OOAD Project Report - Failless

팀원 소개

Team 인원 수: 2명

이름	학번	전공	강점	업무 경험
남윤찬	20223069	소프트웨어학부	많은 협업 개발 경험	모바일 프로그래밍 강의와 교내 해커톤에서 spring boot를 사용해본 경험
신진욱	20223098	소프트웨어학부	팀원 간의 소통 및 갈등 관리	리액트를 활용한 분실물 관리 시스템 프로젝트 개발 경험

1. Project 주제, Vision & Scope

프로젝트 주제

스타트업에 도움을 주는 정보 및 스타트업 대표님들의 경험 공유, 많은 스타트업 대표님들의 문제점 및 궁금점 소통을 통해 실패를 줄일 수 있는 방향을 찾아갈 수 있게 한다.

Project Vision

국내 스타트업 대표와 예비 대표들이 정보 및 인사이트를 얻을 수 있는 칼럼 및 커뮤니티

1. 인사이트 제공: 초기 스타트업 대표들이 창업 과정에 필요한 정보 또는 운영 과정에서 겪는 위기와 실패에 대처할 수 있는 정보를 얻어갈 수 있다.
2. 정보의 집약: 커뮤니티 활동을 통해 스타트업과 관련된 다양한 정보들이 모이게 된다.

Project Scope

1. 프로젝트 개발 스택

   프론트 - 웹사이트 페이지 구현 (React.js)

   RESTAPI 설계 및 연동

   백 - erd 설계, 로컬 서버 구현 (Spring Boot, MySQL)

2. 프로젝트 요구 사항 (Actor & Use Case)

   Actor

   - 회원, 비회원, 매니저(관리자)

   Use Case

아티클 읽기	아티클 작성	커뮤니티 글 읽기	커뮤니티 글 작성	커뮤니티 댓글 작성	대표에게 컨택하기

   - 대표에게 컨택하기 (커뮤니티에서 컨택할 수 있는 기능까지만 구현)

     +a) 컨택 확인하기 - 쪽지로 넘어온 컨택을 읽는 알림 창 (노션의 수신함 같은 쪽지 알림창을 하나의 기능으로)

→ 마이페이지는 없으니 (글 수정, 삭제 기능 아웃) 헤더바 오른쪽에 (로그인 or 로그아웃) / 쪽지함 로고→ 이렇게 배치하고 쪽지함을 클릭하면 컨택 내역들 나오게 하고 각 내역당 답장하기 버튼과 X 삭제 버튼이 있다. 답장하기 버튼을 누른다면 답장 모달 띄우기

→ 이 모달에는 답장하기 기능 (ex: 노션 수신함)

Scenario 도출

- **Main**

```
사용자가 웹에 접속하면 아티클 목록 페이지를 보게 된다.
아티클 목록에서 하나를 선택하면 읽을 수 있도록 아티클 상세 페이지로 넘어간다.

사용자가 커뮤니티 탭을 클릭하면 게시글을 읽고 쓸 수 있는 커뮤니티 페이지로 넘어간다.
게시글 목록에서 한 게시물을 클릭하면 해당 게시글 상세 페이지로 넘어간다.

게시글의 작성자가 스타트업 대표라면, 그 대표에게 컨택 쪽지를 보낼 수도 있다.
또한 클릭한 게시글 상세 페이지 하단에 댓글을 작성할 수도 있다.

사용자는 헤더 영역의 로그인 버튼을 눌러 로그인 / 회원가입을 할 수 있다.
그 옆에 있는 알람 버튼을 클릭하면 수신한 컨택을 확인할 수 있다.

매니저(관리자)는 아티클 페이지에서 새 아티클을 작성할 수 있다.
```

- **Alternate**

```
"비회원 사용자"가
1. 커뮤니티 창에서 게시글에 접근 하거나 글쓰기를 클릭 하면
2. 헤더 영역의 알람 버튼을 클릭하면
→ 로그인이 필요하다는 모달이 뜬다.

로그인이 필요하다는 모달에서
취소하기 버튼을 누르면 창이 닫히기만 하고
로그인 버튼을 클릭하면 로그인 창으로 넘어간다.

"매니저(관리자)"가 아티클 목록 페이지(메인 페이지)에 들어오면
아티클 목록 상단에 글쓰기 버튼이 하나 더 생긴다.
글쓰기를 클릭해 아티클을 작성할 수 있다.

헤더 영역의 로그인 버튼을 클릭하면 로그인 페이지로 넘어간다.
로그인 페이지에서 회원가입 버튼을 누르면 회원가입 페이지로 넘어간다.
회원가입을 완료하면 로그인 페이지로 돌아오고 로그인을 하면 아티클 페이지를 보여준다.

+a) 커뮤니티 페이지에서 아티클 | 커뮤니티 탭 하단에 주제 별로 카테고리가 있고 클릭하면 필터링 된다.
```

2. 기능적인 요구 사항 (Functional Requirements)

Use Case Description

Use case name	아티클 읽기
Scenario	아티클 본문을 독자에게 보여준다.
Triggering event	독자가 한 아티클을 클릭한다.
Brief description	아티클 페이지에서 아티클들을 목록화해서 제목만 보여주고 독자가 아티클을 클릭하면 아티클의 내용을 볼 수 있게 한다.
Actors	비회원, 회원, 매니저(관리자)
Related use cases	없음
Stakeholders	매니저(관리자), 아티클 대상자, 독자
Preconditions	아티클 목록에 한 개 이상 아티클이 있어야 한다.
Post conditions	없음
Flow of activities	- **Actor** 1. 아티클 페이지에서 아티클의 제목 목록을 본다. 2. 아티클 페이지에서 아티클 목록 중 하나의 아티클을 클릭하여 읽는다. - **System** 1.1 아티클들을 제목만 목록화하여 보여준다. 2.1 클릭된 아티클의 제목과 본문을 보여준다.
Exception conditions	없음

Use case name	아티클 작성
Scenario	매니저(관리자)가 아티클을 새로 작성한다.
Triggering event	매니저(관리자)가 아티클 작성 버튼을 클릭한다.
Brief description	매니저(관리자) 아이디로 로그인하여 아티클 페이지에서 아티클 작성 버튼을 클릭하고 아티클을 작성하여 아티클 목록이 업데이트 되게 한다.
Actors	매니저(관리자)
Related use cases	없음
Stakeholders	매니저(관리자)
Preconditions	로그인 시 매니저 권한이 확인되어 아티클 목록에 아티클 작성 버튼이 띄워져야 한다.
Post conditions	아티클 목록이 업데이트 된다.
Flow of activities	- **Actor** 1. 매니저(관리자) 계정으로 로그인한다. 2. 아티클 페이지에서 아티클 작성 버튼을 클릭하여 새로운 아티클을 작성한다. - **System** 1.1 로그인한 계정이 매니저(관리자) 권한을 가지는 지 판단한다. 1.2 매니저(관리자) 계정이 맞으면 아티클 목록 위에 아티클 작성 버튼을 보인다. 2.1 아티클 작성 페이지로 전환한다. 2.2 작성된 아티클 내용을 저장한다. 2.3 아티클 목록을 업데이트 한다.
Exception conditions	없음

