Node.js 와 MongoDB I

04 Express.js 와 REST API

/* elice */

목차

🎯 수강목표

1. Express.js의 Middleware 이해하기

Express.js의 핵심기능인 Middleware에 대해 이해하고
Middleware의 작성과 사용법을 학습한다.

2. REST API 이해하기

REST API란 무엇인지에 대해 이해하고
REST API를 구성하는 방법에 대해 학습한다.

3. REST API 작성과 테스트

지금까지 학습한 내용을 바탕으로
Express.js를 이용하여 간단한 메모 API를 작성해 보고
Postman을 사용하여 API를 테스트해 본다.

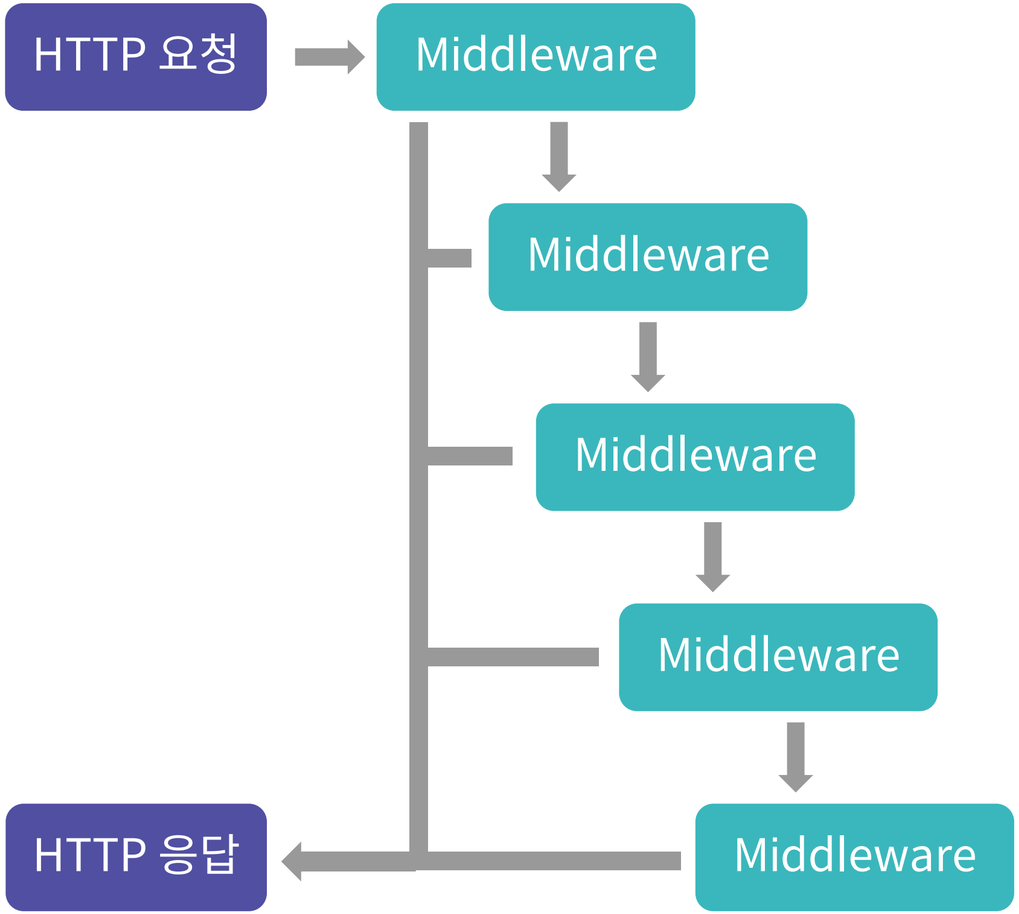
01

Express.js의 Middleware

Middleware란?

