

Java/Spring 테스트를 추가하고 싶은 개발자들의 오답노트

방향성 탐색 (2)



Contents

01

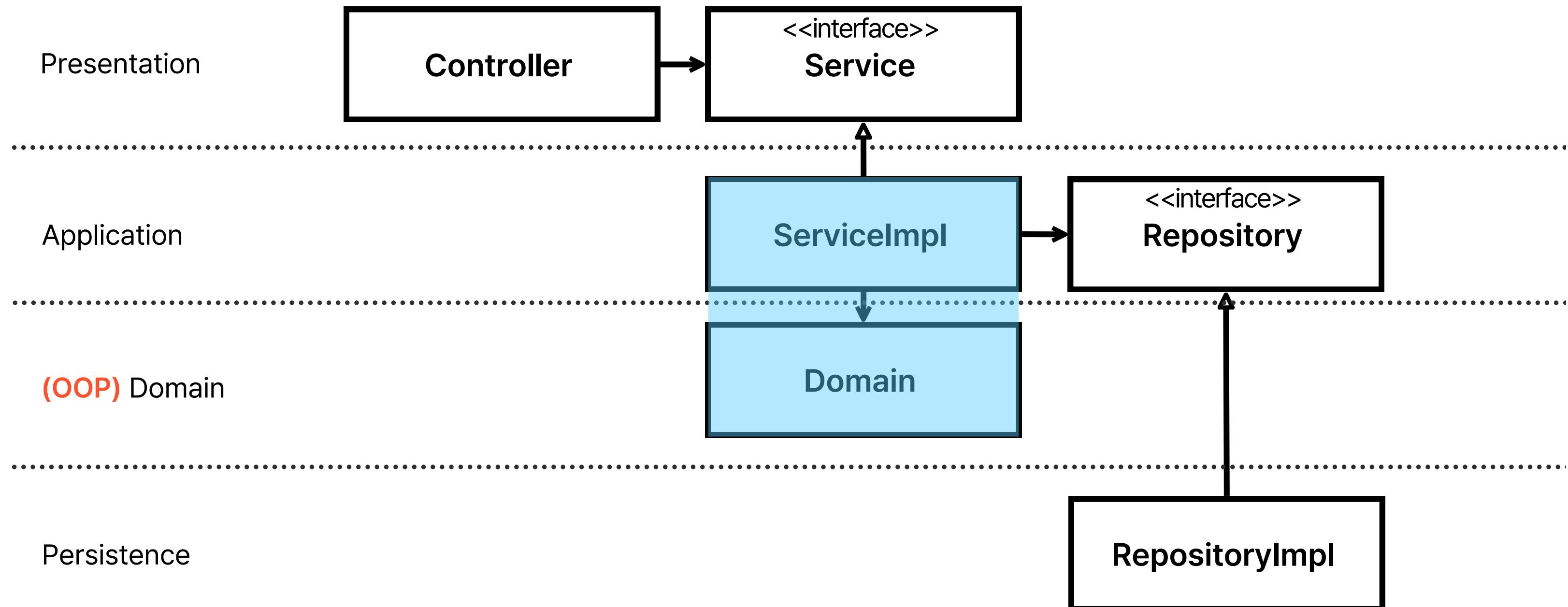
테스트의 범위

02

실기전 추가 내용

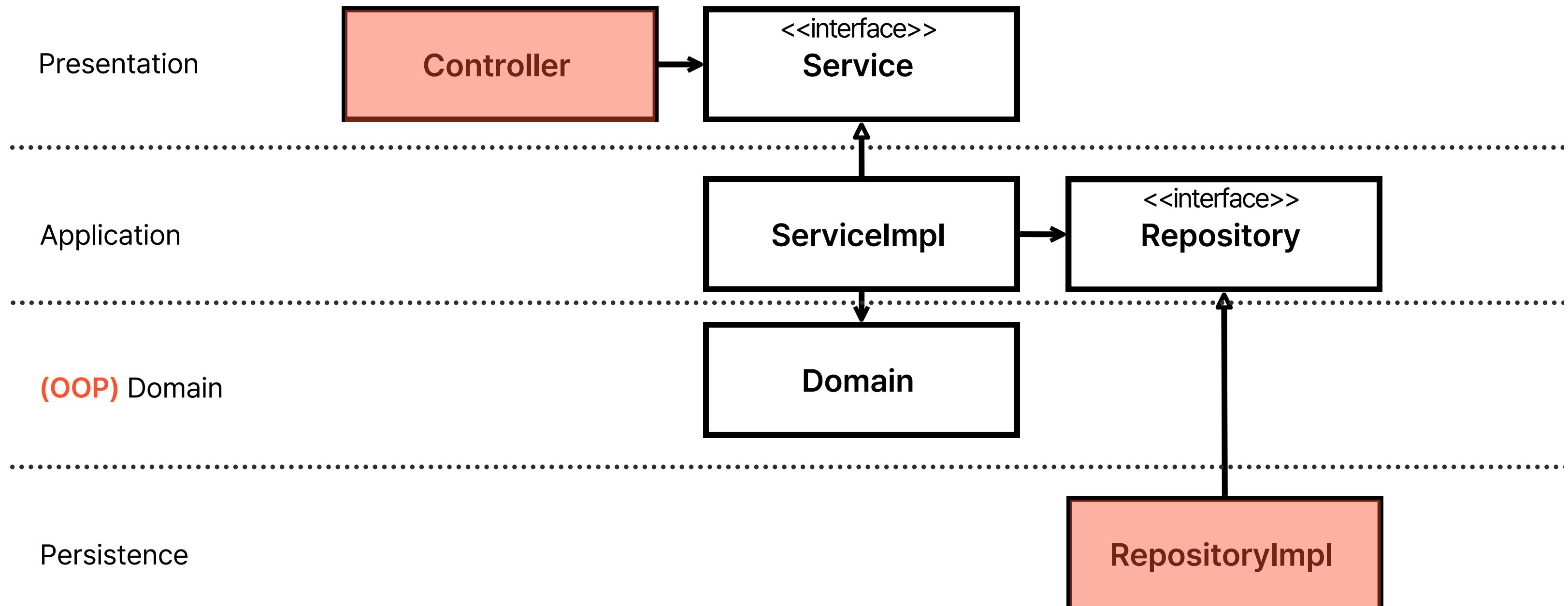
테스트의 범위

집중해야 하는 테스트



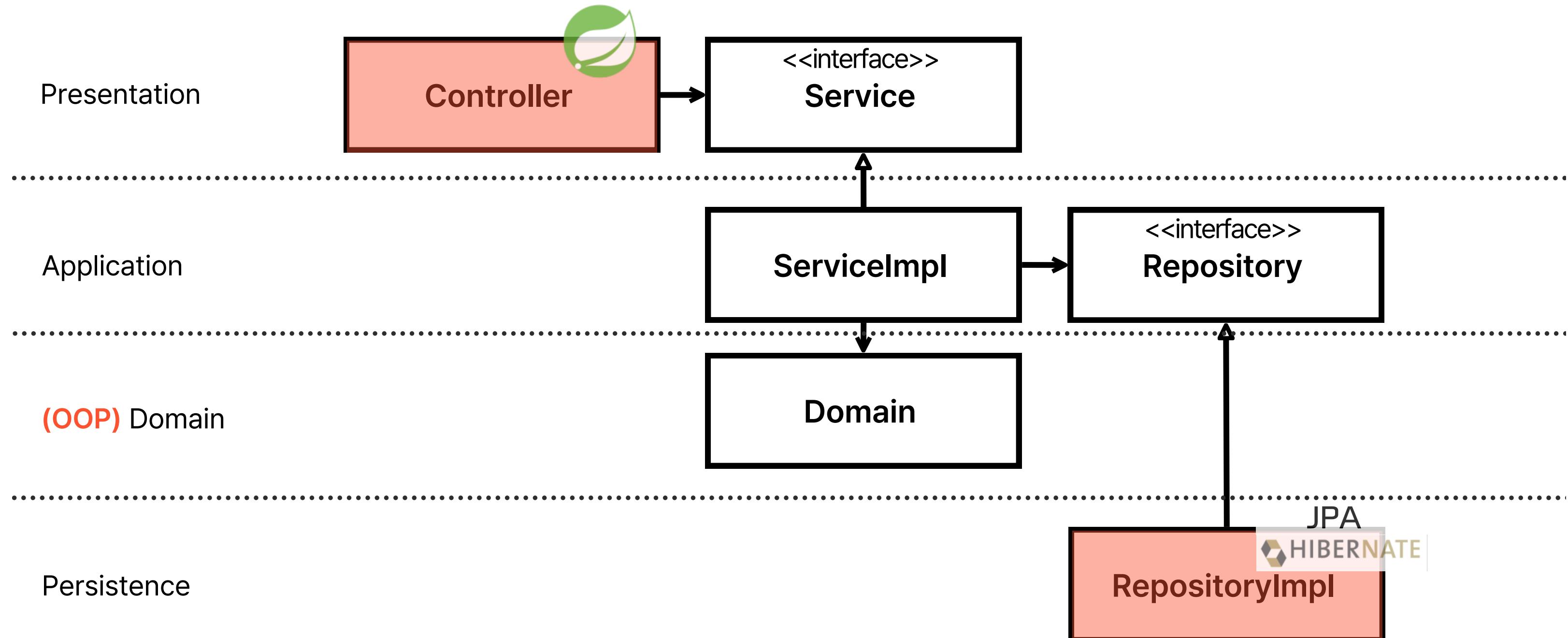
테스트의 범위

이 둘은 왜 테스트 범위로 잡지 않으려 할까요?