Use case name	커뮤니티 글 읽기

Scenario	독자가 커뮤니티 글을 읽으려 한다.
Triggering event	커뮤니티 글을 클릭한다.
Brief description	커뮤니티 목록에 글의 제목만 보이게 나열 되어 있다. 이 중 하나를 클릭하면 글 내용을 볼 수 있는 페이지로 전환된다.
Actors	비회원, 회원, 매니저(관리자)
Related use cases	컨택하기, 댓글달기, 로그인 모달
Stakeholders	커뮤니티 게시글 작성자, 독자
Preconditions	각 게시글마다 제목과 내용이 담겨 있어야 한다. 게시글 목록에 한 개 이상 게시글이 있어야 한다. 게시글 목록이 최근에 쓴 글 순서여야 한다. 로그인 되어있어야 한다.
Post conditions	없음
Flow of activities	- **Actor** 1. 커뮤니티 페이지에서 게시글의 제목 목록을 본다. 2. 커뮤니티 페이지에서 게시글 목록 중 하나의 게시글을 클릭하여 읽는다. - **System** 1.1 게시글들을 제목만 목록화하여 보여준다. 2.1 로그인 유무를 판단한다. 2.2 클릭한 유저가 로그인을 한 유저라면 게시글 제목과 내용을 보여준다.
Exception conditions	비회원이 커뮤니티 글을 읽으려고 한다면 로그인 모달을 띄운다.

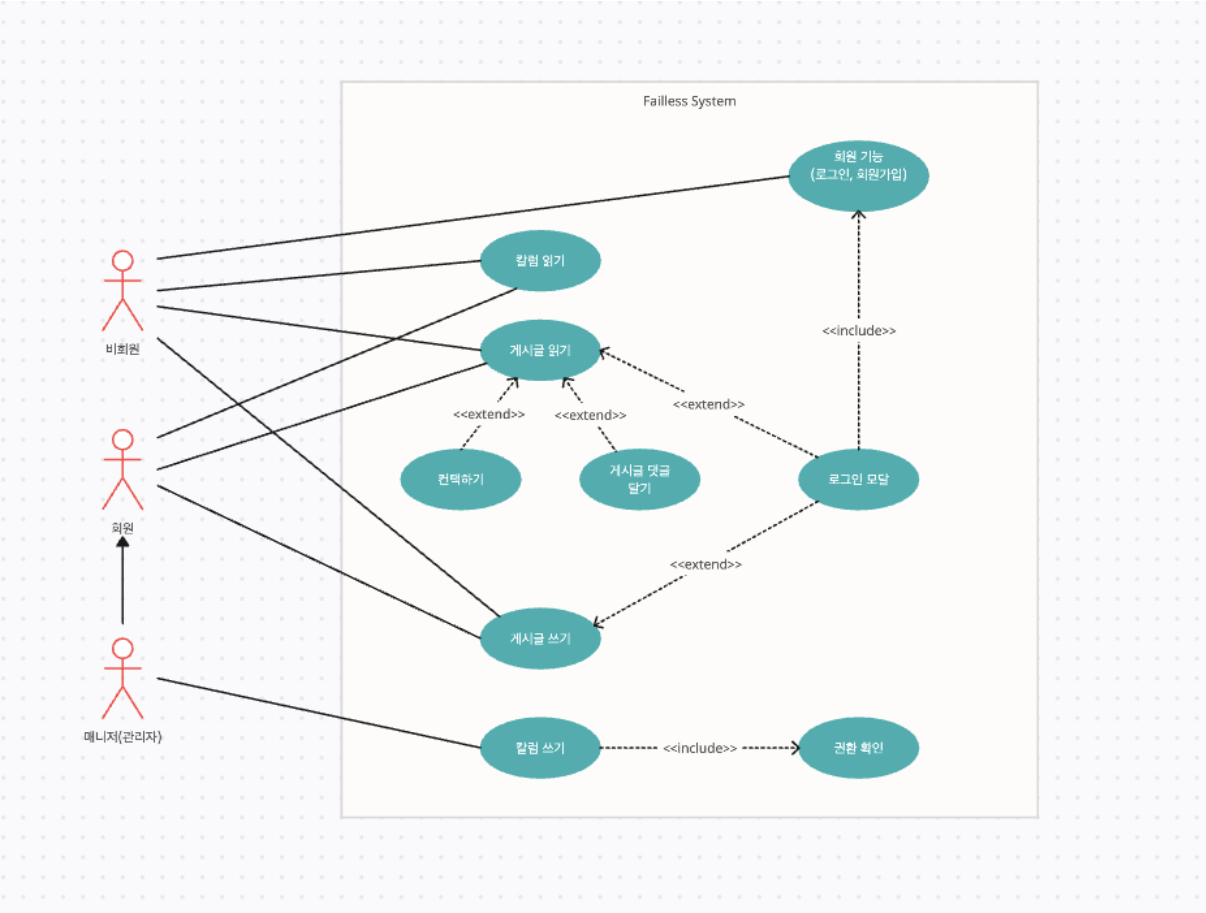
Use case name	커뮤니티 글 작성
Scenario	독자가 새로운 커뮤니티 게시글을 작성한다.
Triggering event	독자가 게시글 작성
Brief description	회원 또는 매니저(관리자)가 커뮤니티 페이지의 게시글 목록 화면에 띄워져 있는 게시물 작성 버튼을 클릭하여 게시글을 작성한다.
Actors	비회원, 회원, 매니저(관리자)
Related use cases	로그인 모달
Stakeholders	게시글 작성자
Preconditions	게시글 작성을 하기 위해 사용자가 로그인이 되어 있어야 한다.
Post conditions	게시글 목록이 최신 쓴 글 순서대로 업데이트 된다.
Flow of activities	- **Actor** 1. 커뮤니티 페이지에서 게시글 작성 버튼을 클릭하여 게시글을 작성한다. - **System** 1.1 로그인 유무를 판단한다. 1.2 클릭한 유저가 로그인을 한 유저라면 게시글 작성 페이지로 전환한다. 1.3 작성된 게시글 내용을 저장한다. 1.4 커뮤니티 게시글 목록을 업데이트 한다.
Exception conditions	비회원이 게시글 작성 버튼을 클릭한다면 로그인 모달을 띄운다.

