포팅메뉴얼

1. 기술 스택 및 개발 환경

사용 도구

- 이슈 관리: JIRA
- 형상 관리: GitLab
- 커뮤니케이션: Mattermost, Notion, Discord
- 디자인: Figma
- 영상: - Movavi, typomotion
- 데이터베이스: ERD Cloud
- CI/CD: EC2, Docker, Jenkins

개발 도구

- Android Studio: 2024.2.2.13
- Intellij: 2024.3.1.1 (Ultimate Edition)

개발 환경

BlockChain

- SSAFY NetWork
- REMIX IDE

Mobile

- Android Studio: 2024.3.1
- Flutter: 3.29.1-stable
- Dart SDK: ^3.7.0
- Android SDK: Android 12 (API level 31)

Back

- JDK:
- Spring Boot:
- Gradle:

Infra & Server

- AWS S3
- AWS EC2: t2.xlarge

- Nginx:

DataBase

- MySQL:
- Redis:

CI/CD

- Jenkins:
- Docker:

Authentication

- jwt:
- oauth

Documentation

- swagger:

- `.gitignore`

```
HELP.md
.gradle
build/
!gradle/wrapper/gradle-wrapper.jar
!**/src/main/**/build/
!**/src/test/**/build/

### STS ###
.apt_generated
.classpath
.factorypath
.project
.settings
.springBeans
.sts4-cache
bin/
!**/src/main/**/bin/
!**/src/test/**/bin/

### IntelliJ IDEA ###
.idea
*.iws
*.iml
*.ipr
out/
!**/src/main/**/out/
!**/src/test/**/out/

### NetBeans ###
/nbproject/private/
/nbbuild/
/dist/
/nbdist/
/.nb-gradle/

### VS Code ###
.vscode/

/src/main/resources/**.yml
/src/main/resources/**.properties

### macOS template
# General
.DS_Store
.AppleDouble
.LSOverride

logs/
```

폴더 구조

환경 변수

Front

- `.env`

```
VITE_SERVER_URL = "https://kukkkukk.duckdns.org"
VITE_SOCKET_WS_URL="wss://kukkkukk.duckdns.org"
```

Mobile

- `.env`

```
# API 설정
API_BASE_URL=https://kukkkukk.duckdns.org

# Blockchain 설정
BLOCKCHAIN_RPC_URL=https://rpc.ssafy-blockchain.com
BLOCKCHAIN_WS_URL=wss://ws.ssafy-blockchain.com
BLOCKCHAIN_CHAIN_ID=31221
PET_REGISTRY_CONTRACT_ADDRESS=0x56e3e3B9d31B070c96e264b645D0763b2DC49e65

# Kakao 설정
KAKAO_APP_KEY=65cfeb1036d02ed518a5b5f1408a0c46
KAKAO_JS_KEY=8d7447c56c08fdc80ddeb952247f4caa
```

- `app/local.properties(android)`

```
kakao.app.key=65cfeb1036d02ed518a5b5f1408a0c46
```

Back

- `application.properties`

```
// application.properties

spring.application.name=KKUKKKUK
spring.datasource.url=jdbc:mysql://localhost:3306/kkukkkuk
spring.datasource.username=root
spring.datasource.password=ssafy
spring.datasource.hikari.idle-timeout=10000
spring.datasource.hikari.maximum-pool-size=20
spring.datasource.hikari.max-lifetime=240000
spring.datasource.hikari.minimum-idle=10
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql=true
spring.servlet.multipart.max-request-size=100MB
spring.servlet.multipart.max-file-size=100MB

jwt.secret=ssafy-gwangju-class2-specialized-project-c206
jwt.access-token-validity=3600000
jwt.refresh-token-validity=604800000

spring.data.redis.host=localhost
spring.data.redis.port=6379

cloud.aws.credentials.access-key=AKIA2S2Y4R4XW6GLOIWM
cloud.aws.credentials.secret-key=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
cloud.aws.region.static=ap-northeast-2
cloud.aws.stack.auto=false
cloud.aws.s3.bucket=kkukkkuk

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=kkuk.ssafy
spring.mail.password=hemfoqmomdszjckv
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required=true
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.writetimeout=1800000
spring.mail.auth-code-expiration-millis=1800000
```

```
openai.api.key=sk-proj-bmk88-Wpd1jztCK7VLFfoZcVqG-fDfC4Hln3y5hVJe5QEYxL97TZ7R
sWpPxmdLKMfHNm9E1V3wT3BlbkFJHAEDqhUMoXsk4qS3pjiB35umkw7HrwdepDk4JIBSHMp22d-GANyeerCUaMgGwsP4IUHmirkuMA
```