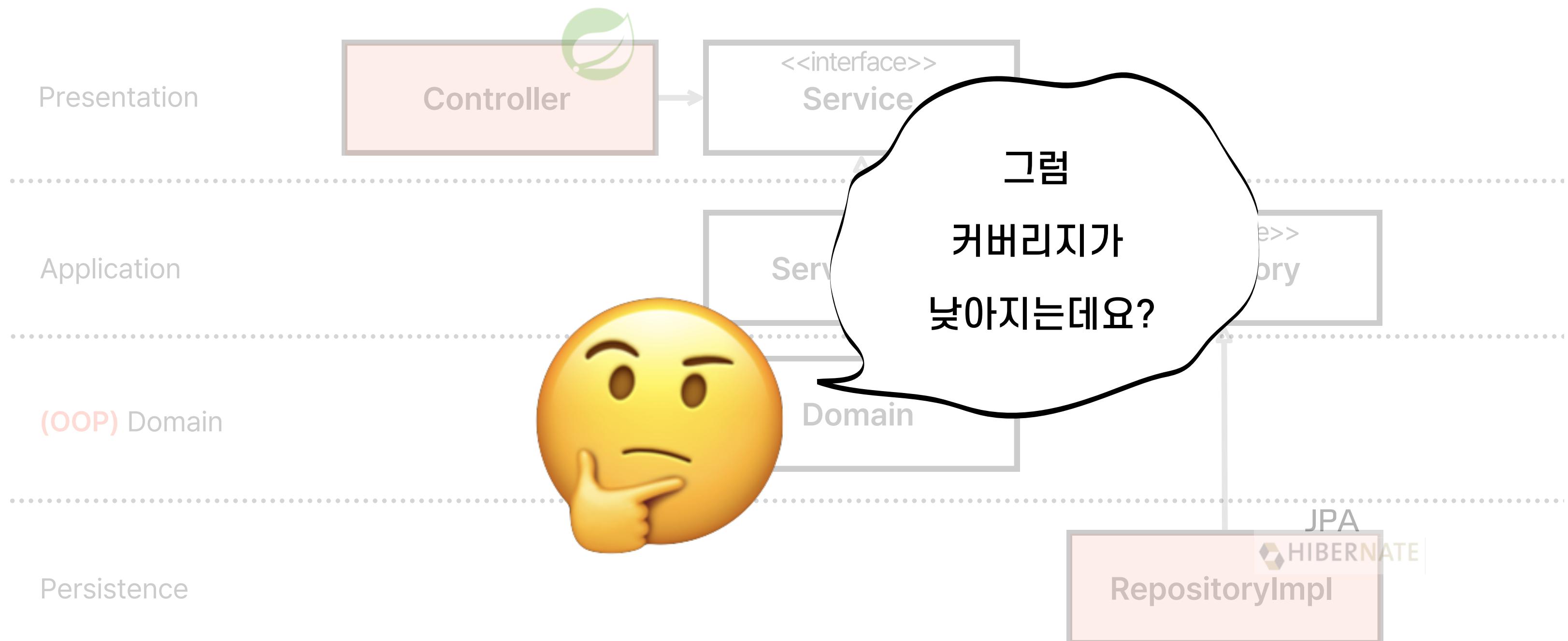


테스트의 범위

프레임워크와 라이브러리들이 잘해줄 것이다

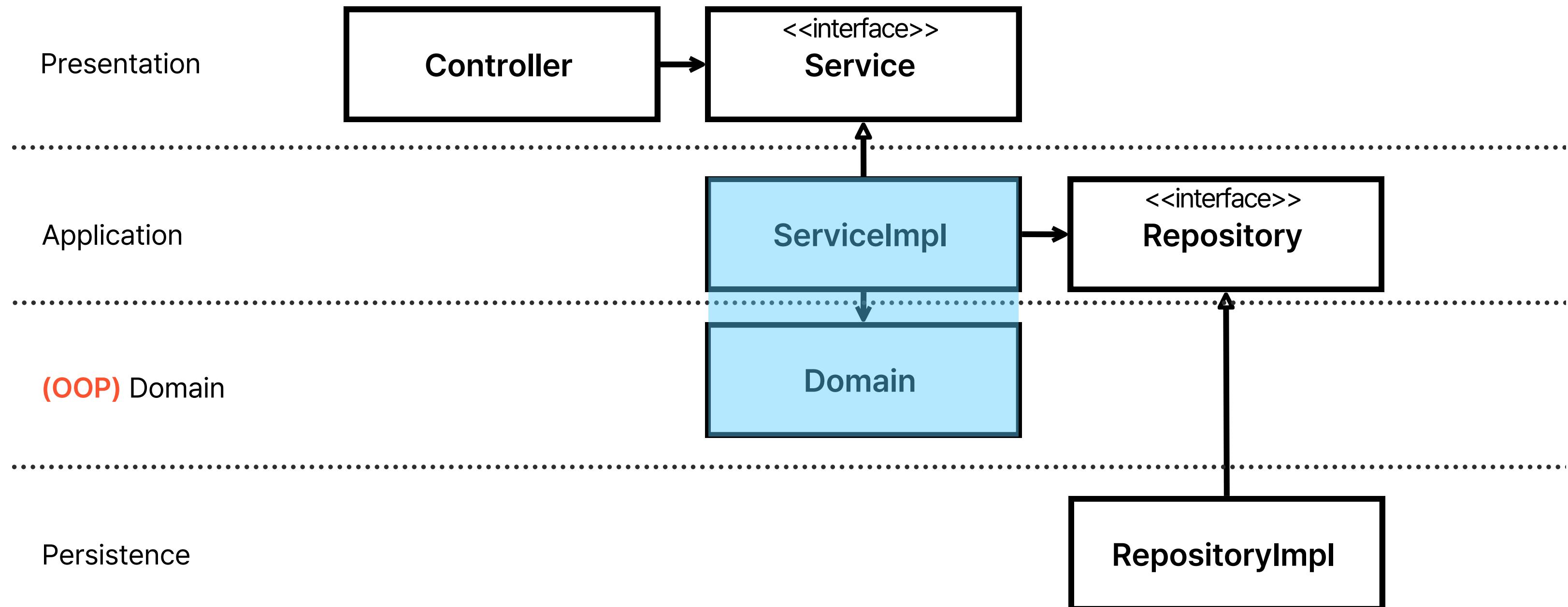


테스트의 범위



테스트의 범위

집중해야 하는 테스트 => 본질



낮은 커버리지 걱정



• Jpa, Spring쪽 테스트를 안하면 커버리지가 너무 낮게 나온다

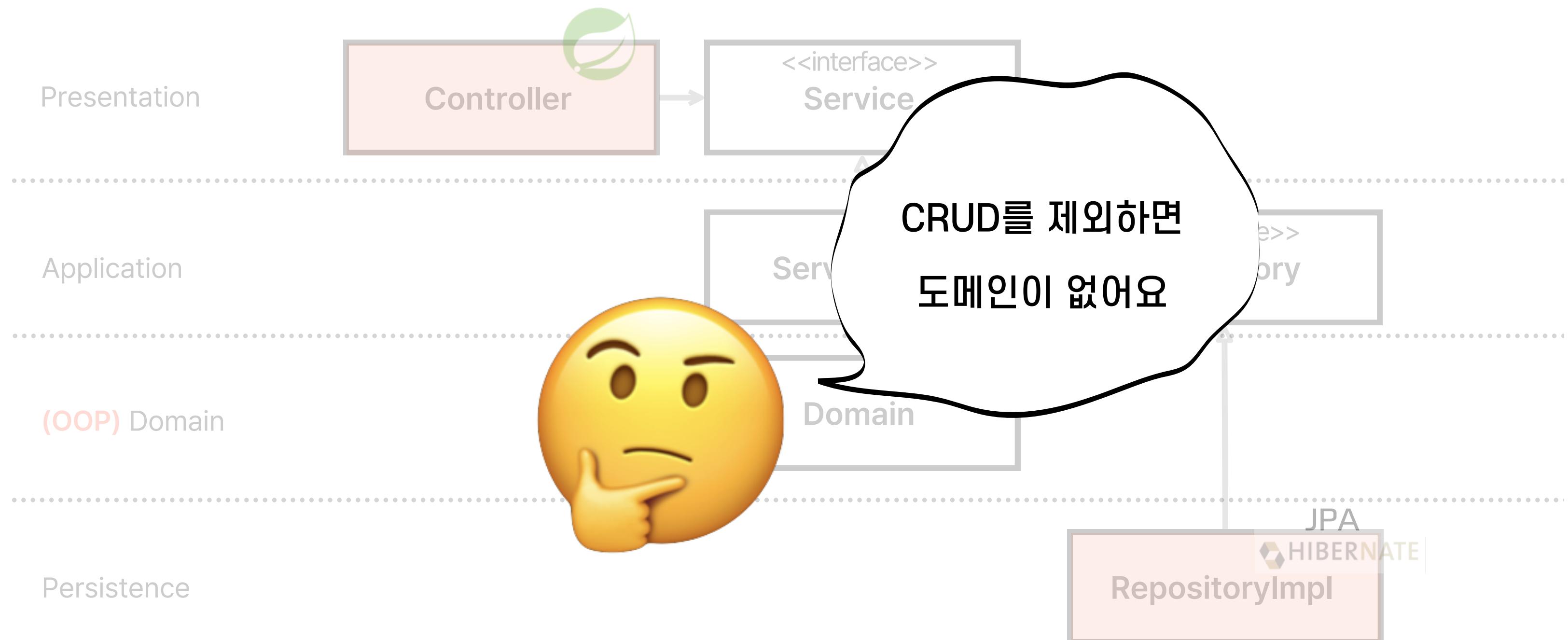


• 도메인이 그만큼 빈약하다는 의미



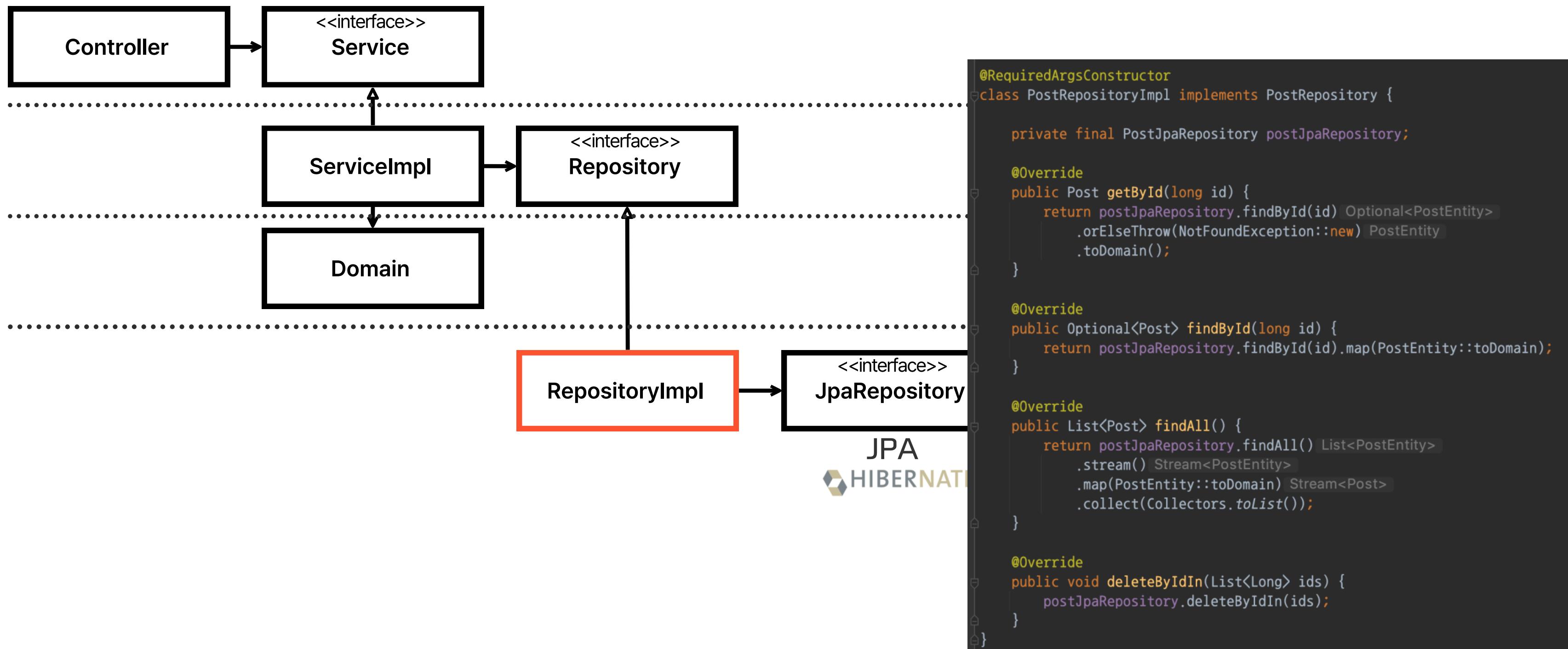
• 서비스의 경쟁력을 의심해봐야 합니다.

낮은 커버리지 걱정



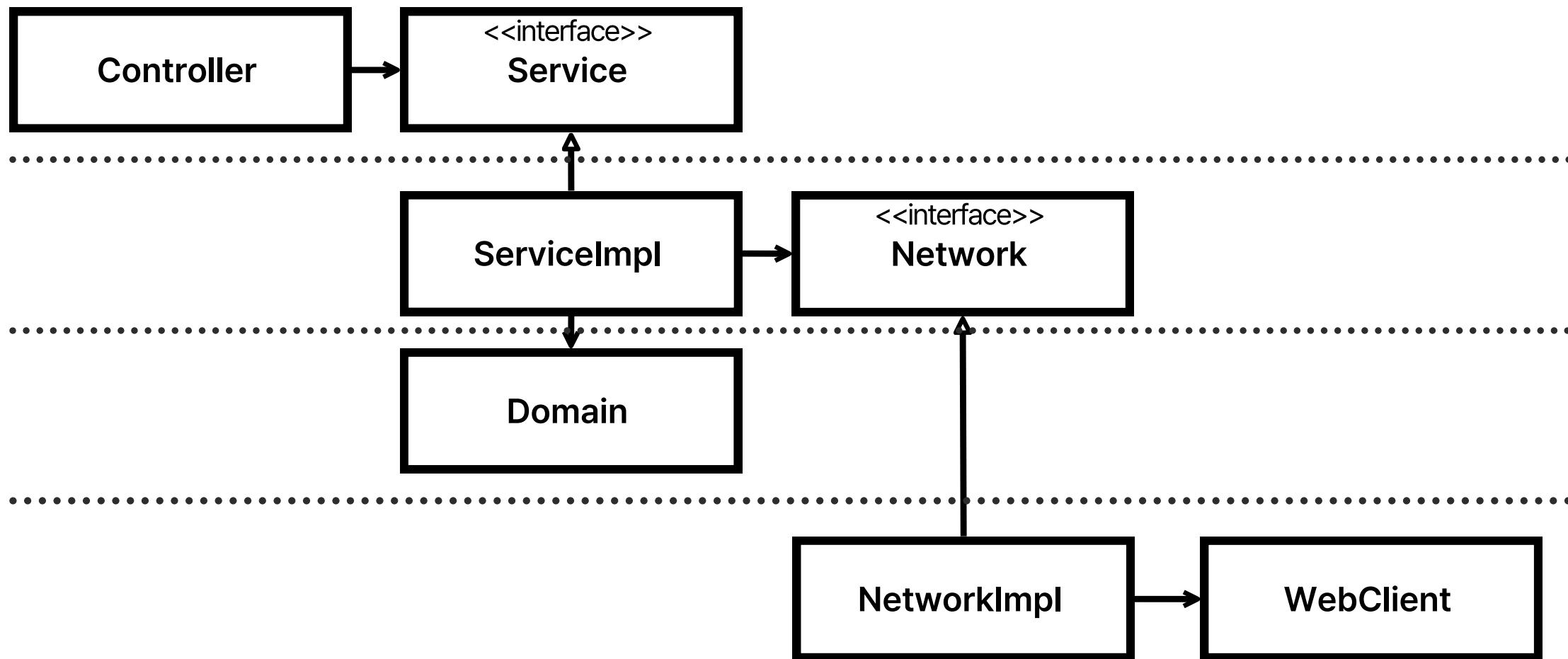
실기전 추가사항 (1)