Use case name	커뮤니티 댓글 작성
Scenario	클릭한 커뮤니티 게시글에서 댓글을 작성한다.

Triggering event	회원 혹은 매니저(관리자)가 게시물에 댓글을 작성한다.
Brief description	회원 혹은 매니저(관리자)가 커뮤니티 페이지에서 특정 게시물을 선택하여 보고 그 게시물에 댓글을 작성한다.
Actors	회원, 매니저(관리자)
Related use cases	게시글 읽기
Stakeholders	댓글 작성자, 게시글 작성자
Preconditions	사용자가 로그인이 되어 있어야 한다.
Post conditions	댓글을 작성하면 댓글 목록이 바로 업데이트 된다.
Flow of activities	- **Actor** 1. 회원이 읽고 있는 게시글에 댓글을 작성한다. - **System** 1.1 회원에게 게시글 페이지에서 댓글창을 제공한다. 1.2 회원이 작성한 댓글이 댓글창에 업데이트한다.
Exception conditions	없음

Use case name	컨택하기
Scenario	열람한 게시글에서 컨택하기 버튼을 눌러 해당 글의 대표에게 쪽지를 보낸다.
Triggering event	커뮤니티 게시글 페이지에서 대표에게 컨택하기 버튼을 누른다.
Brief description	회원 혹은 매니저(관리자)가 선택한 게시물에서 대표에게 컨택하기 버튼을 눌러 해당 글의 대표에게 컨택을 위한 쪽지를 남긴다.
Actors	회원, 매니저(관리자)
Related use cases	게시글 읽기
Stakeholders	컨택 보내는 유저, 컨택 받는 유저
Preconditions	사용자가 로그인이 되어 있어야 한다.
Post conditions	해당 게시글의 대표의 수신함을 업데이트 한다.
Flow of activities	- **Actor** 1. 컨택하기 버튼을 클릭한다. 2. 해당 글의 대표에게 쪽지를 보낸다. - **System** 1.1 회원에게 게시글 페이지에서 컨택하기 버튼을 제공한다. 2.1 해당 게시글의 대표에게 전달할 쪽지 모달을 제공한다. 2.2 작성된 쪽지 내용을 대표 수신함에 전송한다.
Exception conditions	없음

Use Case Diagram



3. 비기능적인 요구 사항 (Non-Functional Requirements) 도출하기

해당 Use Case / 업무	Non-Functional Requirement 내역	Quality	Quality Attributes
아티클 및 커뮤니티 글 읽기	글 상세 내용을 신속하게 화면에 나타나도록 한다.	Performance Efficiency	평균 2초만에 글 상세 내용이 화면에 나타나도록 한다.
댓글 업데이트	댓글을 작성하면 신속하게 댓글 목록을 업데이트 한다.	Performance Efficiency	댓글을 작성 하자마자 댓글 목록이 업데이트 되게 한다.

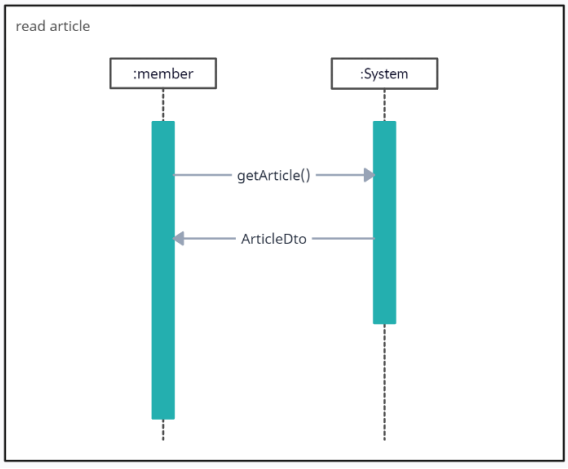
4. Domain Model 산출물

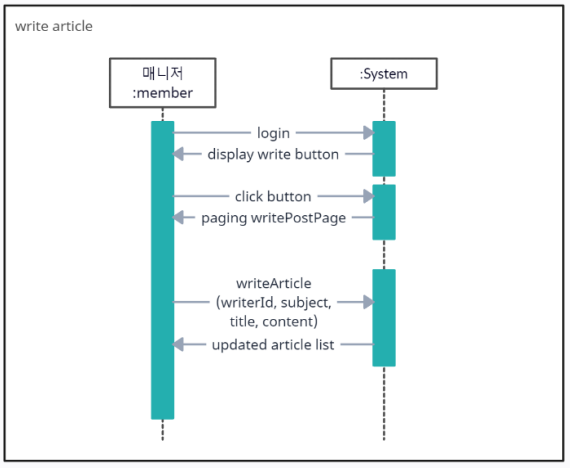
Class 이름	Concept 설명	주요 Attributes
Member	시스템에서 활동하는 주체	로그인 유무 판단 정보 관리자 판단 정보 쪽지함
Article	아티클의 CRUD를 처리하는 controller	작성자 작성 날짜 주제 제목 본문 내용
Contact	사용자 간의 컨택하기 메시지 송수신을 처리하는 controller	송신자 정보 수신자 정보 컨택 메시지 내용

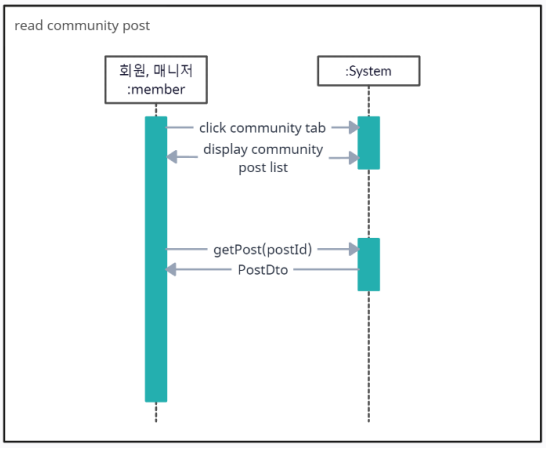
CommunityPost	커뮤니티 글의 CRUD를 처리하는 controller	작성자 작성 날짜 게시글 제목 게시글 내용
Comment	커뮤니티 게시글에 작성되는 댓글 dto	작성자 정보 댓글 내용

