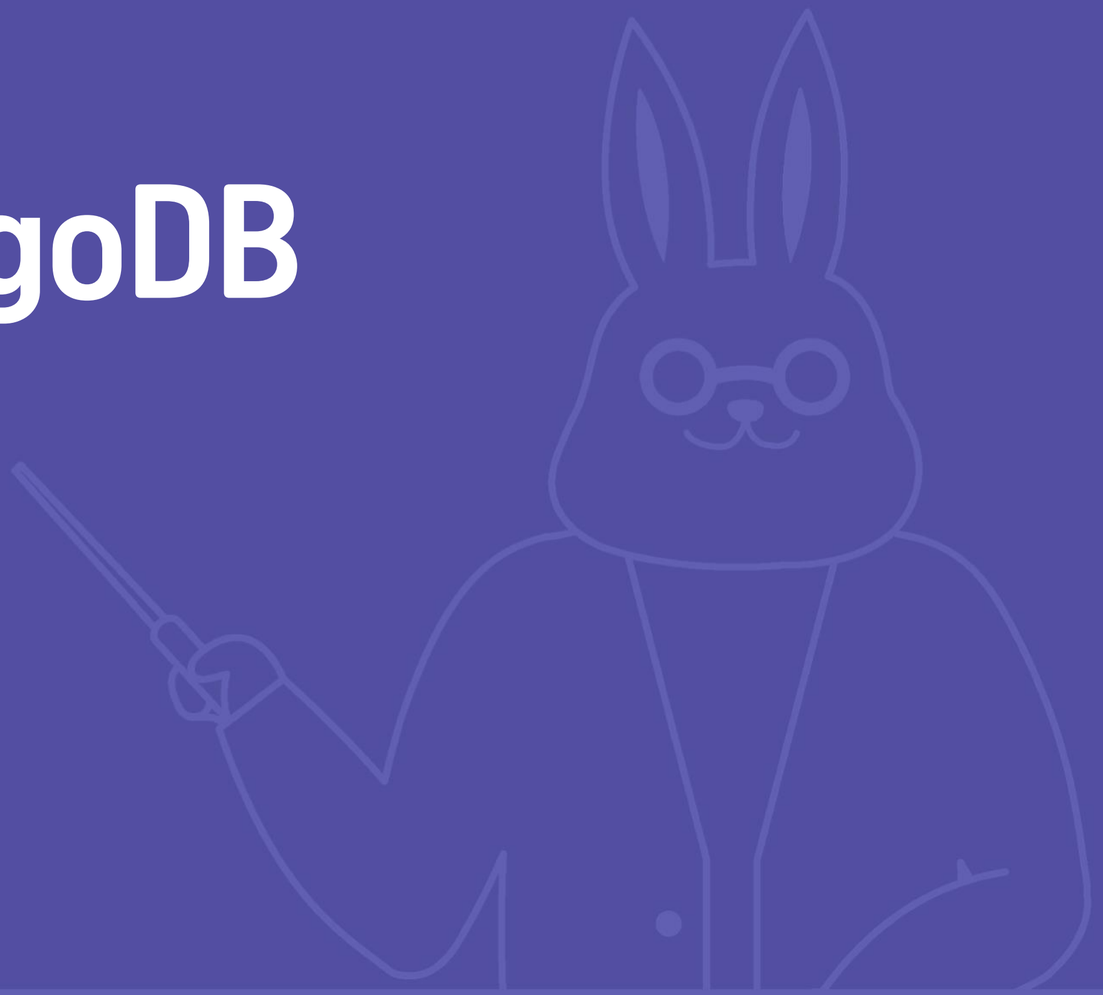
Express.js와 MongoDB

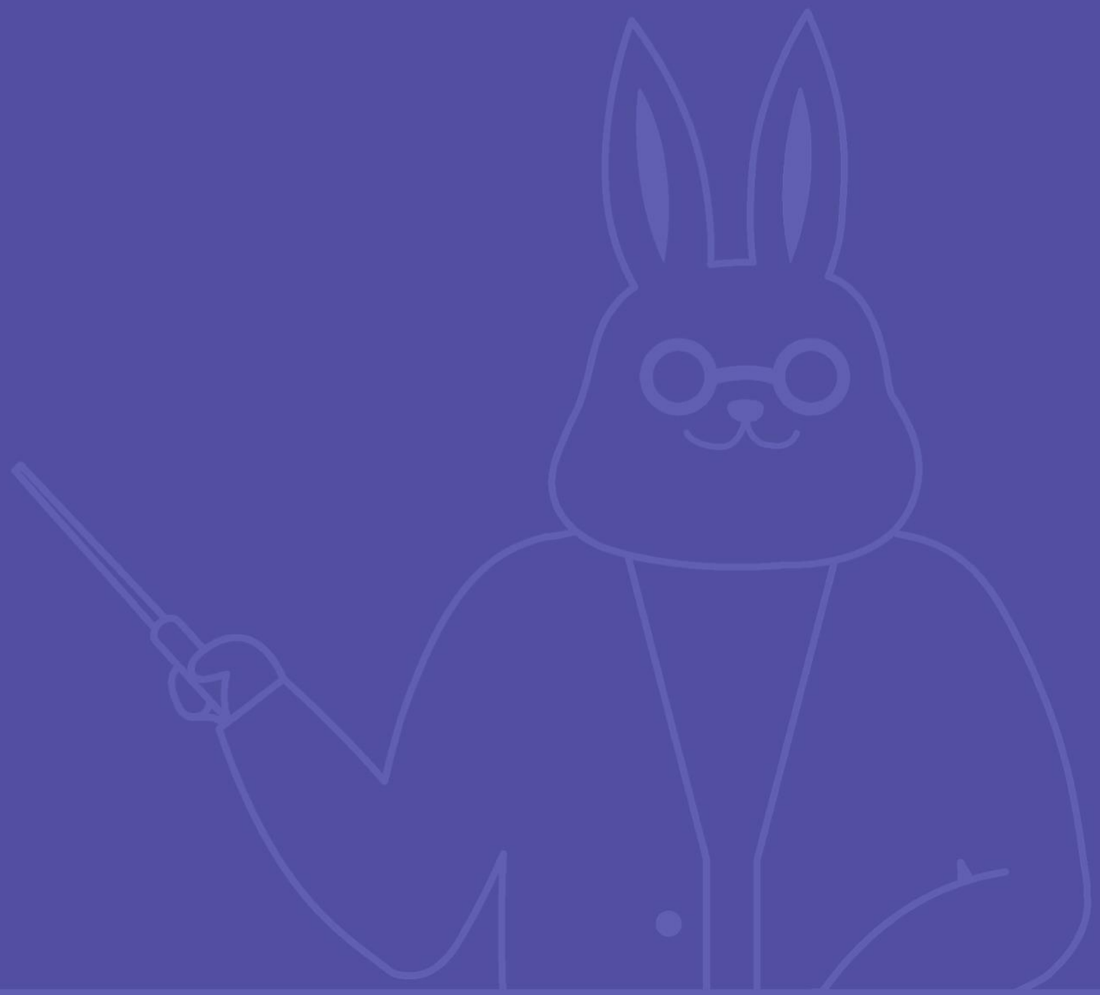