블록체인

```
let didRegistryAddress = '0x56e3e3B9d31B070c96e264b645D0763b2DC49e65';

const NETWORK_CONFIG = {
   rpcUrl: 'https://rpc.ssafy-blockchain.com',
   wsUrl: 'wss://ws.ssafy-blockchain.com',
   chainId: '31221', //10진수임.
   chainName: 'SSAFY',
   nativeCurrency: {
     name: 'ETH',
     symbol: 'ETH',
     decimals: 18
   }
 };
```

2. CI/CD 구축

Jenkins 세팅

```
pipeline {
   agent any

   environment {
      GIT_REPO        = 'https://lab.ssafy.com/s12-blockchain-nft-sub1/S12P21C206.git'
      FE_TARGET_BRANCH = 'fe'
      FE_IMAGE_NAME    = 'cod0216/frontend'
      BE_TARGET_BRANCH = 'be'
      BE_IMAGE_NAME    = 'cod0216/backend'
   }

   triggers {
      gitlab(
         triggerOnPush: true,
         triggerOnMergeRequest: true,
         branchFilterType: "NameBasedFilter",
         targetBranchRegex: '^(fe|be)$'
      )
   }

   stages {
      stage('Debug Environment') {
         steps {
            echo "gitlabSourceBranch: ${env.gitlabSourceBranch}"
            echo "gitlabTargetBranch: ${env.gitlabTargetBranch}"
         }
      }

      // =================== FE Pipeline ===================
      stage('Clone FE Repository') {
         when {
            expression { env.gitlabSourceBranch == FE_TARGET_BRANCH }
         }
         steps {
            echo "Cloning FE repository..."
            git branch: FE_TARGET_BRANCH, url: GIT_REPO, credentialsId: 'gitlab'
         }
      }

      stage('Load FE Environment File') {
         when {
            expression { env.gitlabSourceBranch == FE_TARGET_BRANCH }
         }
         steps {
            echo "Loading .env from Jenkins secret file credentials..."
            withCredentials([file(credentialsId: 'FE_env', variable: 'FE_ENV')]) {
               sh 'mkdir -p frontend'
               sh 'chmod -R 777 frontend'
               sh 'cp $FE_ENV frontend/.env'
            }
```

```
        }
    }

    stage('Install & Build FE') {
        when {
            expression { env.gitlabSourceBranch == FE_TARGET_BRANCH }
        }
        steps {
            echo "Installing dependencies and building FE project..."
            dir('frontend') {
                sh 'npm install'
                sh 'CI=false npm run build'
            }
        }
    }

    stage('Build Docker Image for FE') {
        when {
            expression { env.gitlabSourceBranch == FE_TARGET_BRANCH }
        }
        steps {
            echo "Building FE Docker image..."
            dir('frontend') {
                sh """
                    docker build -f /home/ubuntu/FE.Dockerfile -t ${FE_IMAGE_NAME}:${env.BUILD_ID} .
                    docker tag ${FE_IMAGE_NAME}:${env.BUILD_ID} ${FE_IMAGE_NAME}:latest
                """
            }
        }
    }

    stage('Docker Login & Push FE') {
        when {
            expression { env.gitlabSourceBranch == FE_TARGET_BRANCH }
        }
        steps {
            echo "Logging in to Docker registry for FE..."
            withCredentials([usernamePassword(credentialsId: 'docker-hub-cred',
            usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
                sh "echo ${DOCKER_PASS} | docker login -u ${DOCKER_USER} --password-stdin"
            }
            echo "Pushing FE Docker image..."
            sh "docker push ${FE_IMAGE_NAME}:${env.BUILD_ID}"
            sh "docker push ${FE_IMAGE_NAME}:latest"
        }
    }

    stage('Deploy FE Container') {
        when {
            expression { env.gitlabSourceBranch == FE_TARGET_BRANCH }
        }
        steps {
            echo "Deploying FE container..."
            sh """
               docker rm -f FrontEnd || true
               docker run -d --name FrontEnd -p 8081:80 ${FE_IMAGE_NAME}:latest
               """
        }
    }

    // ==================== BE Pipeline ====================
    stage('Clone BE Repository') {
        when {
            expression { env.gitlabSourceBranch == BE_TARGET_BRANCH }
        }
        steps {
            echo "Cloning BE repository..."
