“ 긍정적인 변화를 만드는 개발자 곽영헌입니다.“

Github : https://github.com/YoungHoney

블로그 : https://younghoney.github.io/

기본정보
Personal Information

이름	곽영헌
생년월일	2000. 05. 15
연락처	010-9889-5275
메일	yhyh5275@naver.com
병역	대한민국 육군 병장 만기전역 (2020.03~2021.09)

학력사항
Education

대학교	2019.03~2025.02 (졸업 예정) 중앙대학교 소프트웨어학과	4.1/4.5
고등학교	2016.03 ~ 2019.02 한솔고등학교 졸업	-

주요 수강 과목
Major Course

알고리즘(3-1)	DP, Greedy, Graph에 관한 알고리즘을 학습	A+
운영체제(3-1)	OS의 자원관리, 프로세스, 동시성 제어에 대해 학습	A+
리눅스 시스템 응용 설계(3-2)	리눅스 커널의 구성과 시스템콜, 프로세스와 스레드, 동시성 제어와 스케줄링 알고리즘을 학습하고 실습을 진행	A+
데이타베이스시스템 (4-1)	관계형 DB의 인덱싱, 쿼리최적화를 학습하고 간단한 블록단위 I/O기반의 DB시스템을 설계 및 구현	A
네트워크응용설계 (4-1)	TCP의 신뢰성 보장, 흐름제어, 혼잡제어의 원리를 학습하고 go언어로 소켓 통신 프로그램을 만들며 실습을 진행	A

자격증 및 어학
Certifications & Language Proficiency

자격증	TOPCIT(608, 수준3)
어학	TOEIC Speaking AL(160)
	TOEIC 880

기술 스택
Skill Set

Programming Language	Java
DB	MariaDB
Framework	Spring
Collaborations	Git, Github
	Jira, Confluence
Cloud	AWS

주요 프로젝트

우리조상알기 - 캡스톤디자인(1)

“우리 집안 우리 가족 조상정보 찾기 서비스“

작업 기간 : 2023.09.01-2023.12.21

작업인원 : 백엔드 2(본인 포함), 프론트1

맡은 역할 : ERD설계, 서비스 코드 작성, 배포

사용 메뉴얼 링크

주요 기술 키워드

1. Java
2. Spring, Spring JPA
3. AWS EC2, RDS-MariaDB
4. Azure OpenAI API
5. Git, Github

이지백엔드 - 캡스톤디자인(2)