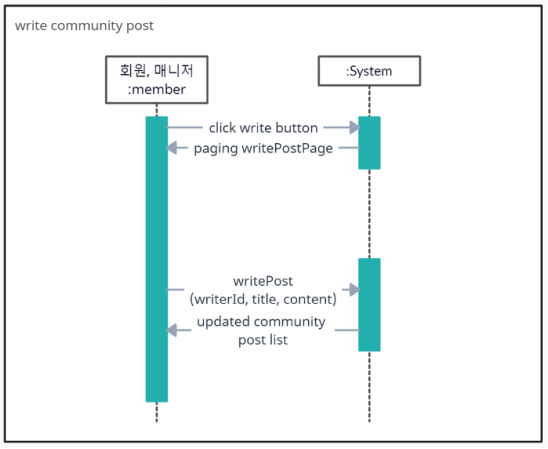
5. Design Model 산출물

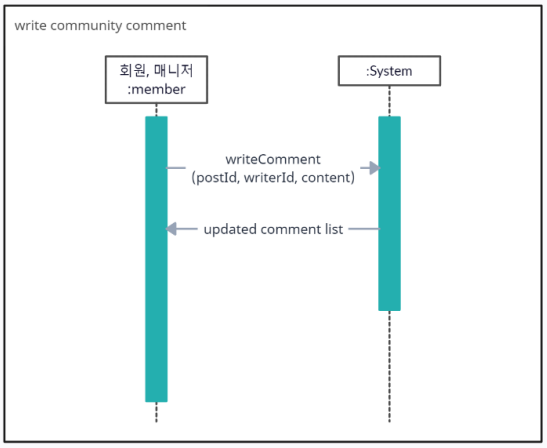
System Sequence Diagram

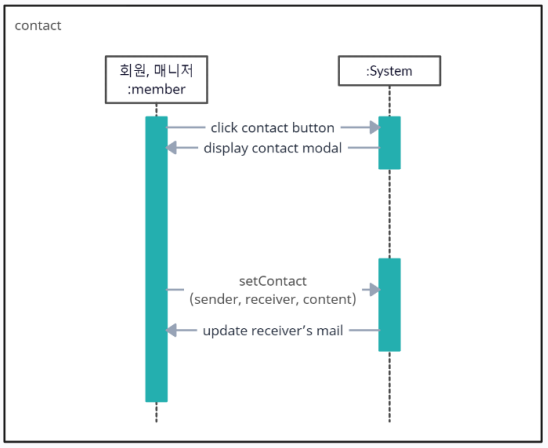












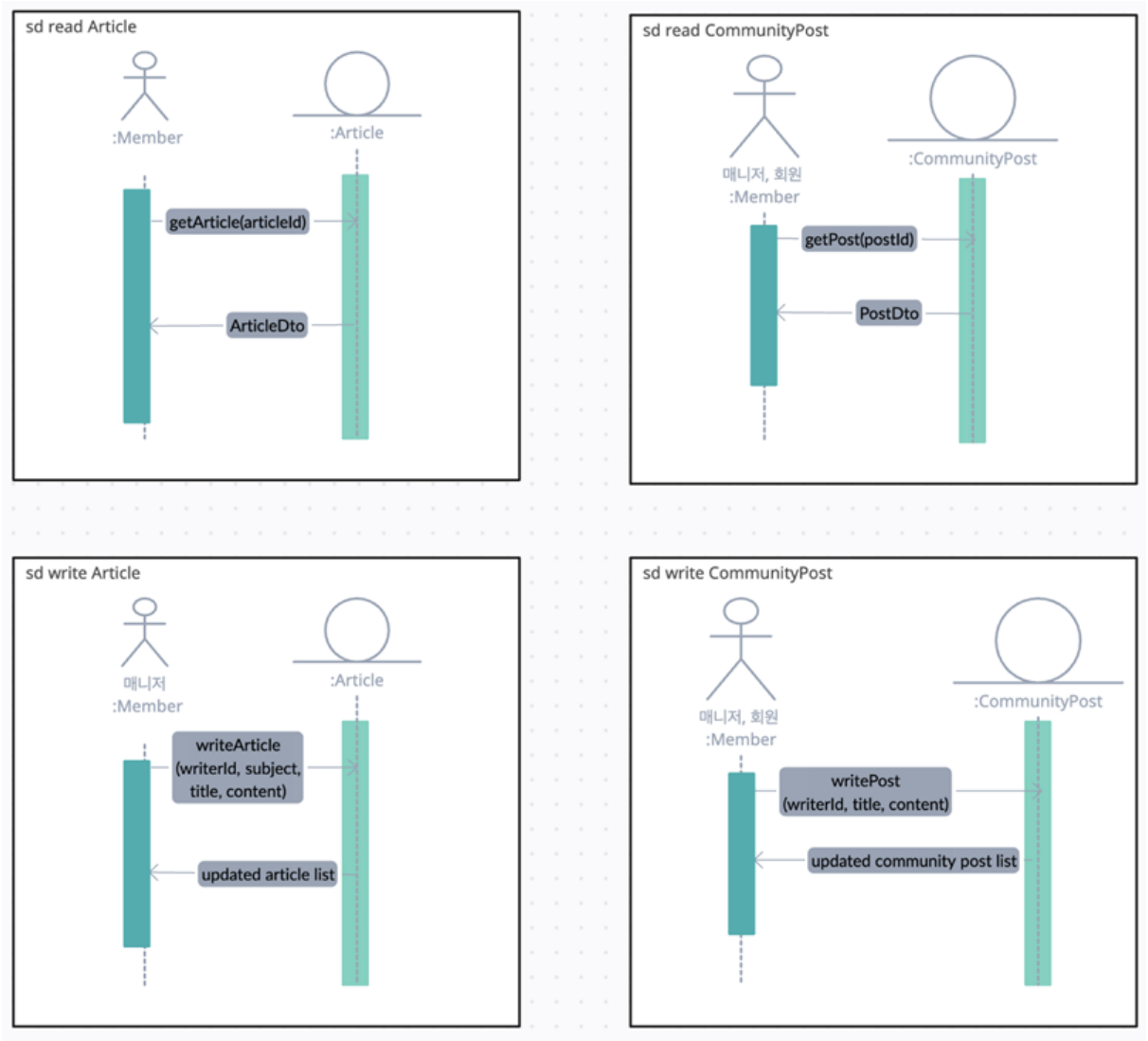
Actor	Event	Scenario	System
모든 유저	→ getArticle(articleId) ← ArticleDto	- **Actor** 1. 아티클 페이지에서 아티클의 제목 목록을 본다. 2. 아티클 페이지에서 아티클 목록 중 하나의 아티클을 클릭하여 읽는다. - **System** 1.1 아티클들을 제목만 목록화하여 보여준다. 2.1 클릭된 아티클의 제목과 본문을 보여준다.	아티클 읽기
매니저	→ login(id,password) ← display write button → click write button ← paging writePostPage → writeArticle(writerId, subject, title, content) ← updated article list	- **Actor** 1. 매니저(관리자) 계정으로 로그인한다. 2. 아티클 페이지에서 아티클 작성 버튼을 클릭하여 새로운 아티클을 작성한다. - **System** 1.1 로그인한 계정이 매니저(관리자) 권한을 가지는 지 판단한다. 1.2 매니저(관리자) 계정이 맞으면 아티클 목록 위에 아티클 작성 버튼을 보인다. 2.1 아티클 작성 페이지로 전환한다. 2.2 작성된 아티클 내용을 저장한다. 2.3 아티클 목록을 업데이트 한다.	아티클 작성
회원, 매니저	→ click community ← display community posts list → getPost(postId) ← PostDto	- **Actor** 1. 커뮤니티 페이지에서 게시글의 제목 목록을 본다. 2. 커뮤니티 페이지에서 게시글 목록 중 하나의 게시글을 클릭하여 읽는다. - **System** 1.1 게시글들을 제목만 목록화하여 보여준다. 2.1 로그인 유무를 판단한다. 2.2 클릭한 유저가 로그인을 한 유저라면 게시글 제목과 내용을 보여준다.	커뮤니티 게시글 읽기
회원, 매니저	→ click write button ← paging writePostPage → writePost(writerId, title, content) ← updated community post list	- **Actor** 1. 커뮤니티 페이지에서 게시글 작성 버튼을 클릭하여 게시글을 작성한다. - **System** 1.1 로그인 유무를 판단한다. 1.2 클릭한 유저가 로그인을 한 유저라면 게시글 작성 페이지로 전환한다. 1.3 작성된 게시글 내용을 저장한다. 1.4 커뮤니티 게시글 목록을 업데이트 한다.	커뮤니티 게시글 작성
회원, 매니저	→ writeComment(postId, writerId, content) ← updated comment list	- **Actor** 1. 회원이 읽고 있는 게시글에 댓글을 작성한다. - **System**	커뮤니티 게시글 댓글 작성

