# ZEEPSEEK 포팅 메뉴얼

## (로컬) Frontend 테스트 명령어

1. FE/src 폴더로 이동합니다.
2. 아래의 명령어를 터미널에 입력해 필요한 모듈을 설치합니다.
    a. node.js 18.x 버전이 필요합니다.

```
# 필요 모듈 설치
npm -i

# 프로젝트 실행
npm run dev
```

---

## (로컬) Backend 테스트 명령

1. BE/zeepseek 폴더로 이동합니다.
2. 아래의 명령어를 터미널에 입력합니다.
    a. Java 17 버전이 필요합니다.

```
./gradlew bootRun
또는
```

```
gradle bootRun
```

3. (파이썬) ML 폴더로 이동합니다.
    a. 파이썬 3.10.x 버전이 필요합니다.
    b. 다음의 명령어를 터미널에 입력해 모듈을 설치합니다.

```
pip install -r requirements.txt
```

1. 설치된 모듈은 다음과 같습니다.

```
fastapi
uvicorn
sqlalchemy
pymysql
pymongo
python-dotenv
numpy<2
scikit-learn
cryptography
pandas
elasticsearch
scikit-surprise
requests
```

---

## (로컬) DataBase 세팅

1. MySQL 8.0.x 버전을 설치합니다.
2. zeepseek 라는 이름의 스키마를 생성합니다.

▼ 아래의 SQL 구문을 실행해 테이블을 생성합니다.

```
USE zeepseek;

-- DROP 작업 시 외래키 검사 비활성화
SET FOREIGN_KEY_CHECKS=0;

DROP TABLE IF EXISTS user_preference;
```

```sql
DROP TABLE IF EXISTS property_score;
DROP TABLE IF EXISTS property;
DROP TABLE IF EXISTS dong;
DROP TABLE IF EXISTS user;
DROP TABLE IF EXISTS transport;
DROP TABLE IF EXISTS restaurant;
DROP TABLE IF EXISTS health;
DROP TABLE IF EXISTS convenience;
DROP TABLE IF EXISTS cafe;
DROP TABLE IF EXISTS chicken;
DROP TABLE IF EXISTS leisure;
DROP TABLE IF EXISTS convinince;

-- DROP 후 외래키 검사 재활성화
SET FOREIGN_KEY_CHECKS=1;

-- 3. (유저) user 테이블 생성
CREATE TABLE user (
   idx INT AUTO_INCREMENT PRIMARY KEY,
   isfirst INT DEFAULT 0,
   isseller INT DEFAULT 0,
   gender INT DEFAULT 0,
   age INT DEFAULT 0,
   refreshtoken TEXT,
   nickname VARCHAR(50),
   provider VARCHAR(20),
   providerid VARCHAR(100),
   UNIQUE KEY unique_social_account (provider, providerid)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 4. (동) dong 테이블 생성 (동 정보와 동 점수를 함께 관리)
CREATE TABLE dong (
   dong_id INT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,          -- 동 이름 (필수)
   gu_name VARCHAR(255) NOT NULL,          -- 구 이름 (필수)
   safe FLOAT,
   leisure FLOAT,
   restaurant FLOAT,
   health FLOAT,
   mart FLOAT,
   convenience FLOAT,
   transport FLOAT,
   updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 5. (매물) property 테이블 생성
CREATE TABLE property (
   property_id INT AUTO_INCREMENT PRIMARY KEY,
   seller_id INT,                    -- 판매자 (필수)
   room_type VARCHAR(20) NOT NULL,          -- 매물유형 (필수)
   contract_type VARCHAR(20) NOT NULL,      -- 거래유형 (필수)
   price VARCHAR(30) NOT NULL,              -- 가격 (필수)
   address VARCHAR(255) NOT NULL,           -- 소재지 (필수)
   description TEXT,                        -- 매물특징
   area VARCHAR(50),                 -- 공급/전용면적
   floor_info VARCHAR(50) ,           -- 해당층/총층
   room_bath_count VARCHAR(50),    -- 방수/욕실수
   maintenance_fee INT DEFAULT 0,           -- 관리비
   move_in_date VARCHAR(50),          -- 입주 가능일
   direction VARCHAR(50),             -- 방향
   image_url TEXT,                          -- 메인사진 URL
   sale_price INT,                          -- 매매가(만원)
   deposit INT,                             -- 보증금(만원)
   monthly_rent INT,                        -- 월세(만원)
   latitude FLOAT,
   longitude FLOAT,
   dong_id INT NOT NULL,                    -- 연결된 동 (필수)
   gu_name VARCHAR(10) NOT NULL,
     dong_name VARCHAR(255) ,
     location POINT,
   created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
   CONSTRAINT fk_property_dong FOREIGN KEY (dong_id) REFERENC
       ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 6. (유저선호도) user_preference 테이블 생성
```

```
CREATE TABLE user_preference (
   user_id INT PRIMARY KEY,
     safe FLOAT DEFAULT 0.0,
     leisure FLOAT DEFAULT 0.0,
     restaurant FLOAT DEFAULT 0.0,
     health FLOAT DEFAULT 0.0,
     convenience FLOAT DEFAULT 0.0,
     transport FLOAT DEFAULT 0.0,
     cafe FLOAT DEFAULT 0.0,
     dong_id INT, NOT null,
     /* 주소 관련 필드 */
     destination VARCHAR(255), /* 전체 주소 - 길이 확장 */
     sido VARCHAR(50),         /* 시/도 */
     sigungu VARCHAR(50),      /* 시/군/구 */
     road_name VARCHAR(100),   /* 도로명 */
     building_info VARCHAR(50), /* 건물 번호 및 정보 */
     detail_address VARCHAR(50), /* 상세 주소 */
     zip_code VARCHAR(10),     /* 우편번호 */

     /* 위치 정보 */
     latitude DECIMAL(10,7),   /* 위도 */
     longitude DECIMAL(11,7),  /* 경도 */
   CONSTRAINT fk_pref_user FOREIGN KEY (user_id) REFERENCES us
       ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 7. transport 테이블 생성 (dong_id 외래키 사용)
CREATE TABLE transport (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   gu_name VARCHAR(255) NOT NULL,
   dong_id INT NOT NULL,
   latitude DECIMAL(10,7) NOT NULL,
   longitude DECIMAL(10,7) NOT NULL,
   type VARCHAR(30) NOT NULL,
   CONSTRAINT fk_transport_dong FOREIGN KEY (dong_id) REFEREN
       ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- 8. restaurant 테이블 생성 (dong_id 외래키 사용)
CREATE TABLE restaurant (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   gu_name VARCHAR(255) NOT NULL,
   dong_id INT NOT NULL,
   latitude DECIMAL(10,7) NOT NULL,
   longitude DECIMAL(10,7) NOT NULL,
   CONSTRAINT fk_restaurant_dong FOREIGN KEY (dong_id) REFEREN
       ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 9. health 테이블 생성 (dong_id 외래키 사용)
CREATE TABLE health (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   gu_name VARCHAR(255) NOT NULL,
   dong_id INT NOT NULL,
   latitude DECIMAL(10,7) NOT NULL,
   longitude DECIMAL(10,7) NOT NULL,
   CONSTRAINT fk_health_dong FOREIGN KEY (dong_id) REFERENCES
       ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 10. convenience 테이블 생성 (dong_id 외래키 사용)
CREATE TABLE convenience (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   gu_name VARCHAR(255) NOT NULL,
   type VARCHAR(30) NOT NULL,
   dong_id INT NOT NULL,
   latitude DECIMAL(10,7) NOT NULL,
   longitude DECIMAL(10,7) NOT NULL,
   CONSTRAINT fk_convenience_dong FOREIGN KEY (dong_id) REFER
       ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- 11. cafe 테이블 생성 (dong_id 외래키 사용)
CREATE TABLE cafe (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   gu_name VARCHAR(255) NOT NULL,
   dong_id INT NOT NULL,
   latitude DECIMAL(10,7) NOT NULL,
   longitude DECIMAL(10,7) NOT NULL,
   CONSTRAINT fk_cafe_dong FOREIGN KEY (dong_id) REFERENCES 
       ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 12. chicken 테이블 생성 (dong_id 외래키 사용)
CREATE TABLE chicken (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   gu_name VARCHAR(255) NOT NULL,
   dong_id INT NOT NULL,
   latitude DECIMAL(10,7) NOT NULL,
   longitude DECIMAL(10,7) NOT NULL,
   CONSTRAINT fk_chicken_dong FOREIGN KEY (dong_id) REFERENC
       ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 13. leisure 테이블 생성 (dong_id 외래키 사용)
CREATE TABLE leisure (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   gu_name VARCHAR(255) NOT NULL,
   dong_id INT NOT NULL,
   latitude DECIMAL(10,7) NOT NULL,
   longitude DECIMAL(10,7) NOT NULL,
   CONSTRAINT fk_leisure_dong FOREIGN KEY (dong_id) REFERENCE
       ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 14. property_score 테이블 생성
CREATE TABLE property_score (
   id INT AUTO_INCREMENT PRIMARY KEY,
   property_id INT NOT NULL,
   transport_count INT DEFAULT 0,
   transport_score FLOAT DEFAULT 0,
   restaurant_count INT DEFAULT 0,
   restaurant_score FLOAT DEFAULT 0,
   health_count INT DEFAULT 0,
   health_score FLOAT DEFAULT 0,
   convenience_count INT DEFAULT 0,
   convenience_score FLOAT DEFAULT 0,
   cafe_count INT DEFAULT 0,
   cafe_score FLOAT DEFAULT 0,
   chicken_count INT DEFAULT 0,
   chicken_score FLOAT DEFAULT 0,
   leisure_count INT DEFAULT 0,
   leisure_score FLOAT DEFAULT 0,
   updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
   CONSTRAINT fk_property_score FOREIGN KEY (property_id) REFER
       ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

▼ 아래의 SQL 구문을 실행해 동 정보를 삽입합니다.