“쉬운 백엔드 개발을 위한 자동 배포, 배포관리 서비스“

작업 기간 : 2024.03.01-2024.06.14

작업인원 : 백엔드 2(본인 포함), 프론트1

맡은 역할 : TypeScript를 사용한 Terraform 스크립트 실행 및 관리

주요 기술 키워드

1. Terraform
2. AWS EC2
3. Git, Github

Yesable - 한이음 멘토링프로젝트

“개인화된 장애인 채용공고 플랫폼“

작업 기간 : 2024.04.01-2024.10.31

작업인원 : 백엔드 2(본인 포함), 프론트 2, AI 1

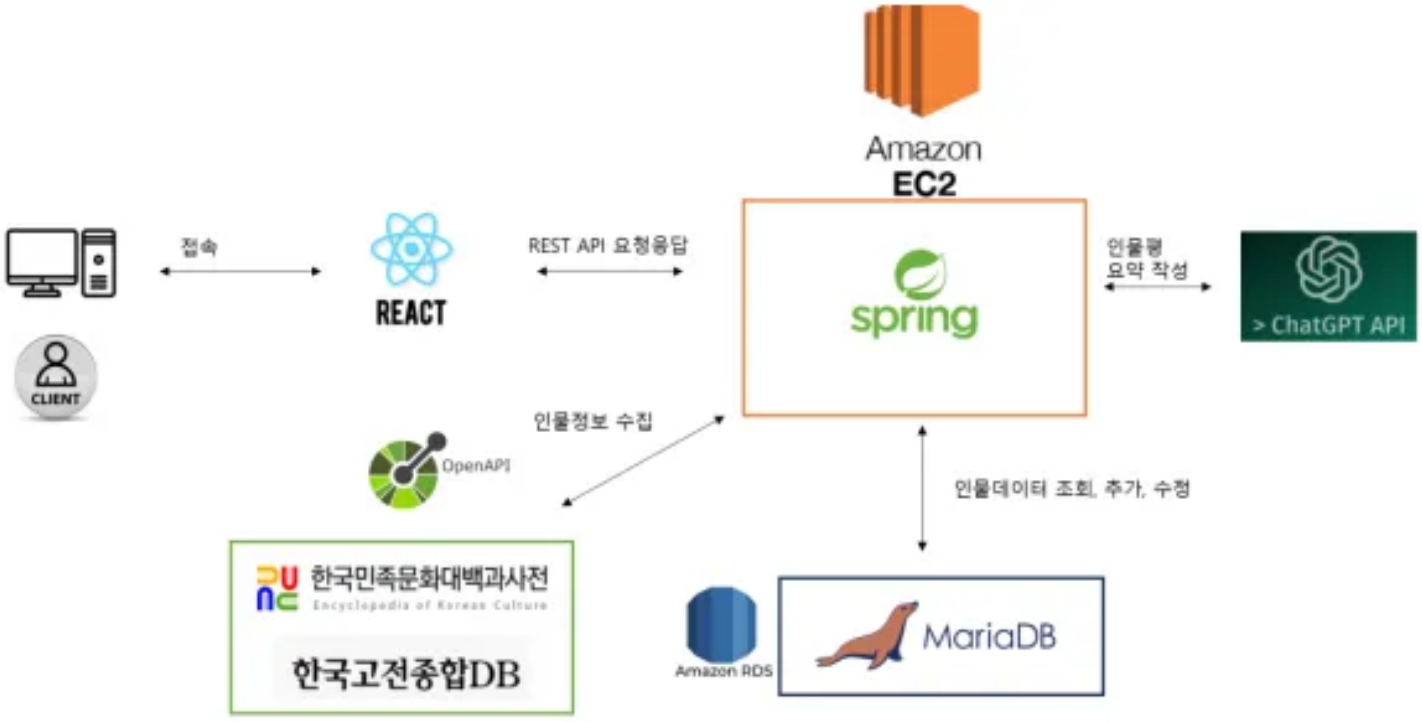
맡은 역할 : 회원가입 및 관리 서버 개발, QA로서 Jira Confluence를 사용하여 개발 산출물과 Scrum 프로세스에 사용되는 양식을 관리

주요 기술 키워드

1. Java
2. Spring, Spring JPA
3. gRPC
4. Git, Github
5. Jira, Confluence

우리조상알기

내가 담당한 역할 (설계, 기능)

