

# 프로젝트 기반 실무형 서비스

Chap04. koa로 서버 시작

김진형

# Framework

어떠한 목적을 달성하기 위해 복잡하게 얽혀있는 문제를 해결하기 위한 구조

# Library & Framework

자동차를 만든다고 가정한다면?



- 엔진의 종류
- 시트의 재질
- 내, 외장재 색상
- 기타 추가 옵션들



- 카 오디오
- 네비게이션
- 커스텀 핸들
- 전자식 거울

# Library & Framework

## framework 를 사용하는 이유

- 체계적인 코드관리로 유지보수가 용이하다
- 기본설계 및 기능 라이브러리를 제공하여 개발 생산성이 높다.
- 이미 알려진 보안등의 문제를 쉽게 해결할 수 있다.
- 누군가 만들어놓은 틀에맞게 만들어야하기에 학습이 필요하다
- 프레임워크에 종속적인 문제가 발생할 수 있다.

**koa web framework**

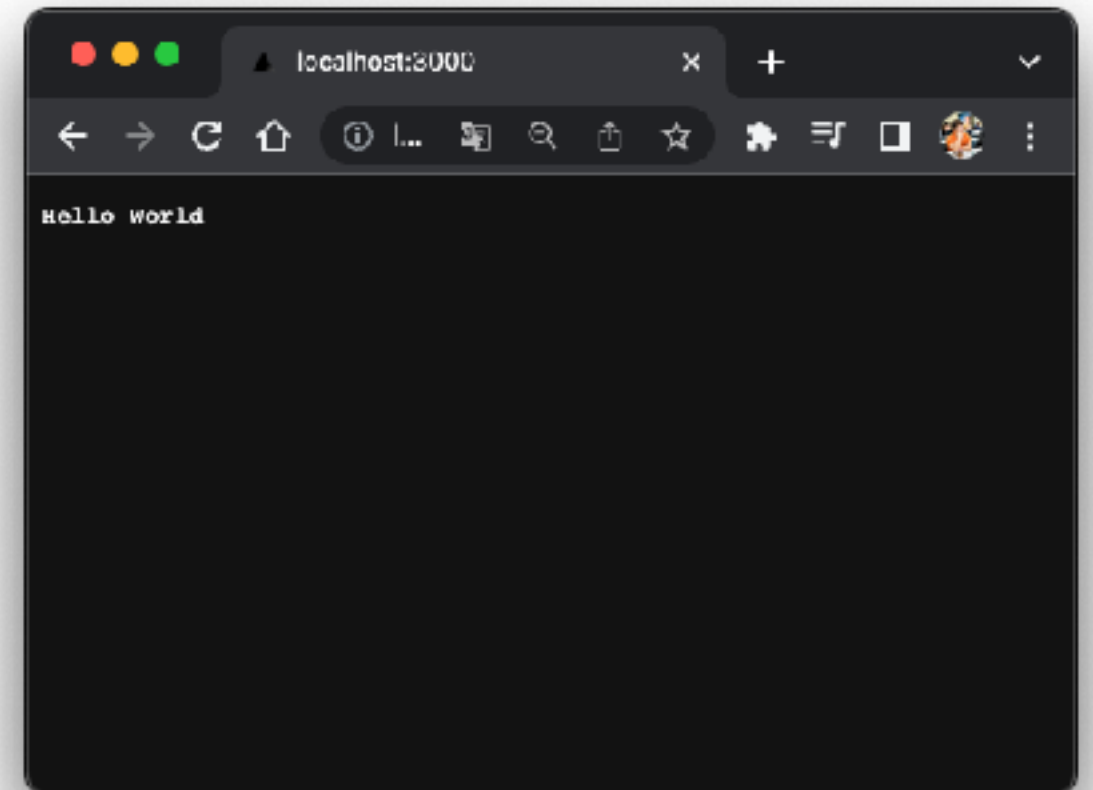
# koa 기본 설치 및 실행

npm install koa

```
const Koa = require('koa');
const app = new Koa();
const port = process.env.PORT || 3000;

app.use(ctx => {
  ctx.body = 'Hello World';
});

app.listen(port, () => {
  console.log(`웹서버 구동... ${port}`);
});
```



# koa-router 설치 및 실행

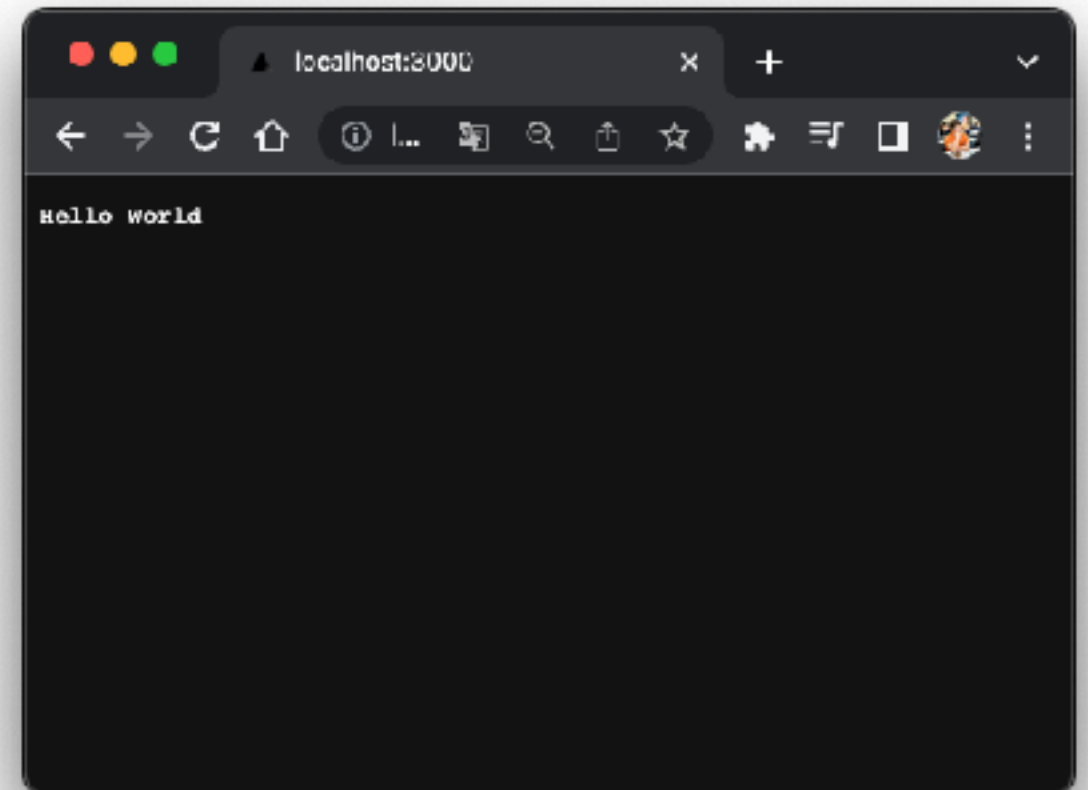
npm install @koa/router

```
const Koa = require('koa');
const Router = require('@koa/router');
const app = new Koa();
const router = new Router();
const port = process.env.PORT || 3000;

router.get('/', (ctx, next) => {
  ctx.body = 'Hello World';
});

app.use(router.routes());
app.use(router.allowedMethods());

app.listen(port, () => {
  console.log(`웹서버 구동... ${port}`);
});
```