미들웨어는 Express.js **동작의 핵심**
HTTP 요청과 응답 사이에서 **단계별 동작을 수행해주는 함수**

Middleware 동작 원리



Express.js의 미들웨어는 HTTP 요청이 들어온 순간부터 시작이 됨

미들웨어는 **HTTP 요청과 응답 객체를 처리**하거나, **다음 미들웨어를 실행**할 수 있음

HTTP 응답이 마무리될 때까지 미들웨어 동작 사이클이 실행됨

02

Middleware의 작성과 사용

middleware 작성법

req, res, next를 가진 함수를 작성하면 해당 함수는 미들웨어로 동작할 수 있음

- ✓ req는 HTTP 요청을 처리하는 객체
- ✓ res는 HTTP 응답을 처리하는 객체
- ✓ next는 다음 미들웨어를 실행하는 함수

Route Handler와 middleware

Route Handler도 **미들웨어의 한 종류**
Route Handler는 **라우팅 함수(get, post, put, delete 등)에 적용된** 미들웨어
일반적인 미들웨어와는 다르게 **path parameter를 사용**할 수 있음

middleware 작성법

middleware-examples

```
const logger = (req, res, next) => {
  console.log(`Request ${req.path}`);
  next();
}

const auth = (req, res, next) => {
  if (!isAdmin(req)) {
    next(new Error('Not Authorized'));
    return;
  }
  next();
}
```

req, res, next를 인자로 갖는 함수를 작성하면 미들웨어가 됨

req, res 객체를 통해 HTTP 요청과 응답을 처리하거나
next 함수를 통해 다음 미들웨어를 호출해야 함

next() 함수가 호출되지 않으면
미들웨어 사이클이 멈추기 때문에 주의

middleware 사용법

middleware 는 적용되는 위치에 따라서
어플리케이션 미들웨어, 라우터 미들웨어, 오류처리 미들웨어로 분류 가능
필요한 동작 방식에 따라 미들웨어를 **적용할 위치**를 결정

middleware 사용법 - 어플리케이션 미들웨어

application middleware

```
app.use((req, res, next) => {
  console.log(`Request ${req.path}`);
  next(); 1
});

app.use(auth); 2

app.get('/', (req, res, next) => {
   res.send('Hello Express'); 3
});
```

use 나 **http method** 함수를 사용하여
미들웨어를 연결할 수 있음

미들웨어를 모든 요청에 공통적으로 적용하기
위한 방법

HTTP 요청이 들어온 순간부터 적용된
순서대로 동작 함

middleware 사용법 - 라우터 미들웨어

router middleware

```
router.use(auth); 3

router.get('/', (req, res, next) => {
  res.send('Hello Router');
}); 4

app.use((req, res, next) => {
  console.log(`Request ${req.path}`);
  next(); 1
});

app.use('/admin', router); 2
```

router 객체에 미들웨어가 적용되는 것 외에는 어플리케이션 미들웨어와 사용 방법은 동일

특정 경로의 라우팅에만 미들웨어를 적용하기 위한 방법

app 객체에 라우터가 적용된 이후로 순서대로 동작함

middleware 사용법 - 미들웨어 서브 스택

middleware sub-stack

```
app.use(middleware1, middlware2, ...);

app.use('/admin', auth, adminRouter);

app.get('/', logger, (req, res, next) => {
  res.send('Hello Express');
});
```

use 나 http method 함수에 **여러 개의 미들웨어를 동시에 적용**할 수 있음

주로 한 개의 경로에 특정해서 미들웨어를 적용하기 위해 사용

전달된 인자의 순서 순으로 동작

오류처리 미들웨어

오류처리 미들웨어는 **일반적으로 가장 마지막에 위치**하는 미들웨어
다른 미들웨어들과는 달리 **err, req, res, next 네 가지 인자**를 가지며,
앞선 미들웨어에서 **next 함수에 인자가 전달되면 실행**됨

오류처리 미들웨어

error handling middleware

```
app.use((req, res, next) => {
  if (!isAdmin(req)) {
    next(new Error('Not Authorized')); 1
    return;
  }
  next();
});

app.get('/', (req, res, next) => {
    res.send('Hello Express');
});

app.use((err, req, res, next) => { 2
  res.send('Error Occurred');
});
```

