Node.js와 MongoDB II

04 Express.js와 MongoDB로 웹서비스 만들기 3

/* elice */

목차

🎯 수강목표

1. JWT와 Cookie의 이해와 사용

JWT와 Cookie의 개념에 대해 이해하고
프로젝트에 이를 적용하는 방법에 대해 학습한다.

2. SMTP와 이메일 발송기능 사용

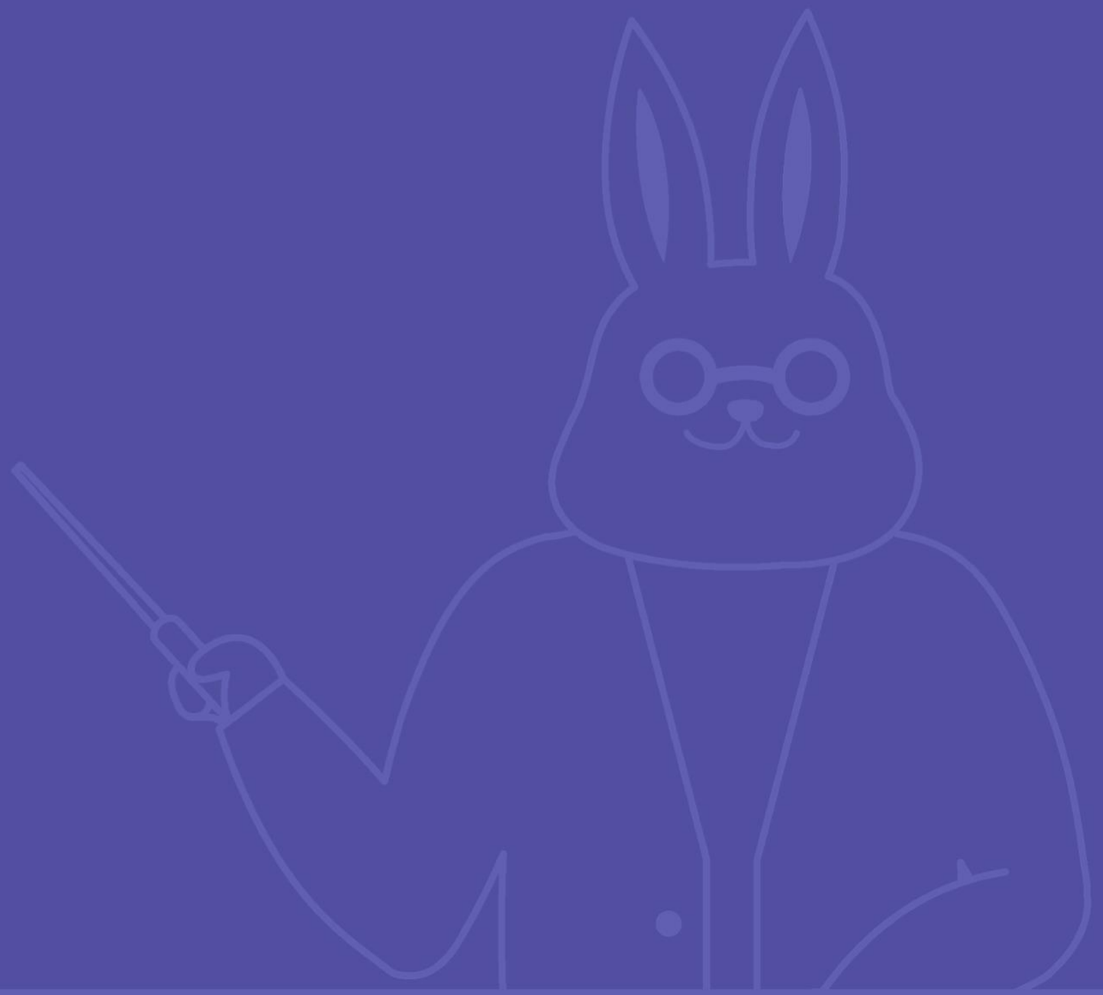
네이버 SMTP 서버와 Nodemailer를 사용하여 이메일을 발송하는 방법에 대해 학습한다.

3. OAuth 2.0의 이해와 사용

OAuth 2.0에 대해 이해하고
Google 로그인을 사용하는 방법에 대해 학습한다.

01

JWT의 이해



JWT란?