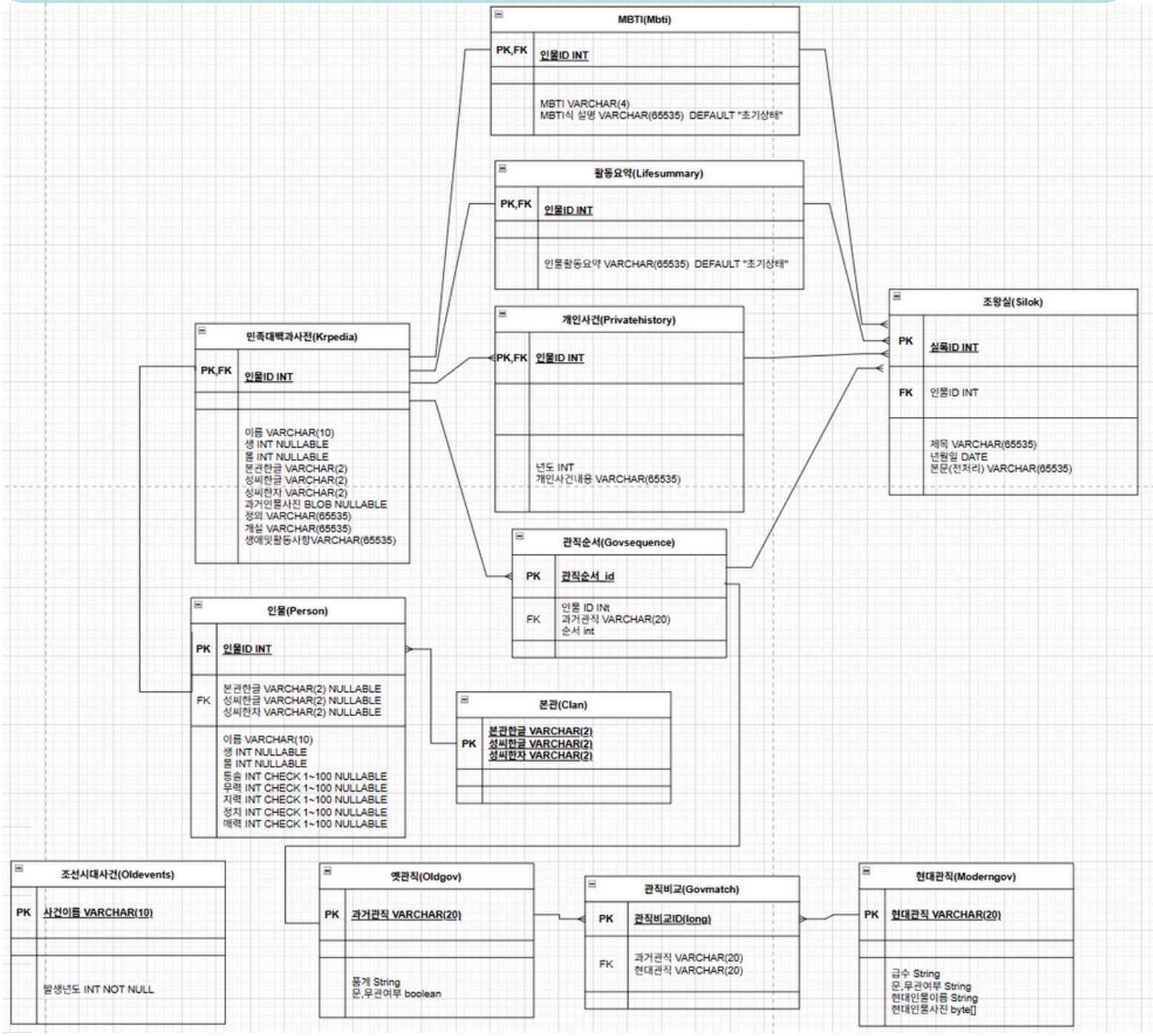


이 프로젝트는 **Java Spring MVC** 기반의 **모놀리식 아키텍처**로, **계층형 구조**를 가진 단일 백엔드 서버가 React 클라이언트로부터의 요청을 수신하여, OpenAPI 사양의 외부 API 서버로부터 동기 방식으로 데이터를 조회하고 MariaDB에 저장한 뒤, Azure OpenAI API와의 동기 통신을 통해 가공한 데이터를 클라이언트에 반환합니다.

또한, 챗봇 기능은 WebSocket 기반의 STOMP 프로토콜을 활용하여 Publish-Subscribe 방식의 비동기 메시지 송수신 구조로 구현하였습니다.

우리조상알기

내가 담당한 역할 (설계, 기능)



프로젝트에서 사용된 ERD스키마입니다. 외부 API서버에서 민족대백과사전, 조왕실, 인물, 본관 테이블을 만들고 Azure OpenAI API를 통해 MBTI, 활동요약, 개인사건, 관직순서 테이블의 데이터를 구성합니다. 옛관직과 현대관직, 조선시대사건 데이터는 사전에 다운받아 구성합니다.

우리조상알기

```java
@Entity
@Data
public class Oldgov {
    @Id
    @Column(name="oldname")
    private String name;

    @Column(name="oldrank")
    private String rank;
    private boolean iswarrior;

    @OneToMany(mappedBy = "oldgov")
    private List<Govmatch> govmatches=new ArrayList<>();
```

```java
@Entity
@Data
public class Moderngov {
    @Id
    @Column(name="modernname")
    private String name;

    @Column(name="modernrank")
    private String rank;

    private boolean iswarrior;

    @Column(name="modernpersonname")
    private String personname;


    @Column(name="modernpersonpicture")
    private String personpicture;


    @OneToMany(mappedBy = "moderngov")
    private List<Govmatch> govmatches=new ArrayList<>();
}
```

```java
@Entity
@Data
public class Govmatch {


    @Id
    @GeneratedValue
    @Column(name="Govmatch_id")
    private Long id;


    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="oldname")
    private Oldgov oldgov;

    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="modernname")
    private Moderngov moderngov;


}
```

앞서 전처리된 관직 데이터의 경우, 위와 같이 사용됩니다. 조선시대의 관직정보를 담은 Oldgov, 현대 공무원의 계급을 나타내는 moderngov는 다대일, 일대다의 관계로 Govmatch에 의해 이어집니다.