의존성 역전 원리를 이용하여 외부를 다룰 겁니다 (ex. jpa)



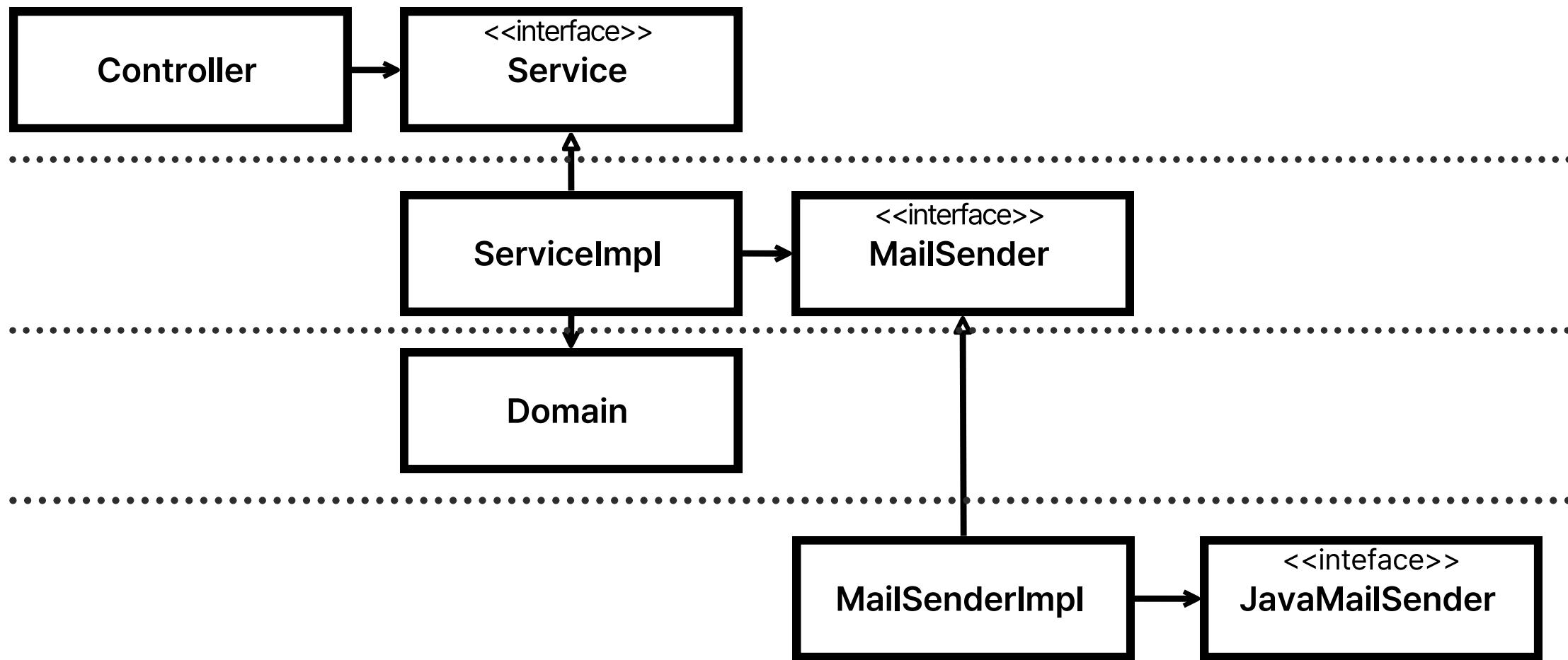
실기전 추가사항 (1)

의존성 역전 원리를 이용하여 외부를 다룰 겁니다 (ex. http 통신)



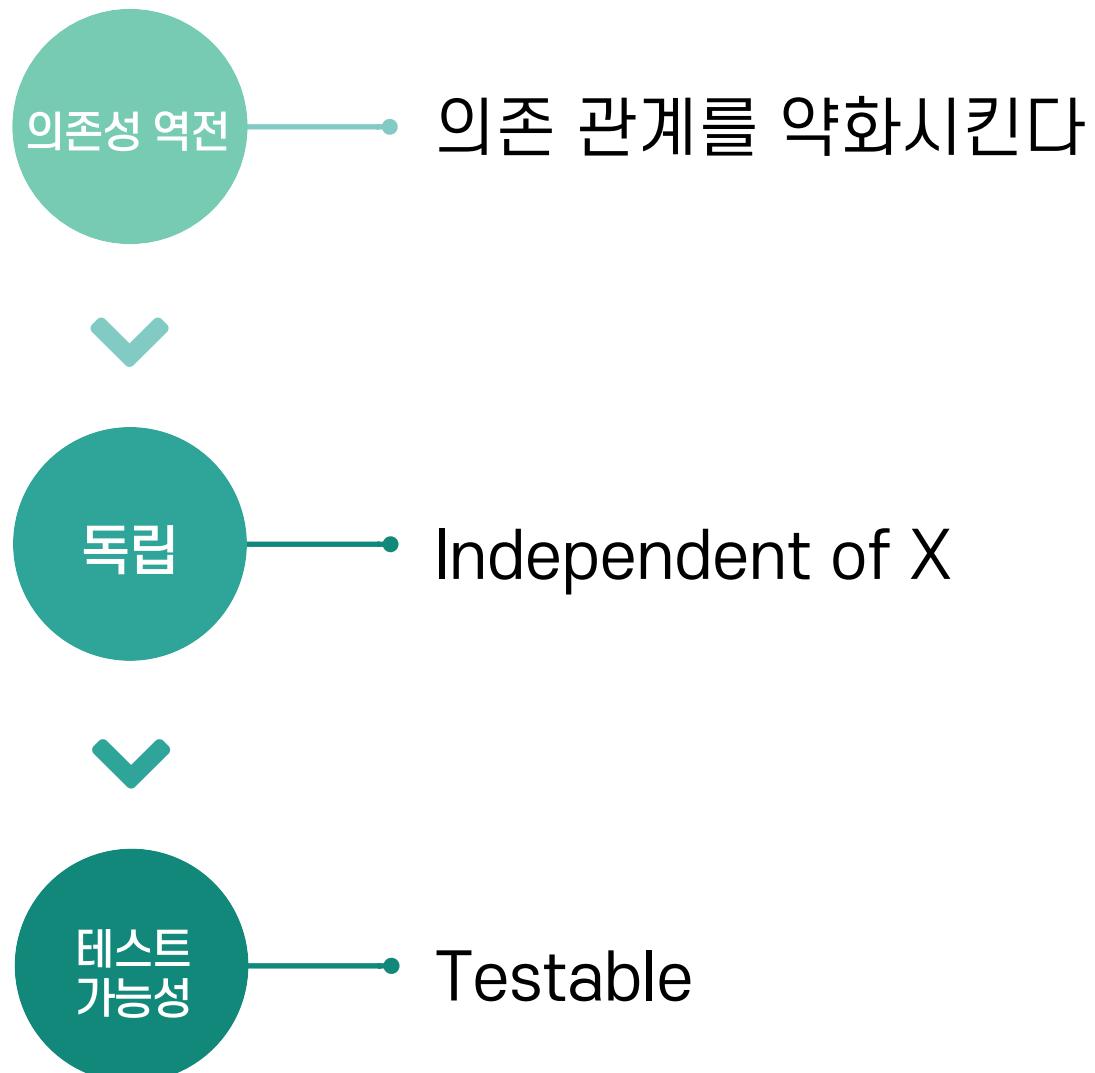
실기전 주의사항 (1)

의존성 역전 원리를 이용하여 외부를 다룰 겁니다 (ex. 이메일 발송)



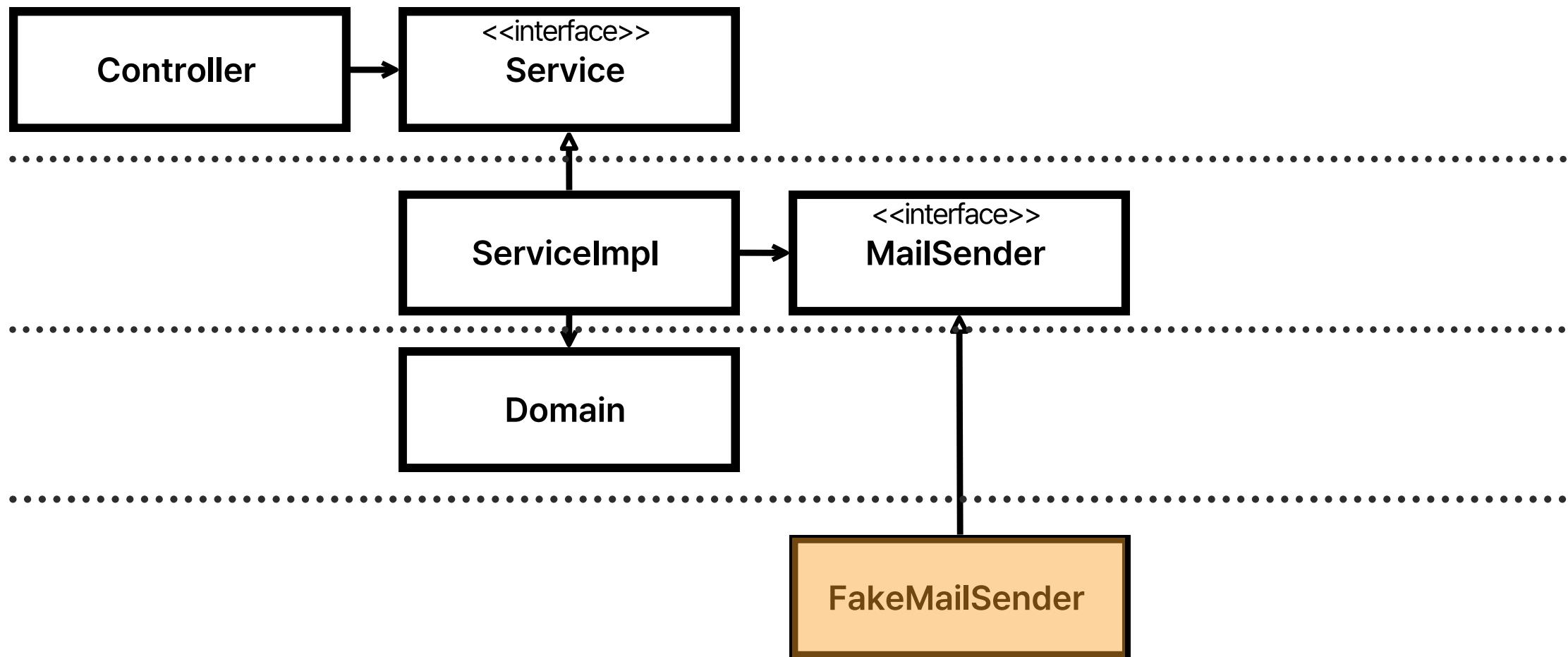
실기전 주의사항 (1)

의존성 역전 원리를 이용하여 외부를 다룰 겁니다



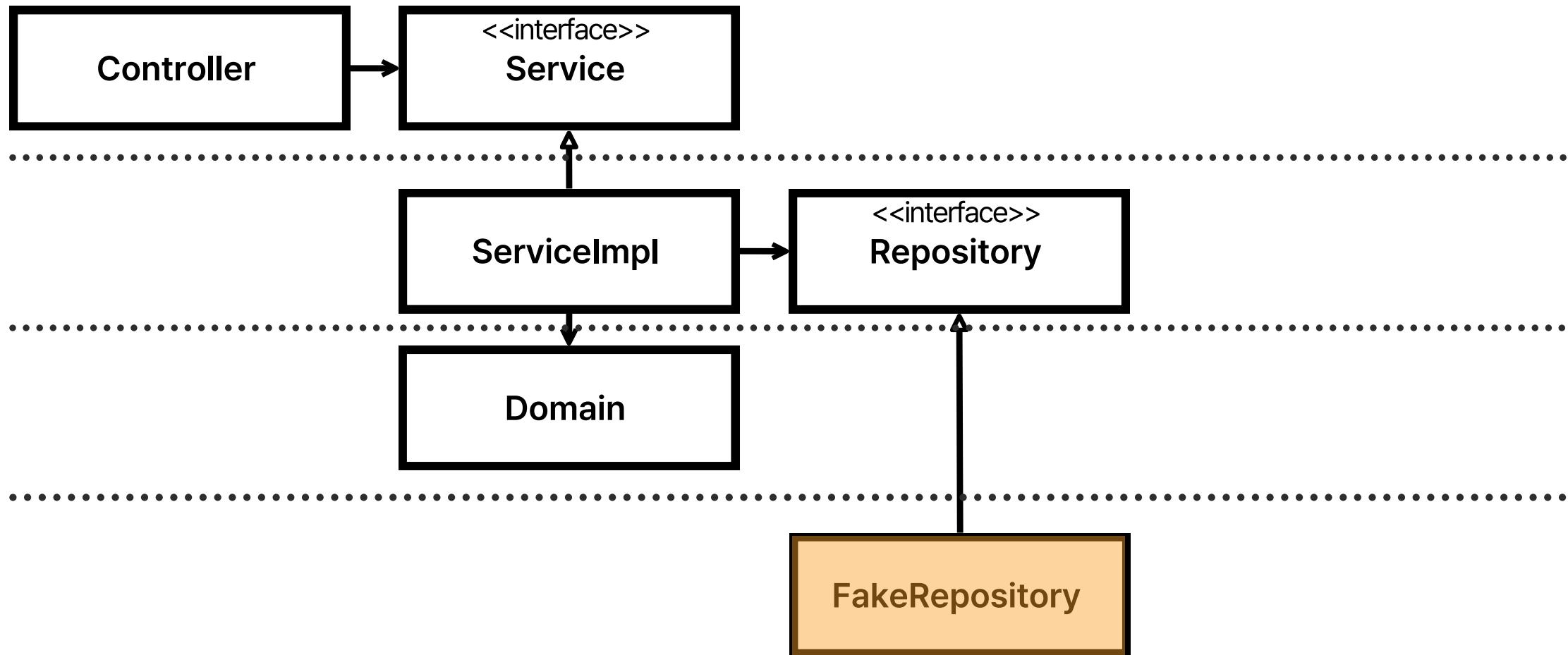
실기전 추가 내용 (1)

