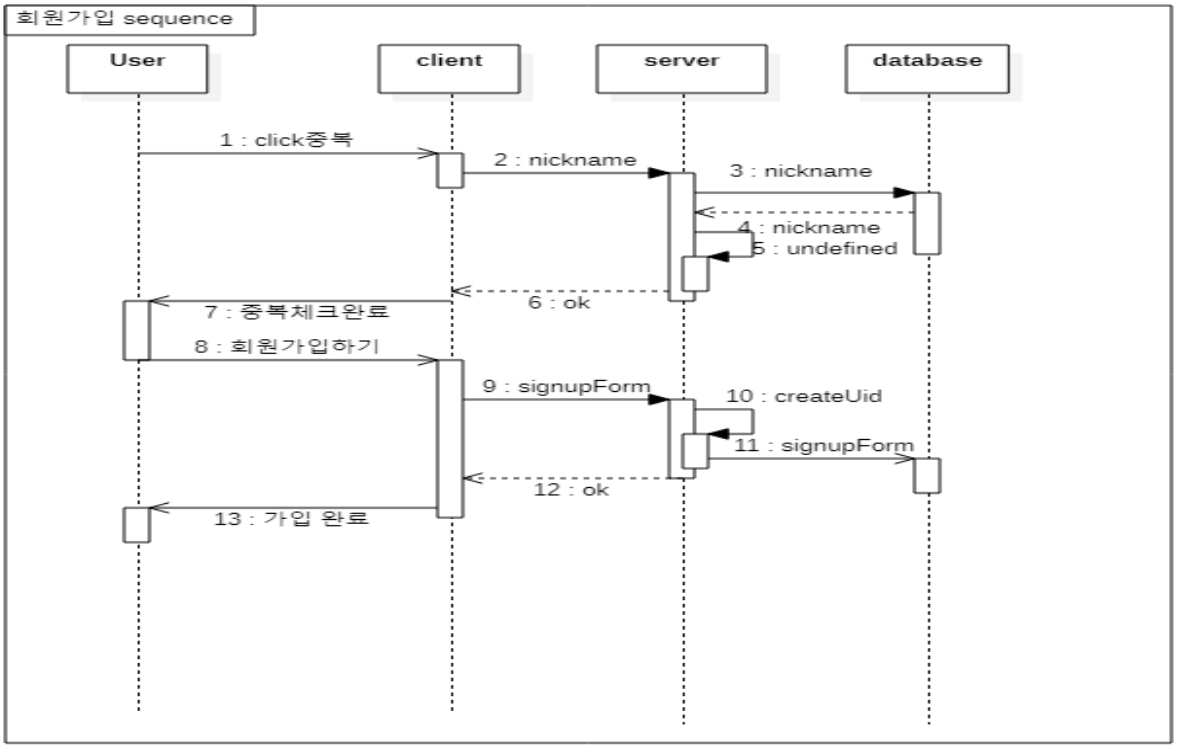
1. 프로젝트 및 일정 관리 노션

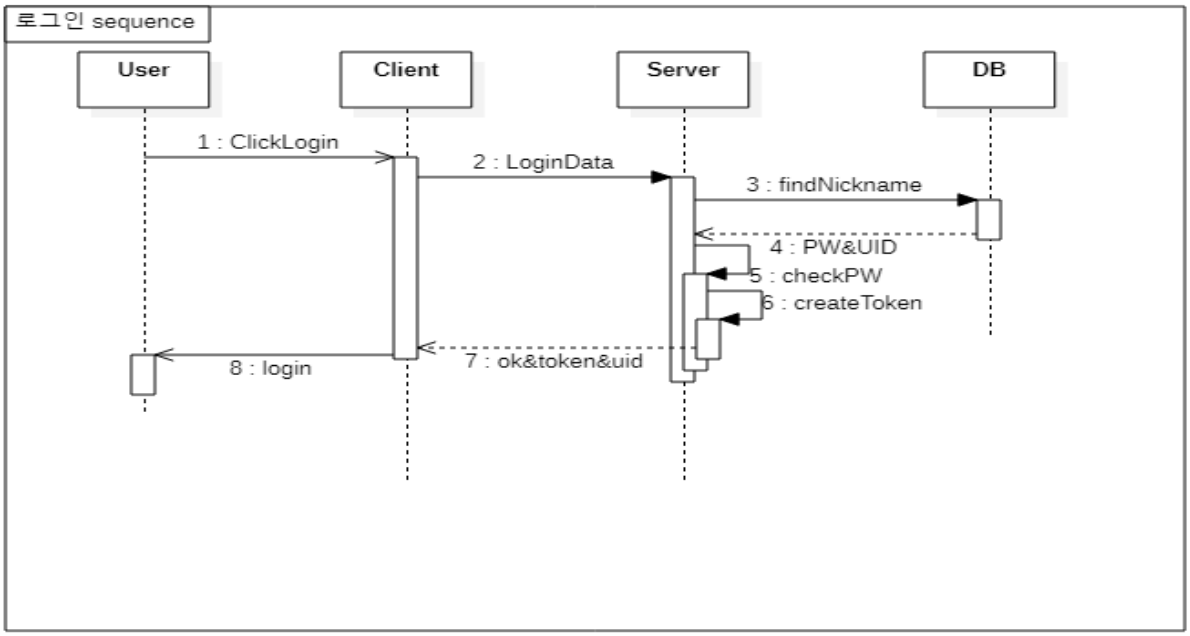
   - https://www.notion.so/Capstone-13-512db785f6b24e43a3749cc283716904