```
LOAD DATA INFILE '/var/lib/mysql-files/seoul_dong_data_new.csv'
INTO TABLE dong
CHARACTER SET utf8mb4
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(
  @address1,
  @address2,
  @address3,
  @population,
  @area,
  @safe,
  @leisure,
  @restaurant,
  @health,
  @mart,
  @convenience,
  @transport
)
SET
  name = @address3,
  gu_name = @address2,
  safe = @safe,
  leisure = @leisure,
  restaurant = @restaurant,
  health = @health,
  mart= @mart,
  convenience = @convenience,
  transport = @transport;
```

▼ 아래의 SQL 구문을 실행해 동 ID를 업데이트합니다.

```
-- 0. 외래키 체크와 안전 업데이트 모드 해제
SET FOREIGN_KEY_CHECKS = 0;
SET SQL_SAFE_UPDATES = 0;

-- 1. dong 테이블의 모든 데이터 초기화
TRUNCATE TABLE dong;

-- 2. CSV 데이터를 dong 테이블에 로드
LOAD DATA INFILE '/var/lib/mysql-files/seoul_dong_data_new.csv'
INTO TABLE dong
CHARACTER SET utf8mb4
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(
  @gu_name,
  @dong_name,
  @safe,
  @leisure,
  @restaurant,
  @health,
  @mart,
  @convenience,
  @transport,
  @dong_id
)
SET
  gu_name = @gu_name,
  name = @dong_name,
  safe = @safe,
  leisure = @leisure,
  restaurant = @restaurant,
  health = @health,
  mart = @mart,
  convenience = @convenience,
  transport = @transport,
  dong_id = @dong_id;

-- 3. dong 테이블의 모든 문자열 컬럼에서 "\r" 문자 제거
UPDATE dong
SET gu_name = REPLACE(gu_name, '\r', ''),
    name = REPLACE(name, '\r', '');

-- 4. property 테이블의 gu_name과 dong_name을 기준으로 dong 테이블과
--    property 테이블의 dong_id를 dong 테이블의 dong_id로 업데이트
UPDATE property p
JOIN dong d ON p.gu_name = d.gu_name AND p.dong_name = d.name
SET p.dong_id = d.dong_id;

-- 5. 외래키 체크와 안전 업데이트 모드 복원
SET FOREIGN_KEY_CHECKS = 1;
SET SQL_SAFE_UPDATES = 1;

select count(*) from property;
select count(*) from property_score;
```

▼ 아래의 SQL 구문을 실행해 카테고리별 위치 및 정보를 삽입합니다.

```sql
-- 1. Transport 데이터
DROP TABLE IF EXISTS staging_transport;
CREATE TABLE staging_transport (
  name VARCHAR(255),
  gu_name VARCHAR(255),
  dong_name VARCHAR(255),
  latitude VARCHAR(50),
  longitude VARCHAR(50),
  type VARCHAR(30)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

LOAD DATA INFILE '/var/lib/mysql-files/transport_data2.csv'
INTO TABLE staging_transport
CHARACTER SET utf8mb4
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(name, gu_name, dong_name, latitude, longitude, type);

INSERT INTO transport (name, gu_name, dong_id, latitude, longitude, t
SELECT
  s.name,
  s.gu_name,
  d.dong_id,
  CAST(ROUND(s.latitude, 7) AS DECIMAL(10,7)),
  CAST(ROUND(s.longitude, 7) AS DECIMAL(10,7)),
  s.type
FROM staging_transport s
JOIN dong d
  ON LOWER(TRIM(s.dong_name)) = LOWER(TRIM(d.name));

DROP TABLE staging_transport;


-- 2. Restaurant 데이터
```

```sql
DROP TABLE IF EXISTS staging_restaurant;
CREATE TABLE staging_restaurant (
  name VARCHAR(255),
  gu_name VARCHAR(255),
  dong_name VARCHAR(255),
  latitude VARCHAR(50),
  longitude VARCHAR(50)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

LOAD DATA INFILE '/var/lib/mysql-files/restaurant_data2.csv'
INTO TABLE staging_restaurant
CHARACTER SET utf8mb4
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(name, gu_name, dong_name, latitude, longitude);

INSERT INTO restaurant (name, gu_name, dong_id, latitude, longitude)
SELECT
  s.name,
  s.gu_name,
  d.dong_id,
  CAST(ROUND(s.latitude, 7) AS DECIMAL(10,7)),
  CAST(ROUND(s.longitude, 7) AS DECIMAL(10,7))
FROM staging_restaurant s
JOIN dong d
  ON LOWER(TRIM(s.dong_name)) = LOWER(TRIM(d.name));

DROP TABLE staging_restaurant;


-- 3. Health 데이터
DROP TABLE IF EXISTS staging_health;
CREATE TABLE staging_health (
  name VARCHAR(255),
  gu_name VARCHAR(255),
  dong_name VARCHAR(255),
  latitude VARCHAR(50),
  longitude VARCHAR(50)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

LOAD DATA INFILE '/var/lib/mysql-files/health_data2.csv'
INTO TABLE staging_health
CHARACTER SET utf8mb4
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(name, gu_name, dong_name, latitude, longitude);

INSERT INTO health (name, gu_name, dong_id, latitude, longitude)
SELECT
  s.name,
  s.gu_name,
  d.dong_id,
  CAST(ROUND(s.latitude, 7) AS DECIMAL(10,7)),
  CAST(ROUND(s.longitude, 7) AS DECIMAL(10,7))
FROM staging_health s
JOIN dong d
  ON LOWER(TRIM(s.dong_name)) = LOWER(TRIM(d.name));

DROP TABLE staging_health;


-- 4. Convenience 데이터
-- CSV 파일 구조: name, type, gu_name, dong_name, latitude, longitude
DROP TABLE IF EXISTS staging_convenience;
CREATE TABLE staging_convenience (
  name VARCHAR(255),
  type VARCHAR(30),
  gu_name VARCHAR(255),
  dong_name VARCHAR(255),
  latitude VARCHAR(50),
  longitude VARCHAR(50)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```sql
LOAD DATA INFILE '/var/lib/mysql-files/convenience_data2.csv'
INTO TABLE staging_convenience
CHARACTER SET utf8mb4
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(name, type, gu_name, dong_name, latitude, longitude);

INSERT INTO convenience (name, gu_name, dong_id, latitude, longituc
SELECT
  s.name,
  s.gu_name,
  d.dong_id,
  CAST(ROUND(s.latitude, 7) AS DECIMAL(10,7)),
  CAST(ROUND(s.longitude, 7) AS DECIMAL(10,7)),
  s.type
FROM staging_convenience s
JOIN dong d
  ON LOWER(TRIM(s.dong_name)) = LOWER(TRIM(d.name));

DROP TABLE staging_convenience;


-- 5. Cafe 데이터
DROP TABLE IF EXISTS staging_cafe;
CREATE TABLE staging_cafe (
  name VARCHAR(255),
  gu_name VARCHAR(255),
  dong_name VARCHAR(255),
  latitude VARCHAR(50),
  longitude VARCHAR(50)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

LOAD DATA INFILE '/var/lib/mysql-files/cafe_data2.csv'
INTO TABLE staging_cafe
CHARACTER SET utf8mb4
```

```
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(name, gu_name, dong_name, latitude, longitude);

INSERT INTO cafe (name, gu_name, dong_id, latitude, longitude)
SELECT
  s.name,
  s.gu_name,
  d.dong_id,
  CAST(ROUND(s.latitude, 7) AS DECIMAL(10,7)),
  CAST(ROUND(s.longitude, 7) AS DECIMAL(10,7))
FROM staging_cafe s
JOIN dong d
  ON LOWER(TRIM(s.dong_name)) = LOWER(TRIM(d.name));

DROP TABLE staging_cafe;


-- 6. Chicken 데이터
DROP TABLE IF EXISTS staging_chicken;
CREATE TABLE staging_chicken (
  name VARCHAR(255),
  gu_name VARCHAR(255),
  dong_name VARCHAR(255),
  latitude VARCHAR(50),
  longitude VARCHAR(50)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

LOAD DATA INFILE '/var/lib/mysql-files/chicken_data2.csv'
INTO TABLE staging_chicken
CHARACTER SET utf8mb4
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(name, gu_name, dong_name, latitude, longitude);
```

```sql
INSERT INTO chicken (name, gu_name, dong_id, latitude, longitude)
SELECT
  s.name,
  s.gu_name,
  d.dong_id,
  CAST(ROUND(s.latitude, 7) AS DECIMAL(10,7)),
  CAST(ROUND(s.longitude, 7) AS DECIMAL(10,7))
FROM staging_chicken s
JOIN dong d
  ON LOWER(TRIM(s.dong_name)) = LOWER(TRIM(d.name));

DROP TABLE staging_chicken;


-- 1. 스테이징 테이블 생성
DROP TABLE IF EXISTS staging_leisure;
CREATE TABLE staging_leisure (
  name VARCHAR(255),
  gu_name VARCHAR(255),
  dong_name VARCHAR(255),
  latitude VARCHAR(50),
  longitude VARCHAR(50)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 2. CSV 파일 로드 (leisure_data2.csv)
LOAD DATA INFILE '/var/lib/mysql-files/leisure_data2.csv'
INTO TABLE staging_leisure
CHARACTER SET utf8mb4
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(name, gu_name, dong_name, latitude, longitude);

-- 3. dong 테이블과 조인 후 leisure 테이블로 데이터 삽입
INSERT INTO leisure (name, gu_name, dong_id, latitude, longitude)
SELECT
```

```
  s.name,
  s.gu_name,
  d.dong_id,
  CAST(ROUND(s.latitude, 7) AS DECIMAL(10,7)),
  CAST(ROUND(s.longitude, 7) AS DECIMAL(10,7))
FROM staging_leisure s
JOIN dong d
  ON LOWER(TRIM(s.dong_name)) = LOWER(TRIM(d.name));

-- 4. 스테이징 테이블 삭제
DROP TABLE staging_leisure;
```

▼ 중간 점검 과정으로, 데이터의 개수를 카운트합니다.