필요한 경우 mock으로 치환하여 테스트



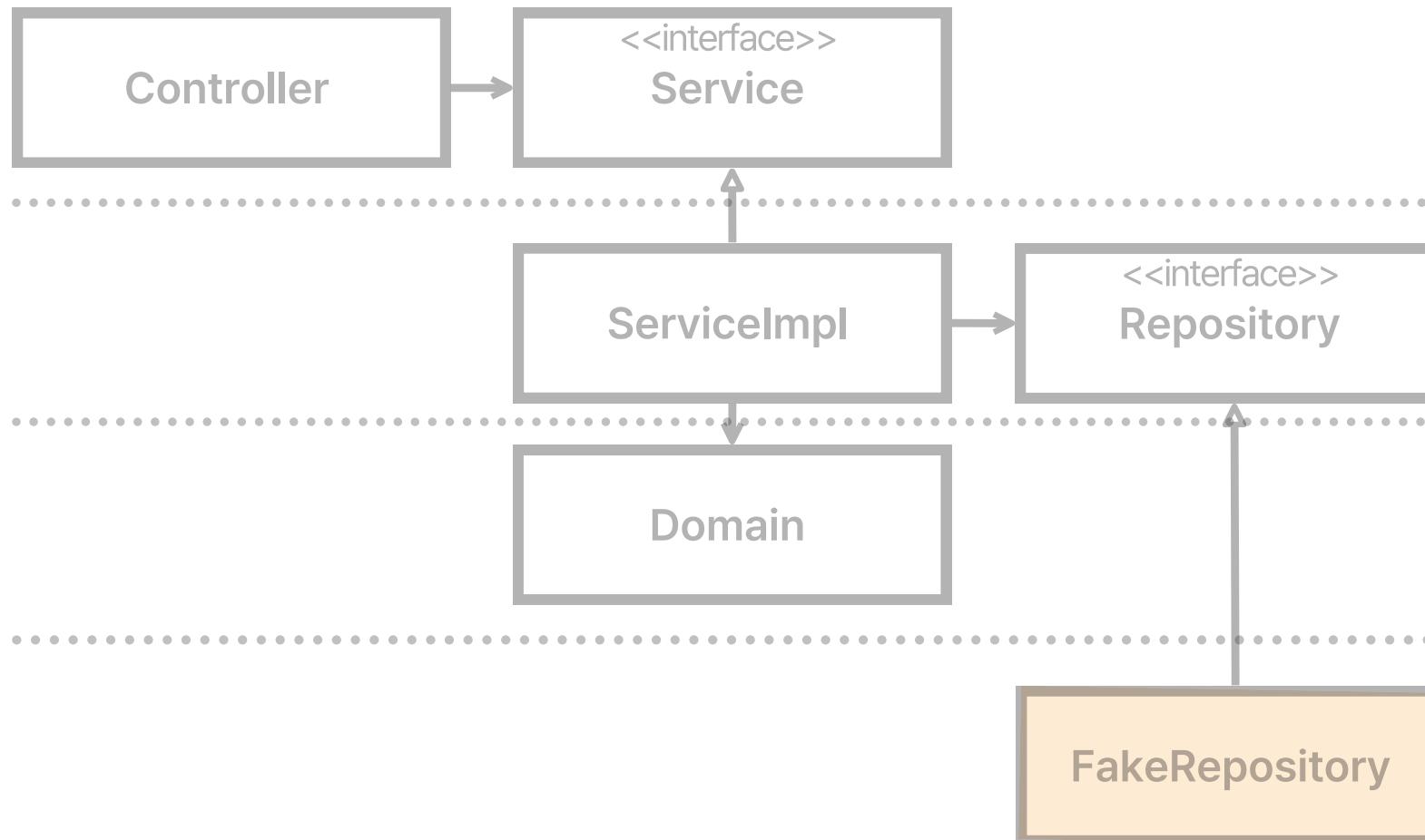
실기전 추가 내용 (1)

필요한 경우 mock으로 치환하여 테스트



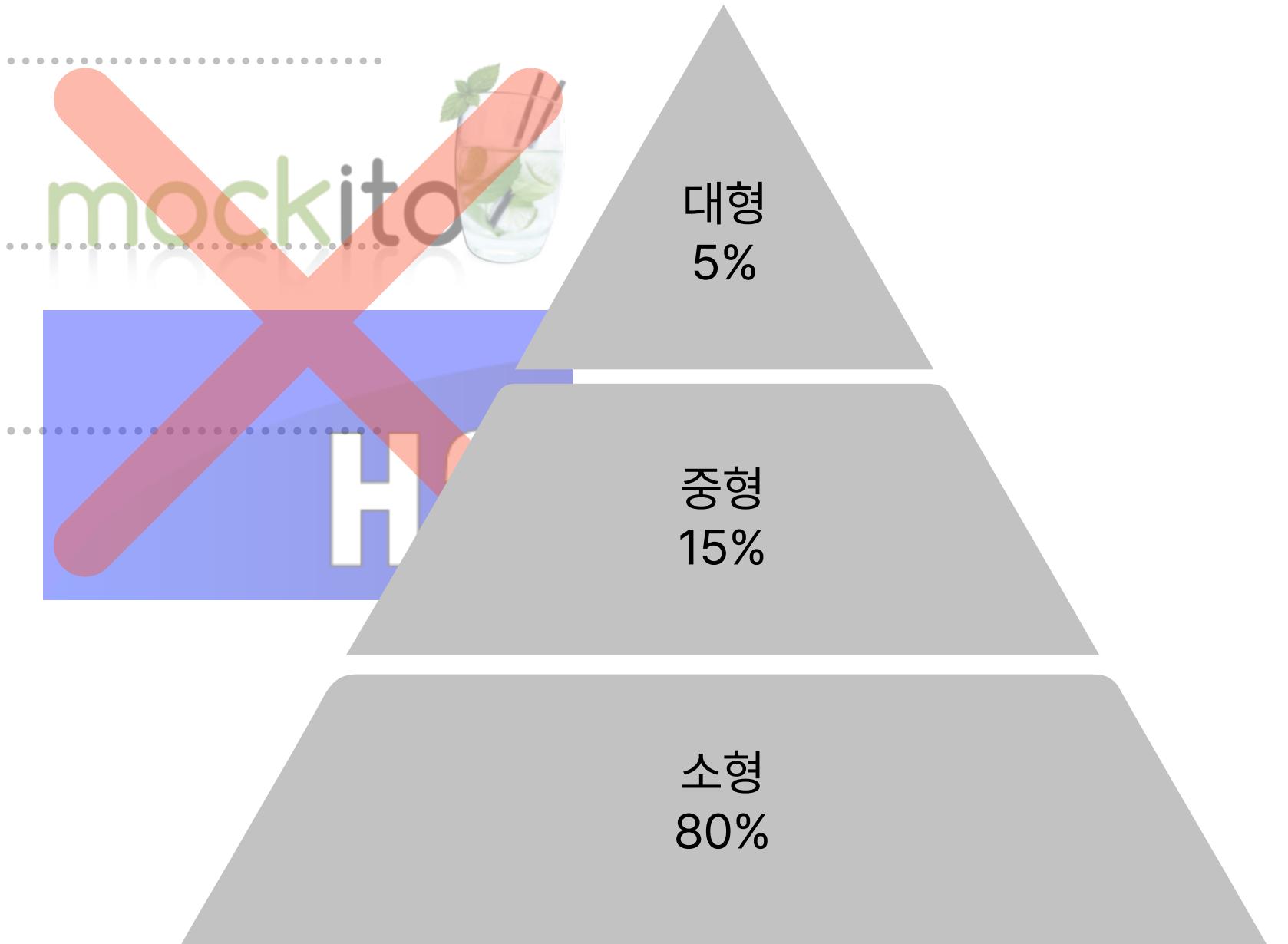
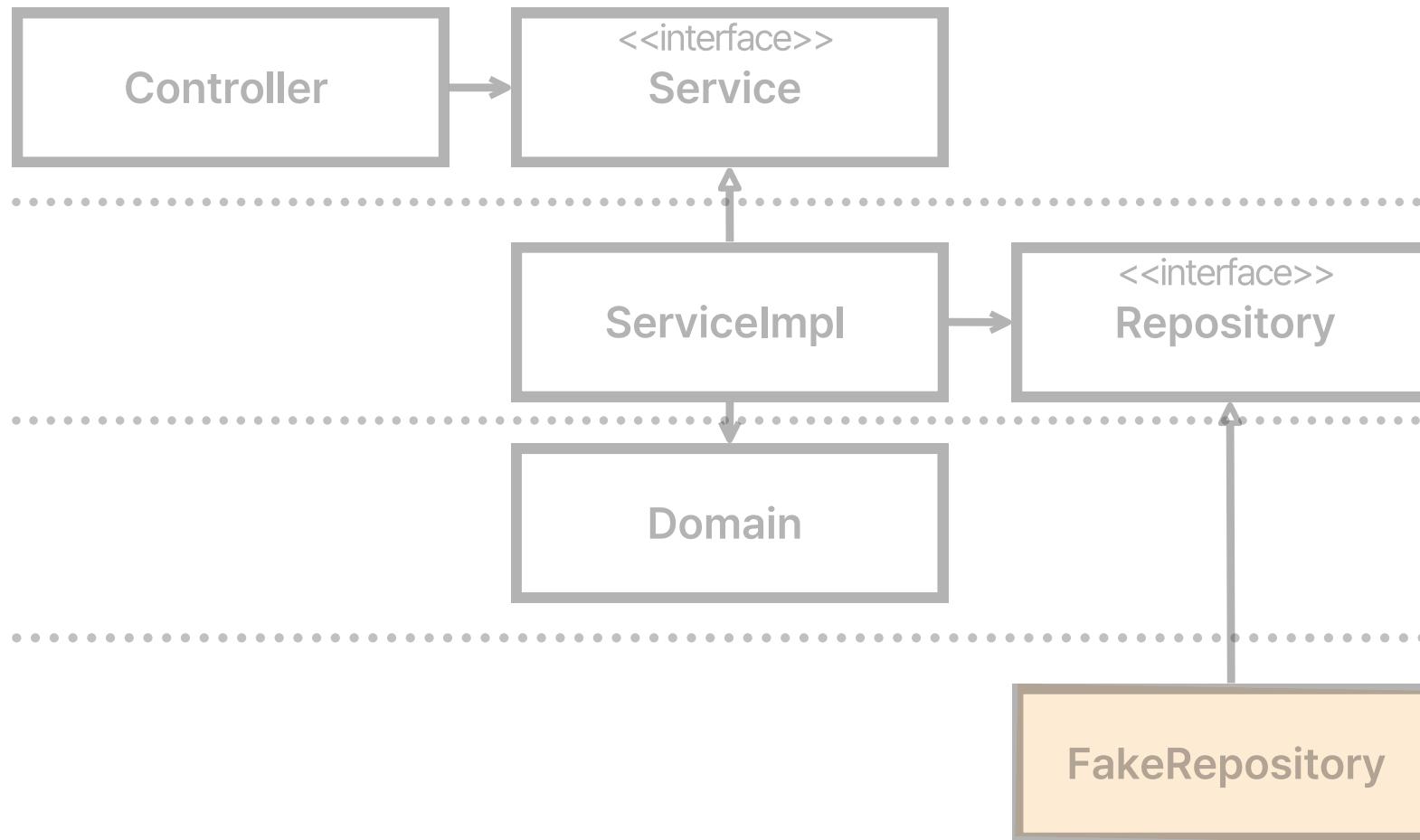
실기전 추가 내용 (2)

don't use Mockito framework



실기전 추가 내용 (2)