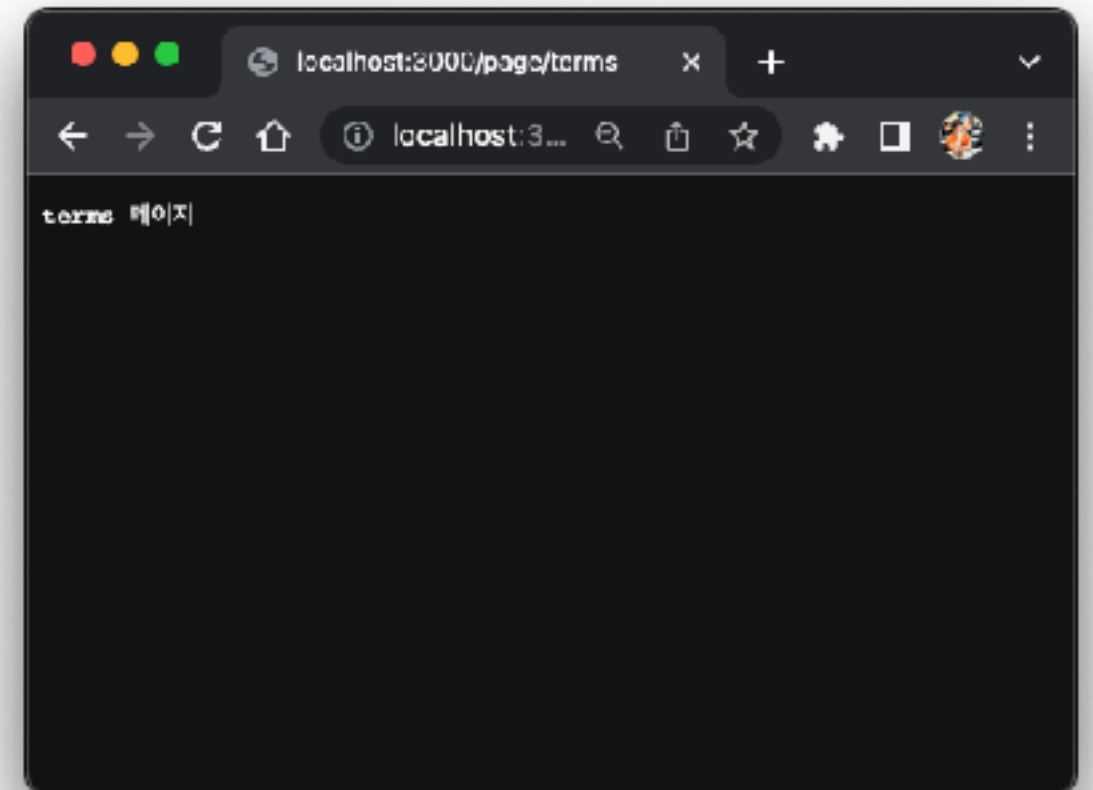
# 기타 router 추가

npm install @koa/router

```
const Koa = require('koa');
const Router = require('@koa/router');
const app = new Koa();
const router = new Router();
const port = process.env.PORT || 3000;

router.get('/', (ctx, next) => {
  ctx.body = 'Hello World';
});
router.get('/sitemap', (ctx, next) => {
  ctx.body = '사이트맵';
});
router.get('/page/:name', (ctx, next) => {
  let name = ctx.params.name;
  ctx.body = `${name} 페이지`;
});
app.use(router.routes());
app.use(router.allowedMethods());

app.listen(port, () => {
  console.log(`웹서버 구동... ${port}`);
});
```





**package.json 활용**

# package.json 의 scripts 영역

npm run start 또는 npm start

```
{
  "name": "chap01",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@koa/router": "^12.0.0",
    "koa": "^2.14.1",
    "koa-bodyparser": "^4.3.0",
    "moment": "^2.29.4"
  }
}
```

# 서버 코드 자동 재시작

매번 코드가 변경될때마다 켜다 켜는 과정을 개선

```
npm install --save-dev nodemon
```

```
{
  "name": "chap01",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon --watch src/ index.js --inspect=0.0.0.0"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@koa/router": "^12.0.0",
    "koa": "^2.14.1",
    "koa-bodyparser": "^4.3.0",
    "moment": "^2.29.4"
  },
  "devDependencies": {
    "nodemon": "^2.0.20"
  }
}
```

**module 분리**

# routes파일로 router 모듈 분리

src/routes.js

```
const Router = require('@koa/router');
const router = new Router();

router.get('/', (ctx, next) => {
  ctx.body = 'Hello World';
});
router.get('/sitemap', (ctx, next) => {
  ctx.body = '사이트맵';
});
router.get('/page/:name', (ctx, next) => {
  let name = ctx.params.name;
  ctx.body = `${name} 페이지`;
});

module.exports = router;
```

✓ CHAP01

> node\_modules

✓ src

routes.js

index.js

package-lock.json

package.json

# 분리된 router 모듈 사용

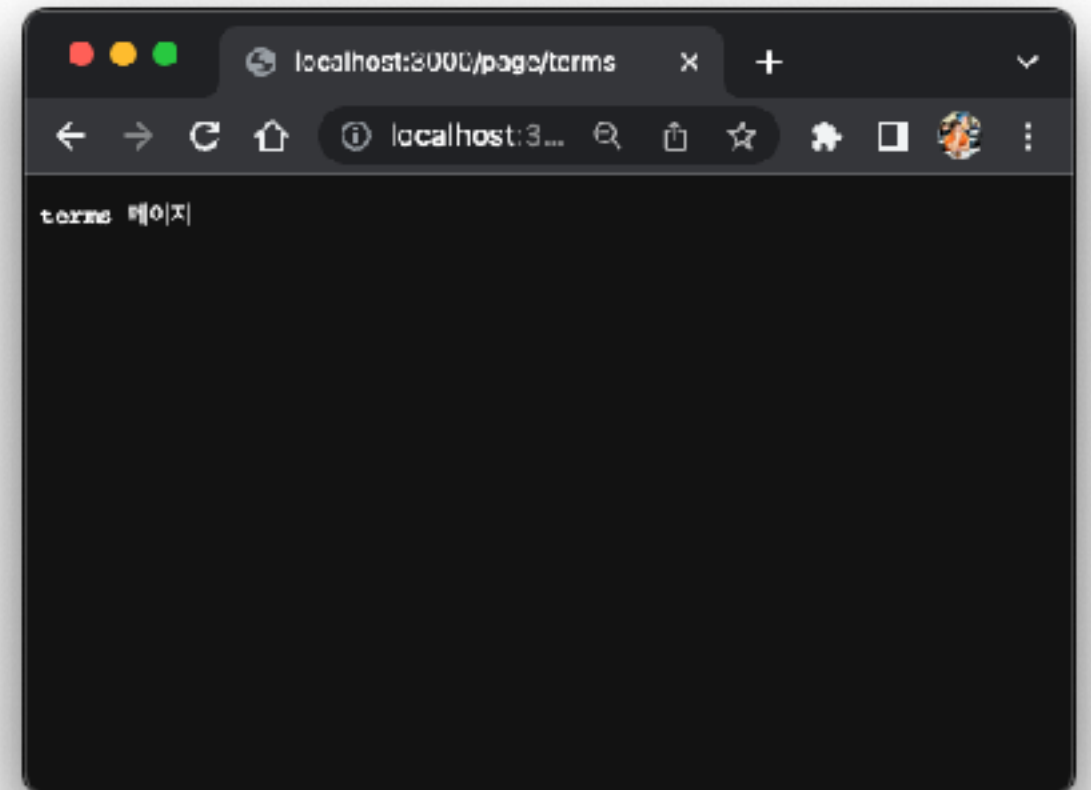
index.js

```
const Koa = require('koa');
const Router = require('@koa/router');
const app = new Koa();
const router = new Router();

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

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

// 서버 실행
app.listen(port, () => {
  console.log(`웹서버 구동... ${port}`);
});
```