JSON **W**eb **T**oken
인증을 위한 정보를 특별한 **저장소를 이용하지 않고,**
전자 서명을 이용하여 확인하는 방법

JWT의 구성

header - 토큰의 타입 (jwt), 데이터 서명 방식
payload - 전달되는 데이터
signature - 헤더와 페이로드의 전자서명

JWT의 구성

JWT 는 **Web Token,** 데이터를 **웹에서 사용하기 위한 스펙**이므로
웹에서 문제없이 사용할 수 있는 문자열로만 구성된 **base64 인코딩**을 사용

JWT 예시

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImVsaWNlQGVsaWNlLmlvIiwibmFtZSI6ImVsaWNlIn0.PdHPPLywm5yjWHyMYcV_W4HG5SX62DzoQxC0G2FuDE0
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "email": "elice@elice.io",
  "name": "elice"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) ☐ secret base64 encoded
```

JWT 홈페이지에서 실행해 볼 수 있는 JWT Encode/Decode

JWT와 보안

JWT의 payload는 단순히 정보를 base64 encode
→ **decode 시 정보 노출**됨 → **민감한 정보는 제외하고 토큰 생성** 필수

서버는 JWT를 생성할 때, **비공개키를 이용하여 서명**을 함
payload를 조작할 경우 **서명이 일치하지 않기 때문에 인증 실패**

JWT 작동 방식

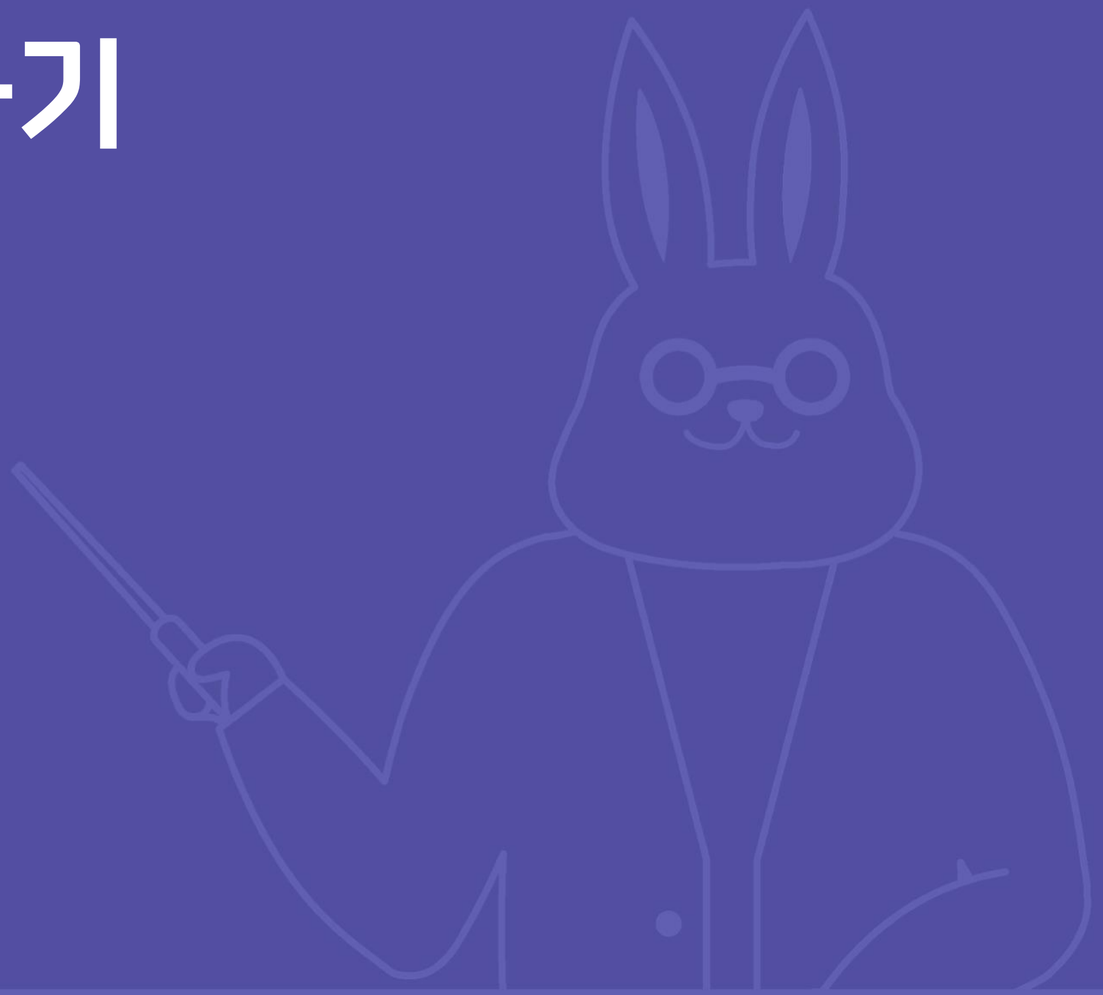
1. 사용자 로그인
2. 서버는 로그인된 유저 정보를 **JWT로 생성**하여 **클라이언트에 전달**
3. 클라이언트는 전달받은 **JWT를 이용**하여 **인증이 필요한 요청에 사용**

JWT 사용 이유

session은 기본적으로 **웹 브라우저의 통신 스펙**
모바일 앱 등, **웹 브라우저가 아닌** 어플리케이션의 경우 이를 활용하기 **부적합함**
JWT를 사용하면 어느 클라이언트에서나 **동일한 방식의 사용자 인증을 구현** 가능

02

JWT + Cookie 사용하기



Cookie란?

웹 서비스에서 사용하는 정보를 **클라이언트에 저장**하고,
HTTP 요청 시 이를 **함께 전송**하여,
클라이언트 정보를 **서버에 전달**하는 기술

Session vs Cookie

Session

클라이언트 정보를 **서버 측 저장소**에 저장하고 사용

Cookie

클라이언트 정보를 **클라이언트 (브라우저)**에 저장하고 사용

JWT + Cookie

Session을 사용한 유저 로그인의 경우
Cookie에 Session ID 저장 → **Session Store**에서 유저 정보 가져오기

JWT를 쿠키에 저장하는 경우
JWT로 요청 → **서명 확인** 후 유저 정보 사용

데이터베이스 접근이 줄어서 효율적인 인증 구현 가능

JWT 로그인 구현하기

1. 기존 **세션**으로 구현된 로그인을 **비활성화**
2. 로그인 로직에서 **JWT 생성** 후 **쿠키로 전달**
3. passport-jwt 패키지로 **JWT 로그인 미들웨어** 작성 및 사용

JWT 로그인 구현하기

code

