

# 프로젝트 기반 실무형 서비스

## Chap05. koa 활용하기

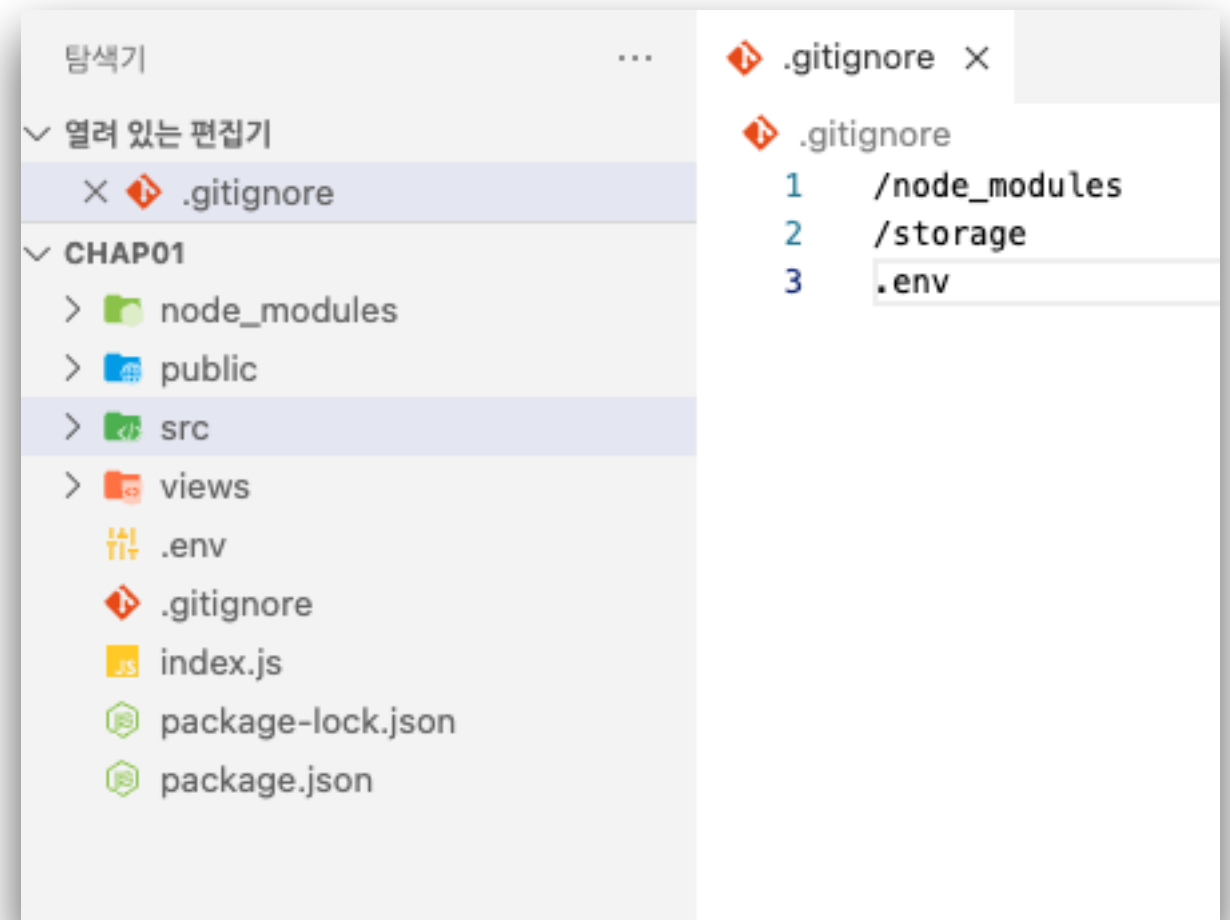
김진형

# git 설정

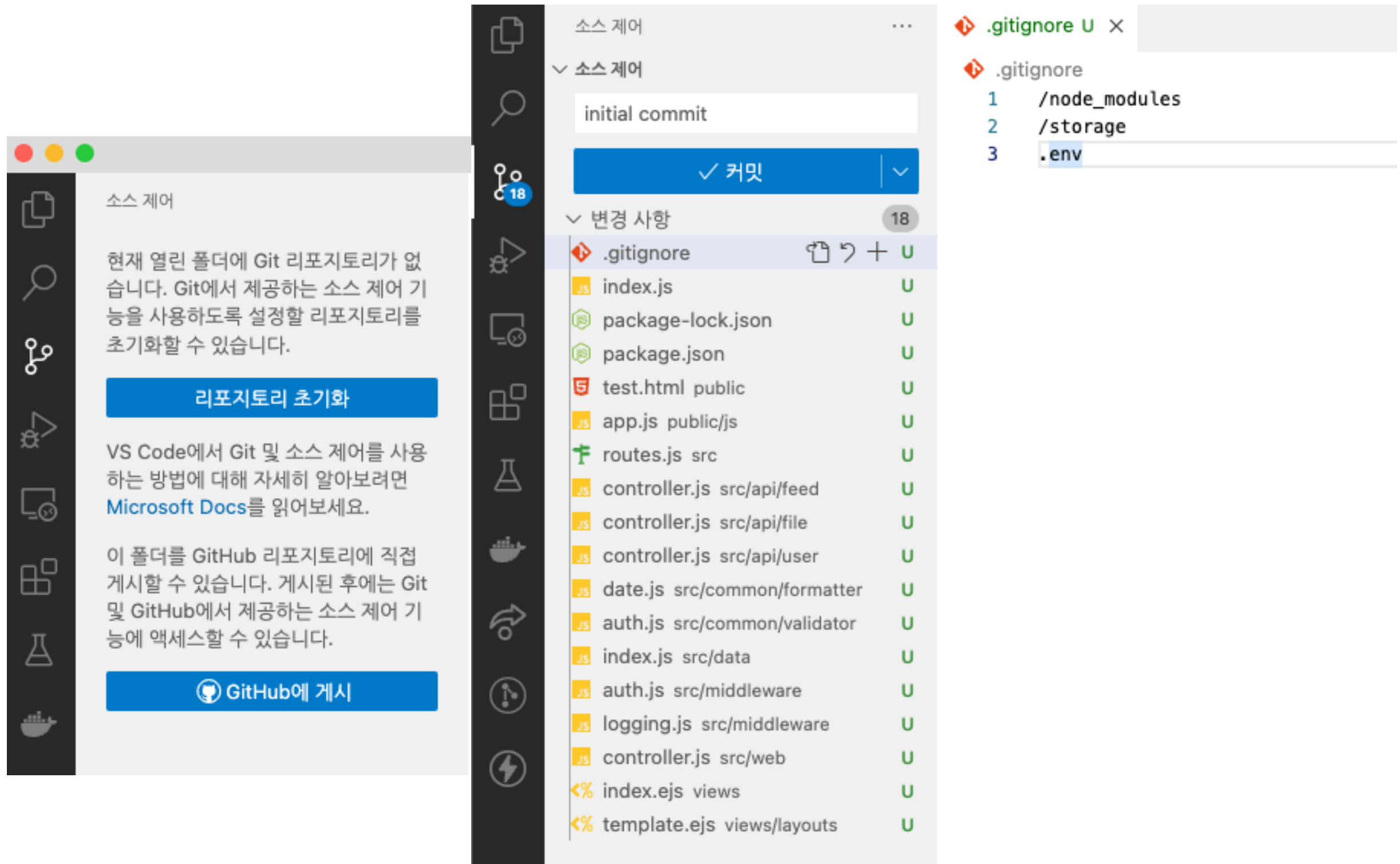
# git 설정

## .ignore 파일 만들기

- 추적하지 않아야 할 파일이나 디렉토리를 지정
- 의존성 라이브러리
- 업로드 데이터
- 로그 데이터
- 암호 키 등



# 파일 추가 및 최초 commit



소스 제어

현재 열린 폴더에 Git 리포지토리가 없습니다. Git에서 제공하는 소스 제어 기능을 사용하도록 설정할 리포지토리를 초기화할 수 있습니다.

**리포지토리 초기화**

VS Code에서 Git 및 소스 제어를 사용하는 방법에 대해 자세히 알아보려면 [Microsoft Docs](#)를 읽어보세요.

이 폴더를 GitHub 리포지토리에 직접 게시할 수 있습니다. 게시된 후에는 Git 및 GitHub에서 제공하는 소스 제어 기능에 액세스할 수 있습니다.

**GitHub에 게시**

소스 제어

initial commit

✓ 커밋

변경 사항 18

- .gitignore
- index.js
- package-lock.json
- package.json
- test.html public
- app.js public/js
- routes.js src
- controller.js src/api/feed
- controller.js src/api/file
- controller.js src/api/user
- date.js src/common/formatter
- auth.js src/common/validator
- index.js src/data
- auth.js src/middleware
- logging.js src/middleware
- controller.js src/web
- index.ejs views
- template.ejs views/layouts

.gitignore U X

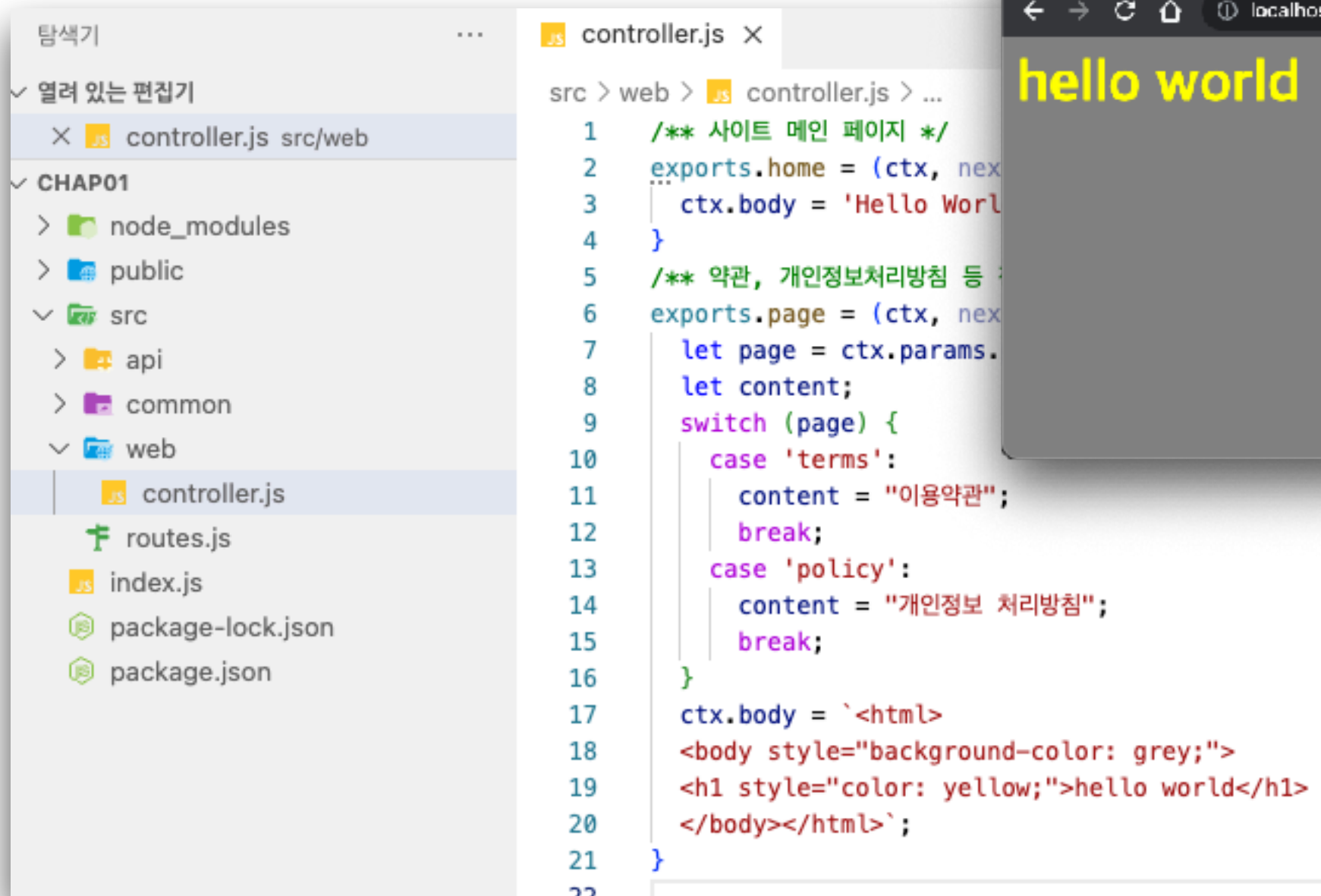
.gitignore

```
1 /node_modules
2 /storage
3 .env
```

**template engine**

# template engine 이 필요한 이유

## 일반적인 적정 content를 출력하는 방법



# template engine

## 장, 단점

- 효율적으로 html 코드를 관리할 수 있다.
- 유지보수가 편하고
- 재사용성이 높다

# template engine 설치

## ejs 설치 및 설정

```
npm install koa-ejs
```

```
const Koa = require('koa');
const Router = require('@koa/router');
const bodyParser = require('koa-bodyparser');
const render = require('koa-ejs');
const path = require('path');
```

```
... .
```

```
// 라우터 설정
```

```
router.use(require('./src/routes').routes());
app.use(router.routes());
app.use(router.allowedMethods());
```

```
// EJS 템플릿엔진
```

```
render(app, {
  layout: null,
  root: path.join(__dirname, '/views'),
  viewExt: 'ejs', cache: false,
});
```

```
... .
```



# ejs 사용하기

## 정적 content를 출력하는 방법 (뷰)

탐색기

...

