21. 백앤드프로그래밍 : Node.js의 Koa프레임워크

21.1 소개하기

21.1.1 백엔드

- 백엔드프로그래밍은 여러가지 환경으로 진행할 수가 있는데 그 중 javascript로 서버를 구현할 수 있는 Node.js를 사용

21.1.2 Node.js

- 구글이 크롬웹브라어저의 V8 자바스크립트엔진을 공개하면서 서버에서 실행가능한 런타임용 자바스크립트를 개발. 이 것이 Node.js

21.1.3 Koa

- Node.js환경에서 웹서버구축할 때 Express, Hapi, Koa등의 웹프레임워크를 사용
- Koa는 Express의 개발팀이 개발한 프레임워크
- Express는 미들웨어, 라우팅, 템플릿, 파일호스팅과 같은 다양한 기능이 내장되어 있다.
- Koa는 미들웨어만 있고 다른 라이브러리를 적용하기 때문에 Express보다 훨씬 가볍다.
- 추가로 async/await문법을 지원하기 때문에 비동기작업을 편하게 할 수 있다.

21.2 작업환경준비

1. `yarn init -y`
2. cat package.json

```
{
"name": "21.koa",
"version": "1.0.0",
"main": "index.js",
"license": "MIT"
}
```

3. yarn add koa

```
{
"name": "21.koa",
"version": "1.0.0",
"main": "index.js",
"license": "MIT",
"dependencies": {
 "koa": "^2.14.2"
}
}
```

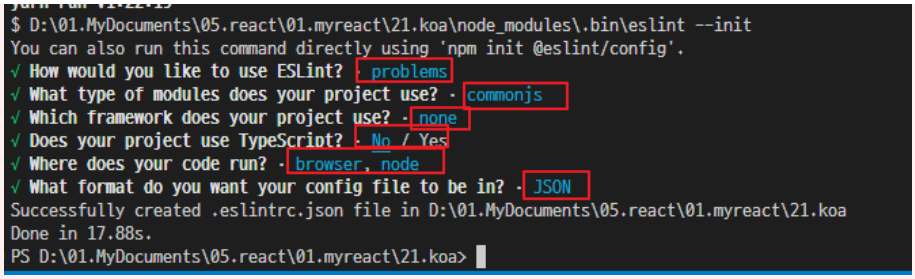
21.2.3 ESLint와 Prettier설정

- VSCode에서 사용하려면 확장 프로그램을 설치해 둔 상태이어야 한다.

1. yarn add --dev eslint
   - `--dev`는 개발용 의존 모듈로 설치 한다은 의미, package.json의 devDependencies에 버전정보가 입력된다.
2. yarn run eslint --init

```
{
"name": "21.koa",
"version": "1.0.0",
"main": "index.js",
"license": "MIT",
"dependencies": {
 "koa": "^2.14.2"
},
"devDependencies": {
 "eslint": "^8.38.0"
}
}
```

3. yarn run eslint --init



.eslintrc.json

```
{
 "env": {
     "browser": true,
     "commonjs": true,
     "es2021": true,
     "node": true
 },
 "extends": "eslint:recommended",
 "overrides": [
 ],
 "parserOptions": {
     "ecmaVersion": "latest"
 },
 "rules": {
 }
}
```

4. Prettier설정 - `.prettierrc 파일작성` ##### .prettierrc

```
{
"singleQuote": true,
"semi": true,
"useTabs": false,
```

```
"tabWidth": 2,
"trailingComma": "all",
"printWidth": 80
}
```

5. yarn add eslint-config-prettier

   - prettier에서 관리하는 코드스타일은 ESLint에서 관리하지 않도록 적용 ##### .eslintrc.json

```
{
"env": {
 "commonjs": true,
 "es6": true,
 "node": true
},
"extends": ["eslint:recommended", "prettier"],
"globals": {
 "Atomics": "readonly",
 "SharedArrayBuffer": "readonly"
},
"parserOptions": {
 "ecmaVersion": 2018
},
"rules": {
}
}
```

6. 작동여부 확인

src/index.js

```
const hello = "hello";
```

- const값을 선언하고 사용하지 않으면 ESLint 기본설정은 이를 에러로 간주
- 이 상황을 비활성화하려면 .eslintrc.json를 수정 ##### .eslintrc.json

```
{
  "env": {
    "commonjs": true,
    "es6": true,
    "node": true
  },
  "extends": ["eslint:recommended", "prettier"],
  "globals": {
    "Atomics": "readonly",
    "SharedArrayBuffer": "readonly"
  },
  "parserOptions": {
    "ecmaVersion": 2018
  },
"rules": {
  "no-unused-vars": "warn",
  "no-console": "off"
}
}
```

