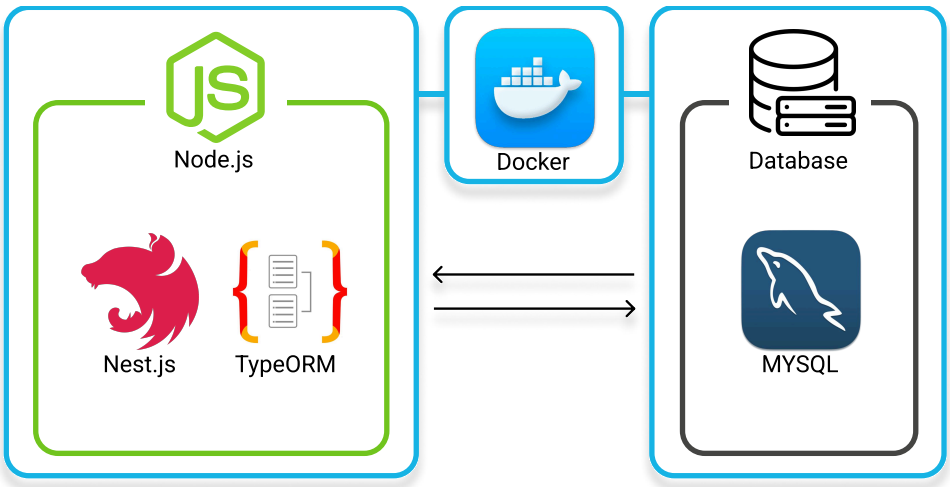
문제 1 Link

기술스텍

Docker, Node.js, NestJS, TypeORM, MySQL, bcrypt, class-validator

아키텍쳐



API Link

Method	Path	Headers	Path params	Query params	Body params
POST	/users	-	-	-	See below

/users

Body params

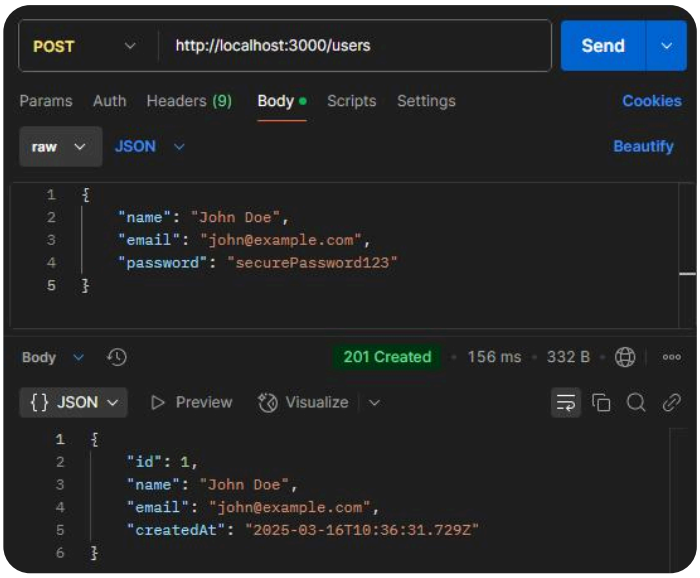
Property	Type	Description
name	string	이름 **minLength : 2** **maxLength : 50**
email	string	이메일 **email형식**
password	string	비밀번호 **minLength : 8** **숫자,문자 포함** **해쉬화 후 저장 (bcrypt)**

Request Example

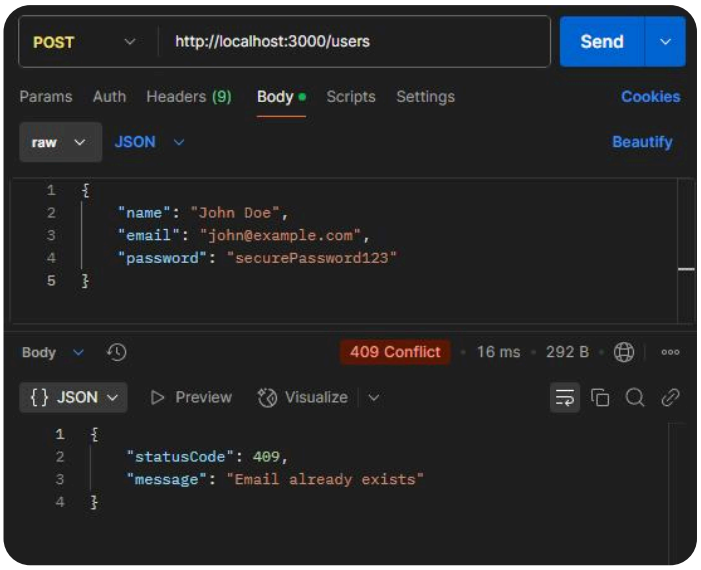
```
{
    "name": "John Doe",
    "email": "john@example.com",
    "password": "securePassword123"
}
```