▽ 열려 있는 편집기

× <% index.ejs views

▽ CHAP01

> node\_modules

> public

> src

▽ views

| <% index.ejs

index.js

package-lock.json

package.json

<% index.ejs ×

views > <% index.ejs > html > body

1 <!DOCTYPE html>

2 <html lang="ko">

3 <head>

4 <meta charset="UTF-8">

5 <meta http-equiv="X-UA-Compatible" content="IE=edge">

6 <meta name="viewport" content="width=device-width, initial-scale=1.0">

7 <title>각종 페이지</title>

8 </head>

9 <body>

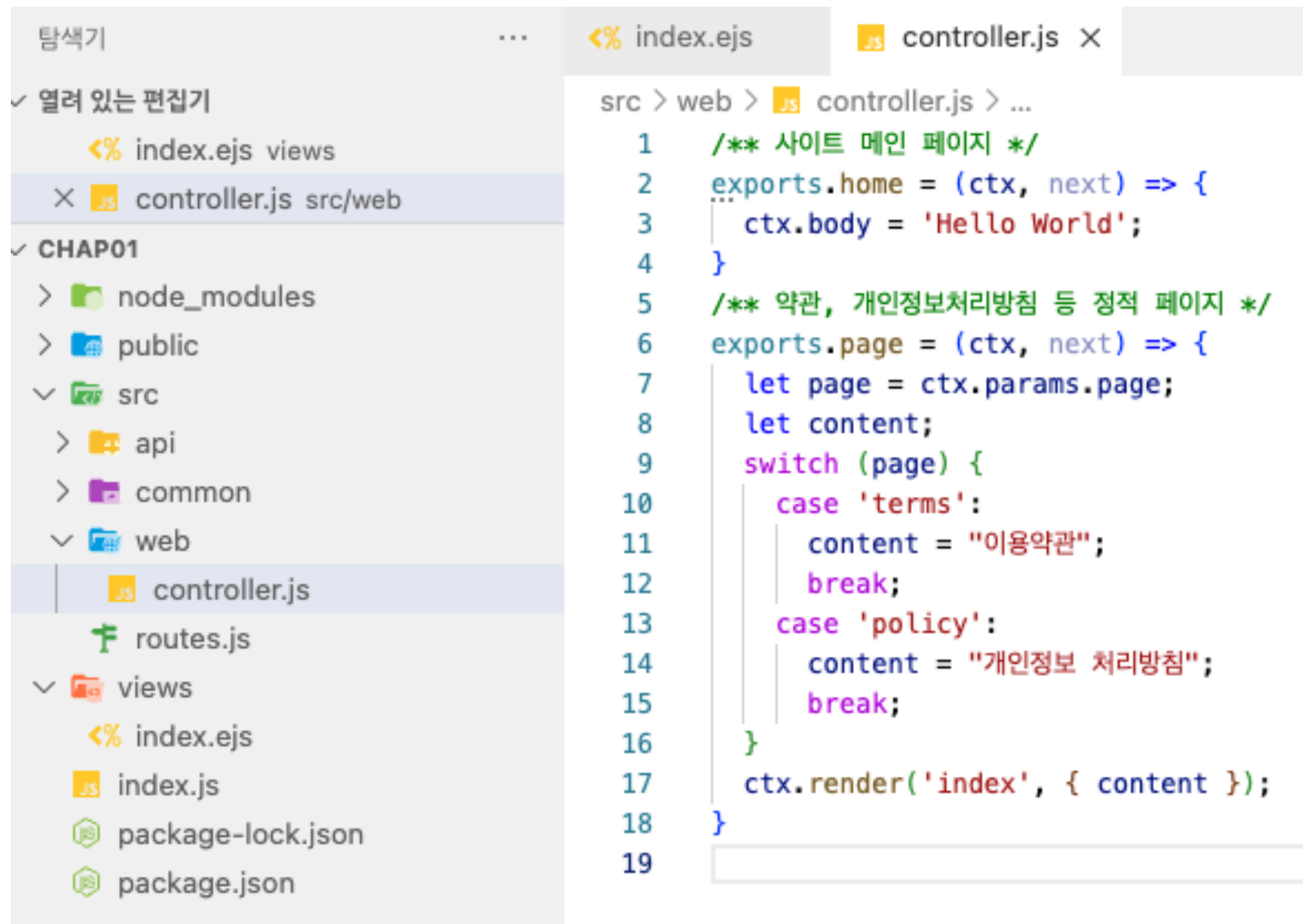
10     페이지의 내용

11 </body>

12 </html>

# ejs 사용하기

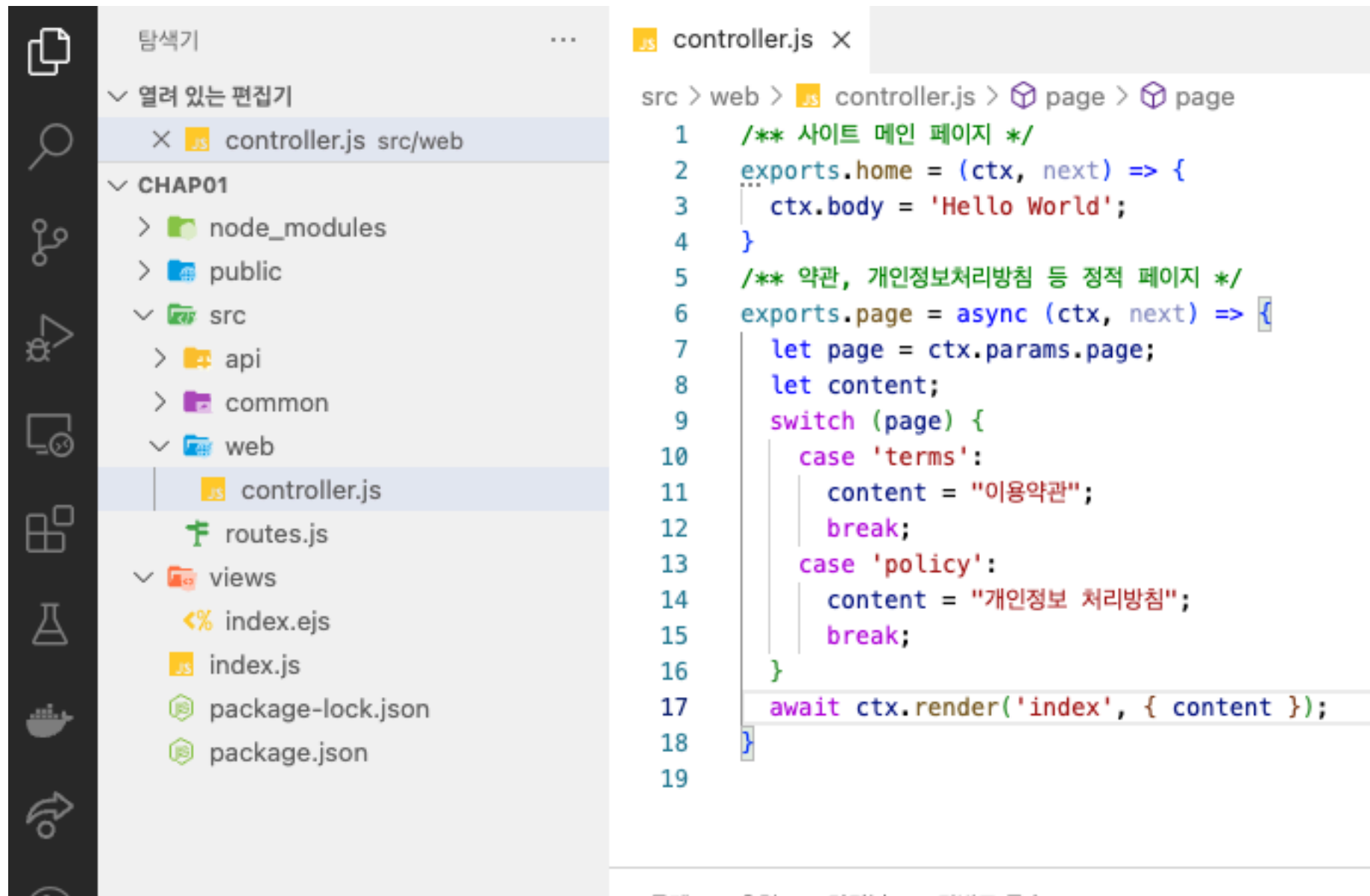
## 동적 content를 출력하는 방법 (서버)



```
src > web > controller.js > ...
1  /** 사이트 메인 페이지 */
2  exports.home = (ctx, next) => {
3    ctx.body = 'Hello World';
4  }
5  /** 약관, 개인정보처리방침 등 정적 페이지 */
6  exports.page = (ctx, next) => {
7    let page = ctx.params.page;
8    let content;
9    switch (page) {
10     case 'terms':
11       content = "이용약관";
12       break;
13     case 'policy':
14       content = "개인정보 처리방침";
15       break;
16   }
17   ctx.render('index', { content });
18 }
19
```

# ejs 사용하기

## 동적 content를 출력하는 방법 (뷰)

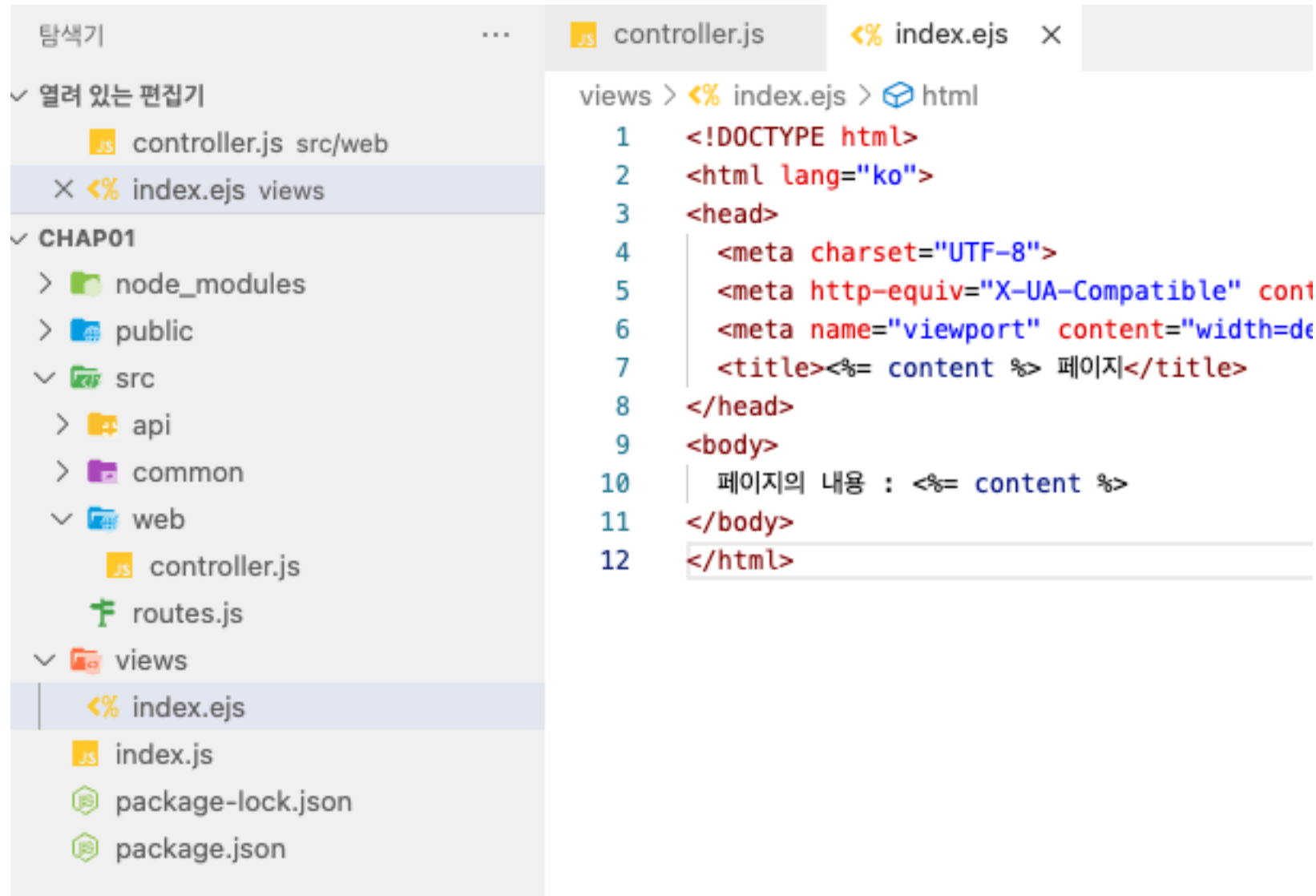


The screenshot shows the VS Code interface. On the left, the file explorer displays the project structure. The 'src/web' directory is expanded, showing 'controller.js' as the selected file. On the right, the code editor shows the content of 'controller.js'. The code defines two export functions: 'home' and 'page'. The 'page' function is an async function that takes 'ctx' and 'next' as arguments. It extracts the 'page' parameter from 'ctx.params', sets 'content' based on a switch statement (handling 'terms' and 'policy'), and then uses 'ctx.render' to render the 'index' view with the 'content' object.

```
controller.js
src > web > controller.js > page > page
1  /** 사이트 메인 페이지 */
2  exports.home = (ctx, next) => {
3    ctx.body = 'Hello World';
4  }
5  /** 약관, 개인정보처리방침 등 정적 페이지 */
6  exports.page = async (ctx, next) => {
7    let page = ctx.params.page;
8    let content;
9    switch (page) {
10     case 'terms':
11       content = "이용약관";
12       break;
13     case 'policy':
14       content = "개인정보 처리방침";
15       break;
16   }
17   await ctx.render('index', { content });
18 }
19
```

