList(PageRequestDTO pageRequestDTO) {
        log.info("getList.....");
        Pageable pageable = PageRequest.of(
            pageRequestDTO.getPage() - 1, //페이지 시작 번호가 0부터 시작하므로
            pageRequestDTO.getSize(),
            Sort.by("pno").descending());

        Page<Object[]> result = productRepository.selectList(pageable);
```

## 목록 기능의 처리

→ ProductService인터페이스에 목록을 PageResponseDTO로 처리하는 getList( )를 추가

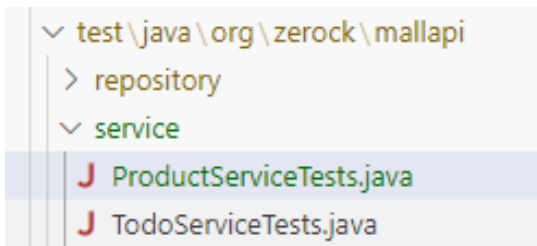


```
List<ProductDTO> dtoList = result.get().map(arr -> {  
  
    Product product = (Product) arr[0];  
    ProductImage productImage = (ProductImage) arr[1];  
  
    ProductDTO productDTO = ProductDTO.builder()  
        .pno(product.getPno())  
        .pname(product.getPname())  
        .pdesc(product.getPdesc())  
        .price(product.getPrice())  
        .build();  
  
    String imageStr = productImage.getFileName();  
    productDTO.setUploadFileNames(List.of(imageStr));  
  
    return productDTO;  
}).collect(Collectors.toList());
```



## 목록 기능의 처리

→ 서비스 목록 기능의 테스트



```
@SpringBootTest
@Log4j2
public class ProductServiceTests {

    @Autowired
    ProductService productService;

    @Test
    public void testList() {

        //1 page, 10 size
        PageRequestDTO pageRequestDTO = PageRequestDTO.builder().build();

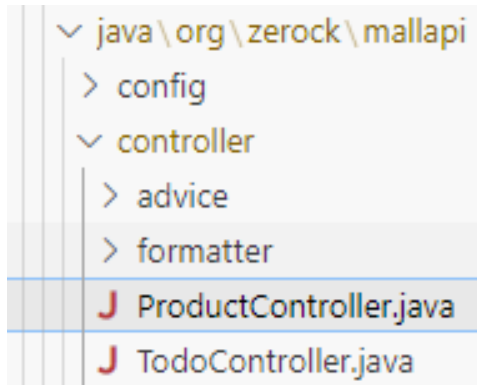
        PageResponseDTO<ProductDTO> result = productService.getList(pageRequestDTO);

        result.getDtoList().forEach(dto -> log.info(dto));

    }
}
```

## 목록 기능의 처리

→ 컨트롤러와 연동 확인



```
@RestController
@RequiredArgsConstructor
@Log4j2
@RequestMapping("/api/products")
public class ProductController {

    private final ProductService productService; //ProductService 주입
    private final CustomFileUtil fileUtil;

    @GetMapping("/list")
    public PageResponseDTO<ProductDTO> list(PageRequestDTO pageRequestDTO) {

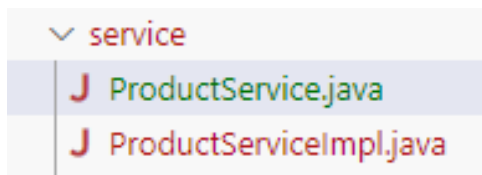
        log.info("list....." + pageRequestDTO);

        return productService.getList(pageRequestDTO);

    }
    ...생략
}
```

## 등록 기능의 처리

→ 서비스의 등록 기능 처리



```
package org.zerock.mallapi.service;
...

public interface ProductService {

    PageResponseDTO<ProductDTO> getList(PageRequestDTO pageRequestDTO);

    Long register(ProductDTO productDTO);

}
```

## 등록 기능의 처리

→ 서비스의 등록 기능 처리



```
@Override
public Long register(ProductDTO productDTO) {

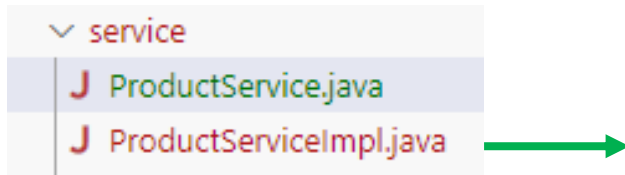
    Product product = dtoToEntity(productDTO);

    Product result = productRepository.save(product);

    return result.getPno();
}
```

## 등록 기능의 처리

→ 서비스의 등록 기능 처리



```
private Product dtoToEntity(ProductDTO productDTO){

    Product product = Product.builder()
        .pno(productDTO.getPno())
        .pname(productDTO.getPname())
        .pdesc(productDTO.getPdesc())
        .price(productDTO.getPrice())
        .build();