2. 요구사항 분석 및 사용자 시나리오

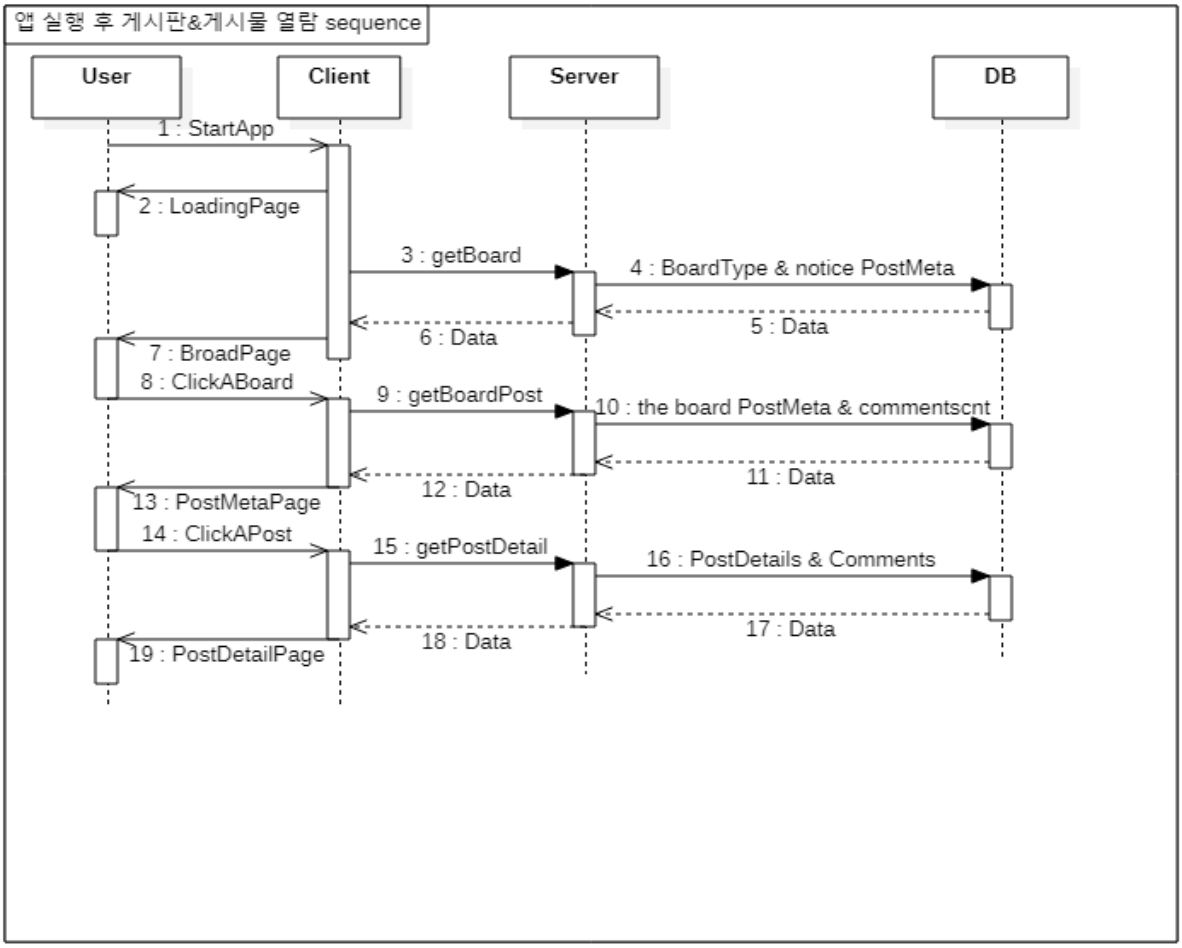
   - 회원가입



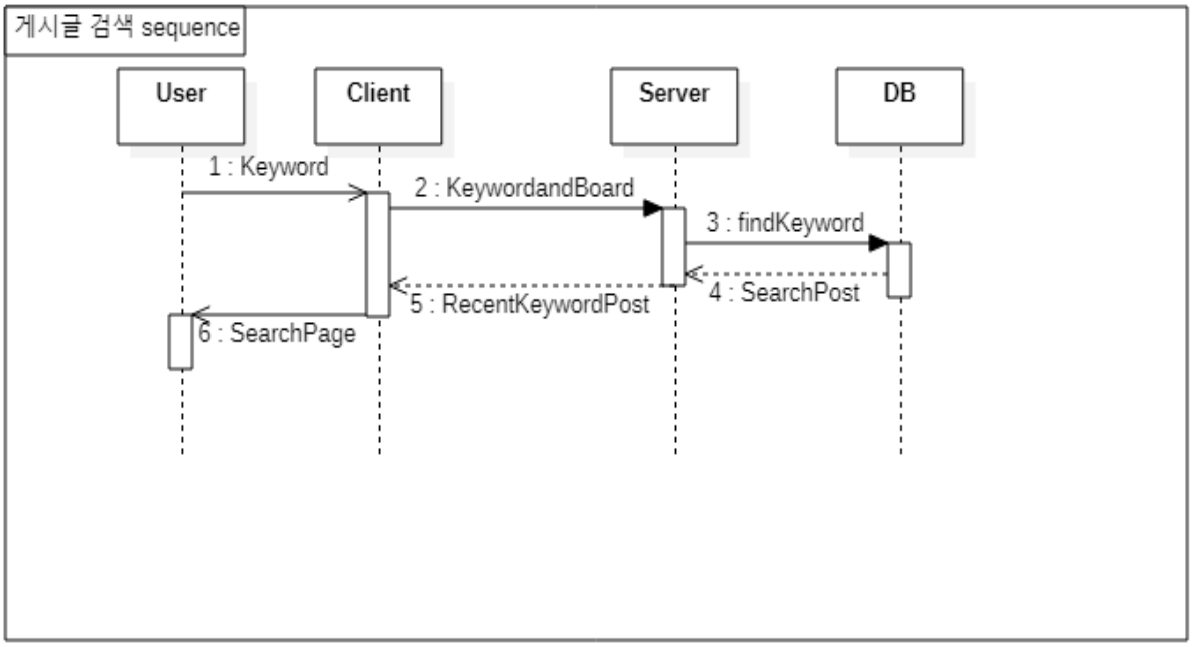
   - 로그인



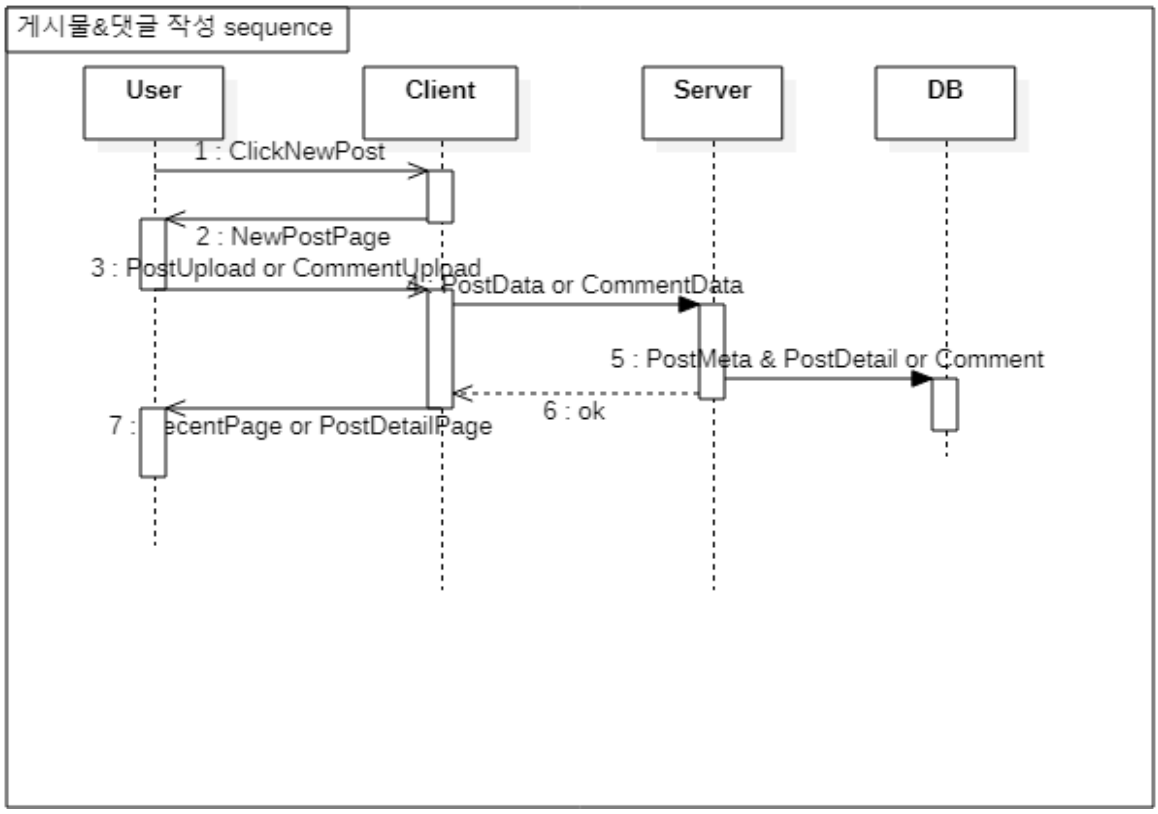
- 게시물 열람


앱 실행 후 게시판&게시물 열람 sequence
User
Client
Server
DB
1 : StartApp
2 : LoadingPage
3 : getBoard
4 : BoardType & notice PostMeta
5 : Data
6 : Data
7 : BroadPage
8 : ClickABoard
9 : getBoardPost
10 : the board PostMeta & commentscnt
11 : Data
12 : Data
13 : PostMetaPage
14 : ClickAPost
15 : getPostDetail
16 : PostDetails & Comments
17 : Data
18 : Data
19 : PostDetailPage