# ejs 사용하기

## 동적 content를 출력하는 방법 (뷰)

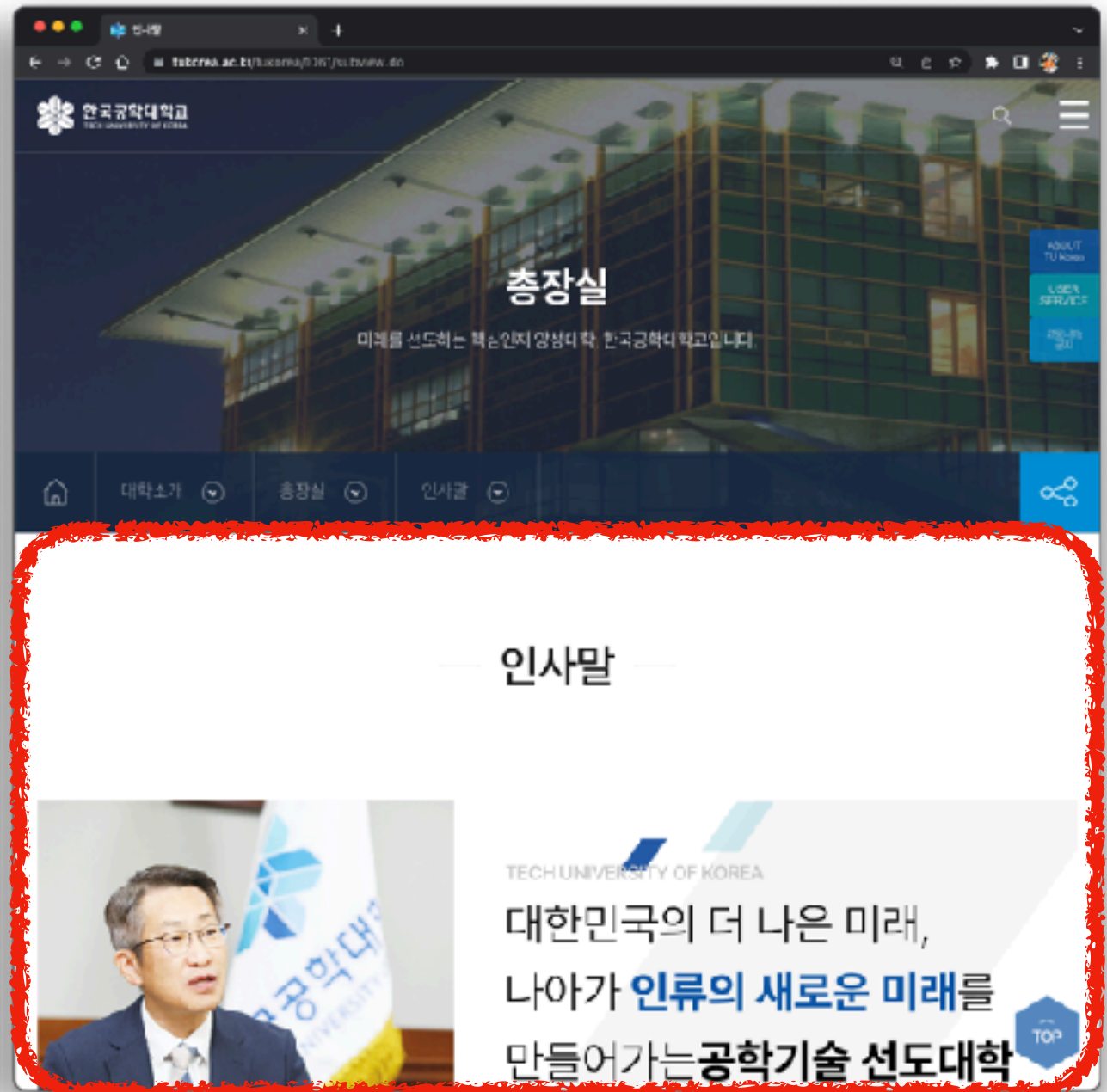
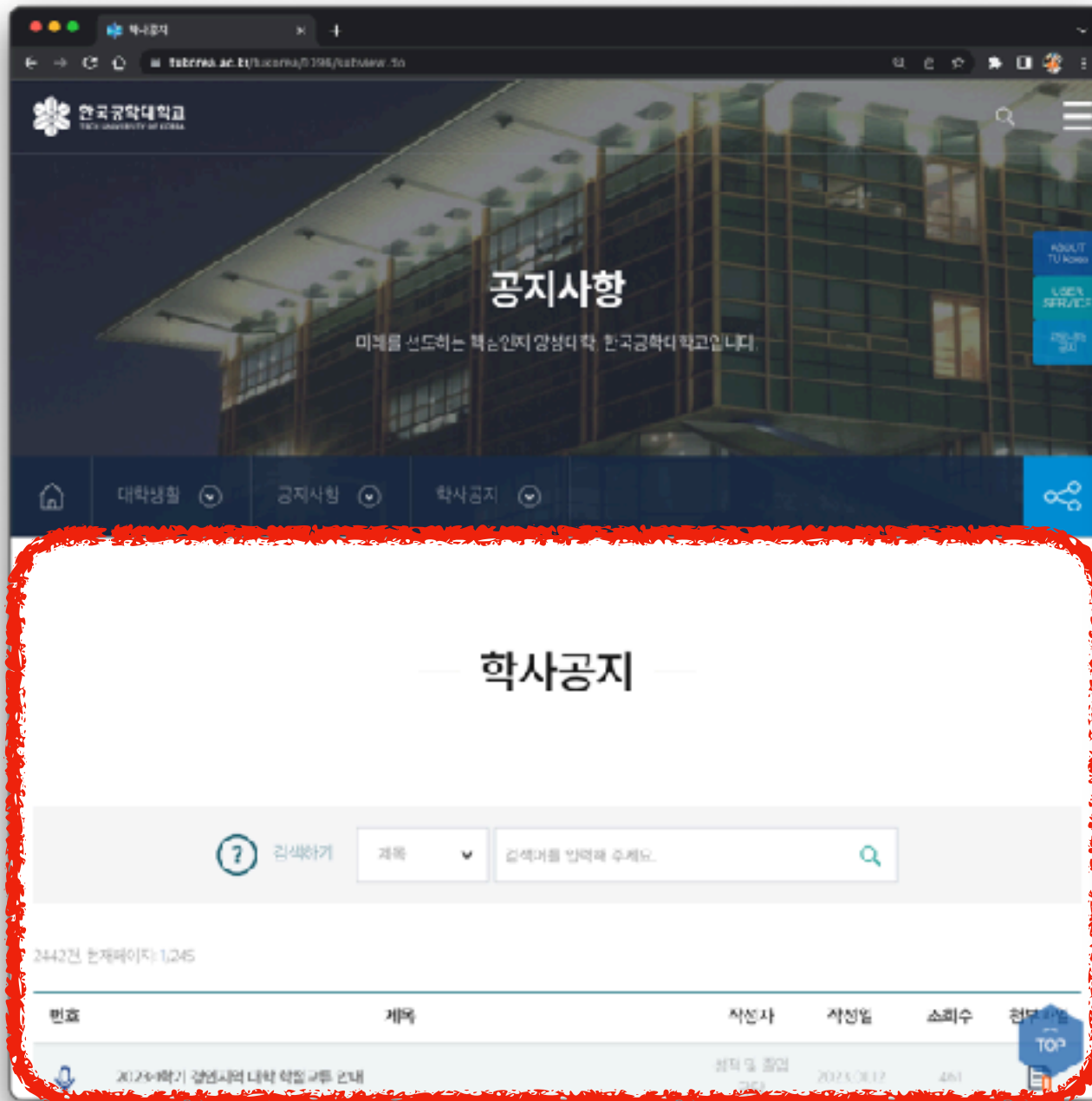


The screenshot shows a code editor interface. On the left is a file explorer with a tree view. The tree is expanded to show the 'views' directory, where 'index.ejs' is selected. Above the tree, there are tabs for 'controller.js' and 'index.ejs'. The main editor area on the right displays the content of 'index.ejs', which is an HTML template. The template includes a DOCTYPE declaration, a lang attribute set to 'ko', a head section with meta tags for charset, http-equiv, and viewport, and a title that uses an EJS expression to output the value of 'content'. The body section also uses an EJS expression to output the value of 'content'.

```
views > <% index.ejs > html
1  <!DOCTYPE html>
2  <html lang="ko">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" cont
6      <meta name="viewport" content="width=de
7      <title><%= content %> 페이지</title>
8  </head>
9  <body>
10     페이지의 내용 : <%= content %>
11 </body>
12 </html>
```

# 페이지 구조 살펴보기

공통인 부분과 변화는 부분



# layout 사용하기

## 설정부분

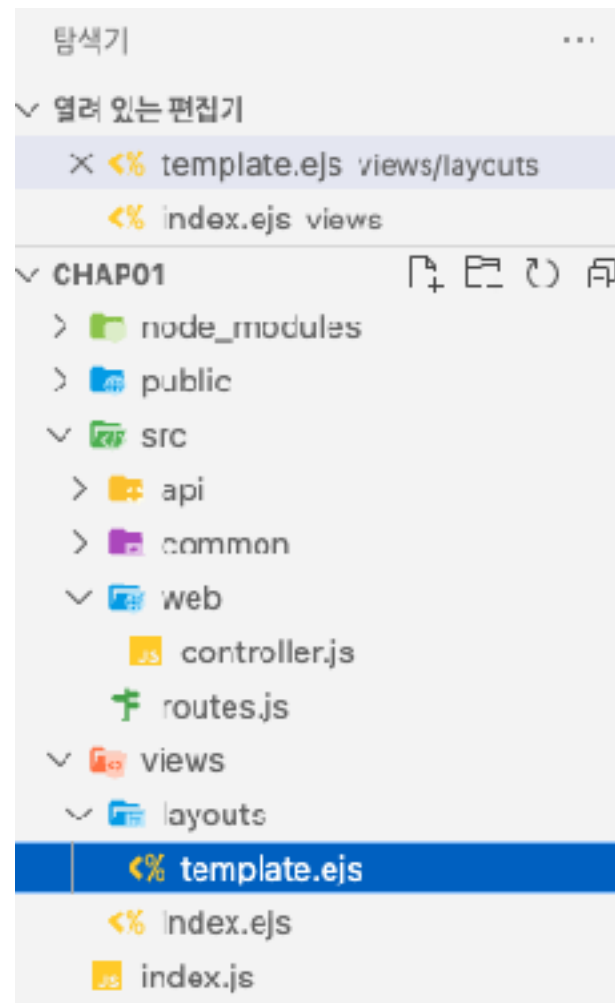
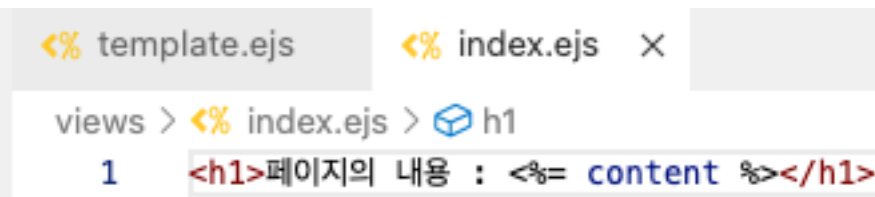
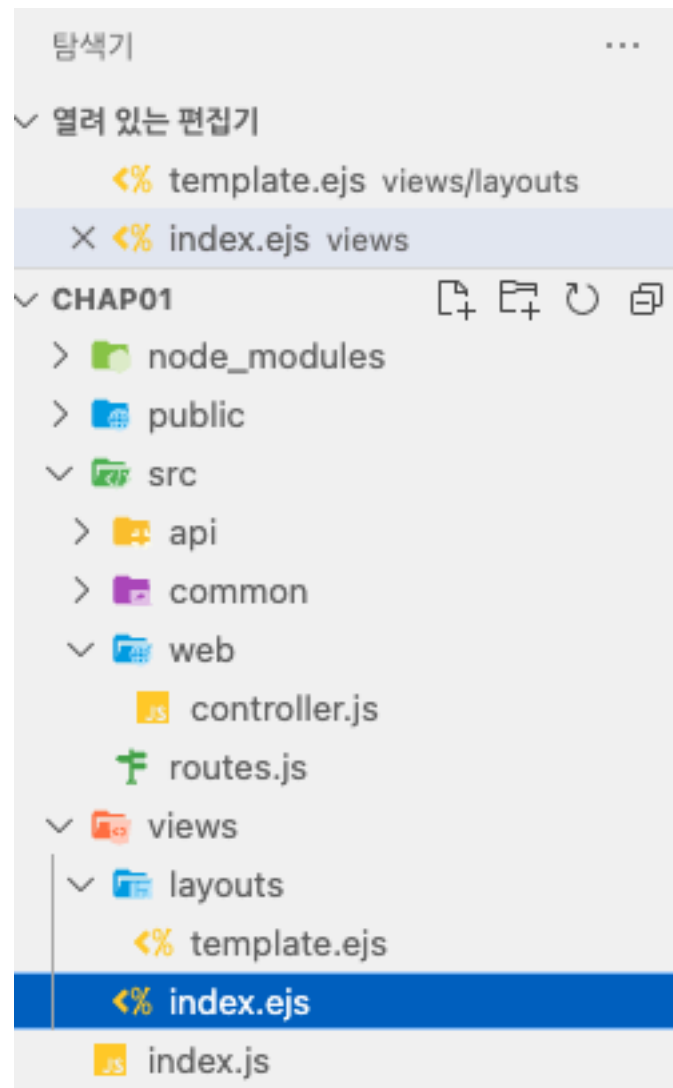
```
const Koa = require('koa');
const Router = require('@koa/router');
const bodyParser = require('koa-bodyparser');
const render = require('koa-ejs');
const path = require('path');

...
// 라우터 설정
router.use(require('./src/routes').routes());
app.use(router.routes());
app.use(router.allowedMethods());

// EJS 템플릿엔진
render(app, {
  // layout: false,
  layout: 'layouts/template',
  root: path.join(__dirname, '/views'),
  viewExt: 'ejs', cache: false,
});
...
```

# layout 사용하기

## layout 파일 생성 및 사용



**middleware**



# middleware

## 정의

- 일반적인 서비스와 기능을 애플리케이션에 제공하는 소프트웨어
- 데이터 관리, 애플리케이션 서비스, 메시징, 인증 및 API관리 등에 활용

“저희는 통신에 암호화가 필요합니다.”

-클라이언트

# 반복되는 코드

## 암호화된 request를 매번 복호화 하는 예제

탐색기

...

✓ 열려 있는 편집기 10(가) 저장되지 않음

◀% template.ejs views/layouts

◀% index.ejs views

● JS controller.js src/api/feed

✓ CHAP01

> node\_modules

> public

✓ src

✓ api

✓ feed

JS controller.js

> user

> common

> web

✚ routes.js

> views

JS index.js

package-lock.json

package.json

◀% template.ejs

◀% index.ejs

JS controller.js ●

src > api > feed > JS controller.js > update > update

```
1  /** 전체 피드보기 */
2  exports.index = (ctx, next) => {
3    let query = ctx.query
4    ctx.body = query;
5  }
6  /** 새 피드 작성 처리 */
7  exports.store = (ctx, next) => {
8    let bodyString = ctx.request.body
9    // 복호화
10   let body = decipherString(bodyString);
11
12   ctx.body = body;
13 }
14 /** 피드 상세보기 */
15 exports.show = (ctx, next) => {
16   let id = ctx.params.id;
17   ctx.body = `${id} 피드 상세`;
18 }
19 /** 피드 수정 */
20 exports.update = (ctx, next) => {
21   let bodyString = ctx.request.body
22   // 복호화
23   let body = decipherString(bodyString);
24 }
```