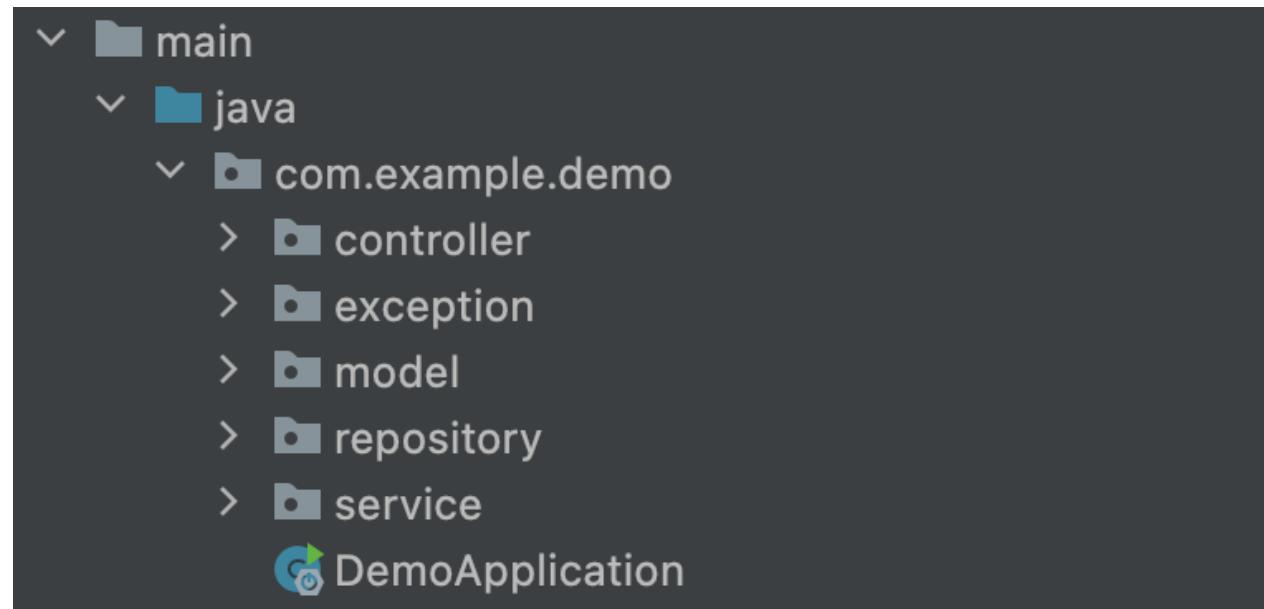
같은 기능을 어떻게 테스트 하느냐에 따라, 테스트의 크기가 결정된다



실기전 추가 내용 (3)

리팩토링 a. 패키지 관리

현재 - layer로 분류



장점

- 구조가 단순하고 처음 접할 때 사용하기 편함

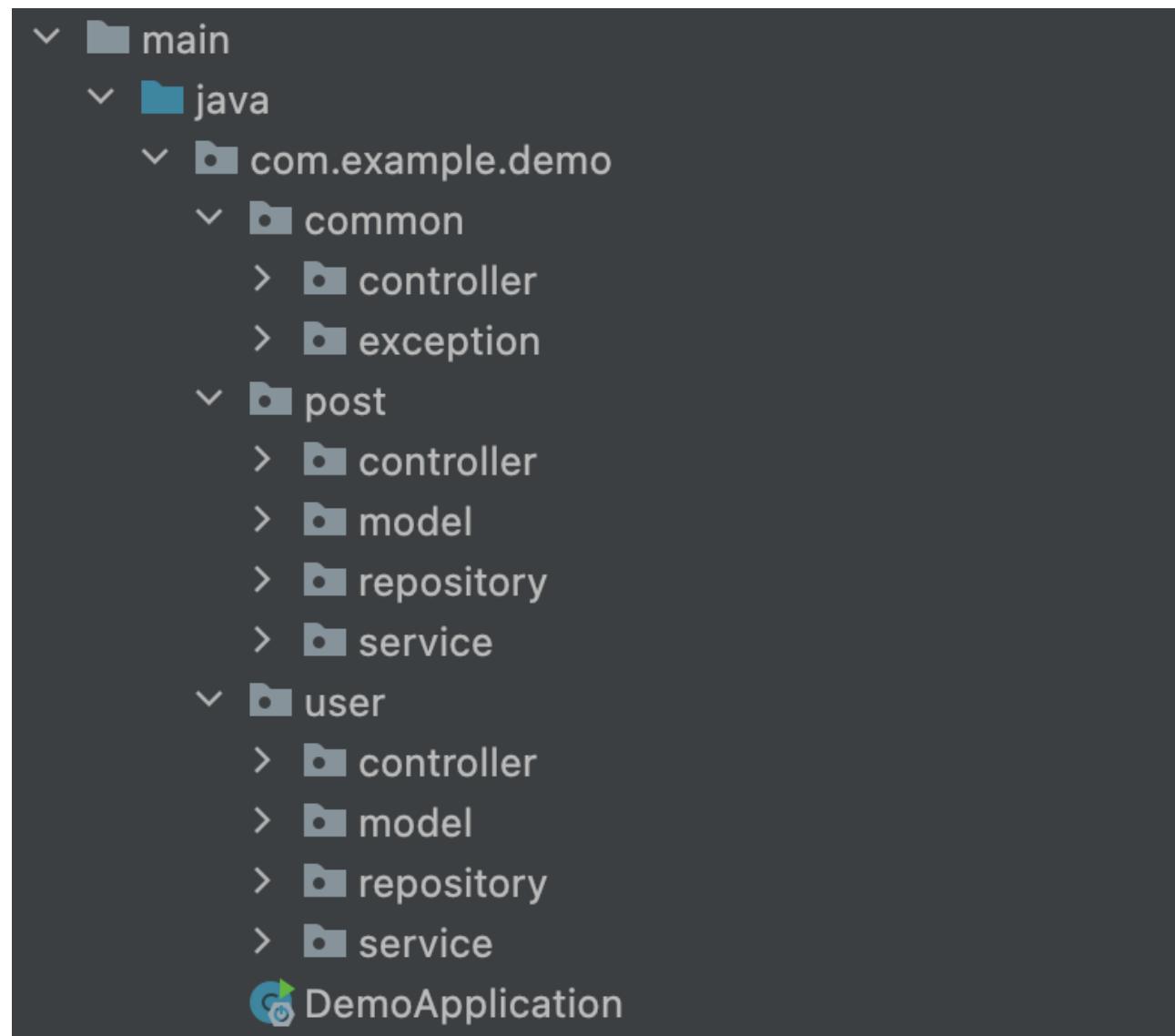
단점

- 도메인이 눈에 보이지 않음
- 동시 작업이 떨어짐
- 사실상 경계, 의존성을 관리하지 않겠다는 의미

실기전 추가 내용 (3)

리팩토링 a. 패키지 관리

현재 - domain/layer로 분류



장점

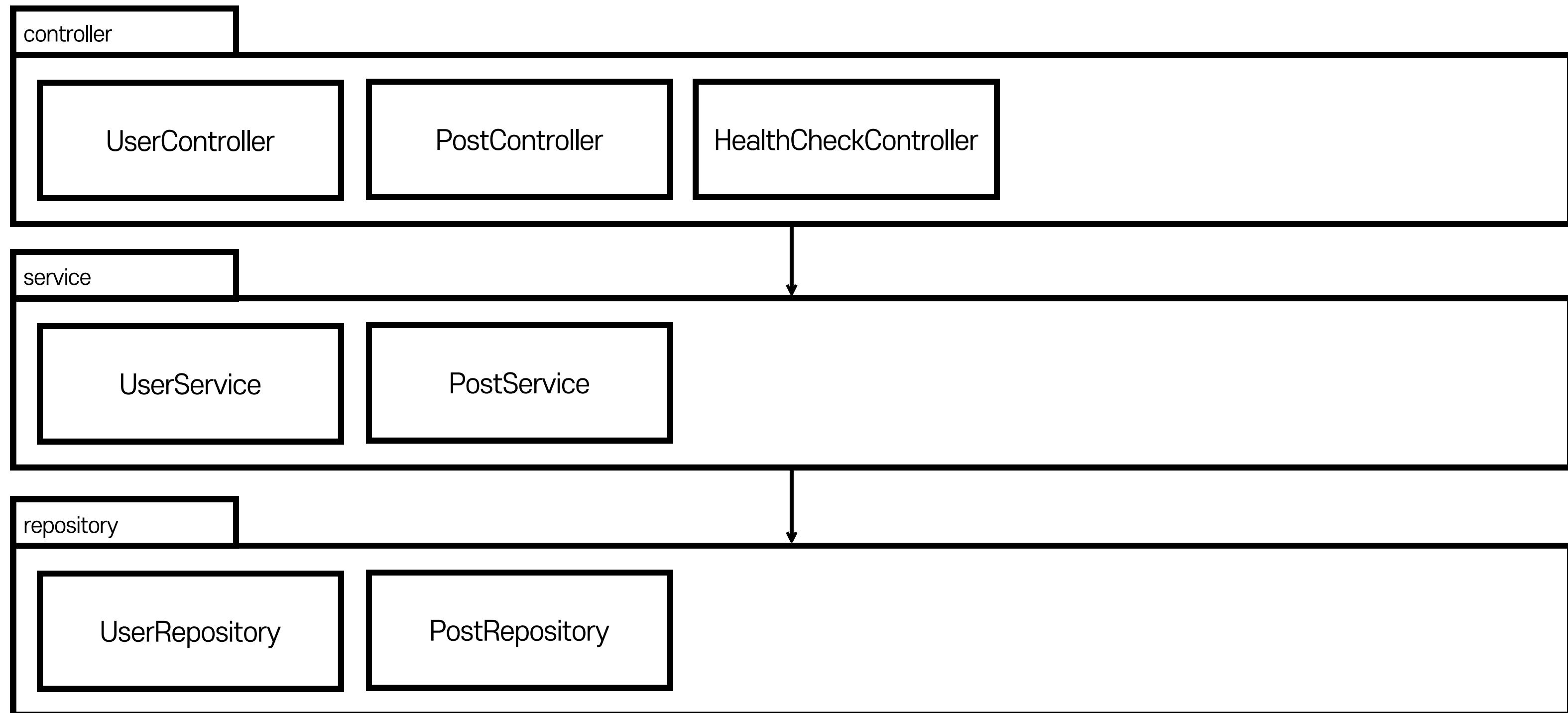
- 어떤 도메인을 다루는 시스템인지 눈에 보임
- 필요에 따라 MSA로 시스템 확장이 가능

단점

- MVC에 익숙한 경우 컴포넌트 찾기가 어색할 수 있음

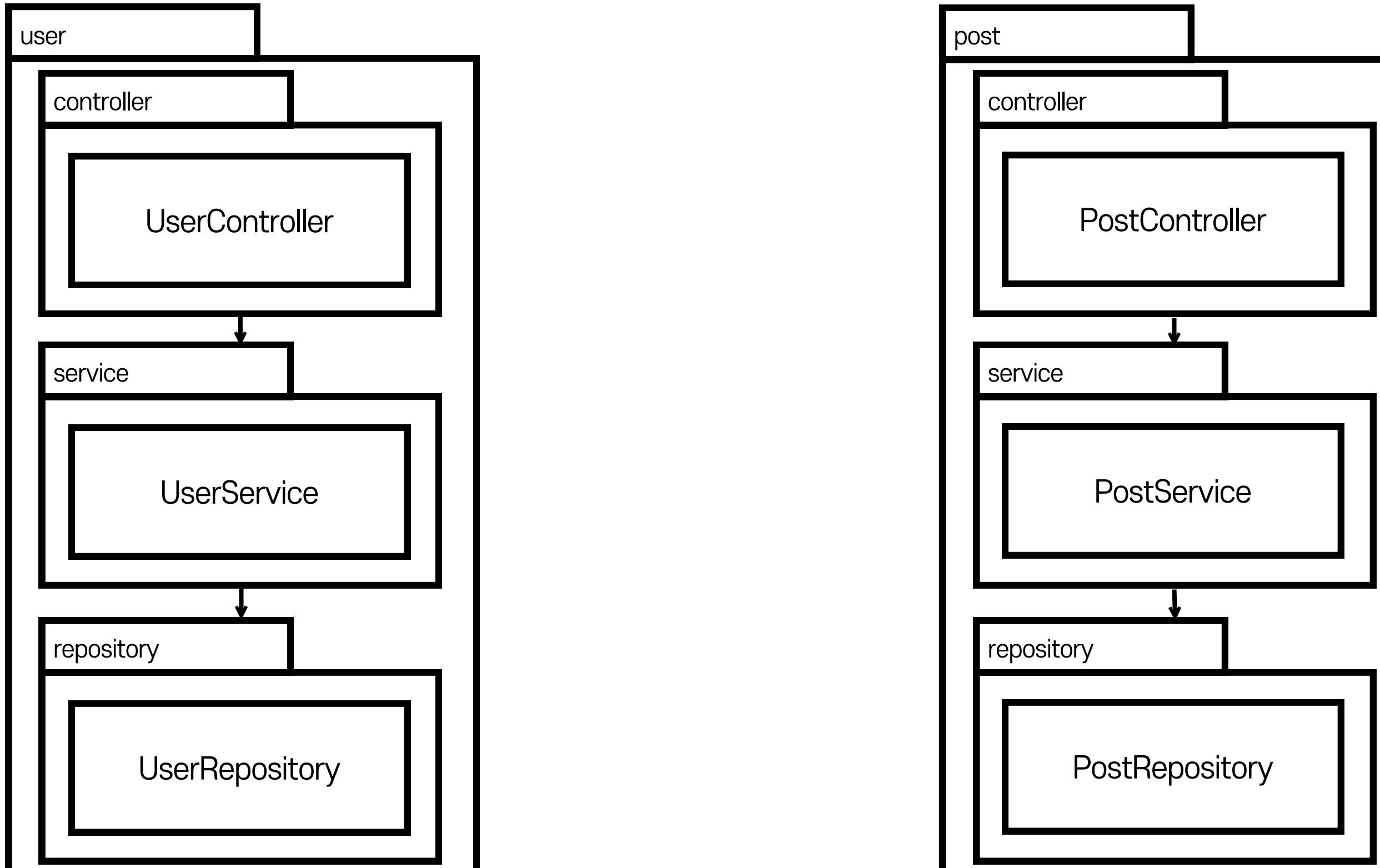
실기전 추가 내용 (3)

