

Node.js와 MongoDB II

02 Express.js와 MongoDB로 웹서비스 만들기 1



목차

01. 최종적으로 만들 웹 서비스 소개
02. Template Engine
03. 게시판 CRUD 만들기
04. Async Request Handler
05. Pagination
06. 추가 - PM2 Process Manager

수강목표

1. Template Engine 사용하기

SSR을 구현하기 위한 기술인 Template Engine에 대해 학습하고 Express.js의 PUG Template Engine 사용 방법을 익힌다.

2. 간단한 CRUD 웹 만들기

Express.js를 이용해 간단한 CRUD 동작을 하는 웹을 만들고 관련 기술들의 사용방법을 익힌다.

3. PM2 Process Manager

PM2 Process Manager를 이용하여 Node.js 어플리케이션을 관리하는 방법에 대해 학습한다.

01

최종적으로 만들 웹 서비스 소개



✓ 최종적으로 만들 웹 서비스

Express.js와 MongoDB로 웹서비스 만들기 1 ~ 3까지 **간단한 게시판을 제작**
기본적인 게시판에서부터 로그인, 메일 발송 등
필요한 기능을 하나씩 추가하며 각 기능에 대해 학습 예정

✓ 게시판을 제작하는 이유

게시판은 **데이터를 작성하고 보여주는** 기능을 구현하기 **가장 기본적인 형태**
웹 서비스 **개발의 기본을 학습**하기 좋음
게시판을 통해 기본기를 잘 다지면 **무엇이든 응용 가능**

