

포트폴리오

---

# 오의석

궁금한 건 못 참는 백엔드 개발자

우보만리(牛步萬里) 나아가겠습니다





# 00

소개



# 01

프로젝트 1  
JUSTUDY



# 02

프로젝트 2  
산너머



# 03

프로젝트 3  
XYZ



# 04

기술 스택

4



# 오의석

생년월일 : 1995.05.12

이메일 : uiseok0512@naver.com

좌우명 : 반성은 하되, 후회는 하지 말자.



## About Me

**우직**

끈임없이 될때까지 도전합니다.

**호기심**

새로운 기술이 있으면 꼭 써보고 싶어합니다.

**즐거움**

사소한 것에도 그 속에서 즐거움을 찾습니다.

## 경력(임베디드 리눅스 개발)

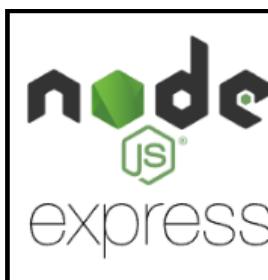
- 품질검사 기기
- Prinker M 제품 설계 ([바로가기](#))
- 제품 시리얼 라벨기(B2B)



- MVC 패턴의 웹사이트를 만들 수 있습니다.
- RESTful한 API를 설계할 수 있습니다.



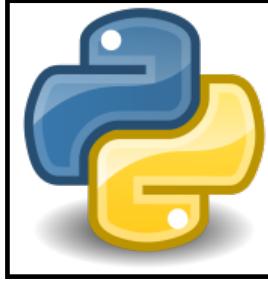
- 데이터베이스와 매핑하여 CRUD를 할 수 있습니다.
- 간단한 API를 호출할 수 있습니다.



- 데이터베이스와 매핑하여 CRUD를 할 수 있습니다.
- 간단한 프로젝트를 만들 수 있습니다.



- 모던 자바로 코드를 개발 할 수 있습니다.
- 알고리즘 구현이 가능합니다.



- 다양한 라이브러리를 활용할 수 있습니다.
- 간단한 프로젝트를 만들 수 있습니다.



- SPI, USART, I2C 등 모듈 통신이 가능합니다.
- 임베디드 리눅스 환경에서의 디바이스 드라이버 개발이 가능합니다.

# JUSTUDY

## 온라인 스터디 사이트

### 소개

IT 개발자 지망생들의 필요한 기술 스택 공부와  
온라인 스터디 모집 및 참여를 독려하며,  
자기 주도 학습을 유도하는 온라인 스터디 사이트 입니다.