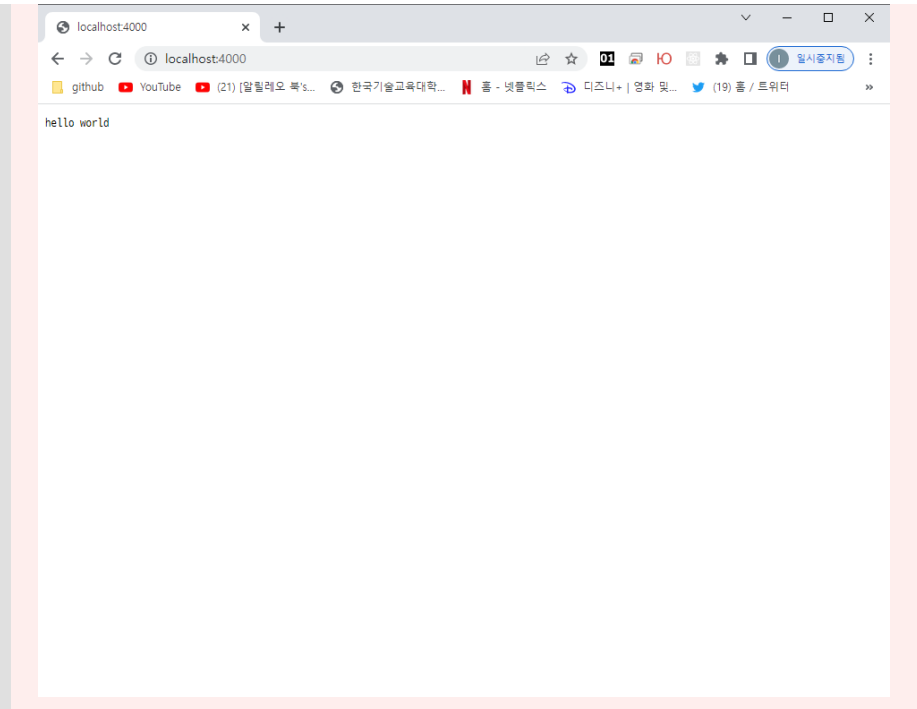
21.3 Koa 기본사용법

21.3.1 서버띄우기

src/index.js

```
const Koa = require("koa");

const app = new Koa();

app.use(ctx => {
  ctx.body = "hello world!!!";
});

app.listen(4000, () => {
  console.log('Listening to port 4000')
});
```

- 서버실행
  - node를 통해 js를 실행할 때 전체경로를 지정하지만 index.js는 예외로 디렉토리까지만 입력해도 된다.
  - `node src` or `node src\index.js`



21.3.2 미들웨어

- Koa 애플리케이션은 `미들웨어 배열로 구성` 되어 있다.
  - app.use()는 미들웨어함수를 애플리케이션에 등록한다.
- 미들웨어함수는 다음과 같은 구조로 되어 있다.

```
(ctx, next) => {
  ... 실행문 ...
}
```

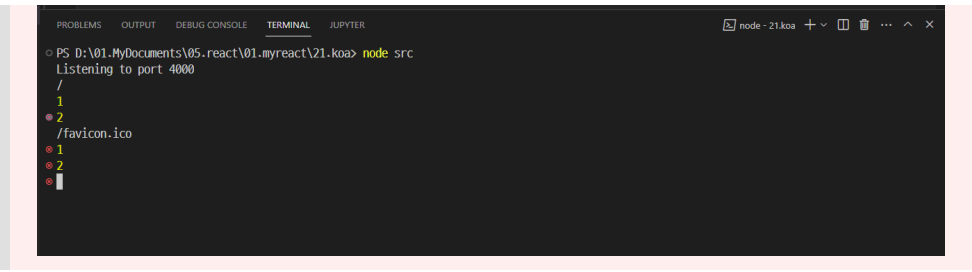
- Koa의 미들웨어함수는 ctx, next 2개의 파라미터를 받는다.
  - ctx는 Context로 웹요청과 응답에 관한 정보를 저장
  - next는 현재 처리중인 미들웨어의 다음 미들웨어를 호출하는 함수
  - next함수를 호출하지 않으면 그 다음 미들웨어는 처리되지 않는다.
  - next함수를 정의하지 않을 경우 `ctx => {}` 형태로 정의한다.

- 미들웨어는 app.use()함수로 등록되는 순서대로 처리된다.

src/index.js

```
const Koa = require("koa");

const app = new Koa();

app.use((ctx, next) => {
  console.log(ctx.url);
  console.log(1);
  next()
});

app.use((ctx, next) => {
  console.log(2);
  next();
});

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

app.listen(4000, () => {
  console.log('Listening to port 4000')
});
```