→ 텍스트 데이터와 이미지, 비디오, 좌표 등의 데이터를 추가하고
표현 방식만 변경하면 완전히 다른 서비스로 보임

✓ 게시판 기능

- ✓ 게시판 목록
- ✓ 게시글 보기
- ✓ 게시글 작성
- ✓ 게시글 수정
- ✓ 게시글 삭제

✓ 회원기능

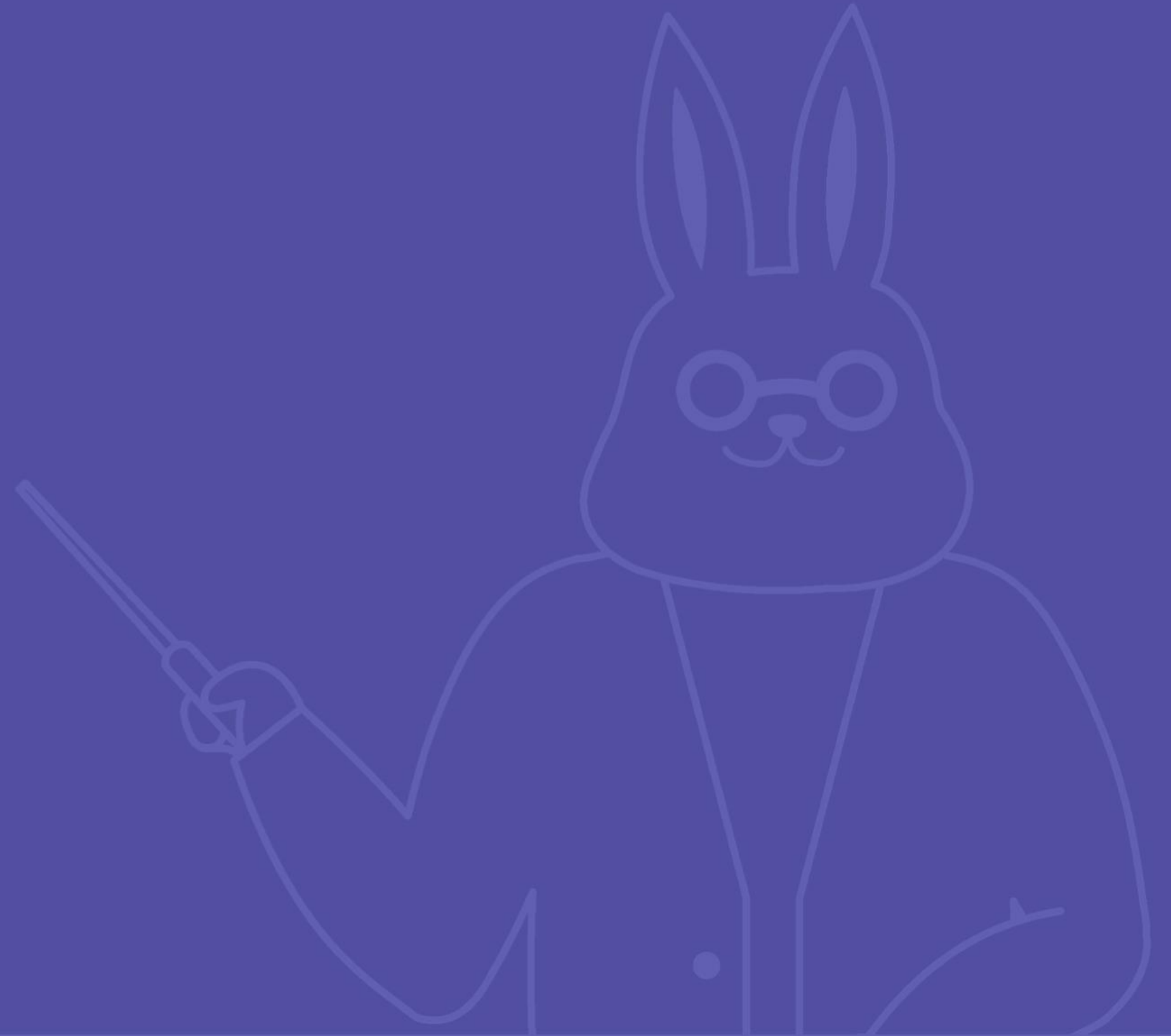
- ✓ 회원가입
- ✓ 로그인
- ✓ 비밀번호 찾기

✓ 추가기능

- ✓ Pagination
- ✓ 구글 로그인
- ✓ 유저 작성글 모아보기

02

Template Engine

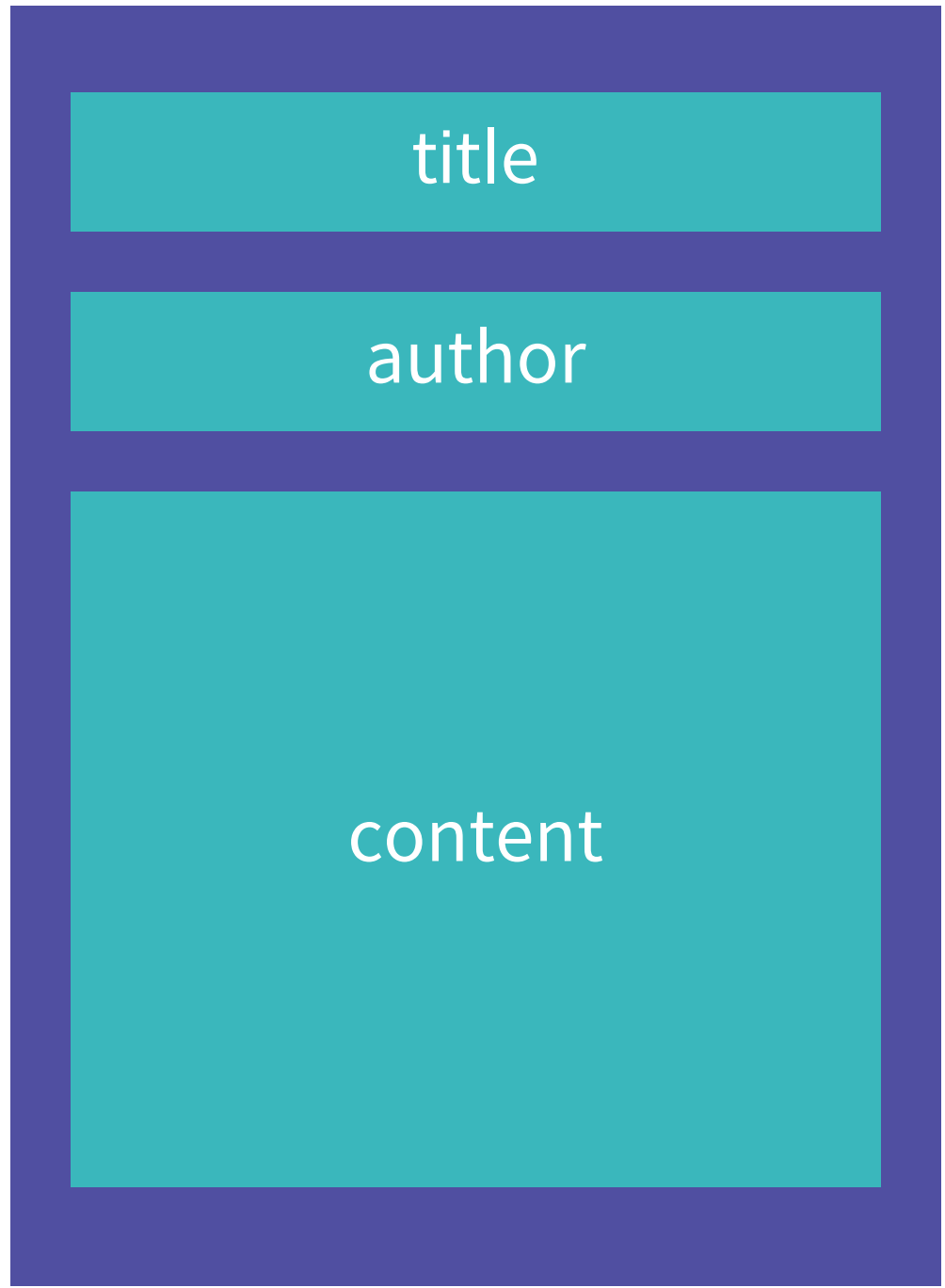


✓ 템플릿 엔진이란?

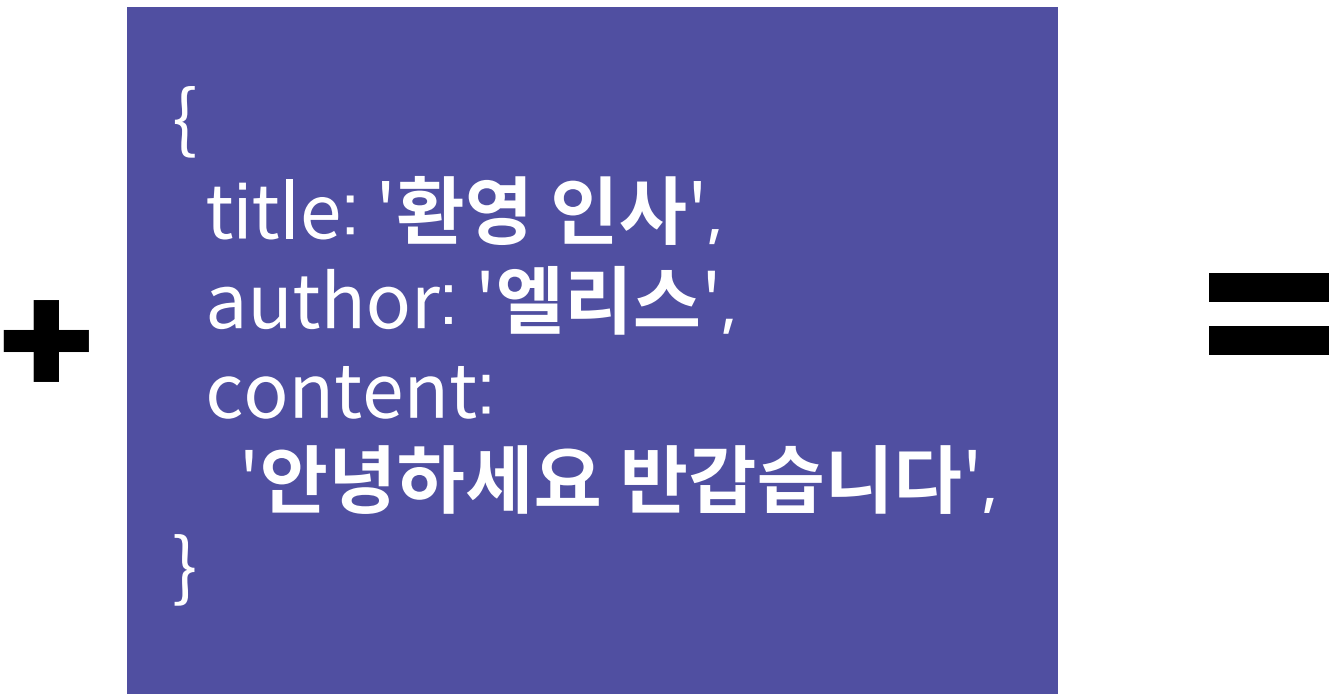
서버에서 클라이언트로 보낼 **HTML의 형태**를 미리 템플릿으로 저장
동작 시에 미리 작성된 템플릿에 데이터를 넣어서 완성된 HTML 생성
템플릿 엔진은 **템플릿 작성 문법**과 작성된 **템플릿을 HTML로 변환**하는 기능을 제공