우리조상알기

내가 담당한 역할 (트러블 슈팅)

DB연동과 관련하여, H2를 사용하다가 AWS RDS-MariaDB로 전환하면서, 분명히 인코딩 방식을 한글-한자에 맞게 수정했는데 적용되지 않았습니다. AWS홈페이지에서도 매개변수를 수정하고, Workbench를 통해 DB에 직접 접속하여 수정했는데도 반영되지 않았습니다.

```
@Entity
@Getter @Setter
public class Lifesummary {
    @Id
    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="person_id")
    private Krpedia krpedia;

    @Lob
    @Column(name="lifesummarycontents",columnDefinition = "TEXT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci")
    private String contents;

    @OneToMany(mappedBy = "lifesummary")
    private List<Silok> siloks=new ArrayList<>();
```

결국엔 위 코드의 contents와 같이 columnDefinition=”TEXT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci”라고 직접적으로 추가를 해서 해결했습니다.

우리조상알기

또, Spring JPA에서 다대다 관계를 피하는게 좋다고 알고있었습니다. 그래서 인물과 그 본관(가문)에 대한 관계를 동명이인을 고려하여 다대다 관계로 상정했는데, 이 경우 그 관계의 복합키를 따로 클래스로 정의해서 다대다 관계를 다대일, 일대다 관계로 만들게 되었습니다.

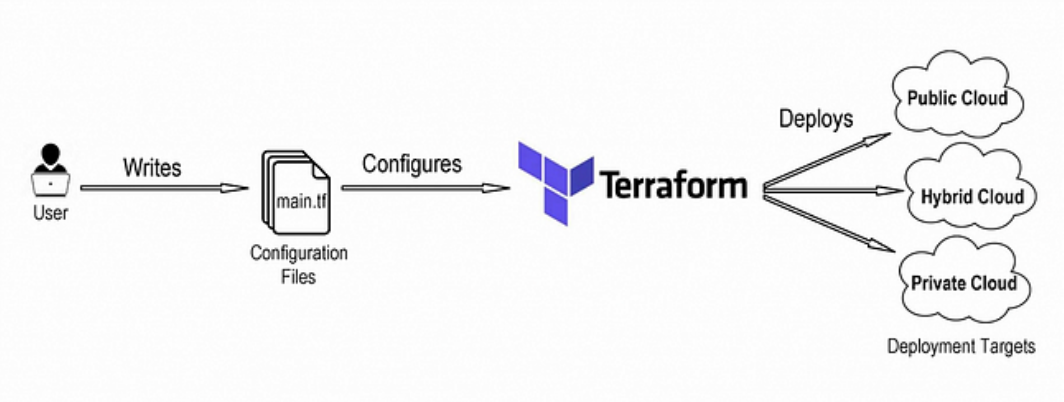
```java
@Entity
@Getter @Setter
public class Person {

    @Id @GeneratedValue
    @Column(name="person_id")
    private Long id;

    @ManyToOne(fetch=FetchType.EAGER)
    @JoinColumns(value = {@JoinColumn(name = "clanHangul"),
            @JoinColumn(name = "surnameHangul"),
            @JoinColumn(name = "surnameHanja")})
    private Clan clan; //본관과 양방향 다대일
```

```java
@Entity
@Getter @Setter
public class Clan {

    @Transient
    @JsonIgnore
    private static final char[] CHO =
            {'ㄱ', 'ㄲ', 'ㄴ', 'ㄷ', 'ㄸ', 'ㄹ', 'ㅁ', 'ㅂ', 'ㅃ'
                    'ㅆ', 'ㅇ', 'ㅈ', 'ㅉ', 'ㅊ', 'ㅋ', 'ㅌ', 'ㅍ'

    @EmbeddedId
   // @Column(length=2)
    private ClanId clanid;

    @JsonIgnore
    @OneToMany(mappedBy = "clan",fetch=FetchType.EAGER)
    private List<Person> persons=new ArrayList<>();
```

```java
@Data
@Embeddable
public class ClanId implements Serializable {

    @Column
    private String clanHangul;
    @Column
    private String surnameHanja;
    @Column
    private String surnameHangul;
```

이지백엔드

내가 담당한 역할 (설계, 기능)



한 사용자가 하나의 Key Pair로 여러 인스턴스를 사용하는 경우를 가정하였고 따라서 작성해야 할 Terraform 스크립트는 다음과 같이 나뉩니다.