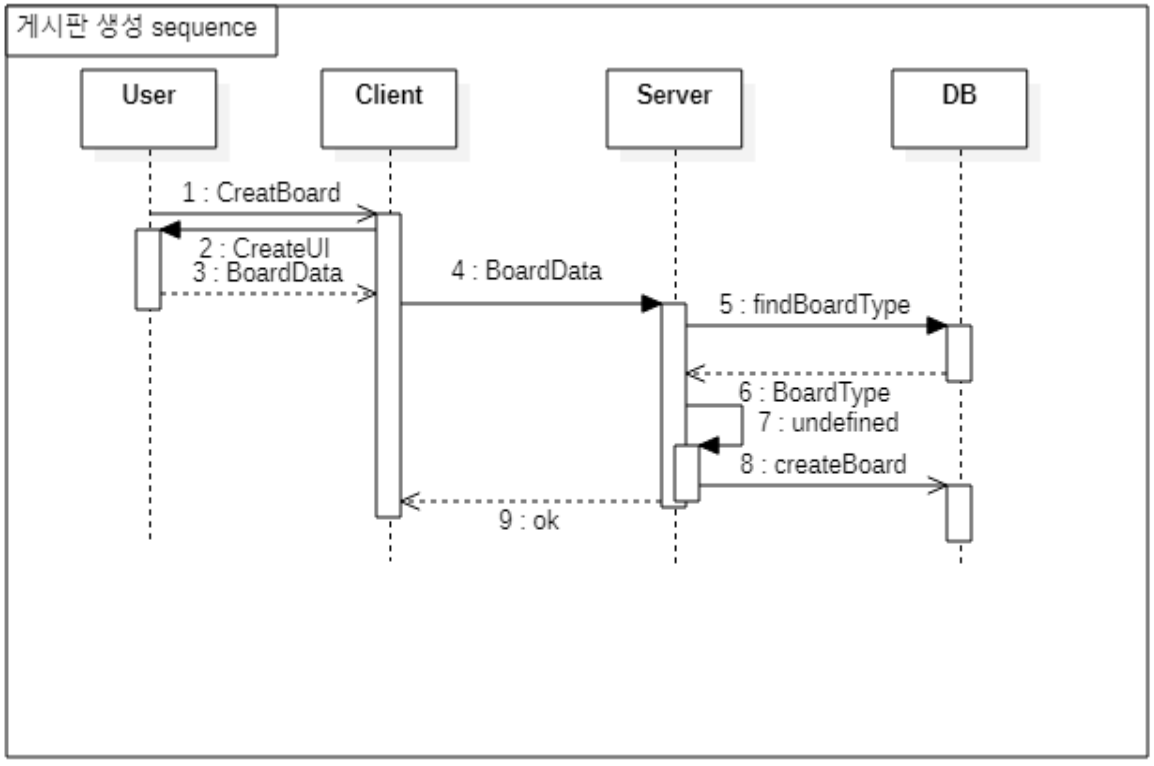
- 게시글 검색


게시글 검색 sequence
User
Client
Server
DB
1 : Keyword
2 : KeywordandBoard
3 : findKeyword
4 : SearchPost
5 : RecentKeywordPost
6 : SearchPage