            git branch: BE_TARGET_BRANCH, url: GIT_REPO, credentialsId: 'gitlab'
        }
    }

    stage('Load Application Properties') {
        when {
            expression { env.gitlabSourceBranch == BE_TARGET_BRANCH }
        }
        steps {
            echo "Loading application.properties from Jenkins secret file credentials..."
            withCredentials([file(credentialsId: 'application_properties', variable: 'APP_PROPS')]) {
                sh 'mkdir -p backend/KKUKKKUK/src/main/resources'
                sh 'chmod -R 777 backend/KKUKKKUK/src/main/resources'
                sh 'cp $APP_PROPS backend/KKUKKKUK/src/main/resources/application.properties'
            }
        }
    }

    stage('Build Spring Boot Application (Gradle)') {
        when {
            expression { env.gitlabSourceBranch == BE_TARGET_BRANCH }
        }
        steps {
            echo "Building Spring Boot application with Gradle and JDK 17 (skipping tests)..."
            dir('backend/KKUKKKUK') {
                sh 'chmod +x gradlew'
                sh './gradlew clean build -x test'
            }
        }
    }

    stage('Build Docker Image for BE') {
        when {
            expression { env.gitlabSourceBranch == BE_TARGET_BRANCH }
        }
        steps {
            echo "Building BE Docker image..."
            dir('backend') {
                sh """
                    docker build -f /home/ubuntu/BE.Dockerfile -t ${BE_IMAGE_NAME}:${env.BUILD_ID} .
                    docker tag ${BE_IMAGE_NAME}:${env.BUILD_ID} ${BE_IMAGE_NAME}:latest
                """
            }
        }
    }

    stage('Docker Login & Push BE') {
        when {
            expression { env.gitlabSourceBranch == BE_TARGET_BRANCH }
        }
        steps {
            echo "Logging in to Docker registry for BE..."
            withCredentials([usernamePassword(credentialsId: 'docker-hub-cred',
            usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
                sh "echo ${DOCKER_PASS} | docker login -u ${DOCKER_USER} --password-stdin"
            }
            echo "Pushing BE Docker image..."
            sh "docker push ${BE_IMAGE_NAME}:${env.BUILD_ID}"
            sh "docker push ${BE_IMAGE_NAME}:latest"
        }
    }

    stage('Deploy BE Container') {
        when {
            expression { env.gitlabSourceBranch == BE_TARGET_BRANCH }
        }
        steps {
            echo "Deploying BE container..."
            sh """
                docker rm -f BackEnd1 || true
                docker rm -f BackEnd2 || true

                docker run -d --name BackEnd1 --network my-network -p 8080:8080 ${BE_IMAGE_NAME}:latest
                docker run -d --name BackEnd2 --network my-network -p 8082:8080 ${BE_IMAGE_NAME}:latest
               """
        }
    }

    stage('Cleanup FE Images') {
        when {
            expression { env.gitlabSourceBranch == FE_TARGET_BRANCH }
        }
        steps {
            echo "Cleaning up older FE images..."
            sh """
                FE_IMAGES=\$(docker images ${FE_IMAGE_NAME} --format "{{.Repository}}:{{.Tag}}"
                | grep -v "${env.BUILD_ID}" | grep -v "latest" || true)
            for img in \$FE_IMAGES; do
              docker rmi \$img || true
            done
          """
        }
      }

      stage('Cleanup BE Images') {
        when {
          expression { env.gitlabSourceBranch == BE_TARGET_BRANCH }
        }
        steps {
          echo "Cleaning up older BE images..."
          sh """
            BE_IMAGES=\$(docker images ${BE_IMAGE_NAME} --format "{{.Repository}}:{{.Tag}}"
            | grep -v "${env.BUILD_ID}" | grep -v "latest" || true)
            for img in \$BE_IMAGES; do
              docker rmi \$img || true
            done
          """
        }
      }
    }

    post {
      success {
        echo "✅ Build and deployment successful!"
      }
      failure {
        echo "❌ Build or deployment failed!"
      }
    }
}
```