1. 한 사용자에 대한 공통 스크립트, 예를들어 여러 인스턴스에 접속하기위한 KeyPair 스크립트
2. 한 사용자에 대한 개별 스크립트, 예를들어 한 사용자가 사용하는 여러 인스턴스에 대한 인스턴스 스크립트

1과 같은경우 Terraform으로 별도의 .tf형식의 파일로 작성했으나, 2와 같은경우 유연한 설정을 위하여 TypeScript코드 내부에서 Terraform 스크립트를 작성하였습니다.

private_key.tf ×

terras > private_key.tf

```
resource "tls_private_key" "cicd_make_key" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

resource "aws_key_pair" "cicd_make_keypair" {
  key_name   = "cicd_key"
  public_key = tls_private_key.cicd_make_key.public_key_openssh
}

resource "local_file" "cicd_downloads_key" {
  filename = "cicd_key.pem"
  content  = tls_private_key.cicd_make_key.private_key_pem
}
```

이지백엔드

deployService.ts 9+

src > services > deployService.ts > ...

```
import { exec,spawn } from 'child_process';
import * as fs from 'fs';
import * as path from 'path';


class DeployService {

    private ami = 'ami-01ed8ade75d4eee2f';
    private instanceType = 't2.micro';
    private privateKeyPath = 'terras/cicd_key.pem';



    createTerraformConfig(instanceName: string, ami: string, instanceType: string): Promise<void>
        return new Promise((resolve, reject) => {
            const config = `

resource "aws_iam_role" "${instanceName}_role" {
  name = "${instanceName}_role"

  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
}

resource "aws_iam_role_policy_attachment" "${instanceName}_role_attach" {
  role       = aws_iam_role.${instanceName}_role.name
  policy_arn = "arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy"
}
```

한편, EC2인스턴스를 실행시키고 코드를 배포하기 전에 몇몇 사전작업이 필요하므로 ssh를 통해 접속하여 각종 명령어를 실행했습니다. 아래는 서버 코드가 담긴 .zip파일을 업로드한 경우, scp로 전송하고 unzip하는 코드입니다.

```
executeUnzipAndListFiles(instanceIp: string, zipFilePath: string, targetDirectory: string, privateKeyPath: string): Promise<string> {
    return new Promise((resolve, reject) => {
        const sshCommand = `ssh -i ${privateKeyPath} -o StrictHostKeyChecking=no ubuntu@${instanceIp} "unzip -o ${zipFilePath} -d ${targetDirectory} && ls ${targetDirectory}"`;
        exec(sshCommand, (error, stdout, stderr) => {
            if (error) {
                reject(error);
                return;
            }
            if (stderr && !stderr.includes("Permanently added")) {
                console.warn(`stderr: ${stderr}`);
            }
            resolve(stdout);
        });
    });
}
```

이지백엔드

인스턴스의 배포 이후 인스턴스 상태확인과 간단한 버전관리 기능이 제공됩니다. 이때, Terraform의 기능으로 서버를 닫거나 재시작하는 것보다 TypeScript 코드를 이용하여 ssh로 접속하는 것이 더 편리하다고 생각하여 이 방법으로 서버관리를 진행했습니다. 아래는 8080포트에 열린 서버를 닫는 코드입니다.

```
executeClosePortCommand(instanceIp: string,privateKeyPath: string): Promise<string> {
    return new Promise((resolve, reject) => {
        const sshCommand = `ssh -i ${privateKeyPath} -o StrictHostKeyChecking=no ubuntu@${instanceIp} "sudo lsof -t -i :8080 | xargs sudo kill -9"`;
        exec(sshCommand, (error, stdout, stderr) => {
            if (error) {
                reject(error);
                return;
            }
            if (stderr) {
                reject(new Error(stderr));
                return;
            }
            resolve(stdout);
        });
    });
}
```

내가 담당한 역할 (트러블 슈팅)

인스턴스를 생성 후 SCP, SSH접속을 하기위해 privatekey파일이 필요합니다. 그러나 배포된 Ubuntu(=Linux)환경에선 키 파일의 권한이 너무 넓게 설정되어있으면 보안문제가 발생하여 연결을 차단하므로 적절히 제한해야 합니다.
linux에서는 chmod를 사용하여 권한설정이 쉽지만 윈도우는 다소 복잡했습니다. “icacls” 명령어를 사용하는 일련의 명령어들인데, 다음과 같습니다.

icacls "..../key.pem" /reset
icacls "..../key.pem" /grant:r "%USERNAME%:R"
icacls "..../key.pem" /inheritance:r