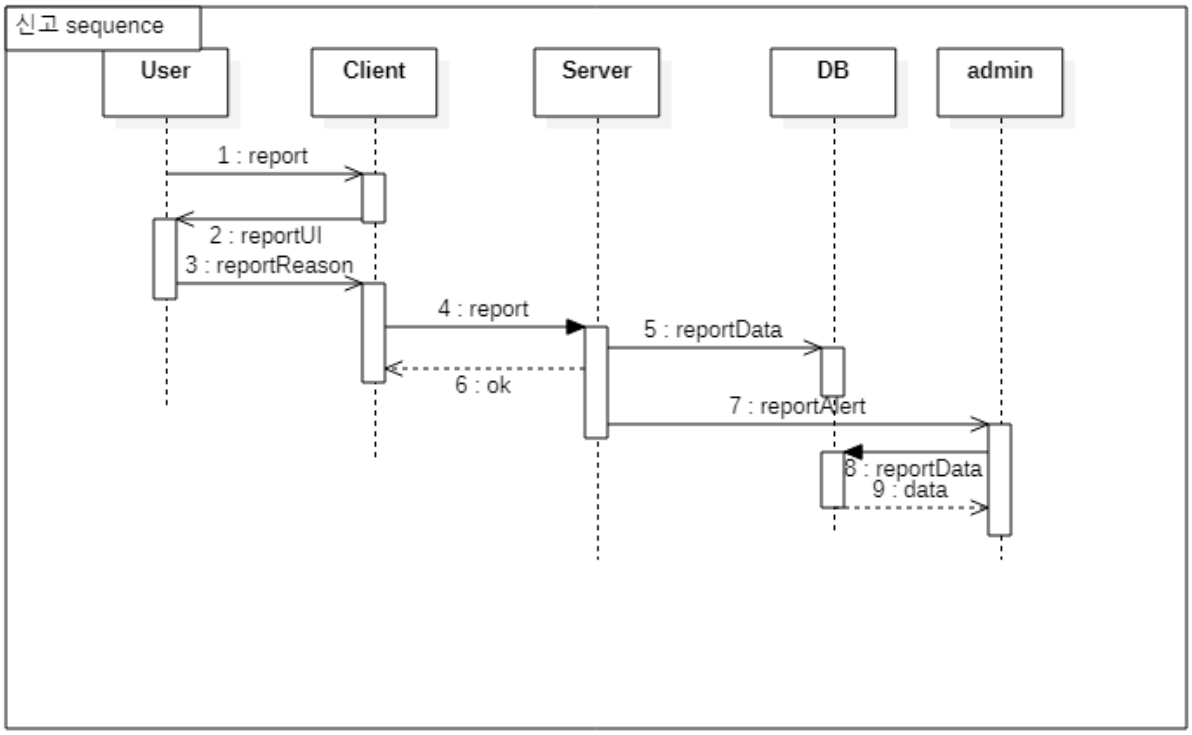
- 게시물 작성


게시물&댓글 작성 sequence
User
Client
Server
DB
1 : ClickNewPost
2 : NewPostPage
3 : PostUpload or CommentUpload
4 : PostData or CommentData
5 : PostMeta & PostDetail or Comment
6 : ok
7 : ecentPage or PostDetailPage