01

+

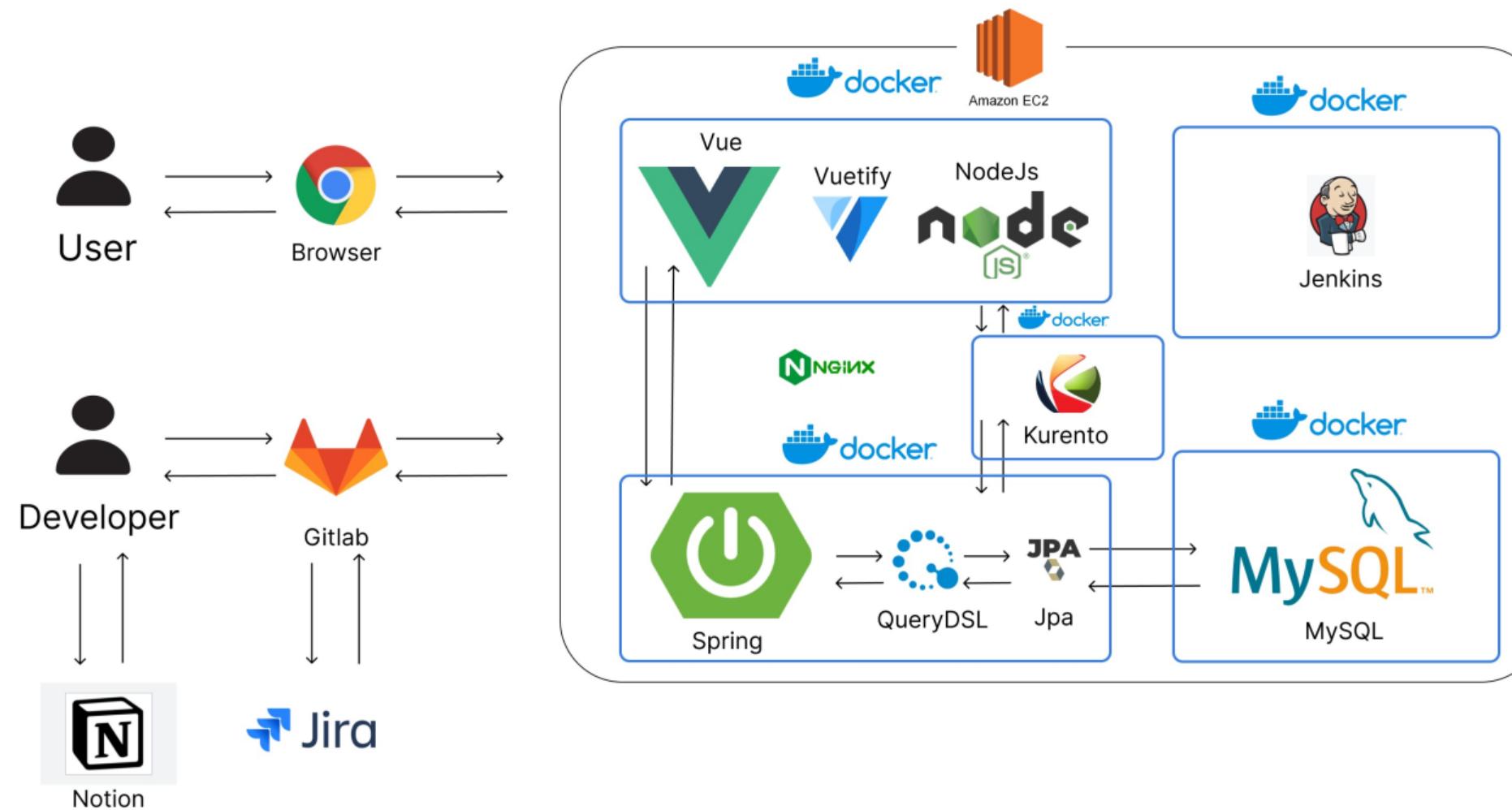
**<메인화면>**

**<공부시간 측정 화면>**

**<화상회의 화면>**

JUSTSTUDY

작업 기간	2023.01.09 ~ 2023.02.17
인력 구성	BE 3명 / FE 3명 (BE 기여도 35%)
프로젝트 목적	온라인 스터디 사이트 개발
프로젝트 내용	스터디원을 모집하고 화상으로 공부할 수 있는 사이트입니다. 더 나아가, 혼자의 개인공부 시간을 측정하여 기록할 수 있습니다.
주요 업무 및 상세 역할	<ul style="list-style-type: none"> <li>[백엔드+프론트엔드] WebRTC를 이용한 화상회의 구현 (화면 구성 + 시그널링 서버 구현)</li> <li>[백엔드] 스케줄러를 이용하여 랭킹 시스템 구현.</li> <li>[백엔드] 공부시간, 랭킹 테이블 설계 및 API 구현.</li> <li>[백엔드+프론트엔드] FaceApi를 이용하여 얼굴인식에 따른 공부시간 측정.</li> </ul>
사용언어 및 개발환경	Spring, QueryDSL, JPA, MySQL, Kurento
참고 자료	<a href="https://github.com/ohuiseok/SPRING_WEBRTC">https://github.com/ohuiseok/SPRING_WEBRTC</a>



## Kurento 선정 이유

Kurento는 저수준 플랫폼이지만, 오픈 소스로써 사용의 제한이 없어졌습니다. 또한, 소스코드를 수정함으로써 확장성이 용이해졌습니다. 실제로 WebRTC의 DataChannel를 에디터에 연결하여, 실시간 동시편집이 가능한 에디터 기능을 추가했습니다.

(OpenVidu와 같은 고수준 플랫폼을 사용하면 많은 기능들이 유로 버전으로 구성되어 있으며, 실제 서비스를 배포할 때 회사간의 협약이 필요했습니다.)

## JPA와 QueryDSL 선정 이유

JPA를 사용하면, ORM 기능과 트랜잭션 관리의 이점이 있습니다. QueryDSL의 경우에는, 동적 쿼리 구성이 가능해지며, 코드로써 SQL문을 구성하기 때문에 유지보수에도 유용합니다.