리팩토링 a. 패키지 관리



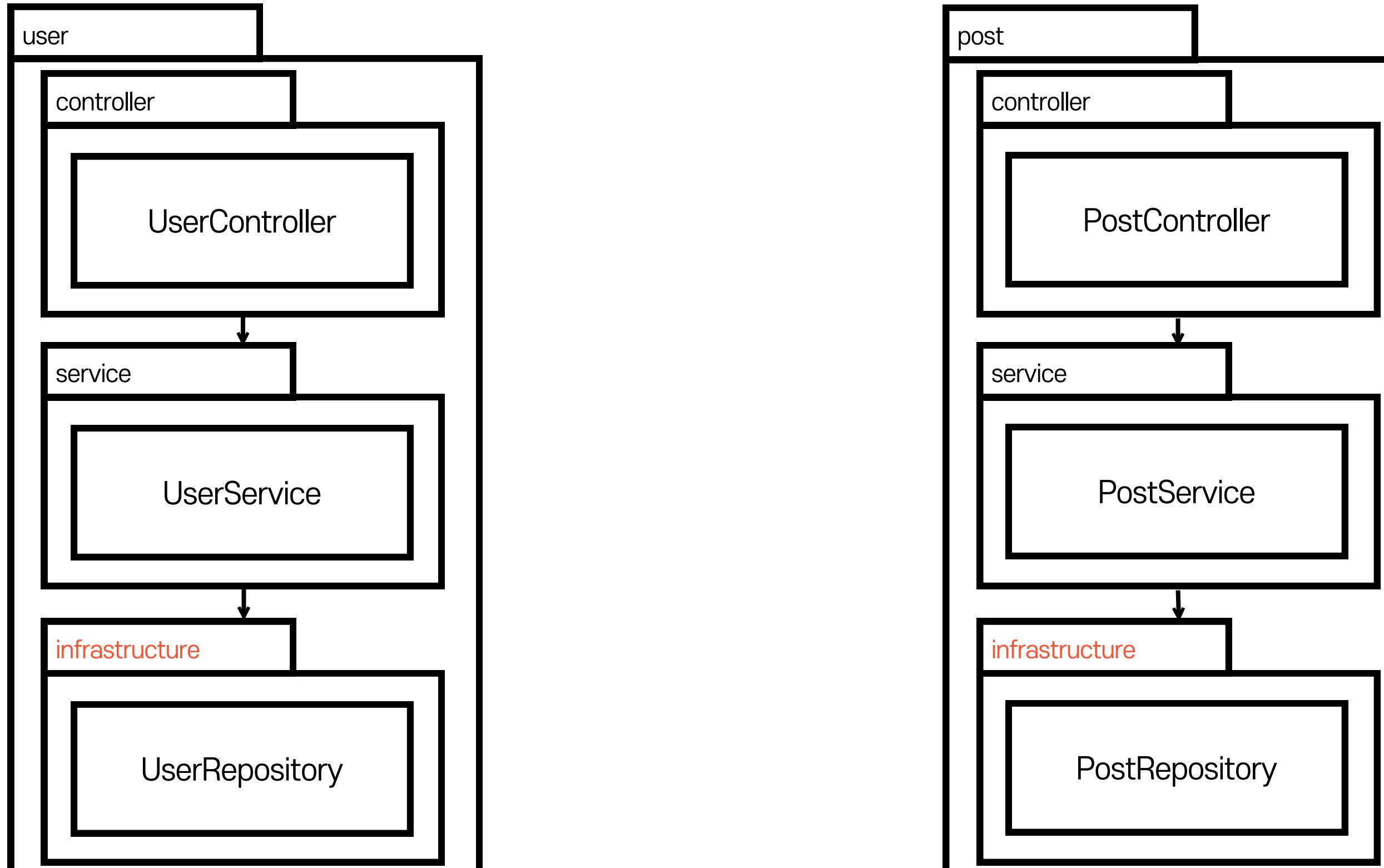
실기전 추가 내용 (3)

리팩토링 a. 패키지 관리



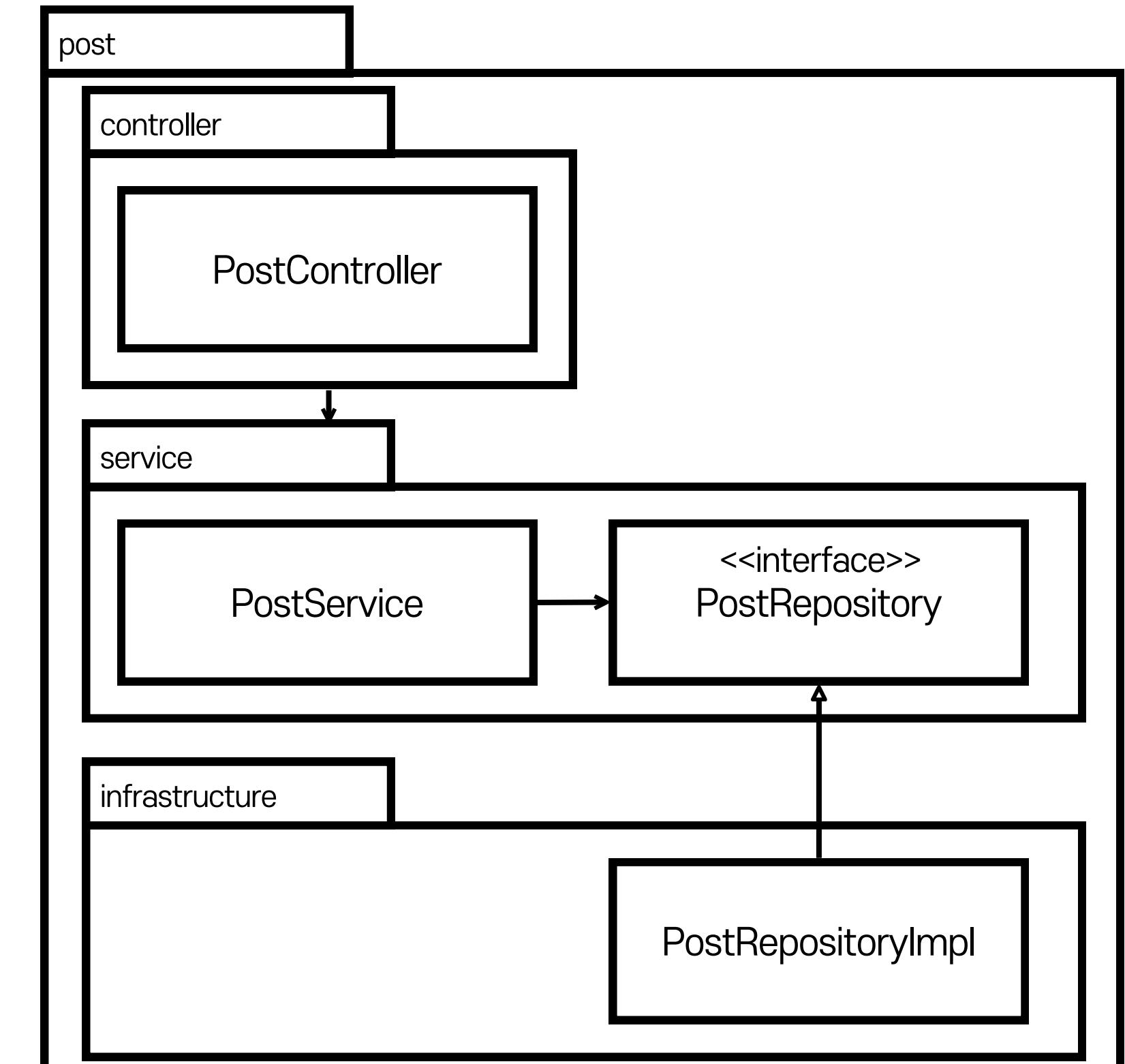
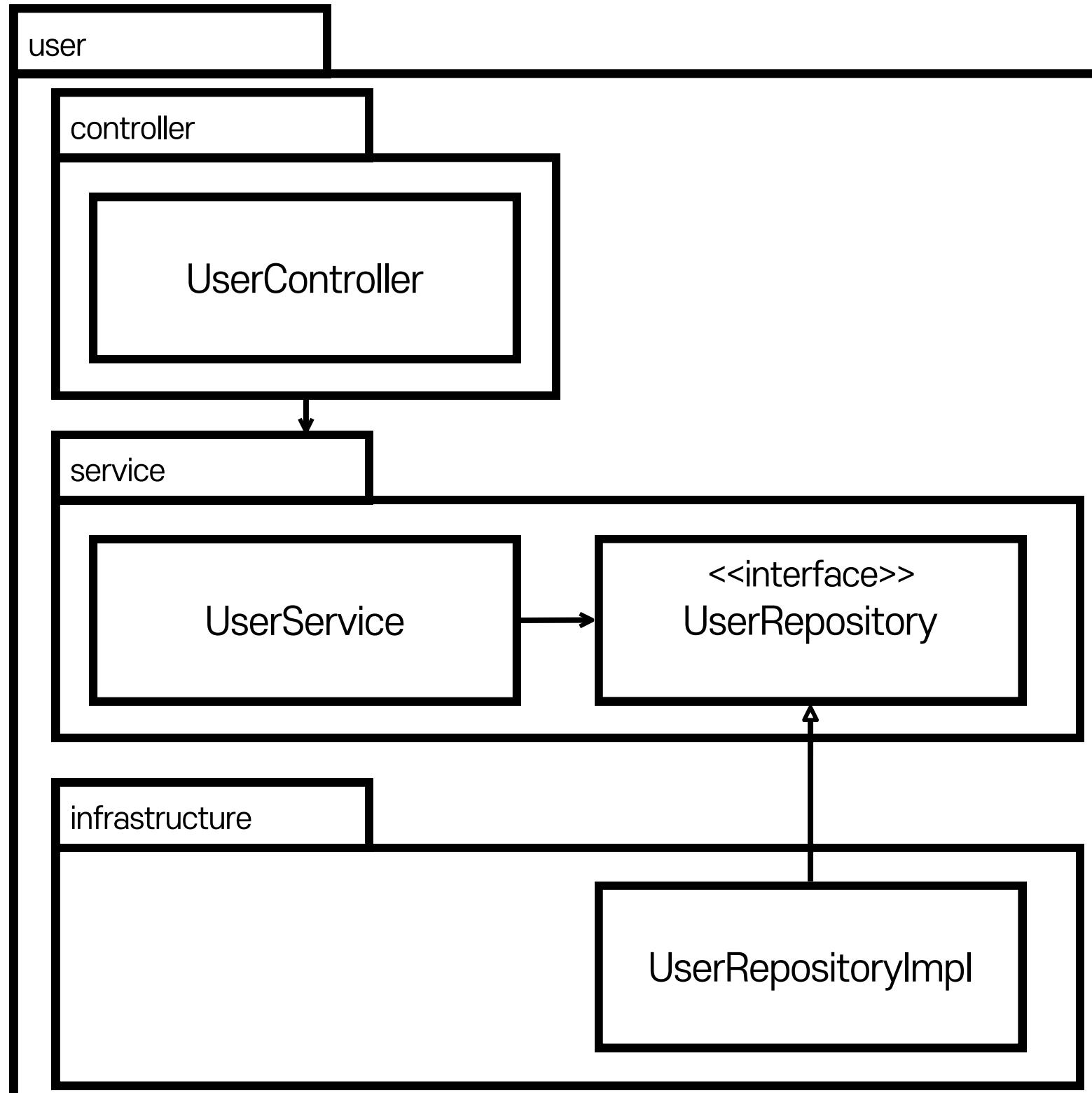
실기전 추가 내용 (3)