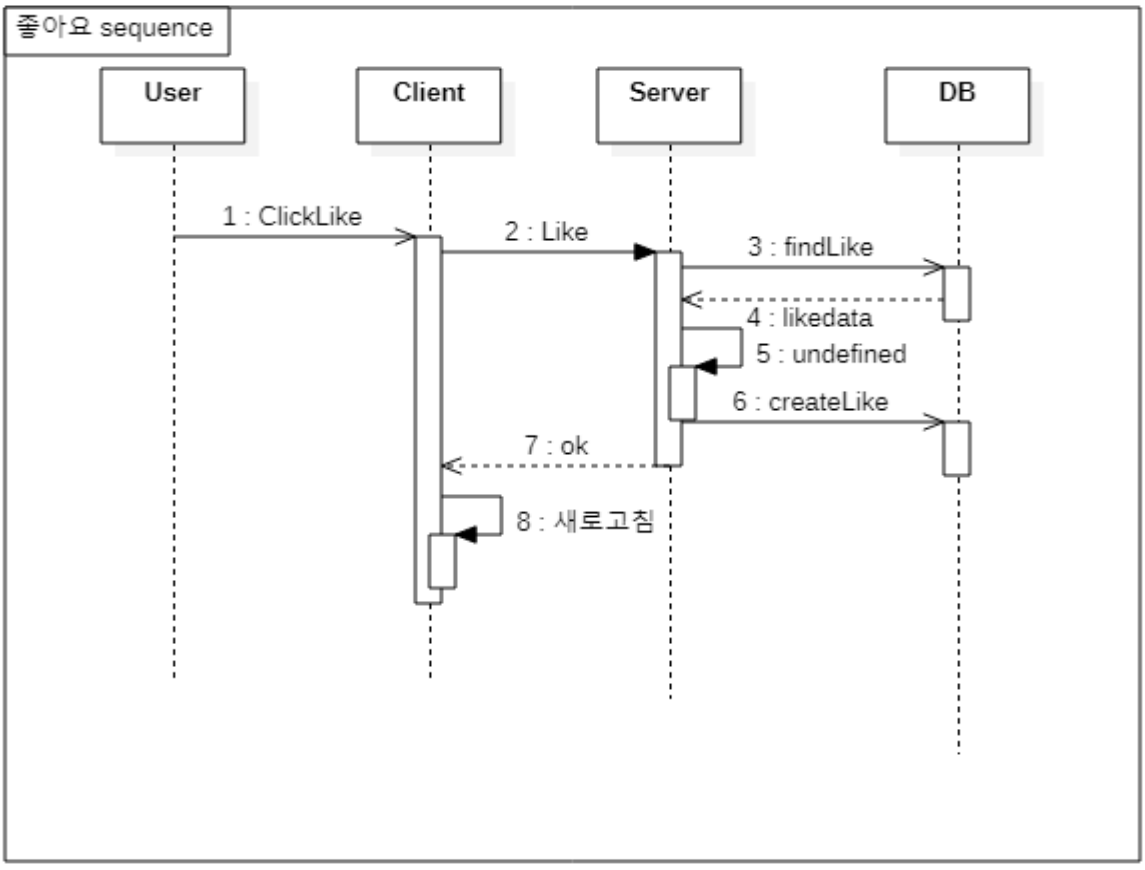
- 게시판 생성


게시판 생성 sequence
User
Client
Server
DB
1 : CreatBoard
2 : CreateUI
3 : BoardData
4 : BoardData
5 : findBoardType
6 : BoardType
7 : undefined
8 : createBoard
9 : ok

- 신고


신고 sequence
User
Client
Server
DB
admin
1 : report
2 : reportUI
3 : reportReason
4 : report
5 : reportData
6 : ok
7 : reportAlert
8 : reportData
9 : data

- 좋아요


좋아요 sequence
User
Client
Server
DB
1 : ClickLike
2 : Like
3 : findLike
4 : likedata
5 : undefined
6 : createLike
7 : ok
8 : 새로고침

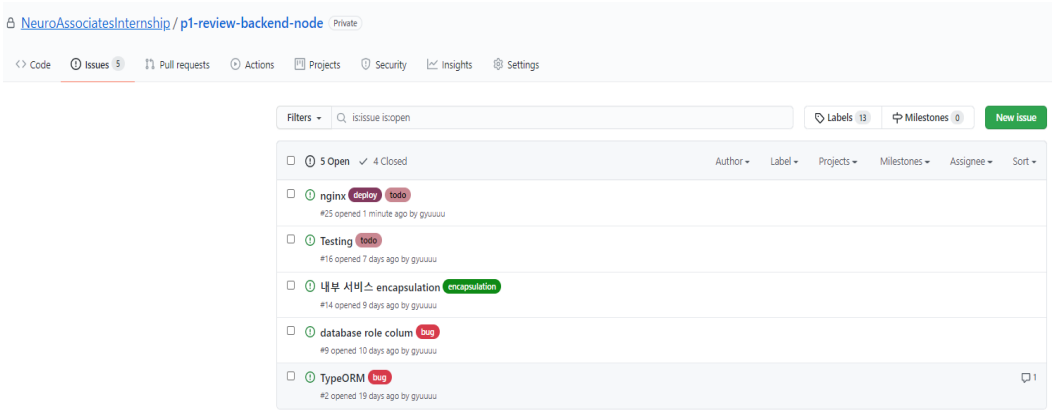
3. API 문서

   팀 공유 노션에 상세히 정리해 두었습니다.

   - https://www.notion.so/API-d6899114a26f48a9add688725ffe2aa6

4. 이슈 관리

   - 개발 진행 중 발생하는 이슈들은 github issue 탭을 통해 관리합니다.



5. 기술 선정

   https://www.notion.so/63f20eba749947f895f76b10c141395b

   - Nest JS 선정 배경

     언어를 typescript로 결정하고 나서 사용할 프레임워크를 결정해야 했다. 이사님께서는 처음에 범용성있고 많이 사용되는 express 프레임워크를 추천해주셨다. 하지만 이전에 동아리 활동에서 express를 사용해서 협업했을 때 불편한 점이 존재했다.

     ■ Express가 사용하기 쉽고, 개발자의 자율성이 높은 프레임워크라는 장점이 있지만 반대로 일정한 구조(틀)이 없어서 개발자들 간의 아키텍처(레이어별, 모듈별...등등)를 통일하는데 어려움이 있었다.

     ■ Error handling을 일일이 해줘야 해서 생산성이 떨어졌다.

     이러한 문제점과 더불어 현재 13팀 개발자들의 상황이 팀장을 제외하면 프로젝트 경험도 없고, 서버 사이드 프로그래밍 경험이 없었다.