## 메인 기능 설명

WebRTC를 연결을 위한 설정

### 상세 내용

- WebRTC 통신을 위해 WebSocket을 이용하여 서로의 데이터를 주고 받았습니다.
- 스프링이 시그널링 서버가 되어 해당 방과 해당 멤버를 매칭하여 kurento서버와 연결시켰습니다.
- 또한, 추가 기능으로 '조용히 시키기, 밴시키기, 제비뽑기' 기능을 구현했습니다.
- WebRTC의 DataChannel을 프론트엔드의 에디터 터와 연결 시켜, 실시간 동시 편집 기능도 넣었습니다.

```
//응답에 대한 리스너 함수
setEventListener({ state, dispatch }) {
  state.webSock.onmessage = (message) => {
    let parsedMessage = JSON.parse(message.data);
    console.info("Received message: " + message.data);
    console.info("Received message: " + parsedMessage.id);

    switch (parsedMessage.id) {
      case "existingParticipants":
        dispatch("onExistingParticipants", parsedMessage);
        break;
      case "newParticipantArrived":
        dispatch("onNewParticipant", parsedMessage);
        break;
      case "participantLeft":
        dispatch("onParticipantLeft", parsedMessage);
        break;
      case "receiveVideoAnswer":
        dispatch("receiveVideoResponse", parsedMessage);
        break;
      case "requestBanVote":
        dispatch("requestBanVote", parsedMessage);
        break;
      case "requestMuteVote":
        dispatch("requestMuteVote", parsedMessage);
        break;
      case "requestExitVote":
        dispatch("requestExitVote");
        break;
      case "ladderResult":
        dispatch("receiveLadderResult", parsedMessage);
        break;
      case "receiveChatMessage":
        dispatch("receiveChatMessage", parsedMessage);
        break;
      case "iceCandidate": //receiveChatMessage
        console.log("iceCandidate");
        console.log(parsedMessage);
        console.log(state.participants);
        state.participants[parsedMessage.name].rtcPeer.addIceCandidate(
          parsedMessage.candidate,
          function (error) {
            if (error) {
              console.error("Error adding candidate: " + error);
              return;
            }
          }
        );
    }
  }
}

@Service
@ServerEndpoint(value = "/groupcall", configurator = ServerEndpointConfig.class)
public class GroupCallService {

  private static final Logger log = LoggerFactory.getLogger(GroupCallService.class);

  private static final Gson gson = new GsonBuilder().create();

  @Autowired
  private RoomManager roomManager;

  @Autowired
  private UserRegistry registry;

  @OnMessage
  public void OnMessage(Session session, String message) throws Exception {
    final JsonObject jsonMessage = gson.fromJson(message, JsonObject.class);

    UserSession user = registry.getBySession(session);

    if (user != null) {
      log.info("Incoming message from user '{}': {}", user.getName(), jsonMessage);
    } else {
      log.info("Incoming message from new user: {}", jsonMessage);
    }

    switch (jsonMessage.get("id").getAsString()) {
      case "joinRoom":
        joinRoom(jsonMessage, session);
        break;
      case "receiveVideoFrom":
        final String senderName = jsonMessage.get("sender").getAsString();
        final UserSession sender = registry.getByName(senderName);
        final String sdpOffer = jsonMessage.get("sdpOffer").getAsString();

        user.receiveVideoFrom(sender, sdpOffer);
        break;
      case "leaveRoom":
        leaveRoom(user);
        break;
      case "ban":
        ban(jsonMessage.get("name").getAsString(), jsonMessage.get("room").getAsString());
        break;
      case "mute":
        mute(jsonMessage.get("name").getAsString(), jsonMessage.get("room").getAsString());
        break;
      case "exit":
        allExit(jsonMessage.get("room").getAsString());
        break;
    }
  }
}
```

# 산너머 등산로 추천 사이트

## 소개

설문 조사를 바탕으로 등산로를 추천해주며,  
산, 등산로, 주변정보를 보여주는 사이트 입니다.

**산너머**

등산하고 싶은 산을 찾아보세요!

**봄 명산으로 떠나봐요!**

북한산(백...), 비슬산, 가까운 산으로 바로 추천

관악산, 청계산, 설악산

**대모산**

대모산 제1코스

전체, 화장실, 운동, 음수대, 주차장, 시종점

1km kakao

**산너머**

람쥐의 추천!

어떤 등산 스타일을 가지고 계신가요?  
람쥐가 산을 추천해줄거예요!

어느 단계의 산을 원하시나요?

동산, 관악산, 한라산

어디에 있는 산을 가고싶나요?

서울, 경기, 강원  
충청, 전라, 경상  
제주

**산너머**

추천하는 대표 등산로

대모산 제6코스  
1.04km 19분 예상

위의 등산로를 기준으로 유사한 등산로를 뽑아봤어요.

