전남 개인 프로젝트 피드백

태그	
날짜	@2023년 1월 4일

노란색은 내가 질문한 내용들

BE 질의응답

▼ 뭔가 만들 때, 틀에 갇힌 느낌인데, 이를 벗어나려면 어떻게 해야 하는지(ex. 유튜브 강의대로 만들려는 느낌)

팀 프로젝트가 끝난 후 해결될 것이다.

부트 캠프, 클론 코딩을 많이 하신 분들이 특히 이런 상태가 많았다.

결과물은 있어도 나 혼자 뭘 하려니 못 하겠다.

부트 캠프는 물어보면 멘토들이 다 해결해준다고 한다😲(교육 업체는 진도율이 중요해서)

성장은 누가 가르쳐준다고 해서 되는 게 절대 아니다.

성장은 계단과 같다.

스스로 고민을 많이 해보면서 성장 할 수 있다.

팀의 동료들과 같이 토론도 하면서 문제를 해결해가는 것을 추천

죽어도 못 하겠으면 그때 멘토를 찾아라~~~

▼ PK를 auto-increment로 해서 사용하면 보안 측면에서 문제가 있다는 것을 알고 있습니다. UUIDv4를 사용하는 글을 읽었는데 실제로는 어떤 방법을 주로 사용하는지 궁금합니다.

둘다 사용합니다.

auto-increment는 편하다.

id 값을 안다고 해서 할 수 있는 게 많지 않다.

기본적으로 API에 대해서 보안 관리를 다 하기도 하고.

회사의 내부 정책에 따른다.

큰 회사는 DB 팀이 있다. 이미 조직의 표준이 다 잡혀있다.

▼ BCrypt는 RAM에 많은 양을 요구한다는 말이 있던데 그러면 자주 사용되는 방식이 무엇인가요?

BCrypt는 RAM에 많은 양을 요구한다는 말 👉사실일 수도 있다.

임베디드 개발자가 아닌 이상 예전보다 신경을 덜 쓴다.

아래 자료는 2018년 기준

분류	NIST(미국) (2015)	CRYPTREC(일본) (2013)	ECRYPT(유럽) (2018)	국내[1] (2018)
해시함수	SHA-224 SHA-256 SHA-384 SHA-512 SHA-512/224 SHA-512/256 SHA3-224 SHA3-256 SHA3-384 SHA3-512	SHA-256 SHA-384 SHA-512	SHA-256 SHA-384 SHA-512 SHA-512/256 SHA3-256 SHA3-384 SHA3-512 SHA3-shake128[3] SHA3-shake256[3] Whirlpool-512 BLAKE-256 BLAKE-384 BLAKE-512	SHA-224 SHA-256 SHA-384 SHA-512 SHA-512/224 SHA-512/256 SHA3-224 SHA3-256 SHA3-384 SHA3-512 LSH-224 LSH-256 LSH-384 LSH-512 LSH-512-224 LSH-512-256

BCrypt를 사용하는 API의 TPS가 어떻게 될 것인가를 따져야 한다.

주로 회원가입, 로그인에 사용할텐데 TPS가 높지 않음

초당 수천명이 로그인, 회원가입을 하는 경우는 거의 없음

즉, BCrypt를 사용하는 API의 사용량이 많지 않으면 크게 문제되지 않음

멘토님은 LG전자에서 회원 시스템 팀에 계셨는데 기획 팀에서 법률 검토를 받은 후 알고리즘을 정해서 개발 팀에 전달했다고 한다.

법이랑 정책과 관련이 있는 부분

▼ 저는 다 구현한 후에 MSA를 적용해봤는데 처음부터 MSA 구조로 시작해야 하나요?

둘다 적용한다. 정답이 없다.

최근에는 설계 단계에서부터 MSA로 개발하는 경우가 많아지고 있다.

5~6년 전에는 Monolithic을 MSA로 바꾸는 작업을 많이 했다.

멘토님은 개인적으로 MSA로 먼저 시작하는 것을 추천하지 않는다고 하신다.

프로젝트는 계속해서 변화한다.

Monolithic보다 MSA는 관리적인 오버헤드가 크다.

이번 캠프에서 MSA로 하라고 한 이유는 실무에서 최근에 MSA 아키텍처를 많이 사용하니까 경험해보라는 취지

서비스들이 쪼개지는 것에 대해 고민을 해보고!