- 서버종료후 재시작

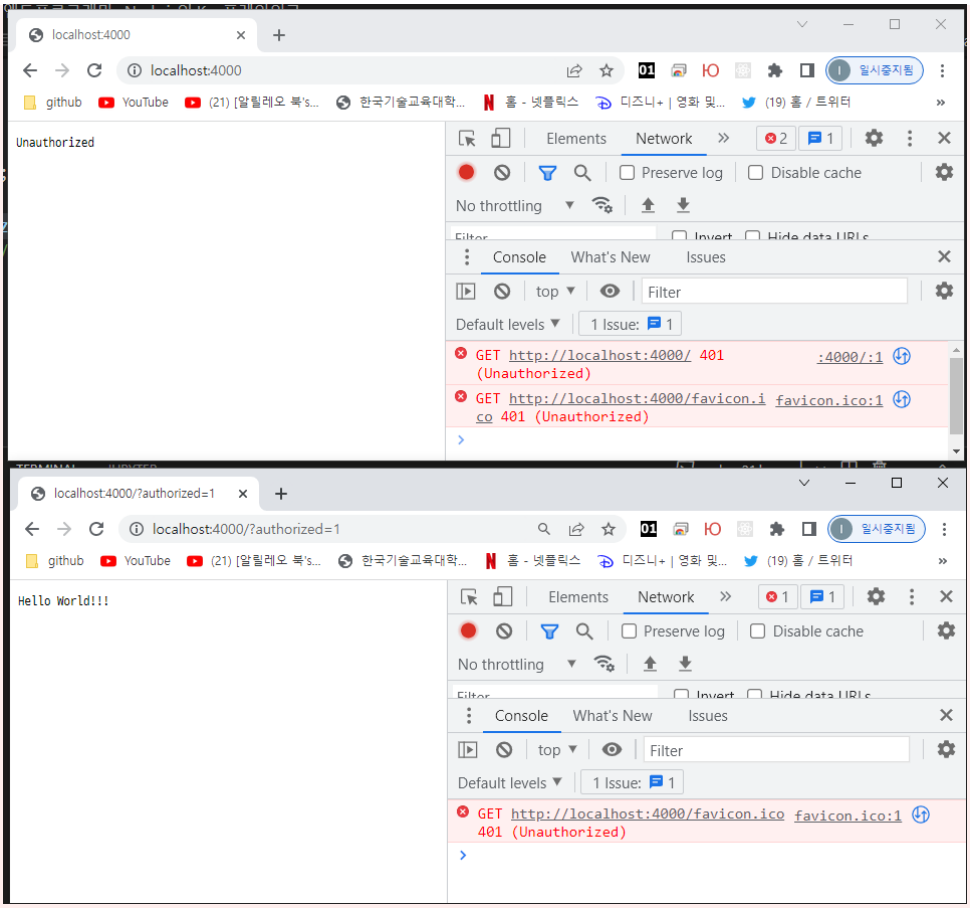


- 첫 번째 미들웨어 next()함수를 주석처리하면 다음 미들웨어함수는 호출되지 않는다.

- 요청경로에 `authorized=1` 쿼리파라미터 테스트 ##### src/index.js

```
const Koa = require("koa");

const app = new Koa();

app.use((ctx, next) => {
  console.log(ctx.url);
  console.log(1);
  if(ctx.query.authorized !== '1') {
    ctx.status = 401; // Unauthorized
    return;
  }
  next()
});

app.use((ctx, next) => {
```

```
  console.log(2);
  next();
});

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

app.listen(4000, () => {
  console.log('Listening to port 4000')
});
```



21.3.2.1 next함수는 promise를 반환

- next함수를 호출하면 Promise를 반환한다. 이는 Koa가 Exress와 차별화되는 부분이다.
- next가 반환하는 Promise는 다음에 처리할 미들웨어가 끝나야 종료가 된다.

src/index.js - next().tnen()

```
const Koa = require("koa");

const app = new Koa();

app.use((ctx, next) => {
  console.log(ctx.url);
  console.log(1);
  if(ctx.query.authorized !== '1') {
    ctx.status = 401; // Unauthorized
    return;
  }
```

```
  next().then(() => {
    console.log('END');
  })
});

app.use((ctx, next) => {
  console.log(2);
  next();
});

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

app.listen(4000, () => {
  console.log('Listening to port 4000')
});
```

21.3.2.2 async/await 사용하기

- Koa는 async/await를 정식으로 지원한다.
- 기존 코드를 async/await를 사용하여 수정

src/index.js - async/await

```
const Koa = require("koa");

const app = new Koa();

app.use(async (ctx, next) => {
  console.log(ctx.url);
  console.log(1);
  if(ctx.query.authorized !== '1') {
    ctx.status = 401; // Unauthorized
    return;
  }
  await next();
  console.log('-------------- END -----------------');
});

app.use((ctx, next) => {
  console.log(2);
  next();
});

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

app.listen(4000, () => {
  console.log('Listening to port 4000')
});
```

In [ ]:

21.4 nodemon 사용하기

- 설치 : `yarn add --dev nodemon`
- package.json 수정 ##### package.json

```
{
"name": "21.koa",
"version": "1.0.0",
"main": "index.js",
"license": "MIT",
"dependencies": {
  "eslint-config-prettier": "^8.8.0",
  "koa": "^2.14.2"
},
"devDependencies": {
  "eslint": "^8.38.0",
  "nodemon": "^2.0.22"
},
"scripts": {
  "start": "node src",
  "start:dev": "nodemon --watch src/ src/index.js"
}
}
```

- "start": "node src"는 서버시작명령
- "start:dev": "nodemon --watch src/ src/index.js"는 nodemon을 통해 서버를 실행하는 명령
- 이제 부터는 다음 명령어를 사용할 수 있다.
  - `yarn start` : 재시작이 필요없을 때
  - `yarn start:dev` : 재시작이 필요할 때
- 서버종료후 yarn start:dev를 실행
- index.js를 수정후 저장하면 서버 자동으로 재실행

src/index.js - async/await

```
const Koa = require("koa");

const app = new Koa();

app.listen(4000, () => {
  console.log('Listening to port 4000')
});
```

21.5 koa-router 사용하기

- 설치: yarn add koa-router

21.5.1 기본사용법

- index.js에 koa-router 적용하기

src/index.js - koa-router

```
const Koa = require("koa");
const Router = require('koa-router');
```

```
const app = new Koa();
const router = new Router();

// 라우터설정
router.get('/', ctx => {
  ctx.body = "<h1>Home page</h1>";
});

router.get('/about', ctx => {
  ctx.body = "<h1>About page</h1>";
});

// app 인스턴스에 라우터 적용
app.use(router.routes()).use(router.allowedMethods());

app.listen(4000, () => {
  console.log('Listening to port 4000')
});
```