용암산 제6코스  
1.03km 26분 예상

불곡산 제1코스  
1.23km 26분 예상

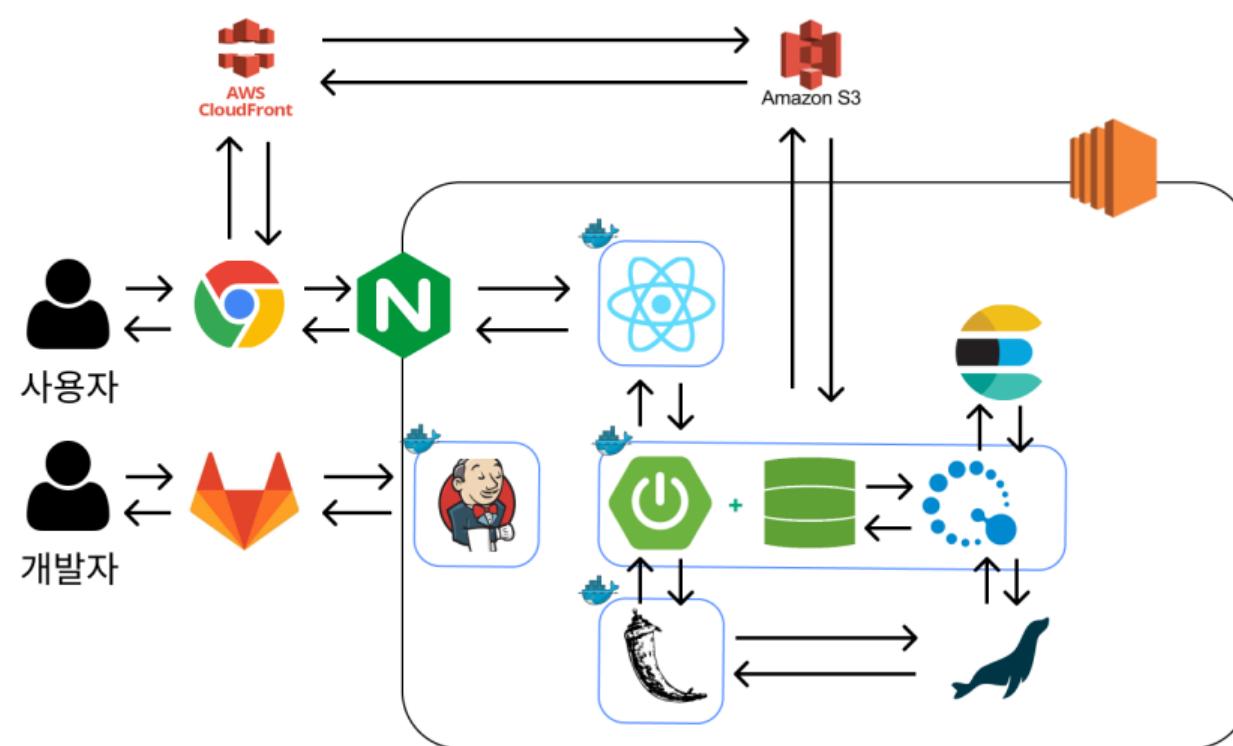
칠보산 제7코스  
1.68km 30분 예상

불곡산 제4코스  
2.14km 37분 예상

용암산 제3코스  
2.02km 34분 예상

작업 기간	2023.02.20 ~ 2023.04.07
인력 구성	BE 4명 / FE 2명 (BE 기여도 40%)
프로젝트 목적	등산로 추천 사이트 개발
프로젝트 내용	등산을 하고 싶은 사용자에게 맞춤형 등산로 정보를 제공하는 서비스입니다.
주요 업무 및 상세 역할	<ul style="list-style-type: none"> <li>[CI/CD] Gitlab, Jenkins, Docker를 이용한 자동배포</li> <li>[백엔드] Express 프레임워크에서 코스 데이터 전처리</li> <li>[백엔드] python을 이용하여 모든 좌표의 고도 데이터 전처리</li> <li>[백엔드] ElasticSearch 검색 엔진을 이용한 산 이름 출력</li> <li>[백엔드] 두 좌표(위도, 경도) 사이 거리 계산을 이용한 SQL문 구현</li> </ul>
사용언어 및 개발 환경	Spring, QueryDSL, JPA, MariaDB, Express, Jenkins, Docker, ElasticSearch
참고 자료	<a href="https://github.com/ohuiseok/Sanneomeo">https://github.com/ohuiseok/Sanneomeo</a>

## 서비스 아키텍처



## ElasticSearch 선정 이유

Nori 플러그인을 설치함으로써 한국어를 토큰화할 수 있어, 단어를 분활하여 검색할 수 있습니다.

또한, ElasticSearch는 검색엔진으로 자동으로 인덱싱하여 효율적인 검색을 가능하게 하는 반전 인덱스가 존재합니다.

## Flask 선정 이유

마이크로 프레임워크라서 미니멀한 디자인과 학습 곡선이 작아 가볍고 유연하여 적용하게 되었습니다. 또한, 필요한 특정 도구와 라이브러리를 선택할 수 있으므로 프로젝트 복잡성이 줄어들고 잠재적으로 성능이 향상 될 수 있기 때문입니다.

## 메인 기능

가장 가까운 등산로를 뽑는 API

## 문제 상황

웹 어플리케이션 단계에서 데이터를 뽑은 후, 계산하고 정렬할 경우 8초의 시간이 소요 되었습니다.

## 문제 해결

- 상위 테이블인 산에도 좌표값을 넣었습니다.

(산테이블->등산로테이블->좌표테이블)

- 가장 가까운 산을 먼저 찾고, 해당 산에서 가장 가까운 좌표를 찾았습니다.

- 해당 좌표가 존재하는 등산로 코스데이터를 출력했습니다.