▼ 팀 프로젝트 때 Spring Data JPA는 가급적 사용하지 말라고 했는데 MyBatis를 쓰면 되나요?

직접 쿼리도 짜보고

MyBatis는 SQL Mapper, Spring Data JPA는 ORM

둘다 많이 쓰인다.

옛날 기술 쓰는 곳들은 MyBatis를 많이 쓰는 것 같다.

정확한 답변을 듣고 싶다면 캠프장님께 질문해보길

▼ 회원 탈퇴를 구현 할 때 아예 삭제해도 되나요? 서비스를 사용하다 보면 다음에 같은 계정으로 다시 가입할 수 없다는 방식을 많이 봤습니다. 삭제된 회원은 따로 보관을 해야 하나요?

기획, 서비스 정책에 달려있다.

회원 아이디는 DB에 남겨도 된다.

이메일, 주소, 실명같은 개인 정보는 DB에 남기면 안 된다.

이메일을 단방향 해싱해서 저장하면 복호화가 안 된다. ➡️ 회원 가입을 시도할 때 비교

회원 탈퇴를 하면 아예 row를 다 삭제하진 않고, UUID는 남겨둔다.

일주일 뒤에 회원 정보를 삭제하는 것은 탈퇴한 회원을 특정한 다른 테이블에 저장하고, 매일 밤 12시에 batch를 돌면서 진짜 탈퇴를 시키기도 한다.

▼ 회원 정보를 수정할 때 횟수에 제한이 있도록 구현을 하나요?

있을 수도 있다.

기획과 정책에 따라 다르다.

요구사항에서 나온다.

▼ 유저 관리처럼 관리자가 사용할 수 있는 기능은 권한이 없는 사용자에게 안 보여지는 게 나을까요? 아니면 접근을 시도하면 안 된다는 알림을 띄울까요?

요구사항

각각의 이유가 있어야 함

▼ Member 테이블에 언제 최종 접근했는지 알 수 있는 필드를 추가한다면 로그인 할 때 마다 갱신하나요?

그렇게 해도 되고 안 해도 된다.

필드에 last_login_time을 둘 수 있다.

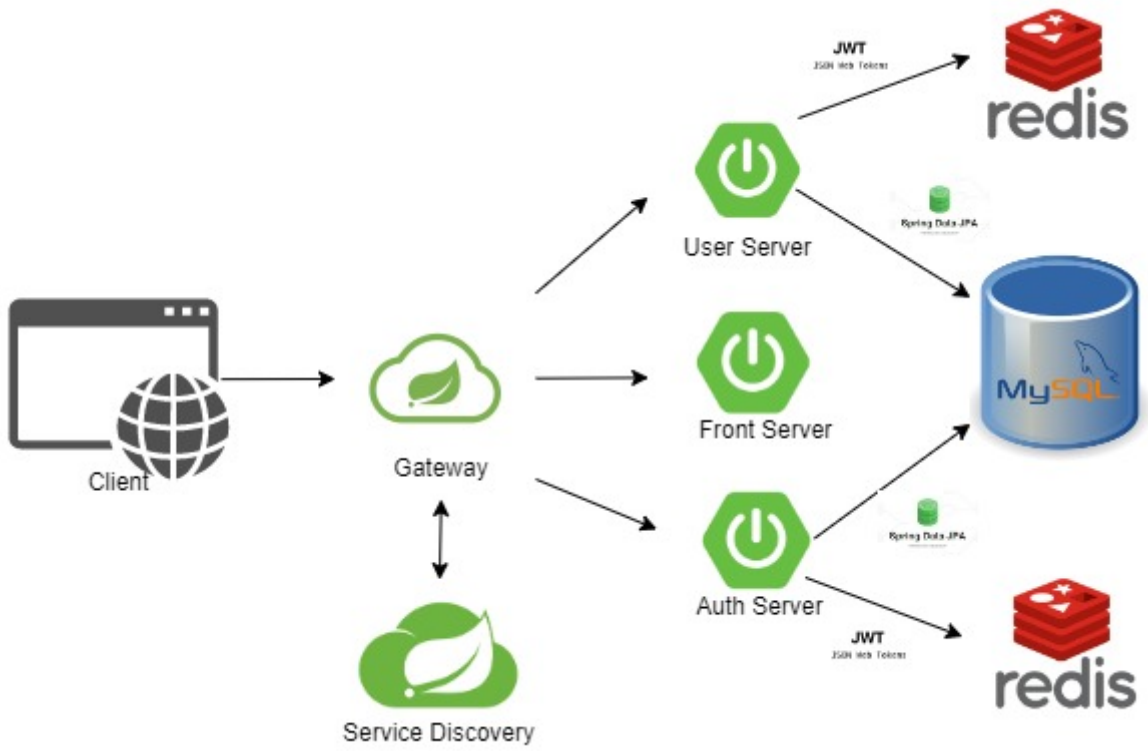
때에 따라서 login_history 테이블을 따로 두기도 한다. ➡️ 데이터 분석으로 사용할거면 권장