테스트 및 검증 - Postman

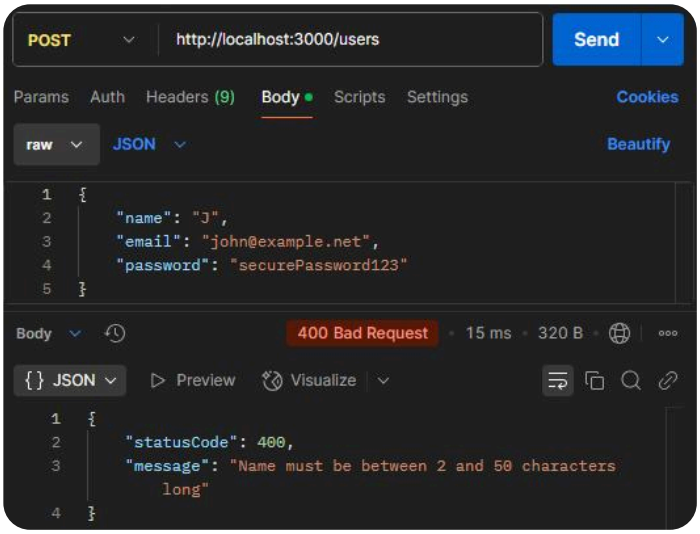
201 (success)



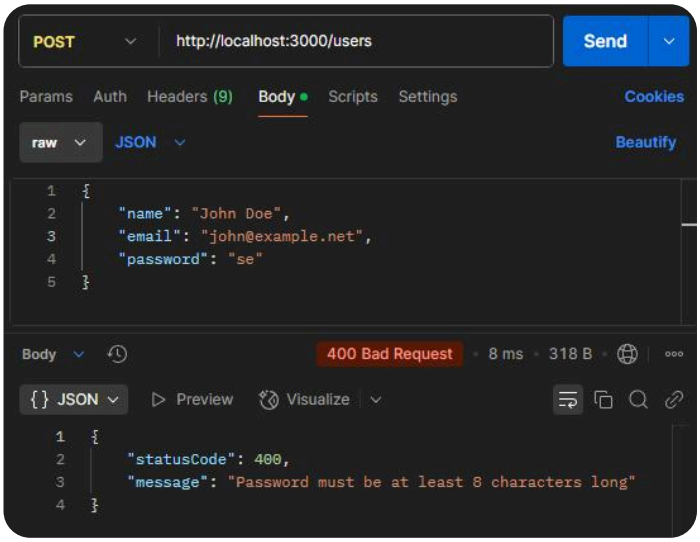
409 (email exists)



400 (name invalid)



400 (password invalid)



비밀번호의 해싱처리

데이터 유출방지

DB에 대한 공격으로 인해 패스워드가 유출 되었을 경우, 일방향으로 해싱된 기존의 비밀번호를 **탈취하기 어렵게하기위함.**

효율적 해싱처리

- Salt 사용을 통한 **해시 값의 다양화**
- bcrypt를 통한 **보안**과 어플리케이션 **성능 최적화**

문제 2 Link

테이블 구조

User Table

Column Name	Description	Data Type	Constraints
id	PK	INT	AUTO_INCREMENT
name	-	VARCHAR(255)	NOT NULL
email	UNIQ	VARCHAR(255)	NOT NULL
password	-	VARCHAR(255)	NOT NULL
created_at	-	DATETIME	DEFAULT CONVERT_TZ (CURRENT_TIMESTAMP, 'UTC', 'Asia/Seoul')
status	-	BOOLEAN	DEFAULT TRUE

Post Table

Column Name	Description	Data Type	Constraints
id	PK	INT	AUTO_INCREMENT
title	-	VARCHAR(255)	NOT NULL
content	-	TEXT	NOT NULL
created_at	-	DATETIME	DEFAULT CONVERT_TZ (CURRENT_TIMESTAMP, 'UTC', 'Asia/Seoul')
user_id	FK	INT	REFERENCES user(id)