```
// app.use(session(...));
// app.use(passport.session());
```

세션 비활성화하기

express-session 패키지 비활성화

passport.session 기능 비활성화

로그인 로직에 JWT 토큰 생성 및 쿠키 전달

code

```
setUserToken = (res, user) => {
  const token = jwt.sign(user, secret);
  res.cookie('token', token);
}

---
router.post('/', passport.authenticate('local'),
  (req, res, next) => {
    setUserToken(res, req.user);

  res.redirect('/');
});
```

res.cookie 함수 사용하여 token을 클라이언트에 **쿠키로 전달**

passport-jwt 사용하기

code

```
const JwtStrategy = require('passport-jwt').Strategy;
const cookieExtractor = (req) => {
 const { token } = req.cookies;

 return token;
};
const opts = {
 secretOrKey: secret,
 jwtFromRequest: cookieExtractor,
}
module.exports = new JwtStrategy(opts, (user, done) => {
 done(null, user);
});

---
passport.use(jwt);
```

passport-jwt 패키지를 이용해

요청된 **JWT 토큰의 서명을 확인**하고
인증하는 기능을 구현

JWT 미들웨어 추가

jwt middleware

```
app.use((req, res, next) => {
  if (!req.cookies.token) {
    next();
    return;
  }

 return passport.authenticate('jwt')(req,
res, next);
});
```

JWT 토큰은 기본적으로 **모든 요청에 포함**

요청에 **토큰이 있는 경우**
로그인된 상태로 처리하기 위해

모든 요청에 공통적으로 적용할 수 있는
미들웨어로 JWT 로그인을 추가

로그아웃

jwt logout

