

Mobile Server

개발 환경

- ❖ Database: MySQL
- ❖ Programming Language: Java
- ❖ Framework: Spring Boot(gradle)
- ❖ IDE: STS

기본 설정

- ❖ 데이터베이스에 사용할 데이터베이스 작성

```
create database adam;
```

기본 설정

- ❖ 프로젝트 생성 – MobileServer
 - ✓ Build Type: Gradle
 - ✓ Language: Java
 - ✓ Dependency: Lombok, MySQL Driver, Spring Boot DevTools, Spring Data JPA, Spring Web, Thymeleaf

기본 설정

- ❖ application.properties 파일 작성

```
server.port=80
```

```
#MySQL
```

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/adam?useUnicode=yes&characterEncoding=UTF-8&serverTimezone=UTC  
spring.datasource.username=root  
spring.datasource.password=wnddkd
```

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

```
spring.jpa.properties.hibernate.show_sql=true
```

```
spring.jpa.properties.hibernate.format_sql=true
```

```
logging.level.org.hibernate.type.descriptor.sql=trace
```

```
spring.jpa.hibernate.ddl-auto=update
```

```
#Live Reload
```

```
spring.devtools.livereload.enabled=true
```

기본 설정

- ❖ 외부 라이브러리 설정
 - ✓ build.gradle 파일의 dependencies 에 추가하고 rebuild
implementation group: 'org.mindrot', name: 'jbcrypt', version: '0.4'

기본 설정

- ❖ Entity 작업

- ✓ MobileServerApplication.java 파일에서 JPA 감시 옵션을 설정

```
import org.springframework.boot.SpringApplication;  
  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;  
  
@EnableJpaAuditing  
@SpringBootApplication  
public class ToDoApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(ToDoApplication.class, args);  
    }  
}
```

기본 설정

- ❖ Entity 작업
 - ✓ model 패키지에 공통된 속성을 소유하는 BaseEntity 생성

```
import java.time.LocalDateTime;
import javax.persistence.Column;
import javax.persistence.EntityListeners;
import javax.persistence.MappedSuperclass;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import lombok.Getter;

@MappedSuperclass
@EntityListeners(value = { AuditingEntityListener.class })
@Getter
abstract class BaseEntity {
    @CreatedDate
    @Column(name = "regdate", updatable = false)
    private LocalDateTime regDate;

    @LastModifiedDate
    @Column(name = "moddate" )
    private LocalDateTime modDate;
}
```

Repository

- ❖ 데이터베이스 작업
 - ✓ 데이터베이스에 접속해서 회원 정보를 위한 테이블을 생성

```
create table tbl_member (
    email varchar(255) not null,
    password varchar(255) not null,
    name varchar(255),
    imageurl varchar(255),
    regdate datetime(6),
    moddate datetime(6),
    lastlogindate datetime(6),
    primary key (email)
) engine=InnoDB DEFAULT CHARSET=utf8;
```

Repository

- ❖ 데이터베이스 작업
 - ✓ 데이터베이스에 접속해서 아이템 정보를 위한 테이블을 생성

```
CREATE TABLE item(  
    itemid INTEGER PRIMARY KEY AUTO_INCREMENT,  
    itemname VARCHAR(100),  
    price INTEGER,  
    description VARCHAR(200),  
    pictureurl VARCHAR(255),  
    member_email varchar(255),  
    foreign key(member_email) references tbl_member(email) on delete cascade  
)engine=InnoDB DEFAULT CHARSET=utf8;
```

Repository

- ❖ Entity 작업
 - ✓ model 패키지에 회원 정보를 위한 Member Entity 생성

```
import lombok.*;  
import java.time.LocalDateTime;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@Getter  
@ToString  
@Table(name = "tbl_member")  
public class Member extends BaseEntity {  
    @Id  
    private String email;  
    private String password;  
    private String name;  
    private String imageurl;  
    private LocalDateTime lastlogindate;  
}
```

Repository

- ❖ Entity 작업
 - ✓ model 패키지에 ITEM 테이블 데이터를 위한 Entity Class 생성

```
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.ManyToOne;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.EqualsAndHashCode;  
import lombok.NoArgsConstructor;  
import lombok.ToString;
```

Repository

- ❖ Entity 작업
 - ✓ model 패키지에 ITEM 테이블 데이터를 위한 Entity Class 생성
 - @Builder
 - @Data
 - @EqualsAndHashCode(callSuper=false)
 - @NoArgsConstructor
 - @AllArgsConstructor
 - @Entity
 - @ToString(exclude = "member")

Repository

❖ Entity 작업

- ✓ model 패키지에 ITEM 테이블 데이터를 위한 Entity Class 생성

```
public class Item{  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long itemid;  
  
    @Column(length = 100, nullable = false)  
    private String itemname;  
  
    @Column  
    private Integer price;  
  
    @Column(length = 400)  
    private String description;  
  
    private String pictureurl;  
  
    @ManyToOne(fetch = FetchType.LAZY)  
    private Member member;  
}
```

Repository

- ❖ Repository 작업

- ✓ Member Entity에 대한 CRUD 작업을 위한 인터페이스를 MemberRepository를 persistence 패키지에 생성

```
public interface MemberRepository extends JpaRepository<Member, String>{  
    List<Member> findMemberByName(String name);  
}
```

Repository

- ❖ Repository 작업

- ✓ Repository의 CRUD 작업을 테스트 하기 위한 클래스를 테스트 디렉토리에 생성하고 작성한 후 테스트

```
@SpringBootTest
```

```
public class RepositoryTest {
```

```
    @Autowired
```

```
    MemberRepository memberRepository;
```

Repository

❖ Repository 작업

- ✓ Repository의 CRUD 작업을 테스트 하기 위한 클래스를 테스트 디렉토리에 생성하고 작성한 후 테스트

```
//회원 가입 - 데이터 삽입
//@Test
public void testInsertMember(){
    Member member =
        Member.builder().email("ggangpae1@gmail.com").password("123456").name("아담").buil
        d();
    memberRepository.save(member);
    System.out.println(member);

    String password = BCrypt.hashpw("123456", BCrypt.gensalt());
    System.out.println(password);
    member =
        Member.builder().email("ggangpae1@naver.com").password(password).name("군계").buil
        d();
    memberRepository.save(member);
    System.out.println(member);
    System.out.println(BCrypt.checkpw("123456", password));
}
```

Repository

- ❖ Repository 작업

- ✓ Repository의 CRUD 작업을 테스트 하기 위한 클래스를 테스트 디렉토리에 생성하고 작성한 후 테스트

```
//모든 회원의 정보 가져오기
//@Test
public void testAllMember(){
    List<Member> list = memberRepository.findAll();
    System.out.println(list);
}
```

Repository

❖ Repository 작업

- ✓ Repository의 CRUD 작업을 테스트 하기 위한 클래스를 테스트 디렉토리에 생성하고 작성한 후 테스트

```
//한 명의 회원 정보 가져오기
```

```
//@Test
```

```
public void testGetMember(){
```

```
    Optional<Member> optional =
```

```
memberRepository.findById("ggangpae1@gmail.com");
```

```
    System.out.println(optional.isPresent());
```

```
    if(optional.isPresent()) {
```

```
        Member member = optional.get();
```

```
        System.out.println(member);
```

```
    }else {
```

```
        System.out.println("존재하지 않는 데이터");
```

```
}
```

Repository

- ❖ Repository 작업
 - ✓ Repository의 CRUD 작업을 테스트 하기 위한 클래스를 테스트 디렉토리에 생성하고 작성한 후 테스트