     이러한 상황에서 express를 사용한다면 Nest Js에 비해서 배워야할 양이 적어 좀 더 빠르게 개발을 할 수 있겠지만, 모두 다른 방식으로 개발할 것이다. 또한 처음 접해본 사람들에게 express의 자율성이라는 것이 오히려 더 큰 어려움으로 다가갈 수 있을 것이라고 판단하였습니다.

     따라서 어느정도 구조화된 프레임워크를 선택해야 했고, 한 명의 리더가 나머지를 이

끌고 가이드를 제공하여 코딩하기 쉬운 기술을 선택해야했다.

Nest는 이에 적합한 기술이라고 생각했습니다. Nest에서 사용하는 모듈화, Dependency Injection 컨셉을 통해 아키텍처를 구조화하였고, 따라서 개발자들 간의 통일성을 확보할 수 있었습니다. 또한 cli툴과 내장 기능을 통해 스켈레톤 코드 작성 및 테스트에 용이하여 협업에 편리함을 가져가주었습니다.

- Database

	SQL	NoSQL
장점	테이블간 관계 표현에 좋다	유연함, 읽기 속도 빠름
단점	스키마가 사전에 계획	무결성 보장 힘듬

진행 중인 프로젝트의 성격 상 데이터베이스의 테이블 간의 관계를 표현하는 것이 중요하고 join쿼리도 자주 발생할 것으로 예상되기 때문에 NoSQL보다는 SQL 데이터베이스를 선택하였다.

6. 클라우드 및 인프라

처음에는 serverless 아키텍처를 기반으로 한 클라우드 서비스를 계획하였다.

실리콘 벨리의 멘토님께서도 처음에는 serverless 형태로 가는 것이 비용이나 운영적인 측면에서 효율적이라고 조언해 주셨다.

하지만 회사내의 상황이 달라져서 serverless보단 docker container의 형태로 배포하기로 결정하였다.

- 앱 팀이 테스트 해보고 사용할 수 있는 서버를 지속적으로 제공해야한다.

  개발 과정에서 백엔드 팀에서 만든 서버를 앱팀에서 지속적으로 사용하면서 개발을 진행해야한다. 처음에는 더미 서버를 제공하였지만 언제까지나 더미 서버로만 개발을 진행 할 순 없기 때문에 모듈별로 개발이 완료될 때마다 배포하여 앱팀이 사용 가능하도록 해야한다.

  이를 클라우드사에서 제공하는 serverless 아키텍처를 사용해도 가능하지만 불필요하게 비용이 발생하게 된다.

- 초기 배포 인프라가 변했다.

  처음 계획에서는 초기(배타버전) 배포도 클라우스 사에서 제공하는 serverless 아키텍처를 사용하려고 했으나, 회사의 장비가 추가됨에 따라 초기에는 온프레미스 환경에 배포하고 이후에 클라우드로 migration하기로 계획을 변경했다.

  이에 따라 두 환경에서 모두 사용가능한 형태로 배포가 이루어 지는 것이 효율적이라고 판단하였다. (두가지 배포시나리오를 따로 두고, 따로 개발하는 것은 매우 비효율적)

Docker는 이런 상황에 적합한 형태라고 생각했습니다. Container 내부에서 격리되어 실행환경을 가지기 때문에 온프레미스, 클라우드 모두 동일하게 작동되는 것을 보장했고, 배포 과정도 동일하게 가져갈 수 있다고 생각했습니다.
또한 AWS의 Fargate라는 서비스가 존재해 container를 serverless처럼 운영할 수 있도록 도와준다. 따라서 현재 container에 추가적인 작업을 수행하지 않아도 동일하게 serverless 환경에서 배포 운영 할 수 있게 된다.

- 플랫폼에 종속되지 않을 수 있다.

  클라우드 상에서 Docker Container를 통해 배포 관리한다면 특정 클라우드 사에 종속되지 않을 수 있다.
  배포 요구사항 중에서 특정 플랫폼에 종속적지 않아야한다는 조건이 있었기 때문에 Docker는 그에 적합한 형태라고 할 수 있다.