```
SELECT 'user' AS table_name, COUNT(*) AS cnt FROM user
UNION ALL
SELECT 'dong', COUNT(*) FROM dong
UNION ALL
SELECT 'property', COUNT(*) FROM property
UNION ALL
SELECT 'user_preference', COUNT(*) FROM user_preference
UNION ALL
SELECT 'transport', COUNT(*) FROM transport
UNION ALL
SELECT 'restaurant', COUNT(*) FROM restaurant
UNION ALL
SELECT 'health', COUNT(*) FROM health
UNION ALL
SELECT 'convenience', COUNT(*) FROM convenience
UNION ALL
SELECT 'cafe', COUNT(*) FROM cafe
UNION ALL
SELECT 'chicken', COUNT(*) FROM chicken
UNION ALL
SELECT 'leisure', COUNT(*) FROM leisure
UNION ALL
SELECT 'property_score', COUNT(*) FROM property_score;
```

▼ 아래의 SQL 구문을 실행해 매물 데이터 입력 검증 트리거를 활성화시킵니다.

```
DELIMITER $$
CREATE TRIGGER before_property_insert
BEFORE INSERT ON property
FOR EACH ROW
BEGIN
  DECLARE tmp_gu_name VARCHAR(255);
  SELECT gu_name INTO tmp_gu_name FROM dong WHERE dong_id =
  IF tmp_gu_name != NEW.gu_name THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'gu_name does 
  END IF;
END$$
DELIMITER ;
```

▼ 아래의 SQL 구문을 실행해 카테고리 위치 테이블 입력 검증 트리거를 활성화시킵니다.

```
DELIMITER $$
CREATE TRIGGER before_cafe_insert
BEFORE INSERT ON cafe
FOR EACH ROW
BEGIN
  DECLARE correct_dong_id INT;
  -- gu_name에 해당하는 신사동의 dong_id를 가져옴
  SELECT dong_id INTO correct_dong_id
  FROM dong
  WHERE name = '신사동' AND gu_name = NEW.gu_name
  LIMIT 1;

  IF correct_dong_id IS NOT NULL THEN
    SET NEW.dong_id = correct_dong_id;
  ELSE
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'cafe insert error: 해당 구에 신사동이 존재하지
  END IF;
END$$
DELIMITER ;
DELIMITER $$
CREATE TRIGGER before_chicken_insert
BEFORE INSERT ON chicken
FOR EACH ROW
BEGIN
  DECLARE correct_dong_id INT;
  SELECT dong_id INTO correct_dong_id
  FROM dong
  WHERE name = '신사동' AND gu_name = NEW.gu_name
  LIMIT 1;

  IF correct_dong_id IS NOT NULL THEN
    SET NEW.dong_id = correct_dong_id;
  ELSE
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'chicken insert error: 해당 구에 신사동이 존재
  END IF;
END$$
DELIMITER ;


DELIMITER $$
CREATE TRIGGER before_convenience_insert
BEFORE INSERT ON convenience
FOR EACH ROW
BEGIN
  DECLARE correct_dong_id INT;
  SELECT dong_id INTO correct_dong_id
  FROM dong
  WHERE name = '신사동' AND gu_name = NEW.gu_name
  LIMIT 1;

  IF correct_dong_id IS NOT NULL THEN
    SET NEW.dong_id = correct_dong_id;
  ELSE
    SIGNAL SQLSTATE '45000'
```

```
      SET MESSAGE_TEXT = 'convenience insert error: 해당 구에 신사동이
  END IF;
END$$
DELIMITER ;


DELIMITER $$
CREATE TRIGGER before_health_insert
BEFORE INSERT ON health
FOR EACH ROW
BEGIN
  DECLARE correct_dong_id INT;
  SELECT dong_id INTO correct_dong_id
  FROM dong
  WHERE name = '신사동' AND gu_name = NEW.gu_name
  LIMIT 1;

  IF correct_dong_id IS NOT NULL THEN
    SET NEW.dong_id = correct_dong_id;
  ELSE
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'health insert error: 해당 구에 신사동이 존재하
  END IF;
END$$
DELIMITER ;


DELIMITER $$
CREATE TRIGGER before_leisure_insert
BEFORE INSERT ON leisure
FOR EACH ROW
BEGIN
  DECLARE correct_dong_id INT;
  SELECT dong_id INTO correct_dong_id
  FROM dong
  WHERE name = '신사동' AND gu_name = NEW.gu_name
  LIMIT 1;
  IF correct_dong_id IS NOT NULL THEN
    SET NEW.dong_id = correct_dong_id;
  ELSE
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'leisure insert error: 해당 구에 신사동이 존재히
  END IF;
END$$
DELIMITER ;


DELIMITER $$
CREATE TRIGGER before_restaurant_insert
BEFORE INSERT ON restaurant
FOR EACH ROW
BEGIN
  DECLARE correct_dong_id INT;
  SELECT dong_id INTO correct_dong_id
  FROM dong
  WHERE name = '신사동' AND gu_name = NEW.gu_name
  LIMIT 1;

  IF correct_dong_id IS NOT NULL THEN
    SET NEW.dong_id = correct_dong_id;
  ELSE
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'restaurant insert error: 해당 구에 신사동이 존
  END IF;
END$$
DELIMITER ;


DELIMITER $$
CREATE TRIGGER before_transport_insert
BEFORE INSERT ON transport
FOR EACH ROW
BEGIN
  DECLARE correct_dong_id INT;
  SELECT dong_id INTO correct_dong_id
  FROM dong
  WHERE name = '신사동' AND gu_name = NEW.gu_name
  LIMIT 1;

  IF correct_dong_id IS NOT NULL THEN
    SET NEW.dong_id = correct_dong_id;
  ELSE
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'transport insert error: 해당 구에 신사동이 존
  END IF;
END$$
DELIMITER ;
```

▼ 아래의 SQL 구문을 실행해 매물 정보 삽입이 정상적인지 확인하는 트리거를 활성화시킵니다.

```
SET SQL_SAFE_UPDATES = 0;
SELECT
    p.property_id,
    p.dong_id,
    p.gu_name,
    d.dong_id AS expected_dong_id,
    d.name
FROM property p
JOIN dong d ON p.gu_name = d.gu_name
WHERE d.name = '신사동';
SET SQL_SAFE_UPDATES = 1;
```

▼ 아래의 SQL 구문을 실행해 매물 정보를 삽입합니다.

```
-- 혹시 매물 테이블 새로 한다면 테이블 지우고 다시 실행


SET FOREIGN_KEY_CHECKS=0;
-- 1. 기존 스테이징 테이블이 존재하면 삭제
DROP TABLE IF EXISTS staging_property;

SET FOREIGN_KEY_CHECKS=1;
```

```
-- 2. 스테이징 테이블 생성
CREATE TABLE staging_property (
   room_type        VARCHAR(20),
   contract_type     VARCHAR(20),
   price            VARCHAR(30),
   address          VARCHAR(255),
   description       TEXT,
   area             VARCHAR(50),
   floor_info        VARCHAR(50),
   room_bath_count   VARCHAR(50),
   maintenance_fee   VARCHAR(50),
   move_in_date      VARCHAR(50),
   direction         VARCHAR(50),
   image_url         TEXT,
   sale_price        VARCHAR(50),
   deposit           VARCHAR(50),
   monthly_rent      VARCHAR(50),
   latitude          VARCHAR(50),
   longitude         VARCHAR(50),
   dong_name         VARCHAR(255),
   gu_name           VARCHAR(255)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 3. CSV 파일의 데이터를 스테이징 테이블에 로드
LOAD DATA INFILE '/var/lib/mysql-files/off_coords_dongname.csv'
INTO TABLE staging_property
CHARACTER SET utf8mb4
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(room_type, contract_type, price, address, description, area, floor_info

select * from staging_property;

-- 4. staging 테이블과 dong 테이블을 내부 조인하여,
--    매칭되는 dong_id가 있는 행만 property 테이블에 삽입
```

```sql
INSERT INTO property (
  room_type,
  contract_type,
  price,
  address,
  description,
  area,
  floor_info,
  room_bath_count,
  maintenance_fee,
  move_in_date,
  direction,
  image_url,
  sale_price,
  deposit,
  monthly_rent,
  latitude,
  longitude,
  dong_id,
  gu_name,
  created_at
)
SELECT
  s.room_type,
  s.contract_type,
  s.price,
  s.address,
  s.description,
  s.area,
  s.floor_info,
  s.room_bath_count,
  CAST(ROUND(CAST(NULLIF(TRIM(s.maintenance_fee), '') AS DECIM
  s.move_in_date,
  s.direction,
  s.image_url,
  CAST(ROUND(CAST(NULLIF(TRIM(s.sale_price), '') AS DECIMAL(10,
  CAST(ROUND(CAST(NULLIF(TRIM(s.deposit), '') AS DECIMAL(10,2)),
  CAST(ROUND(CAST(NULLIF(TRIM(s.monthly_rent), '') AS DECIMAL(
  CAST(ROUND(s.latitude, 7) AS FLOAT),
  CAST(ROUND(s.longitude, 7) AS FLOAT),
  d.dong_id,
  s.gu_name,
  NOW()
FROM staging_property s
JOIN dong d
  ON LOWER(TRIM(s.dong_name)) = LOWER(TRIM(d.name));
-- 5. 작업 완료 후 스테이징 테이블 삭제
DROP TABLE staging_property;
```

▼ 아래의 SQL 구문을 실행해 각 매물의 동 정보를 삽입합니다.

```
SET SQL_SAFE_UPDATES = 0;

UPDATE property p
JOIN dong d ON p.dong_id = d.dong_id
SET p.dong_name = d.name;
```

▼ 개행문자, 특수문자 등이 포함되어있을 경우, 이를 제거합니다.

```
SET SQL_SAFE_UPDATES = 0;
UPDATE property
SET
  room_type       = REPLACE(room_type, '\r', ''),
  contract_type   = REPLACE(contract_type, '\r', ''),
  price           = REPLACE(price, '\r', ''),
  address         = REPLACE(address, '\r', ''),
  description     = REPLACE(description, '\r', ''),
  area            = REPLACE(area, '\r', ''),
  floor_info      = REPLACE(floor_info, '\r', ''),
  room_bath_count = REPLACE(room_bath_count, '\r', ''),
  move_in_date    = REPLACE(move_in_date, '\r', ''),
  direction       = REPLACE(direction, '\r', ''),
  image_url       = REPLACE(image_url, '\r', ''),
  gu_name         = REPLACE(gu_name, '\r', ''),
  dong_name       = REPLACE(dong_name, '\r', '');
SET SQL_SAFE_UPDATES = 1;
```

▼ 특정 구(관악구, 강남구)의 동 ID를 재조정합니다.