# middleware의 활용

user의 요청과 route 사이의 매개역할



로깅

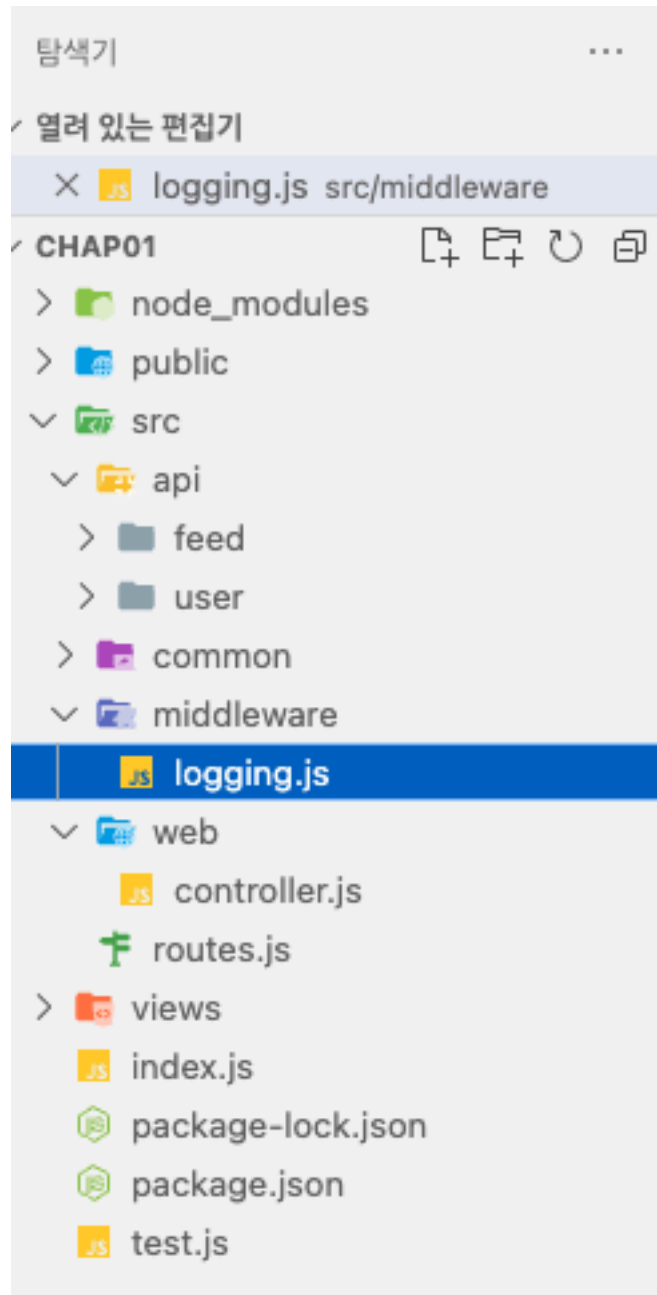
parser

복호화

인증

# middleware 만들어보기

## 간단한 로깅 예제

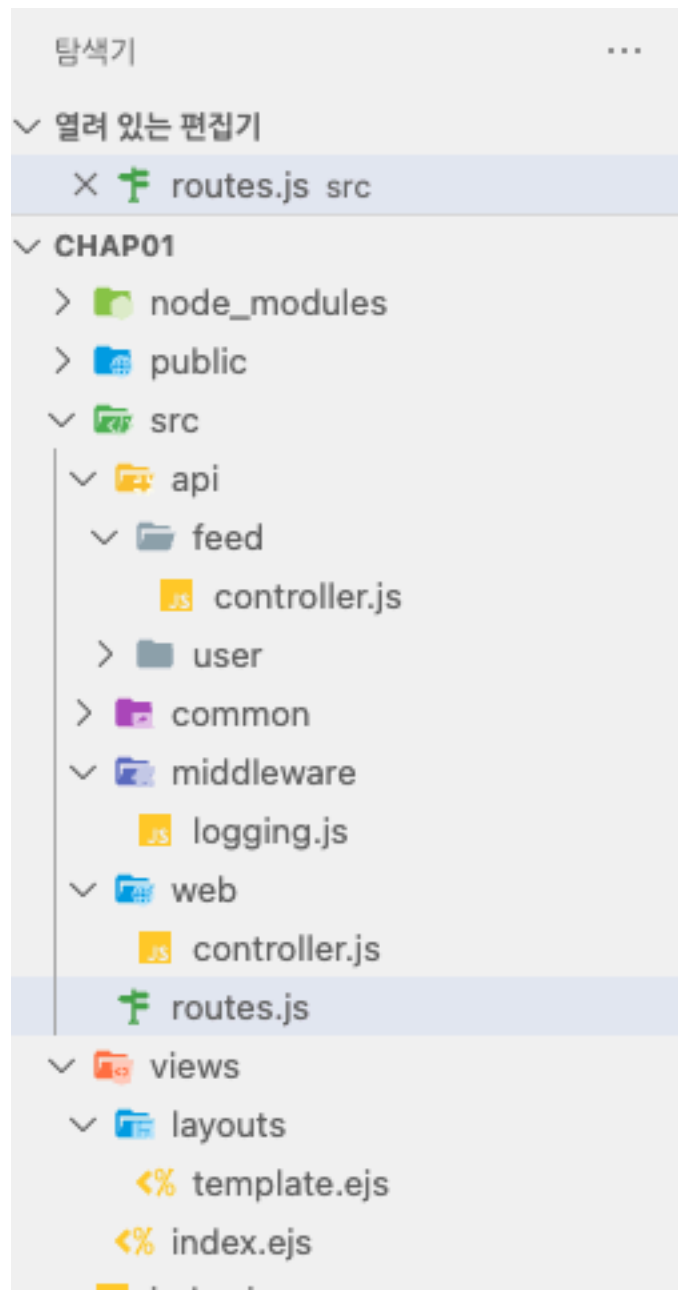


```
logging.js ×

src > middleware > logging.js > myLogging > myLogging
1  exports.myLogging = async (ctx, next) => {
2    let clientIp = ctx.request.ip;
3    console.log(`${clientIp.replace("::ffff:", "")} 주소에서 요청 : ${ctx.originalUrl}`);
4    await next();
5  }
```

# middleware 적용하기

## 일부 라우트에 적용



```
routes.js ×
src > routes.js > ...
1
2  const Router = require('@koa/router');
3  const router = new Router();
4
5  const { myLogging } = require('./middleware/logging')
6
7
8  const webController = require('./web/controller');
9  const apiUserController = require('./api/user/controller');
10 const apiFeedController = require('./api/feed/controller');
11
12 router.get('/', myLogging, webController.home);
13 router.get('/page/:page', myLogging, webController.page);
14
15 router.get('/api/user/:id', apiUserController.info);
16
17 router.get('/api/feed', apiFeedController.index);
18 router.post('/api/feed', apiFeedController.store);
19 router.get('/api/feed/:id', apiFeedController.show);
20 router.put('/api/feed/:id', apiFeedController.update);
21 router.delete('/api/feed/:id', apiFeedController.delete);
22
23 module.exports = router;
```

# middleware 적용하기

## 선언 하위 라우트에 일괄 적용

탐색기

...

열려 있는 편집기

× routes.js src

CHAP01

> node\_modules

> public

src

api

feed

controller.js

user

common

middleware

logging.js

web

controller.js

routes.js

views

layouts

template.ejs

index.ejs

index.js

package-lock.json

routes.js ×

src > routes.js > ...

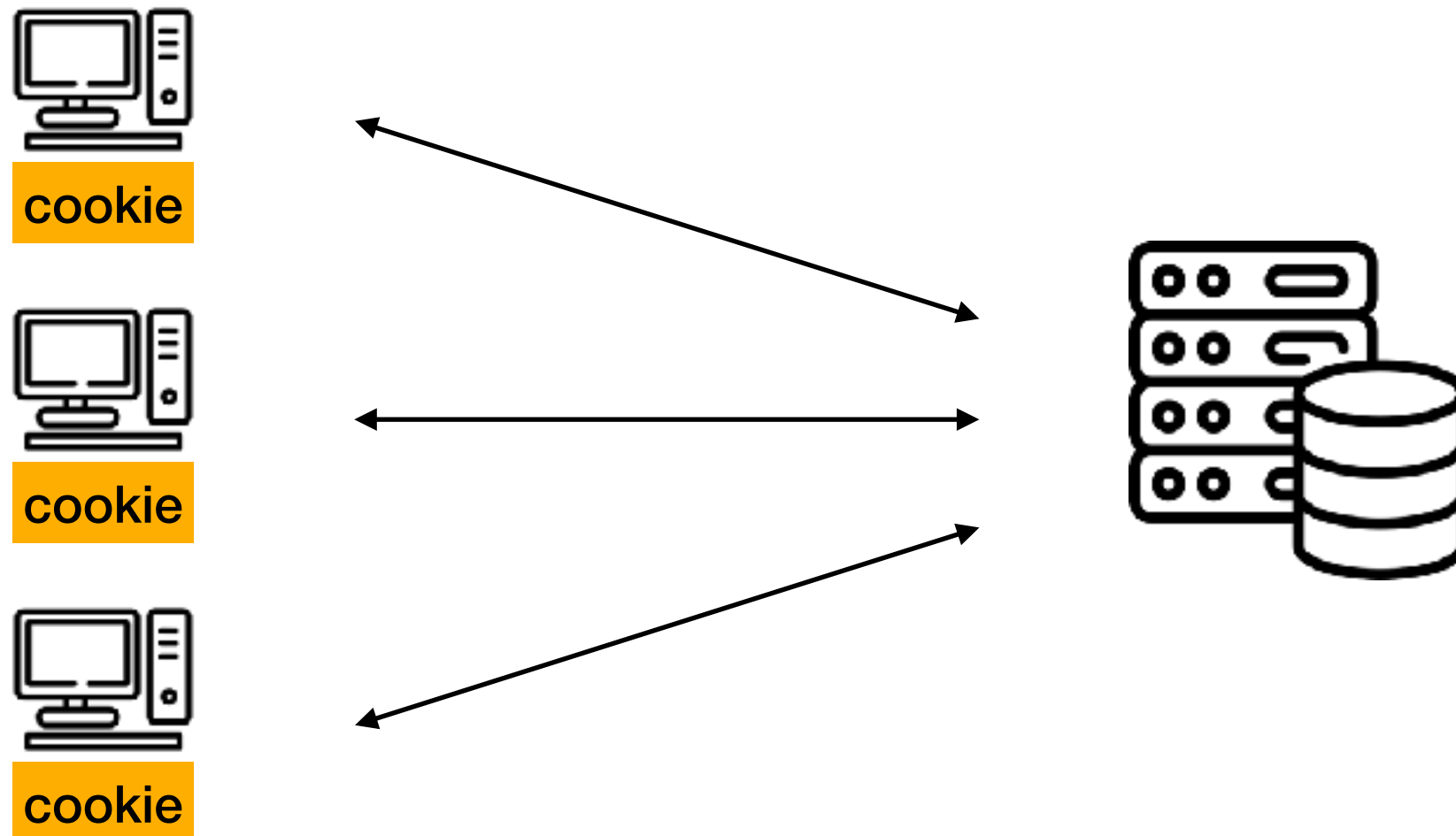
```
1
2  const Router = require('@koa/router');
3  const router = new Router();
4
5  const { myLogging } = require('./middleware/logging')
6
7
8  const webController = require('./web/controller');
9  const apiUserController = require('./api/user/controller');
10 const apiFeedController = require('./api/feed/controller');
11
12 router.use(myLogging);
13
14 router.get('/', webController.home);
15 router.get('/page/:page', webController.page);
16
17 router.get('/api/user/:id', apiUserController.info);
18
19 router.get('/api/feed', apiFeedController.index);
20 router.post('/api/feed', apiFeedController.store);
21 router.get('/api/feed/:id', apiFeedController.show);
22 router.put('/api/feed/:id', apiFeedController.update);
23 router.delete('/api/feed/:id', apiFeedController.delete);
24
25 module.exports = router;
```