- SQL문에서 거리 계산을 하였으며, 가까운 순으로 정렬하여 데이터를 추출하였습니다.

```

@Override
public Optional<NearMountainResponseDto> findMountainSequenceByDistance(BigDecimal latitude, BigDecimal longitude) {
    NumberExpression<Double> distanceExpression = acos(sin(radians(Expressions.constant(latitude)))
        .multiply(sin(radians(mountain.latitude)))
        .add(cos(radians(Expressions.constant(latitude)))
            .multiply(cos(radians(mountain.latitude)))
            .multiply(cos(radians(Expressions.constant(longitude)).subtract(
                radians(mountain.longitude))))))
        .multiply(6371000));
    Path<Double> distancePath = Expressions.numberPath(Double.class, "distance");

    return Optional.ofNullable(
        queryFactory
            .select(
                Projections.constructor(NearMountainResponseDto.class, mountain.mountainSeq,
                    Expressions.as(distanceExpression, distancePath)))
            )
            .from(mountain)
            .rightJoin(course)
            .on(mountain.mountainSeq.eq(course.mountainSeq))
            .groupBy(mountain.mountainSeq)
            .orderBy((ComparableExpressionBase<Double>) distancePath).asc())
            .fetchFirst());
}

@RequiredArgsConstructor
public class CourseRepositoryImpl implements CourseRepositoryCustom {
    private final JPAQueryFactory queryFactory;
    @Override
    public Optional<NearTrailResponseDto> findNearTrailByMountainSequence(String sequence,
        BigDecimal latitude, BigDecimal longitude) {
        NumberExpression<Double> distanceExpression = acos(sin(radians(Expressions.constant(latitude)))
            .multiply(sin(radians(trailPath.latitude)))
            .add(cos(radians(Expressions.constant(latitude)))
                .multiply(cos(radians(trailPath.latitude)))
                .multiply(cos(radians(Expressions.constant(longitude)).subtract(
                    radians(trailPath.longitude))))))
            .multiply(6371000));
        Path<Double> distancePath = Expressions.numberPath(Double.class, "distance");

        return Optional.ofNullable(
            queryFactory
                .select(
                    Projections.constructor(NearTrailResponseDto.class, course.courseSeq, trailPath.latitude, trailPath.longitude,
                        Expressions.as(distanceExpression, distancePath)))
                )
                .from(mountain)
                .leftJoin(course)
                .on(mountain.mountainSeq.eq(course.mountainSeq))
                .leftJoin(courseTrails)
                .on(course.courseSeq.eq(courseTrails.courseSeq))
                .leftJoin(trail)
                .on(courseTrails.trailSeq.eq(trail.trailSeq))
                .leftJoin(trailPath)
                .on(trail.trailSeq.eq(trailPath.trailSeq))
                .where(mountain.mountainSeq.eq(sequence))
                .groupBy(course.courseSeq)
                .orderBy((ComparableExpressionBase<Double>) distancePath).asc())
                .fetchFirst());
    }
}

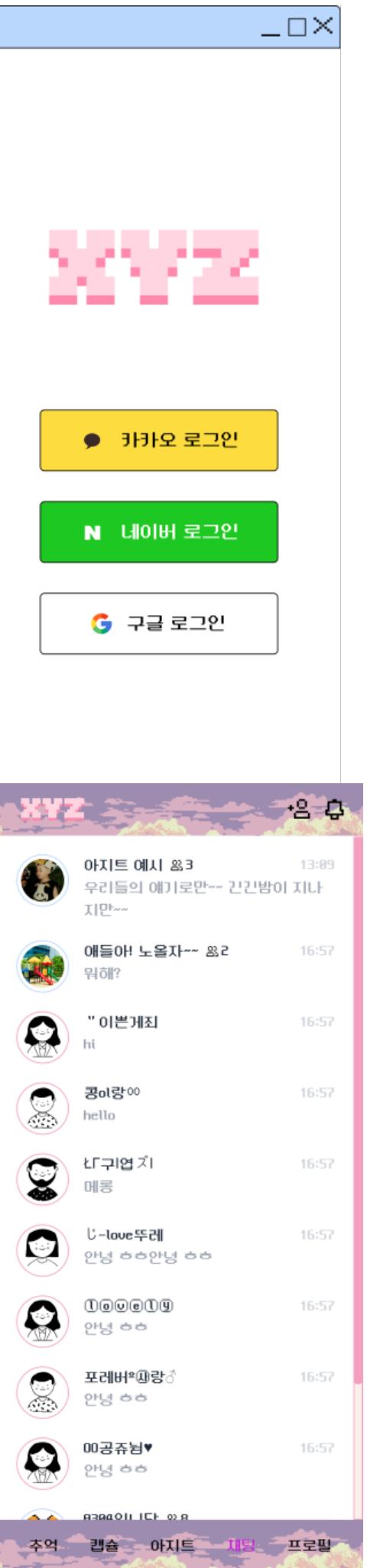
```