✓ 템플릿 엔진이란?

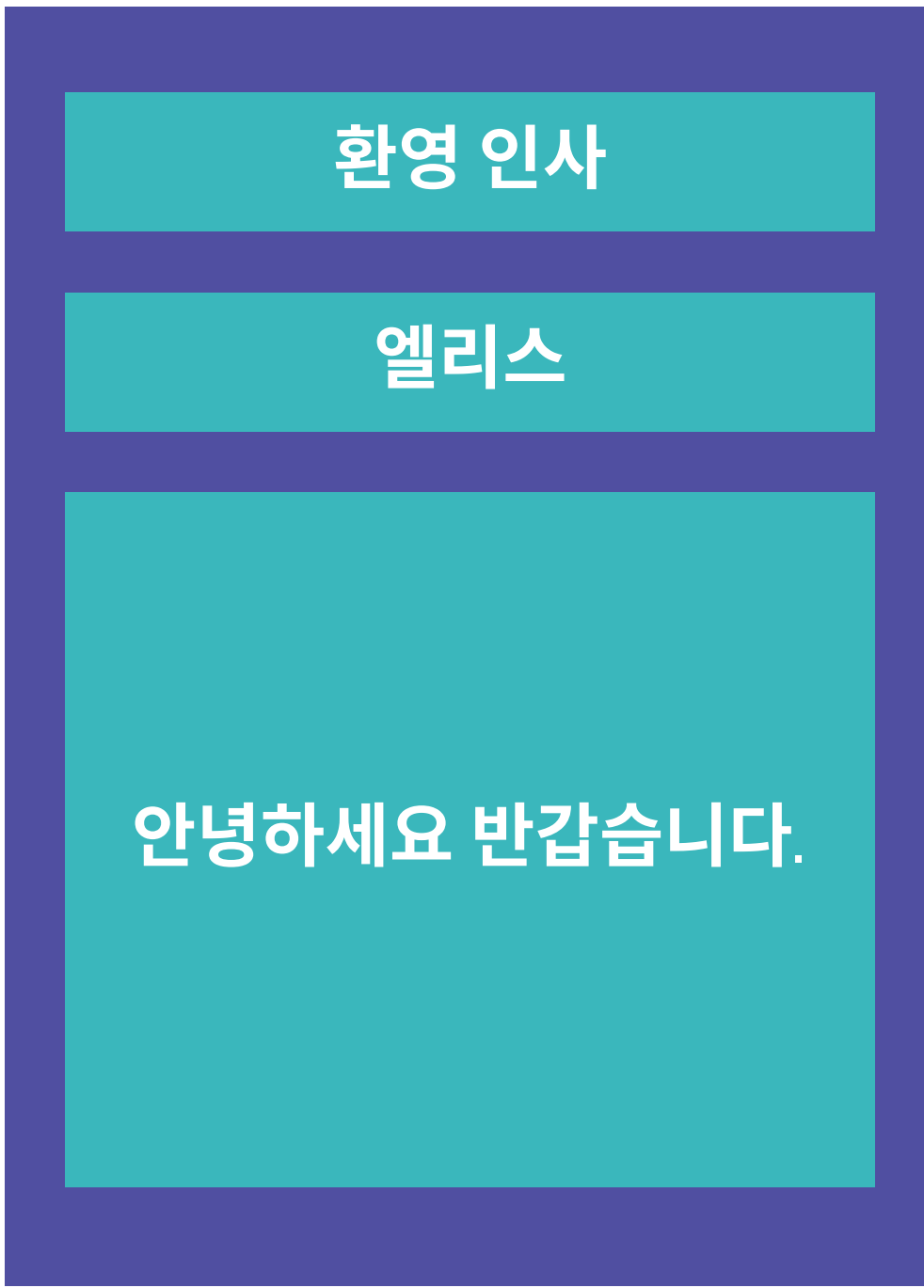
템플릿



데이터



화면



✓ Express.js의 템플릿 엔진

EJS - html과 유사한 문법의 템플릿 엔진

Mustache - 간단한 데이터 치환 정도만 제공하는 경량화된 템플릿 엔진

Pug - 들여쓰기 표현식을 이용한 간략한 표기와 레이아웃 등 강력한 기능을 제공

✓ 이번 강의에서 사용할 템플릿 엔진 소개

Pug

들여쓰기 표현식을 이용해 가독성이 좋고 개발 생산성이 높음
HTML을 잘 모르더라도 문법적 실수를 줄일 수 있음
layout, include, mixin 등 강력한 기능 제공