가장 아래 적용된 err, req, res, next를 인자로 갖는 함수가 오류처리 미들웨어

이전에 적용된 미들웨어 중 next에 인자를 넘기는 경우 중간 미들웨어들은 뛰어넘고

오류처리 미들웨어가 바로 실행됨

함수형 middleware

하나의 미들웨어를 작성하고, **작동 모드를 선택하면서 사용**하고 싶을 경우
미들웨어를 함수형으로 작성하여 사용할 수 있음

Ex) API별로 사용자의 권한을 다르게 제한하고 싶은 경우

함수형 middleware

함수형 미들웨어

```
const auth = (memberType) => {
  return (req, res, next) => {
    if (!checkMember(req, memberType)) {
      next(new Error(`member not ${memberType}`));
      return;
    }
    next();
  }
}

app.use('/admin', auth('admin'), adminRouter);

app.use('/users', auth('member'), userRouter);
```

auth 함수는 **미들웨어 함수를 반환하는 함수**

auth **함수 실행 시 미들웨어의 동작이 결정**되는 방식으로 작성됨

일반적으로 **동일한 로직에 설정값만 다르게** 미들웨어를 사용하고 싶을 경우에 활용됨

Middleware Libraries

Express.js는 다양한 미들웨어들이 **이미 만들어져 라이브러리로 제공**됨
유용한 미들웨어를 npm 을 통해 추가하여 사용할 수 있음
Express.js 홈페이지나 **npm 온라인 저장소**에서 찾아볼 수 있음