```
optional =  
memberRepository.findById("ggangpae1@kakao.com");  
System.out.println(optional.isPresent());  
  
if(optional.isPresent()) {  
    Member member = optional.get();  
    System.out.println(member);  
}else {  
    System.out.println("존재하지 않는 데이터");  
}  
}
```

Repository

- ❖ Repository 작업
 - ✓ Repository의 CRUD 작업을 테스트 하기 위한 클래스를 테스트 디렉토리에 생성하고 작성한 후 테스트

```
//이름 중복 체크
@Test
public void testGetName(){
    List <Member> list =
memberRepository.findMemberByName("아담");
    if(list.size() > 0) {
        System.out.println("이미 존재하는 이름");
    }else {
        System.out.println("이미 존재하지 않는 이름");
    }
    list = memberRepository.findMemberByName("이브");
    if(list.size() > 0) {
        System.out.println("존재하는 이름");
    }else {
        System.out.println("존재하지 않는 이름");
    }
}
```

Repository

- ❖ Repository 작업
 - ✓ Repository의 CRUD 작업을 테스트 하기 위한 클래스를 테스트 디렉토리에 생성하고 작성한 후 테스트

```
//회원 정보 수정
//@Test
public void testUpdateMember(){
    String password = BCrypt.hashpw("123456", BCrypt.gensalt());
    Member member =
Member.builder().email("ggangpae1@gmail.com").password(password).name("아담").buil
d();
    memberRepository.save(member);
    System.out.println(member);
}
```

Repository



Repository 작업

- ✓ Repository의 CRUD 작업을 테스트 하기 위한 클래스를 테스트 디렉토리에 생성하고 작성한 후 테스트

```
//회원 정보 삭제
//@Test
public void testDeleteMember(){
    Member member =
        Member.builder().email("ggangpae1@naver.com").build();
    memberRepository.delete(member);

    Optional<Member> optional =
        memberRepository.findById("ggangpae1@naver.com");
    System.out.println(optional.isPresent());
    if(optional.isPresent()) {
        member = optional.get();
        System.out.println(member);
    }else {
        System.out.println("존재하지 않는 데이터");
    }
}
```

Repository

- ❖ Repository 작업

- ✓ persistency 패키지에 ItemRepository 인터페이스를 생성하고 작성

```
import java.util.List;
```

```
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;
```

```
import com.adamsoft.model.Item;  
import com.adamsoft.model.Member;
```

```
@Repository  
public interface ItemRepository extends JpaRepository<Item, Long>{  
    List<Item> findItemByMember(Member member);  
}
```

Repository

❖ Repository 작업

- ✓ RepositoryTest 클래스에서 테스트

```
@Autowired  
ItemRepository itemRepository;  
  
//삽입 확인  
//@Test  
public void testInsertItem(){  
    Member member =  
Member.builder().email("ggangpae1@gmail.com").build();  
    Item item = Item.builder().itemname("사과").price(3000).description("아침에  
좋은 과일").pictureurl("apple.png").member(member).build();  
    itemRepository.save(item);  
    item = Item.builder().itemname("배").price(3000).description("수분 보충에  
좋은 과일").pictureurl("pear.png").member(member).build();  
    itemRepository.save(item);  
    System.out.println(item);  
}
```

Repository

- ❖ Repository 작업

- ✓ RepositoryTest 클래스에서 테스트

- //테이블의 데이터 전체 보기

- //@Test

- public void testAllItem(){

- List<Item> list = itemRepository.findAll();

- System.out.println(list);

- }

Repository

- ❖ Repository 작업

- ✓ RepositoryTest 클래스에서 테스트

```
//페이지 과 정렬  
//@Test  
public void testSortItem() {  
    Sort sort = Sort.by("itemid").descending();  
    Pageable pageable = PageRequest.of(0, 10, sort);  
  
    Page<Item> result = itemRepository.findAll(pageable);  
    result.get().forEach(item -> {  
        System.out.println(item);  
    });  
}
```

Repository

- ❖ Repository 작업

- ✓ RepositoryTest 클래스에서 테스트

```
//Member를 이용한 조회  
//@Test  
public void testMemberItem(){  
    Member member =  
        Member.builder().email("ggangpae1@gmail.com").build();  
    List<Item> list = itemRepository.findItemByMember(member);  
    System.out.println(list);  
}
```

Repository

- ❖ Repository 작업

- ✓ RepositoryTest 클래스에서 테스트

```
//데이터 1개 가져오기
//@Transactional
//@Test
public void testGetItem(){
    Optional<Item> optional = itemRepository.findById(10L);
    System.out.println(optional.isPresent());
    if(optional.isPresent()) {
        Item item = optional.get();
        System.out.println(item);
        System.out.println(item.getMember());
    }else {
        System.out.println("존재하지 않는 데이터");
    }
}
```

Repository

- ❖ Repository 작업

- ✓ RepositoryTest 클래스에서 테스트

```
optional = itemRepository.findById(20L);
System.out.println(optional.isPresent());

if(optional.isPresent()) {
    Item item = optional.get();
    System.out.println(item);
    System.out.println(item.getMember());
} else {
    System.out.println("존재하지 않는 데이터");
}
}
```

Repository

❖ Repository 작업

- ✓ RepositoryTest 클래스에서 테스트

```
//아이템 정보 수정
//@Test
@Transactional
public void testUpdateItem(){
    Member member =
Member.builder().email("ggangpae1@gmail.com").build();
    Item item =
Item.builder().itemid(10L).itemname("사과").price(3000).description("아침에 정말 좋은
과일").pictureurl("apple.png").member(member).build();
    itemRepository.save(item);

    Optional<Item> optional = itemRepository.findById(10L);
    System.out.println(optional.isPresent());
    if(optional.isPresent()) {
        item = optional.get();
        System.out.println(item);
        System.out.println(item.getMember());
    }else {
        System.out.println("존재하지 않는 데이터");
    }
}
```

Repository

- ❖ Repository 작업

- ✓ RepositoryTest 클래스에서 테스트

```
//아이템 정보 삭제  
//@Test  
public void testDeleteItem(){  
    itemRepository.deleteById(10L);
```

```
Optional<Item> optional = itemRepository.findById(10L);  
System.out.println(optional.isPresent());  
if(optional.isPresent()) {  
    Item item = optional.get();  
    System.out.println(item);  
    System.out.println(item.getMember());  
}else {  
    System.out.println("존재하지 않는 데이터");  
}
```

Repository

- ❖ Repository 작업

- ✓ RepositoryTest 클래스에서 테스트

```
Item item = Item.builder().itemid(11L).build();
itemRepository.delete(item);

optional = itemRepository.findById(10L);
System.out.println(optional.isPresent());
if(optional.isPresent()) {
    item = optional.get();
    System.out.println(item);
    System.out.println(item.getMember());
} else {
    System.out.println("존재하지 않는 데이터");
}
}
```

Service

❖ DTO

- ✓ Member 요청을 받아서 저장하고 출력을 위한 DTO 클래스를 생성 - MemberDTO

```
import java.time.LocalDateTime;
import org.springframework.web.multipart.MultipartFile;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Data
@AllArgsConstructor
@NoArgsConstructor
public class MemberDTO {
    private String email;
    private String password;
    private String name;
    private String imageurl;
    private MultipartFile image;
    private LocalDateTime lastLoginDate;
    private LocalDateTime regDate;
    private LocalDateTime modDate;
}
```

Service

- ❖ Service 작업
 - ✓ Member 요청을 받아서 처리할 메서드의 원형을 가진 Service 인터페이스를 생성 – MemberService