02 REST API



/* elice */

01

REST API

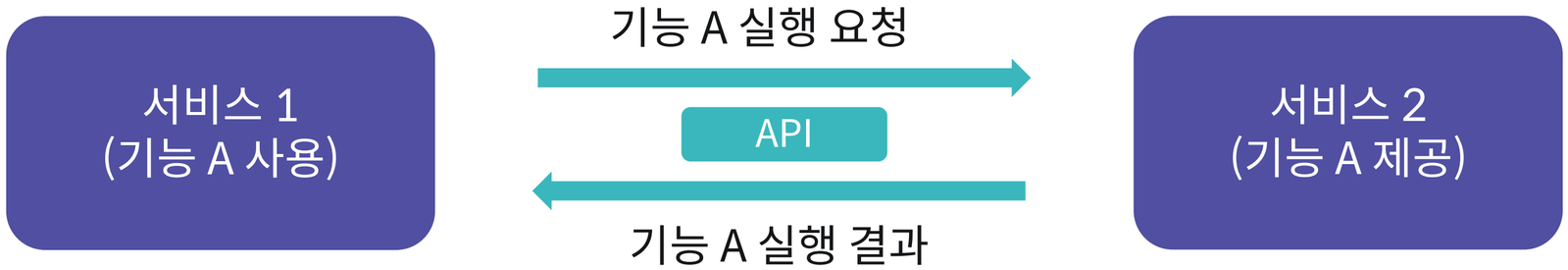


REST API란?