21.5.2 라우트 파라미터와 쿼리

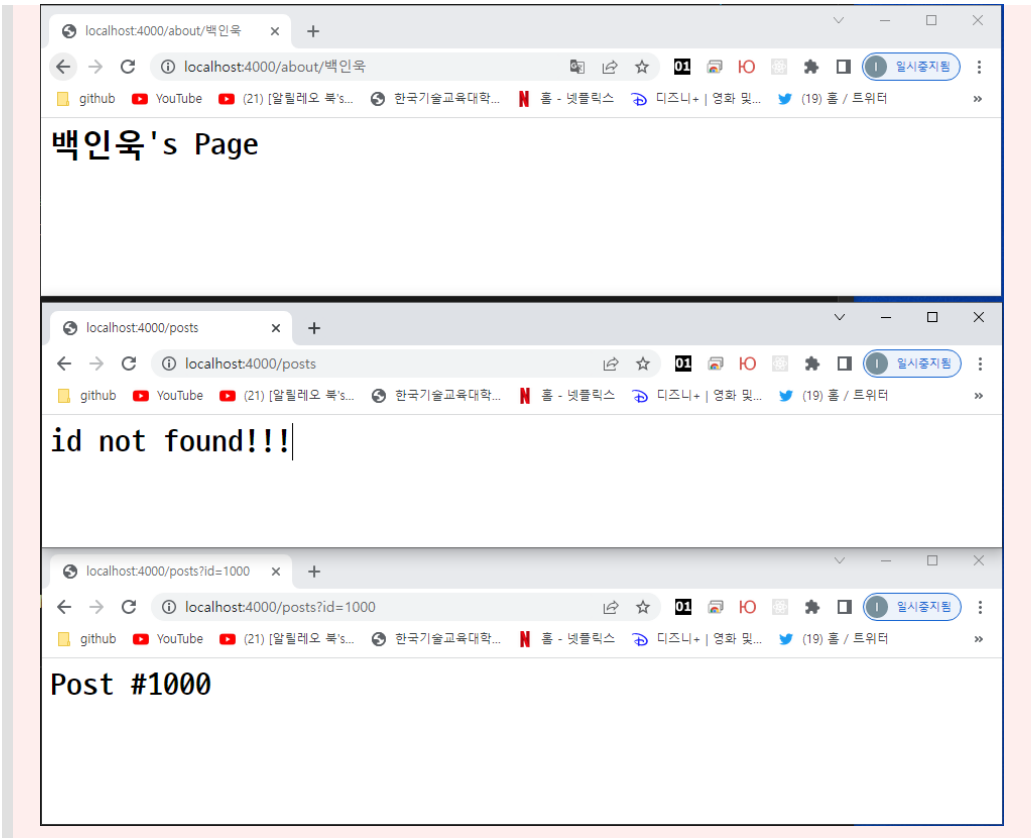
- 라우트 파라미터를 설정할 때 `about/:name`처럼 콜론:을 사용하여 경로를 설정한다.
- `파라미터가 있을 수도 있고 없을 수도 있다면 about/:name? 처럼 파라미터뒤에 ?를 사용`한다.
- 이렇게 설정한 파라미터는 `함수 ctx.params 객체에서 조회`할 수 있다.

src/index.js - 파라미터와 쿼리

```
const Koa = require("koa");
const Router = require('koa-router');

const app = new Koa();
const router = new Router();

// 라우터설정
router.get('/', ctx => {
  ctx.body = "<h1>Home page</h1>";
});

router.get('/about/:name?', ctx => {
  const { name } = ctx.params;
  // name의 존재유무에 따라 다른 결과를 출력
  ctx.body = name ? `<h1>${name}'s Page</h1>` : '<hAbout Page</h1>';
});

// router.get('/about', ctx => {
//   ctx.body = "<h1>About page</h1>";
// });

router.get('/posts', ctx => {
  const { id } = ctx.query;
  // id존재유무확인
  ctx.body = id ? `<h1>Post #${id}</h1>` : '<h1>id not found!!!</h1>';
});
```

```
// app 인스턴스에 라우터 적용
app.use(router.routes()).use(router.allowedMethods());

app.listen(4000, () => {
  console.log('Listening to port 4000')
});
```

- http://localhost:4000/about/gilbaek
- http://localhost:4000/posts
- http://localhost:4000/posts?id=1000



21.5.3 RESET API

- RESET API는 요청에 따라 get, post, delete, put, patch를 사용한다.
- RESET API
  - POST /posts
  - GET /posts
  - GET /posts/:id
  - DELETE /posts/:id
  - PATCH /posts/:id
  - GET /posts/:id/comments
  - DELETE /posts/:id/comments/:comentsId