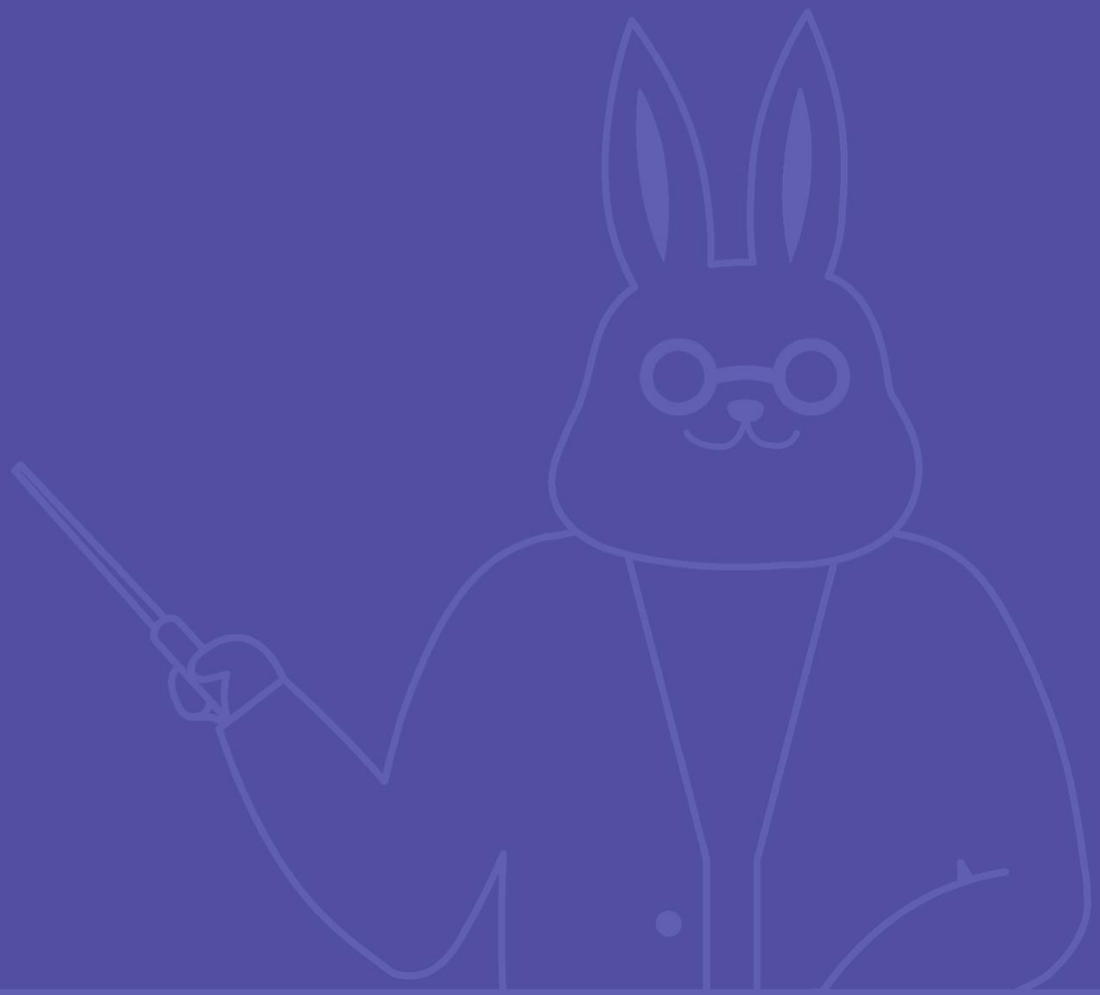
Ex) cors, multer, passport 등

Middleware 요약

미들웨어는 **HTTP 요청과 응답 사이**에서 동작하는 **함수**
req, res, next를 인자로 갖는 함수는 미들웨어로 동작할 수 있음
app 혹은 router 객체에 연결해서 사용 가능
next에 인자를 넘기는 경우 **오류처리 미들웨어**가 실행됨
미들웨어에 값을 설정하고 싶은 경우는 **함수형 미들웨어**로 작성 가능

03

REST API

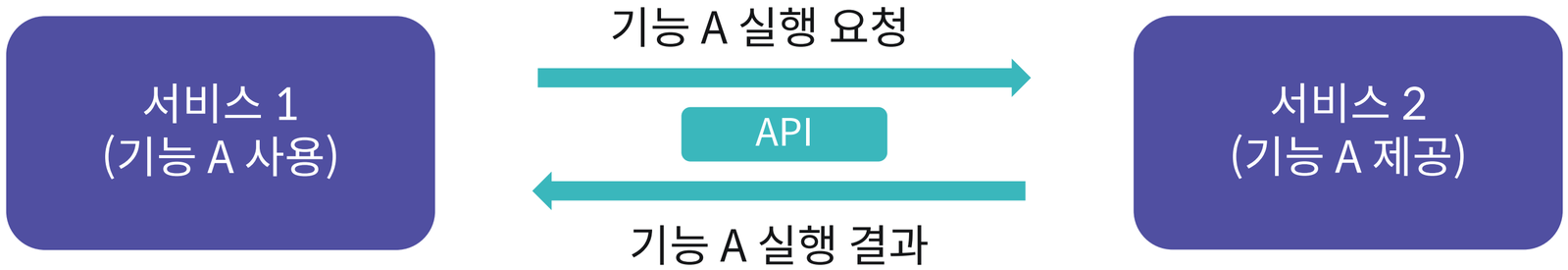


REST API란?

REST + **API**

REST 아키텍처를 준수하는 **웹 API**
RESTful API라고 부르기도 함

API란?



Application **P**rogramming **I**nterface

서비스나 프로그램 간에 **미리 정해진 기능**을 **실행 할 수 있도록 하는 규약**

운영체제 API, 프로그램언어 API, 웹 API 등이 있음

REST 란?

REpresentational **S**tate **T**ransfer
웹에서 **자료를 전송**하기 위한 **표현 방법**에 대한 아키텍처
REST를 정확하게 구현하기 위해선 **많은 제한조건**이 있지만,
기본적인 **REST 가이드**를 따르면 조금 더 **좋은 구조의 API를 구성**할 수 있음

REST API 기본 가이드 - HTTP Method의 사용

REST API는 API의 동작을 **HTTP method + 명사형 URL**로 표현함
/posts 라는 URL은 **'게시글'이라는 자원**을 가리킨다고 할 때,
GET- 가져오기, **POST** - 새로 만들기, **PUT** - 수정하기, **DELETE** - 삭제하기 의
HTTP method와 결합하여 **API 동작을 정의**하여야 함