리팩토링 a. 패키지 관리 (이름 변경)



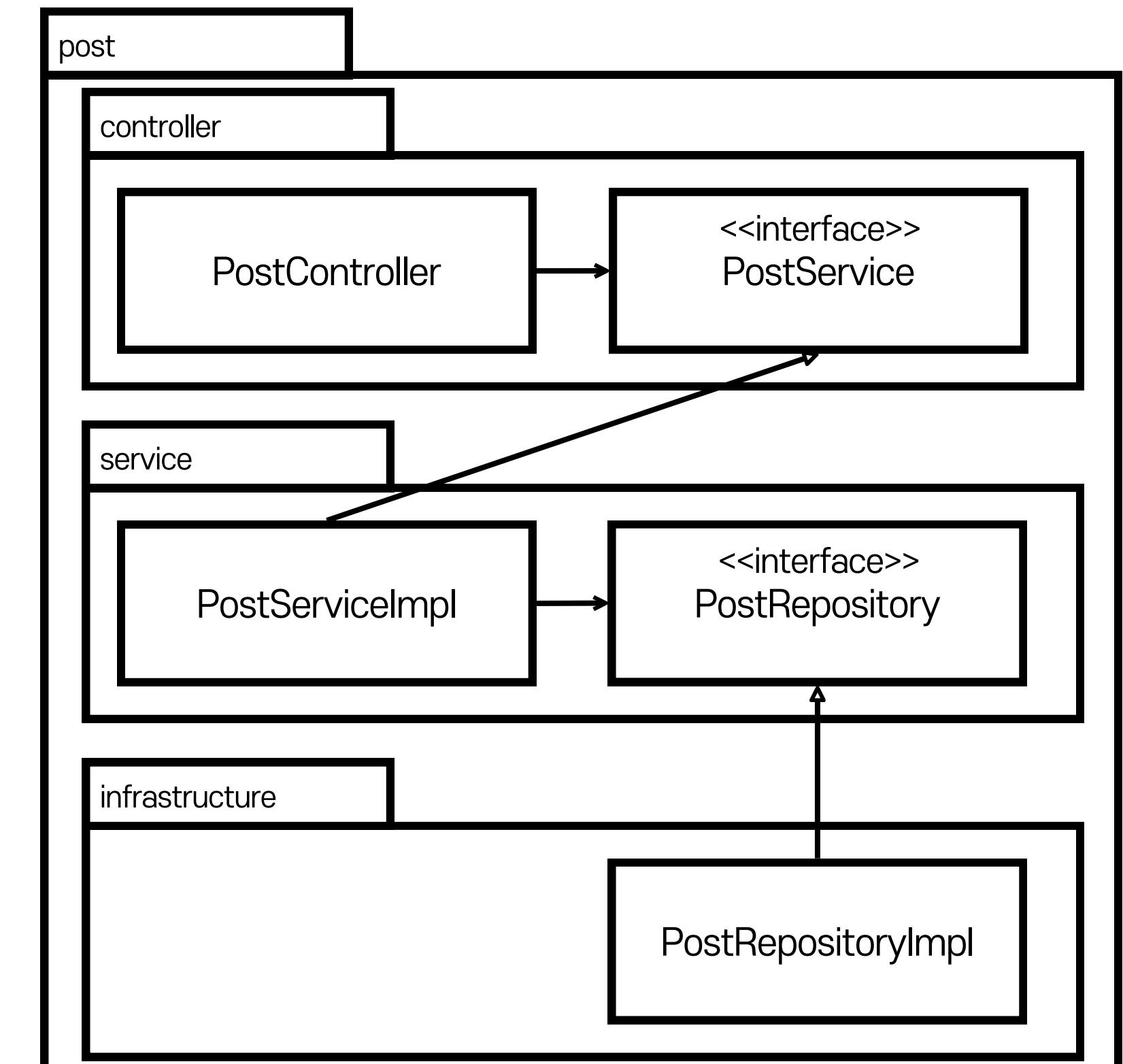
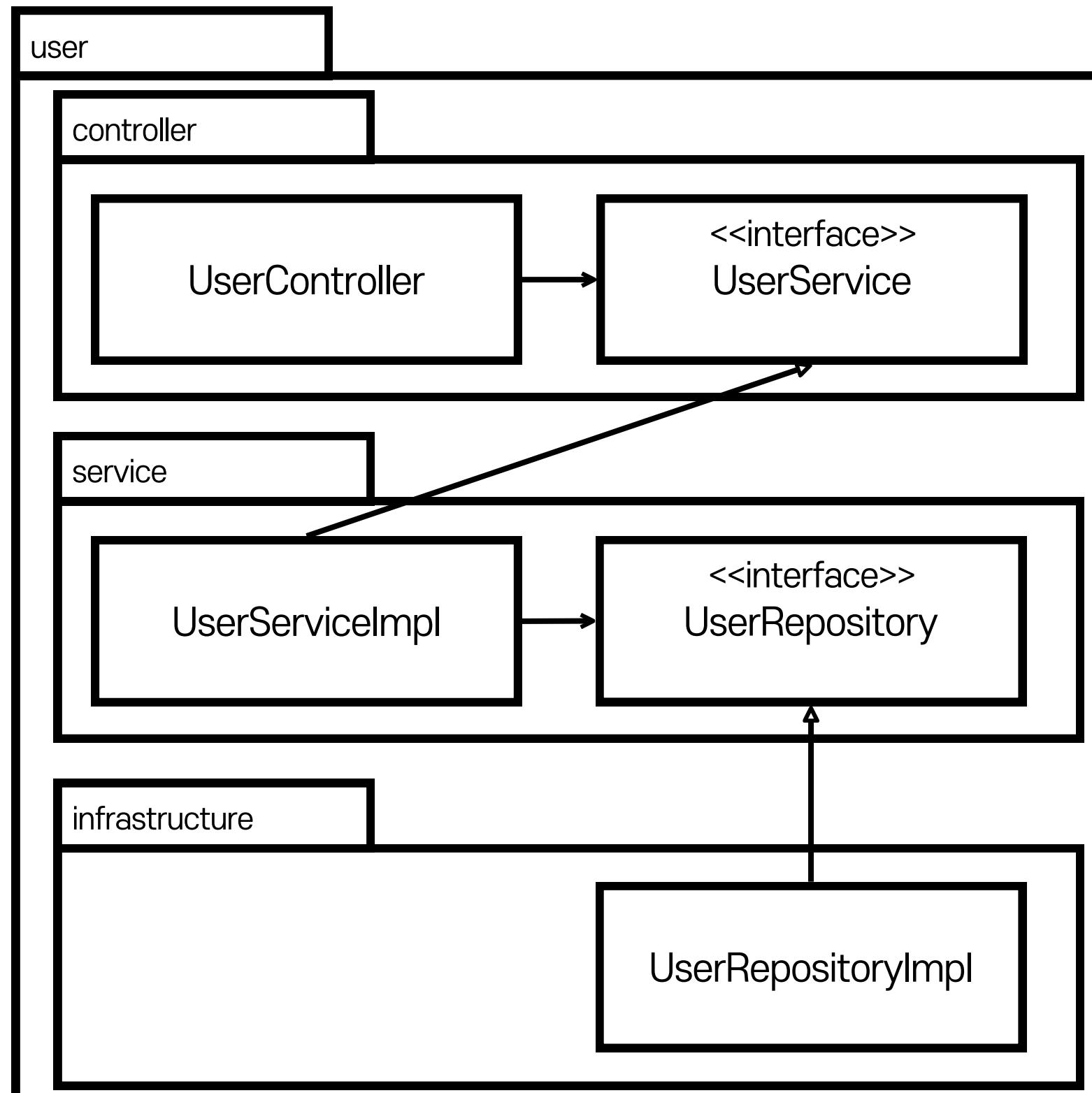
실기전 추가 내용 (3)

리팩토링 a. 패키지 관리 + 의존성 역전



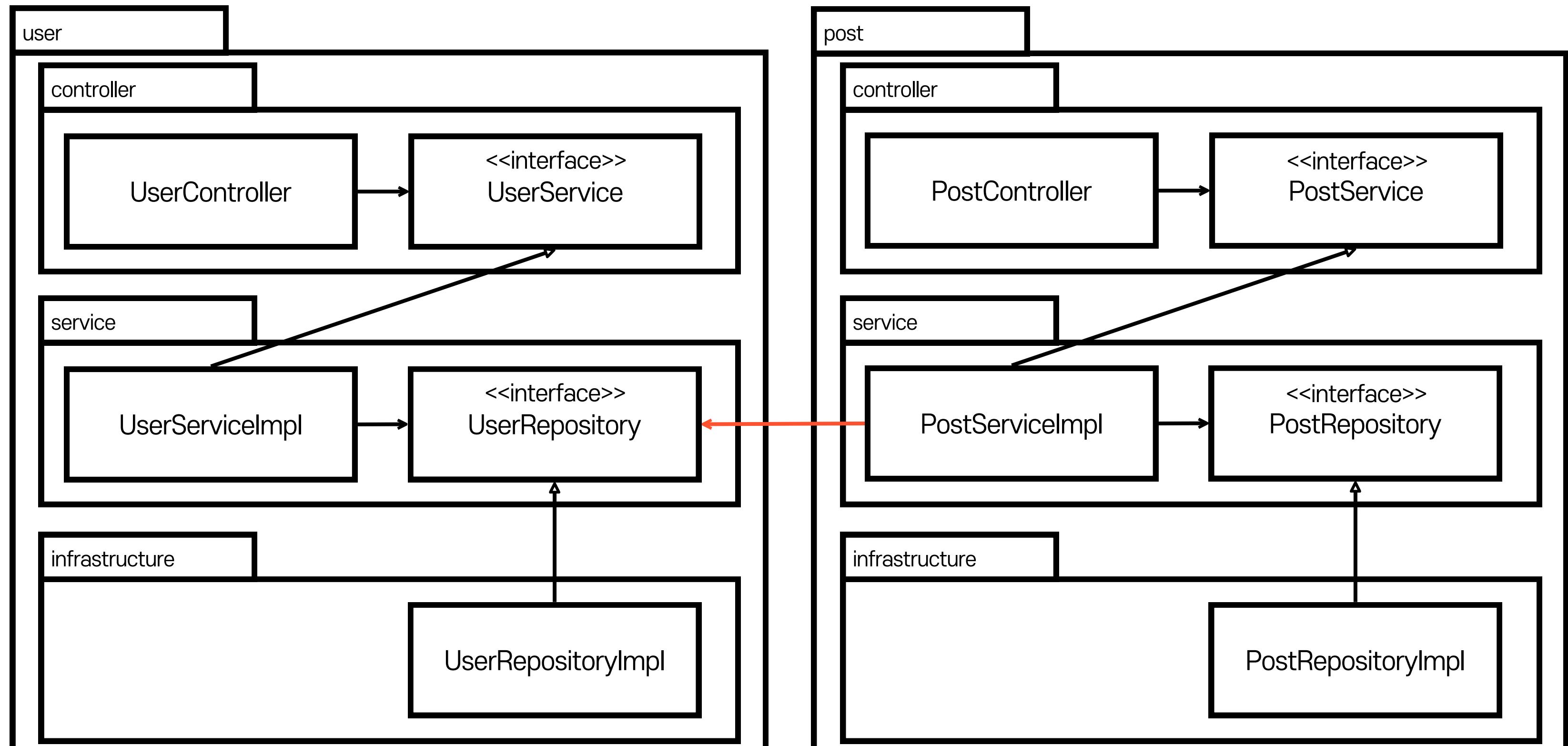
실기전 추가 내용 (3)