```
UPDATE transport t
JOIN dong d ON t.gu_name = d.gu_name
SET t.dong_id = d.dong_id
WHERE d.name = '신사동'
  AND t.dong_id != d.dong_id
  AND t.id > 0;



 ----중복 삭제
 DELETE ps
FROM property_score ps
JOIN property p ON ps.property_id = p.property_id
WHERE (p.gu_name = '강남구' AND p.dong_id = 87)
   OR (p.gu_name = '관악구' AND p.dong_id = 16);

DELETE FROM property
WHERE (gu_name = '강남구' AND dong_id = 87)
   OR (gu_name = '관악구' AND dong_id = 16);
```

▼ 아래의 SQL 구문을 실행해 매물 점수를 계산 및 저장합니다.

```
curl -X POST "http://localhost:8000/recalculate/batch" \
     -H "Content-Type: application/json" \
     -d '{
           "batch_size": 10000,
           "max_workers": 12
         }'


curl -X POST "http://localhost:8000/recalculate/incomplete" \
     -H "Content-Type: application/json" \
     -d '{
           "batch_size": 10000,
           "max_workers": 12
         }'



     #로그 확인
     docker logs -f recommend_container | grep '\[Progress\]'
```

▼ balltree 알고리즘 적용을 위해 스카마에 공간 인덱싱을 적용합니다.

```
3. 전체 과정 요약
#안전 업데이트 모드 문제 해결:

#세션에서 안전 업데이트 모드 비활성화:

SET SQL_SAFE_UPDATES = 0;
#또는 WHERE절에 인덱스 컬럼 추가:


UPDATE property
SET location = ST_GeomFromText(CONCAT('POINT(', longitude, ' ', lati
WHERE property_id IS NOT NULL;
#공간 컬럼 location 업데이트:

#매물의 longitude와 latitude를 사용해 POINT형 공간 데이터를 생성:


UPDATE property
SET location = ST_GeomFromText(CONCAT('POINT(', longitude, ' ', lati
WHERE property_id IS NOT NULL;
#location 컬럼을 NOT NULL로 변경:


ALTER TABLE property
MODIFY location GEOMETRY NOT NULL;
#SPATIAL 인덱스 생성:
```

```
ALTER TABLE property
ADD SPATIAL INDEX(location);
```

3. MongoDB를 설치합니다.
    a. 8.0.x 버전이 필요합니다.
4. 다음의 구문들을 실행합니다.

▼ 댓글

```
NoSQL(MongoDB)
(댓글)
COMMENT
id
property_id 매물 id
nickname 댓글 작성자
context 댓글

{
  "_id": { "$oid": "623a1f7a5c3a4e2e5f6b7a1b" },
  "property_id": 101,
  "nickname": "JohnDoe",
  "context": "This property looks amazing!"
}
```

▼ 찜

```

매물 찜(FavoriteProps)
{
  "_id": "someUniqueId",
  "userId": 123,
  "propertyId": 10,
  "title": "강남 아파트",
  "price": 850000000,
  "contractType": "매매",
  "address": "서울특별시 강남구 역삼동",
  "dongId": 33,
  "latitude": 37.498095,
  "longitude": 127.027610,
  "createdAt": "2025-03-11T09:30:00Z"
}

동네 찜 (FavoriteDongs)
{
  "_id": "someUniqueId",
  "userId": 123,
  "dongId": 33,
  "dongName": "강남구 역삼동",
  "safe": 8.5,
  "leisure": 7.8,
  "restaurant": 9.2,
  "health": 7.5,
  "convenience": 8.9,
  "transport": 9.0,
  "cafe": 8.0,
  "bar": 7.3,
  "createdAt": "2025-03-11T09:30:00Z"
}
```

5. ElasticSearch를 설치합니다.
    a. 8.17.x 버전이 필요합니다.
6. 다음의 구문을 실행합니다.

▼ 액티비티 로그 생성

```
NoSQL(일라스틱서치)
(액티비티로그)
activity_log
time (로그찍힌시간)
action (click, favorite, view)
email (이메일)


{
  "id": "activity_log_001",
  "user_id" : "db 인덱스값",
  "property_id" : "property_id"
  "time": "2025-03-10T12:34:56Z",
  "action": "click", (찜, 클릭, 댓글, qa)
  "age" : 20,
  "gender" : "male"
}
```

▼ 동 정보