# XYZ

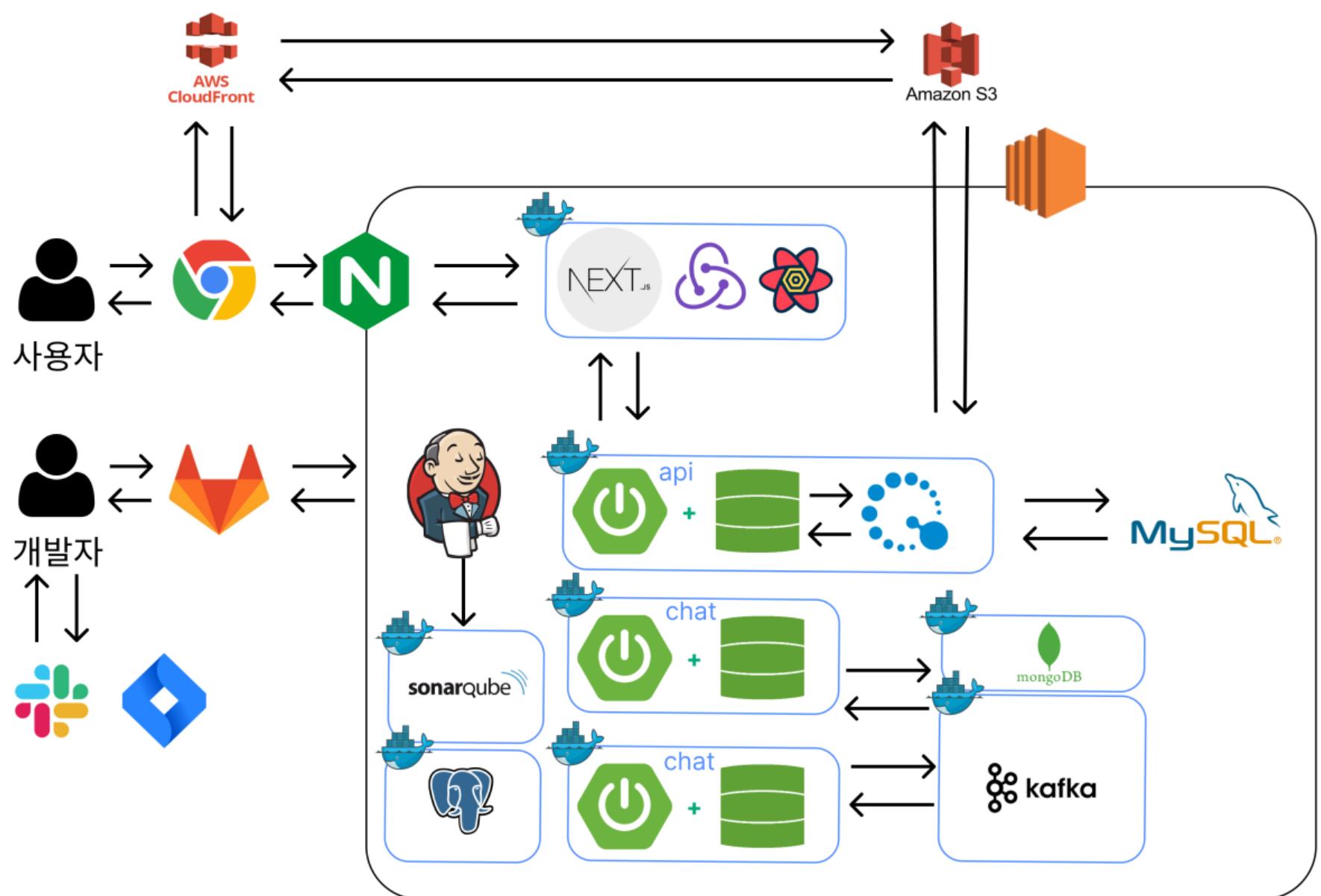
## 레트로 SNS 사이트

### 소개

친구들끼리 추억을 공유할 수 있으며,  
과거와 현재에 대해 이야기를 나눌 수 있는 공간을 제공하는 사이트입니다.



