TTD 웹 포팅 메뉴얼

1. 환경변수

- Jenkins

```
id:human3452

pw:asdasd12
```

- Frontend dockerfile

```
# Specify base image
FROM node:18.17.0

# Set working directory
WORKDIR /app

# Copy package.json and package-lock.json before other files
# Utilise Docker cache to save re-installing dependencies if unchanged
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy all files
COPY ./ ./

# Build application
RUN npm run build

# Specify the command to run
CMD [ "npm", "start" ]

# Expose the port the app runs on
EXPOSE 3000
```

- package.json

```
{
  "name": "ttd",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.14.1",
    "@testing-library/react": "^13.0.0",
    "@testing-library/user-event": "^13.2.1",
    "@types/jest": "^27.0.1",
    "@types/node": "^16.7.13",
    "@types/react": "^18.0.0",
    "@types/react-dom": "^18.0.0",
    "firebase": "^10.5.2",
    "lodash": "^4.17.21",
    "react": "^18.2.0",
    "react-anchor-link-smooth-scroll": "^1.0.12",
    "react-dom": "^18.2.0",
    "react-player": "^2.13.0",
    "react-router-dom": "^6.18.0",
    "react-scripts": "5.0.1",
    "styled-components": "^6.1.0",
    "styled-reset": "^4.5.1",
    "typescript": "*",
    "web-vitals": "^2.1.0"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "@typescript-eslint/eslint-plugin": "^6.4.0",
    "eslint": "^8.0.1",
    "eslint-config-prettier": "^9.0.0",
    "eslint-config-standard-with-typescript": "^39.1.1",
    "eslint-plugin-import": "^2.29.0",
    "eslint-plugin-n": "^15.0.0 || ^16.0.0 ",
    "eslint-plugin-prettier": "^5.0.1",
    "eslint-plugin-promise": "^6.0.0",
    "eslint-plugin-react": "^7.33.2",
    "prettier": "3.0.3"
  }
}
```

2. 빌드

Front-End

1. npm i
2. npm start

3. 배포

1. Nginx 설치

```
sudo apt install nginx
```

2. Nginx 설정

```
server {
        listen 80;
        server_name k9b302.p.ssafy.io;
        return 301 https://k9b302.p.ssafy.io$request_uri;
}

server {
       listen 443 ssl http2;
       listen [::]:443 ssl http2;
       server_name k9b302.p.ssafy.io;

                # ssl 인증서 적용하기 - ssl 인증키의 위치
        ssl_certificate /etc/letsencrypt/live/k9b302.p.ssafy.io/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/k9b302.p.ssafy.io/privkey.pem;

    location / {
        proxy_pass http://127.0.0.1:3000/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

        location /api/ {
                proxy_pass http://localhost:8080/;
                charset utf-8;

                proxy_http_version 1.1;

                                # 요청과 응답 중 Connection 헤더 설정, upgrade로 설정함으로
                                # 일부 특별한 상황에서 연결을 업그레이드
                proxy_set_header Connection "upgrade";

                                # Upgrade 헤더 값을 설정, 클라이언트 요청의 upgrade 값 유지
                proxy_set_header Upgrade $http_upgrade;

                                # Host 헤더 값을 클라이언트 요청의  Host 헤더 값으로 설정
                                # 프록시에 요청을 할 때 원래 호스트 정보를 유지하기 위해
                proxy_set_header Host $http_host;
        }

}
```

3. ssl 인증

3-1 인증서 발급

```
#let’s Encrypt 설치
sudo apt-get insall letsencrypt

#cerbot 설치
$ apt-get install python3-certbot-nginx

#cerbot 동작
sudo certbot --nginx

#ssl 인증서 발급
$ certbot certonly --nginx -d <도메인 주소>
```

3-2 인증서 주소

```
cd /etc/letsencrypt/live/인증서발급받은 도메인주소

ex) cd /etc/letsencrypt/live/j9b207.p.ssafy.io
```

4. Jenkins Pipeline

```
pipeline {
    agent any

    environment {
        CONTAINER_NAME = "jenkins-frontend-container"
        IMAGE_NAME = "jenkins-frontend-image"
        ENV_FILE = ""
    }
    tools {
        nodejs 'npm'
    }
    stages {
        stage('Git Clone') {
            steps {
                git branch: 'Frontend', credentialsId: 'b302admin', url: 'https://lab.ssafy.com/s09-final/S09P31B302.git'
                sh "docker images"
            }
        }
        stage('Build') {
            steps {
        // Using the credentials binding plugin to bind the secret file to an env variable
                withCredentials([file(credentialsId: 'ENVFILE', variable: 'SECRET_ENV_FILE')]) {
            sh '''
                cd TTD_Frontend/ttd

                # Copy the secret .env file to your project directory
                cp $SECRET_ENV_FILE .env

                # Continue with the build
                npm cache clean --force
                npm install .
                CI=false npm run build
            '''
        }
    }
}


        stage('Docker delete') {
            steps {
                script {
                    try {
                        // 컨테이너가 존재하면 삭제합니다.
                        sh "docker stop ${CONTAINER_NAME}"
                        sh "docker rm -f ${CONTAINER_NAME}"
                    } catch (Exception e) {
                        // 컨테이너가 존재하지 않는 경우 에러가 발생할 수 있으므로, 에러를 무시합니다.
                        echo "Docker container ${CONTAINER_NAME} does not exist. Skipping deletion."
                    }

                    try {
                        // 이미지가 존재하면 삭제합니다.
                        sh "docker image rm ${IMAGE_NAME}"
                    } catch (Exception e) {
                        // 이미지가 존재하지 않는 경우 에러가 발생할 수 있으므로, 에러를 무시합니다.
                        echo "Docker image ${IMAGE_NAME} does not exist. Skipping deletion."
                    }
                }
            }

            post {
                success {
                    sh 'echo "docker delete Success"'
                }
                failure {
                    sh 'echo "docker delete Fail"'
                }
            }
        }
        stage('Dockerizing'){
            steps{
                sh 'echo " Image Bulid Start"'
                sh """
                    cd TTD_Frontend
                    cd ttd
                    ls -la
                    docker build -t ${IMAGE_NAME} .
                """
            }
            post {
                success {
                    sh 'echo "Bulid Docker Image Success"'
                }
                failure {
                    sh 'echo "Bulid Docker Image Fail"'
                }
            }
        }
        stage('Deploy') {
            steps {
                sh "docker run --name ${CONTAINER_NAME} -d -p 3000:3000 ${IMAGE_NAME}"
            }
            post {
                success {
                    echo 'deploy success'
                }
                failure {
                    echo 'deploy failed'
                }
            }
        }
    }
}
```