		1.1 회원에게 게시글 페이지에서 댓글창을 제공한다. 1.2 회원이 작성한 댓글이 댓글창에 업데이트한다.	
회원, 매니저	→ Click contact button ← display contact Modal → setContact(sender, receiver, content) ← update receiver's mail	- **Actor** 1. 컨택하기 버튼을 클릭한다. 2. 해당 글의 대표에게 쪽지를 보낸다. - **System** 1.1 회원에게 게시글 페이지에서 컨택하기 버튼을 제공한다. 2.1 해당 게시글의 대표에게 전달할 쪽지 모달을 제공한다. 2.2 작성된 쪽지 내용을 대표 수신함에 전송한다.	컨택하기

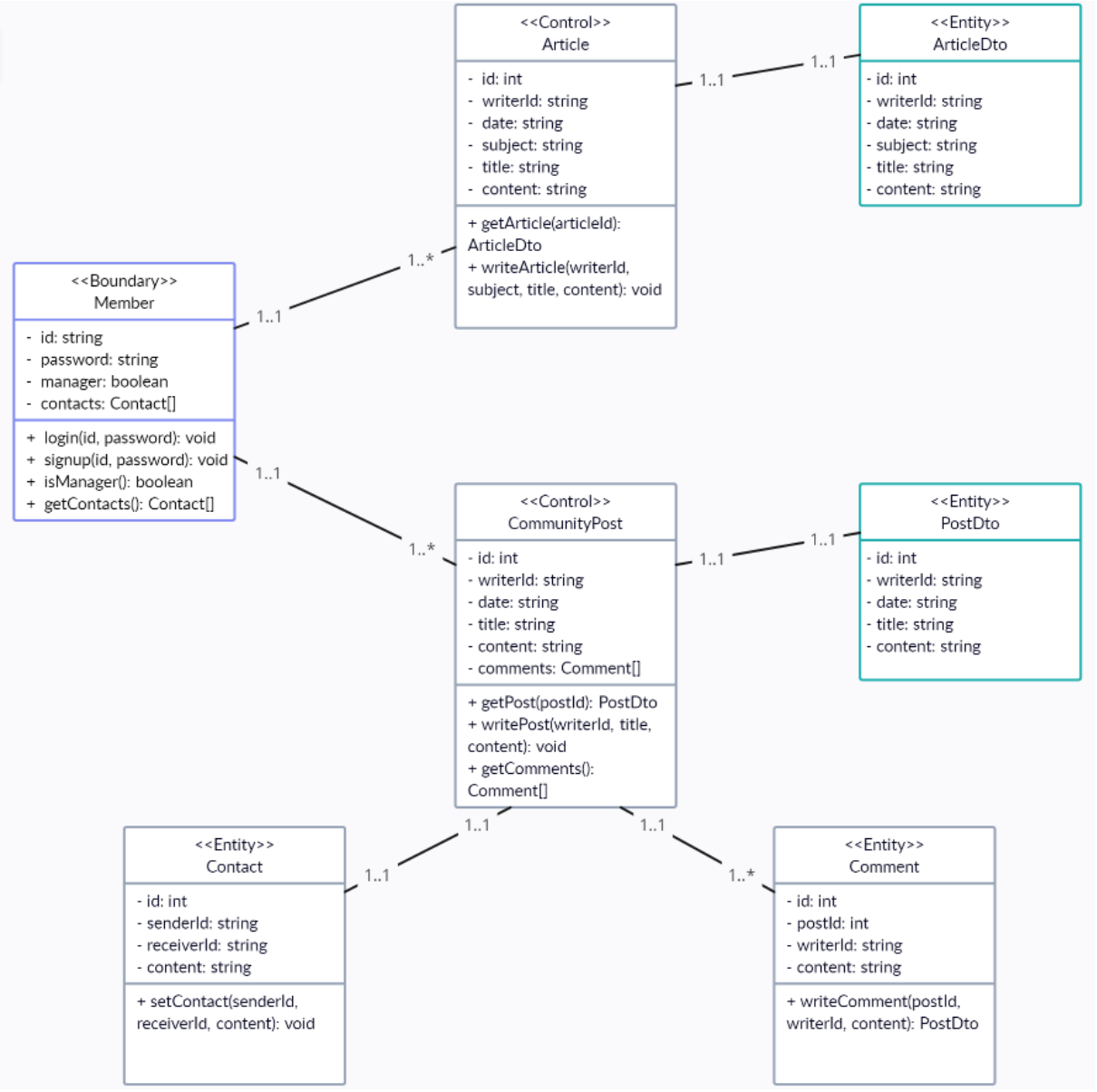
Operation Contract

Operation	Cross references	Preconditions	Postconditions
getArticle(postId)	아티클 읽기	아티클 목록에 한 개 이상 아티클이 있어야 한다.	없음
writeArticle(writerId, subject, title, content)	아티클 작성	로그인 시 매니저 권한이 확인 되어 있어야 한다.	아티클 목록이 업데이트 되었다.
getPost(postId)	커뮤니티 게시글 읽기	Actor가 로그인 되어있어야한다. 커뮤니티 게시글 목록에 한 개 이상 게시글이 있어야 한다.	없음
writePost(writerId, title, content)	커뮤니티 게시글 작성	Actor가 로그인 되어 있어야 한다.	커뮤니티 게시글 목록이 업데이트 되었다.
writeComment(postId, writerId, content)	커뮤니티 게시글 댓글 작성	Actor가 로그인이 되어 있어야 한다.	댓글 목록이 바로 업데이트 되었다.
setContact(senderId, receiverId, content)	컨택하기	Actor가 로그인이 되어 있어야 한다.	해당 게시글의 대표의 수신함이 업데이트 되었다.