Tag Table

Column Name	Description	Data Type	Constraints
id	PK	INT	AUTO_INCREMENT
name	UNIQ	VARCHAR(255)	NOT NULL
created_at	-	DATETIME	DEFAULT CONVERT_TZ (CURRENT_TIMESTAMP, 'UTC', 'Asia/Seoul')

PostTag Table

Column Name	Description	Data Type	Constraints
post_id	PK, FK	INT	REFERENCES post(id)
tag_id	PK, FK	INT	REFERENCES tag(id)

스키마 정의

```sql
CREATE TABLE User (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    created_at DATETIME DEFAULT CONVERT_TZ(CURRENT_TIMESTAMP, 'UTC', 'Asia/Seoul'),
    status BOOLEAN DEFAULT TRUE
);

CREATE TABLE Post (
    post_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    created_at DATETIME DEFAULT CONVERT_TZ(CURRENT_TIMESTAMP, 'UTC', 'Asia/Seoul'),
    user_id INT,
    FOREIGN KEY (user_id) REFERENCES User(user_id)
);

CREATE TABLE Tag (
    tag_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) UNIQUE NOT NULL,
    created_at DATETIME DEFAULT CONVERT_TZ(CURRENT_TIMESTAMP, 'UTC', 'Asia/Seoul'),
);

CREATE TABLE PostTag (
    post_id INT,
    tag_id INT,
    PRIMARY KEY (post_id, tag_id),
    FOREIGN KEY (post_id) REFERENCES Post(post_id),
    FOREIGN KEY (tag_id) REFERENCES Tag(tag_id)
);
```

인덱싱

```sql
CREATE INDEX idx_user_id ON Post(user_id);
CREATE INDEX idx_tag_id ON PostTag(tag_id);
```

성능개선사항

파티셔닝

DB의 사이즈가 커짐에 따라 파티셔닝을 진행
ex) Post테이블을 created_at 필드를 기준으로 파티셔닝
최근 게시물의 접근 빈도가 높으므로 **데이터 활용 패턴**에 맞춰 주기별로 파티셔닝을 수행

→ **DB 성능 향상, 데이터 보관 정책이나 아카이빙, 삭제 작업에 용이**

어플리케이션 레벨 캐싱 (redis 등)

자주 조회되거나 최신의 Post에 대해 **어플리케이션 레벨에서 캐싱**을 구현

→ **DB 성능 향상, 빠른 응답 시간, 비용절감 등**

문제 3

OAuth와 JWT ?

저는 OAuth와 JWT가 각각 다른 특성과 장점을 가지고 있습니다.
이러한 이유로, **두 가지 기술을 모두 활용하여 서비스를 개발하는게 좋다고 생각합니다.**

이유 및 장점

OAuth ?

OAuth의 가장 큰 장점은 **접근성**과 **신뢰성**이라고 생각합니다.
사용자에게 회원가입의 번거로움을 줄여주며,
네이버나 카카오와 같은 신뢰할 수 있는 국내 소셜 플랫폼을 통해
로그인 및 인증을 할 수 있게 함으로써 사용자의 신뢰를 얻을 수 있습니다.
추가적으로 인증 프로토콜을 제3자 애플리케이션이 처리함으로써 **보안성**이 향상됩니다.

JWT ?

JWT는 각 요청에 필요한 **인증 정보를 포함**하고 있어서,
서버가 사용자의 상태를 저장할 필요가 없습니다. 이로 인해 추가적인 저장소나 메모리가
필요하지 않아, **인프라 비용을 절감**할 수 있는 장점이 있습니다.