Yesable

내가 담당한 역할 (설계, 기능)

CoreUser — CopyOfUser

userseq	Key	NOT NULL UNIQUE	int
id	Field9	Domain	varchar(30)
password	Field2	Domain	varchar(50)
role	Field3	개인회원/회사회원	Enum
email	Field4	개인이메일/회사전체이메일	varchar(50)
phone_number	Field	개인 번호 / 회사일 경우 회사 번호	varchar(15)
name	Field5	개인이름/회사이름	varchar(15)
gender	Field6	Domain	Enum('male','femal','etc')
date	Field7	생일/나이관련	DATE
authorities	Field8	auth관련한 필드/DB에는 String으로 저장	varchar(30)

PrivateUser — CopyOfUserOnboarding

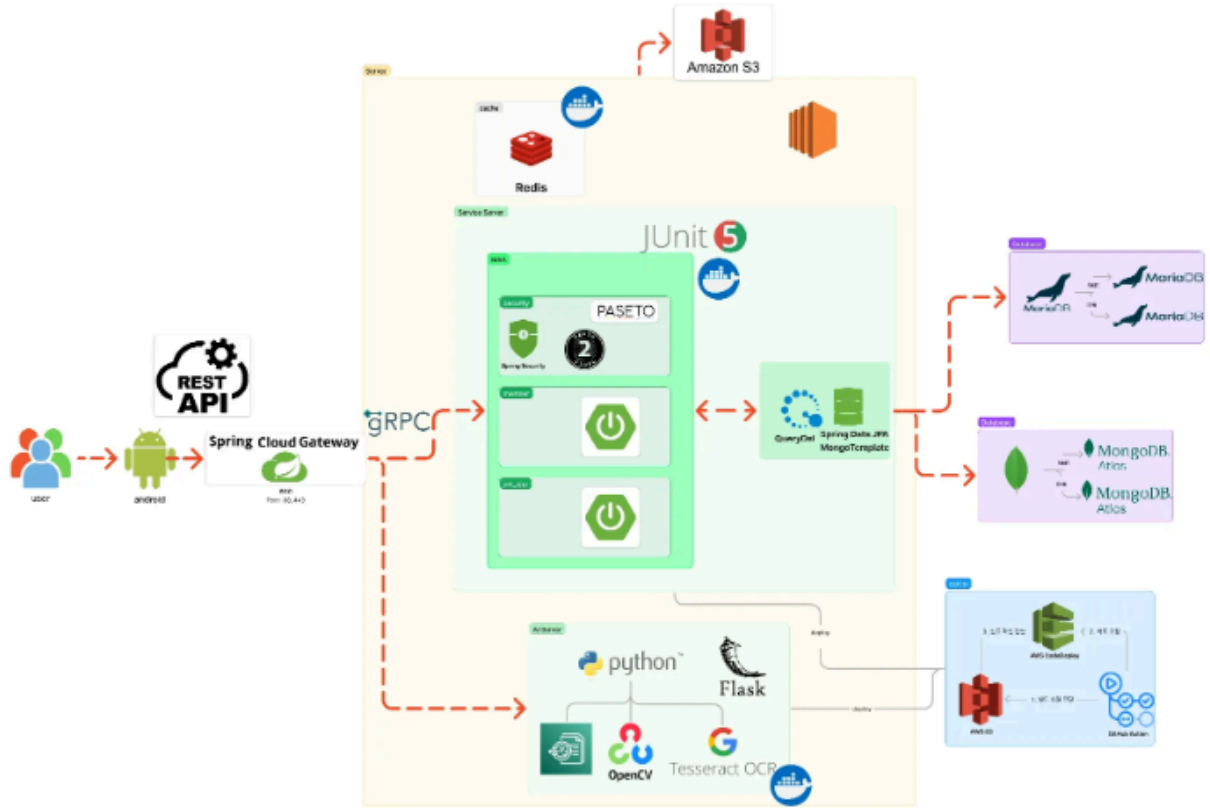
userseq	Key	NOT NULL UNIQUE	int
username	Field12	Domain	varchar(15)
location	Field	상세X, 적절한 지역단위	varchar(100)
interest_fields	Field2	관심분야 IT, 서비스, 제조업..	ENUM
skills	Field5	Domain	varchar(300)
work_type	Field3	정규직 파트타임	ENUM
education	Field6	초중고졸	ENUM
personality	Field8	Domain	varchar(100)
hobbies	Field9	Domain	varchar(100)
support_needs	Field10	지원요구사항 / 입구 경사로..	varchar(200)
disability_type	Field11	Domain	ENUM