21.5.4 라우트 모듈화

src/api/index.js

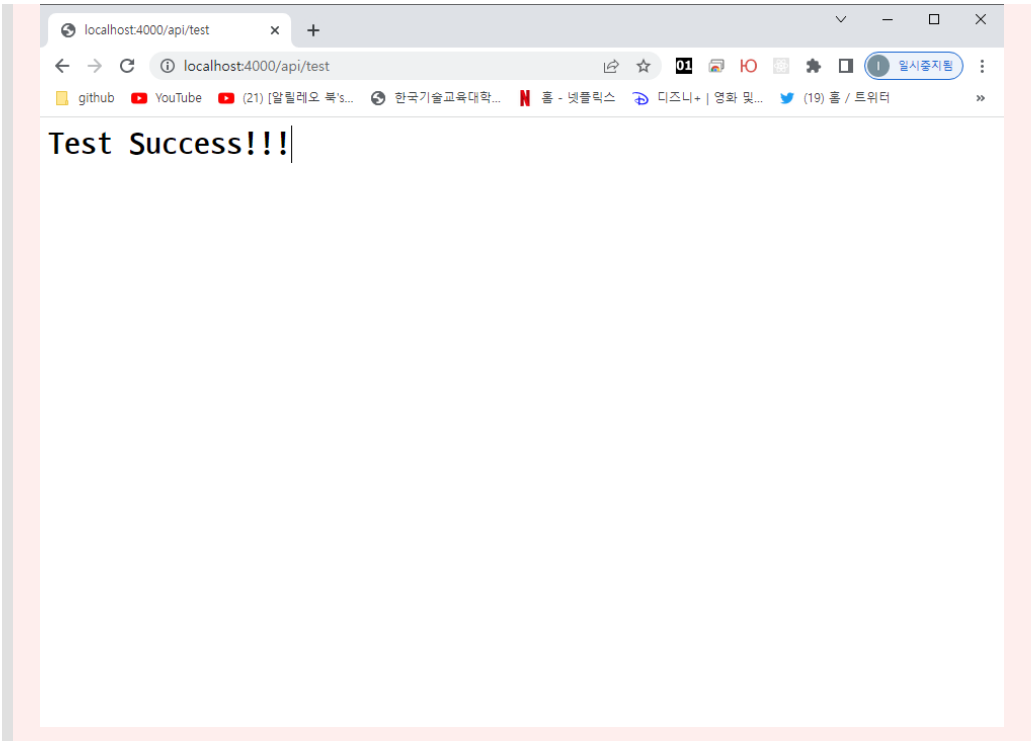
```
const Router = require('koa-router');
  const api = new Router();

  api.get('/test', ctx => {
```

```
  ctx.body = "<h1>Test Success!!!</h1>";
});

// 라우터 내보내기
module.exports = api;
```

src/index.js - 라우트모듈화

```
const Koa = require("koa");
const Router = require('koa-router');

const api = require('./api');

const app = new Koa();
const router = new Router();

// 라우터설정
router.use('/api', api.routes()); // api 라우트 적용

// app 인스턴스에 라우터 적용
app.use(router.routes()).use(router.allowedMethods());

app.listen(4000, () => {
  console.log('Listening to port 4000')
});
```

- http://localhost/api/test



21.5.5 posts 라우트 생성

src/api/posts/index.js

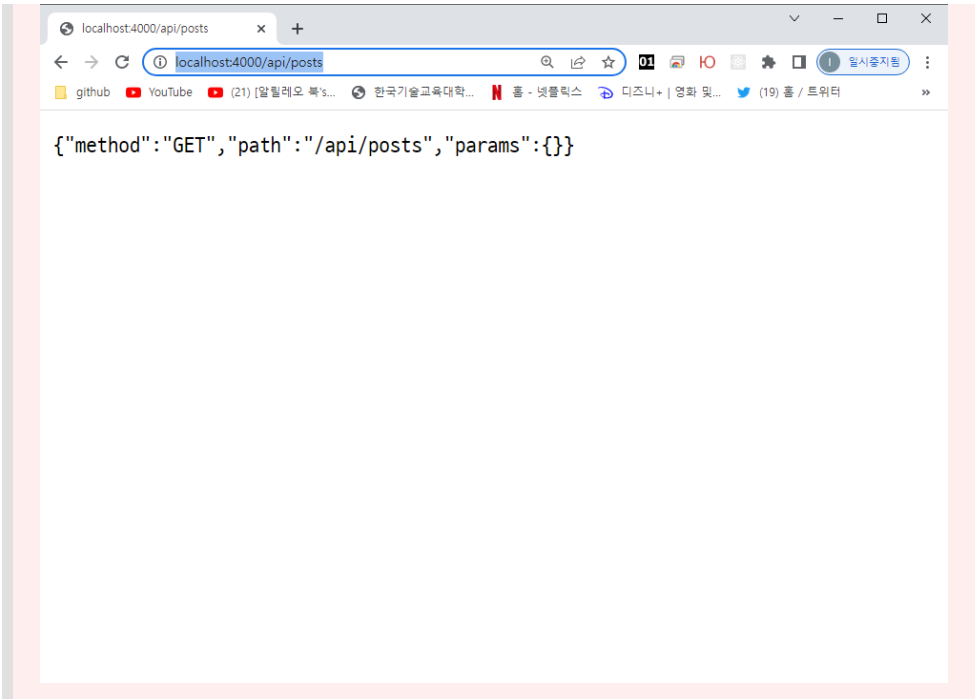
```
const Router = require('koa-router');

const posts = new Router();

const printInfo = ctx => {
```

```
    ctx.body = {
      method: ctx.method,
      path: ctx.path,
      params: ctx.params
    }
}

posts.get('/', printInfo);
posts.post('/',printInfo);
posts.get('/:id', printInfo);
posts.delete('/:id', printInfo);
posts.put('/:id', printInfo);
posts.patch('/:id',printInfo);

module.exports = posts;
```

src/api/index.js

```
const Router = require('koa-router');
  const posts = require('./posts')

  const api = new Router();

  api.get('/test', ctx => {
    ctx.body = "<h1>Test Success!!!</h1>";
  });

  api.use('/posts', posts.routes());

  // 라우터 내보내기
  module.exports = api;
```