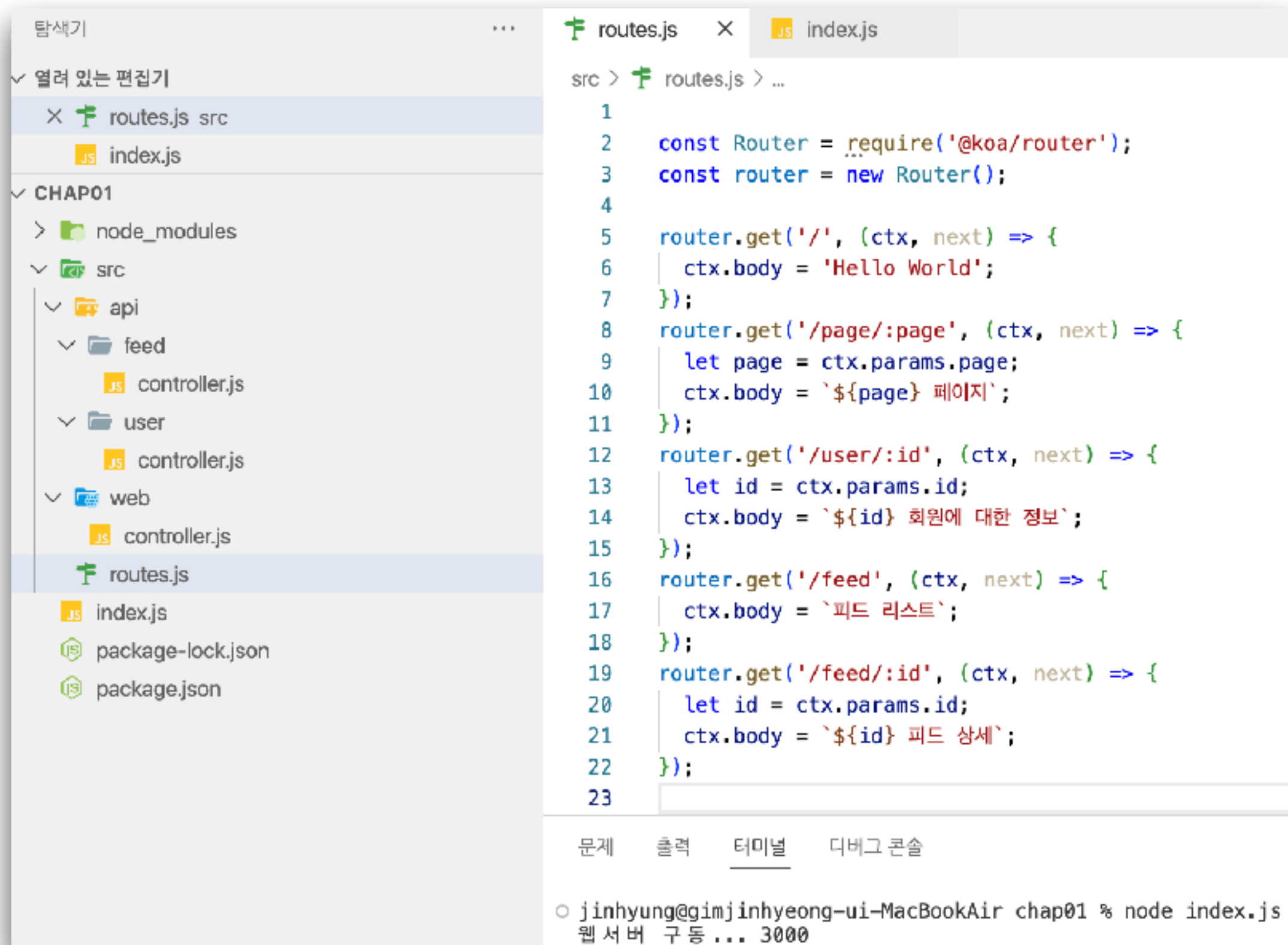
# 페이지에 대한 계획

어떤 기능들을 어떤 url에 배치할까

GET	웹페이지	http://localhost:3000/	홈페이지 (간단한 소개 페이지)
GET		<u>http://localhost:3000/page/terms</u>	이용약관 페이지
GET		<u>http://localhost:3000/page/policy</u>	개인정보 처리방침 페이지
GET	회원	http://localhost:3000/user/:id	해당하는 id의 회원정보 보기
GET	피드	<u>http://localhost:3000/feed</u>	피드 페이지 (피드 리스트들 조회)
POST		<u>http://localhost:3000/feed</u>	피드 작성
GET		<u>http://localhost:3000/feed/:id</u>	해당 피드의 상세정보 보기
PUT		<u>http://localhost:3000/feed/:id</u>	해당 피드의 글 수정하기
DELETE		<u>http://localhost:3000/feed/:id</u>	해당 피드 삭제하기

# domain “영역” “집합”

## domain 기반으로 모듈 분리



The image shows a code editor interface with a file explorer on the left and a code editor on the right.

**File Explorer (Left):**

- 탐색기 (Search)
- 열려 있는 편집기 (Open Editors)
  - routes.js src
  - index.js
- CHAP01
  - node\_modules
  - src
    - api
      - feed
        - controller.js
      - user
        - controller.js
      - web
        - controller.js
      - routes.js (selected)
    - index.js
    - package-lock.json
    - package.json

**Code Editor (Right):**

routes.js

```
src > routes.js > ...
1
2 const Router = require('@koa/router');
3 const router = new Router();
4
5 router.get('/', (ctx, next) => {
6   ctx.body = 'Hello World';
7 });
8 router.get('/page/:page', (ctx, next) => {
9   let page = ctx.params.page;
10  ctx.body = `${page} 페이지`;
11 });
12 router.get('/user/:id', (ctx, next) => {
13   let id = ctx.params.id;
14   ctx.body = `${id} 회원에 대한 정보`;
15 });
16 router.get('/feed', (ctx, next) => {
17   ctx.body = `피드 리스트`;
18 });
19 router.get('/feed/:id', (ctx, next) => {
20   let id = ctx.params.id;
21   ctx.body = `${id} 피드 상세`;
22 });
23
```

문제 출력 터미널 디버그 콘솔

○ jinhyung@gimjinhyeong-ui-MacBookAir chap01 % node index.js  
웹 서버 구동 ... 3000



# domain 기반 분리

## 웹페이지 영역 분리

routes.js

src > routes.js > ...

```
1
2  const Router = require('@koa/router');
3  const router = new Router();
4
5  const webController = require('./web/controller');
6
7  router.get('/', webController.home);
8  router.get('/page/:page', webController.page);
9
10 router.get('/user/:id', (ctx, next) => {
11   let id = ctx.params.id;
12   ctx.body = `${id} 회원에 대한 정보`;
13 });
14 router.get('/feed', (ctx, next) => {
15   ctx.body = `피드 리스트`;
16 });
17 router.post('/feed', (ctx, next) => {
18   ctx.body = `피드 작성 완료`;
19 });
20 router.get('/feed/:id', (ctx, next) => {
21   let id = ctx.params.id;
22   ctx.body = `${id} 피드 상세`;
23 });
```

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 = (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.body = content;
18 }
19
```

# domain 기반 분리

## 사용자 영역 분리

```
routes.js
src > routes.js > ...
1
2 const Router = require('@koa/router');
3 const router = new Router();
4
5 const webController = require('./web/controller');
6 const apiUserController = require('./api/user/controller');
7
8 router.get('/', webController.home);
9 router.get('/page/:page', webController.page);
10
11 router.get('/user/:id', apiUserController.info);
12
13 router.get('/feed', (ctx, next) => {
14   ctx.body = `피드 리스트`;
15 });
16 router.post('/feed', (ctx, next) => {
17   ctx.body = `피드 작성 완료`;
18 });
19 router.get('/feed/:id', (ctx, next) => {
20   let id = ctx.params.id;
21   ctx.body = `${id} 피드 상세`;
22 });
23
```

```
controller.js .../web
src > api > user > controller.js > info > info
1 /** 해당 id의 회원정보들 */
2 exports.info = (ctx, next) => {
3   let id = ctx.params.id;
4   ctx.body = `${id} 회원에 대한 정보`;
5 }
```

# domain 기반 분리

## 피드 영역 분리

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