# Authentication



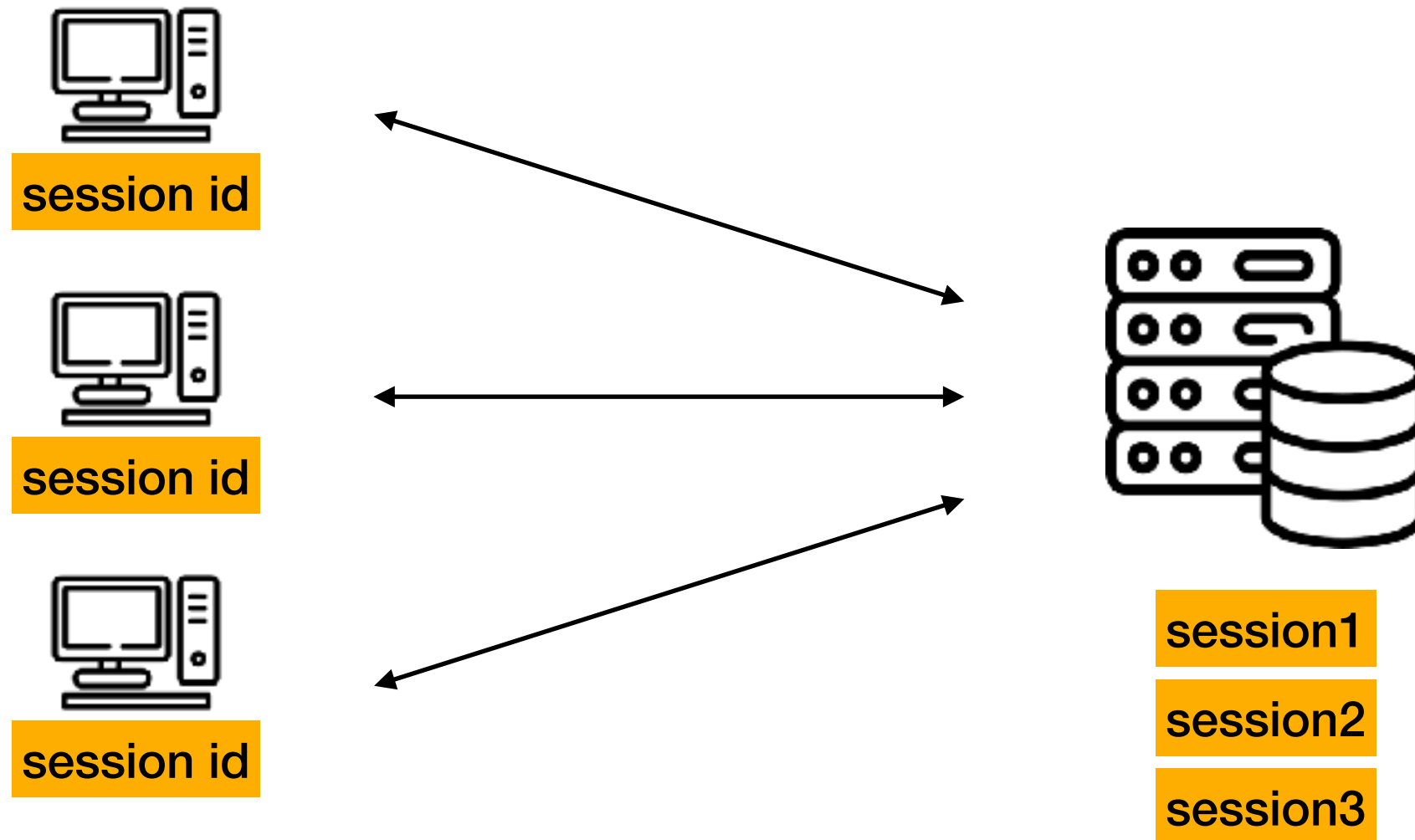
# Authentication

cookie 를 활용한 인증 방식



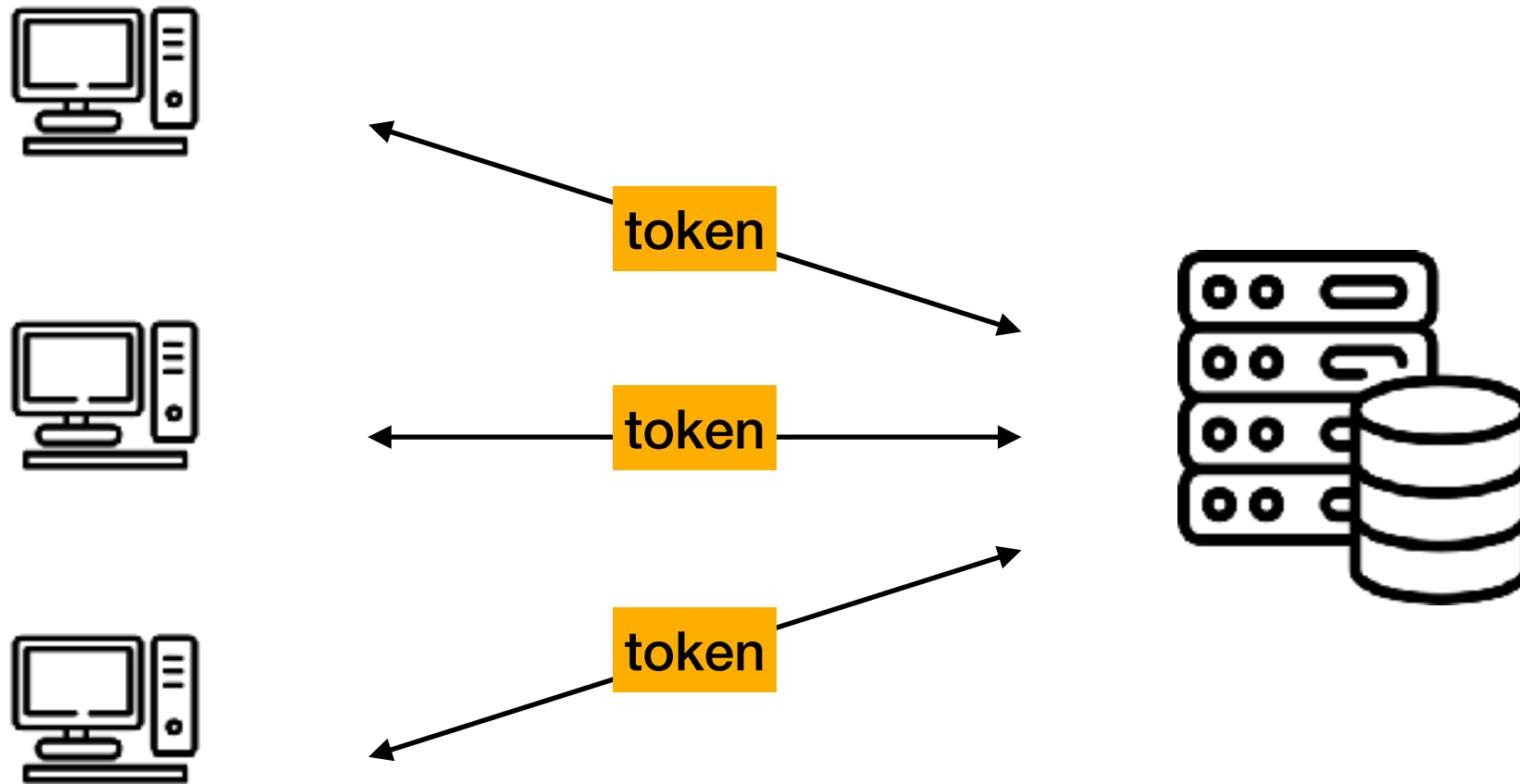
# Authentication

session 을 활용한 인증 방식



# Authentication

token 을 활용한 인증 방식



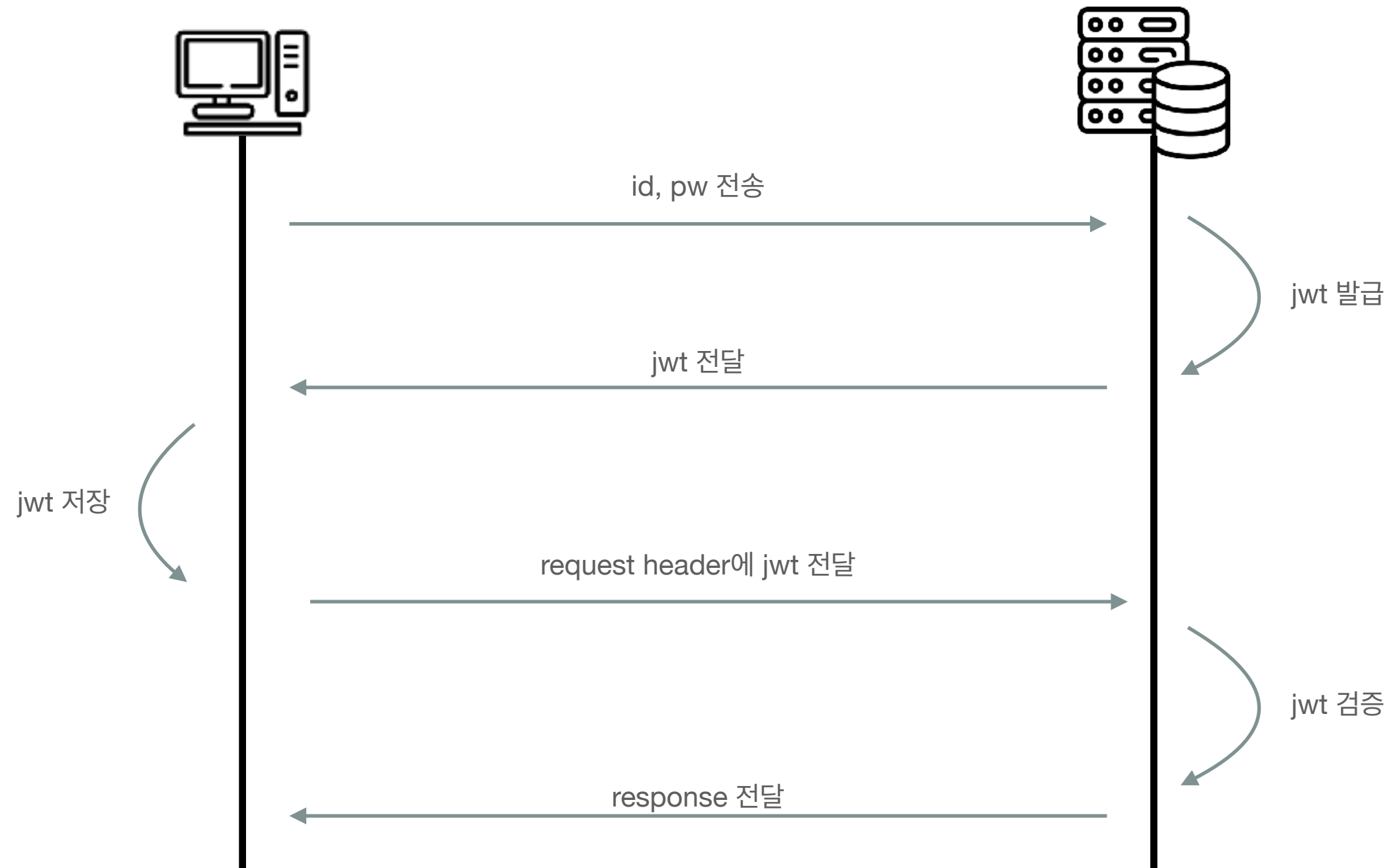
# JWT

## json web token

- stateless 환경에서 사용자 데이터를 주고 받을 수 있다.
- 토큰 내부에 권한 정보나 서비스 정보를 포함할 수 있다.
- 구성은 Header, Payload, Signature 로 구성된다.
- Header
  - 사용한 해시 타입과 알고리즘의 종류
- Payload
  - 사용자 정보나 권한등의 정보
- Signature
  - 암호화에 대한 전자서명

# JWT 소개

## 토큰의 실행 flow



# jwt

## jwt 설치

npm install jsonwebtoken

```
const Koa = require('koa');
const Router = require('@koa/router');
const bodyParser = require('koa-bodyparser');
const render = require('koa-ejs');
const path = require('path');
```

... .

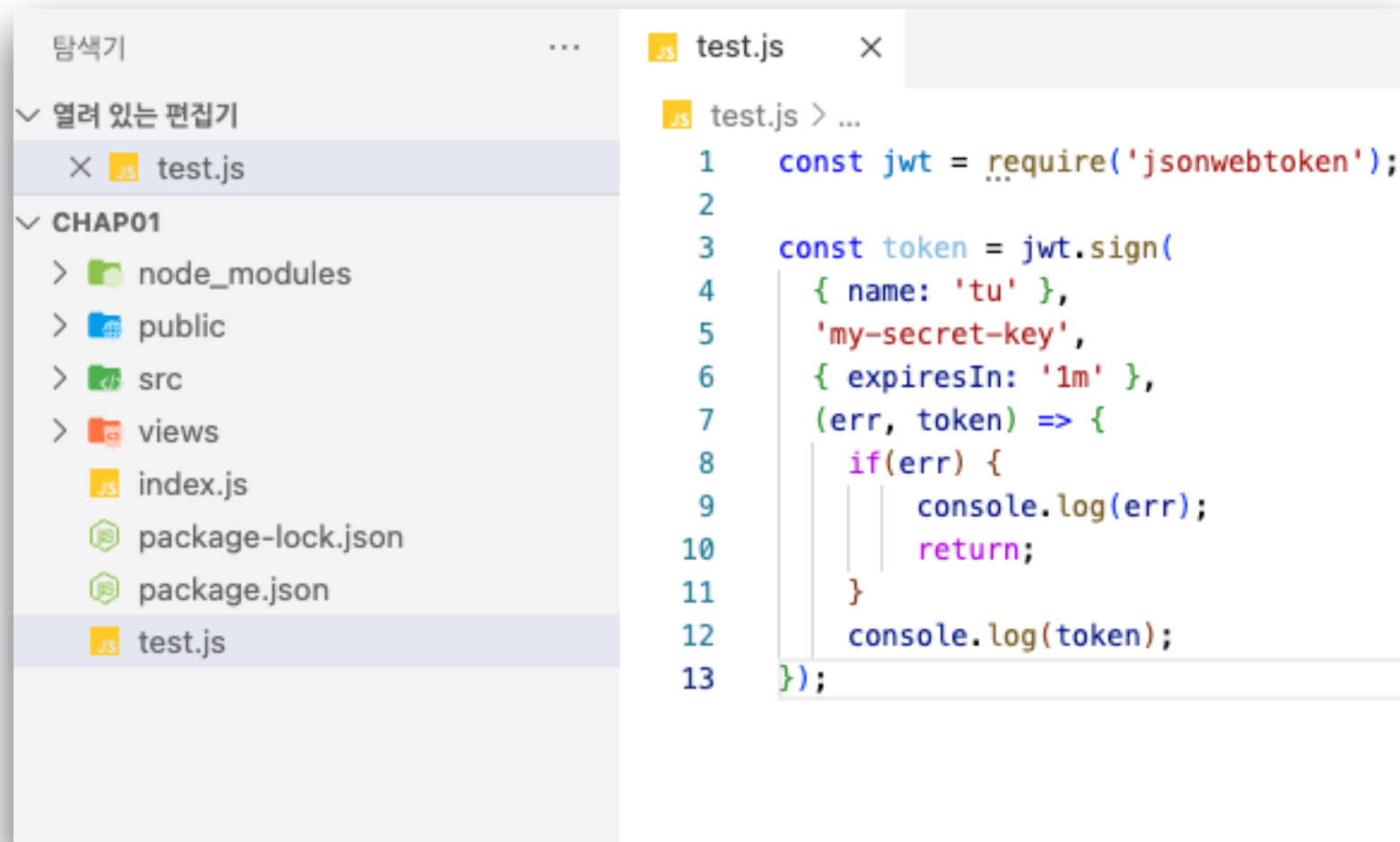
```
// 라우터 설정
router.use(require('./src/routes').routes());
app.use(router.routes());
app.use(router.allowedMethods());
```

```
// EJS 템플릿엔진
render(app, {
  layout: null,
  root: path.join(__dirname, '/views'),
  viewExt: 'ejs', cache: false,
});
```

... .

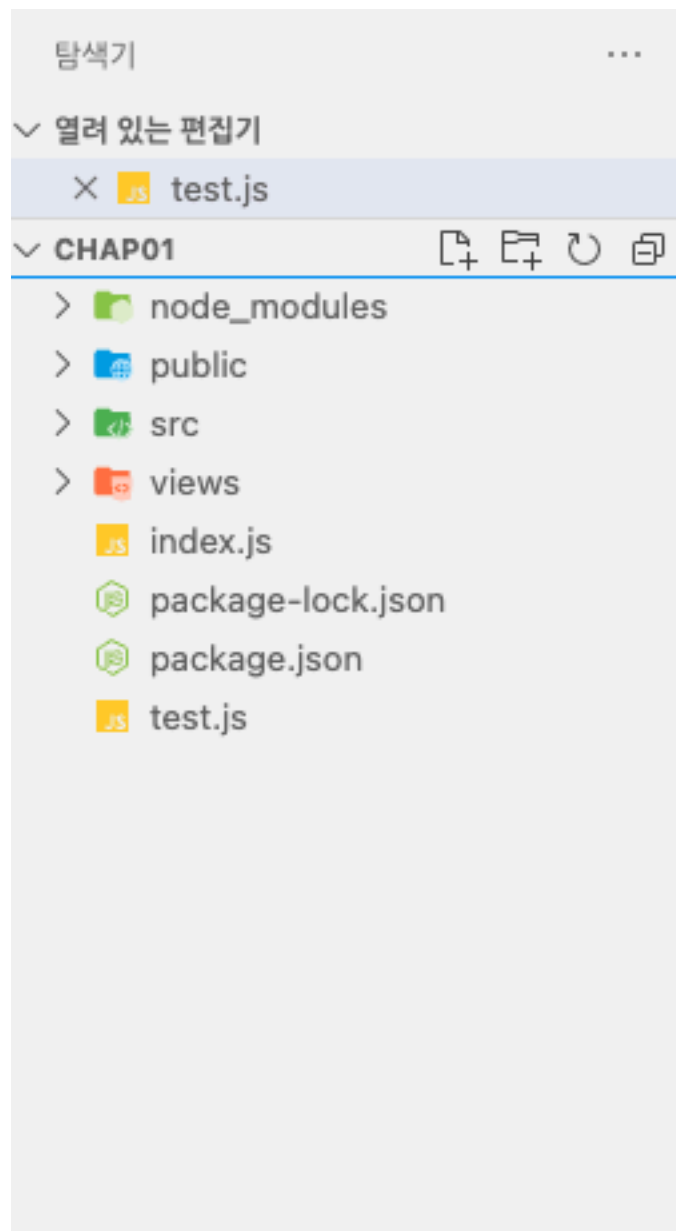
# jwt 실행

## jwt 토큰 생성 예제



# jwt 실행

## jwt 토큰 검증 예제



```
test.js > ...  
1  const jwt = require('jsonwebtoken');  
2  
3  // const token = jwt.sign(  
4  //   { name: 'tu' },  
5  //   'my-secret-key',  
6  //   { expiresIn: '1m' },  
7  //   (err, token) => {  
8  //     if(err) {  
9  //       console.log(err);  
10 //       return;  
11 //     }  
12 //     console.log(token);  
13 //   });  
14  
15  
16 jwt.verify(token, 'my-secret-key', (error, decoded) => {  
17   if(error) {  
18     console.error(error);  
19     return;  
20   }  
21   console.log(decoded);  
22 });
```