▼ 신규 서비스 성능 테스트를 할 때 vuser, TPS는 어떻게 산정하는지 궁금합니다.

성능 요구사항

admin같은 서비스는 성능이 안 중요함

▼ MSA 아키텍처 리뷰



괜찮음.

하지만, MSA의 Best practice는 서비스마다 독립적인 DB를 가지는 것

DB를 공유하게 되면 서비스들끼리 API 호출을 할일이 없어진다.

즉, 네트워크를 통한 API로만 통신을 해야 한다.

Q. 더 나눠야 하거나 잘못 나눠진 부분이 있나요?

정답 없음

Q. 프런트엔드만 서버를 따로 두었는데 관련 파일이 별로 없으면 생성하지 않는 게 나을까요?

모든 서비스는 크기와 상관없이 독립적으로 배포, 운영되어야 함

▼ 효율적인 아키텍처를 만들기 위해 사전에 어떤 점을 준비하면 좋은지 궁금합니다.

공부를 많이 해야 한다.

▼ 파일, 변수명을 작명하는데 좋은 방법이 있는지 궁금합니다.

clean code 책에서 2장이 의미 있는 이름 읽어보기

▼ 로그인 성공시 access token은 body에, refresh token은 cookie에 내려주도록 했습니다. 쿠키를 전달할 때 httpServletResponse 객체를 사용하게 되는데 이 객체를 Service 단까지 내려주는게 적절하지 않은 것 같아 controller ➡️ service 간 토큰 전달을 위해 TokenResponse라는 dto 객체를 사용해서 서비스에서 토큰 정보를 받아오고 컨트롤러에서 httpServletResponse을 사용해서 쿠키를 담는 식으로 구현했습니다. HttpServletResponse를 서비스까지 내려주는 게 더 나은 방법일까요?

아니요.

controller와 service는 역할, 책임이 다릅니다.

service는 HTTP와 관련이 없다.

▼ controller → service → repository 구조로 개발을 하였는데 이때 dto를 생성하는 곳에 dto 클래스를 두었다. controller에서 생성하는 응답 dto일 경우 controller.dto 패키지에 위치, service에서 생성하는 응답 dto일 경우 service.dto 패키지에 위치) 이렇게 하는 게 적절한 방법일지 궁금합니다.

정답은 없다.

dto를 어디에서 사용할건지에 따라서 넣을지 밖으로 뺄지 생각해보기

여러 곳에서 사용되면 밖으로 빼기

▼ 비밀번호 암호화에서 strength (해시 함수 몇 번 돌릴지)를 어떻게 결정하는지 궁금합니다. 보안과 api 응답 속도 사이에 트레이드 오프가 존재하는데 이 부분은 어떻게 결정하는 게 좋을까요?

요구사항을 설계할 때 결정

개발팀과 보안팀이 같이 결정

▼ 아키텍처 설계 과정이 궁금합니다. 예를 들어 DB 이중화나 캐시 서버 같은 것들은 처음 개발할 때부터 고려하는 것인지 아니면 성능 테스트를 하면서 개선점을 찾고 서버 성능을 올리기 위해 도입을 고민하는 것인지 궁금합니다.

요구사항 단계에서 결정

스타트업은 처음에는 DB 이중화를 안 한다.

큰 회사는 미리 다 세팅한다.

설계 단계에서도 정하고, 성능 테스트를 하면서 필요하면 인프라를 튜닝한다.