```
import org.mindrot.jbcrypt.BCrypt;
public interface MemberService {
    public String registerMember(MemberDTO dto);
    public MemberDTO loginMember(MemberDTO dto);
    public MemberDTO getMember(MemberDTO dto);
    public void updateMember(MemberDTO dto);
    public void deleteMember(MemberDTO dto);
```

Service

❖ Service 작업

- ✓ Member 요청을 받아서 처리할 메서드의 원형을 가진 Service 인터페이스를 생성 – MemberService

```
public default Member dtoToEntity(MemberDTO dto){  
    String password = BCrypt.hashpw(dto.getPassword(), BCrypt.gensalt());  
  
    Member member =  
        Member.builder()  
            .email(dto.getEmail())  
            .password(password)  
            .name(dto.getName())  
            .imageurl(dto.getImageurl())  
            .lastlogindate(dto.getLastLoginDate())  
            .build();  
  
    return member;  
}
```

Service

❖ Service 작업

- ✓ Member 요청을 받아서 처리할 메서드의 원형을 가진 Service 인터페이스를 생성 – MemberService

```
public default MemberDTO entityToDto(Member member){  
    MemberDTO dto =  
        MemberDTO.builder()  
            .email(member.getEmail())  
            .name(member.getName())  
            .imageurl(member.getImageurl())  
            .regDate(member.getRegDate())  
            .modDate(member.getModDate())  
            .lastLoginDate(member.getLastlogindate())  
            .build();  
  
    return dto;  
}  
  
}
```

Service

❖ Service 작업

- ✓ Member 요청을 받아서 처리할 메서드를 구현한 ServiceImpl 클래스를 생성 – MemberServiceImpl

```
import java.util.Optional;
import org.mindrot.jbcrypt.BCrypt;
import org.springframework.stereotype.Service;

import com.adamsoft.dto.MemberDTO;
import com.adamsoft.model.Member;
import com.adamsoft.persistence.MemberRepository;

import lombok.RequiredArgsConstructor;

@Service
@RequiredArgsConstructor
public class MemberServiceImpl implements MemberService {
    private final MemberRepository memberRepository;
```

Service

- ❖ Service 작업
 - ✓ Member 요청을 받아서 처리할 메서드를 구현한 ServiceImpl 클래스를 생성 – MemberServiceImpl

```
@Override  
public String registerMember(MemberDTO dto) {  
    Member member = dtoToEntity(dto);  
    memberRepository.save(member);  
    return member.getEmail();  
}
```

Service

❖ Service 작업

- ✓ Member 요청을 받아서 처리할 메서드를 구현한 ServiceImpl 클래스를 생성 – MemberServiceImpl

```
@Override  
public MemberDTO loginMember(MemberDTO dto) {  
    Optional<Member> optional = memberRepository.findById(dto.getEmail());  
    MemberDTO result = null;  
    if(optional.isPresent()) {  
        Member member = optional.get();  
        if(BCrypt.checkpw(dto.getPassword(), member.getPassword())){  
            result = entityToDto(member);  
            result.setPassword(dto.getPassword());  
            ZonedDateTime nowUTC =  
ZonedDateTime.now(ZoneId.of("UTC"));  
            LocalDateTime now =  
nowUTC.withZoneSameInstant(ZoneId.of("Asia/Seoul")).toLocalDateTime();  
            result.setLastLoginDate(now);  
            memberRepository.save(dtoToEntity(result));  
        }  
    }  
    return result;  
}
```

Service

❖ Service 작업

- ✓ Member 요청을 받아서 처리할 메서드를 구현한 ServiceImpl 클래스를 생성 – MemberServiceImpl

```
@Override  
public MemberDTO getMember(MemberDTO dto) {  
    Optional<Member> optional = memberRepository.findById(dto.getEmail());  
    MemberDTO result = null;  
    if(optional.isPresent()) {  
        Member member = optional.get();  
        result = entityToDto(member);  
    }  
    return result;  
}  
  
@Override  
public void updateMember(MemberDTO dto) {  
    Member member = dtoToEntity(dto);  
    memberRepository.save(member);  
}
```

Service

- ❖ Service 작업
 - ✓ Member 요청을 받아서 처리할 메서드를 구현한 ServiceImpl 클래스를 생성 – MemberServiceImpl

```
@Override  
public void deleteMember(MemberDTO dto) {  
    memberRepository.deleteById(dto.getEmail());  
}  
}
```

Service

- ❖ Service 작업
 - ✓ Service 계층을 테스트 하기 위한 클래스를 생성하고 테스트 –ServiceTest

```
import org.junit.jupiter.api.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import com.adamsoft.dto.MemberDTO;  
import com.adamsoft.service.MemberService;  
  
@SpringBootTest  
public class ServiceTest {  
    @Autowired  
    MemberService memberService;  
  
    //회원 가입  
    //@Test  
    public void testInsertMember(){  
        MemberDTO dto =  
        MemberDTO.builder().email("ggangpae1@gmail.com").password("123456").name("아담").  
        imageUrl("adam.png").build();  
        String email = memberService.registerMember(dto);  
        System.out.println(email);  
    }  
}
```

Service

- ❖ Service 작업
 - ✓ Service 계층을 테스트 하기 위한 클래스를 생성하고 테스트 -ServiceTest

```
//로그인
//@Test
public void testLoginMember(){
    MemberDTO dto =
MemberDTO.builder().email("ggangpae1@gmail.com").password("123456").build();
    MemberDTO result = memberService.loginMember(dto);
    System.out.println(result);

    dto =
MemberDTO.builder().email("ggangpae1@gmail.com").password("1234567").build();
    result = memberService.loginMember(dto);
    System.out.println(result);

    dto =
MemberDTO.builder().email("ggangpae1@naver.com").password("123456").build();
    result = memberService.loginMember(dto);
    System.out.println(result);
}
```

Service

❖ Service 작업

- ✓ Service 계층을 테스트 하기 위한 클래스를 생성하고 테스트 -ServiceTest

```
//회원 정보 가져오기
//@Test
public void testGetMember(){
    MemberDTO dto =
MemberDTO.builder().email("ggangpae1@gmail.com").build();
    MemberDTO result = memberService.getMember(dto);
    System.out.println(result);

    dto = MemberDTO.builder().email("ggangpae1@naver.com").build();
    result = memberService.loginMember(dto);
    System.out.println(result);
}

//회원 정보 수정
//@Test
public void testUpdateMember(){
    MemberDTO dto =
MemberDTO.builder().email("ggangpae1@gmail.com").password("1234567").name("군계"
).imageurl("rusia.png").build();
    memberService.updateMember(dto);
}
```

Service

- ❖ Service 작업
 - ✓ Service 계층을 테스트 하기 위한 클래스를 생성하고 테스트 -ServiceTest

```
//회원 정보 삭제
//@Test
public void testDeleteMember(){
    MemberDTO dto =
MemberDTO.builder().email("ggangpae1@gmail.com").build();
    memberService.deleteMember(dto);
}
```

Service

❖ Service 작업

- ✓ Item을 출력하고 매개변수를 받기 위한 ItemDTO 클래스를 생성하고 작성