```
res.cookie('token', null, {
  maxAge: 0,
});
```

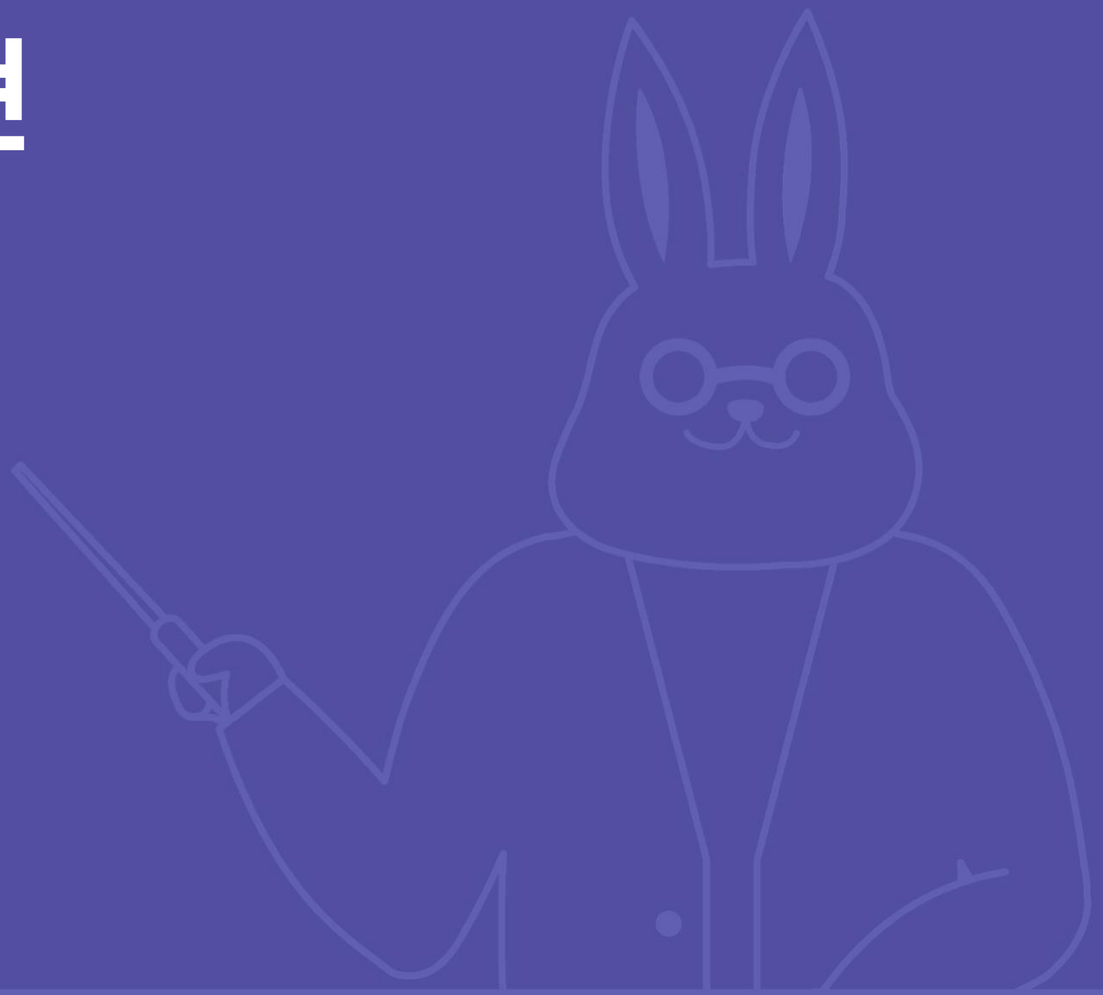
로그아웃은 간단하게 **클라이언트 쿠키를 삭제**하여 처리 가능

token 값을 null로 전달하는 것과 함께,

cookie의 만료 시간을 0으로 설정하여
클라이언트가 쿠키를 바로 만료시키도록 전달

03

회원 비밀번호 찾기 구현



회원 비밀번호 찾기 흐름 설명

1. **임의의 문자열**로 비밀번호 초기화
2. 초기화된 문자열을 **메일로 전달** → 메일 발송기능 개발 필요
3. 초기화 후 첫 로그인 시 **비밀번호 변경 요청**

메일 발송기능 구현 방법

SMTP 서버 이용

- 네이버 구글 등의 메일서버를 이용하여 무료로 발송 가능
- 메일 발송 및 관리 기능 직접 개발 필요

메일 발송 서비스 이용 (Mailgun, Sendgrid ...)

- 메일 발송 api 제공 및 관리용 웹페이지 제공
- 사용량에 따라 유료 과금

SMTP란?

Simple **M**ail **T**ransfer **P**rotocol

메일 전송을 위한 표준 규약

SMTP 서버란 표준 규약을 통해 **메일을 전송하는 기능을 구현**한 서버

Node.js에서 메일 발송하기

Nodemailer 패키지를 사용하여 **SMTP 서버를 통해 메일을 발송**할 수 있음
SMTP 서버를 직접 **만들고 운영하는 것은 비효율적**
메일 기능을 제공하는 서비스 제공자들은 SMTP 서버를 사용할 수 있게 제공함
ex) Gmail, Naver

Nodemailer + Gmail 사용하기

Nodemailer 에서 Gmail을 사용하기 위해서는 **앱 비밀번호 설정**이 필요
구글 계정설정 → 보안 → 앱 비밀번호 추가

생성된 앱 비밀번호는 **다시 확인할 수 없으므로 기록 필수**

Nodemailer + Gmail 사용하기

nodemailer

```
const nodemailer = require('nodemailer');

const transport = nodemailer
  .createTransport({
    service: 'Gmail',
    auth: {
      user: "google account",
      pass: "app password",
    },
  });

...
```

```
...
const message = {
  from: "login account",
  to: "mail address",
  subject: "title",
  text: "message"
};

transport.sendMail(message, (err, info) => {
  if (err) {
    console.error('err', err);
    return;
  }

  console.log('ok', info);
})
```

비밀번호 초기화 기능 개발

랜덤 패스워드

```
function generateRandomPassword() {
  return Math.floor(
    Math.random() * (10 ** 8)
  ).toString().padStart('0', 8);
}
---
router.post('/reset-password', asyncHandler(... => {
  const { email } = req.body;
  const randomPassword = generateRandomPassword();
  await User.findOneAndUpdate({ email }, {
    password: getHash(randomPassword),
  });