```
src > api > feed > controller.js > delete > de
1  /** 전체 피드보기 */
2  exports.index = (ctx, next) => {
3    ctx.body = `피드 리스트`;
4  }
5  /** 새 피드 작성 처리 */
6  exports.store = (ctx, next) => {
7    ctx.body = `피드 작성 완료`;
8  }
9  /** 피드 상세보기 */
10 exports.show = (ctx, next) => {
11   let id = ctx.params.id;
12   ctx.body = `${id} 피드 상세`;
13 }
14 /** 피드 수정 */
15 exports.update = (ctx, next) => {
16   let id = ctx.params.id;
17   ctx.body = `${id} 피드 수정`;
18 }
19 /** 피드 삭제 */
20 exports.delete = (ctx, next) => {
21   let id = ctx.params.id;
22   ctx.body = `${id} 피드 수정`;
23 }
```

# 데이터 처리

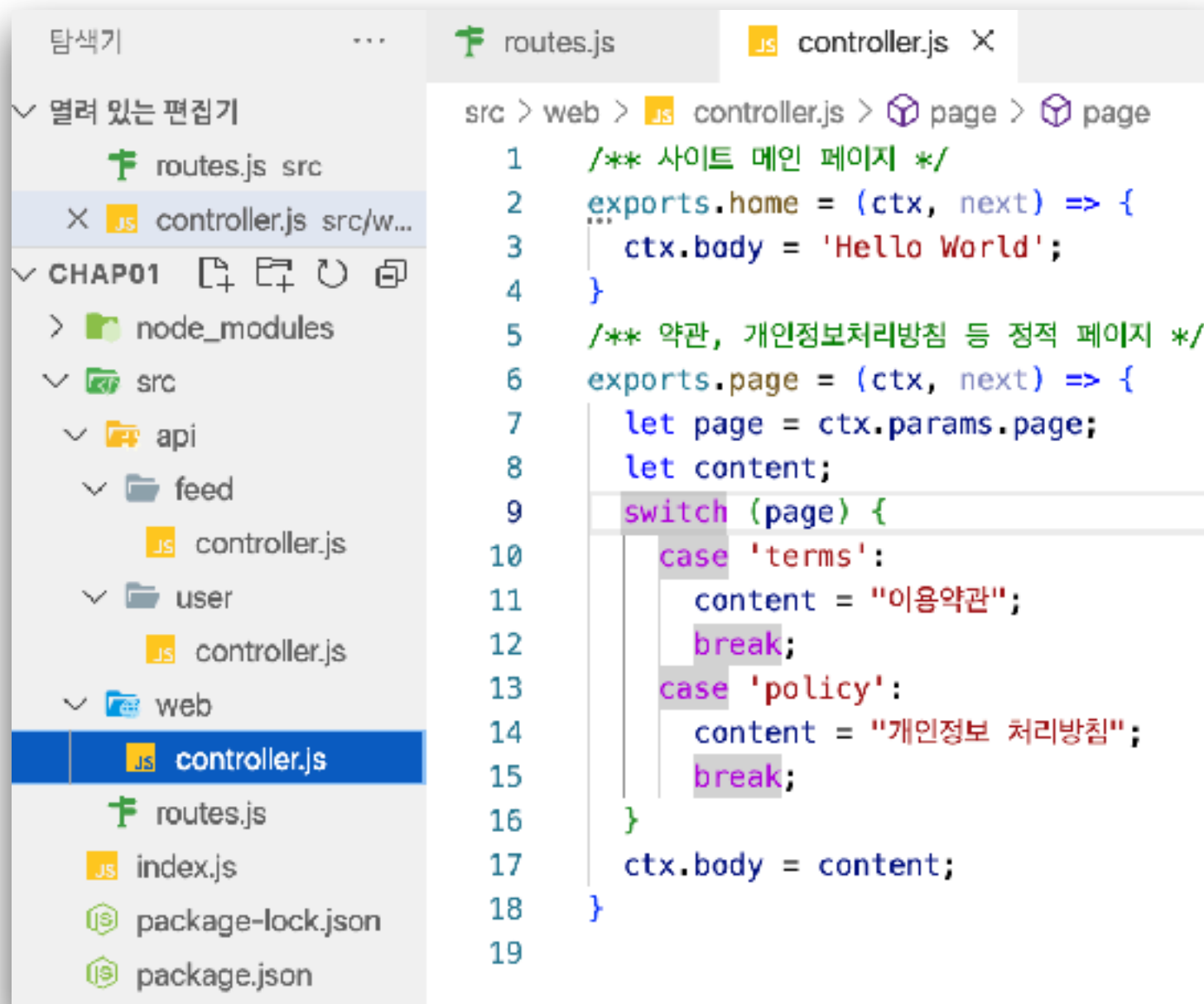
# request 데이터 타입

## query, param, body

- param
  - 주소에 포함된 변수
  - http://localhost:3000/page/:page 라우트에 http://localhost:3000/page/terms 호출
  - page 변수에는 terms가 저장
- query
  - url에 같이 넘어오는 데이터
  - key=value 의 쌍으로 구성되어있으며 &으로 구분하여 여러개 전송
  - url과 구분하기위하여 ? 문자를 사용
- body
  - POST또는 PUT 전송에서 request body에 담긴 데이터
  - 큰 용량을 처리하기에 용이

# param 예제

[GET] /page/:name

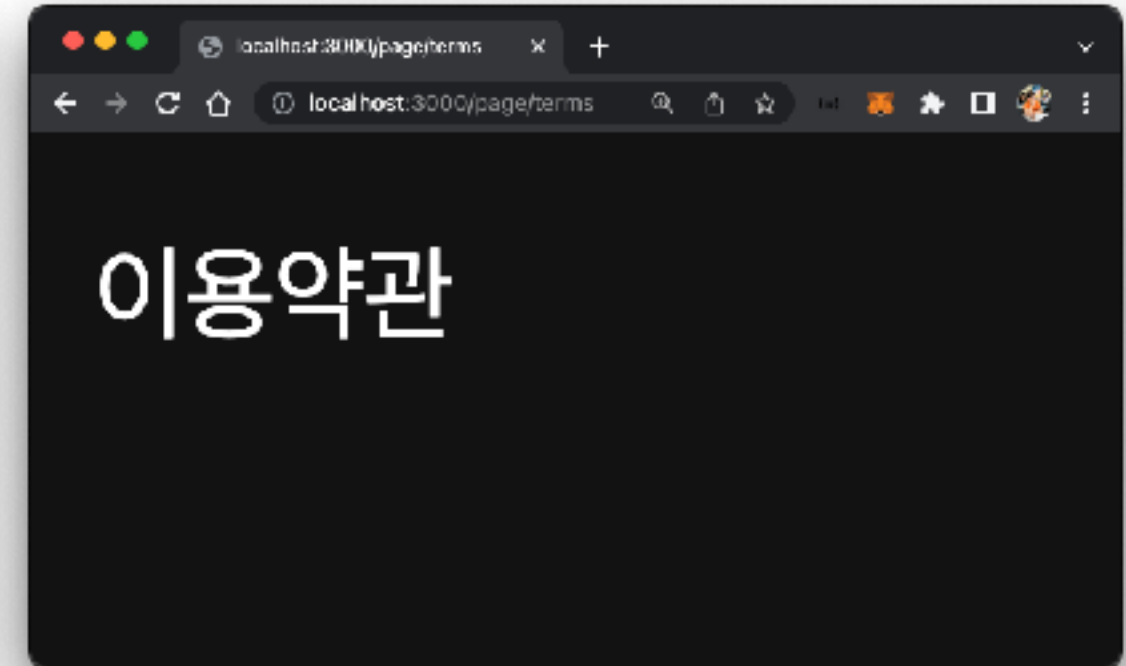


The screenshot shows a code editor with two tabs: `routes.js` and `controller.js`. The `controller.js` tab is active, showing the following code:

```
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 = (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.body = content;
18 }
19
```

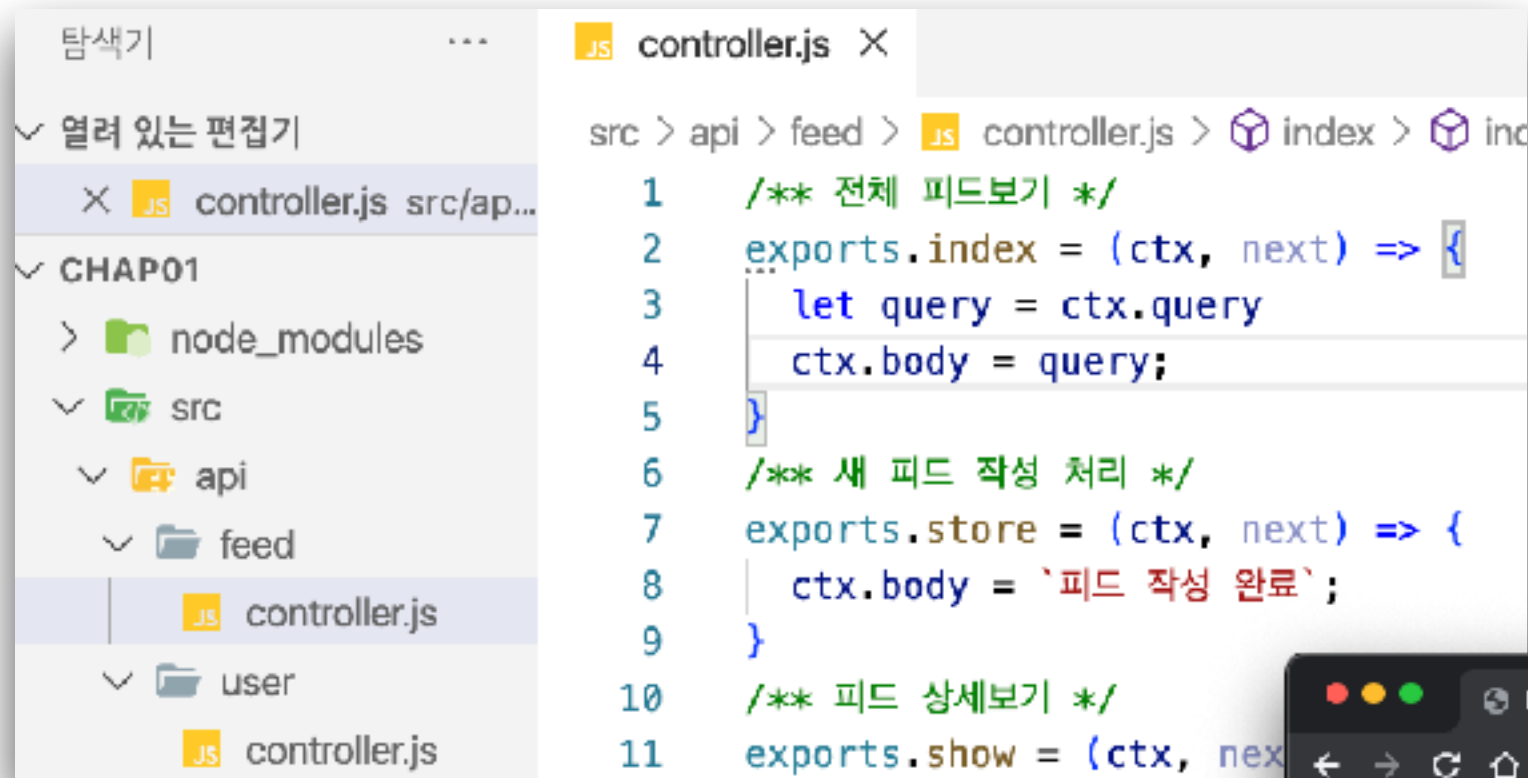
The left sidebar shows the project structure:

- src
  - api
    - feed
      - controller.js
    - user
      - controller.js
    - web
      - controller.js (selected)
      - routes.js
      - index.js
      - package-lock.json
      - package.json



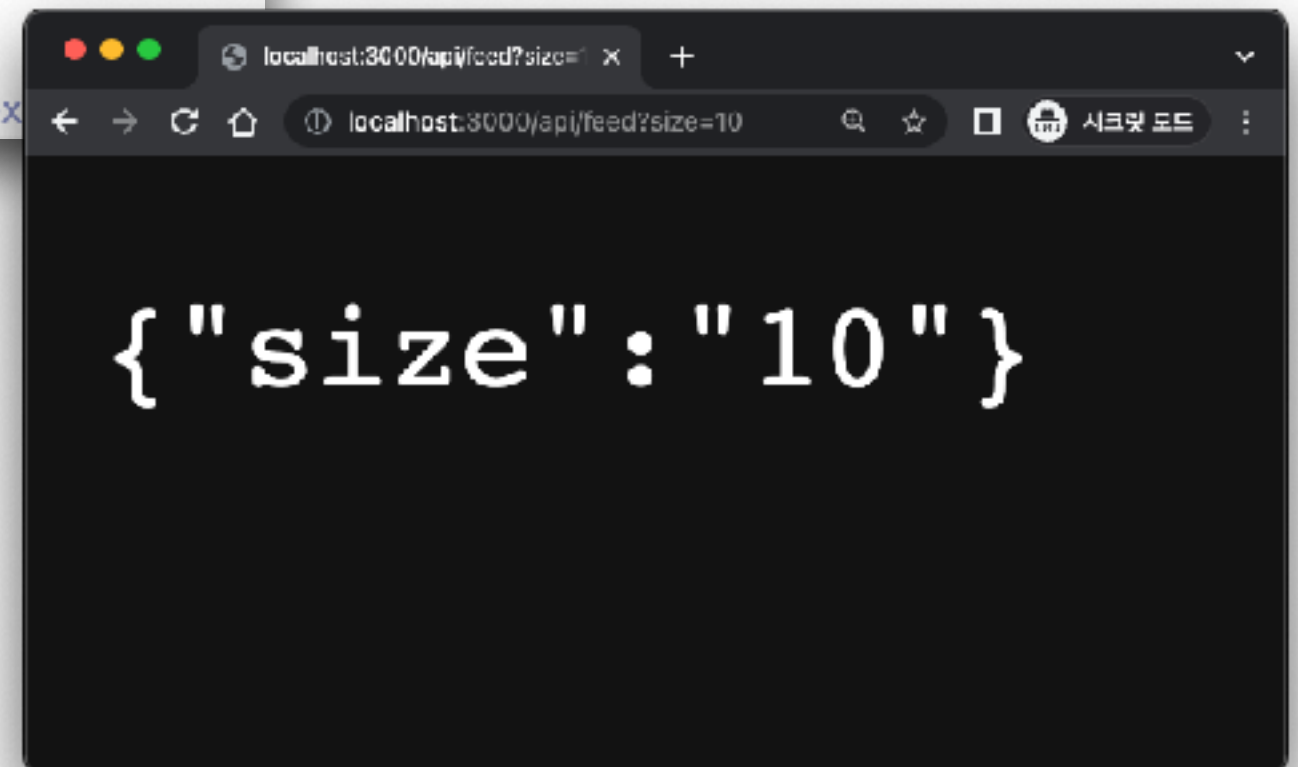
# query 예제

## [GET] /feed



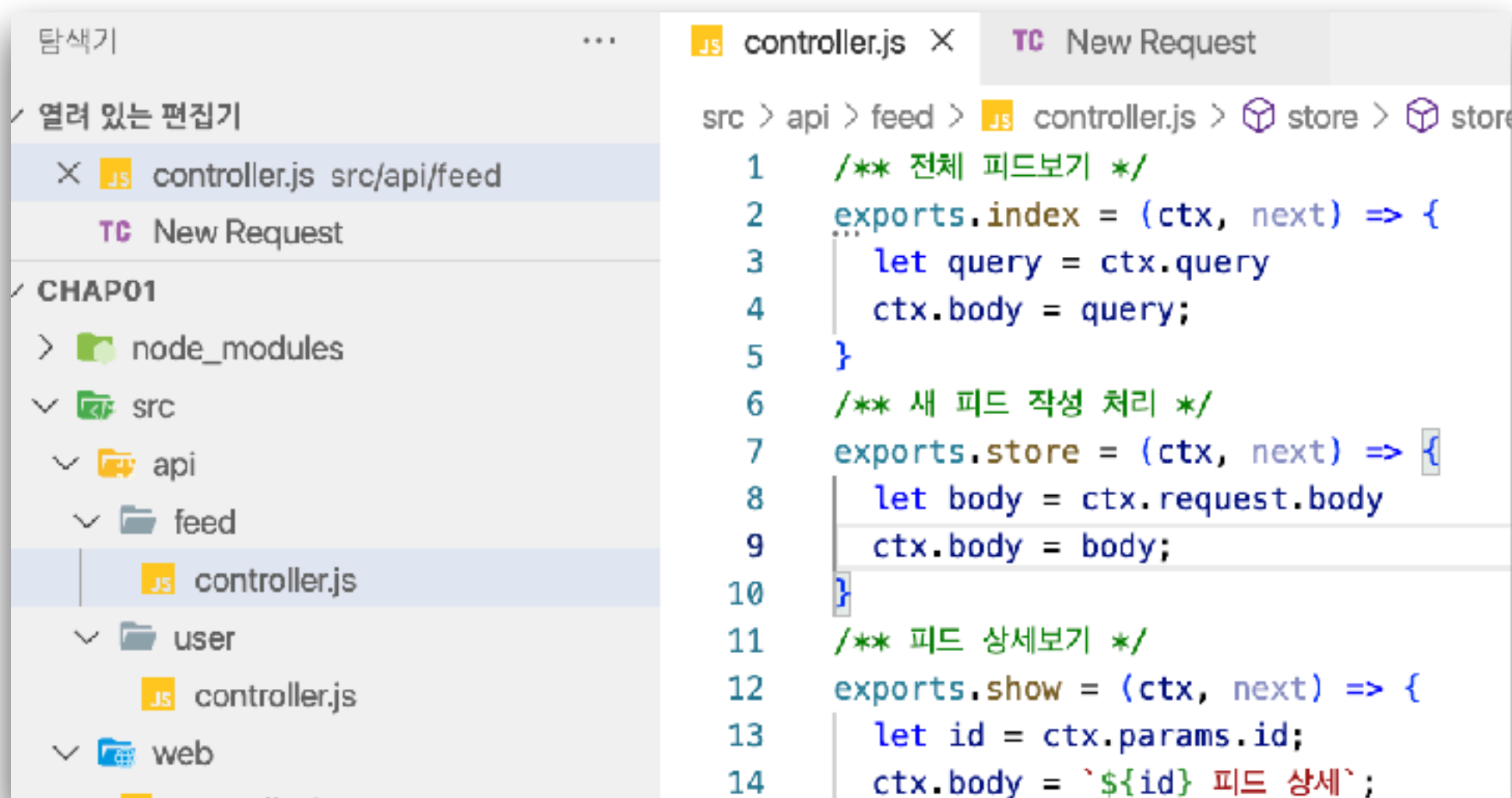
The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the project structure: src > api > feed > controller.js. The code editor shows the content of controller.js:

```
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    ctx.body = `피드 작성 완료`;
9  }
10 /** 피드 상세보기 */
11 exports.show = (ctx, next) => {
```



# body 예제

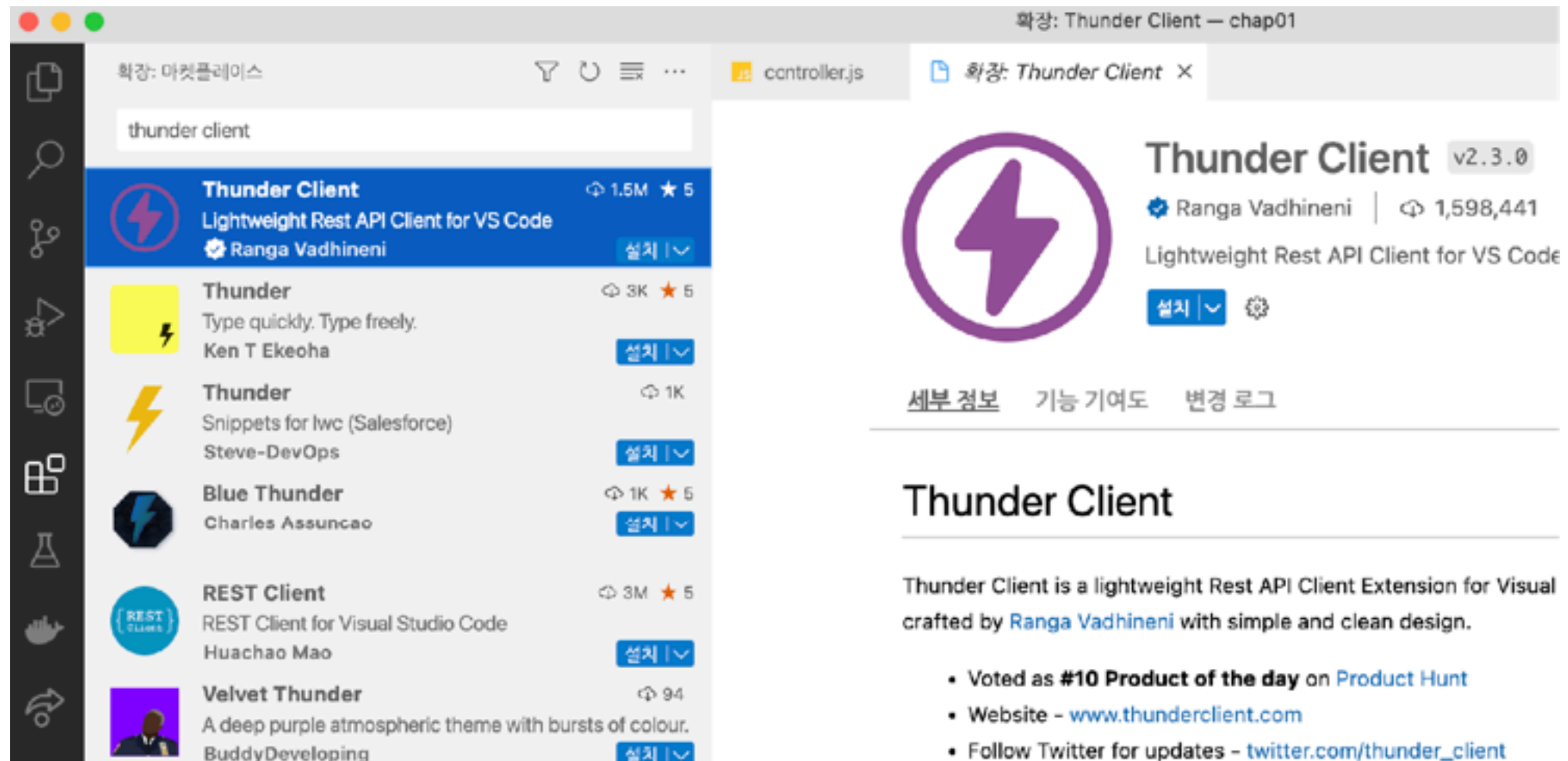
## [POST] /feed





# post 요청을 위한 api 테스트 확장 설치

## Thunder Client



The screenshot shows the Visual Studio Code interface with the 'Thunder Client' extension installed. The left sidebar displays the '확장: 마켓플레이스' (Extensions: Marketplace) view, showing a search for 'thunder client'. The right sidebar shows the details for the 'Thunder Client' extension by Ranga Vadhineni, version 2.3.0. The extension is described as a 'Lightweight Rest API Client for VS Code' and has 1,598,441 downloads. The details page includes a description, a list of features, and links to the website and Twitter.

확장: Thunder Client — chap01

확장: 마켓플레이스

thunder client

**Thunder Client** v2.3.0  
Lightweight Rest API Client for VS Code  
Ranga Vadhineni | 1,598,441  
설치

**Thunder**  
Type quickly. Type freely.  
Ken T Ekeoha  
설치

**Thunder**  
Snippets for lwc (Salesforce)  
Steve-DevOps  
설치

**Blue Thunder**  
Charles Assuncao  
설치

**REST Client**  
REST Client for Visual Studio Code  
Huachao Mao  
설치

**Velvet Thunder**  
A deep purple atmospheric theme with bursts of colour.  
BuddyDeveloping  
설치

세부 정보 기능 기여도 변경 로그

### Thunder Client

Thunder Client is a lightweight Rest API Client Extension for Visual crafted by [Ranga Vadhineni](#) with simple and clean design.

- Voted as **#10 Product of the day** on [Product Hunt](#)
- Website - [www.thunderclient.com](http://www.thunderclient.com)
- Follow Twitter for updates - [twitter.com/thunder\\_client](https://twitter.com/thunder_client)

# POST 요청 테스트

예상하지 못한 결과...

The screenshot displays the Thunder Client interface. The top bar shows the application name 'THUNDER CLIENT' and several tabs: 'controller.js', 'New Request', and 'index.js'. The 'New Request' tab is active, showing a POST request to 'http://localhost:3000/api/feed'. The request body is configured as 'Form-encode' with two fields: 'message' (checked) and 'name' (unchecked). The 'message' field contains the text '첫번째 피드첫번째 피드'. The 'name' field contains the text 'value'. The response status is '204 No Content'. The left sidebar shows the 'Activity' tab with a filter activity input and a list of requests, including the current POST request to 'localhost:3000/api/feed'.

THUNDER CLIENT

New Request

Activity Collections Env

filter activity

POST localhost:3000/api/feed just now

POST http://localhost:3000/api/feed Send

Query Headers 2 Auth Body 1 Tests Pre Run New

Json Xml Text Form Form-encode Graphql Binary

Form Encoded

☒ message 첫번째 피드첫번째 피드

☐ name value

Status: 204 No Content

Response Headers 2

1

# request body를 사용하기 위해 body-parser 추가

```
npm install koa-bodyparser
```

```
const Koa = require('koa');
const Router = require('@koa/router');
const bodyParser = require('koa-bodyparser');
const app = new Koa();
const router = new Router();

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

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

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

// 서버 실행
app.listen(port, () => {
  console.log(`웹서버 구동... ${port}`);
});
```

# POST 요청 테스트

성공

The screenshot displays the Thunder Client interface with a new request configured and executed. The request is a POST to `http://localhost:3000/api/feed` using form encoding. The response is a 200 OK status with a JSON body containing a message.

**Request Configuration:**

- Method: POST
- URL: `http://localhost:3000/api/feed`
- Body Type: Form-encode
- Form Fields:
  - ☒ message: 첫번째 피드첫번째 피드
  - ☐ name: value

**Response Details:**

- Status: 200 OK
- Size: 46 Bytes
- Time: 11 ms
- Response Body:

```
1 {
2   "message": "첫 번째 피드첫 번째 피드"
3 }
```

# JSON

## JavaScript Object Notation

- key, value 쌍의 DATA 교환방식
- key, value의 집합인 object와 array 타입 2가지
- 자바스크립트 객체 표기법과 매우 비슷하다
- 대부분의 서버와 클라이언트의 교류에서 사용한다.