Docker 파일

```
// BE.Dockerfile
FROM openjdk:17-jdk-alpine

COPY KKUKKKUK/build/libs/*.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]

// FE.Dockerfile
FROM nginx:alpine
RUN rm -rf /usr/share/nginx/html/*
COPY dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

NginX

```
upstream backend {
      server localhost:8080;
      server localhost:8082;
}

upstream websocket_backend {
    server localhost:8080;
}

server {
    client_max_body_size 20M;
    server_name kukkkukk.duckdns.org;

    location /{
        proxy_pass http://127.0.0.1:8081;
        proxy_intercept_errors on;
        error_page 404 /index.html;
    }
    location ~ ^/(swagger|webjars|configuration|swagger-resources|v2|v3|csrf){
        proxy_pass http://backend;
    }

    location /api {
        proxy_pass http://backend;
    }

    location /app {
        proxy_pass http://websocket_backend;
    }

    location /kkukkkuk {
        proxy_pass http://websocket_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;

        proxy_read_timeout 3600;
        proxy_send_timeout 3600;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/kukkkukk.duckdns.org/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/kukkkukk.duckdns.org/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}

server {
    if ($host = kukkkukk.duckdns.org) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    listen 80;
    server_name kukkkukk.duckdns.org;
    return 404; # managed by Certbot


}
```

3. 빌드 및 배포

빌드

Mobile

```
cd ~/{project-root}
dart run build_runner build --delete-conflicting-outputs // freezed, json_serializable 등의 코드 생성 라이브러리를
flutter build apk --release --target-platform=android-arm64
```

BackEnd

```
cd ~/{project-root}
./gradlew build
```

FrontEnd

```
cd ~/{project-root}
npm install
npm run build
```

NginX

```
sudo nginx -t
sudo nginx -s reload
```

배포

CI/CD

```
GitLab에서 `fe` 또는 `be` 브랜치에 Push 또는 Merge Request 발생 시, Jenkins 파이프라인이 자동으로 실행되어 다음 과정을 수행합니다.
```

FrontEnd

```
1. FE 브랜치가 감지되면 FE 코드를 클론합니다.
2. Jenkins Credential로부터 `.env` 환경 파일을 로드합니다.
3. `npm install`, `npm run build`를 통해 프로젝트를 빌드합니다.
4. `/home/ubuntu/FE.Dockerfile`을 사용하여 Docker 이미지를 빌드하고 태깅합니다.
5. Docker Hub에 로그인 후, `latest` 및 Build ID 태그로 이미지를 푸시합니다.
6. 기존 컨테이너 제거 후, 새 컨테이너로 배포합니다. (포트: 8081)
```

BackEnd

```
1. BE 브랜치가 감지되면 BE 코드를 클론합니다.
2. Jenkins Credential로부터 `application.properties` 파일을 로드합니다.
3. `./gradlew clean build -x test` 명령어로 Spring Boot 애플리케이션을 빌드합니다.
4. `/home/ubuntu/BE.Dockerfile`을 사용하여 Docker 이미지를 빌드하고 태깅합니다.
5. Docker Hub에 로그인 후, `latest` 및 Build ID 태그로 이미지를 푸시합니다.
6. 기존 컨테이너 2개를 제거하고, 동일한 이미지로 8080, 8082 포트에 두 개의 컨테이너를 배포합니다.

두 개의 BE 컨테이너는 Nginx의 Round-Robin 방식으로 로드 밸런싱 됩니다.
```

4. 외부 서비스 및 활용 정보

Mobile

- kakao login API
- kakao Map API
- Google ML Kit

FrontEnd

- MetaMask
- SSAFY BlockChain Network
- Ethereum Coin

BackEnd

- OpenAi API
- S3