▼ 저의 경우 zuul 서버와 auth 서버가 따로 구동되고 있는데 이러한 경우에도 위와 같이 요청을 매번 보내도 괜찮을지 궁금합니다. 괜찮을지에 대해서 구체적으로 서술하자면 feign은 restTemplate와 같이 http 요청을 보내주는 것으로 알고 있습니다. 그렇기에 access 토큰 확인이 필요한 매 요청마다 http 요청을 보내는 것이 괜찮은지 궁금합니다. 이후에 role에 대한 확인을 하는 것도 access 토크 검증과 같은 과정을 거치기에 auth 서버는 api gateway 서버와 합치는 것이 더 좋은 선택일지도 궁금합니다.

zuul 서버는 단일화된 end-point 역할, 공통화된 기능(로깅, 인증, 인가) 처리

합치지 않는 게 맞다.

API Gateway는 최대한 가볍게 만든다.

▼ token을 조회하고 그 토큰의 expireTime 그리고 redis에 저장될 accessToken의 expiredTime을 갱신하여 주었습니다. 즉 토큰 값은 변화가 없고 expireTime만 변화가 있는 것 입니다. expireTime만 변경시키고 token 값은 유지하는 것이 괜찮은지 궁금합니다.

토큰 로테이션

일반적으로 로테이션 하는 것을 권장하기도 한다.

▼ jwt 토큰에 어느 정도의 정보가 들어가도 괜찮은지 궁금합니다.

개인정보, 보안에 민감하지 않은 정도

▼ email 인증을 위해 uuid와 key와 함께 email 서버에 auth email을 보내달라는 요청을 kafka를 통해 전달하는 코드입니다. 코드를 작성하고 난 이후에 이 경우에는 feign과 같이 ‘요청을 하는 것이 좋은가?’ 아님 위에 구현한 것처럼 ‘message, event’를 발생시키는 것이 좋은가?’하는 생각이 들었습니다. 제가 공부한 바로는 요청의 경우 return 값이 필요한 경우 그렇지 않은 경우는 event, message가 알맞다는 생각을 하였습니다. kafka와 같은 툴은 언제 사용하면 좋은지 가이드가 필요합니다. 또 위의 kafka의 경우 가장 기본적인 설정만 하였는데 kafka를 사용할 때 주의해야할 점이 어떤 것이 있는지 궁금합니다.

Event Driven Architecture를 왜 사용하냐를 알아야한다.

Coupling 때문에 사용한다.

각각의 서비스들간의 결합도를 낮출려고 비동기를 사용

▼ 단순한 boolean 값을 반환할 경우나 void의 경우에도 다음과 같이 통일하여 결과를 반환하여 주어야 할까요? 또 저는 Front에게 반환하는 정보는 ApiResponse를 통해 반환하고 아닌 경우 단순 DTO로 반환하였는데 이는 좋은 판단인지 궁금합니다.

```
{
  timeStamp : 타임스탬프,
  code : API별 성공코드,
  message : 성공 메시지,
  data : {데이터}
}
```

일관성이 중요하다. ApiResponse를 반환하는 것을 권장

▼ DTO는 어떠한 계층까지 사용하는 것이 좋은지 궁금합니다. 지금 저의 경우 DTO를 통해 controller에서 service 계층으로 데이터를 이동시키는 데 이전에는 DTO를 controller단에만 국한시켰습니다. DTO, 즉 data Transfer object이기에 data를 이동시키는 것이 이해하지만 그 계층이 어디까지가 적절한지 궁금합니다.

내가 하는 것에 대해 내부적인 근거가 있으면 된다.

이름에 너무 집착할 필요는 없다.

▼ GroupController는 HTTP Method를 다양하게 사용하여 구현했습니다. restful한 API를 구현하기 위한 조건 중 하나로 url에 동사가 들어가지 않는다는 조건이 있는 것으로 알고 있습니다. 그리고 최대한 Post와 Get Method만 사용하는 것으로 알고 있습니다. 그렇다면 Post와 Delete는 어떻게 구분할 수 있을지 url을 명령하는 팁이 궁금합니다.

https://stripe.com/docs/api

기업이 작성한 API를 참고하자.

동사를 쓰면 안 된다에 집착하지마.