✓ pug 문법 소개

index.pug

```
html
  head
    title= title
  body
    h1#greeting 안녕하세요
    a.link(href="/") 홈으로
```

HTML 닫기 태그 없이
들여쓰기로 블록을 구분

= 을 이용해서 전달받은 변수 사용 가능

id나 class는 태그 뒤에 이어서 바로 사용
() 을 이용해서 attribute 사용

✓ pug 문법 소개

each, if

```
each item in arr
  if item.name == 'new'
    h1 New Document
  else
    h1= `${item.name}`
```

each ~ in 표현식으로
주어진 **배열의 값을 순환**하며
HTML 태그를 만들 수 있음

if, else if, else를 이용해
주어진 값의 **조건을 확인**하여
HTML 태그를 만들 수 있음

✓ pug 문법 소개

layout

```
--- layout.pug ---
html
  head
    title= title
  body
    block content
--- main.pug ---
extends layout
block content
  h1 Main Page
```

block을 포함한 템플릿을 선언하면
해당 템플릿을 layout으로 사용할 수 있음

layout을 **extends** 하면
block 부분에 작성한 HTML 태그가 포함됨

반복되는 웹사이트의 틀을 작성해 두고
extends 하며 개발하면 매우 편리한 기능

✓ pug 문법 소개

include

```
---title.pug---  
h1= title  
---main.pug  
extend layout  
block content  
  include title  
  div.content  
    안녕하세요  
  pre  
    include article.txt
```

자주 반복되는 구문을 미리 작성해 두고
include 하여 사용할 수 있음

일반적인 **텍스트 파일**도
include 하여 템플릿에 포함 가능

✓ pug 문법 소개

mixin

```
--- listItem.pug ---
mixin listItem(title, name)
  tr
    td title
    td name
--- main.pug ---
include listItem
table
  tbody
    listItem('제목', '이름')
```

mixin 을 사용하여 템플릿을 함수처럼 사용할 수 있게 선언할 수 있음

include는 값을 지정할 수 없지만
mixin 은 파라미터를 지정하여
값을 넘겨받아 템플릿에 사용할 수 있음

✓ Express.js와 pug의 연동

express + pug

```
--- app.js ---
app.set('views',
  path.join(__dirname, 'views'));
app.set('view engine', 'pug');

--- request handler ---
res.render('main', {
  title: 'Hello Express',
});
```

app.set을 이용해
템플릿이 저장되는 디렉터리를 지정하고,
어떤 템플릿 엔진을 사용할지 지정할 수 있음

res.render 함수는
app.set에 지정된 값을 이용해
화면을 그리는 기능을 수행함

render 함수의 첫 번째 인자는 **템플릿의 이름**
두 번째 인자는 **템플릿에 전달되는 값**

✓ Express.js의 app.locals

app.locals

```
--- app.js ---  
app.locals.appName = "Express"  
  
--- main.pug ---  
h1= appName  
  
// <h1>Express</h1>
```

Express.js의 **app.locals**를 사용하면 **render 함수에 전달되지 않은** 값이나 함수를 사용할 수 있음

템플릿에 **전역으로 사용될 값**을 지정하는 역할

✓ express-generator 사용 시 템플릿 엔진 지정하기

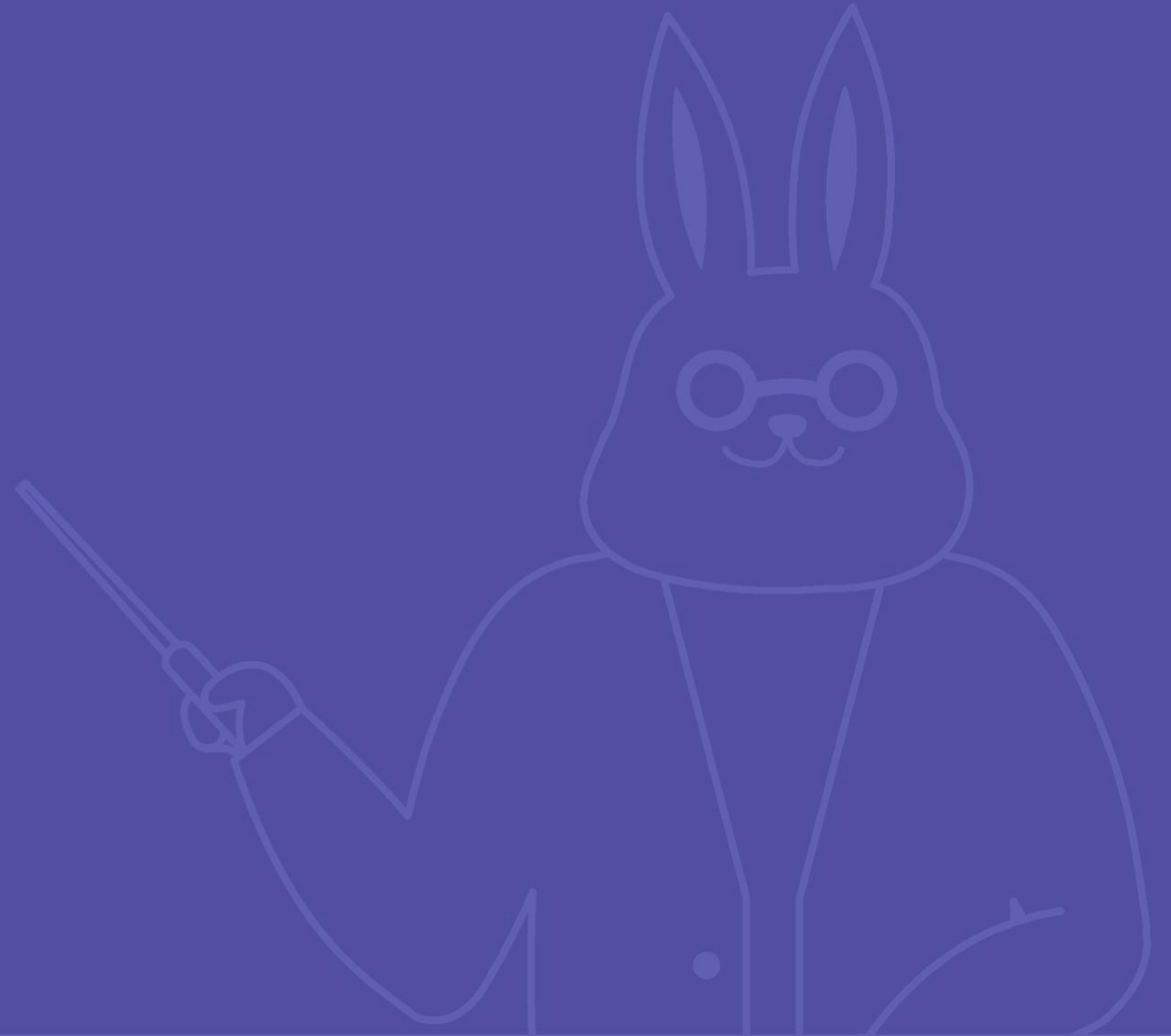
express with view

```
$ express --view=pug myapp
```