작업 기간	2023.04.10 ~ 2023.05.19
인력 구성	BE 3명 / FE 3명 (BE 기여도 33%)
프로젝트 목적	타임캡슐 기반의 레트로 SNS 사이트
프로젝트 내용	그룹기반의 SNS로, 추억앨범 저장 기능과 타임캡슐 기능이 있습니다. 그 시절 감성 프로필과 채팅이라는 부가기능도 있습니다.
주요 업무 및 상세 역할	<p>[CI/CD] 무중단 배포 및 소나큐브를 이용한 정적분석</p> <p>[백엔드] S3를 이용한 파일 관리.</p> <p>[백엔드] 스프링 시큐리티, OAuth2.0, JWT를 이용한 소셜 로그인.</p> <p>[백엔드] Kafka와 Webflux의 SSE를 이용한 채팅 서비스.</p> <p>[백엔드] 유저 및 마이룸 API 구현</p>
사용언어 및 개발 환경	Spring, QueryDSL, JPA, MySQL, MongoDB, Jenkins, Docker, Kafka, WebFlux, Spring Security
참고 자료	<a href="https://github.com/ohuiseok/Retro_SNS">https://github.com/ohuiseok/Retro_SNS</a>



## SonarQube & PostgreSQL 선정 이유

소나큐브 7.8부터는 MySQL을 지원하지 않게 되어 PostgreSQL을 사용하게 되었습니다.

## Kafka

WebSocket을 사용할 경우, 소켓의 최대 할당량이 조 존재하여 많은 EC2서버가 필요해 집니다.  
따라서 메세지큐와 SSE를 이용하여, 극복하고자 했습니다.

## MongoDB

채팅은 엄청나게 많은 데이터가 쏟아져 나오며, 이를 모두 관계형 데이터베이스로 처리하는데는 다소 비효율적이고 어려운 부분이 있습니다.  
또한, 스키마가 존재하여 그 형식을 엄격히 지켜야 합니다.  
따라서 json 형식으로 저장하는 MongoDB가 더 적합하다고 생각이 들었습니다.

## 채팅부분을 여러 도커로 나눈 이유

메세지큐와 SSE를 이용해도 EC2 인스턴스의 성능에 따라 한계가 존재합니다.  
결국 인스턴스가 추가로 필요할 것으로 생각했습니다.  
더 나아가 MSA 구조를 사용해야 할 수도 있다는 생각이 들어서, 실험적으로 채팅 기능만 따로 분리하였습니다.

## 메인 기능

Kafka와 SSE를 이용한 채팅 구현

### 상세 내용

- 채팅방에 들어가면 MongoDB에서 채팅 내용을 불러옵니다.
- Post API로 메세지를 전달하면, MongoDB에 저장과 동시에 Kafka로 들어갑니다  
(파일일 경우 파일을 S3에 저장하고, 해당 URL과 파일 태입을 저장 및 송신합니다.)

- kafka에서 나온 데이터는 채팅방에 맞게 SSE로 전달됩니다.

```

@RestController
@RequiredArgsConstructor
@RequestMapping("/chat")
@Slf4j
public class ChatRestController {
    private final MongoDBService mongoDBService;
    private final Producer producer;
    private final Consumer consumer;
    private final S3UploadService s3UploadService;

    @GetMapping("/history")
    List<Chat> getHistoryChat(@RequestParam(name = "room") String room, @RequestParam(name = "id", required = false) Long id) {
        log.error("getHistoryChat {} {}", room, id);
        if (null == id)
            return mongoDBService.getHistory(room);
        else
            return mongoDBService.getHistory(room, id);
    }

    @PostMapping("/message")
    String transferChar(@RequestBody KafkaMessage kafkaMessage) {
        producer.orderSend("xyz", kafkaMessage);
        return "SUCCESS";
    }
}

@Service
@Slf4j
public class Consumer {
    private final Map<String, DirectProcessor<KafkaMessage>> processorMap = new ConcurrentHashMap<>();
    private final Map<String, Flux<KafkaMessage>> fluxMap = new ConcurrentHashMap<>();

    public Flux<KafkaMessage> getKafkaMessages(String room) {
        return fluxMap.computeIfAbsent(room, r -> {
            DirectProcessor<KafkaMessage> processor = DirectProcessor.create();
            processorMap.put(r, processor);
            return processor.share();
        });
    }

    @KafkaListener(topics = "xyz", groupId = "${kafka.group.id:${random.uuid}}", containerFactory = "listenerContainer")
    public void updateQty(KafkaMessage kafkaMessage) throws Exception {
        log.info("CONSUME PAYLOAD : {}", kafkaMessage);

        // Send the Kafka message to all subscribers in a specific room
        String room = kafkaMessage.getRoom(); // Assuming KafkaMessage has a getRoom method
        DirectProcessor<KafkaMessage> processor = processorMap.get(room);
        if (processor != null) {
            processor.onNext(kafkaMessage);
        }
    }
}

@Service
@Slf4j
@RequiredArgsConstructor
public class Producer {
    private final KafkaTemplate<String, KafkaMessage> kafkaTemplate;
    private final MongoRepository mongoDBRepository;
    private final SequenceGeneratorService sequenceGeneratorService;

    public void orderSend(String topic, KafkaMessage message) {
        LocalDateTime localDateTimeNow = LocalDateTime.now();
        ZonedDateTime zonedDateTime = ZonedDateTime.now(ZoneId.of("Asia/Seoul"));
        String time = zonedDateTime.format(DateTimeFormatter.ofPattern("yyyy-MM-
        Long curId = sequenceGeneratorService.generateSequence(Chat.SEQUENCE_NAME);

        message.setTime(time);
        message.setId(curId);
        kafkaTemplate.send(topic, message);
        log.info("Kafka Producer send data from the order service = {}", message);
        mongoDBRepository.insert(
            Chat.builder()
                .id(curId)
                .text(message.getText())
                .name(message.getName())
                .room(message.getRoom())
                .time(time)
                .type("text")
                .build());
    }
}