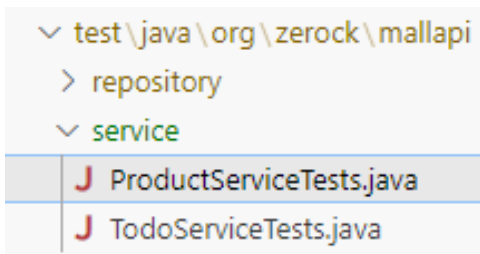
    //업로드 처리가 끝난 파일들의 이름 리스트
    List<String> uploadFileNames = productDTO.getUploadFileNames();

    if(uploadFileNames == null){ return product; }

    uploadFileNames.stream().forEach(uploadName -> {
        product.addImageString(uploadName);
    });
    return product;
}
```

## 등록 기능의 처리

→ Test



```
@SpringBootTest
@Log4j2
public class ProductServiceTests {

    @Autowired
    ProductService productService;

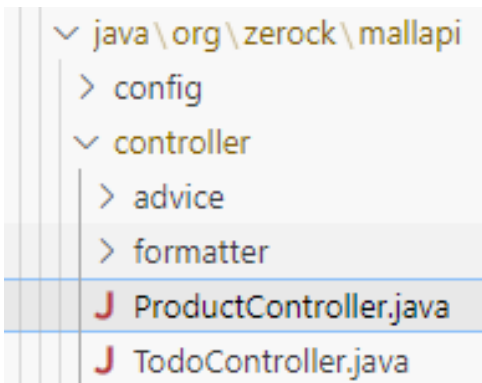
    ...생략

    @Test
    public void testRegister() {
        ProductDTO productDTO = ProductDTO.builder()
            .pname("새로운 상품")
            .pdesc("신규 추가 상품입니다.")
            .price(1000)
            .build();

        //uuid가 있어야 함
        productDTO.setUploadFileNames(
            List.of( UUID.randomUUID()+"_"+"Test1.jpg",
                    UUID.randomUUID()+"_"+"Test2.jpg"));
        productService.register(productDTO);
    }
}
```

## 등록 기능의 처리

→ 컨트롤러와 연동 확인



```
@PostMapping("/")
public Map<String, Long> register(ProductDTO productDTO){

    log.info("register: " + productDTO);

    List<MultipartFile> files = productDTO.GetFiles();

    List<String> uploadFileNames = saveMultipart(files);

    productDTO.setUploadFileNames(uploadFileNames);

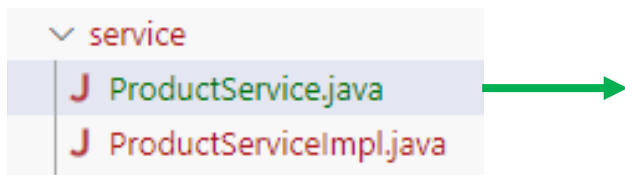
    log.info(uploadFileNames);

    //서비스 호출
    Long pno = productService.register(productDTO);

    return Map.of("RESULT", pno);
}
```

## 조회 기능의 처리

→ 서비스의 조회 기능 처리

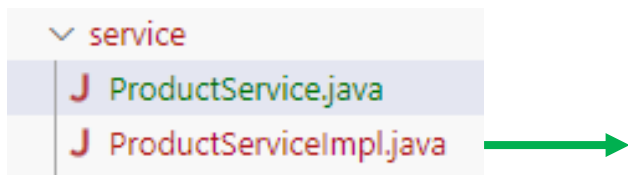


```
public interface ProductService {  
    PageResponseDTO<ProductDTO> getList(PageRequestDTO pageRequestDTO);  
    Long register(ProductDTO productDTO);  
    ProductDTO get(Long pno);  
}
```



## 조회 기능의 처리

→ 서비스의 조회 기능 처리



```
@Override
public ProductDTO get(Long pno) {

    java.util.Optional<Product> result = productRepository.selectOne(pno);

    Product product = result.orElseThrow();

    ProductDTO productDTO = entityToDTO(product);

    return productDTO;
}
```

## 조회 기능의 처리

→ 서비스의 조회 기능 처리



```
private ProductDTO entityToDTO(Product product){

    ProductDTO productDTO = ProductDTO.builder()
        .pno(product.getPno())
        .pname(product.getPname())
        .pdesc(product.getPdesc())
        .price(product.getPrice())
        .build();

    List<ProductImage> imageList = product.getImageList();

    if(imageList == null || imageList.size() == 0 ){
        return productDTO;
    }