CompanyUser — CopyOfUntitled

userseq	Key	NOT NULL UNIQUE	int
Business_number	Field6	사업자등록번호(XXX-XX-XXXXX)	varchar(10)
ceo_name	Field7	Domain	varchar(15)
business_area	Field9	Domain	varchar(100)
comp_class	Field10	Domain	ENUM
hr_name	Field11	Domain	varchar(15)
hr_phone	Field12	Domain	varchar(15)
hr_email	Field13	Domain	varchar(50)
CompanyLocation	Field	도로명 주소 도입에서 문자열 저장으로 변경	varchar(50)

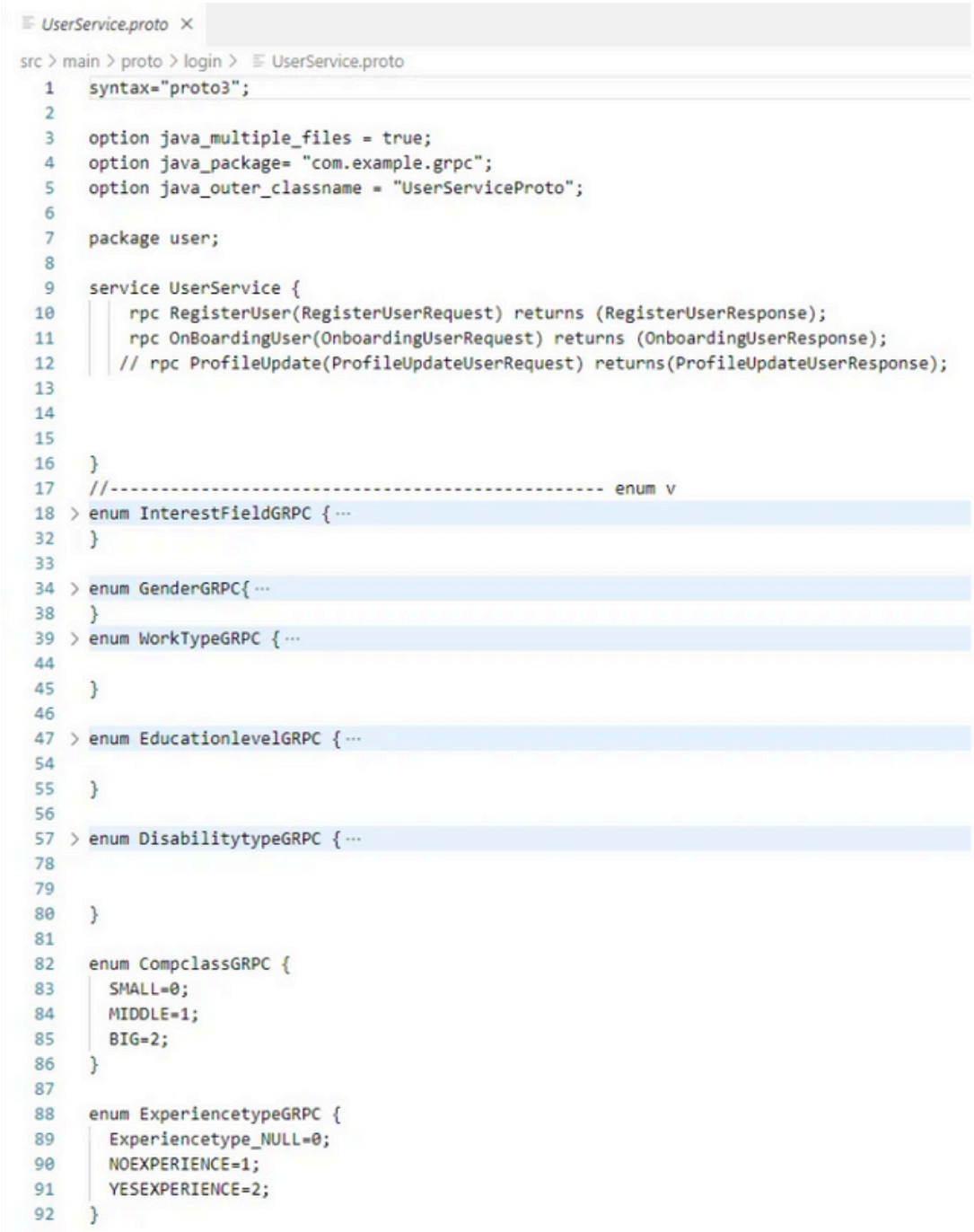
Experience — Untitled

userseq	Key	NOT NULL UNIQUE	int
type	Field	신입, 경력	ENUM
company_name	Field2	입했던 회사 이름	varchar(15)
work_start_date	Field3	경력 시작 날짜	Date
work_end_date	Field4	경력 끝난 날짜	Date
department	Field5	직무 종류	varchar(10)
describe	Field6	주요 업무 내용	varchar(100)