express-generator는 기본적으로 **jade**라는 **템플릿 엔진**을 사용
jade는 **pug의 이전 이름**으로, 최신 지원을 받기 위해선
템플릿 엔진을 pug로 지정 해야 함
--view 옵션을 사용하여 **템플릿 엔진을 지정**할 수 있음

03

게시판 CRUD 만들기



✓ CRUD란?

Create, Read, Update, Delete

데이터를 다루는 네 가지 기본적인 기능
일반적으로 위 네 가지에 대한 구현이 가능해야
서비스 개발에 필요한 요구사항을 충족할 수 있음

✓ 게시판에서의 CRUD

Create

게시판은 **게시글을 작성**할 수 있어야 함

게시글 작성 시 제목, 내용, 작성자, 작성 시간 등의 **정보를 기록** 함

게시글의 제목과 내용은 최소 n글자 이상이어야 함

✓ 게시판에서의 CRUD

Read

게시판은 **게시글의 목록**과 **게시글의 상세**를 볼 수 있어야 함

게시글 목록은 제목과 작성자 작성 시간의 **간략화된 정보**를 보여줌

게시글 상세는 제목, 작성자, 내용, 작성 시간, 수정 시간 등의 **상세한 정보**를 보여줘야 함

✓ 게시판에서의 CRUD

Update

게시판의 게시글은 **수정이 가능**해야 함

게시글 수정 시 **제목과 내용이 수정 가능**하고, 수정 시간이 기록되어야 함

게시글의 제목과 내용은 최소 n글자 이상이어야 함

게시글 수정은 작성자만 가능해야 함

✓ 게시판에서의 CRUD

Delete

게시판의 게시글은 **삭제가 가능**해야 함

게시글 삭제 시 목록과 상세에서 게시글이 접근되지 않아야 하며

게시글 삭제는 작성자만 가능해야 함

✓ Express.js + Mongoose로 CRUD 구현하기 - 모델 선언하기

post 모델

```
--- ./models/schemas/post.js
const mongoose, { Schema } = require('mongoose');
const shortId = require('./types/short-id');

module.exports = new Schema({
  shortId,
  title: String,
  content: String,
  author: String,
}, {
  timestamps: true,
});

--- ./models/index.js ---
exports.Post = mongoose.model('Post', PostSchema)
```

MongoDB의 ObjectId는
URL에 사용하기 좋은 값이 아니기 때문에
대체할 수 있는 아이디를 shortId로 생성

제목, 내용, 작성자를 String 타입으로
스키마에 선언
(회원가입 로그인 후 작성자 연동)

timestamps 옵션으로 **작성 시간, 수정
시간을 자동으로 기록**해 줌

✔ Express.js + Mongoose로 CRUD 구현하기 - shortId

shortId

```
const { nanoid } = require('nanoid');

const shortId = {
  type: String,
  default: () => {
    return nanoid();
  },
  require: true,
  index: true,
}

module.exports = shortId;
```

ObjectId를 대체할 shortId 타입을
Mongoose Custom Type으로 선언

중복 없는 문자열을 생성해 주는
nanoid 패키지 활용

default를 이용해 모델 생성 시 자동으로
ObjectId를 대체할 아이디 생성

✓ Express.js + Mongoose로 CRUD 구현하기 - 게시물 작성

게시글 작성 흐름

1. `/posts?write=true` 로 작성페이지 접근
2. `<form action="/posts" method="post">` 이용해 post 요청 전송
3. `router.post` 이용하여 **post 요청 처리**
4. `res.redirect` 이용하여 post 완료 처리

✓ 게시물 작성 - 작성 페이지 만들기

./routes/posts.js

```
const { Router } = require('express');
const router = Router();

router.get('/', (req, res, next) => {
  if (req.query.write) {
    res.render('posts/edit');
    return;
  }
  ...
});

...
module.exports = router;
```

./views/posts/edit.pug

```
...
form(action="/posts", method="post")
  table
    tbody
      tr
        th 제목
        td: input(type="text" name="title")
      tr
        th 내용
        td: textarea(name="content")
      td
        td(colspan="2")
          input(type="submit" value="등록")
```


✓ 게시물 작성 - POST 요청 처리하기

./routes/posts.js

```
const { Post } = require('./models');
...
router.post('/', async (req, res, next) => {
  const { title, content } = req.body;
  try {
    await Post.create({
      title,
      content,
    });
    res.redirect('/');
  } catch (err) {
    next(err);
  }
});

...
```

✓ Express.js + Mongoose로 CRUD 구현하기 - 게시물 목록 및 상세

게시글 목록 및 상세 흐름

1. **/posts** 로 목록 페이지 접근
2. `` 이용하여 상세 URL Link
3. `router.get('/:shortId')` **path parameter** 이용하여 요청 처리