REST API 기본 가이드 - URL 표현법

REST API URL의 자원은 **복수형으로 표현**되며,
하나의 자원에 대한 접근은 **복수형 + 아이디**를 통해 특정 자원에 접근함
/posts 는 **'게시글 전체'**를 칭하는 URL이라고 할 때,
/posts/1 은 **'1번 게시글'**이라는 자원을 표현함

REST API 기본 가이드 - 계층적 자원

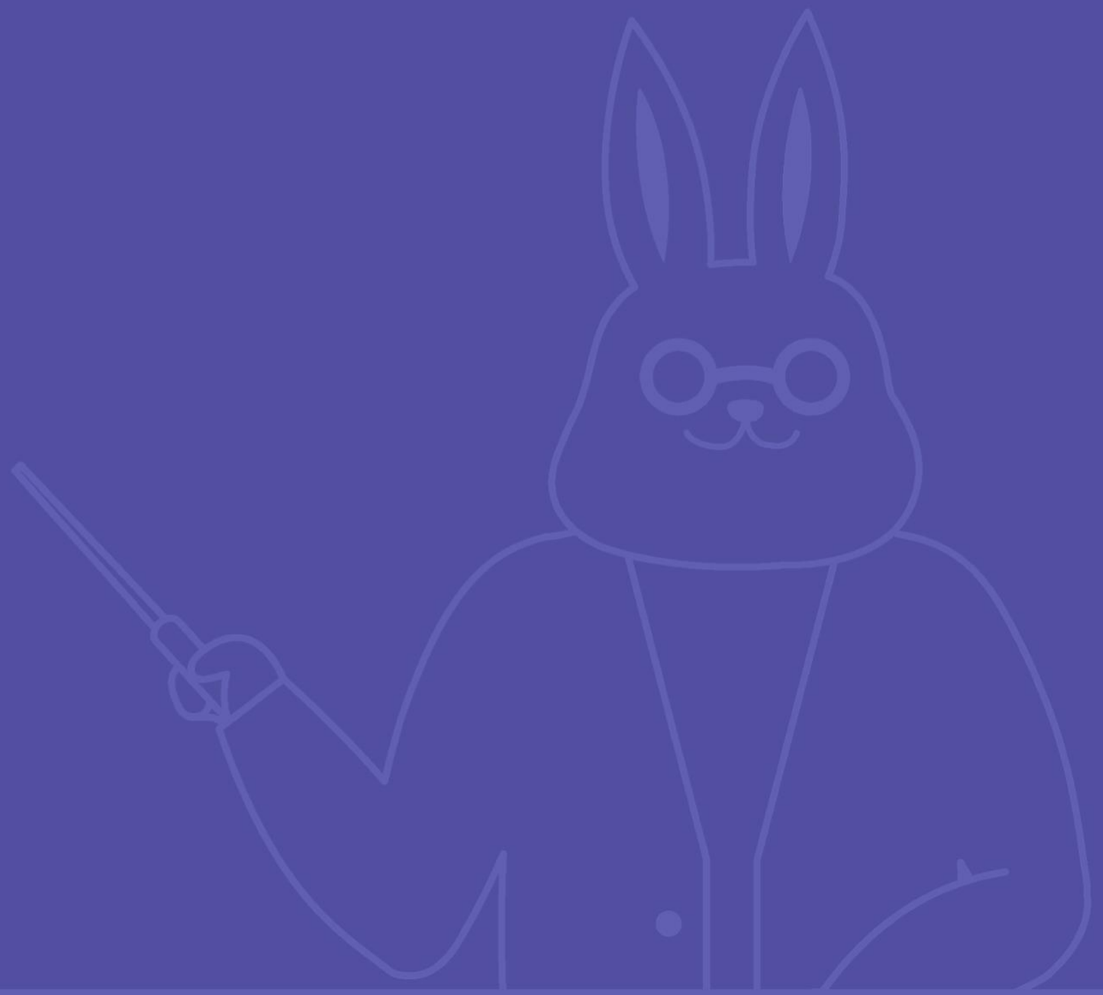
REST API는 URL을 통해 자원을 **계층적으로 표현**함
/users/1/posts 라는 URL은 **'1번 유저의 게시글 전체'**라는 자원을 나타냄