Sequence Diagram

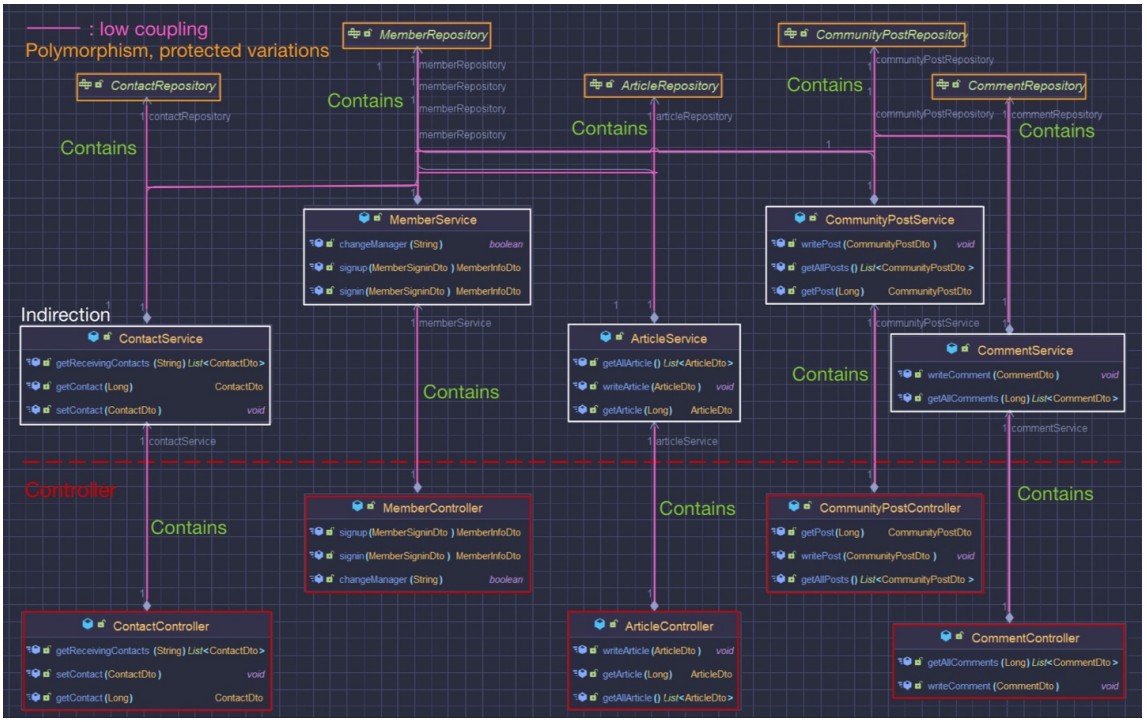


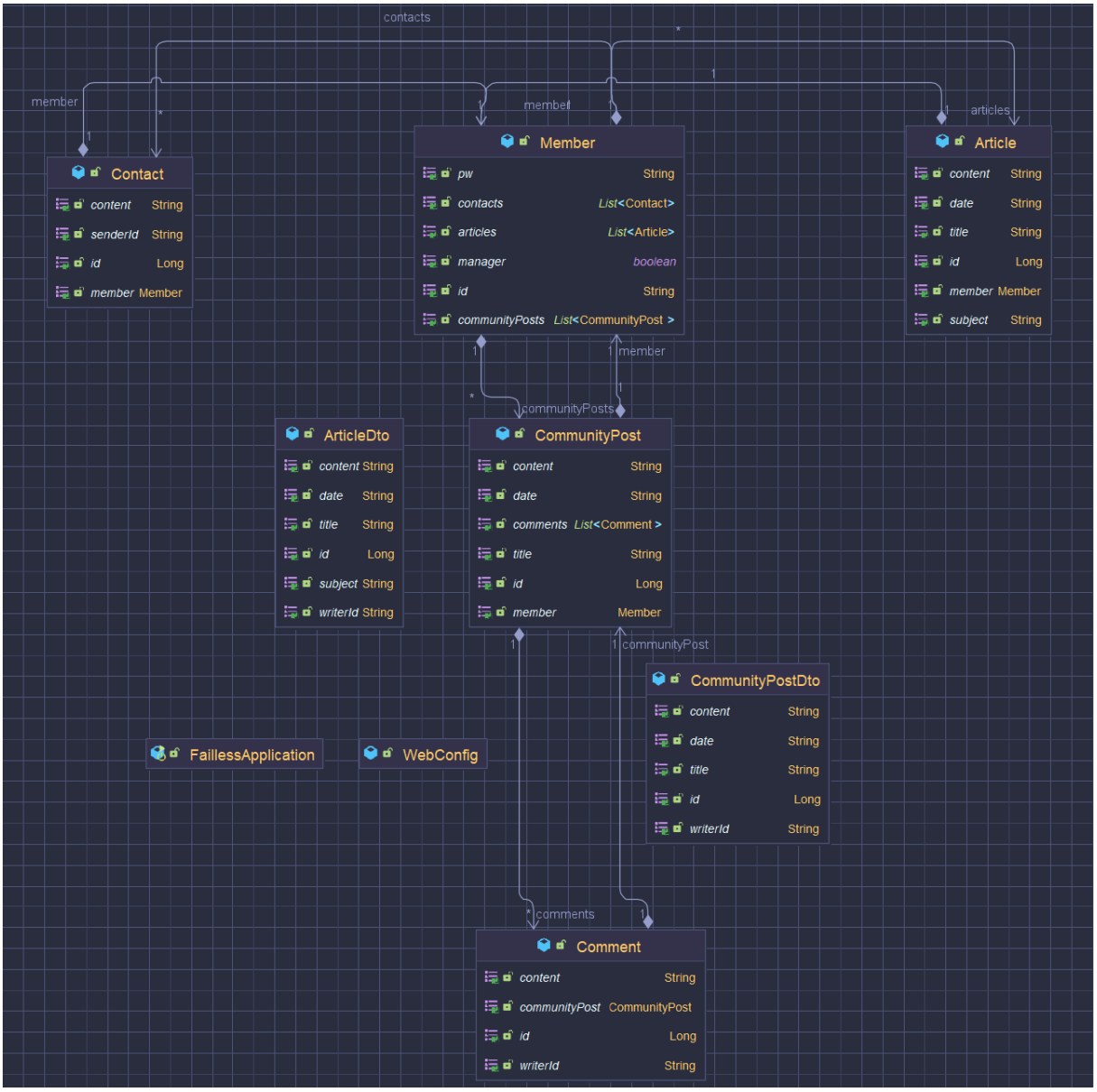
(Before) Class Diagram

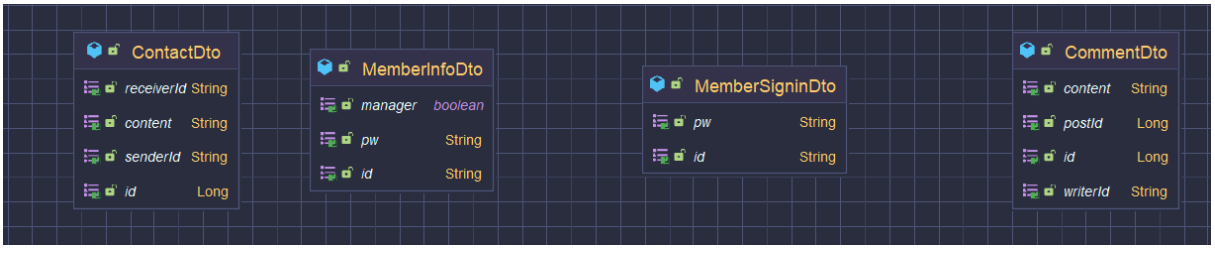


6. SW Design Pattern을 적용한 (after) Class Diagram

(After) Class Diagram


: low coupling
Polymorphism, protected variations
MemberRepository
CommunityPostRepository
ContactRepository
ArticleRepository
CommentRepository
Contains
Indirection
MemberService
CommunityPostService
ContactService
ArticleService
CommentService
Controller
MemberController
CommunityPostController
ContactController
ArticleController
CommentController





Design Refinement

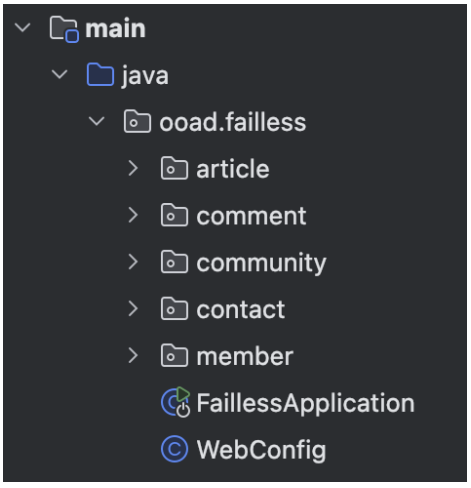
Before 단계의 Class	After 단계의 Class	적용한 설계 개념	Architecture Design Rationale	NFR과 QA에 대한 영향 분석
Member	Member, MemberRepository, MemberService, MemberController	Controller, Indirection, Polymorphism, Protected	Member: 사용자와 관련된 attributes를 가진다 MemberRepository: JPA를 통해 사용자 정보를 DB에 작업한다.	각 클래스가 미치는 영향이 유사하다 생각하여 하나로 정리하였습니다.

		Variations, Low Coupling	MemberService: Controller를 통해 들어온 로그인 등과 같은 사용자 요청을 처리한다. MemberController: 로그인, 회원가입과 같은 사용자 정보에 대한 요청을 받는다.	
Article	Article, ArticleRepository, ArticleService, ArticleController	Controller, Indirection, Polymorphism, Protected Variations, Low Coupling	Article: 아티클의 속성 정보를 가진다. ArticleRepository: MemberRepository와 유사하다. ArticleService: 아티클 작성, 수정, 특정 아티클 리턴과 같은 실제 기능을 수행한다. ArticleController: 아티클 읽기, 작성과 같은 요청을 받는다	⬇️