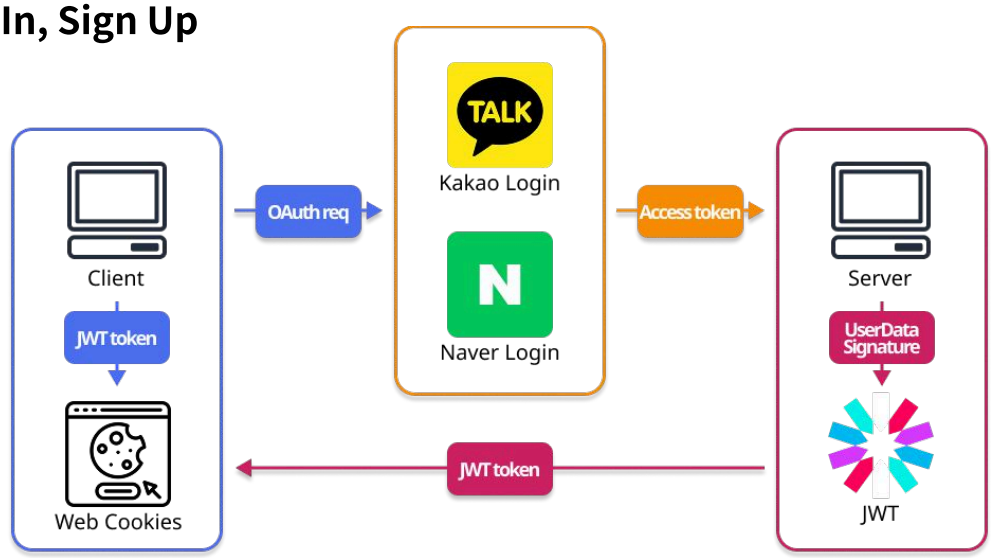
BO 서비스에서?

BO 서비스에서 **내부망**을 사용한다면 **당연히 OAuth는 사용할 수 없습니다.**
하지만 외부망과 혼용하는 서비스를 제공한다면 OAuth를 사용하지 않을 이유가 전혀 없습니다.
인증 프로토콜을 외부에 맡김으로써 **보안성이 향상**되고, **인프라 비용도 절감**할 수 있습니다.
간편하게 필요한 정보만 가져오고, 추가로 필요한 정보를 저장함으로써
가입 등의 **절차를 간소화**할 수 있습니다. 소셜 연동 기능도 적절히 추가할 수 있게 됩니다.

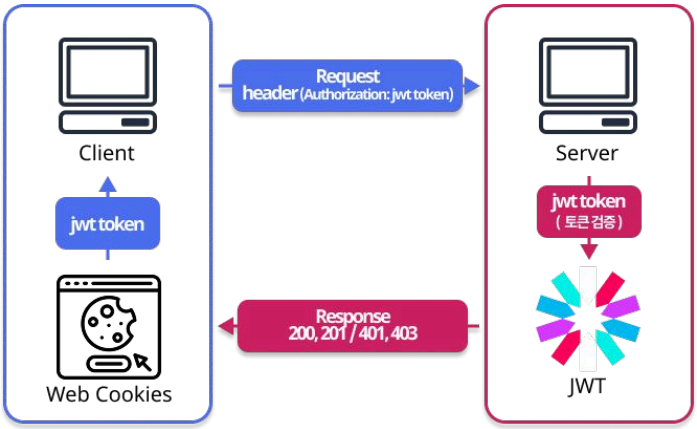
JWT는 로그인 정보를 유지하고 권한도 포함하여, **권한별로 보장하는 정보와 기능을**
구별할 수 있게 해줍니다. 따라서 JWT는 당연히 필요한 기능입니다.

구현방법 - 아키텍쳐

Sign In, Sign Up



Request, Respone



구현경험 Link

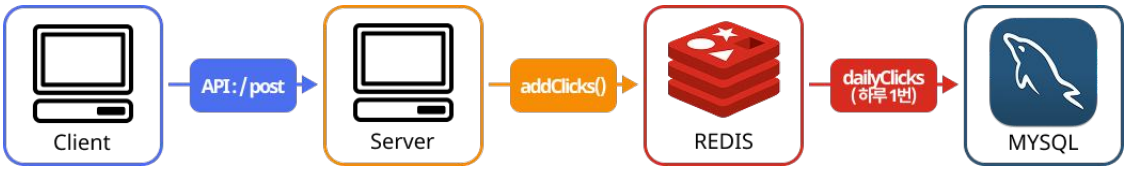
저는 개인 프로젝트에서 OAuth와 JWT를 통합한 시스템을 구현한 경험이 있습니다.
네이버와 **카카오**를 이용하여 소셜 로그인을 구현했습니다.
사용자가 처음 네이버나 카카오로 로그인할 때, **OAuth 서버에서 받은 토큰**을 해시화하여
이메일과 이름과 같은 필요한 정보와 함께 제 데이터베이스에 저장했습니다.
이후 소셜 로그인을 할 때마다 OAuth 서버에서 받은 토큰을 판단하여,
JWT에 **정보(email)과 역할(admin, customer)**를 담아
클라이언트 측에서 쿠키에 일정 시간 동안 저장하도록 했습니다.