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

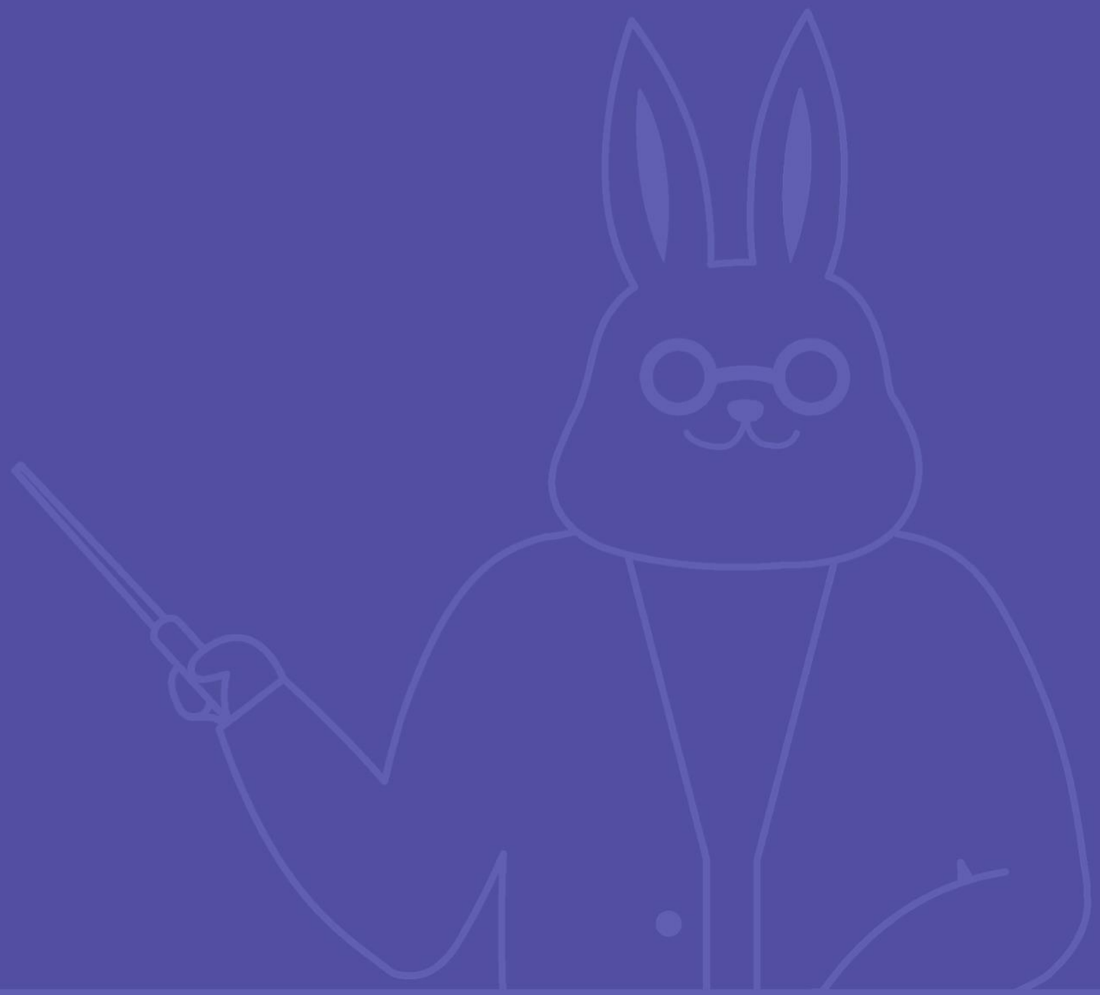
REST API는 URL을 통해 자원을 **계층적으로 표현**함
/users/1/posts 라는 URL은 **'1번 유저의 게시글 전체'**라는 자원을 나타냄

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

02

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' }]

03

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

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io