  await sendEmail(email, '...', randomPassword);
  res.redirect('/');
}));
```

generateRandomPassword()는
임의의 문자열을 만들어주는 함수

email을 받아서 **생성된 임의의 문자열**로
사용자의 비밀번호 초기화 후,

초기화한 **비밀번호를 메일로 발송**

초기화 후 로그인 시 비밀번호 변경 요청

비밀번호 변경

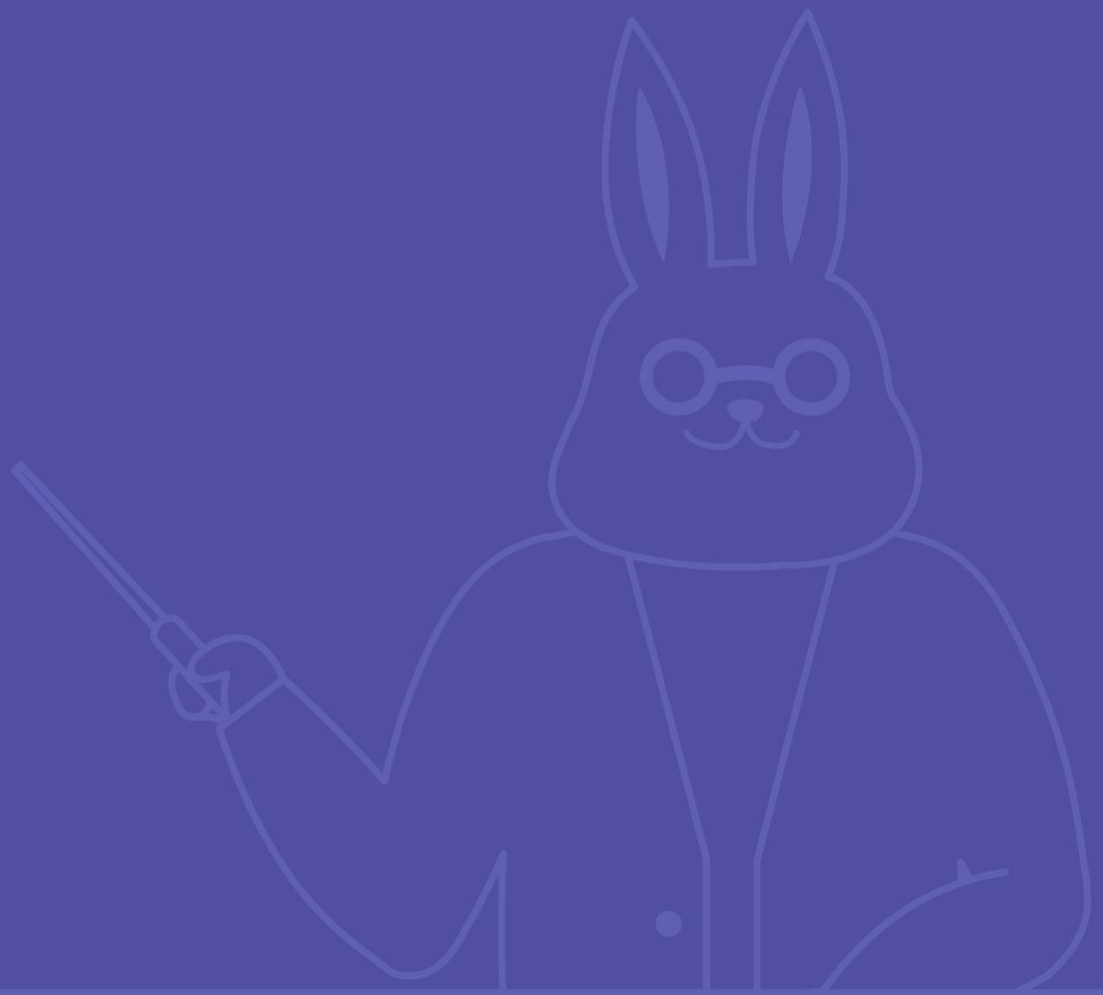
```
const UserSchema = ...
  passwordReset: {
    type: Boolean,
    default: false,
  }
...

---

router.post('/reset-password', ...
  await User.findOneAndUpdate({
    ...
    passwordReset: true,
  });
```

비밀번호 변경 요청

```
function checkPasswordReset(req, res, next) {
  if (req.user && req.user.passwordReset) {
    res.redirect('/update-password');
    return;
  }