관리자 역할이 필요한 API 요청이나 고객 정보가 필요한 API 요청에서는
헤더에 포함된 JWT를 검증하여 적절한 응답을 제공하는 시스템을 구현했습니다.
이를 통해 OAuth를 통해 안전하게 인증하고, **JWT를 통해 효율적으로 인가**를
수행하는 시스템을 성공적으로 구축할 수 있었습니다.

문제 4

아키택쳐

클릭



클라이언트가 포스트를 조회할 때마다 Redis에 실시간으로 조회수를 증가시키고, 24시간 동안 누적된 데이터를 매일 한 번 DB에 일일단위로 저장합니다.

장점
- **실시간 성능 향상** (Redis), **DB 부하 감소**

단점
- **데이터 손실 위험** (Redis의 비정상적 종료나 재시작시 데이터 손실)
  → RDB 스냅샷, AOF 로그 설정으로 데이터 손실 방지

좋아요

클라이언트가 좋아요 요청을 보낼 때마다 해당 사용자의 ID와 포스트 ID를 DB에 저장하는 방식입니다. 각 사용자는 특정 포스트에 대해 한 번만 좋아요를 누를 수 있으며, 데이터베이스에 고유 제약 조건을 설정하여 중복을 방지합니다.

장점
- **중복 클릭 및 조작 방지, 사용자 경험 개선**

단점
- **DB접근 횟수 증가** (좋아요를 할때마다 DB에 접근, 데이터의 Refresh도 필요)
  → 좋아요시 DB에 데이트를 업데이트함과 동시에 Redis에서 상태 업데이트
  추가적인 DB호출이 불필요
  WS나 SSE를 이용해서 다른 클라이언트에게 변경사항을 전송

테이블 Link

Likes Table

Column Name	Description	Data Type	Constraints
id	PK	INT	AUTO_INCREMENT
user_id	FK	INT	NOT NULL, REFERENCES user(id)
post_id	FK	INT	NOT NULL, REFERENCES post(id)
created_at	-	DATETIME	DEFAULT CONVERT_TZ (CURRENT_TIMESTAMP, 'UTC', 'Asia/Seoul')
UNIQUE	-	-	(user_id, post_id)

DailyClicks Table

Column Name	Description	Data Type	Constraints
id	PK	INT	AUTO_INCREMENT
post_id	FK	INT	NOT NULL, REFERENCES post(id)
date	-	DATE	NOT NULL
clicks	-	INT	NOT NULL
UNIQUE	-	-	(post_id, date)

스키마

SQL

```sql
CREATE TABLE Likes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    post_id INT NOT NULL,
    created_at DATETIME DEFAULT CONVERT_TZ(CURRENT_TIMESTAMP, 'UTC', 'Asia/Seoul'),
    UNIQUE (user_id, post_id),
    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (post_id) REFERENCES Post(post_id)
);

CREATE TABLE DailyClicks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    post_id INT NOT NULL,
    date DATE NOT NULL,
    clicks INT NOT NULL,
    UNIQUE (post_id, date),
    FOREIGN KEY (post_id) REFERENCES Post(post_id)
);
```

클릭 수 조작 방지 (클라이언트)

쿠키 및 로컬 스토리지

- 사용자가 포스트를 조회할 때, 쿠키나 로컬 스토리지에 해당 정보를 저장하고, 일정 시간 내에 동일한 포스트에 대한 추가 조회를 무시합니다.

디바운싱

- 포스트 클릭 이후 일정 시간 내에 추가적인 클릭의 발생을 무시합니다.

웹 보안 솔루션 (Cloudflare, CAPTCHA)

- 비정상적인 조회 패턴이 감지시 악의적인 봇 트래픽이나, 공격을 방지합니다.

클릭 수 조작 방지 (서버)

세션 및 사용자 ID 기반 제한

- 동일한 사용자의 ID로 짧은 시간 동안 비정상적으로 많은 클릭 요청이 발생하면 이를 무시하거나 응답하지 않게 설정합니다.

IP 주소 기반 제한

- 동일한 사용자의 IP로 짧은 시간 동안 비정상적으로 많은 클릭 요청이 발생하면 이를 무시하거나 응답하지 않게 설정합니다.

로그 및 모니터링

- 클릭 이벤트를 로그에 기록, 특정 사용자가 비정상적으로 많은 클릭을 생성하는 경우 이를 감지하여 차단합니다.