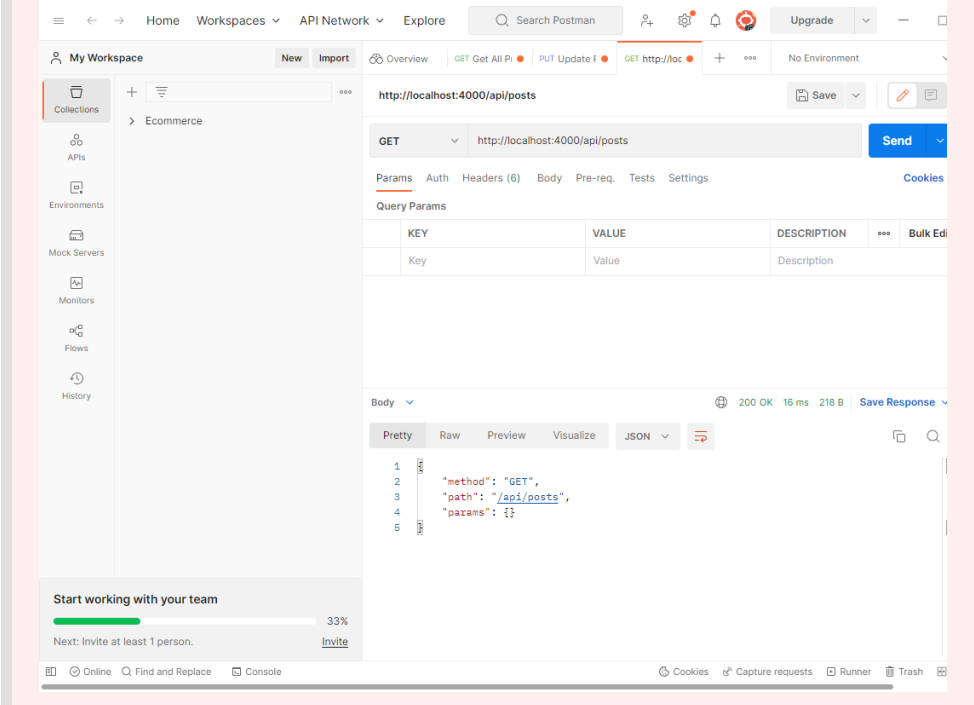
- http://localhost:4000/api/posts



21.5.5.1 Postman설치 및 사용

- 다운노드 & 설치 : https://www.getpostman.com

- postman 실행
  - get : http://localhost:4000/api/posts
  - patch : http://localhost:4000/api/posts/10
  - put : http://localhost:4000/api/posts/10
  - delete: http://localhost:4000/api/posts/10



21.5.5.2 컨트롤러파일 작성

- 임시로 자바스크립트의 array기능을 사용하여 구현
- koa-bodyparser를 설치, Request Body에 JSON형식으로 데이터를 전달하면 파싱
- `yarn add koa-bodyparser`

src/index.js

```
const Koa = require("koa");
const Router = require('koa-router');
const bodyParser = require('koa-bodyparser');

const api = require('./api');

const app = new Koa();
const router = new Router();

// 라우터설정
router.use('/api', api.routes()); // api 라우트 적용

// 라우터적용전에 bodyParser적용
app.use(bodyParser());

// app 인스턴스에 라우터 적용
app.use(router.routes()).use(router.allowedMethods());

app.listen(4000, () => {
```

```
  console.log('Listening to port 4000')
});
```

src/api/posts/posts.ctrl.js

```
let postId = 1; // id의 초깃값입니다.

// posts 배열 초기 데이터
const posts = [
  {
    id: 1,
    title: '제목',
    body: '내용',
  },
];

/* 포스트 작성
POST /api/posts
{ title, body }
*/
exports.write = ctx => {
  // REST API의 request body는 ctx.request.body에서 조회할 수 있습니다.
  const { title, body } = ctx.request.body;
  postId += 1; // 기존 postId 값에 1을 더합니다.
  const post = { id: postId, title, body };
  posts.push(post);
  ctx.body = post;
};

/* 포스트 목록 조회
GET /api/posts
*/
exports.list = ctx => {
  ctx.body = posts;
};

/* 특정 포스트 조회
GET /api/posts/:id
*/
exports.read = ctx => {
  const { id } = ctx.params;
  // 주어진 id 값으로 포스트를 찾습니다.
  // 파라미터로 받아 온 값은 문자열 형식이니 파라미터를 숫자로 변환하거나,
  // 비교할 p.id 값을 문자열로 변경해야 합니다.
  const post = posts.find(p => p.id.toString() === id);
  // 포스트가 없으면 오류를 반환합니다.
  if (!post) {
    ctx.status = 404;
    ctx.body = {
      message: '포스트가 존재하지 않습니다.',
    };
    return;
  }
  ctx.body = post;
};

/* 특정 포스트 제거
```