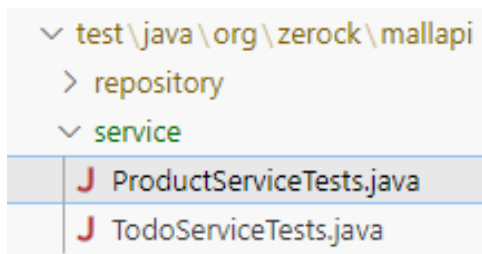
    List<String> fileNameList = imageList.stream().map(productImage ->
        productImage.getUuid()+"_"+productImage.getFileName()).toList();

    productDTO.setUploadFileNames(fileNameList);

    return productDTO;
}
```

## 조회 기능의 처리

→ Test



```
@Test
public void testRead() {

    //실제 존재하는 번호로 테스트
    Long pno = 12L;

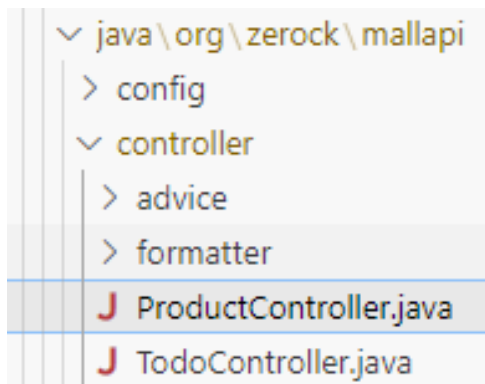
    ProductDTO productDTO = productService.get(pno);

    log.info(productDTO);
    log.info(productDTO.getUploadFileNames());

}
```

## 조회 기능의 처리

→ 컨트롤러 연동 확인



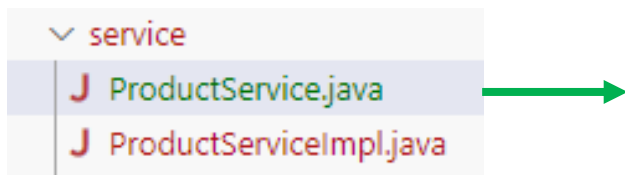
```
@GetMapping("/{tno}")
public ProductDTO read(@PathVariable(name="tno") Long tno) {

    return productService.get(tno);

}
```

## 서비스의 수정 기능 처리

→ 서비스의 수정 기능 처리



```
public interface ProductService {  
    PageResponseDTO<ProductDTO> getList(PageRequestDTO pageRequestDTO);  
    Long register(ProductDTO productDTO);  
    ProductDTO get(Long pno);  
    void modify(ProductDTO productDTO);  
}
```

## 서비스의 수정 기능 처리

→ 서비스의 수정 기능 처리



```
@Override
public void modify(ProductDTO productDTO) {

    //step1 read
    Optional<Product> result=productRepository.findById(productDTO.getPno());

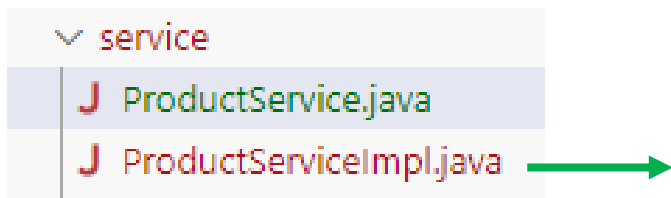
    Product product = result.orElseThrow();

    //change pname, pdesc, price
    product.changeName(productDTO.getPname());
    product.changeDesc(productDTO.getPdesc());
    product.changePrice(productDTO.getPrice());

    //upload File -- clear first
    product.clearList();
}
```

## 서비스의 수정 기능 처리

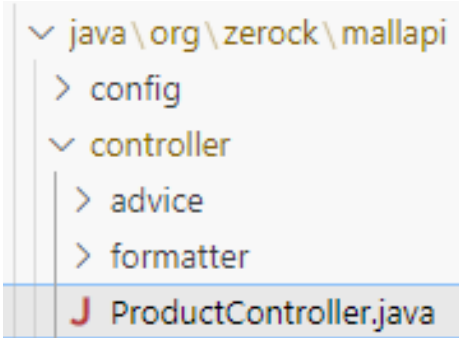
→ 서비스의 수정 기능 처리