# 토큰 생성 함수 생성

탐색기

열려 있는 편집기

controller.js src/api/user

CHAP01

node\_modules

public

src

api

feed

user

controller.js

common

middleware

web

routes.js

views

index.js

package-lock.json

package.json

test.js

controller.js X

src > api > user > controller.js > generteToken

```
4  /** 해당 id의 회원정보들 */
5  exports.info = (ctx, next) => {
6    let id = ctx.params.id;
7    ctx.body = `${id} 회원에 대한 정보`;
8  }
9
10 exports.register = async (ctx, next) => {
11   // 회원가입 처리 모듈
12
13   let token = await generteToken({name: 'my-name'});
14   ctx.body = token;
15 }
16
17 exports.login = async (ctx, next) => {
18   // 로그인 모듈
19
20   let token = await generteToken({name: 'my-name'});
21   ctx.body = token;
22 }
23
24 /**
25  * jwt 토큰 생성
26  * @param {object} payload 추가적으로 저장할 payload
27  * @returns {string} jwt 토큰string
28  */
29 let generteToken = (payload) => {
30   return new Promise((resolve, reject) => {
31     jwt.sign(payload, SECRET_KEY, (error, token) => {
32       if(error) { reject(error); }
33       resolve(token);
34     })
35   })
36 }
```

# 라우트에 추가

탐색기

...

controller.js routes.js

열려 있는 편집기

controller.js src/api/user

routes.js src

CHAP01

node\_modules

public

src

api

feed

user

controller.js

common

middleware

web

routes.js

views

index.js

package-lock.json

package.json

test.js

src > routes.js > ...

```
1
2  const Router = require('@koa/router');
3  const router = new Router();
4
5  const { myLogging } = require('./middleware/logging')
6
7
8  const webController = require('./web/controller');
9  const apiUserController = require('./api/user/controller');
10 const apiFeedController = require('./api/feed/controller');
11
12 router.use(myLogging);
13
14 router.get('/', webController.home);
15 router.get('/page/:page', webController.page);
16
17 router.post('/api/user/register', apiUserController.register);
18 router.post('/api/user/login', apiUserController.login);
19 router.get('/api/user/:id', apiUserController.info);
20
21 router.get('/api/feed', apiFeedController.index);
22 router.post('/api/feed', apiFeedController.store);
23 router.get('/api/feed/:id', apiFeedController.show);
24 router.put('/api/feed/:id', apiFeedController.update);
```

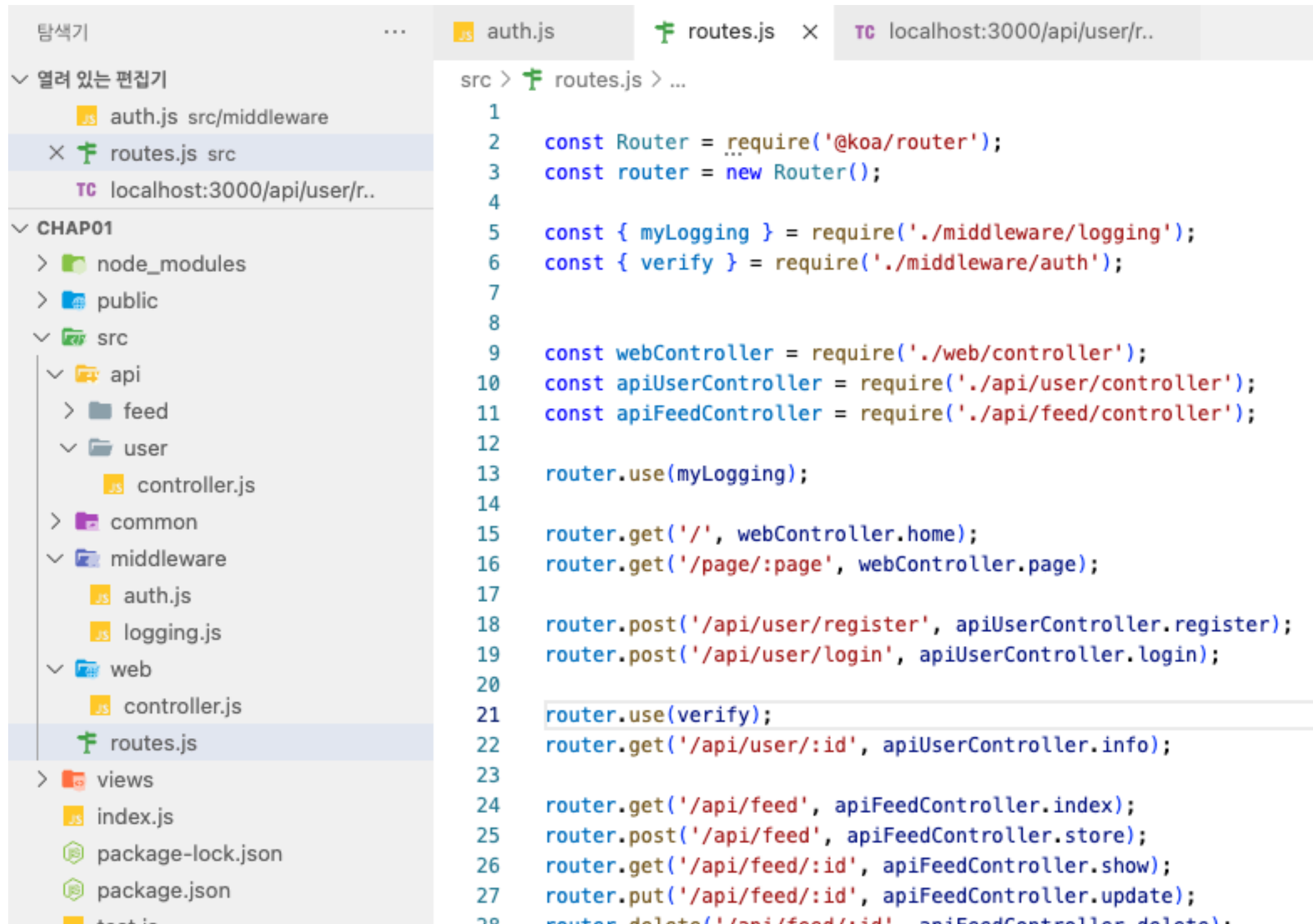
# jwt 검증 예제

## 토큰 검증용 middleware 추가

The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'src' directory containing 'api', 'common', 'middleware', and 'web' subdirectories. The 'middleware' directory is expanded, showing 'auth.js' and 'logging.js'. The code editor shows the 'auth.js' file with the following code:

```
src > middleware > auth.js > verify > verify
1  const jwt = require('jsonwebtoken');
2  const SECRET_KEY = 'my-secret-key';
3
4  exports.verify = async (ctx, next) => {
5    var token = ctx.request.headers['token']
6    jwt.verify(token, SECRET_KEY, (error, decoded) => {
7      if(error) {
8        ctx.body = '로그인을 해야합니다';
9        return;
10     }
11     next();
12   })
13 }
```

# 인증이 필요한 route에 검증 추가



The screenshot shows a code editor with three tabs: `auth.js`, `routes.js`, and `localhost:3000/api/user/r..`. The `routes.js` tab is active, showing the following code:

```
src > routes.js > ...
1
2 const Router = require('@koa/router');
3 const router = new Router();
4
5 const { myLogging } = require('./middleware/logging');
6 const { verify } = require('./middleware/auth');
7
8
9 const webController = require('./web/controller');
10 const apiUserController = require('./api/user/controller');
11 const apiFeedController = require('./api/feed/controller');
12
13 router.use(myLogging);
14
15 router.get('/', webController.home);
16 router.get('/page/:page', webController.page);
17
18 router.post('/api/user/register', apiUserController.register);
19 router.post('/api/user/login', apiUserController.login);
20
21 router.use(verify);
22 router.get('/api/user/:id', apiUserController.info);
23
24 router.get('/api/feed', apiFeedController.index);
25 router.post('/api/feed', apiFeedController.store);
26 router.get('/api/feed/:id', apiFeedController.show);
27 router.put('/api/feed/:id', apiFeedController.update);
28 router.delete('/api/feed/:id', apiFeedController.delete);
```

The left sidebar shows the file explorer with the following structure:

- CHAP01
  - node\_modules
  - public
  - src
    - api
      - feed
      - user
        - controller.js
      - common
    - middleware
      - auth.js
      - logging.js
    - web
      - controller.js
    - routes.js (selected)
  - views
    - index.js
    - package-lock.json
    - package.json

# 토큰 테스트

The screenshot shows a web client interface with a file explorer on the left, a request configuration area in the center, and a response area on the right.

**File Explorer (Left):** Shows a project structure with folders like `node_modules`, `public`, and `src`. The `src` folder is expanded, showing subfolders `api`, `common`, `middleware`, and `web`. The `api` folder is further expanded, showing `feed` and `user` subfolders. The `routes.js` file is selected.

**Request Configuration (Center):**

- Method: GET
- URL: `http://localhost:3000/api/feed`
- Buttons: Send
- Tabs: Query, Headers (selected), Auth, Body, Tests, Pre Run New
- Http Headers section (Raw view):
  - ☒ User-Agent: Thunder Client (https://www.thun
  - ☒ Accept: application/json
  - ☒ token: eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp
  - ☐ header: value

**Response (Right):**

- Status: 200 OK
- Size: 2 Bytes
- Time: 4 ms
- Response tab selected, showing: 1 {}

# 설정파일 분리하기

# 설정 파일 분리하기

## 설치

```
npm install dotenv
```

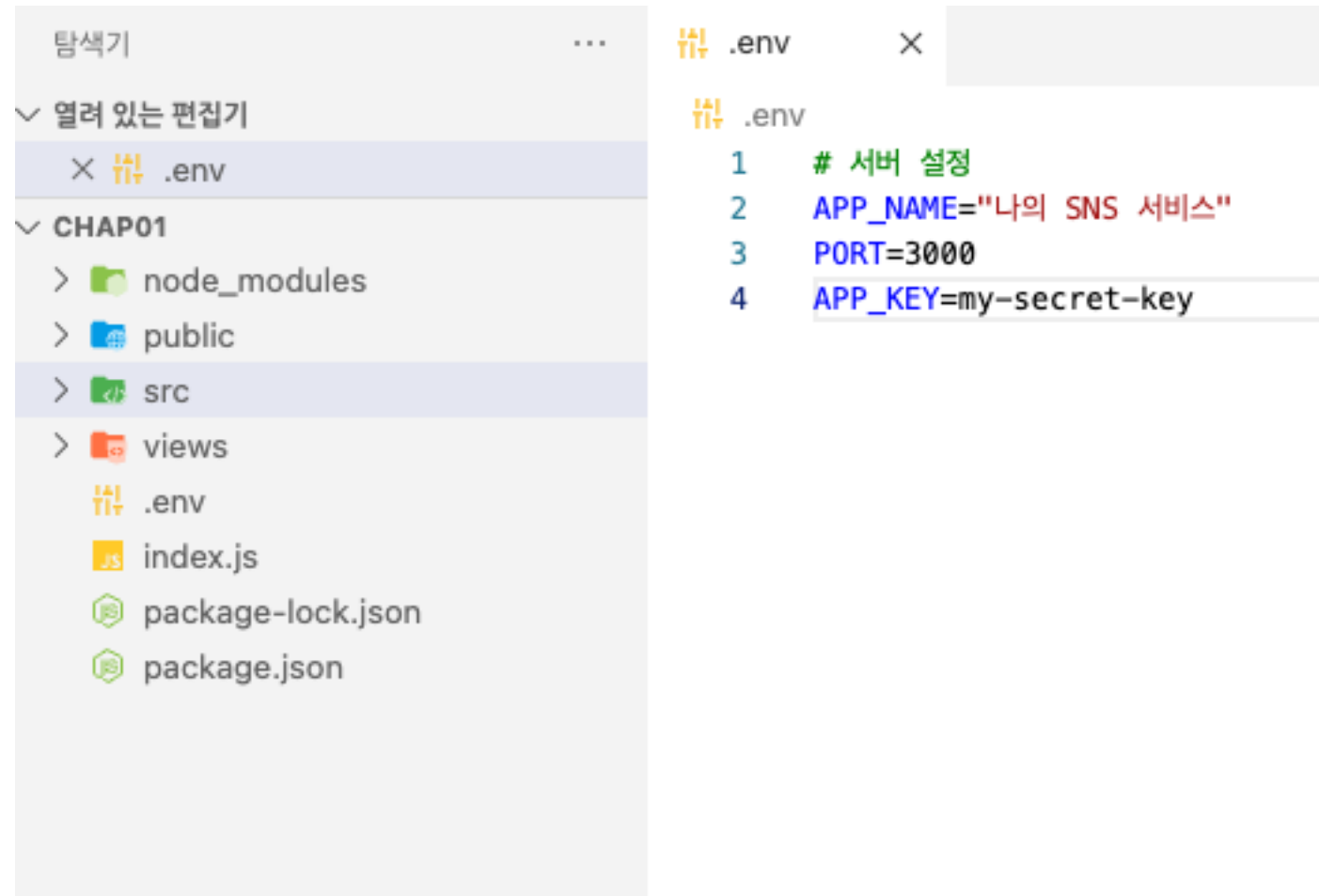
```
require('dotenv').config();
const Koa = require('koa');
const Router = require('@koa/router');
const bodyParser = require('koa-bodyparser');
const render = require('koa-ejs');
const path = require('path');
const app = new Koa();
const router = new Router();

// 서버 실행 포트
const port = process.env.PORT || 3000;

// 바디파서
app.use(bodyParser({formLimit: '5mb'}));
```