다른 지역에서 자주 나오는 질문

1. 도메인 로직 분리에 대한 TIP과 예시
    a. 정답 x. 많은 것을 시도해보세요.
    b. 요구사항을 쭉 정리해보자.
        i. 사용자 스토리에 대해 알아보기
            1. 자주 등장하는 명사를 뽑기 ➡️ Entity가 될 가능성
2. 아키텍처 설계의 핵심
    a. Responsibility, Coupling, Cohesion, Dependency
3. SRP
    a. 각 클래스, 메소드, API 등은 명확한 하나의 책임/역할을 갖고 있어야 함
4. Low Coupling and High Cohesion
    a. 클래스 간의 의존성은 최대한 적어야 함, 다른 곳의 변경으로부터 영향을 최대한 적게 받아야 함
    b. 각 클래스는 꼭 필요한, 서로 협력하는 필드와 메소드로 구성되어야 함
5. Dependency Management
    a. 의존을 어떤 방법으로 할 것인지?
    b. 의존성의 수준을 어떻게 관리 할 것인지?
    c. 그래픽으로 나타내서 볼 수도 있다.
6. 서비스에 들어가는 메소드의 범위가 궁금합니다. 예를 들어, 컨트롤러에서 이메일 검증의 경우 “컨트롤러에서 따로 메서드로 구현하는지” 아니면 “서비스에 따로 클래스를 만들어 관리”가 맞는지 궁금합니다.
    a. https://blog.frankdejonge.nl/where-does-validation-live/
    컨트롤러에서는 shapes, types만 체크
    멘토님은 validation은 service에서 해야 한다고 하셨다.

셀프 코드리뷰 가이드

1. 네이밍
    a. 이름은 누구나 이해하기 쉽고 일관되게 작성해야 한다.
    b. 다수가 협업시에 동사와 명사의 일관된 사용이 중요
    c. 이름은 길더라도 명확한 의미를 담고 있어야 함
2. 제어문
    a. 가능한 2 depth까지 가지 않도록 작성

b. Guard를 활용

```
public String getCookieValue(Cookie[] cookies, String cookieName) {
        if(cookies != null) {
                for (Cookie cookie : cookies) {
                        if (cookie.getName().equals(cookieName)) {
                                return cookie.getValue();
                        }
                }
        }
        return null;
}
```

c. Cyclomatic Complexity

i. 프로그램의 제어 흐름을 node, edge로 표현한 그래프를 기반으로 계산

3. 검증 로직과 비즈니스 로직의 분리

```
public void saveUser(email, password) throws UserException {
    validateEmailFormat(email)
    validateEmailDuplication(email)
    validatePassword(password)

    User user = new User(email, password)
    user.save()
}
```

4. 로그를 남길 때는 logging framework를 쓸 것
5. 불필요한 주석 금지

신입은 주로 **개발, 기술, 코드**에만 집중한다.

이것조차도 사실은 어렵다. 익숙해지려면 시간이 오래 걸리니까

경력, 인사이트가 생기면 **아키텍처, 테스팅, 품질, 요구사항, 운영, 유지보수**를 보게 된다.

단순히 코드를 잘 짜는 것이 아닌 전체적인 흐름을 아는 게 좋은 개발자가 되는 방법이다.

요구사항 ➡️ 아키텍처 ➡️ 개발/코드/기술 ➡️ 테스트 /품질 ➡️ 운영

고객과 회사를 만족시키는 제품을 어떻게 만들 것인가

아키텍처는 요구사항에 의해서 도출된다.

요구사항은 기능 요구사항, 비기능 요구사항(보안, 성능 등)이 있다.

모든 아키텍처는 Trade-off가 있다.

📚 멘토님의 책 추천

1. 클린 아키텍처
2. 만들면서 배우는 클린 아키텍처
3. 리팩터링 2판
4. 개발자에서 아키텍트로
5. 소프트웨어 아키텍처 101
6. 엔터프라이즈 애플리케이션 아키텍처 패턴
7. 아파치 카프카 애플리케이션 프로그래밍 with 자바
8. 가상 면접 사례로 배우는 대규모 시스템 설계 기초

📌 멘토님의 블로그 추천

1. https://martinfowler.com/

🖥️ 멘토님의 사이트 추천

https://github.com/mauricioaniche/ck

https://refactoring.guru/refactoring/smells