```
{
  "dong_id": 1,
  "name": "DongA",
  "safe": 4.5,
  "leisure": 4.0,
  "restaurant": 4.2,
  "health": 3.8,
  "convenience": 4.1,
  "transport": 4.0,
  "cafe": 4.3,
  "bar": 3.9
}
```

---

## 1. 사용 도구

a. 이슈관리 : JIRA

b. 형상 및 버전 관리 : GitLab

c. Doc 제작 및 소통 : Mattermost, Notion

d. 목업 제작 : Figma

e. UCC : Movavi

f. CI/CD : Docker, Jenkins

## 2. 개발 환경

a. 서버 환경 : Ubuntu 22.04

b. IDE : VSCode, Intellij Ultimate

c. 원격 접속 : MobaXterm

d. 백엔드 : SpringBoot, FastAPI

e. 프론트엔드 : React, Redux, Javascript

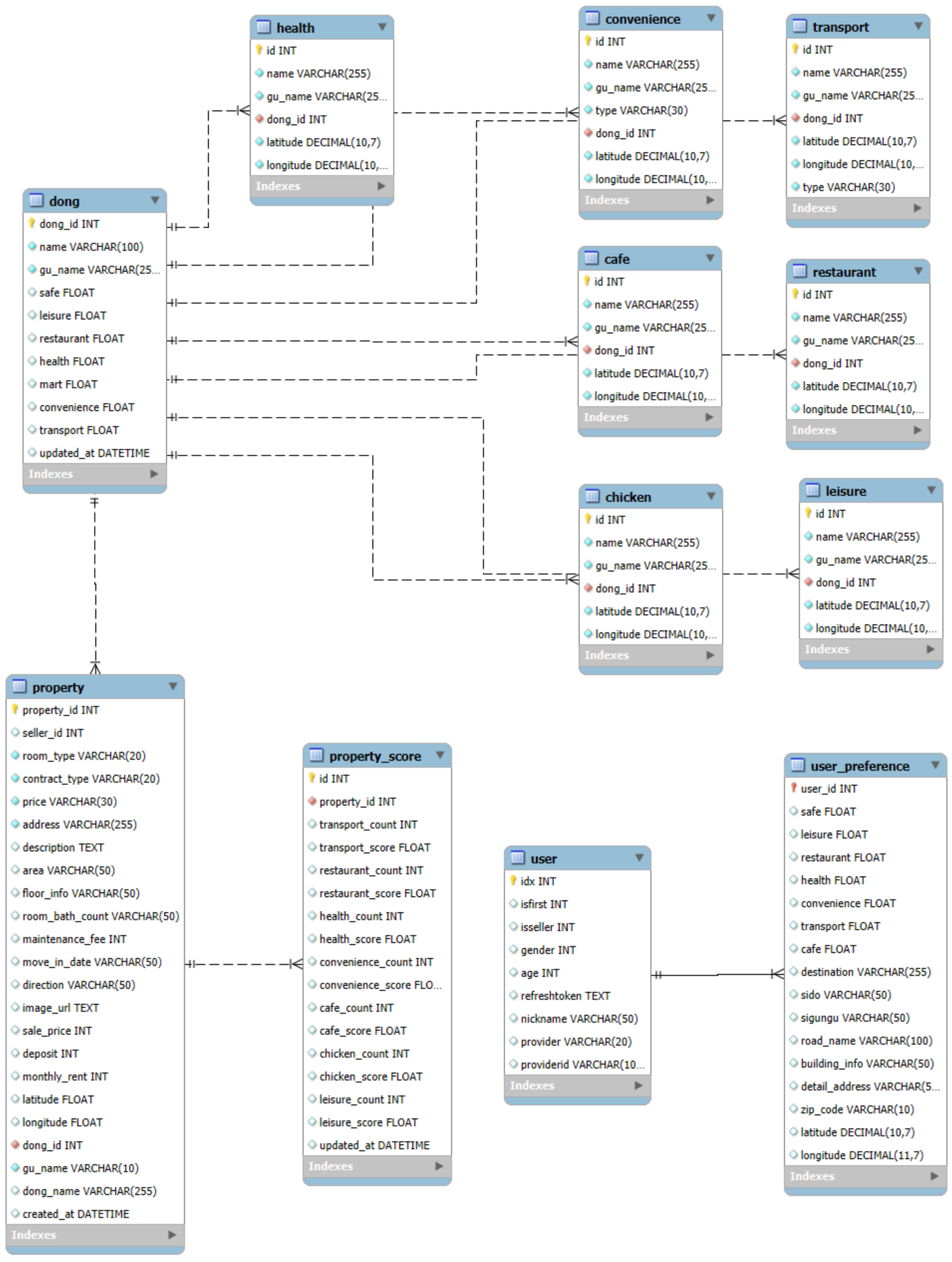
f. Database

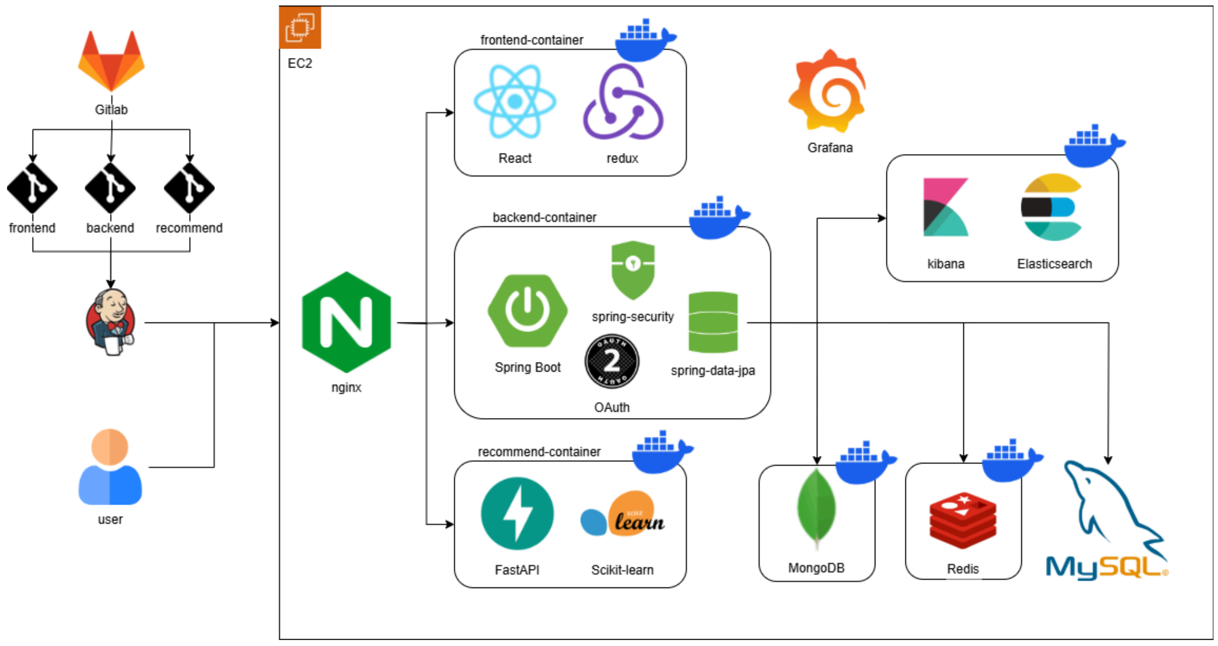
    a. MongoDB

    b. ElasticSearch

    c. MySQL

## 3. 시스템 아키텍쳐 및 기술 스택


health
id INT
name VARCHAR(255)
gu_name VARCHAR(25...
dong_id INT
latitude DECIMAL(10,7)
longitude DECIMAL(10,...
Indexes
convenience
id INT
name VARCHAR(255)
gu_name VARCHAR(25...
type VARCHAR(30)
dong_id INT
latitude DECIMAL(10,7)
longitude DECIMAL(10,...
Indexes
transport
id INT
name VARCHAR(255)
gu_name VARCHAR(25...
dong_id INT
latitude DECIMAL(10,7)
longitude DECIMAL(10,...
type VARCHAR(30)
Indexes
dong
dong_id INT
name VARCHAR(100)
gu_name VARCHAR(25...
safe FLOAT
leisure FLOAT
restaurant FLOAT
health FLOAT
mart FLOAT
convenience FLOAT
transport FLOAT
updated_at DATETIME
Indexes
cafe
id INT
name VARCHAR(255)
gu_name VARCHAR(25...
dong_id INT
latitude DECIMAL(10,7)
longitude DECIMAL(10,...
Indexes
restaurant
id INT
name VARCHAR(255)
gu_name VARCHAR(25...
dong_id INT
latitude DECIMAL(10,7)
longitude DECIMAL(10,...
Indexes
chicken
id INT
name VARCHAR(255)
gu_name VARCHAR(25...
dong_id INT
latitude DECIMAL(10,7)
longitude DECIMAL(10,...
Indexes
leisure
id INT
name VARCHAR(255)
gu_name VARCHAR(25...
dong_id INT
latitude DECIMAL(10,7)
longitude DECIMAL(10,...
Indexes
property
property_id INT
seller_id INT
room_type VARCHAR(20)
contract_type VARCHAR(20)
price VARCHAR(30)
address VARCHAR(255)
description TEXT
area VARCHAR(50)
floor_info VARCHAR(50)
room_bath_count VARCHAR(50)
maintenance_fee INT
move_in_date VARCHAR(50)
direction VARCHAR(50)
image_url TEXT
sale_price INT
deposit INT
monthly_rent INT
latitude FLOAT
longitude FLOAT
dong_id INT
gu_name VARCHAR(10)
dong_name VARCHAR(255)
created_at DATETIME
Indexes
property_score
id INT
property_id INT
transport_count INT
transport_score FLOAT
restaurant_count INT
restaurant_score FLOAT
health_count INT
health_score FLOAT
convenience_count INT
convenience_score FLO...
cafe_count INT
cafe_score FLOAT
chicken_count INT
chicken_score FLOAT
leisure_count INT
leisure_score FLOAT
updated_at DATETIME
Indexes
user
idx INT
isfirst INT
isseller INT
gender INT
age INT
refreshtoken TEXT
nickname VARCHAR(50)
provider VARCHAR(20)
providerid VARCHAR(10...
Indexes
user_preference
user_id INT
safe FLOAT
leisure FLOAT
restaurant FLOAT
health FLOAT
convenience FLOAT
transport FLOAT
cafe FLOAT
destination VARCHAR(255)
sido VARCHAR(50)
sigungu VARCHAR(50)
road_name VARCHAR(100)
building_info VARCHAR(50)
detail_address VARCHAR(5...
zip_code VARCHAR(10)
latitude DECIMAL(10,7)
longitude DECIMAL(11,7)
Indexes



### 3-1. 프론트엔드

- React - 18.3.1
- Redux - 2.6.1
- javascript - ES6

### 3-2. 백엔드

- SpringBoot - 3.4.3
- SpringSecurity(OAuth2 Jose) - 6.4.3
- Spring Data JPA - 3.4.3
- Redis - 7.4.2
- FastAPI - 0.115.12
- Python - 3.10.11

### 3-3. DB

- MySQL - 8.0.41
- ElasticSearch - 8.17.3
- Kibana - 8.17.3
- MongoDB - 8.0.6

### 3-4. 인프라

- Nginx - 1.27.4
- Jenkins - 2.492.2
- Grafana - 9.1.7
- Loki - 2.8.2

## 4. 서버 설정

▼ 백엔드 application.properties

```
spring.application.name=zeepseek

server.port=8081

# MySQL Configuration
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:j12e203.p.ssafy.io}:3
spring.datasource.username=ssafy
spring.datasource.password=ssafyssafy
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# MongoDB Configuration
spring.data.mongodb.uri=${MONGODB_URI}
# elasticsearch config
elasticsearch.host=${ELASTICSEARCH_HOST:j12e203.p.ssafy.io}
elasticsearch.username=${ES_USERNAME}
elasticsearch.password=${ES_PASSWORD}

# gpt api key
openai.api.key=${OPENAI_API_KEY}
kakao.api.key=${KAKAO_REST_API_KEY}
recommendation.api.url=http://recommend_container:8000/recommend

# define TMap API KEY
tmap.api.key=vTMUKZtyfw1SNexj28OW39aSJipWeUIV6DuCEIMW

# JPA default configuration
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Diale
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true

# OAuth configs
spring.security.oauth2.client.registration.kakao.client-name=Kakao
spring.security.oauth2.client.registration.naver.client-name=Naver
spring.security.oauth2.client.provider.naver.authorization-uri=https://nid.n
spring.security.oauth2.client.provider.naver.token-uri=https://nid.naver.co
spring.security.oauth2.client.provider.naver.user-info-uri=https://openapi.
spring.security.oauth2.client.provider.naver.user-name-attribute=respons

# KaKao OAuth2 configs
spring.security.oauth2.client.registration.kakao.client-id=38c66a61f9b3af
spring.security.oauth2.client.registration.kakao.client-secret=ff3y5nGKvxl
spring.security.oauth2.client.registration.kakao.redirect-uri=https://j12e20
spring.security.oauth2.client.registration.kakao.authorization-grant-type=
spring.security.oauth2.client.registration.kakao.scope=profile_nickname,a
spring.security.oauth2.client.provider.kakao.authorization-uri=https://kaut
spring.security.oauth2.client.provider.kakao.token-uri=https://kauth.kakac
spring.security.oauth2.client.provider.kakao.user-info-uri=https://kapi.kak
spring.security.oauth2.client.provider.kakao.user-name-attribute=id
spring.security.oauth2.client.registration.kakao.client-authentication-meth
# Naver OAuth2 configs
spring.security.oauth2.client.registration.naver.client-id=a4OkI4nf4Y547x
spring.security.oauth2.client.registration.naver.client-secret=TlmZstA9Qd
spring.security.oauth2.client.registration.naver.redirect-uri=https://j12e20
spring.security.oauth2.client.registration.naver.authorization-grant-type=a
spring.security.oauth2.client.registration.naver.scope=name,email

# Define oauth2s' endpoints
app.auth.login-endpoint=/api/v1/auth/login
app.auth.logout-endpoint=/api/v1/auth/logout
app.auth.refresh-endpoint=/api/v1/auth/refresh
app.auth.redirect-endpoint=/api/v1/auth/redirect

# Additional oauth configs
app.auth.cookie.secure=false
app.auth.cookie.httpOnly=true

# JWT Default configs
app.auth.token-secret=HelloThisIsAirdexSpeakingAsYouKnowILikeGoingT

# set Access token's expiration time(THAT MEANS MILLISECONDS)
app.auth.access-token-expiration-msec=3600000

# set Access token's expiration time(THAT MEANS MILLISECONDS)
app.auth.refresh-token-expiration-msec=604800000

# Set loggers' POV
logging.level.org.springframework.security=DEBUG
logging.level.org.springframework.web=DEBUG

# OAuth's log
logging.level.com.zeepseek.backend.domain.auth.service.AuthServiceImp
logging.level.com.zeepseek.backend.domain.auth.security.jwt.JwtTokenP
logging.level.com.zeepseek.backend.domain.auth.controller.AuthControlle
logging.level.com.zeepseek.backend.domain.user.repository.UserReposit

# logs about Transaction(can find Transactional errors)
logging.level.org.springframework.transaction=DEBUG
logging.level.org.springframework.orm.jpa=DEBUG

#Hibernate log
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

logging.level.root=INFO
logging.level.com.zeepseek.backend=DEBUG

# disabled spring security
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.se
# Swagger Configuration
springdoc.api-docs.path=/v3/api-docs
springdoc.swagger-ui.path=/swagger-ui.html
springdoc.swagger-ui.enabled=true
springdoc.default-consumes-media-type=application/json
springdoc.default-produces-media-type=application/json