```
{  
  "이름" : "홍길동",  
  "나이" : 55,  
  "성별" : "남",  
  "주소" : "서울특별시 양천구 목동",  
  "특기" : ["검술", "코딩"],  
  "가족관계" : {"#": 2, "아버지": "홍판서", "어머니": "춘섬"},  
  "회사" : "경기 수원시 팔달구 우만동"  
}
```

정적 파일

# 정적 파일들 접근

## node 서버를 이용하지 않아도 되는 파일들

npm install koa-static

열려 있는 편집기

controller.js src/api/feed

index.js

CHAP01

node\_modules

public

src

api

feed

controller.js

user

controller.js

web

controller.js

routes.js

index.js

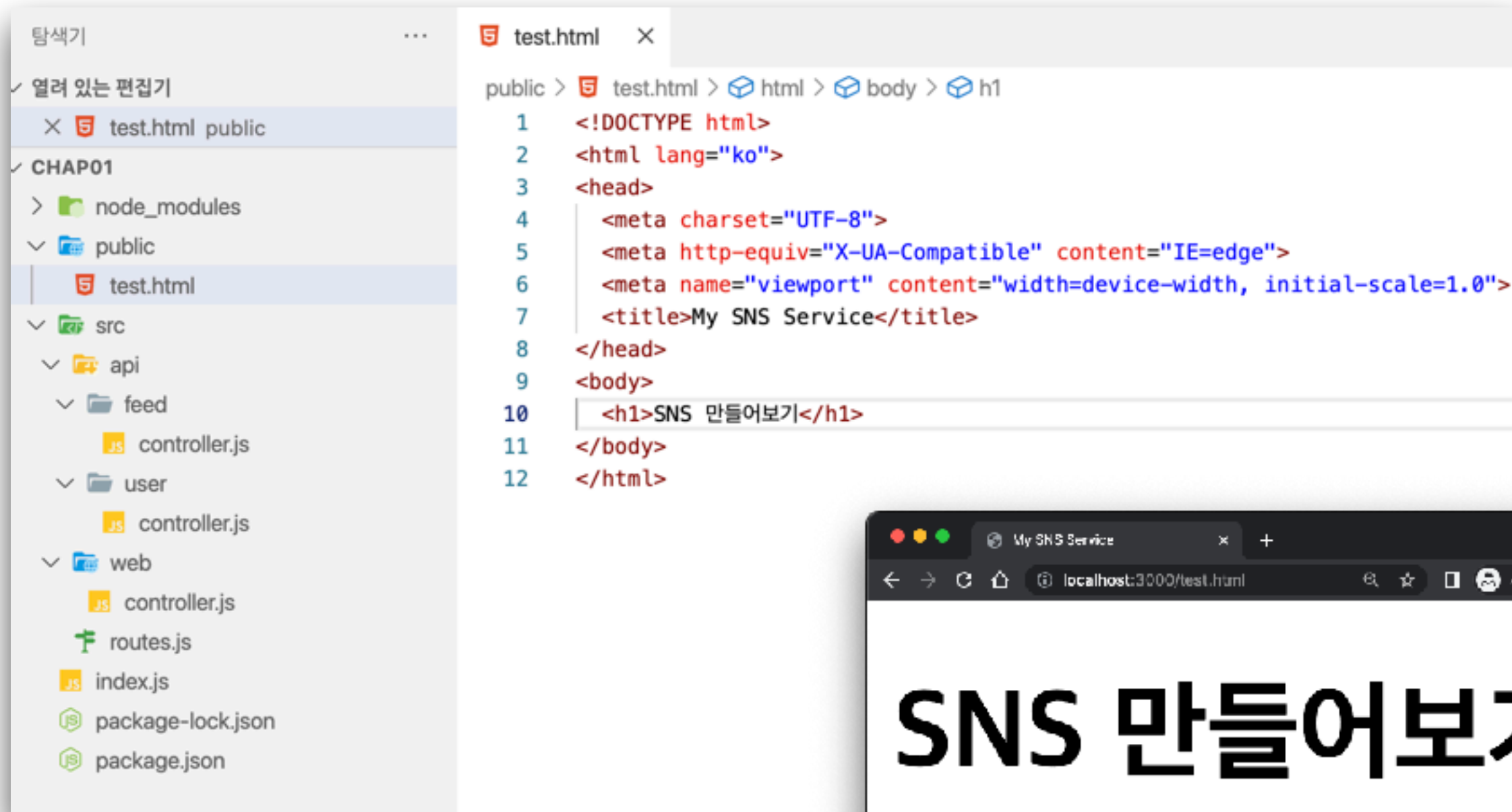
package-lock.json

index.js > ...