CommunityPost	CommunityPost, CommunityPostRepository, CommunityPostService, CommunityPostController	Controller, Indirection, Polymorphism, Protected Variations, Low Coupling	CommunityPost: 커뮤니티 글에 대한 속성 정보를 가진다. CommunityPostRepository: MemberRepository와 유사하다. CommunityPostService: 커뮤니티 글 작성, 수정, 특정 커뮤니티 글 리턴과 같은 실제 기능을 수행한다. CommunityPostController: 커뮤니티 글 읽기, 작성과 같은 요청을 받는다	⬇️
Contact	Contact, ContactRepository, ContactService, ContactController	Controller, Indirection, Polymorphism, Protected Variations, Low Coupling	Contact: 컨택하기에 대한 속성을 가진다. ContactRepository: MemberRepository와 유사하다.MemberRepository와 유사하다. ContactService: 컨택하기 기능을 수행한다. ContactController: 컨택하기에 대한 요청을 받는다.	⬇️
Comment	Comment, CommentRepository, CommentService, CommentController	Controller, Indirection, Polymorphism, Protected Variations, Low Coupling	Comment: 댓글의 속성을 가진다. CommentRepository: MemberRepository와 유사하다. CommentService: 컨트롤러로 받아온 요청을 실제로 실행하여 결과를 리턴해준다. CommentController: 댓글 작성, 불러오기와 같은 요청을 받는다	⬇️
	→ Spring Framework에 기반하여 프로젝트를 설계하여 전반적으로 MVC 모델의 형태를 띄고 있다고 생각합니다.	정석적인 설계로는 컨트롤러와 서비스 모두 인터페이스를 두고 구현	Controller 클래스들은 RestAPI로 client의 요청을 받아들입니다. Repository 클래스들은 현재는 JPA를 사용중이	성능: 클래스들이 각각의 역할에 맞게 최적화되어, 전반적인 성능이 향

		체를 만들었어야 하지만, 프로젝트의 크기가 작고 변동할 일이 적다고 판단하여 Polymorphism과 Protected Variations보다는 Low Coupling과 Indirection에 초점을 두고 설계해 보았습니다.	지만 MemoryRepository를 사용하는 등 repository 구현체를 바꾸면 의존성 주입만 해주면 되기 때문에 Polymorphism과 Protected Varation이 적용되었다고 생각합니다.	상됩니다. 확장성: 각 구성 요소들이 독립적으로 확장 가능하여, 시스템이 커지더라도 유연하게 대응할 수 있습니다. 유지보수성: 낮은 결합도와 높은 응집도로 인해 코드의 유지보수와 확장이 용이합니다. 보안성: 기능별로 분리되어 보안 취약점을 최소화하고, 중요한 기능에 대한 보안 조치를 강화할 수 있습니다. 사용성: 빠르고 안정적인 서비스 제공으로 인해 사용자 경험이 향상됩니다.