# 설정 파일 분리하기

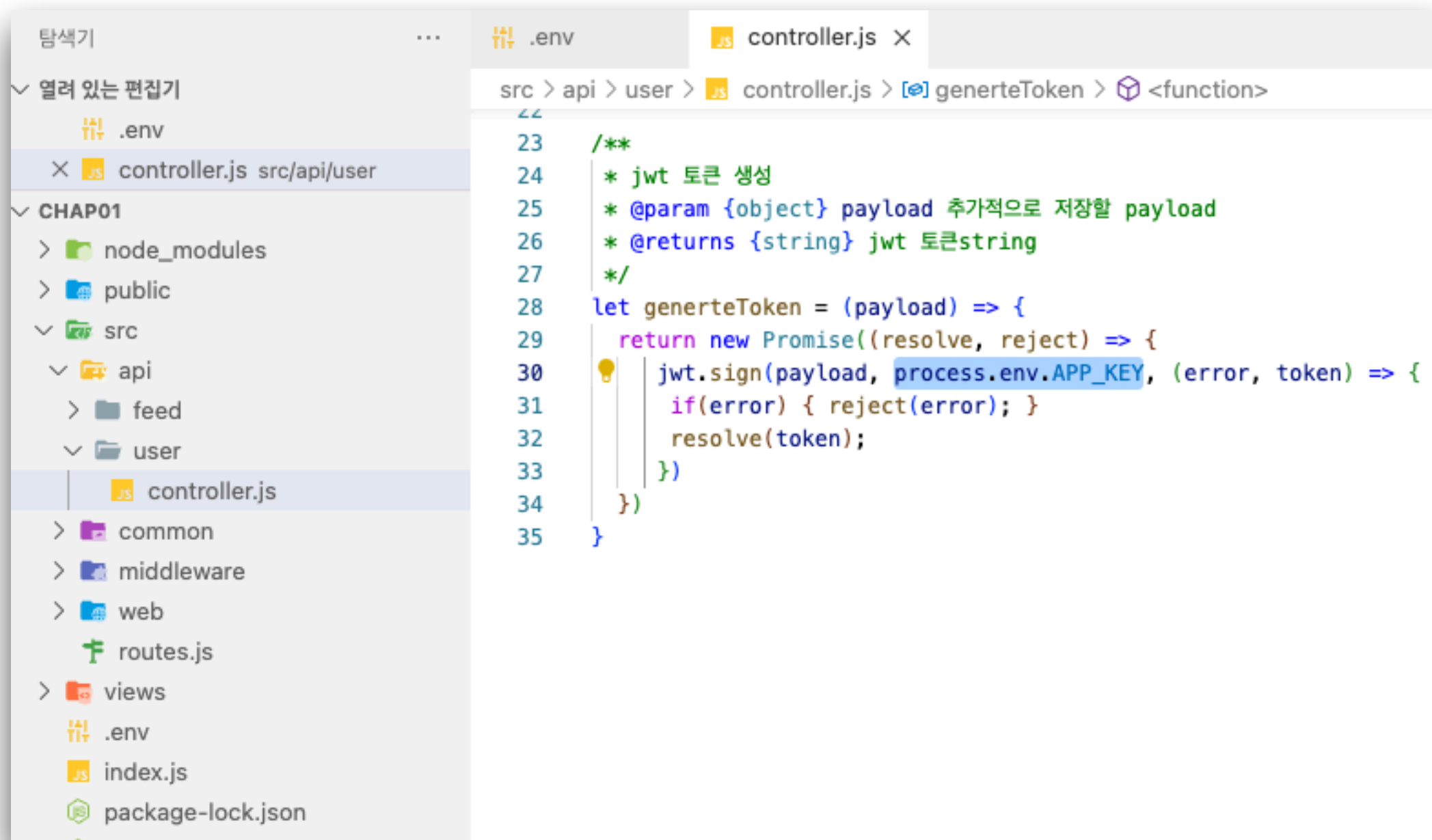
## .env 파일 만들기





# 코드 내 참조 변경하기

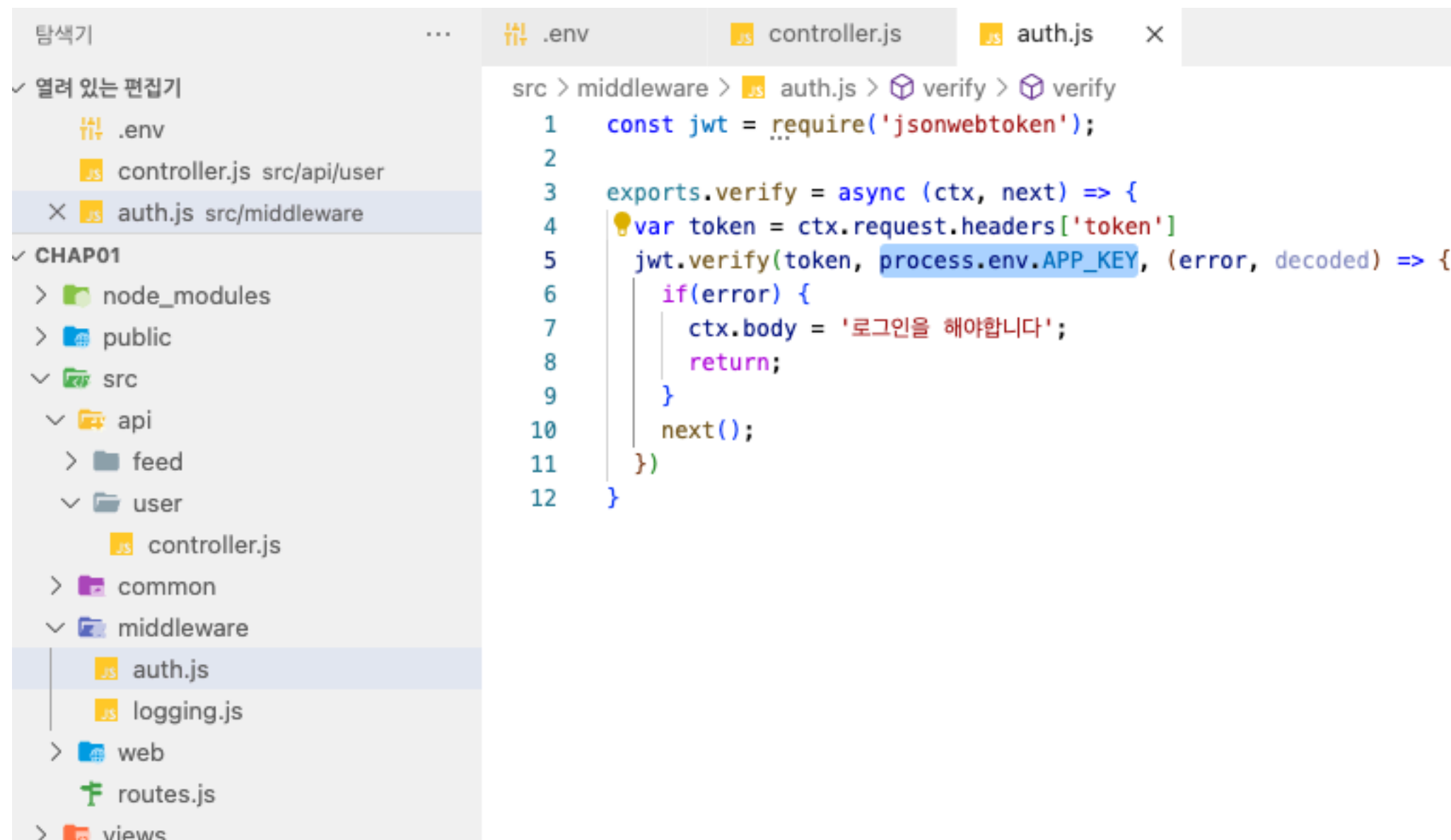
## 토큰 생성 함수



```
src > api > user > controller.js > generteToken > <function>
23  /**
24   * jwt 토큰 생성
25   * @param {object} payload 추가적으로 저장할 payload
26   * @returns {string} jwt 토큰string
27   */
28  let generteToken = (payload) => {
29    return new Promise((resolve, reject) => {
30      jwt.sign(payload, process.env.APP_KEY, (error, token) => {
31        if(error) { reject(error); }
32        resolve(token);
33      })
34    })
35  }
```

# 코드 내 참조 변경하기

## 토큰 검증 middleware



VS Code interface showing the file explorer on the left and the code editor on the right.

**File Explorer (Left):**

- 열려 있는 편집기
  - .env
  - controller.js src/api/user
  - auth.js src/middleware
- CHAP01
  - node\_modules
  - public
  - src
    - api
      - feed
      - user
        - controller.js
    - common
    - middleware
      - auth.js
      - logging.js
    - web
    - routes.js
    - views

**Code Editor (Right):**

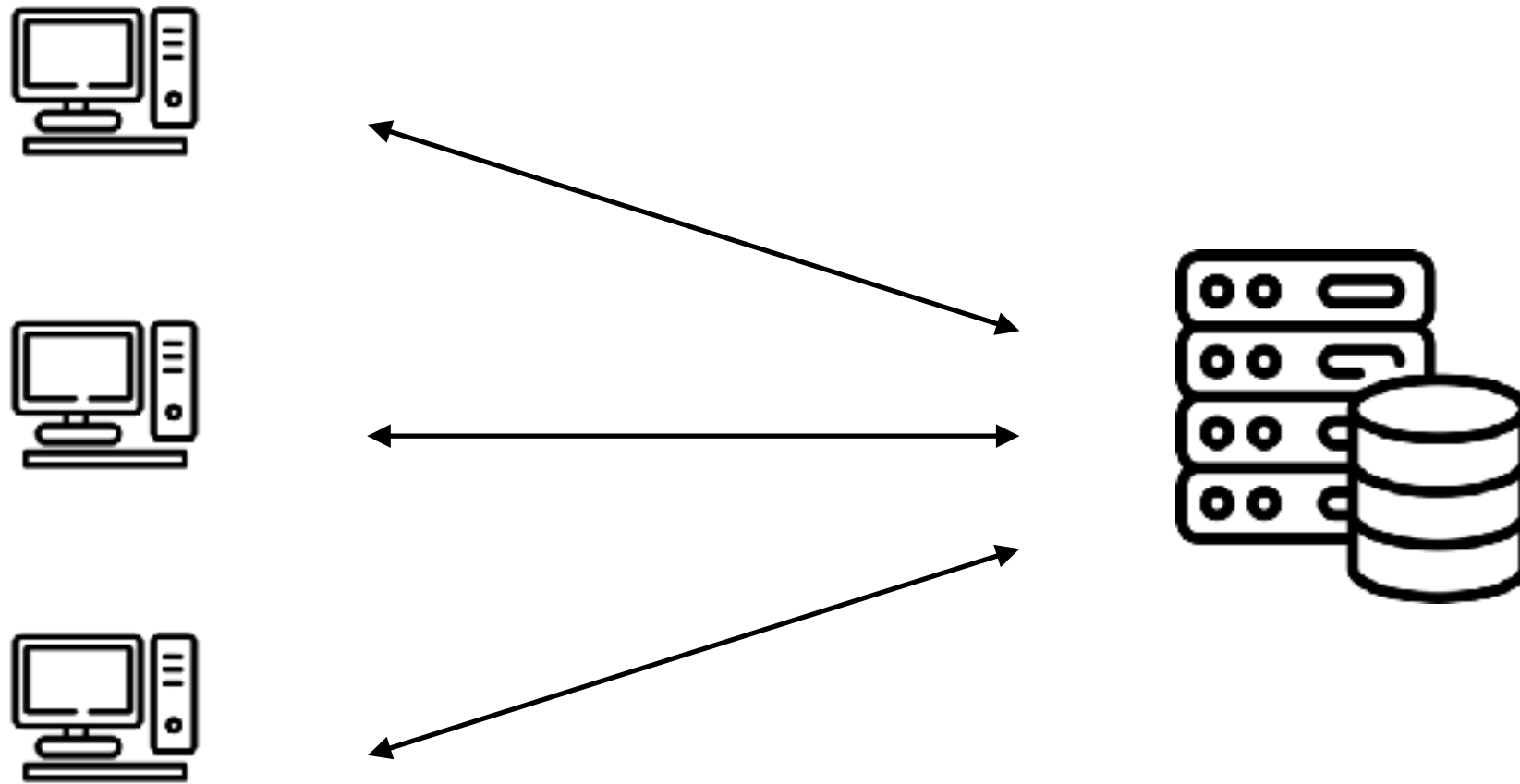
src > middleware > auth.js > verify > verify

```
1  const jwt = require('jsonwebtoken');
2
3  exports.verify = async (ctx, next) => {
4    var token = ctx.request.headers['token']
5    jwt.verify(token, process.env.APP_KEY, (error, decoded) => {
6      if(error) {
7        ctx.body = '로그인을 해야합니다';
8        return;
9      }
10     next();
11   })
12 }
```