  next();
}

---

router.post('/update-password', ...
  await User.findOneAndUpdate({
    ...
    passwordReset: false,
  });
```

04

OAuth의 이해



OAuth란?

Open **Auth**orization
서비스 제공자가 **다른 서비스에게 데이터를 제공하기 위해**
서비스 사용자에게 제공하는 사용자 **인증방식의 표준**

OAuth 동작 방식

1. 서비스 제공자에게 **인증 요청**
2. 인증 완료 후 사용자 정보를 **요청한 서비스로 전달**
3. 인증 정보를 이용해 **서비스 제공자의 데이터 사용**

OAuth 사용 예시

구글 캘린더 연동 서비스

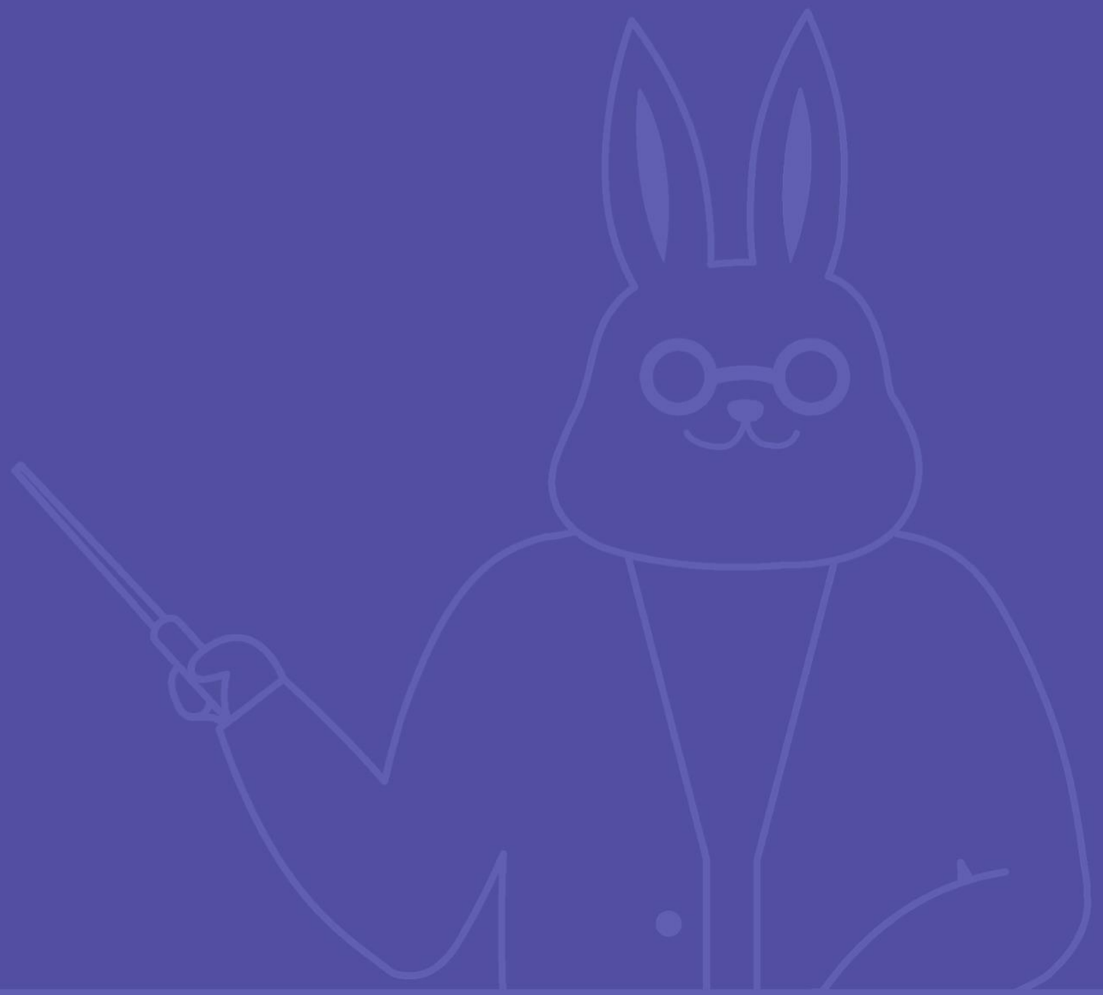
1. 구글 OAuth 인증 요청
2. 인증된 OAuth Token을 기록
3. OAuth Token을 사용하여 구글 캘린더 API 사용

OAuth와 로그인

OAuth는 **사용자의 인증을 제공**하는 표준
이를 활용하여, 로그인 기능을 **간편하게 구성**할 수 있음
웹 서비스 제공자는 아이디, 비밀번호 로그인을 **구현할 필요 없음**
웹 서비스 사용자는 로그인 시 아이디, 비밀번호를 **입력할 필요 없음**

05

구글 로그인 구현하기



구글 로그인 구현 순서

- 구글 클라우드 플랫폼 프로젝트 생성
- API 및 서비스 → OAuth 동의화면 설정
- 사용자 인증정보 → OAuth 클라이언트 ID 만들기
- passport-google-oauth20 연동

passport-google-oauth20

passport-strategy 인터페이스의 **구글 로그인 구현체**
OAuth 인증을 구현하기 위해서는 인증 요청, 데이터 수신 등의 **복잡한 작업 필요**

passport-google-oauth20 는 **손쉽게 구글 OAuth 2.0을 구현**해 주는 패키지

passport-google-oauth20 작성

google strategy

```
const GoogleStrategy =
  require('passport-google-oauth20').Strategy;

const config = {
  clientID: 'clientID',
  clientSecret: 'clientSecret',
  callbackURL: 'callbackUrl',
};