REST API 정리

REST API는 **REST 아키텍처를 준수하는 웹 API**를 의미하며,
URL을 통한 **자원의 표현 방법**과, HTTP method를 통한 **API 동작의 정의** 정도만
사용해도 훌륭한 REST API를 구현할 수 있음

04

JSON



JSON 이란?

JavaScript **O**bject **N**otation
자바스크립트에서 **객체를 표현하는 표현식**으로 시작함
데이터를 표현하는 방법이 **단순하고 이해하기 쉬워서**
웹 API에서 **데이터를 전송할 때 표현식**으로 주로 사용됨

JSON을 사용하는 이유

웹 API는 기본적으로 데이터를 **문자열로 전송**하게 됨
어떤 **객체를 웹 API를 통해서 문자열로 전달**하기 위해 JSON을 사용함

JSON vs XML

animals.json

```
[
  { name: 'cat', legs: 4 },
  { name: 'chicken', legs: 2 },
]
```

animals.xml

```
<array>
  <item>
    <name>cat</name>
    <legs>4</legs>
  </item>
  <item>
    <name>chicken</name>
    <legs>2</legs>
  </item>
</array>
```

JSON이 더욱 **적은 표현식**을 사용하여 **데이터를 효과적**으로 표현함

JSON 가이드 - Object

JSON에서 Object는 **{ key: value }**로 표현함
value에는 **어떤 값이라도 사용**될 수 있음 (문자열, 숫자, JSON 객체 등)

Ex) { name: 'elice', age: 5, nationality: 'korea' }

JSON 가이드 - Array

JSON에서 Array는 **[item1, item2]** 로 표현함
item에는 **어떤 값이라도 사용**될 수 있음 (문자열, 숫자, JSON 객체 등)

Ex) ['first', 10, { name: 'bob' }]

05

Express.js 로 REST API 구현하기

/* elice */

목표

데이터베이스 없이 Node.js 모듈 활용
간단한 메모의 작성, 수정, 삭제, 확인기능 **API 구현**
express-generator를 사용하지 않고 **MVC 패턴 구현**

MVC 패턴

MVC 패턴은 웹 서비스의 가장 대표적인 **프로젝트 구성 패턴**으로
프로젝트의 **기능들을 어떻게 분리할지**에 대한 하나의 구성 방법
Model - View - Controller를 구분하여 프로젝트 구조를 구성하는 것

MVC 패턴 - Model

Model은 **데이터에 접근하는 기능** 또는 **데이터 그 자체**를 의미함
데이터의 읽기, 쓰기는 Model을 통해서만 이루어지도록 구성해야 함

MVC 패턴 - View

View는 **데이터를 표현하는 기능**을 의미함
주로 **Controller에 의해 데이터를 전달**받고
전달받은 데이터를 **화면에 표시**해주는 기능을 담당

MVC 패턴 - Controller

Controller는 **Model을 통해 데이터에 접근**하여,
처리 결과를 View로 전달하는 기능을 의미함
웹 서비스에선 주로 **라우팅 함수가 해당 기능을 수행**함

Express.js 로 MVC 패턴 구현하기

Node.js의 **모듈화를 이용**하여 MVC 패턴을 구현할 수 있음
JSON API를 구현하는 경우,
Node.js는 **기본적으로 JSON을 처리**하는 방법을 가지고 있기 때문에
View는 생략될 수 있음

프로젝트 구현 사항

JavaScript의 **Array 함수 사용하여 데이터 처리** 구현
router와 route handler를 사용하여 **HTTP 요청, 응답 처리** 구현
오류처리 미들웨어를 사용하여, 오류를 처리하는 방법 구현
정의되지 않은 라우팅에 대해 **404 오류 처리** 구현