```

# 기술 스택

# 기술 스택

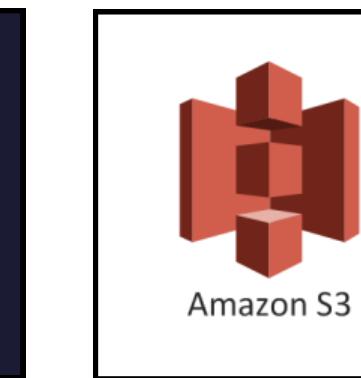
04



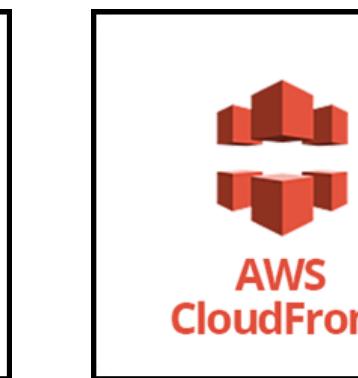
Spring Security



JWT  
JSON WEB Token



Amazon S3



AWS  
CloudFront



Amazon  
EC2



Spring Webflux



NGINX ®



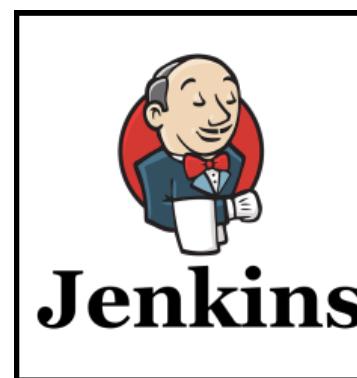
MySQL®



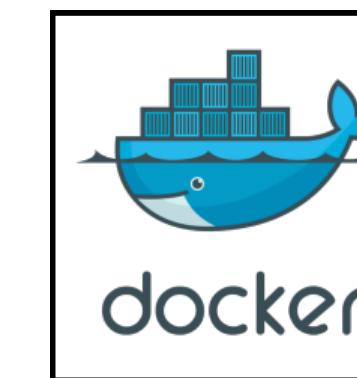
MariaDB



mongoDB



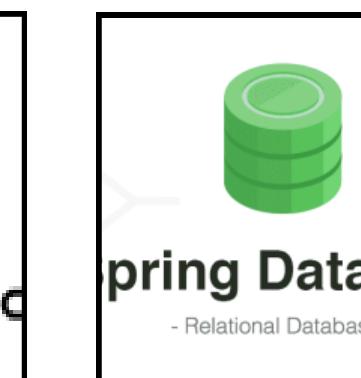
Jenkins



docker

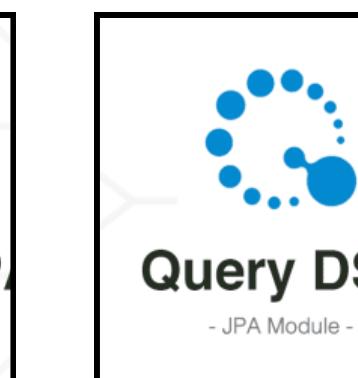


Kurento



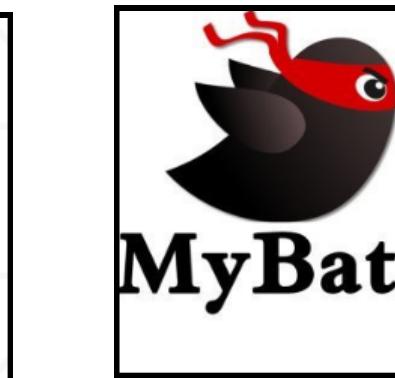
Spring Data JPA

- Relational Database -

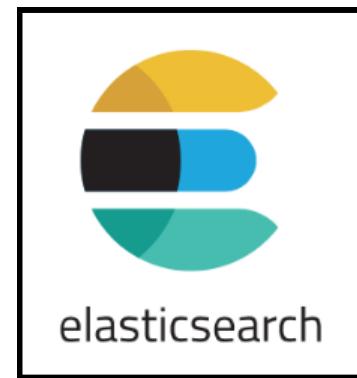


Query DSL

- JPA Module -



MyBatis



elasticsearch



포트폴리오 사이트

<https://uiseok-portfolio.netlify.app/>

THANK YOU!

읽어주셔서 감사합니다.