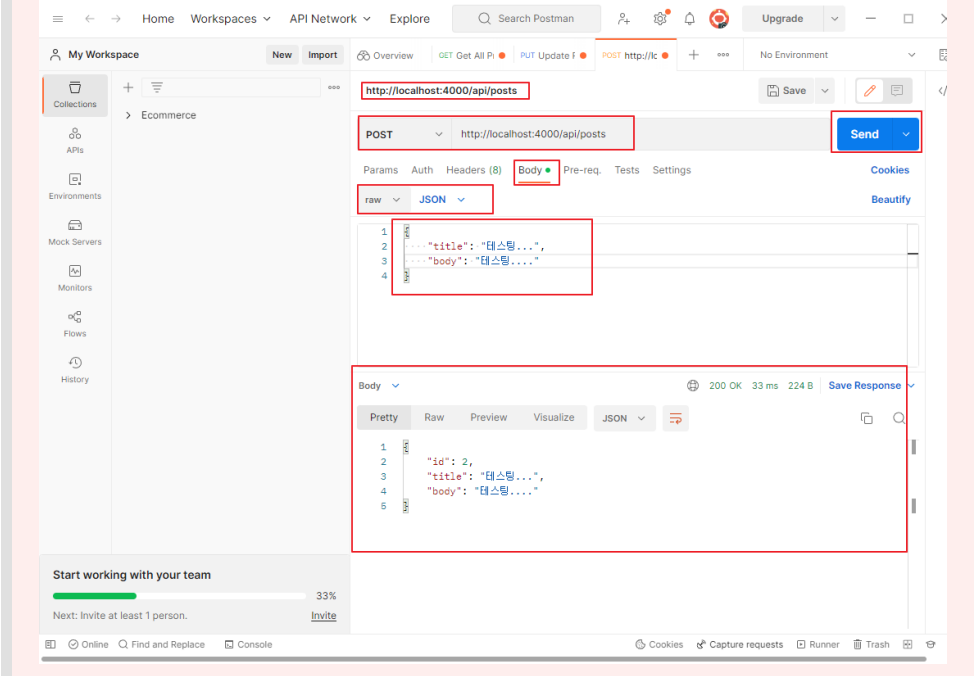
```
DELETE /api/posts/:id
*/
exports.remove = ctx => {
  const { id } = ctx.params;
  // 해당 id를 가진 post가 몇 번째인지 확인합니다.
  const index = posts.findIndex(p => p.id.toString() === id);
  // 포스트가 없으면 오류를 반환합니다.
  if (index === -1) {
    ctx.status = 404;
    ctx.body = {
      message: '포스트가 존재하지 않습니다.',
    };
    return;
  }
  // index번째 아이템을 제거합니다.
  posts.splice(index, 1);
  ctx.status = 204; // No Content
};

/* 포스트 수정(교체)
PUT /api/posts/:id
{ title, body }
*/
exports.replace = ctx => {
  // PUT 메서드는 전체 포스트 정보를 입력하여 데이터를 통째로 교체할 때 사용합
니다.
  const { id } = ctx.params;
  // 해당 id를 가진 post가 몇 번째인지 확인합니다.
  const index = posts.findIndex(p => p.id.toString() === id);
  // 포스트가 없으면 오류를 반환합니다.
  if (index === -1) {
    ctx.status = 404;
    ctx.body = {
      message: '포스트가 존재하지 않습니다.',
    };
    return;
  }
  // 전체 객체를 덮어씌웁니다.
  // 따라서 id를 제외한 기존 정보를 날리고, 객체를 새로 만듭니다.
  posts[index] = {
    id,
    ...ctx.request.body,
  };
  ctx.body = posts[index];
};

/* 포스트 수정(특정 필드 변경)
PATCH /api/posts/:id
{ title, body }
*/
exports.update = ctx => {
  // PATCH 메서드는 주어진 필드만 교체합니다.
  const { id } = ctx.params;
  // 해당 id를 가진 post가 몇 번째인지 확인합니다.
  const index = posts.findIndex(p => p.id.toString() === id);
  // 포스트가 없으면 오류를 반환합니다.
  if (index === -1) {
```

```
    ctx.status = 404;
    ctx.body = {
      message: '포스트가 존재하지 않습니다.',
    };
    return;
  }
  // 기존 값에 정보를 덮어씌웁니다.
  posts[index] = {
    ...posts[index],
    ...ctx.request.body,
  };
  ctx.body = posts[index];
};
```

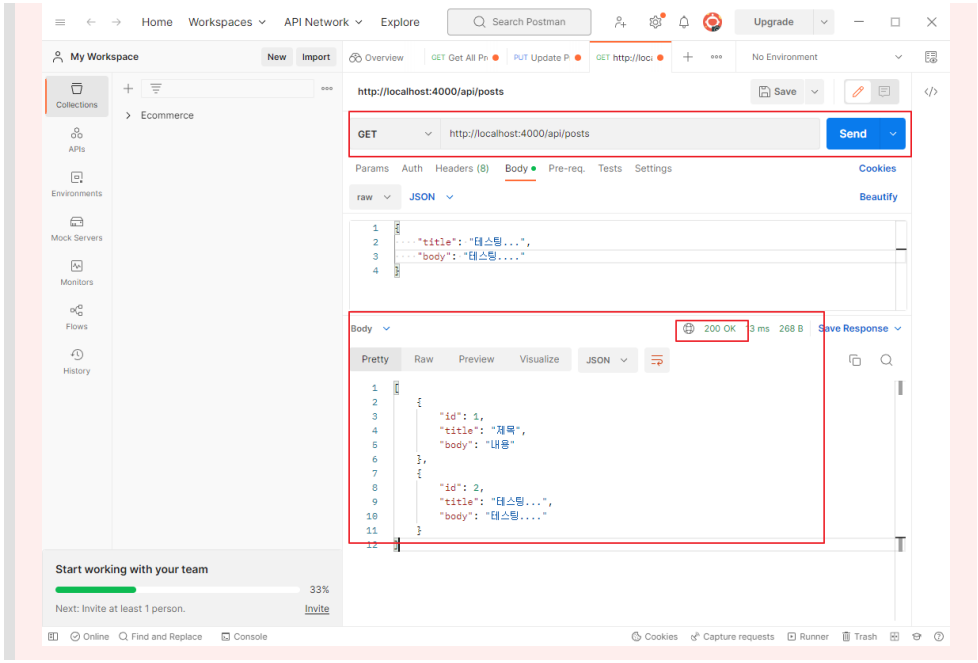
src/api/posts/index.js

```
const Router = require('koa-router');
const postsCtrl = require('./posts.ctrl');

const posts = new Router();

posts.get('/', postsCtrl.list);
posts.post('/', postsCtrl.write);
posts.get('/:id', postsCtrl.read);
posts.delete('/:id', postsCtrl.remove);
posts.put('/:id', postsCtrl.replace);
posts.patch('/:id', postsCtrl.update);

module.exports = posts;
```