메모 목록 구현하기

models/note.js

```
let notes = [
  {
    id: 1,
    title: 'first note',
    content: 'My first note is here.'
  }
];

exports.list = () => {
  return notes.map(({ id, title }) => ({
    id,
    title,
  }));
}
```

routes/notes.js

```
const { Router }= require('express');
const Note = require('../models/note');

const router = Router();

router.get('/', (req, res, next) => {
  const notes = Note.list();
  res.json(notes);
});
```

메모 상세 구현하기

models/note.js

```
exports.get = (id) => {
  const note = notes.find(
    (note) => note.id === id
  );

  if (!note) {
    throw new Error('Note not found');
  }
  return note;
}
```

routes/notes.js

```
router.get('/:id', (req, res, next) => {
  const id = Number(req.params.id);

  try {
    const note = Note.get(id);
    res.json(note);
  } catch (e) {
    next(e);
  }
});
```

메모 작성 구현하기

models/note.js

```
exports.create = (title, content) => {
  const { id: lastId } =
    notes[notes.length - 1];
  const newNote = {
    id: lastId + 1,
    title,
    content,
  };
  notes.push(newNote);
  return newNote;
}
```

routes/notes.js

```
router.post('/', (req, res, next) => {
  const { title, content } = req.body;
  const note = Note.create(title, content);
  res.json(note);
});
```

메모 수정 구현하기

models/note.js

```
exports.update = (id, title, content) => {
  const index = notes.findIndex(
    (note) => note.id === id
  );

  if (index < 0) {
    throw new Error('Note not found for update');
  }
  const note = notes[index];
  note.title = title;
  note.content = content;
  notes[index] = note;
  return note;
}
```

routes/notes.js

```
router.put('/:id', (req, res, next) => {
  const id = Number(req.params.id);
  const { title, content } = req.body;

  try {
    const note =
      Note.update(id, title, content);
    res.json(note);
  } catch (e) {
    next(e);
  }
});
```

메모 삭제 구현하기

models/note.js

```
exports.delete = (id) => {
  if (!notes.some((note) => note.id === id)) {
    throw new Error(
      'Note not found for delete'
    );
  }

  notes = notes.filter(note => note.id !== id);

  return;
}
```

routes/notes.js

```
router.delete('/:id', (req, res, next) => {
  const id = Number(req.params.id);

  try {
    Note.delete(id);
    res.json({ result: 'success' });
  } catch (e) {
    next(e);
  }
});
```

JSON 데이터 처리 미들웨어 사용하기

index.js

```
app.use(express.json());
```

express.js 는 **기본적으로** HTTP body에 전달되는 **JSON 데이터를 처리하지 못함**

express에서 기본적으로 제공해 주는 **express.json() 미들웨어를 사용**해야 JSON 데이터를 사용할 수 있음

오류 처리 미들웨어 구현하기

index.js

```
app.use((err, req, res, next) => {
  res.status(500);

  res.json({
    result: 'fail',
    error: err.message,
  });
});
```