✓ 게시물 목록 및 상세 - 게시물 목록 구현하기

./routes/posts.js

```
router.get('/', async (req, res, next) => {  
  const posts = await Post.find({});  
  res.render('posts/list', { posts });  
});
```

./views/posts/list.pug

```
...  
table  
  tbody  
    each post in posts  
      tr  
        td  
          a(href=`/posts/${post.shortId}`)  
            = post.title  
        td= post.author  
        td= formatDate(post.createdAt)  
  tfoot  
    tr  
      td(colspan="3")  
        a(href="/posts?write=true")  
          등록하기
```

✓ 게시물 목록 및 상세 - formatDate 함수 추가하기

app.js

```
const dayjs = require('dayjs')

app.locals.formatDate = (date) => {
  return dayjs(date).format('YYYY-MM-DD HH:mm:ss');
}
```

✓ 게시물 목록 및 상세 - 게시물 상세 구현하기

./routes/posts.js

```
router.get('/:shortId', async (req, res, next) => {
  const { shortId } = req.params;
  const post = await Post.findOne({ shortId });
  if (!post) {
    next(new Error('Post Not Found'));
    return;
  }
  ...
  res.render('posts/view', { post });
});
```

./views/posts/view.pug

```
...
table
  tbody
    tr
      td(colspan="2")= post.title
    tr
      td= post.author
      td= formatDate(post.createdAt)
    tr
      td(colspan="2"): pre= post.content
    tr
      td: a(href='/posts/${post.shortId}?edit=true')
        수정
      td
        button(onclick=`deletePost("${post.shortId}")`)
          삭제
```

✓ Express.js + Mongoose로 CRUD 구현하기 - 게시물 수정

게시글 수정 흐름

1. `/posts/{shortId}?edit=true` 로 수정페이지 접근
2. 작성페이지를 수정페이지로도 동작하도록 작성
3. `<form action="/posts/:shortId" method="post">` 로 post 요청 전송

※ html form은 PUT method를 지원하지 않기 때문에 post 사용

✔ 게시물 수정 - 수정 페이지 만들기

./routes/posts.js

```
router.get('/:shortId', async (req, res, next) => {
  ...
  if (req.query.edit) {
    res.render('posts/edit', { post });
  }
  ...
});
```

./views/posts/edit.pug

```
...
- var action = post ? '/posts/${post.shortId}' : "/posts"
form(action=action, method="post")
  table
    tbody
      tr
        th 제목
        td: input(type="text" name="title" value=post&&post.title)
      tr
        th 내용
        td: textarea(name="content")= post&&post.content
      td
        td(colspan="2")
          - var value = post ? "수정" : "등록"
          input(type="submit" value=value)
```

✓ 게시물 수정 - 수정 요청 처리하기

./routes/posts.js

```
...
router.post('/:shortId', async (req, res, next) => {
  const { shortId } = req.params;
  const { title, content } = req.body;
  const post = await Post.findOneAndUpdate({ shortId }, {
    title, content,
  });
  if (!post) {
    next(new Error('Post Not Found'));
    return;
  }
  res.redirect(`/posts/${shortId}`);
});
```


✔ Express.js + Mongoose로 CRUD 구현하기 - 게시물 삭제

게시글 삭제 흐름

1. 게시물 상세 페이지에 **삭제 버튼** 추가
2. html form은 **DELETE** 메서드를 지원하지 **않음**
3. JavaScript를 이용해 **fetch** 함수로 **HTTP Delete 요청** 전송
4. router.delete의 **응답**을 **fetch**에서 처리

✓ 게시물 삭제 - HTTP Delete 요청 전송 및 응답 처리

posts/view.pug

```
td
  button.delete(
    onClick=`deletePost("${post.shortId}")`
  ) 삭제
...

```

```
...
script(type="text/javascript").
  function deletePost(shortId) {
    fetch('/posts/' + shortId, { method: 'delete' })
      .then((res) => {
        if (res.ok) {
          alert('삭제되었습니다.');
```

window.location.href = '/posts';

```
        } else {
          alert('오류가 발생했습니다.');
```

console.log(res.statusText);

```
        }
      })
      .catch((err) => {
        console.log(err);
        alert('오류가 발생했습니다.');
```

});

```
  }
```

✓ 게시물 삭제 - DELETE 요청 처리하기

./routes/posts.js

```
const { Post } = require('./models');
...

router.delete('/:shortId', async (req, res, next) => {
  const { shortId } = req.params;
  try {
    await Post.delete({ shortId });
    res.send('OK');
  } catch (err) {
    next(err);
  }
});

...
```

04

Async Request Handler



✓ request handler의 오류처리

request handler에서 오류를 처리하기 위한 방법

- ✓ promise().**catch(next)**
- ✓ async function, **try ~ catch, next**

✓ async request handler

async의 비동기 처리는 매우 편리하지만,
매번 **try - catch 구문을 작성하는 것은** 귀찮고 실수하기 쉬움
request handler를 async function으로 작성하면서
try ~ catch, next를 자동으로 할 수 있도록 구성한 아이디어

✓ async request handler

asyncHandler

```
const asyncHandler = (requestHandler) => {
  return async (req, res, next) => {
    try {
      await requestHandler(req, res);
    } catch (err) {
      next(err);
    }
  }
}

---
router.get('/', asyncHandler(async (req, res) => {
  const posts = await Posts.find({});
  if (posts.length < 1) {
    throw new Error('Not Found');
  }
  res.render('posts/list', { posts });
}));
```