```
1  const Koa = require('koa');
2  const Router = require('@koa/router');
3  const bodyParser = require('koa-bodyparser');
4  const app = new Koa();
5  const router = new Router();
6
7  //서버 실행 포트
8  const port = process.env.PORT || 3000;
9
10 // 바디파서
11 app.use(bodyParser({formLimit: '5mb'}));
12
13 // 정적 파일
14 app.use(require('koa-static')(`${__dirname}/public`));
15
16 // 라우터 설정
17 router.use(require('./src/routes').routes());
18 app.use(router.routes());
19 app.use(router.allowedMethods());
20
```

# 정적 파일들 접근

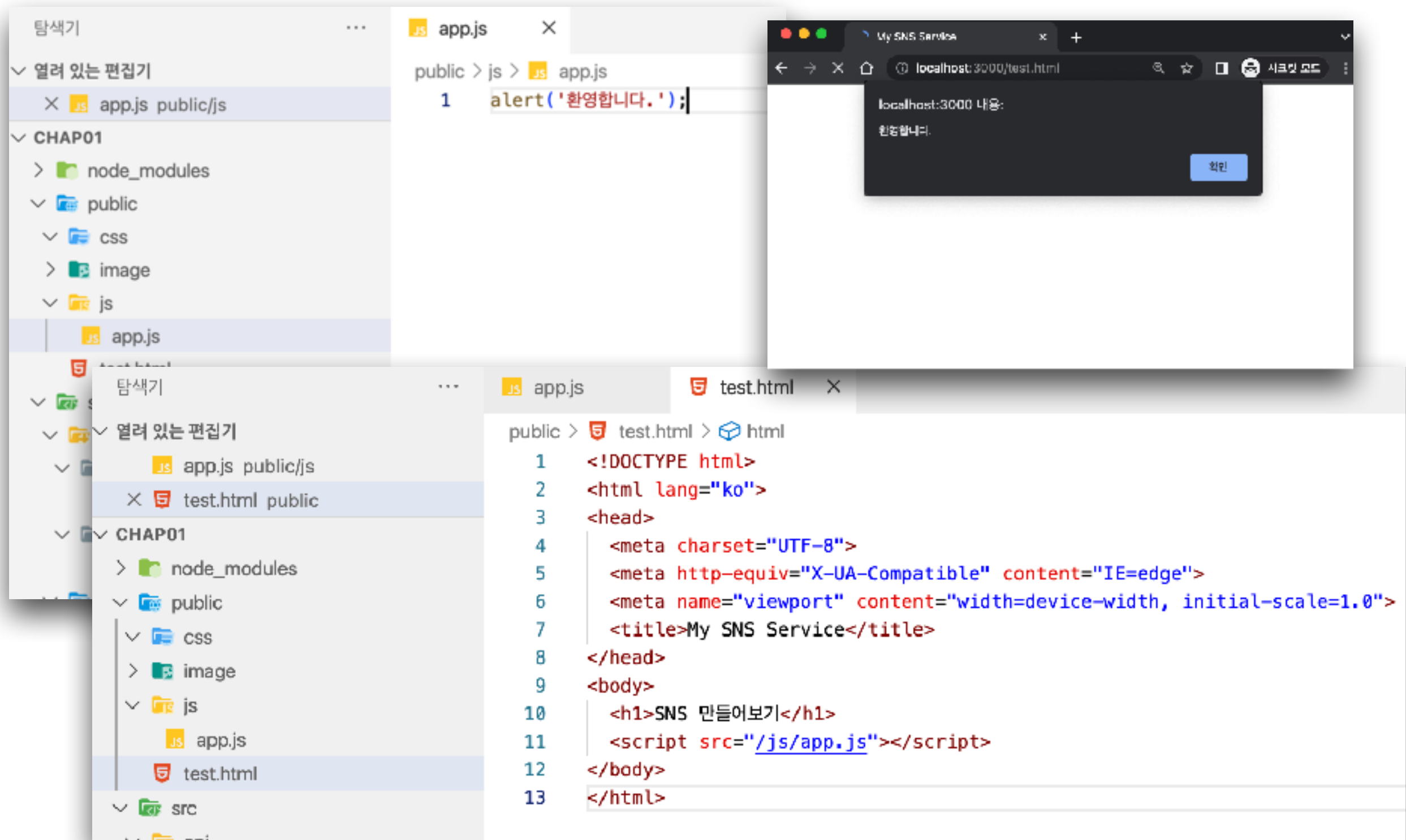
## test.html 파일 생성 및 실행





# 정적 파일들 접근

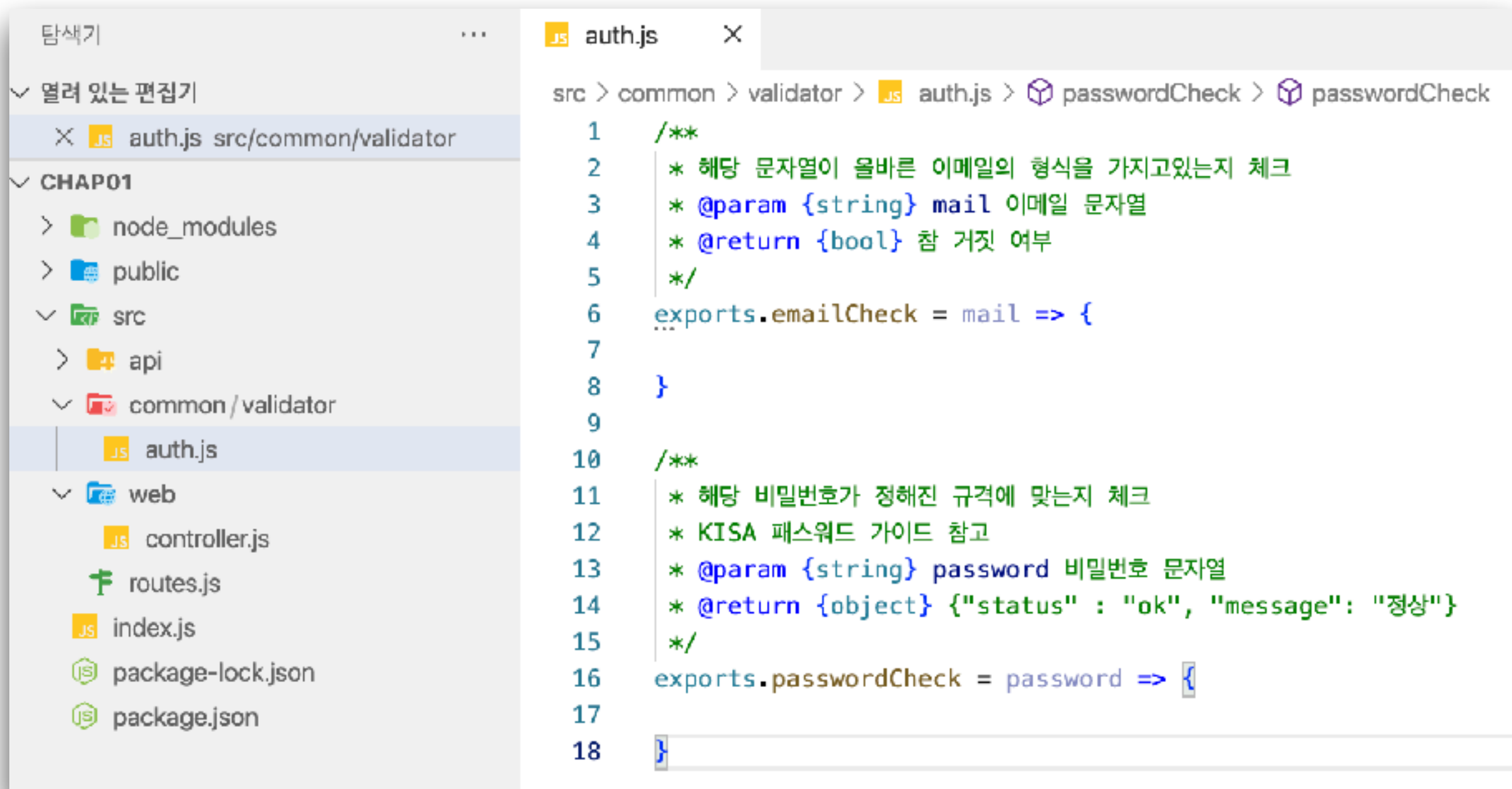
browser 에서 실행되는 javascript



공동 사용 모듈

# 공통 모듈 작성

여러 파일 또는 여러 프로젝트에서 사용할 만한 모듈들



The image shows a code editor window with a file explorer on the left and a code editor on the right.

**File Explorer (Left):**

- 탐색기 (Search)
- 열려 있는 편집기 (Opened Editors):
  - auth.js src/common/validator
- CHAP01
  - node\_modules
  - public
  - src
    - api
    - common/validator
      - auth.js (Selected)
    - web
      - controller.js
      - routes.js
      - index.js
      - package-lock.json
      - package.json

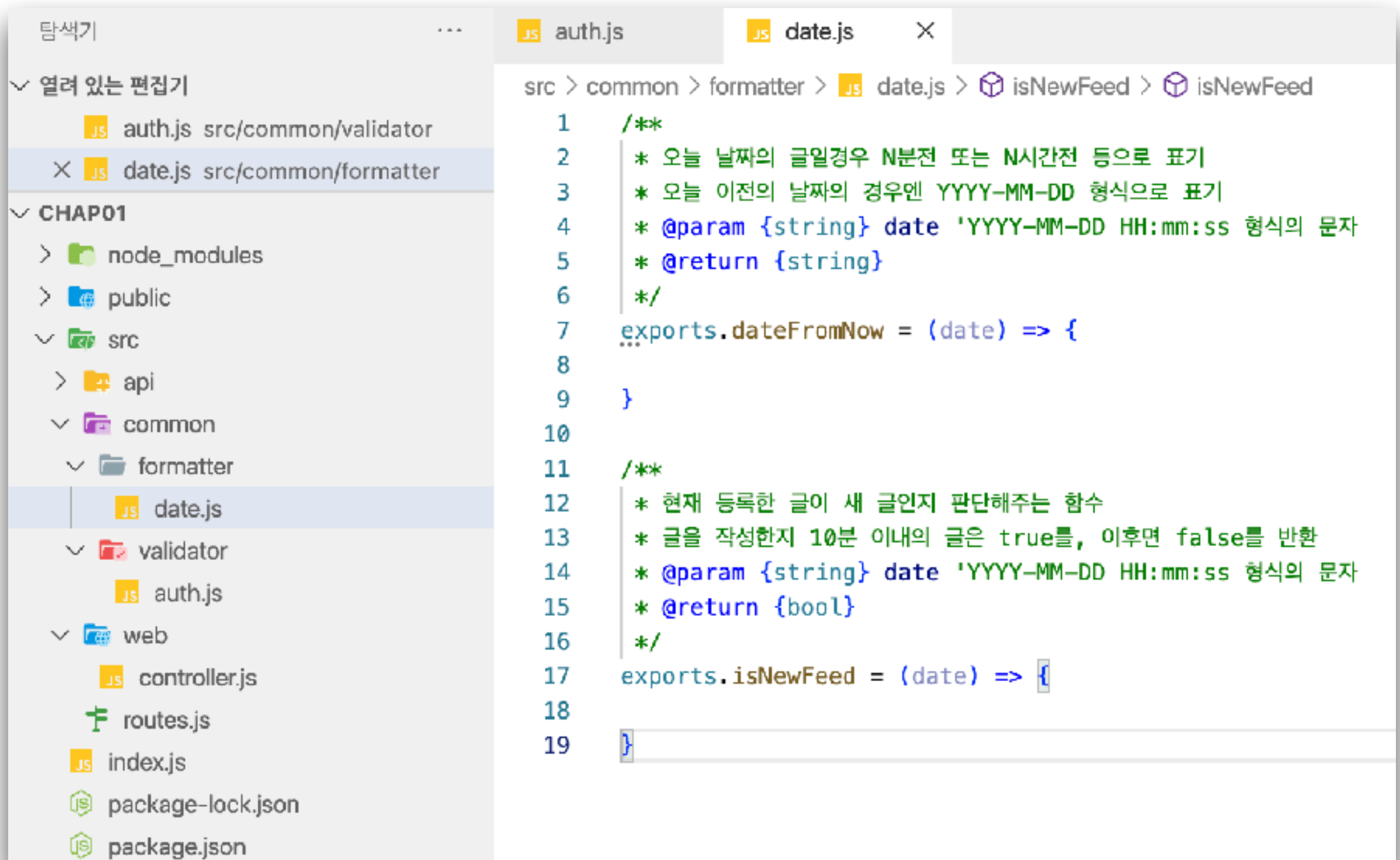
**Code Editor (Right):**

src > common > validator > auth.js > passwordCheck > passwordCheck

```
1  /**
2   * 해당 문자열이 올바른 이메일의 형식을 가지고있는지 체크
3   * @param {string} mail 이메일 문자열
4   * @return {bool} 참 거짓 여부
5   */
6  exports.emailCheck = mail => {
7
8  }
9
10 /**
11 * 해당 비밀번호가 정해진 규격에 맞는지 체크
12 * KISA 패스워드 가이드 참고
13 * @param {string} password 비밀번호 문자열
14 * @return {object} {"status" : "ok", "message": "정상"}
15 */
16 exports.passwordCheck = password => {
17
18 }
```

# 공통 모듈 작성

## 날짜 출력관련 포맷터 작성



탐색기

열려 있는 편집기

- auth.js src/common/validator
- date.js src/common/formatter

CHAP01

- node\_modules
- public
- src
  - api
  - common
    - formatter
      - date.js
    - validator
      - auth.js
  - web
    - controller.js
    - routes.js
    - index.js
- package-lock.json
- package.json

src > common > formatter > date.js > isNewFeed > isNewFeed

```
1  /**
2   * 오늘 날짜의 글일경우 N분전 또는 N시간전 등으로 표기
3   * 오늘 이전의 날짜의 경우엔 YYYY-MM-DD 형식으로 표기
4   * @param {string} date 'YYYY-MM-DD HH:mm:ss' 형식의 문자
5   * @return {string}
6   */
7  exports.dateFromNow = (date) => {
8
9  }
10
11 /**
12 * 현재 등록된 글이 새 글인지 판단해주는 함수
13 * 글을 작성한지 10분 이내의 글은 true를, 이후면 false를 반환
14 * @param {string} date 'YYYY-MM-DD HH:mm:ss' 형식의 문자
15 * @return {bool}
16 */
17 exports.isNewFeed = (date) => {
18
19 }
```