# Kakao Map API Configuration
# íê²½ ë³ìì ì£¼ìëì§ ìì ê²½ì° ê¸°ë³¸ê° ì¬ì©
# VITE_APP_KAKAO_MAP_API_KEY=${VITE_APP_KAKAO_MAP_API_KEY:ì¹
```

▼ 프론트엔드 package.json

```
{
  "name": "fe",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "devDependencies": {
    "@eslint/js": "^9.21.0",
    "@types/proj4": "^2.5.6",
    "@types/react": "^19.0.10",
    "@types/react-dom": "^19.0.4",
    "@vitejs/plugin-react": "^4.3.4",
    "eslint": "^9.21.0",
    "eslint-plugin-react-hooks": "^5.1.0",
    "eslint-plugin-react-refresh": "^0.4.19",
    "globals": "^15.15.0",
    "vite": "^6.2.0",
    "vite-plugin-pwa": "^0.21.1"
  },
  "dependencies": {
    "@reduxjs/toolkit": "^2.6.1",
    "axios": "^1.8.4",
    "framer-motion": "^12.6.2",
    "lodash": "^4.17.21",
    "proj4": "^2.15.0",
    "rc-slider": "^11.1.8",
    "react": "^18.3.1",
    "react-daum-postcode": "^3.2.0",
    "react-dom": "^18.3.1",
    "react-icons": "^5.5.0",
    "react-redux": "^9.2.0",
    "react-responsive-carousel": "^3.2.23",
    "react-router-dom": "^7.3.0",
    "recharts": "^2.15.1"
  }
}
```

▼ EC2 nginx.conf

```
server {
    listen 80;
    server_name j12e203.p.ssafy.io;

    return 301 https://$host$request_uri;
}



server {
    listen 443 ssl;
    server_name j12e203.p.ssafy.io;

    ssl_certificate /etc/nginx/certs/fullchain.pem;
    ssl_certificate_key /etc/nginx/certs/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;


    # Jenkins reverse proxy 설정
    location /jenkins/ {
        proxy_pass http://docker.host.internal:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location / {
        proxy_pass http://frontend_container:5173;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api/ {
        proxy_pass http://backend_container:8081;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

▼ 도메인 기반 Certbot 인증 키 생성

```
sudo apt update
sudo apt install certbot python3-certbot-nginx
sudo certbot --nginx -d j12e203.p.ssafy.io
```

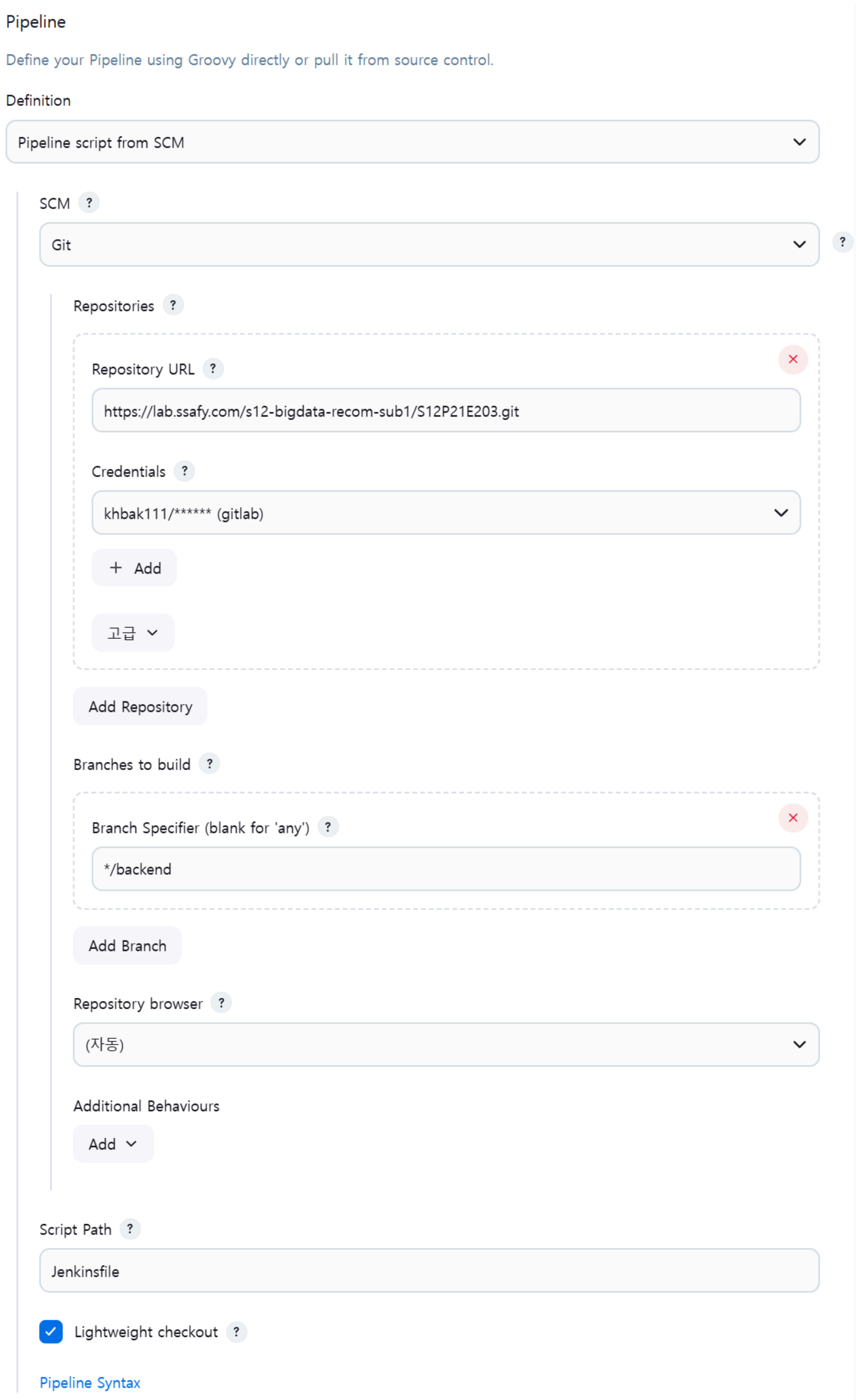
## 5. 빌드 및 배포

### 5-1. 젠킨스 컨테이너 실행

```
docker run -d \
-p 8080:8080 -p 50000:50000 \
-v jenkins_home:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
--group-add $(stat -c '%g' /var/run/docker.sock) \
-e JENKINS_REMEMBER_ME_COOKIE_KEY=$(openssl rand -hex 32) \
--name jenkins jenkins/jenkins:lts
```

## 5-2. 파이프라인 구성

1. GitLab 플러그인 설치
2. Jenkinsfile, Dockerfile 설정
3. 저장소 경로 입력 시 https://GitLabID@Repo주소.git 형태로 설정
    a. 계정 액세스 토큰으로 관리
    b. 특정 브랜치 설정
4. 프로젝트 액세스 토큰 생성 후 젠킨스 Credentials 부분에 GitLab API 토큰으로 사용
5. 위 설정을 백엔드, 프론트엔드, 추천 컨테이너에 별도로 적용해 총 3개의 파이프라인 사


Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://lab.ssafy.com/s12-bigdata-recom-sub1/S12P21E203.git

Credentials ?

khbak111/****** (gitlab)

+ Add

고급

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/backend

Add Branch

Repository browser ?

(자동)

Additional Behaviours

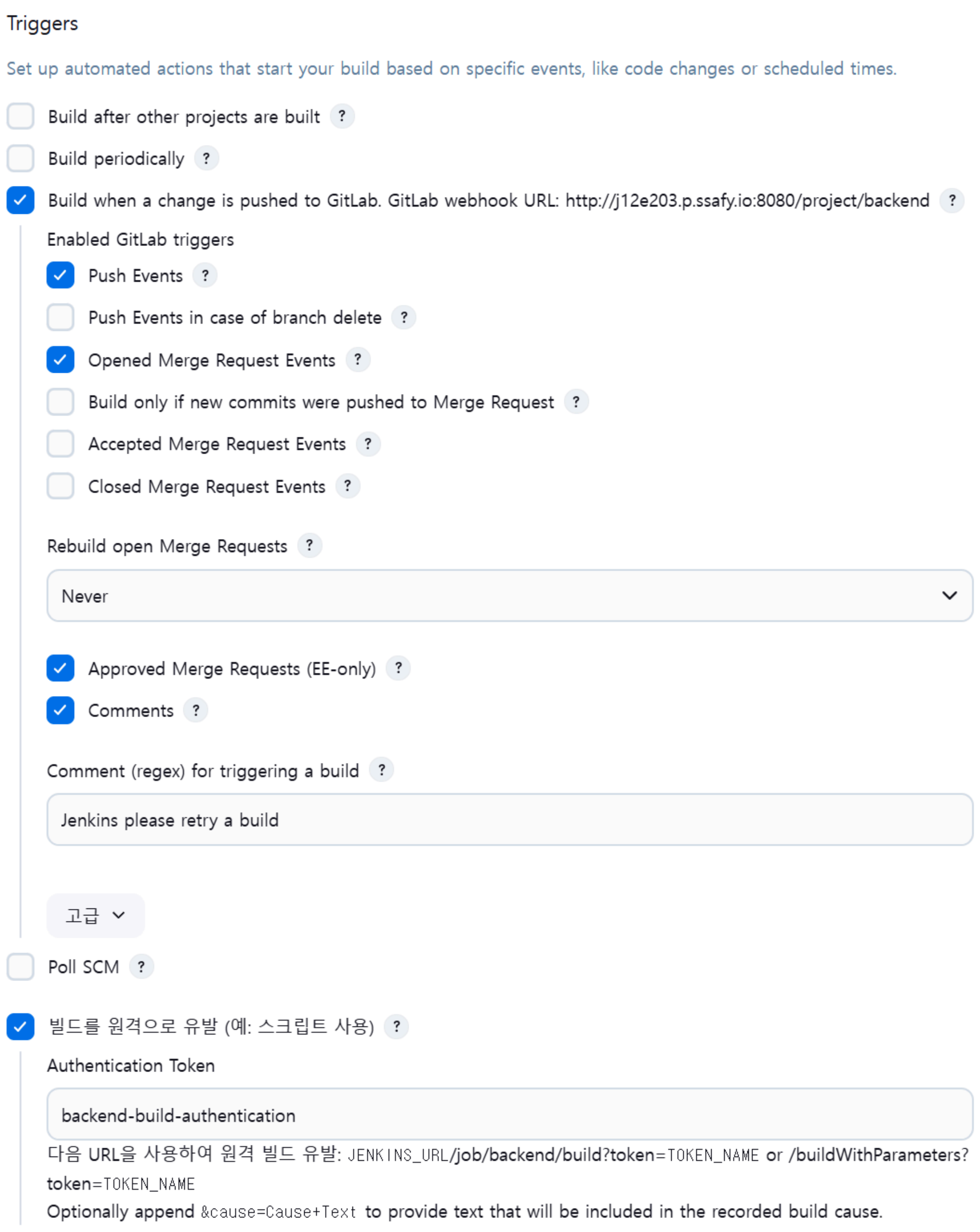
Add

Script Path ?