...
new GoogleStrategy(config,
  (accessToken, refreshToken, profile, done) => {
    const { email, name } = profile._json;
    ..
    // create or update user
```

구글 로그인이 완료된 후 **결과를 전달받는 부분**

OAuth 클라이언트 설정값 및 완료 결과를 전달받을 **callbackURL**을 config로 설정

accessToken, refreshToken은
다른 구글 API들을 사용하기 위한 토큰
(본 프로젝트에서는 사용하지 않음)

profile은 전달받은 유저 정보. 이를 이용해 유저를 생성하거나 OAuth 정보를 업데이트 함

passport-google-oauth20 사용

authenticate

```
passport.use(google);

---

router.get('/google',
  passport.authenticate('google', {
    scope: ['profile', 'email']
  }));

router.get('/google/callback',
  passport.authenticate('google', {
    failureRedirect: '/login'
  }), (req, res, next) => {

  res.redirect('/');
});
```

/auth/google 접근 시 자동으로 **구글 로그인 페이지**로 넘어감

로그인 완료 후 로그인 정보를 **/auth/google/callback으로 전달**해 줌 (config에 설정된 주소)

전달받은 데이터는 **strategy에서 처리**

처리가 완료되면 **request handler 실행**

passport-google-oauth20 사용

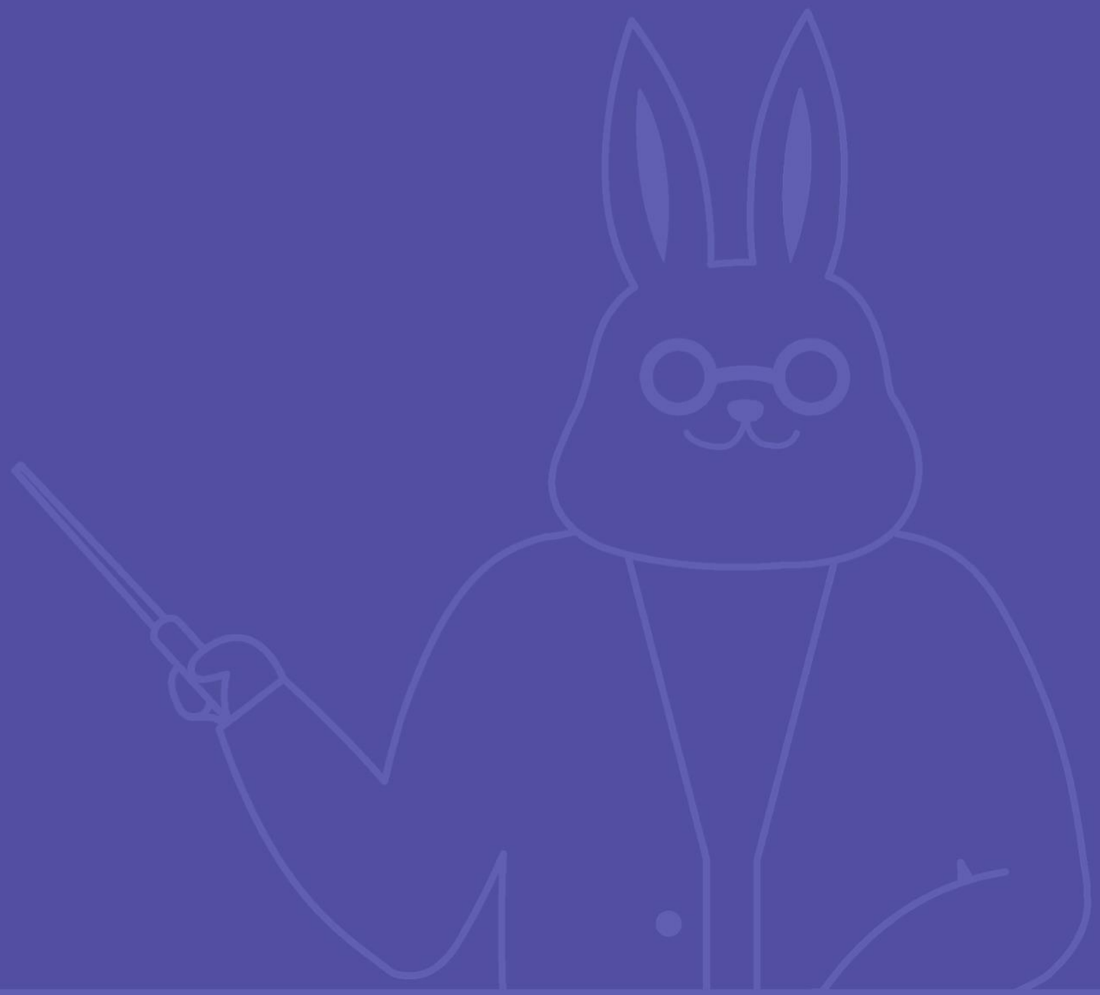
login.pug