```
List<String> uploadFileNames = productDTO.getUploadFileNames();  
if(uploadFileNames != null && uploadFileNames.size() > 0 ){  
    uploadFileNames.stream().forEach(uploadName -> {  
        product.addImage(uploadName);  
    });  
}  
  
productRepository.save(product);  
}
```

## 서비스의 수정 기능 처리

→ 컨트롤러 연동 확인



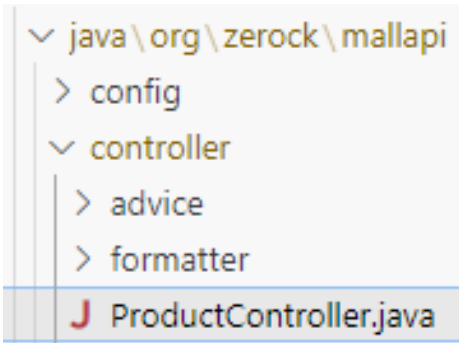
```
@PutMapping("/{pno}")
public Map<String, String> modify(@PathVariable(name="pno")Long pno, ProductDTO
productDTO) {

    productDTO.setPno(pno);
    ProductDTO oldProductDTO = productService.get(pno);
    //기존의 파일들 (데이터베이스에 존재하는 파일들 - 수정 과정에서 삭제되었을 수 있음)
    List<String> oldFileNames = oldProductDTO.getUploadFileNames();
    //새로 업로드 해야 하는 파일들
    List<MultipartFile> files = productDTO.GetFiles();
    //새로 업로드되어서 만들어진 파일 이름들
    List<String> currentUploadFileNames = fileUtil.saveFiles(files);
    //화면에서 변화 없이 계속 유지된 파일들
    List<String> uploadedFileNames = productDTO.getUploadFileNames();
    //유지되는 파일들 + 새로 업로드된 파일 이름들이 저장해야 하는 파일 목록이 됨
    if(currentUploadFileNames != null && currentUploadFileNames.size() > 0) {
        uploadedFileNames.addAll(currentUploadFileNames);
    }
}
```



## 서비스의 수정 기능 처리

→ 컨트롤러 연동 확인



```
//수정 작업
productService.modify(productDTO);

if(oldFileNames != null && oldFileNames.size() > 0){

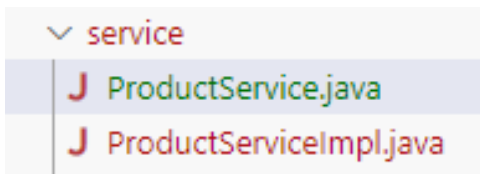
    //지워야 하는 파일 목록 찾기
    //예전 파일들 중에서 지워져야 하는 파일이름들
    List<String> removeFiles = oldFileNames
        .stream()
        .filter(fileName -> uploadedFileNames.indexOf(fileName) == -1)
        .collect(Collectors.toList());

    //실제 파일 삭제
    fileUtil.deleteFiles(removeFiles);
}

return Map.of("RESULT", "SUCCESS");
}
```

## 삭제 기능의 처리

→ 서비스의 삭제 기능 처리



```
@Transactional
public interface ProductService {

    PageResponseDTO<ProductDTO> getList(PageRequestDTO pageRequestDTO);

    Long register(ProductDTO productDTO);

    ProductDTO get(Long pno);

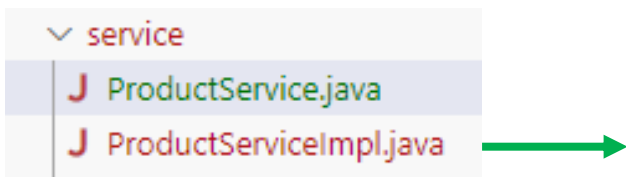
    void modify(ProductDTO productDTO);

    void remove(Long pno);

}
```

## 삭제 기능의 처리

→ 서비스의 삭제 기능 처리



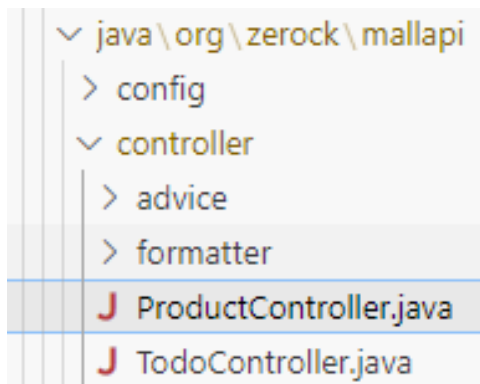
```
@Override
public void remove(Long pno) {

    productRepository.updateToDelete(pno , true);

}
```

## 삭제 기능의 처리

→ 컨트롤러 연동 확인



```
@DeleteMapping("/{pno}")
public Map<String, String> remove(@PathVariable(name="pno") Long pno) {

    // 삭제해야할 파일들 알아내기
    List<String> oldFileNames = oldProductDTO.getUploadFileNames();

    productService.remove(pno);

    fileUtil.deleteFiles(oldFileNames);

    return Map.of("RESULT", "SUCCESS");

}
```

감사합니다.