MSA 구조를 채택하고 gRPC 기반으로 마이크로서비스간 통신을 하기로 했습니다. 프론트엔드와 맞닿는 부분은 RESTful API <-> gRPC를 변환하는 게이트웨이를 두기로 했습니다.

Yesable



User관련 서비스를 담당하는 gRPC통신을 위한 .proto파일입니다.

Yesable

```java
@Mapper
public interface MemberMapper {
    //toDTO: gRPC의 각 엔티티를 JPA 스타일의 엔티티로 전환하기 위해 일단 DTO로 변경
    //toEntity : DTO형식의 엔티티를 JPA스타일의 엔티티로 전환
    MemberMapper INSTANCE = Mappers.getMapper(clazz:MemberMapper.class);

    @Mapping(target = "authorities", source = "authoritiesList")
    CoreUserDTO toDTO(CoreUserGRPC entity);



    CoreUser toEntity(CoreUserDTO dto);



    @Mapping(source = "interestFieldList", target = "interestField")
    @Mapping(source = "workTypeList", target = "workType")
    @Mapping(source = "skillsList", target = "skills")
    @Mapping(source = "experiencesList",target="experiences") //이하 super의 coreuser관련
    @Mapping(source="coreUser.id",target="id")
    @Mapping(source="coreUser.password",target="password")
    @Mapping(source="coreUser.email",target="email")
    @Mapping(source="coreUser.phoneNumber",target="phoneNumber")
    @Mapping(source="coreUser.name",target="name")
    @Mapping(source="coreUser.gender",target="gender")
    @Mapping(source="coreUser.dateOfBirth",target="dateOfBirth")
    @Mapping(source="coreUser.authoritiesList",target="authorities")
    PrivateUserDTO grpcToDto(PrivateUserGRPC grpc);

    PrivateUser dtoToEntity(PrivateUserDTO dto);


    @Mapping(source="coreUser.id",target="id")
    @Mapping(source="coreUser.password",target="password")
    @Mapping(source="coreUser.email",target="email")
    @Mapping(source="coreUser.phoneNumber",target="phoneNumber")
    @Mapping(source="coreUser.name",target="name")
    @Mapping(source="coreUser.gender",target="gender")
    @Mapping(source="coreUser.dateOfBirth",target="dateOfBirth")
    @Mapping(source="coreUser.authoritiesList",target="authorities")
    @Mapping(source="hrEmail",target="hr_email")
    @Mapping(source="hrPhone",target="hr_phone")
    @Mapping(source="hrName",target="hr_name")
    CompanyUserDTO grpcToDto(CompanyUserGRPC grpc);

    CompanyUser dtoToEntity(CompanyUserDTO dto);

    //----------------------------------------이하 추가 매핑 메소드------------------------- f

    default Collection<GrantedAuthority> map(ProtocolStringList ls) {

        return ls.stream()
                .map(SimpleGrantedAuthority::new)
                .collect(Collectors.toCollection(ArrayList::new));
```

gRPC통신 객체 <-> DTO객체 <-> JPA저장을 위한 객체 변환을 담당하는 Mapstruct기반의 Mapper입니다. gRPC를 통해 자동으로 생성된 코드는 네이밍이나 구조가 특이해서 자동으로 생성된 코드의 이름을 보고 직접 Mapping을 진행했습니다.

Yesable

또, Scrum으로 진행된 프로젝트의 QA로서, Jira, Confluence를 활용하여 프로젝트 진행의 산출물과 일정관리 양식을 작성했습니다.

내가 담당한 역할 (트러블 슈팅, 프로젝트 중단)

그러나 프로젝트가 진행되다가 프론트쪽에서 구현이 지연되고, AI쪽에서도 크게 진행되지 않는 등 문제가 심해지면서 아쉽게도 일정에 맞추지 못해 프로젝트가 중단되게 되었습니다.