```
import org.springframework.web.multipart.MultipartFile;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@Builder  
@NoArgsConstructor  
@AllArgsConstructor  
public class ItemDTO {  
    private Long itemid;  
    private String itemname;  
    private Integer price;  
    private String description;  
    private String pictureurl;  
    private MultipartFile image;  
    private String email;  
}
```

Service

- ❖ Service 작업
 - ✓ 페이지 단위 요청을 처리하기 위한 PageRequestItemDTO 클래스를 생성하고 작성

```
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
```

```
@Builder
@AllArgsConstructor
@Data
public class PageRequestItemDTO {
    private int page;
    private int size;
    public PageRequestDTO(){
        this.page = 1;
        this.size = 10;
    }
    public Pageable getPageable(Sort sort){
        return PageRequest.of(page - 1, size, sort);
    }
}
```

Service

- ❖ Service 작업
 - ✓ 페이지 단위 결과를 처리하기 위한 PageResultItemDTO 클래스를 생성하고 작성

```
@Data  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
public class PageResultItemDTO {  
    private String error;  
    //DTO리스트  
    private List<ItemDTO> itemList;  
  
    //전체 페이지 개수  
    private int totalPage;  
  
    //현재 페이지 번호  
    private int page;  
  
    //목록 사이즈  
    private int size;  
  
    //시작 페이지 번호, 끝 페이지 번호  
    private int start, end;
```

Service

- ❖ Service 작업

- ✓ 페이지 단위 결과를 처리하기 위한 PageResultItemDTO 클래스를 생성하고 작성

```
//이전, 다음  
private boolean prev, next;  
  
//페이지 번호 목록  
private List<Integer> pageList;  
  
public void makePageList(Pageable pageable){  
    this.page = pageable.getPageNumber() + 1; // 0부터 시작하므로 1을 추가  
    this.size = pageable.getPageSize();  
  
    int tempEnd = (int)(Math.ceil(page/10.0)) * 10;  
    start = tempEnd - 9;  
    prev = start > 1;  
  
    end = totalPage > tempEnd ? tempEnd: totalPage;  
    next = totalPage > tempEnd;  
    pageList = IntStream.rangeClosed(start, end).boxed().collect(Collectors.toList());  
}  
}
```

Service

- ❖ Service 작업

- ✓ Item 요청을 처리하기 위한 메서드의 원형을 위한 ItemService 인터페이스를 생성

```
public interface ItemService {  
    public Long registerItem(ItemDTO dto);  
    public ItemDTO getItem(ItemDTO dto);  
    public void updateItem(ItemDTO dto);  
    public void deleteItem(ItemDTO dto);  
    public PageResultItemDTO getList(PageRequestItemDTO pageRequestDTO);
```

Service

❖ Service 작업

- ✓ Item 요청을 처리하기 위한 메서드의 원형을 위한 ItemService 인터페이스를 생성

```
public default Item dtoToEntity(ItemDTO dto){  
    Item item =  
        Item.builder()  
            .itemid(dto.getItemId())  
            .itemname(dto.getItemname())  
            .price(dto.getPrice())  
            .description(dto.getDescription())  
            .pictureurl(dto.getPictureurl())  
            .member(Member.builder().email(dto.getEmail()).build())  
            .build();  
  
    return item;  
}
```

Service

- ❖ Service 작업

- ✓ Item 요청을 처리하기 위한 메서드의 원형을 위한 ItemService 인터페이스를 생성

```
public default ItemDTO entityToDto(Item item){  
    ItemDTO dto =  
        ItemDTO.builder()  
            .itemid(item.getItemId())  
            .itemname(item.getItemname())  
            .price(item.getPrice())  
            .description(item.getDescription())  
            .pictureurl(item.getPictureurl())  
            .email(item.getMember().getEmail())  
            .build();  
  
    return dto;  
}
```

}

Service

❖ Service 작업

- ✓ Item 요청을 처리하기 위한 메서드를 구현한 ItemServiceImpl 클래스를 생성

```
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import lombok.RequiredArgsConstructor;

@Service
@RequiredArgsConstructor
public class ItemServiceImpl implements ItemService {
    private final ItemRepository itemRepository;
```

Service

- ❖ Service 작업
 - ✓ Item 요청을 처리하기 위한 메서드를 구현한 ItemServiceImpl 클래스를 생성

```
@Override  
public Long registerItem(ItemDTO dto) {  
    Item item = dtoToEntity(dto);  
    itemRepository.save(item);  
    return item.getItemId();  
}  
  
@Override  
public ItemDTO getItem(ItemDTO dto) {  
    Long itemid = dto.getItemId();  
    Optional<Item> optional = itemRepository.findById(itemid);  
    if(optional.isPresent()) {  
        return entityToDto(optional.get());  
    }  
    return null;  
}
```

Service

- ❖ Service 작업

- ✓ Item 요청을 처리하기 위한 메서드를 구현한 ItemServiceImpl 클래스를 생성

```
@Override  
public void updateItem(ItemDTO dto) {  
    Item item = dtoToEntity(dto);  
    System.out.println(item);  
    itemRepository.save(item);  
}
```

```
@Override  
public void deleteItem(ItemDTO dto) {  
    Long itemid = dto.getItemId();  
    itemRepository.deleteById(itemid);  
}
```

Service

- ❖ Service 작업
 - ✓ Item 요청을 처리하기 위한 메서드를 구현한 ItemServiceImpl 클래스를 생성

```
@Override  
public PageResultItemDTO getList(PageRequestItemDTO pageRequestDTO) {  
    Sort sort = Sort.by("itemid").descending();  
  
    Pageable pageable = PageRequest.of(pageRequestDTO.getPage()-1,  
pageRequestDTO.getSize(), sort);  
    Page <Item> page = itemRepository.findAll(pageable);  
  
    PageResultItemDTO result = new PageResultItemDTO();  
    result.makePageList(pageable);  
    result.setTotalPage(page.getTotalPages());  
  
    List <ItemDTO> list = new ArrayList<ItemDTO>();  
    page.get().forEach(item -> {  
        list.add(entityToDto(item));  
    });  
    result.setItemList(list);  
  
    return result;  
}
```

Service

- ❖ Service 작업
 - ✓ ItemService를 테스트 하기 코드를 테스트 클래스에 작성하고 확인

```
@Autowired  
ItemService itemService;  
  
//@Test  
public void testInsertItem(){  
    for(int i=0; i<100; i++) {  
        ItemDTO dto = ItemDTO.builder()  
            .itemname("apple_" + i)  
            .price(3000)  
            .description("사과_" + i)  
            .pictureurl("apple_" + i + ".png")  
            .email("ggangpae1@gmail.com")  
            .build();  
  
        Long itemid = itemService.registerItem(dto);  
        System.out.println(itemid);  
    }  
}
```

Service

- ❖ Service 작업
 - ✓ ItemService를 테스트 하기 코드를 테스트 클래스에 작성하고 확인

```
//데이터 1개 가져오기
//@Test
public void testGetItem(){
    ItemDTO dto = ItemDTO.builder().itemid(100L).build();
    ItemDTO result = itemService.getItem(dto);
    System.out.println(result);

    dto = ItemDTO.builder().itemid(200L).build();
    result = itemService.getItem(dto);
    System.out.println(result);
}
```

Service

- ❖ Service 작업
 - ✓ ItemService를 테스트 하기 코드를 테스트 클래스에 작성하고 확인