7. 구현 결과 Snapshot

프로젝트 구조



각 엔티티 별로 패키지를 나누어 설계하였습니다. 패키지 별로 엔티티, 컨트롤러, 서비스, 리포지토리 클래스가 각각 존재합니다. 그리고 필요 시 DTO 객체를 추가하기도 하였습니다.

패키지 내부의 코드 구조는 유사한 부분이 많아, article 패키지의 코드 내용을 대표적으로 보여드리겠습니다.

Entity

```java
package ooad.failless.article;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import ooad.failless.member.entities.Member;

@Getter  6 usages  clzlol
@Entity
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Article {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column
    private String date;
    @Column
    private String subject;
    @Column
    private String title;
    @Column
    private String content;

    @ManyToOne
    @JoinColumn(name = "writerId", nullable = false)
    private Member member;
}
```

엔티티 클래스 내에는 DB에 저장될 테이블의 컬럼들이 속성값으로 들어있습니다. Key가 될 값에는 @Id 어노테이션을 붙여주고, 그 외 컬럼들은 @Column 어노태이션을 붙여 Spring이 이들을 해석하고 DB에 저장할 수 있게 합니다.

```java
@OneToMany(mappedBy = "member")
private List<Article> articles;
```

그리고 엔티티가 가질 연관관계 또한 이처럼 어노태이션을 활용하는데, 여기서 사용한 다 대 일 관계에서는 '일' 쪽인 Member에는 리스트를 사용해 사용자가 작성한 아티클들을 저장할 수 있게 해주었고, '다' 쪽인 Article에는 @JoinColumn 어노테이션을 함께 사용해 작성자가 누구인지 알 수 있게 구현하였습니다.

Controller

```java
package ooad.failless.article;

import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController  clzlol
@RequiredArgsConstructor
public class ArticleController {

    private final ArticleService articleService;

    @PostMapping("/article")  clzlol
    public void writeArticle(@RequestBody ArticleDto articleDto) { articleService.writeArticle(articleDto); }

    @GetMapping("/article/all")  clzlol
    public List<ArticleDto> getAllArticle() { return articleService.getAllArticle(); }

    @GetMapping("/article")  clzlol
    public ArticleDto getArticle(@RequestParam Long id) { return articleService.getArticle(id); }
}
```

컨트롤러에는 @RestController 어노테이션을 붙여 RestAPI 요청을 이 컨트롤러가 받을 것임을 스프링에게 알려줍니다. 그리고 필요한 HTTP 메서드 요청에 대해 어떤 서비스를 실행할지 각 메서드 안에 Service 객체와 연결해 주었습니다.

Service

```java
package ooad.failless.article;

import lombok.RequiredArgsConstructor;
import ooad.failless.member.MemberRepository;
import ooad.failless.member.entities.Member;
import org.springframework.stereotype.Service;

import java.util.List;

@Service  1 usage  clzlol
@RequiredArgsConstructor
public class ArticleService {

    private final ArticleRepository articleRepository;
    private final MemberRepository memberRepository;

    public void writeArticle(ArticleDto articleDto) {  1 usage  clzlol
        Member member = memberRepository.findById(articleDto.getWriterId())
                        .orElseThrow(() -> new IllegalArgumentException("id doesn't exist"));

        articleRepository.save(Article.builder()
                        .id(articleDto.getId())
                        .date(articleDto.getDate())
                        .title(articleDto.getTitle())
                        .subject(articleDto.getSubject())
                        .content(articleDto.getContent())
                        .member(member)
                        .build());
    }

```

```java
    public List<ArticleDto> getAllArticle() {  1 usage  clzlol
        List<Article> articleList = articleRepository.findAll();
        if (articleList.isEmpty()) {
            throw new IllegalArgumentException("article list is empty");
        }

        return articleList.stream()  Stream<Article>
                .map(article -> ArticleDto.builder()
                        .id(article.getId())
                        .date(article.getDate())
                        .subject(article.getSubject())
                        .title(article.getTitle())
                        .content(article.getContent())
                        .writerId(article.getMember().getId())
                        .build())  Stream<ArticleDto>
                .toList();
    }

    public ArticleDto getArticle(Long id) {  1 usage  clzlol
        Article article = articleRepository.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("wrong article id"));

        return ArticleDto.builder()
                .id(article.getId())
                .title(article.getTitle())
                .subject(article.getSubject())
                .date(article.getDate())
                .content(article.getContent())
                .writerId(article.getMember().getId())
                .build();
    }

}

```

서비스에는 Controller로부터 받은 사용자 요청에 대한 처리를 구현했습니다. Article에서는 아티클 리스트를 띄워줄 수 있도록 아티클 리스트를 리턴해주거나, 특정 아티클을 선택했을 때 그것을 리턴해주거나, 아티클을 새로 작성할 수 있는 메서드를 작성해두었습니다.

Repository

```java
package ooad.failless.article;

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

@Repository  1 usage  clzlol
public interface ArticleRepository extends JpaRepository<Article, Long> {
}

```

리포지토리는 JpaRepository 인터페이스를 상속하여 구현했습니다. Spring이 Jpa를 사용해 알아서 MySQL의 데이터베이스에 사용자 요청을 처리할 수 있도록 이렇게 구성하였습니다. 이는 Jpa가 아니라 메모리를 사용한 리포지토리를 사용할 때 구현체만 교체해주면 되기 때문에 다형성을 지킬 수 있게 설계되어 있습니다.

그 외 다른 패키지의 엔티티, 컨트롤러, 서비스, 리포지토리도 유사한 기능들이 구현되어있습니다. 차이점은 연관관계의 매핑 정도가 있습니다.

WebConfig

```java
package ooad.failless;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration  clzlol
public class WebConfig implements WebMvcConfigurer {

    @Override  no usages  clzlol
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping( pathPattern: "/**")
                .allowedOrigins("*")
                .allowedMethods("GET", "POST", "PUT", "DELETE")
                .allowedHeaders("Authorization", "Content-Type")
                .exposedHeaders("Custom-Header")
                .maxAge(3600);
    }
}

```

WebConfig 클래스는 기존 설계에는 없었던 클래스인데, 프론트에서 Spring 서버로 요청을 했을 때, CORS 에러가 발생하는 것을 해결하기 위해 추가한 Config 코드입니다.