**multipart**

# 서버에서의 파일 처리

## 클라이언트와 서버의 역할



# 파일 업로드 준비

## multer와 @koa/multer 설치

```
npm install multer @koa/multer
```

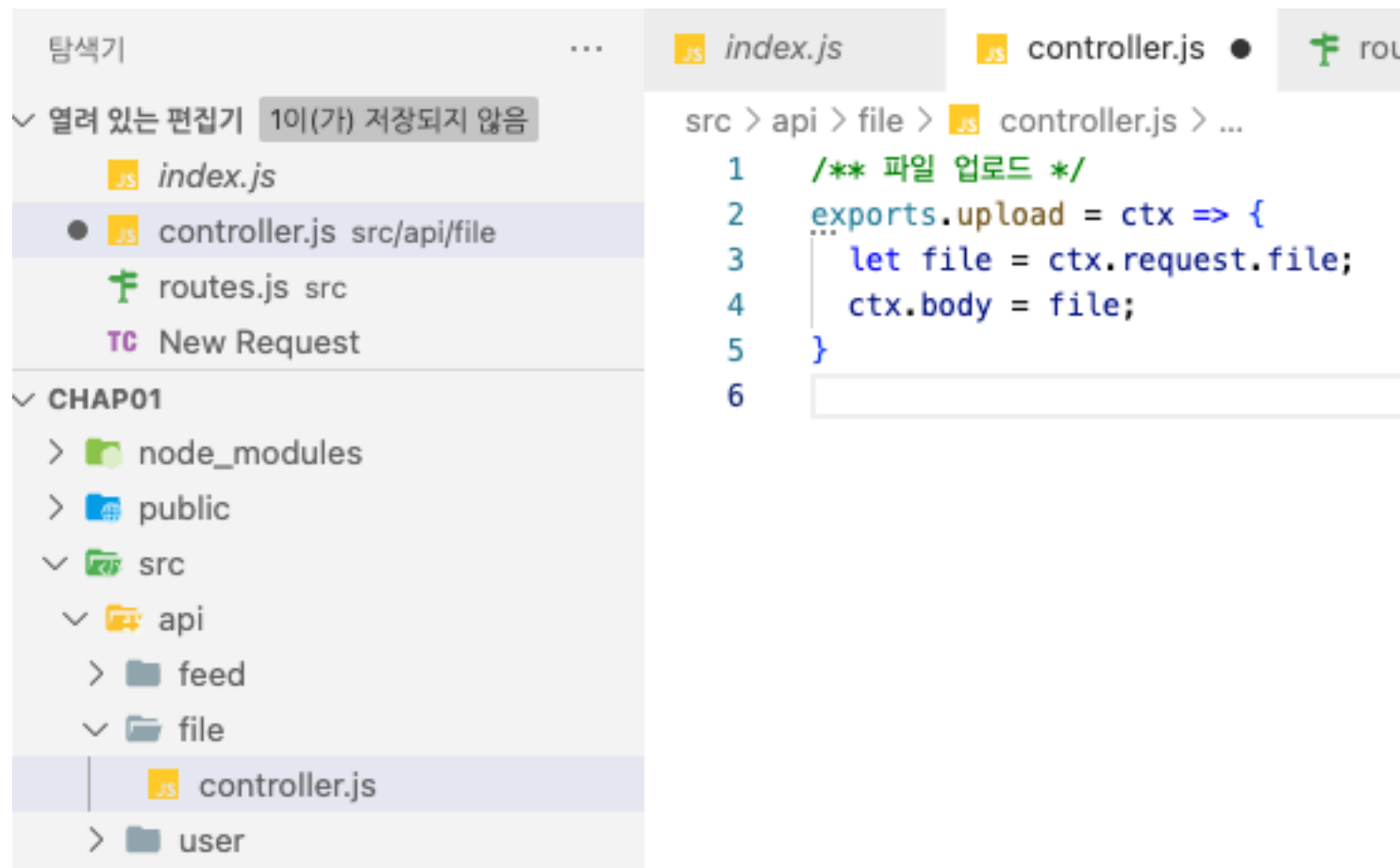
```
require('dotenv').config();
const Koa = require('koa');
const Router = require('@koa/router');
const bodyParser = require('koa-bodyparser');
const render = require('koa-ejs');
const path = require('path');
const app = new Koa();
const router = new Router();

// 서버 실행 포트
const port = process.env.PORT || 3000;

// 바디파서
app.use(bodyParser({formLimit: '5mb'}));
```

# 파일 업로드 함수 만들기

## file controller



# router 연결

## file controller 연결

The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a `src` directory containing `api`, `common`, `middleware`, `storage`, `uploads`, and `web` subdirectories. The `api` directory contains `feed`, `file`, and `user` subdirectories. The `file` directory contains `controller.js`. The `web` directory contains `controller.js`. The `routes.js` file is selected in the file explorer.

The code editor shows the `routes.js` file with the following code:

```
src > routes.js > ...
1  const Router = require('@koa/router');
2  const router = new Router();
3  const multer = require('@koa/multer');
4  const path = require('path');
5  const upload = multer({
6    dest: path.resolve(__dirname, '../', 'storage')
7  });
8
9  const { myLogging } = require('./middleware/logging');
10 const { verify } = require('./middleware/auth');
11
12 const webController = require('./web/controller');
13 const apiUserController = require('./api/user/controller');
14 const apiFeedController = require('./api/feed/controller');
15
16 router.use(myLogging);
17
18 router.post('/file/upload', upload.single('file'), require('./api/file/controller').upload);
19
20 router.get('/', webController.home);
21 router.get('/page/:page', webController.page);
22
23 router.post('/api/user/register', apiUserController.register);
24 router.post('/api/user/login', apiUserController.login);
25
26 router.use(verify);
27
```

# 파일 업로드 테스트

## api 테스트 도구로 테스트하기

index.jscontroller.jsroutes.jslocalhost:3000/file/uploa.. X

POSThttp://localhost:3000/file/uploadSend

QueryHeaders 2AuthBody 1TestsPre Run New

JsonXmlTextFormForm-encodeGraphqlBinary

Form Fields

☐ field namevalue

☒ Files

Files

☒ file

파일 선택

선.. IMG\_3662.jpeg

☐ field name

파일 선택

선.. Select file

Status: 200 OKSize: 289 BytesTime: 21 ms

ResponseHeaders 4CookiesResultsDocs{}≡

```
1  {
2    "fieldname": "file",
3    "originalname": "IMG_3662.jpeg",
4    "encoding": "7bit",
5    "mimetype": "image/jpeg",
6    "destination": "/Users/jinhyung/Desktop/chap01/storage",
7    "filename": "9583bcddb2d8f913da81c1bd87a7970",
8    "path": "/Users/jinhyung/Desktop/chap01/storage
           /9583bcddb2d8f913da81c1bd87a7970",
9    "size": 245945
10 }
```



**database**

# 데이터베이스

## mysql 설치

← → ↺ 🏠 dev.mysql.com/downloads/mysql/

### MySQL Community Downloads

← MySQL Community Server

**General Availability (GA) Releases** Archives ⓘ

#### MySQL Community Server 8.0.31

Select Operating System:  
Microsoft Windows ▼

Looking for previous GA versions?

**Recommended Download:**

##### MySQL Installer for Windows

All MySQL Products. For All Windows Platforms. In One Package.

Starting with MySQL 8.0 the MySQL Installer package replaced the standalone MSI packages.

Windows (x86, 32 & 64-bit), MySQL Installer MSI

[Go to Download Page >](#)

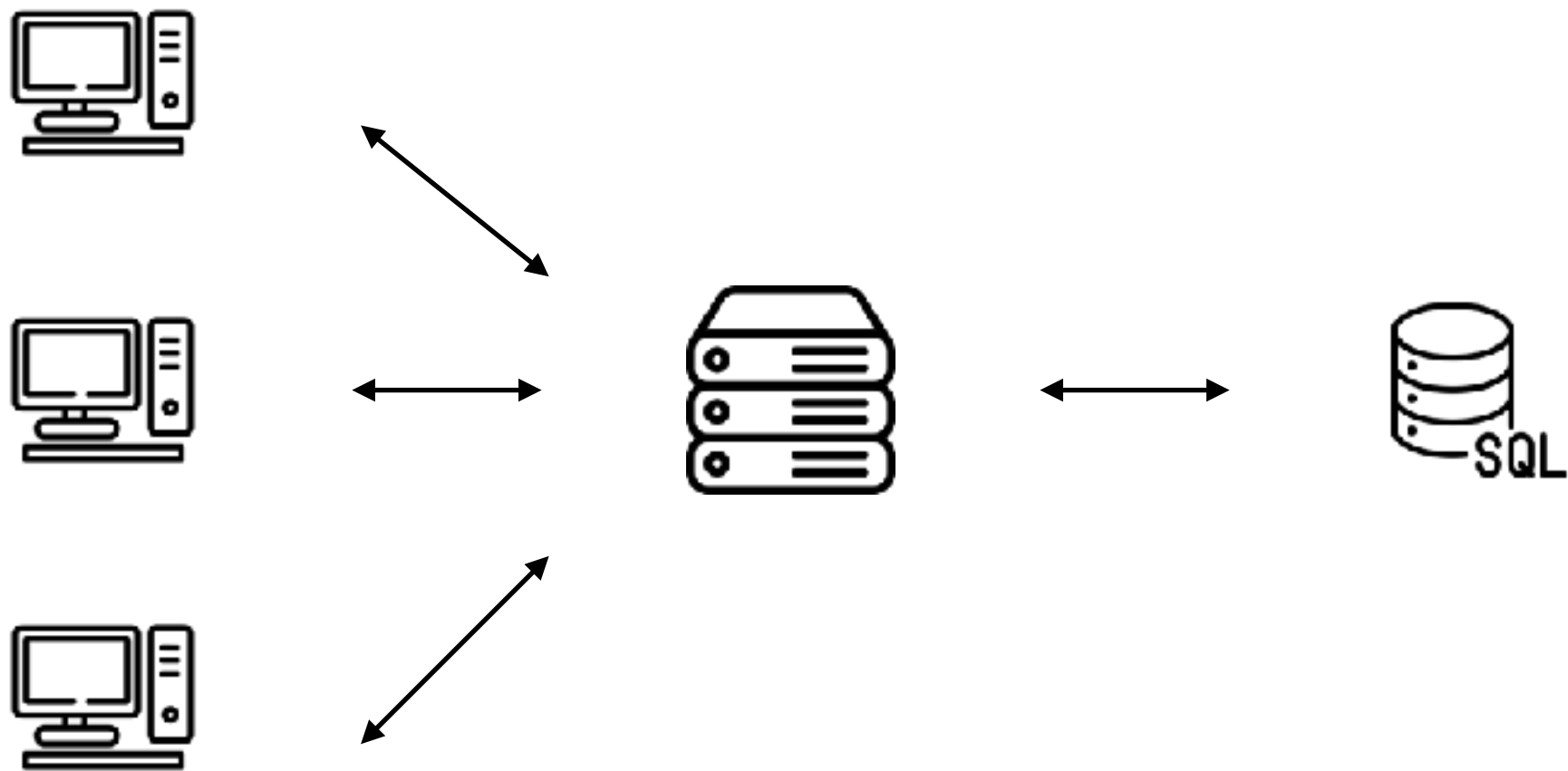
**Other Downloads:**

Windows (x86, 64-bit), ZIP Archive (mysql-8.0.31-win64.zip)	8.0.31	222.3M	<a href="#">Download</a>
MD5: e9135ee4988e41a932f89de2b8661d1ad   <a href="#">Signature</a>			
Windows (x86, 64-bit), ZIP Archive Debug Binaries & Test Suite (mysql-8.0.31-win64-debug-test.zip)	8.0.31	555.6M	<a href="#">Download</a>
MD5: 1a07b0a0309f0a0d01e0e25ebf011e0e07   <a href="#">Signature</a>			

⚠ We suggest that you use the [MD5 checksums](#) and [GnuPG signatures](#) to verify the integrity of the packages you download.

# 데이터베이스

## 데이터베이스 서버의 구조



# 데이터베이스

## mysql2 설치 및 기본실행

```
npm install mysql2
```

```
// get the client
const mysql = require('mysql2');

// create the connection to database
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
});

// simple query
connection.query(
  'SELECT * FROM `table` WHERE `name` = "Page" AND `age` > 45',
  function(err, results, fields) {
    console.log(results); // results contains rows returned by server
    console.log(fields); // fields contains extra meta data about results, if available
  }
);

// with placeholder
connection.query(
  'SELECT * FROM `table` WHERE `name` = ? AND `age` > ?',
  ['Page', 45],
  function(err, results) {
    console.log(results);
  }
);
```

# 데이터베이스

## 서버와 데이터베이스의 다양한 연결방법

### Installation

---

MySQL2 is free from native bindings and can be installed on Linux, Mac OS or Windows without any issues.

```
npm install --save mysql2
```

### First Query

---

Statement

### Using Prepared Statements

---

Prepared Statement

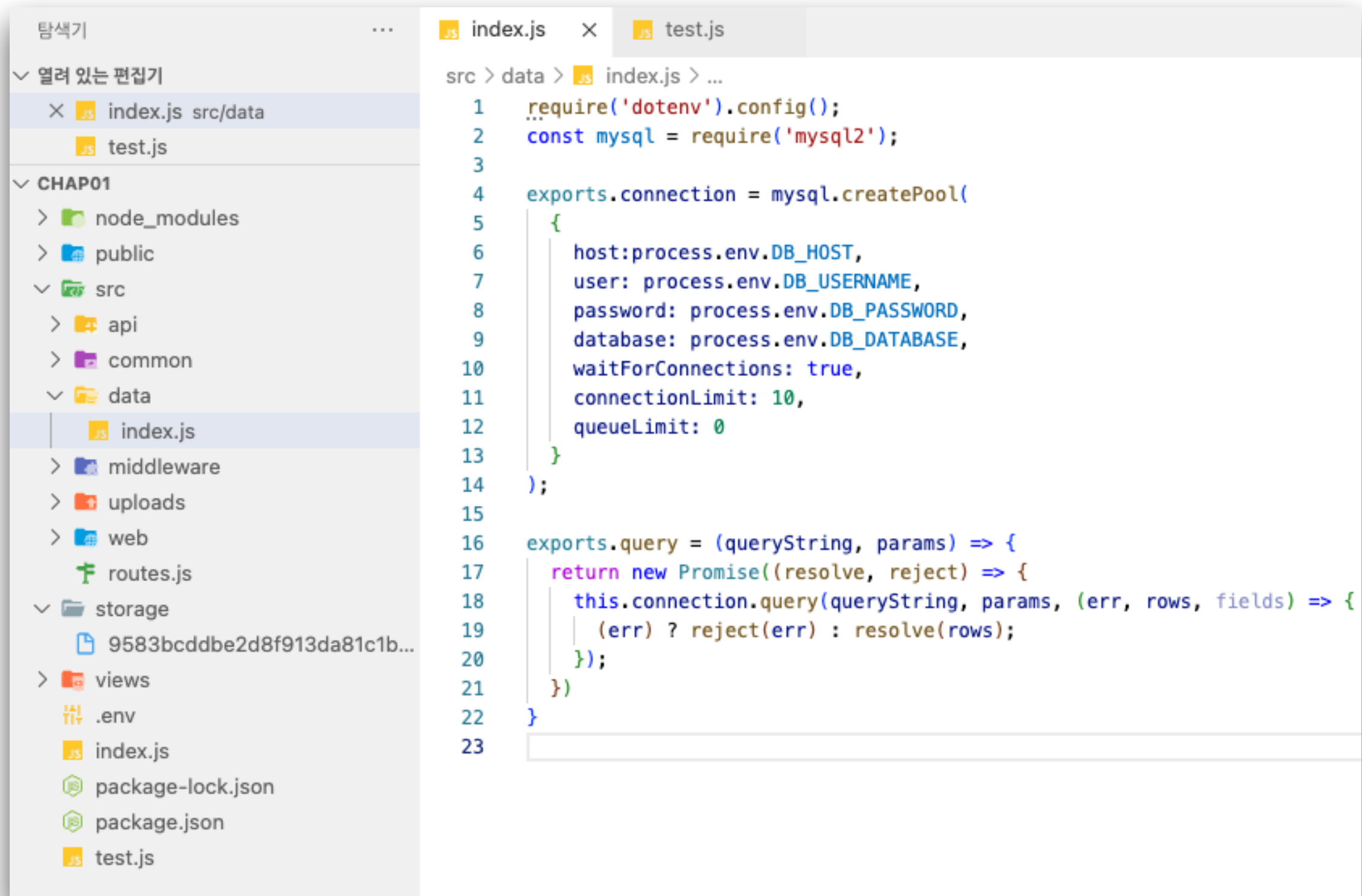
### Using connection pools

---

Connection Pool

# 데이터베이스

## 쉽게 사용할 수 있는 함수 생성



# 데이터베이스

## .env 에 서버정보 등록