```
...
td: a(href="/auth/google") 구글로 로그인하기
```

/auth/google로 link 시
passport가 자동으로
구글 로그인 페이지로 이동 시켜 주고,
로그인 결과를 /auth/google/callback으로
전달해 줌

06

추가 - Nginx 사용하기



/* elice */

Nginx란?

최근 신규 프로젝트에서 **가장 많이 채택**되고 있는 **웹 서버 소프트웨어**
웹 서버 소프트웨어란, **HTTP 요청**을 받아
파일이나 프로그램 실행 결과를 **HTTP 응답**으로 보내주는 소프트웨어

Nginx를 사용하는 이유

Java - Tomcat, PHP - fastcgi 등
다른 언어가 **HTTP 요청을 처리를 위한 의존성**이 있는 것에 반해,
Node.js 는 기본적으로 **HTTP 요청을 수신하고, 응답**하는 기능이 이미 있음.
→ 웹 서버 소프트웨어 없이도 **스스로 동작 가능**

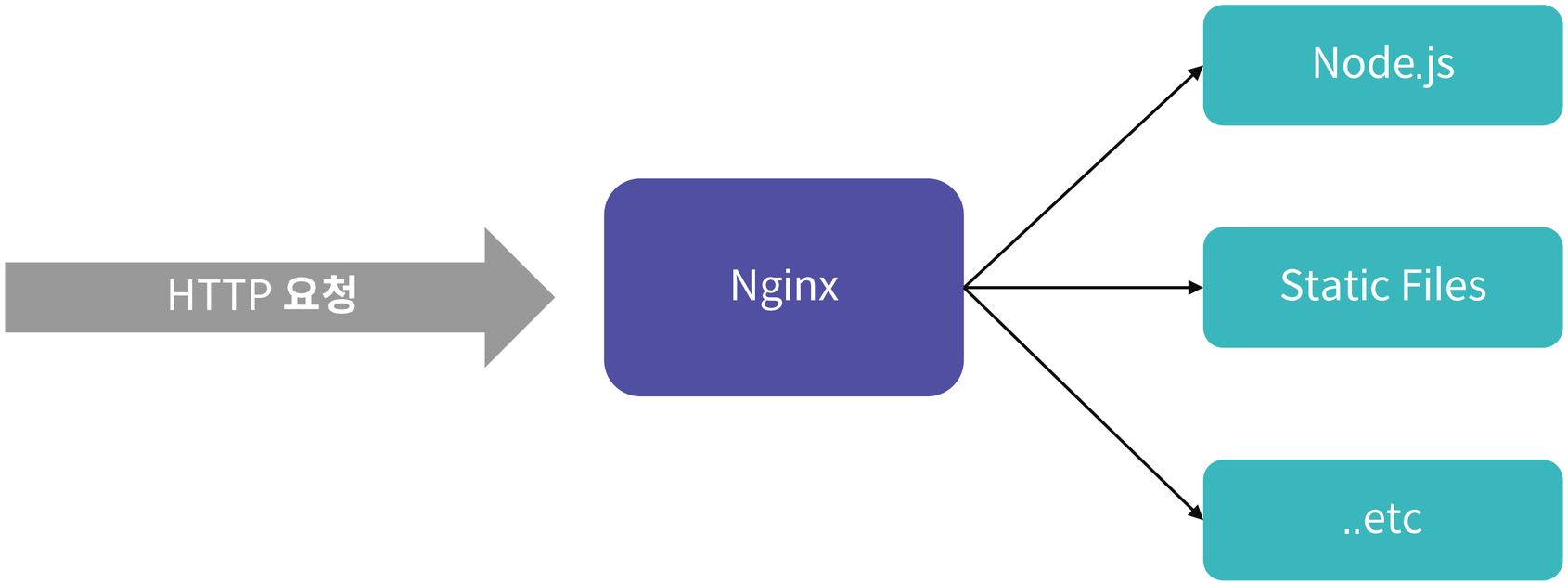
Nginx를 사용하는 이유

하지만, **HTTPS**, **도메인 연결**, **static file caching 등**의 기능을 사용하기 위해
Nginx 같은 **웹 서버 소프트웨어는 필수**
→ Node.js 단독으로 **production-level 서비스를 구축할 수는 없음**

Nginx + Node.js

Nginx의 **reverse-proxy** 기능을 사용해, Node.js와 Nginx를 연결할 수 있음
reverse-proxy는, **HTTP 요청을 다른 서버에 전달**하는 기능
Nginx 가 HTTP 요청을 받아, 설정된 내용에 해당하는 요청만 **Node.js로 전달**

Nginx + Node.js



Nginx + Node.js 설정 파일 예시

code

```
server {
 listen 80;
 server_name www.example.com;

 location / {
  proxy_pass http://localhost:3000;
  proxy_http_version 1.1;
 }
}
```

http://www.example.com으로 접속한
모든 요청을 localhost:3000으로 전달하는
설정 파일

HTTPS, file caching들의 작업은 Nginx의
설정 방법을 참고하여 추가 가능

https://www.nginx.com/resources/wiki/start/topics/examples/full/

크레딧

/* elice */

코스 매니저
이재성

콘텐츠 제작자
최규범

강사
최규범

감수자
최규범

디자이너
강혜정

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io