```
//아이템 정보 수정
//@Transactional
//@Test
public void testUpdateItem(){

    ItemDTO dto =
ItemDTO.builder().itemid(101L).itemname("배").price(3200).description("수분이 많은
과일").pictureurl("pear.png").email("ggangpae1@gmail.com").build();
    itemService.updateItem(dto);

    dto = ItemDTO.builder().itemid(100L).build();
    ItemDTO result = itemService.getItem(dto);
    System.out.println(result);
}
```

Service

- ❖ Service 작업
 - ✓ ItemService를 테스트 하기 코드를 테스트 클래스에 작성하고 확인

```
//아이템 정보 삭제
//@Test
public void testDeleteItem(){
    ItemDTO dto = ItemDTO.builder().itemid(100L).build();
    itemService.deleteItem(dto);
}

@Test
public void testPageListItem(){
    PageRequestItemDTO request = new PageRequestItemDTO();
    request.setPage(3);
    request.setSize(10);
    PageResultItemDTO result = itemService.getList(request);
    System.out.println(result);
}
```

Controller

- ❖ 준비 작업
 - ✓ 이미지 업로드를 처리하기 위해서 application.properties 파일에 설정 추가

```
spring.servlet.multipart.enabled=true  
spring.servlet.multipart.location=/Users/adam/Documents/data  
spring.servlet.multipart.max-request-size=30MB  
spring.servlet.multipart.max-file-size=10MB  
  
com.adamsoft.upload.path=/Users/adam/Documents/data
```

Controller

- ❖ 준비 작업
 - ✓ MemberDTO 클래스에 에러 발생 시 에러 메시지를 저장하기 위한 속성을 추가
private String error;

Controller

❖ 준비 작업

- ✓ Member 요청을 처리할 Controller를 생성하고 회원 가입 처리 메서드를 작성

```
@RestController  
@RequestMapping("member")  
@RequiredArgsConstructor  
public class MemberController {  
    @Value("${com.adamsoft.upload.path}")  
    private String uploadPath;  
  
    private final MemberService memberService;  
}
```

Controller

❖ 회원 가입 요청 처리

- ✓ Member 삽입 요청을 처리하기 위해서 MemberServiceImpl 클래스에 속성 과 디렉토리를 생성해주는 메서드를 추가

```
@Value("${com.adamsoft.upload.path}")
private String uploadPath;
```

```
private String makeFolder() {
    String str =
LocalDate.now().format(DateTimeFormatter.ofPattern("yyyy/MM/dd"));
    String realUploadPath = str.replace("//", File.separator);
    // make directory -----
    File uploadPathDir = new File(uploadPath, realUploadPath);
    if (uploadPathDir.exists() == false) {
        uploadPathDir.mkdirs();
    }
    return realUploadPath;
}
```

Controller

❖ 회원 가입 요청 처리

- ✓ MemberServiceImpl 클래스의 Member 삽입 요청을 처리하는 메서드를 수정

```
@Override  
public String registerMember(MemberDTO dto) {  
    MemberDTO result = getMember(dto);  
    if(result != null) {  
        return "이미 존재하는 이메일";  
    }
```

```
List<Member> list =  
memberRepository.findMemberByName(dto.getName());  
if(list.size() > 0){  
    return "이미 존재하는 이름";  
}
```

Controller

❖ 회원 가입 요청 처리

- ✓ MemberServiceImpl 클래스의 Member 삽입 요청을 처리하는 메서드를 수정

```
MultipartFile uploadFile = dto.getImage();
if(uploadFile != null && uploadFile.isEmpty() == false) {
    //실제 파일 이름 IE나 Edge는 전체 경로가 들어오므로
    String originalName = uploadFile.getOriginalFilename();
    String fileName =
originalName.substring(originalName.lastIndexOf("\\") + 1);

    log.info("fileName: " + fileName);
    String realUploadPath = makeFolder();
```

Controller

❖ 회원 가입 요청 처리

- ✓ Member 삽입 요청을 처리하기 위해서 MemberServiceImpl 클래스를 수정

```
//UUID  
String uuid = UUID.randomUUID().toString();  
//저장할 파일 이름 중간에 _를 이용해서 구분  
String saveName = uploadPath + File.separator +  
realUploadPath + File.separator + uuid + fileName;  
Path savePath = Paths.get(saveName);  
  
try {  
    uploadFile.transferTo(savePath);  
  
} catch (IOException e) {  
    System.out.println(e.getLocalizedMessage());  
    e.printStackTrace();  
}  
dto.setImageurl(realUploadPath + File.separator +  
fileName);  
}  
Member member = dtoToEntity(dto);  
memberRepository.save(member);  
return null;  
}
```

Controller

❖ 회원 가입 요청 처리

- ✓ MemberController에 회원 가입 요청을 처리하는 메서드를 작성

```
@PostMapping("/register")
public ResponseEntity<?> registerMember(MemberDTO dto) {
    MemberDTO response = null;
    try {
        String msg = memberService.registerMember(dto);
        if(msg != null) {
            response = MemberDTO.builder().error(msg).build();
        }else {
            response =
                MemberDTO.builder().email(dto.getEmail()).build();
        }
    } catch (Exception e) {
        String error = e.getMessage();
        response = MemberDTO.builder().error(error).build();
    }
    return ResponseEntity.ok().body(response);
}
```

Controller

- ❖ 회원가입 요청 처리
 - ✓ PostMan을 이용해서 테스트

The screenshot shows the Postman application interface. The top bar indicates a POST method and the URL `localhost/member/register`. The "Body" tab is selected, showing form-data parameters: `email` (value: `ggangpae2@gmail.com`), `password` (value: `123456`), `image` (value: `car01.jpg`), and `name` (value: `이브`). Below the table, the "Body" tab is active, showing the raw JSON response:

```
1  "error": null,
2  "email": "ggangpae2@gmail.com",
3  "password": null,
4  "name": null,
5  "imageurl": null,
6  "image": null,
7  "lastLoginDate": null,
8  "regDate": null,
9  "modDate": null
```

The status bar at the bottom right shows `Status: 200 OK Time: 127 ms Size: 316 B`.

Controller

❖ 로그인 처리

- ✓ MemberController 클래스에 로그인 처리 메서드를 작성

```
@PostMapping("/login")
public ResponseEntity<?> loginMember(MemberDTO dto) {
    MemberDTO response = null;
    try {
        MemberDTO memberDTO = memberService.loginMember(dto);
        if(memberDTO != null) {
            response = MemberDTO.builder()
                .email(memberDTO.getEmail())
                .name(memberDTO.getName())
                .imageurl(memberDTO.getImageurl())
                .lastLoginDate(memberDTO.getLastLoginDate())
                .regDate(memberDTO.getRegDate())
                .modDate(memberDTO.getModDate())
                .build();
        }else {
            response = MemberDTO.builder().error("아이디나
비밀번호가 틀렸습니다.").build();
        }
    return ResponseEntity.ok().body(response);
}
```

Controller

- ❖ 로그인 처리

- ✓ MemberController 클래스에 로그인 처리 메서드를 작성