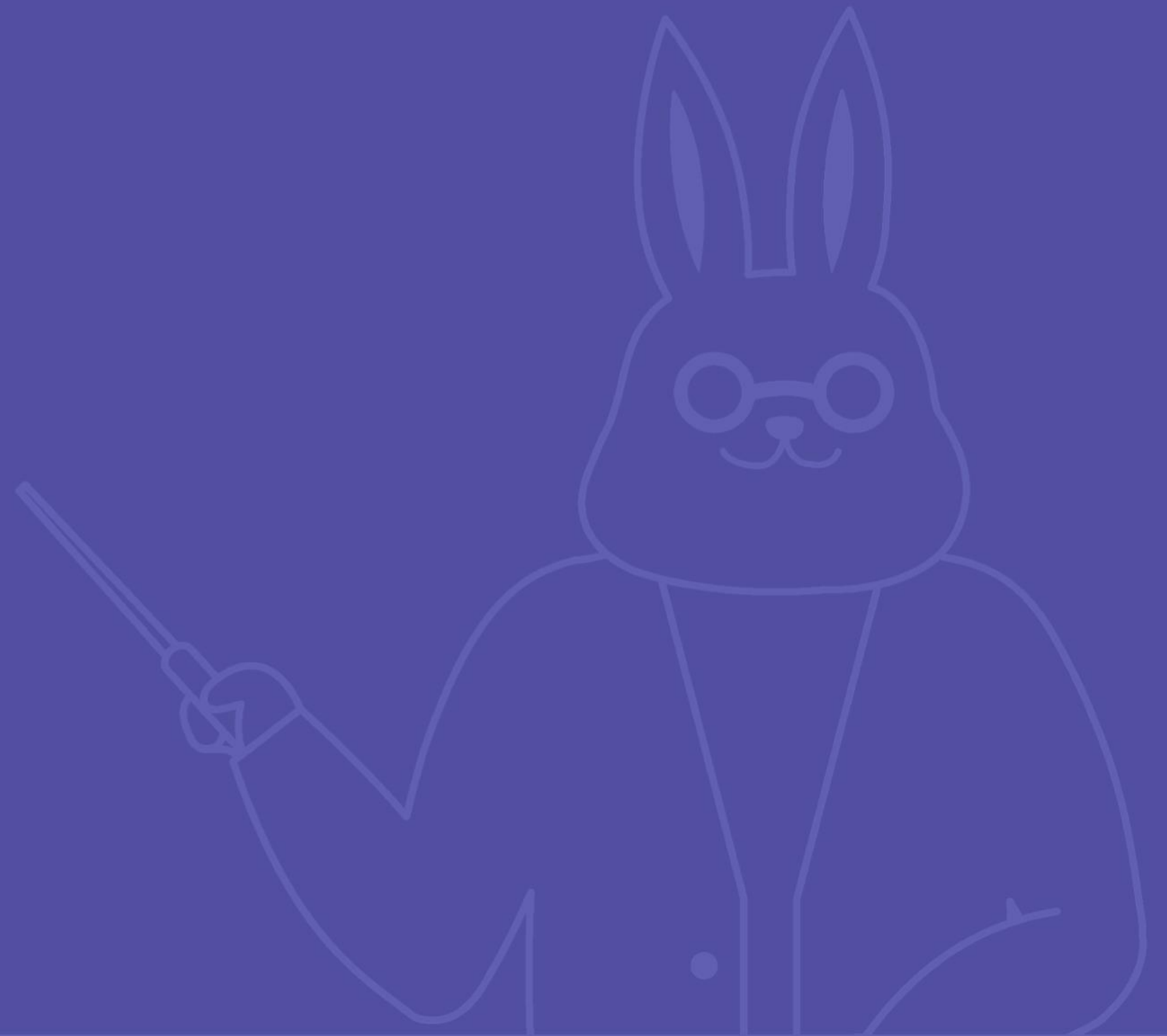
asyncHandler 는 **requestHandler**를
매개변수로 갖는 함수형 미들웨어

전달된 requestHandler는 try ~ catch로
감싸져 **asyncHandler 내에서 실행**되고,

throw 되는 에러는 자동으로
오류처리 미들웨어로 전달되도록 구성됨.

05

Pagination



✓ Pagination이란?

데이터가 많아지면 한 페이지의 목록에 **모든 데이터를 표현하기 어려움**

Pagination은 데이터를 **균일한 수로 나누어 페이지로 분리**하는 것

EX) 10개씩 나누어 1페이지에는 1~10번까지, 2페이지엔 11~20번까지 보여주기

✓ Express.js + Mongoose의 Pagination

```
router.get(... => {  
  const page =  
    Number(req.query.page || 1)  
  const perPage =  
    Number(req.query.perPage || 10)  
  
  ...  
})
```

page - 현재 페이지

perPage - 페이지 당 게시글 수

/posts?**page=1&perPage=10**

일반적으로 **url query** 를 사용해 전달

query는 **문자열로 전달**되기 때문에

Number 로 **형변환**이 필요함

✓ Express.js + Mongoose의 Pagination

code

```
router.get(... => {  
  ...  
  const total = await Post  
    .countDocument({});  
  const posts = await Post.find({})  
    .sort({ createdAt: -1 })  
    .skip(perPage * (page - 1))  
    .limit(perPage);  
  const totalPages =  
    Math.ceil(total / perPage);  
  ...  
}
```

MongoDB 의 **limit**과 **skip**을 사용하여
pagination 구현 가능

limit - 검색 **결과 수 제한**

skip - 검색 시 **포함하지 않을 데이터 수**

pagination 시에는 **데이터의 순서**가
유지될 수 있도록 **sort**를 사용할 수 있도록 함

게시글 수 / 페이지 당 게시글 수 = 총 페이지 수

✓ Express.js + Mongoose의 Pagination

pagination

```
mixin pagination(path)
  p
  - for(let i = 1; i <= totalPages; i++)
    a(href=`${path}?page=${i}&perPage=${perPage}`)
      if i == page
        b= i
      else
        = i
    = " "
  ---
  include pagination
  tr
    td
      +pagination("/posts")
```