리팩토링 a. 패키지 관리 + 의존성 역전



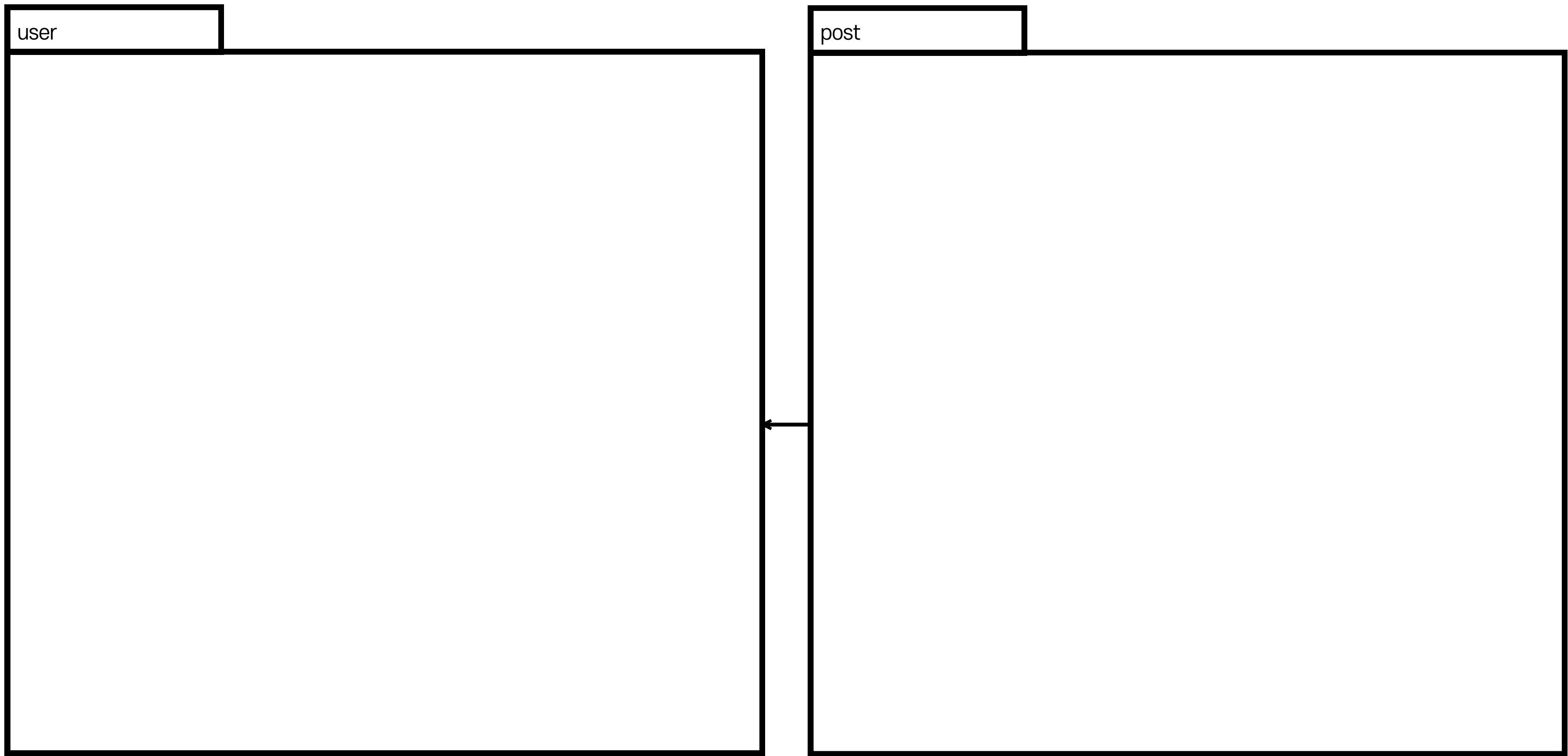
실기전 추가 내용 (3)

리팩토링 a. 패키지 관리 + 의존성 역전



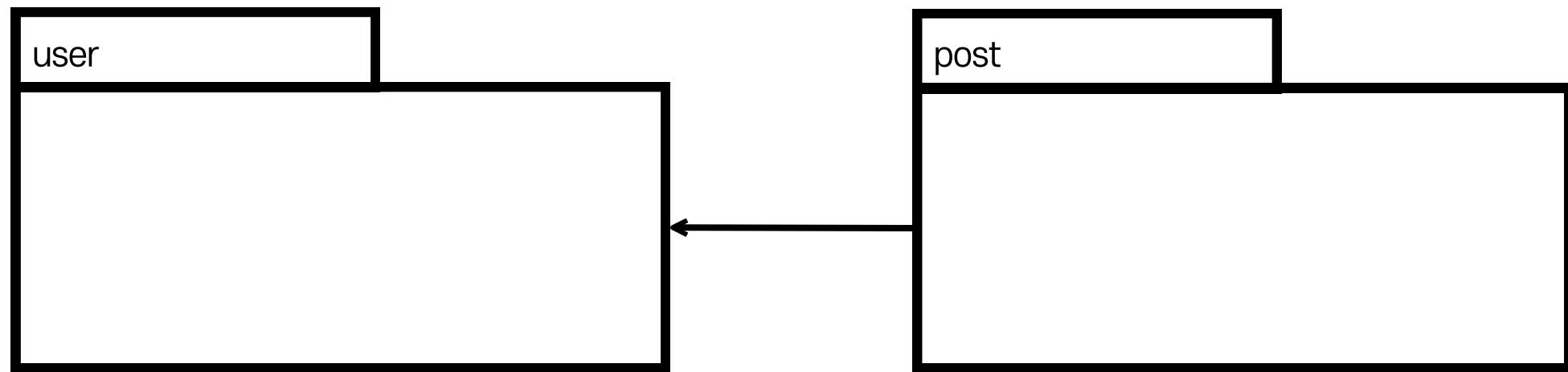
실기전 추가 내용 (3)

리팩토링 b. 패키지 의존성



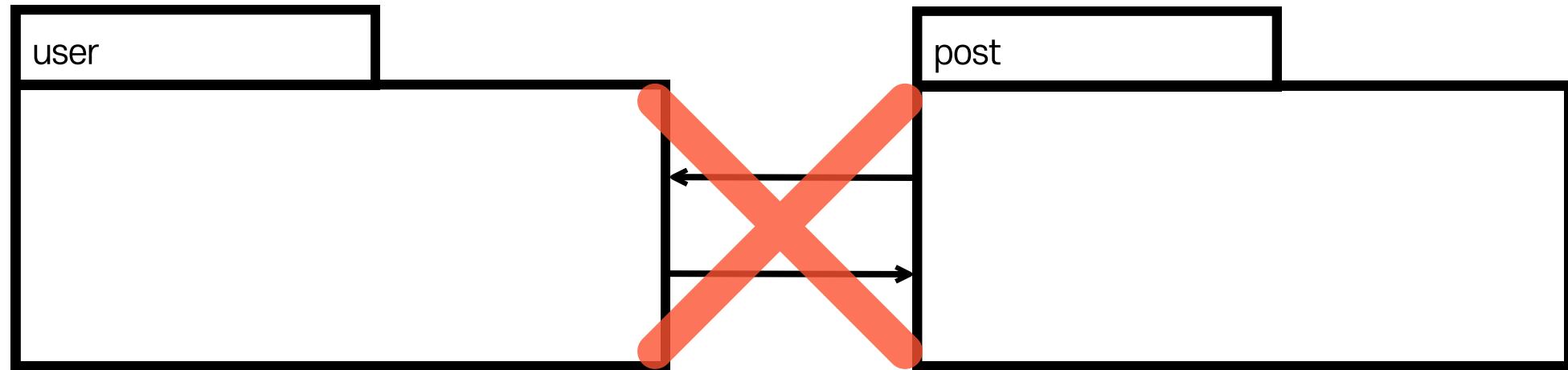
실기전 추가 내용 (3)

리팩토링 b. 패키지 의존성



실기전 추가 내용 (3)

리팩토링 b. 패키지 의존성 : 순환 참조가 생기는지 의식하며 개발해야 합니다

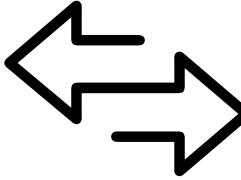


실기전 추가 내용 (3)

리팩토링 c. Jpa 엔티티와 도메인 모델을 분리

Jpa 엔티티 | DB CRUD

```
@Getter  
@Setter  
@Entity  
@Table(name = "users")  
public class UserEntity {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column(name = "email")  
    private String email;  
  
    @Column(name = "nickname")  
    private String nickname;  
  
    @Column(name = "address")  
    private String address;  
  
    @Column(name = "certification_code")  
    private String certificationCode;  
  
    @Column(name = "status")  
    @Enumerated(EnumType.STRING)  
    private UserStatus status;  
  
    @Column(name = "last_login_at")  
    private Long lastLoginAt;  
}
```



도메인 모델 | 비즈니스 모델

```
public class User {  
    private Long id;  
    private String email;  
    private String nickname;  
    private String address;  
    private String certificationCode;  
    private UserStatus status;  
    private Long lastLoginAt;  
}
```

실기전 추가 내용 (3)

리팩토링 d. setter를 없애고 domain / vo으로 로직 이동

```
@Transactional
public UserEntity createUser(UserCreateDto userCreateDto) {
    UserEntity userEntity = new UserEntity();
    userEntity.setEmail(userCreateDto.getEmail());
    userEntity.setAddress(userCreateDto.getAddress());
    userEntity.setStatus(UserStatus.PENDING);
    userEntity.setCertificationCode(UUID.randomUUID().toString());
    userEntity = userRepository.save(userEntity);
    String certificationUrl = generateCertificationUrl(userEntity);
    sendCertificationEmail(userCreateDto.getEmail(), certificationUrl);
    return userEntity;
}

@Transactional
public void login(long id) {
    UserEntity userEntity = userRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("User", id));
    userEntity.setLastLoginAt(Clock.systemUTC().millis());
}

@Transactional
public void verifyEmail(long id, String certificationCode) {
    UserEntity userEntity = userRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("User", id));
    if (!certificationCode.equals(userEntity.getCertificationCode())) {
        throw new CertificationCodeNotMatchedException();
    }
    userEntity.setStatus(UserStatus.ACTIVE);
}
```

실기전 추가 내용 (3)

리팩토링 d. setter를 없애고 domain / vo으로 로직 이동

```
@Transactional
public UserEntity createUser(UserCreateDto userCreateDto) {
    UserEntity userEntity = new UserEntity();
    userEntity.setEmail(userCreateDto.getEmail());
    userEntity.setAddress(userCreateDto.getAddress());
    userEntity.setStatus(UserStatus.PENDING);
    userEntity.setCertificationCode(UUID.randomUUID().toString());
    userEntity = userRepository.save(userEntity);
    String certificationUrl = generateCertificationUrl(userEntity);
    sendCertificationEmail(userCreateDto.getEmail(), certificationUrl);
    return userEntity;
}

@Transactional
public void login(long id) {
    UserEntity userEntity = userRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("User", id));
    userEntity.setLastLoginAt(Clock.systemUTC().millis());
}

@Transactional
public void verifyEmail(long id, String certificationCode) {
    UserEntity userEntity = userRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("User", id));
    if (!certificationCode.equals(userEntity.getCertificationCode()))
        throw new CertificationCodeNotMatchedException();
    userEntity.setStatus(UserStatus.ACTIVE);
}
```

도메인 모델 | 비즈니스 모델

```
public class User {
    private Long id;
    private String email;
    private String nickname;
    private String address;
    private String certificationCode;
    private UserStatus status;
    private Long lastLoginAt;
}
```

실기전 추가 내용 (3)

리팩토링 d. setter를 없애고 domain / vo으로 로직 이동

도메인 모델 | 비즈니스 모델

```
public class User {  
    private Long id;  
    private String email;  
    private String nickname;  
    private String address;  
    private String certificationCode;  
    private UserStatus status;  
    private Long lastLoginAt;  
}
```



그리고 여기를 테스트 할 겁니다

실기전 추가 내용 (3)

리팩토링 e. CQRS (Command and Query Responsibility Segregation)

- “ 명령과 질의의 책임 분리
- “ 메소드를 명령과 질의로 나누자. (더 넓게는 클래스까지도)

실기전 추가 내용 (3)

명령(Command)

“ 상태를 바꾸는 메소드

- 명령 메소드는 void 타입이어야 합니다.
- 편의상 명령 메소드가 종종 return this 하는 경우도 있는데, 이렇게 되서도 안됩니다.

실기전 추가 내용 (3)

질의(Query)

“ 상태를 물어보는 메소드

- 질의 메소드는 상태를 변경해서는 안됩니다.

실기전 추가 내용 (3)

리팩토링 e. CQRS (Command and Query Responsibility Segregation)

“ 하나의 메소드는 명령이나 쿼리여야하며, 두 가지 기능을 모두 가져서는 안된다. 명령은 객체의 상태를 변경할 수 있지만, 값을 반환하지 않는다. 쿼리는 값을 반환하지만 객체를 변경하지 않는다.

마이클 C. 페더스, 레거시 코드 활용 전략 손대기 두려운 낡은 코드, 안전한 변경과 테스트 기법, 이정문, 심윤보 역, (에이콘출판사, 2018-09-28), 218p

실기전 추가 내용 (3)

리팩토링 e. CQRS (Command and Query Responsibility Segregation)

“ Repository 대신 Reader / Writer

Repository	Object	Type
UserReader	UserReadonly	VO
UserWriter	UserEditable	Editable object

RETURN;

**Testcase
with architecture**

Java/Spring 테스트를 추가하고 싶은 개발자들의 오답노트