가장 마지막 미들웨어로 오류 처리 미들웨어를 적용하면
모든 라우팅에 **공통적인 오류처리 로직**을 적용할 수 있음

정의되지 않은 라우팅에 404 오류 처리하기

index.js

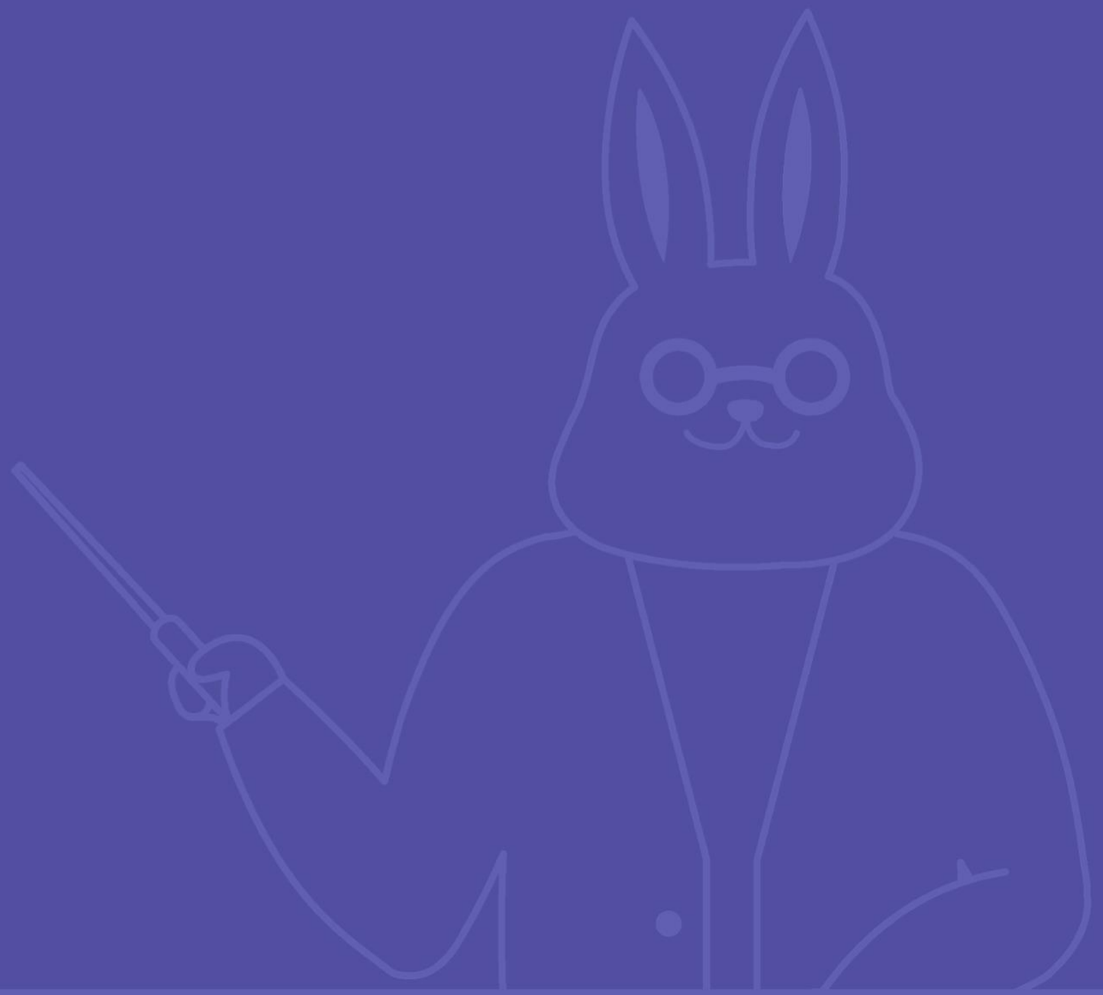
```
app.use((req, res, next) => {
  res.status(404);
  res.send({
    result: 'fail',
    error: `Page not found ${req.path}`
  });
});
```

모든 라우팅이 적용된 이후에 사용되는
미들웨어는 **설정된 경로가 없는 요청**을
처리하는 Route Handler로 동작함

Express.js 는 기본적인 404 페이지를 가지고
있지만, 직접 처리가 필요 한 경우
이와 같은 Route Handler를 추가해야 함

06

Postman 사용하기



Postman 소개

Postman은 **API를 테스트할 수 있는 도구**로,
HTTP 요청을 손쉽게 작성하여 테스트해 볼 수 있게 도움
추가로 **API를 문서화** 할 수 있는 기능 및 다양한 도구를 제공함

Postman으로 API 문서화하기

- ✓ collection 만들기
- ✓ api request 만들기
- ✓ document 작성하기
- ✓ 전체 문서 확인하기

Postman으로 API 테스트하기

- ✓ HTTP Method 설정하기
- ✓ query param 사용하기
- ✓ path variable 사용하기
- ✓ body 사용하기

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