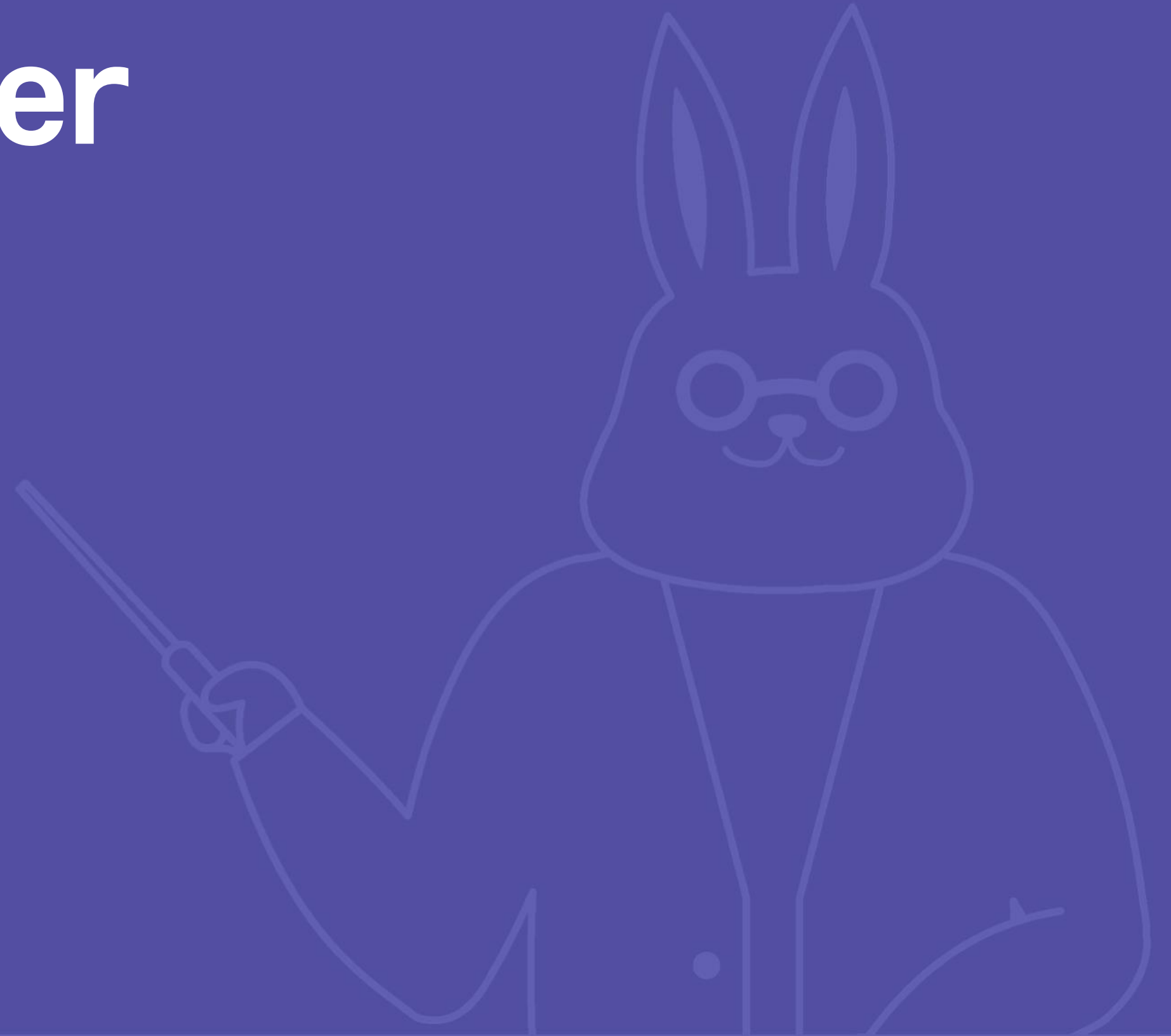
pagination을 **mixin**으로 선언

pagination이 필요한 페이지에서
해당 템플릿을 include한 후,
+pagination으로 **mixin** 을 사용 함

현재 페이지는 b 태그로 굵게 표시

06

PM2 Process Manager



✓ PM2란?

Node.js 작업을 관리해주는 **Process Manager**

node 명령어로 실행 시 **오류 발생**이나 **실행 상태 관리**를 할 수 없음

pm2 는 작업 관리를 위한 **다양한 유용한 기능**을 제공해 줌

✓ PM2를 사용하는 이유

안정적인 프로세스 실행 - 오류발생 시 자동 재실행

빠른 개발환경 - 소스 코드 변경 시 자동 재실행

배포 시 편리한 관리 - pm2 에 모든 프로세스를 한 번에 관리

✓ PM2 사용 방법

ecosystem.config.js

```
module.exports = {  
  apps : [{  
    name: 'simple-board',  
    script: './bin/www',  
    watch: '.',  
    ignore_watch: 'views',  
  }],  
};  
---  
$ pm2 start
```

\$ pm2 init simple 혹은
\$ pm2 init 명령어를 이용하여
pm2 **설정파일 예제**를 만들 수 있음.

예제를 수정하여 설정파일을 생성한 후,
\$ pm2 start 명령어를 실행하면
어플리케이션을 **pm2 데몬으로 실행**해 줌

개발 시 watch 옵션 사용하여
파일 변경 시 서버 자동 재실행 구성

✓ PM2 Example

id	name	namespace	version	mode	pid	uptime	🔄	status	cpu	mem	user	watching
0	simple-board	default	0.0.0	fork	15776	2s	0	online	70.4%	60.3mb	bbulb	enabled

Process List

[0] simple-board Mem: 45 MB CPU:

simple-board Logs

Custom Metrics

Heap Size 18.22 MiB
Heap Usage 94.34 %
Used Heap Size 17.19 MiB
Active requests 0

Metadata

App Name simple-board
Namespace default
Version 0.0.0
Restarts 0

left/right: switch boards | up/down/mouse: scroll | Ctrl-C: exit

To go further check out <https://pm2.io/>

PM2 실행화면 예시 - pm2 status, pm2 monit

크레딧

/* elice */

코스 매니저

이재성

콘텐츠 제작자

최규범

강사

최규범

감수자

최규범

디자이너

김루미

연락처

TEL

070-4633-2015

WEB

<https://elice.io>

E-MAIL

contact@elice.io