Jenkinsfile

✓ Lightweight checkout ?

Pipeline Syntax


Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

- [ ] Build after other projects are built ?
- [ ] Build periodically ?
- [✓] Build when a change is pushed to GitLab. GitLab webhook URL: http://j12e203.p.ssafy.io:8080/project/backend ?

  Enabled GitLab triggers
  - [✓] Push Events ?
  - [ ] Push Events in case of branch delete ?
  - [✓] Opened Merge Request Events ?
  - [ ] Build only if new commits were pushed to Merge Request ?
  - [ ] Accepted Merge Request Events ?
  - [ ] Closed Merge Request Events ?

  Rebuild open Merge Requests ?

  Never

  - [✓] Approved Merge Requests (EE-only) ?
  - [✓] Comments ?

  Comment (regex) for triggering a build ?

  Jenkins please retry a build

  고급

- [ ] Poll SCM ?
- [✓] 빌드를 원격으로 유발 (예: 스크립트 사용) ?

  Authentication Token

  backend-build-authentication

  다음 URL을 사용하여 원격 빌드 유발: `JENKINS_URL`/job/backend/build?token=`TOKEN_NAME` or /buildWithParameters?token=`TOKEN_NAME`
  Optionally append `&cause=Cause+Text` to provide text that will be included in the recorded build cause.

## 환경변수 파일(.env)

```
# 소셜 로그인 관련 환경변수

# API 기본 URL
```

```
VITE_API_BASE_URL=https://j12e203.p.ssafy.io/api

# 카카오맵 API 키
VITE_APP_KAKAO_MAP_API_KEY=2f0c32ff649b3fdadea601325ca90f62

# 카카오 OAuth 설정
VITE_KAKAO_CLIENT_ID=38c66a61f9b3af37dec7da2f800c1199
VITE_KAKAO_REDIRECT_URI=https://j12e203.p.ssafy.io/auth/kakao/callback

# 네이버 OAuth 설정
VITE_NAVER_CLIENT_ID=a4OkI4nf4Y547xepGxtu
VITE_NAVER_REDIRECT_URI=https://j12e203.p.ssafy.io/auth/naver/callback

# 네이버 임의의 STRING
VITE_NAVER_STATE_STRING=HELLOTHISISAIRDEXSPEAKING

# Vite 환경변수
VITE_APP_KAKAO_MAP_API_KEY=cc0ca3e9bb02a221293a1f7b3548ed08
VITE_API_BASE_URL=j12e203.p.ssafy.io:8081/api/v1
VITE_NAVER_CLIENT_ID=TCPhJEJuZVE0JN6kaJy3
VITE_NAVER_REDIRECT_URI=http://j12e203.p.ssafy.io:3000/auth/naver/callba
```

## 도커파일

백엔드

```
# ------------------ Build Stage ------------------
  FROM gradle:8.2-jdk17 AS builder
  WORKDIR /home/gradle/project

  # 1. 의존성 캐싱을 위해 Gradle Wrapper와 설정 파일들을 먼저 복사
  COPY --chown=gradle:gradle gradlew gradlew.bat settings.gradle build.gra
  COPY --chown=gradle:gradle gradle ./gradle

  # 2. 명시적으로 gradlew 파일에 실행 권한 부여
  RUN chmod +x gradlew

  # 3. 의존성 다운로드 (캐시 활용)
```

```
RUN ./gradlew --no-daemon dependencies

# 4. 나머지 소스 전체 복사
COPY --chown=gradle:gradle . .

# 5. 빌드 전에 다시 한 번 권한을 확인한 후 애플리케이션 빌드 (테스트 스킵)
RUN chmod +x gradlew && ./gradlew clean bootJar --no-daemon

# ----------------- Run Stage -----------------
FROM openjdk:17-jdk-slim
WORKDIR /app

# 빌드 단계에서 생성된 jar 파일만 복사하여 최종 이미지 경량화
COPY --from=builder /home/gradle/project/build/libs/*.jar app.jar

EXPOSE 8081
ENTRYPOINT ["java", "-jar", "app.jar"]
```

프론트엔드

```
# 1단계: Build Stage - Node.js 22를 사용하여 Vite 앱 빌드
FROM node:22-alpine AS builder
WORKDIR /app

# package.json 및 package-lock.json (또는 yarn.lock) 복사 후 의존성 설치
COPY package*.json ./
RUN npm install

# 환경 변수 파일을 먼저 복사 (이 부분이 중요합니다)
COPY .env ./

# 소스 코드 복사 후 빌드
COPY . .
RUN npm run build

# 2단계: Production Stage - 정적 파일을 제공하기 위해 serve 사용
FROM node:22-alpine
```

```
WORKDIR /app

# 빌드 단계에서 생성된 빌드 결과물을 복사
COPY --from=builder /app/dist ./dist

# .env 파일도 복사 (명시적으로 위치 지정)
COPY .env ./

EXPOSE 5173
# serve 패키지를 전역으로 설치하여 정적 파일 서빙
RUN npm install -g serve
CMD ["serve", "-s", "dist", "-l", "5173"]
```

파이썬 백엔드

```
FROM python:3.9-slim

# 터미널 로그가 바로 출력되도록 설정
ENV PYTHONUNBUFFERED=1

# 빌드 도구 설치 (gcc 등)
RUN apt-get update && apt-get install -y build-essential

WORKDIR /app

# 의존성 파일 복사 및 설치
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 전체 애플리케이션 코드 복사
COPY ./app /app/app

# uvicorn으로 FastAPI 앱 실행
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

## Docker-compose

백엔드

```yaml
version: '3'
services:
  be:
    build: ./BE/zeepseek
    container_name: backend_container
    ports:
      - "8081:8081"
    environment:
      - MYSQL_HOST=host.docker.internal
      - MONGO_HOST=mongodb
      - ELASTICSEARCH_HOST=elasticsearch
      - OPENAI_API_KEY=${OPENAI_API_KEY}
      - MONGODB_URI=${MONGODB_URI}
      - ES_USERNAME=${ES_USERNAME}
      - ES_PASSWORD=${ES_PASSWORD}
      # 백엔드 내에서 사용할 Redis 연결 정보:
      # 기본 캐시용 Redis (primary)
      - SPRING_REDIS_PRIMARY_HOST=property_redis
      - SPRING_REDIS_PRIMARY_PORT=6379
      # 랭킹 데이터 전용 Redis (ranking_redis)
      - SPRING_REDIS_RANKING_HOST=ranking_redis
      - SPRING_REDIS_RANKING_PORT=6378
      - KAKAO_REST_API_KEY=${KAKAO_REST_API_KEY}
    extra_hosts:
      - "host.docker.internal:host-gateway"
    restart: always
    networks:
      - e203

  property_redis:
    image: redis:7-alpine
    container_name: property_redis_container
    ports:
      - "63810:6379"   # 외부 포트 6381, 내부는 6379
    restart: always
    networks:
      - e203
  ranking_redis:
    image: redis:7-alpine
    container_name: ranking_redis_container
    ports:
      - "6382:6378"   # 외부 포트 6382, 내부는 6378
    command: ["redis-server", "--port", "6378"]
    restart: always
    networks:
      - e203


networks:
  e203:
    external: true
```

프론트엔드

```
version: '3'
services:
  be:
    build: ./BE/zeepseek
    container_name: backend_container
    ports:
      - "8081:8081"
    environment:
      - MYSQL_HOST=host.docker.internal
      - MONGO_HOST=host.docker.internal
    extra_hosts:
      - "host.docker.internal:host-gateway"
    restart: always
    networks:
      - e203

  fe:
    build: ./FE
    container_name: frontend_container
    # fe 서비스는 내부에서 5173 포트를 사용 (Vite 빌드 결과물을 serve 명령으로 서빙)
    expose:
      - "5173"
    restart: always
    networks:
      - e203

  recommend:
    build: ./ML
    container_name: recommend_container
    ports:
      - "8000:8000"
    environment:
      - MYSQL_HOST=host.docker.internal
      - MONGO_HOST=host.docker.internal
      - ES_HOST=host.docker.internal
    extra_hosts:
      - "host.docker.internal:host-gateway"
    restart: always
    networks:
      - e203

networks:
  e203:
    external: true
```

파이썬 백엔드

```
version: '3'
services:

  recommend:
    build: ./ML
    container_name: recommend_container
    ports:
      - "8000:8000"
    environment:
      - MYSQL_HOST=host.docker.internal
      - MONGO_HOST=host.docker.internal
      - ES_HOST=host.docker.internal
    extra_hosts:
      - "host.docker.internal:host-gateway"
    restart: always
    networks:
      - e203

networks:
  e203:
    external: true
```

▼ 파이썬 설정 파일

- elasticsearch.py

```
from elasticsearch import Elasticsearch

def get_es_client():
    return Elasticsearch(
        "http://elasticsearch:9200",
        basic_auth=("fastapi_user", "e203@Password!"),
        verify_certs=False
    )
```

- database.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
import os
from dotenv import load_dotenv

load_dotenv()  # .env 파일에 있는 환경 변수 로드

# 환경 변수에서 MYSQL_HOST를 읽고, 없으면 기본값(j12e203.p.ssafy.io)을 사
mysql_host = os.getenv("MYSQL_HOST", "j12e203.p.ssafy.io")

# 연결 문자열 생성 (MYSQL_HOST를 사용)
MYSQL_DATABASE_URL = os.getenv(
    "MYSQL_DATABASE_URL",
    f"mysql+pymysql://ssafy:ssafyssafy@{mysql_host}:3306/zeepseek"
```

```
)

# 엔진 생성 및 세션 설정
engine = create_engine(
    MYSQL_DATABASE_URL,
    echo=True,
    pool_size=15,        # 기본 커넥션 풀 크기 증가
    max_overflow=15,     # 추가 허용 가능한 커넥션 수
    pool_timeout=30,     # 커넥션을 기다리는 최대 시간(초)
    pool_recycle=1800    # 커넥션 재활용 시간(초, 30분)
)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=e
```