```
    catch (Exception e) {  
        String error = e.getMessage();  
        response = MemberDTO.builder().error(error).build();  
    }  
    return ResponseEntity.ok().body(response);  
}
```

Controller

❖ 로그인 처리

✓ PostMan을 이용해서 테스트

The screenshot shows the Postman application interface. The top bar indicates a POST request to 'localhost/member/login'. The 'Body' tab is selected, showing form-data parameters: 'email' (ggangpae2@gmail.com), 'password' (123456), 'image' (car01.jpg), and 'name' (이브). Below the body, the response tab displays a JSON object with fields: error (null), email (ggangpae2@gmail.com), password (null), name (이브), imageUrl (2022/06/15/824a6682-d4d7-498e-bb17-5aa8c59b36adcar01.jpg), image (null), lastLoginDate (2022-06-15T17:35:02.701442), regDate (2022-06-15T17:25:37.176864), and modDate (2022-06-15T17:34:04.580253). The status bar at the bottom right shows a 200 OK response with a time of 192 ms and a size of 446 B.

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> email	ggangpae2@gmail.com			
<input checked="" type="checkbox"/> password	123456			
<input type="checkbox"/> image	car01.jpg X			
<input type="checkbox"/> name	이브			
Key	Value	Description		

Body Cookies (1) Headers (5) Test Results

Status: 200 OK Time: 192 ms Size: 446 B Save Response

Pretty Raw Preview Visualize JSON

```
1 "error": null,
2 "email": "ggangpae2@gmail.com",
3 "password": null,
4 "name": "이브",
5 "imageUrl": "2022/06/15/824a6682-d4d7-498e-bb17-5aa8c59b36adcar01.jpg",
6 "image": null,
7 "lastLoginDate": "2022-06-15T17:35:02.701442",
8 "regDate": "2022-06-15T17:25:37.176864",
9 "modDate": "2022-06-15T17:34:04.580253"
```

Controller

❖ 회원 정보 가져오기

- ✓ MemberController 클래스에 한 명의 회원 정보를 가져오는 메서드를 작성

```
@GetMapping("/get")
public ResponseEntity<?> getMember(MemberDTO dto) {
    MemberDTO response = null;
    try {
        MemberDTO memberDTO = memberService.getMember(dto);
        if(memberDTO != null) {
            response = MemberDTO.builder()
                .email(memberDTO.getEmail())
                .name(memberDTO.getName())
                .imageurl(memberDTO.getImageurl())
                .build();
        }else {
            response = MemberDTO.builder().error("잘못된
아이디입니다.").build();
        }
        return ResponseEntity.ok().body(response);
    } catch (Exception e) {
        String error = e.getMessage();
        response = MemberDTO.builder().error(error).build();
    }
    return ResponseEntity.ok().body(response);
}
```

Controller

- ❖ 회원 정보 가져오기
 - ✓ PostMan을 이용해서 테스트

The screenshot shows the Postman interface with a GET request to `localhost/member/get`. The Body tab is selected, showing form-data parameters:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> email	ggangpae2@gmail.com			
<input type="checkbox"/> password	123456			
<input type="checkbox"/> image	car01.jpg			
<input type="checkbox"/> name	이브			
Key	Value	Description		

Below the table, the Body tab is selected, showing the JSON response:

```
1 "error": null,
2 "email": "ggangpae2@gmail.com",
3 "password": null,
4 "name": "이브",
5 "imageurl": "2022/06/15/824a6682-d4d7-498e-bb17-5aa8c59b36adcar01.jpg",
6 "image": null,
7 "lastLoginDate": null,
8 "regDate": null,
9 "modDate": null
```

The status bar at the bottom indicates: Status: 200 OK Time: 22 ms Size: 374 B Save Response

Controller

❖ 회원 정보 수정

- ✓ MemberServiceImpl 클래스 회원 정보를 수정하는 메서드를 수정

```
@Override  
public void updateMember(MemberDTO dto) {  
    if(dto.getImage() != null && dto.getImage().isEmpty() == false) {  
        MultipartFile uploadFile = dto.getImage();  
  
        //실제 파일 이름 IE나 Edge는 전체 경로가 들어오므로  
        String originalName = uploadFile.getOriginalFilename();  
        String fileName =  
originalName.substring(originalName.lastIndexOf("\\") + 1);  
  
        log.info("fileName: " + fileName);  
        String realUploadPath = makeFolder();
```

Controller

❖ 회원 정보 수정

- ✓ MemberServiceImpl 클래스 회원 정보를 수정하는 메서드를 수정

```
//UUID  
String uuid = UUID.randomUUID().toString();  
//저장할 파일 이름 중간에 _를 이용해서 구분  
String saveName = uploadPath + File.separator +  
realUploadPath + File.separator + uuid + fileName;  
Path savePath = Paths.get(saveName);  
  
try {  
    uploadFile.transferTo(savePath);  
  
} catch (IOException e) {  
    System.out.println(e.getLocalizedMessage());  
    e.printStackTrace();  
}  
dto.setImageurl(realUploadPath + File.separator +  
fileName);  
}else {  
    dto.setImageurl(getMember(dto).getImageurl());  
}  
Member member = dtoToEntity(dto);  
memberRepository.save(member);  
}
```

Controller

❖ 회원 정보 수정

- ✓ MemberController 클래스에 한 명의 회원 정보를 수정하는 메서드를 작성

```
@PostMapping("/update")
public ResponseEntity<?> updateMember(MemberDTO dto) {
    MemberDTO response = null;
    try {
        memberService.updateMember(dto);
        response = MemberDTO.builder().build();
    } catch (Exception e) {
        String error = e.getMessage();
        response = MemberDTO.builder().error(error).build();
    }
    return ResponseEntity.ok().body(response);
}
```

Controller

❖ 회원 정보 수정

✓ PostMan을 이용해서 테스트

The screenshot shows the Postman application interface. The request method is set to POST, the URL is localhost/member/update, and the Body tab is selected. The body contains four key-value pairs: email (ggangpae3@gmail.com), password (123456), image (car07.jpg), and name (아담1). The response status is 200 OK, and the JSON response body is:

```
1  "error": null,
2  "email": null,
3  "password": null,
4  "name": null,
5  "imageurl": null,
6  "image": null,
7  "lastLoginDate": null,
8  "regDate": null,
9  "modDate": null
```

Controller

❖ 회원 탈퇴

- ✓ MemberController 클래스에 한 명의 회원 정보를 삭제하는 메서드를 작성

```
@PostMapping("/delete")
public ResponseEntity<?> deleteMember(MemberDTO dto) {
    MemberDTO response = null;
    try {
        memberService.deleteMember(dto);
        response = MemberDTO.builder().build();
    } catch (Exception e) {
        String error = e.getMessage();
        response = MemberDTO.builder().error(error).build();
    }
    return ResponseEntity.ok().body(response);
}
```

Controller

❖ 회원 탈퇴

- ✓ PostMan을 이용해서 테스트



The screenshot shows the Postman application interface. A POST request is being made to the endpoint `localhost/member/delete`. The **Body** tab is selected, showing form-data parameters:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> email	ggangpae1@naver.com			
<input type="checkbox"/> password	123456			
<input type="checkbox"/> image	car01.jpg			
<input type="checkbox"/> name	이브			
Key	Value	Description		

Below the table, there are tabs for Body, Cookies (1), Headers (5), and Test Results. The Body tab is active. On the right, the response status is shown as `200 OK` with a response size of `299 B`. The response body is displayed in a JSONpretty-printed format:

```
1
2     "error": null,
3     "email": null,
4     "password": null,
5     "name": null,
6     "imageurl": null,
7     "image": null,
8     "lastLoginDate": null,
9     "regDate": null,
10    "modDate": null
```

Controller

- ❖ 파일 다운로드

- ✓ MemberService 인터페이스에 파일 다운로드를 위한 메서드를 선언

```
public ResponseEntity<Object> download(String path);
```

Controller

❖ 파일 다운로드

- ✓ MemberServiceImpl 클래스에 파일 다운로드를 위한 메서드를 구현