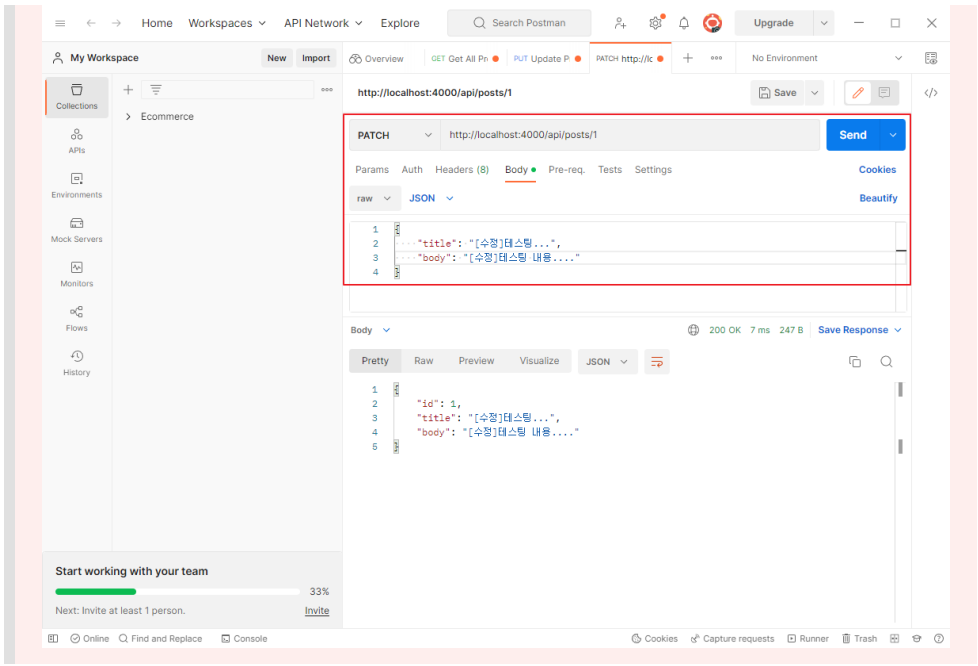
- POST : http://localhost:4000/api/posts



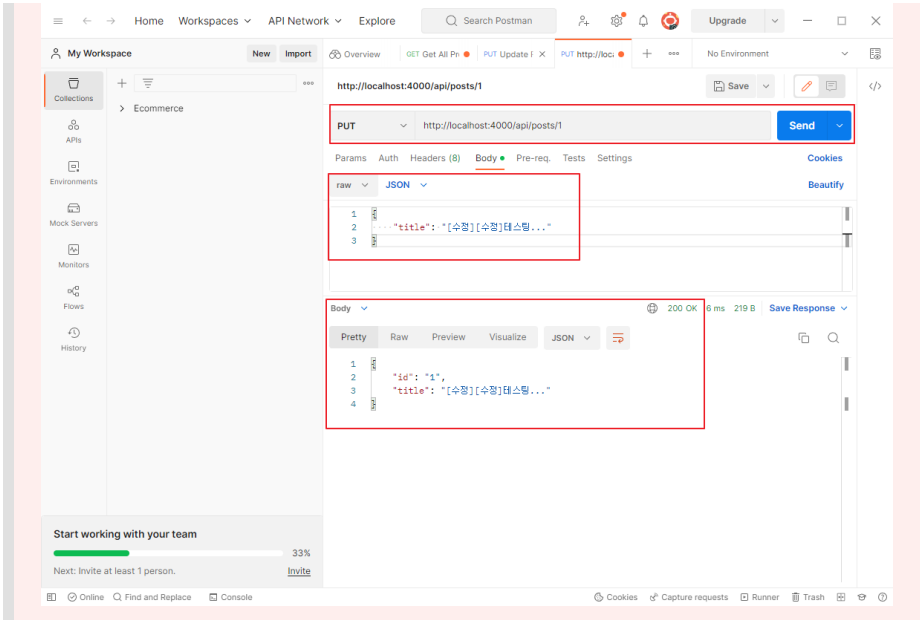
- GET : http://localhost:4000/api/posts



- PATCH : http://localhost:4000/api/posts/1



- PUT : http://localhost:4000/api/posts/1
  - title만 수정할 경우 body데이터는 삭제가 되기 때문에 put방식일 경우 전체열을 지정해야 한다.


Home
Workspaces
API Network
Explore
Search Postman
Upgrade
My Workspace
New
Import
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
Ecommerce
Overview
No Environment
http://localhost:4000/api/posts/1
Save
PUT
http://localhost:4000/api/posts/1
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
"title": "[수정][수정]테스팅..."
Body
200 OK
6 ms
219 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
"id": "1",
"title": "[수정][수정]테스팅..."
Start working with your team
33%
Next: Invite at least 1 person.
Invite
Online
Find and Replace
Console
Cookies
Capture requests
Runner
Trash