- requirements.txt

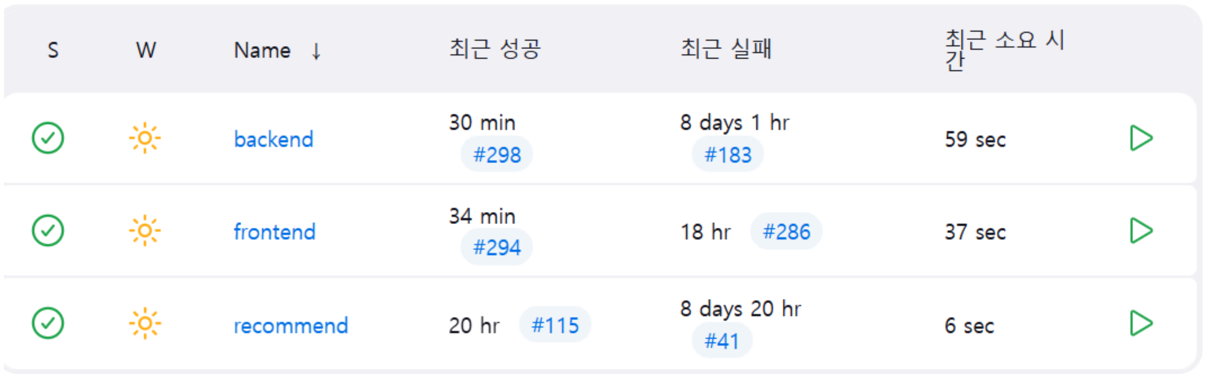
```
fastapi
uvicorn
sqlalchemy
pymysql
pymongo
python-dotenv
numpy<2
scikit-learn
cryptography
pandas
elasticsearch
scikit-surprise
requests
```

배포

- forntend ,backend,recommend 모두 Jenkinsfile 에서 docker 명령어로 배포됨
- CI/CD 파이프라인에 필요한 환경 변수 파일은 위 .env 파일을 사용
- CI/CD - 젠킨스 사용
    - 젠킨스 파일 위치
        - frotnend/Jenkinsfile

- backend/Jenkinsfile
- recommend/Jenkinsfile

◦ CI/CD에 필요 환경 변수 파일은 위와 같음



| S | W | Name ↓ | 최근 성공 | 최근 실패 | 최근 소요 시간 | |
|---|---|---|---|---|---|---|
| ✓ | ☀ | backend | 30 min #298 | 8 days 1 hr #183 | 59 sec | ▷ |
| ✓ | ☀ | frontend | 34 min #294 | 18 hr #286 | 37 sec | ▷ |
| ✓ | ☀ | recommend | 20 hr #115 | 8 days 20 hr #41 | 6 sec | ▷ |

▼ 백엔드 Jenkinsfile

```
import groovy.json.JsonOutput

pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }
        stage('Setup Environment Variables') {
            steps {
                echo "API key 환경 변수 설정 중..."
                withCredentials([file(credentialsId: 'backend_env', variable: 'ENV
                    sh 'cat $ENV_FILE > ${WORKSPACE}/.env'
                    sh 'echo "환경 변수 파일이 생성되었습니다."'
                    sh 'echo "환경 변수 파일 내용:" && cat ${WORKSPACE}/.env'
                    sh 'ls -la ${WORKSPACE}/'
                }
            }
        }
        stage('Prepare Environment') {
            steps {
```

```
            echo "네트워크 e203 확인 및 생성..."
            sh '''
              if ! docker network inspect e203 > /dev/null 2>&1; then
                echo "네트워크 e203가 존재하지 않습니다. 생성합니다."
                docker network create e203
              else
                echo "네트워크 e203가 이미 존재합니다."
              fi
            '''
          }
        }
        stage('Build & Deploy Backend') {
          steps {
            echo "Backend 서비스를 재빌드 및 재배포합니다..."
            sh 'docker-compose build --force-rm'
            sh 'docker-compose up -d --no-deps'
            script {
              if (currentBuild.number % 20 == 0) {
                echo "20회 빌드 주기 도달 - 사용하지 않는 Docker 자원 정리합니
                sh 'docker system prune -f'
              } else {
                echo "Docker 정리 건너뜁니다. (빌드 번호: ${currentBuild.num
              }
            }
          }
        }
      }
      post {
        success {
          script {
            def mattermostWebhook = 'https://meeting.ssafy.com/hooks/7w
            def payload = JsonOutput.toJson([text: "# :data: Jenkins Job < '
            sh "curl -i -X POST -H 'Content-Type: application/json' -d '${pay
          }
        }
        failure {
          script {
            def mattermostWebhook = 'https://meeting.ssafy.com/hooks/7w
```

```
                def payload = JsonOutput.toJson([text: "# :warning: :data: Jenki
                sh "curl -i -X POST -H 'Content-Type: application/json' -d '${pay
            }
        }
        always {
            echo "Backend Pipeline 완료."
        }
    }
}
```

▼ 프론트엔드 Jenkinsfile

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                // 기존 SCM 설정 사용
                checkout scm
            }
        }

        // 환경 변수 설정 -> OAuth용 credentials 추가
        stage('Setup Environment Variables') {
            steps {
                echo "OAuth 환경 변수 설정 중..."
                withCredentials([file(credentialsId: 'oauth2_env', variable: 'ENV_F
                    // 환경 변수 파일을 작업 디렉토리에 임시로 저장
                    sh 'cat $ENV_FILE > ${WORKSPACE}/.env'

                    // 환경 변수 파일이 생성되었는지 확인
                    sh 'echo "환경 변수 파일이 생성되었습니다."'

                    // 환경 변수 내용 확인 (마스킹 처리 없이)
                    sh 'echo "환경 변수 파일 내용:" && cat ${WORKSPACE}/.env'

                    sh 'ls -la ${WORKSPACE}/'
                }
            }
        }

        stage('Prepare Environment') {
            steps {
                echo "네트워크 e203 확인 및 생성..."
                sh '''
                    if ! docker network inspect e203 > /dev/null 2>&1; then
                        echo "네트워크 e203가 존재하지 않습니다. 생성합니다."
                        docker network create e203
                    else
                        echo "네트워크 e203가 이미 존재합니다."
                    fi
                '''
            }
        }
        stage('Build & Deploy Frontend & Nginx') {
            steps {
                echo "Frontend를 재빌드 및 재배포합니다..."
                sh 'docker rmi -f frontend_fe || true'  // 기존 이미지 강제 삭제 (실패ㅎ
                sh 'docker-compose build --no-cache fe' // 캐시 없이 새로 빌드
                sh 'docker-compose up -d fe'           // 컨테이너 시작
            }
        }
    }
    post {
        success {
            script {
                def mattermostWebhook = 'https://meeting.ssafy.com/hooks/7w
                def payload = """{
                    "text": "# :world_map: Jenkins Job < '${env.JOB_NAME}' > 빌
                }"""
                sh "curl -i -X POST -H 'Content-Type: application/json' -d '${pay
            }
        }
        failure {
            script {
                def mattermostWebhook = 'https://meeting.ssafy.com/hooks/7w
                def payload = """{
                    "text": "# :warning: :world_map: Jenkins Job < '${env.JOB_NA
                }"""
                sh "curl -i -X POST -H 'Content-Type: application/json' -d '${pay
            }
        }
        always {
            echo "frontend Pipeline 완료."
        }
    }
}
```

▼ 파이썬 백엔드 Jenkinsfile

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }
        stage('Prepare Environment') {
            steps {
                echo "네트워크 e203 확인 및 생성..."
                sh '''
                    if ! docker network inspect e203 > /dev/null 2>&1; then
                        echo "네트워크 e203가 존재하지 않습니다. 생성합니다."
                        docker network create e203
                    else
                        echo "네트워크 e203가 이미 존재합니다."
                    fi
                '''
            }
        }
        stage('Cleanup Old Containers') {
            steps {
                echo "기존 컨테이너 정리 중..."
                sh 'docker-compose down'
            }
        }
        stage('Build & Deploy Recommend') {
            steps {
                echo "Recommend 서비스를 재빌드 및 재배포합니다..."
                // docker-compose.yml 파일에 정의된 'recommend' 서비스만 재빌드
                sh 'docker-compose up -d --no-deps --build recommend'
            }
        }
    }
    post {
        success {
            script {
                def mattermostWebhook = 'https://meeting.ssafy.com/hooks/7w
                def payload = """{
                    "text": "# :robot_ai: Jenkins Job < '${env.JOB_NAME}' > 빌드
                }"""
                sh "curl -i -X POST -H 'Content-Type: application/json' -d '${pay
            }
        }
        failure {
            script {
                def mattermostWebhook = 'https://meeting.ssafy.com/hooks/7w
                def payload = """{
                    "text": "# :warning: :robot_ai: Jenkins Job < '${env.JOB_NAM
                }"""
                sh "curl -i -X POST -H 'Content-Type: application/json' -d '${pay
            }
        }
        always {
            echo "Recommend Pipeline 완료."
        }
    }
}
```