```
@Override  
public ResponseEntity<Object> download(String path) {  
    try {  
        Path filePath = Paths.get(uploadPath + File.separator + path);  
        System.out.println(uploadPath + File.separator + path);  
        Resource resource = new  
InputStreamResource(Files.newInputStream(filePath)); // 파일 resource 얻기  
  
        File file = new File(path);  
  
        HttpHeaders headers = new HttpHeaders();  
  
        headers.setContentDisposition(ContentDisposition.builder("attachment").filename(file.ge  
tName()).build()); // 다운로드 되거나 로컬에 저장되는 용도로 쓰이는지를 알려주는 헤더  
  
        return new ResponseEntity<Object>(resource, headers,  
HttpStatus.OK);  
    } catch(Exception e) {  
        return new ResponseEntity<Object>(null, HttpStatus.CONFLICT);  
    }  
}
```

Controller

❖ 파일 다운로드

- ✓ MemberController 클래스에 파일 다운로드를 위한 메서드를 구현

```
@GetMapping("/download")
public ResponseEntity<Object> download(String path) {
    return memberService.download(path);
}
```

Controller

❖ 준비 작업

- ✓ Item 요청을 처리하는 ItemController 클래스를 생성

```
@RestController  
 @RequestMapping("item")  
 @RequiredArgsConstructor  
 public class ItemController {  
     private final ItemService itemService;  
 }
```

Controller

❖ 준비 작업

- ✓ Item 요청을 처리하기 위해서 ItemDTO 클래스에 에러 메시지를 저장할 속성 추가

```
private String error;
```

Controller

❖ 준비 작업

- ✓ Item 요청을 처리하기 위해서 ItemServiceImpl 클래스에 속성 과 메서드 추가

```
@Value("${com.adamsoft.upload.path}")
private String uploadPath;

private String makeFolder() {
    String str =
LocalDate.now().format(DateTimeFormatter.ofPattern("yyyy/MM/dd"));
    String realUploadPath = str.replace("//", File.separator);
    // make directory -----
    File uploadPathDir = new File(uploadPath, realUploadPath);
    if (uploadPathDir.exists() == false) {
        uploadPathDir.mkdirs();
    }
    return realUploadPath;
}
```

Controller

❖ 준비 작업

- ✓ Item 요청을 처리하기 위해서 ItemServiceImpl 클래스에 속성 과 메서드 추가

```
private void updateDate() {
    try(PrintWriter pw = new PrintWriter(new
    FileOutputStream("./updatedate.dat"));) {
        String str =
    LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
        System.out.println(str);
        pw.println(str);
        pw.flush();
    }catch(Exception e) {
        System.out.println(e.getLocalizedMessage());
    }
}
```

Controller

- ❖ 페이지 단위 요청

- ✓ Item 요청을 처리하는 ItemController 클래스에 페이지 단위로 데이터를 요청을 처리하는 메서드를 구현

```
@GetMapping("/list")
public ResponseEntity<?> getList(PageRequestItemDTO dto) {
    PageResultItemDTO response = null;
    try {
        response = itemService.getList(dto);

    } catch (Exception e) {
        String error = e.getMessage();
        response = PageResultItemDTO.builder().error(error).build();
    }
    return ResponseEntity.ok().body(response);
}
```

Controller

❖ 페이지 단위 요청

The screenshot shows the Postman application interface. At the top, there is a header bar with the method dropdown set to "GET", the URL input field containing "localhost/item/list", and a "Send" button. Below the header, there are tabs for "Params", "Authorization", "Headers (9)", "Body", "Pre-request Script", "Tests", and "Settings". The "Body" tab is currently selected, indicated by an orange underline. Under the "Body" tab, there are several options: "none", "form-data" (which is selected), "x-www-form-urlencoded", "raw", "binary", and "GraphQL". Below these options is a table with columns: KEY, VALUE, DESCRIPTION, and Bulk Edit. The table contains five rows, each with a checked checkbox in the first column:

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	email	ggangpae1@gmail.com			
<input checked="" type="checkbox"/>	itemname	배			
<input checked="" type="checkbox"/>	description	나주			
<input checked="" type="checkbox"/>	price	4000			

In the last row, the "Value" column contains "car24.jpg" with an "X" icon to its right. At the bottom of the "Body" section, there are tabs for "Body", "Cookies (1)", "Headers (5)", and "Test Results", with "Body" being the active tab. To the right of these tabs, there is a status indicator showing "200 OK 38 ms 1.75 KB" and a "Save Response" button. Below the status, there are buttons for "Pretty", "Raw", "Preview", "Visualize", and "JSON" (which is selected). On the far right, there are icons for copy, search, and refresh.

Body Results

```
1  {
2      "error": null,
3      "itemList": [
4          {
5              "itemid": 215,
6              "itemname": "배",
7              "price": 4000,
8              "description": "나주",
9              "pictureurl": "2022/06/17/cdc9d0cf-277c-4d7c-9c72-79ed5425fb62car24.jpg",
--          }
--      ]
--  }
```

Controller

❖ 데이터 삽입

- ✓ 파일 업로드를 위해서 ItemServiceImpl 클래스에 데이터 삽입 요청 처리 메서드를 수정

```
@Override  
public Long registerItem(ItemDTO dto) {  
    MultipartFile uploadFile = dto.getImage();  
    if(uploadFile!= null && uploadFile.isEmpty() == false) {  
  
        //실제 파일 이름 IE나 Edge는 전체 경로가 들어오므로  
        String originalName = uploadFile.getOriginalFilename();  
        String fileName =  
originalName.substring(originalName.lastIndexOf("WW") + 1);  
        String realUploadPath = makeFolder();  
  
        //UUID  
        String uuid = UUID.randomUUID().toString();  
        //저장할 파일 이름 중간에 _를 이용해서 구분  
        String saveName = uploadPath + File.separator +  
realUploadPath + File.separator + uuid + fileName;  
        Path savePath = Paths.get(saveName);
```

Controller

❖ 데이터 삽입

- ✓ 파일 업로드를 위해서 ItemServiceImpl 클래스에 데이터 삽입 요청 처리 메서드를 수정

```
try {  
    uploadFile.transferTo(savePath);  
  
} catch (IOException e) {  
    System.out.println(e.getLocalizedMessage());  
    e.printStackTrace();  
}  
dto.setPictureurl(realUploadPath + File.separator +  
fileName);  
}  
  
Item item = dtoToEntity(dto);  
itemRepository.save(item);  
updateDate();  
return item.getItemId();  
}
```

Controller

❖ 데이터 삽입

- ✓ ItemController 클래스에 데이터를 삽입하는 요청을 구현

```
@PostMapping("/register")
public ResponseEntity<?> registerItem(ItemDTO dto) {
    ItemDTO response = null;
    try {
        Long itemid = itemService.registerItem(dto);
        response = ItemDTO.builder().itemid(itemid).build();

    } catch (Exception e) {
        String error = e.getMessage();
        response = ItemDTO.builder().itemid(0L).description(error).build();
    }
    return ResponseEntity.ok().body(response);
}
```

Controller

❖ 데이터 삽입

The screenshot shows the Postman application interface for making a POST request to the endpoint `localhost/item/register`. The request method is set to `POST`. The `Body` tab is selected, showing a `form-data` structure with the following fields:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> email	ggangpae1@gmail.com			
<input checked="" type="checkbox"/> itemname	배			
<input checked="" type="checkbox"/> description	나주			
<input checked="" type="checkbox"/> price	4000		x	
<input checked="" type="checkbox"/> image	car24.jpg		x	

Below the table, the `Body` tab is selected, showing the raw JSON response:

```
1 {  
2   "itemid": 215,  
3   "itemname": null,  
4   "price": null,  
5   "description": null,  
6   "pictureurl": null,  
7   "image": null,  
8   "email": null  
9 }
```

Controller

- ❖ 데이터 가져오기

- ✓ ItemController 클래스에 데이터 1개를 가져오는 요청을 구현

```
@GetMapping("/get")
public ResponseEntity<?> getItem(ItemDTO dto) {
    ItemDTO response = null;
    try {
        response = itemService.getItem(dto);
    } catch (Exception e) {
        String error = e.getMessage();
        response = ItemDTO.builder().itemid(0L).description(error).build();
    }
    return ResponseEntity.ok().body(response);
}
```

Controller

❖ 데이터 가져오기

The screenshot shows the Postman application interface for making a GET request to `localhost/item/get`. The request method is set to `GET`. The `Body` tab is selected, showing the following form-data parameters:

Key	Value	Description
itemname	미	
description	나주	
price	4000	
image	car24.jpg	
itemid	120	

Below the table, the `Body` tab is selected, showing the JSON response:

```
1 {  
2   "itemid": 120,  
3   "itemname": "apple_5",  
4   "price": 3000,  
5   "description": "사과_5",  
6   "pictureurl": "apple_5.png",  
7   "image": null,  
8   "email": "ggangpae1@gmail.com"  
9 }
```

Controller

❖ 데이터 수정

- ✓ ItemServiceImpl 클래스의 데이터를 수정하는 메서드를 수정

```
@Override  
public void updateItem(ItemDTO dto) {  
    if(dto.getImage() != null && dto.getImage().isEmpty() == false) {  
        MultipartFile uploadFile = dto.getImage();  
  
        //실제 파일 이름 IE나 Edge는 전체 경로가 들어오므로  
        String originalName = uploadFile.getOriginalFilename();  
        String fileName =  
originalName.substring(originalName.lastIndexOf("\\") + 1);  
  
        String realUploadPath = makeFolder();  
  
        //UUID  
        String uuid = UUID.randomUUID().toString();  
        //저장할 파일 이름 중간에 _를 이용해서 구분  
        String saveName = uploadPath + File.separator +  
realUploadPath + File.separator + uuid + fileName;  
        Path savePath = Paths.get(saveName);
```

Controller

❖ 데이터 수정

- ✓ ItemServiceImpl 클래스의 데이터를 수정하는 메서드를 수정

```
try {
    uploadFile.transferTo(savePath);

} catch (IOException e) {
    System.out.println(e.getLocalizedMessage());
    e.printStackTrace();
}
dto.setPictureurl(realUploadPath + File.separator + 
fileName);
}else {
    dto.setPictureurl(getItem(dto).getPictureurl());
}
Item item = dtoToEntity(dto);
itemRepository.save(item);
updateDate();
}
```

Controller

❖ 데이터 수정

- ✓ ItemController 클래스에 데이터를 수정하는 요청을 구현

```
@PostMapping("/update")
public ResponseEntity<?> updateItem(ItemDTO dto) {
    ItemDTO response = null;
    try {
        itemService.updateItem(dto);
        response = itemService.getItem(dto);
    } catch (Exception e) {
        String error = e.getMessage();
        response = ItemDTO.builder().itemid(0L).description(error).build();
    }
    return ResponseEntity.ok().body(response);
}
```

Controller

❖ 데이터 수정

The screenshot shows the Postman application interface for making a POST request to the endpoint `localhost/item/update`. The request method is set to `POST`, and the URL is `localhost/item/update`. The `Body` tab is selected, showing the following form-data parameters:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> email	ggangpae1@gmail.com			
<input checked="" type="checkbox"/> itemname	무화과			
<input checked="" type="checkbox"/> description	목포			
<input checked="" type="checkbox"/> price	2500			
<input checked="" type="checkbox"/> image	car24.jpg			

Below the table, the `Body` tab is active, showing the raw JSON representation of the request body:

```
1 {  
2   "itemid": 120,  
3   "itemname": "무화과",  
4   "price": 2500,  
5   "description": "목포",  
6   "pictureurl": "2022/06/17/c732a5e4-bf09-4ece-bd2e-295048e5ae04car24.jpg",  
7   "image": null,  
8   "email": "ggangpae1@gmail.com"  
9 }
```

Controller

- ❖ 데이터 삭제
 - ✓ ItemServiceImpl 클래스의 데이터를 삭제하는 메서드를 수정

```
@Override  
public void deleteItem(ItemDTO dto) {  
    Long itemid = dto.getItemId();  
    itemRepository.deleteById(itemid);  
    updateDate();  
}
```

Controller

❖ 데이터 삭제

- ✓ ItemController 클래스에 데이터를 삭제하는 요청을 구현

```
@PostMapping("/delete")
public ResponseEntity<?> deleteItem(ItemDTO dto) {
    ItemDTO response = null;
    try {
        itemService.deleteItem(dto);
        response = ItemDTO.builder().itemid(dto.getItemId()).build();
    } catch (Exception e) {
        String error = e.getMessage();
        response = ItemDTO.builder().itemid(0L).description(error).build();
    }
    return ResponseEntity.ok().body(response);
}
```

Controller

❖ 데이터 삭제

The screenshot shows the Postman application interface. The URL in the header is `localhost/item/delete`. The method dropdown shows `POST`, and the body tab is selected. The body type is set to `form-data`. The table below lists several parameters:

Key	Value	Description
description	국산	
price	2500	
image	car24.jpg	
itemid	120	
pictureurl	apple_5.png	

Below the table, the response section shows a status of `200 OK` with `37 ms` and `270 B` response size.

```
1 {"itemid": 120,
2 "itemname": null,
3 "price": null,
4 "description": null,
5 "pictureurl": null,
6 "image": null,
7 "email": null}
```

Controller

- ❖ 마지막 수정 시간을 전송
 - ✓ ItemService 인터페이스에 마지막 수정 시간을 전송하는 메서드를 선언

```
public String updatedate();
```

Controller

- ❖ 마지막 수정 시간을 전송

- ✓ ItemServiceImpl 클래스에 마지막 수정 시간을 전송하는 메서드를 선언

```
@Override  
public String updatedate() {  
    try(BufferedReader br = new BufferedReader(new InputStreamReader(new  
        FileInputStream("./updatedate.dat")))) {  
        String str = br.readLine();  
        return str;  
    }catch(Exception e) {  
        System.out.println(e.getLocalizedMessage());  
        return null;  
    }  
}
```

Controller

- ❖ 마지막 수정 시간을 전송
 - ✓ ItemController 클래스에 마지막 수정 시간을 전송하는 메서드를 선언

```
@GetMapping("/updatedate")
public ResponseEntity<?> updateDate() {

    String updatedate = itemService.updatedate();
    Map<String, String> response = new HashMap<>();
    response.put("updatedate", updatedate);
    return ResponseEntity.ok().body(response);
}
```

Controller

❖ 데이터 삭제

The screenshot shows the Postman application interface. At the top, the URL is set to `localhost/item/updatedate`. Below the URL, the method is selected as `GET`. The `Body` tab is active, showing the following data:

Key	Value	Description
description	목포	
price	2500	
image	car24.jpg	
itemid	129	
pictureurl	apple_5.png	

At the bottom of the body section, there is a JSON preview:

```
1 {  
2   "updatedate": "2022-06-17 12:27:23"  
3 }
```