웹 프로그래밍

4. To list 작성 심화.

류영표 강사
ryp1662@gmail.com

강사 소개

류영표

Youngpyo Ryu

동국대학교 수학과/응용수학 석사수료

前 Upstage AI X 네이버 부스트 캠프 AI tech 1~ 6기 멘토

前 Innovation on Quantum & CT(IQCT) 이사

前 한국파스퇴르연구소 Image Mining 인턴(Deep learning)

前 ㈜셈웨어(수학컨텐츠, 데이터 분석 개발 및 연구인턴)

강의 경력

- 현대자동차 연구원 강의 (인공지능/머신러닝/딥러닝/강화학습)
- ㈜ 모두의연구소 Aiffel 1기 퍼실리테이터(인공지능 교육)
- 인공지능 자연어처리(NLP) 기업데이터 분석 전문가 양성과정 멘토
- 공공데이터 청년 인턴 / SW공개개발자대회 멘토
- 고려대학교 선도대학 소속 30명 딥러닝 집중 강의
- 이젠 종로 아카데미(파이썬, ADSP 강사)
- 최적화된 도구(R/파이썬)을 활용한 애널리스트 양성과정(국비과정) 강사
- 한화, 하나금융사, 한전 KDN, IBK 기업은행 교육
- 인공지능 신뢰성 확보를 위한 실무 전문가 자문 활동
- 인공지능 학습용 데이터 구축 사업 품질검증 전문가 자문 활동
- 보건 · 바이오 AI활용 S/W개발 및 응용전문가 양성과정 강사
- Upstage AI X KT 융합기술원 기업교육 모델최적화 담당 조교

주요 프로젝트 및 기타사항

- 의료 영상을 이용한 갑상선암 연관 유전변이 예측 플랫폼 개발 고도화
- 개인 맞춤형 당뇨병 예방·관리 인공지능 시스템 개발 및 고도화(안정화)
- 폐플라스틱 이미지 객체 검출 경진대회 3위
- 인공지능(AI)기반 데이터 사이언티스트 전문가 양성과정 1기 수료
- 제 1회 산업 수학 스터디 그룹 (질병에 영향을 미치는 유전자 정보 분석)
- 제 4,5회 산업 수학 스터디 그룹 (피부암, 유방암 분류)

패키지 설정

- 프로젝트 만들기 전에 터미널에 입력을 해보자.

- npm init -> package.json 생성(설치한 템플릿을 저장하는 용도)
- npm install express -> node_modules 폴더 설치
- npm install ejs --save -> html 기능을 하는 파일 확장자.

```
"dependencies": {
  "ejs": "^3.1.9",
  "express": "^4.18.2"
}
```

- package.json에 이런식으로 들어가게 됨.

Express 프레임워크

- Express는 node.js를 빠르고 간결하게 사용할 수 있게 해줌.
- 특징)

  - Express는 서버 사이드 프레임워크로 node.js의 api를 단순화하고, 유용한 기능을 추가해 쉽게 서버를 구축할 수 있게 해준다.

  - 코드의 양이 줄여주고 유지 보수가 쉽게 해준다.

  - 확장성을 지향한다. 불필요한 간섭이 없고 사용자가 필요한 라이브러리를 추가해서 확장 가능하다.

```
npm init
//package.json 파일을 작성한다
npm install express --save
//디렉토리에 Express를 설치하고 dependencies 목록에 추가한다.
```

```
PS C:\Users\young\Documents\Test\js> npm install express --save

added 64 packages, and audited 65 packages in 4s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

ejs 프레임워크

- HTML을 동적으로 생성하기 위한 템플릿 엔진 중 하나. 이는 JavaScript 코드를 HTML에 삽입하여 동적으로 웹 페이지를 생성할 수 있게 해줌.
- EJS는 Node.js와 함께 사용되며, Express와 같은 Node.js 웹 애플리케이션 프레임워크와 호환성이 뛰어남.

특징

1. 간편한 문법: EJS는 HTML에 JavaScript 코드를 삽입하기 위해 <% %>와 같은 간단한 문법을 사용합니다.
2. 재사용 가능한 코드: EJS는 템플릿으로서의 역할을 하므로, 동일한 코드 조각을 여러 페이지에서 재사용할 수 있습니다.
3. 프론트엔드와 백엔드의 통합: EJS를 사용하면 프론트엔드와 백엔드 로직을 하나의 파일에서 관리할 수 있어 개발 생산성을 향상시킵니다.
4. 프레임워크와의 호환성: 주요 Node.js 웹 프레임워크들과의 호환성이 높아 Express와 함께 많이 사용됩니다.

- <% 자바스크립트 코드 %>
- <%= 데이터 변수 %>

ejs 프레임워크

1. EJS 설치하기 : 먼저 프로젝트 디렉터리에서 npm을 사용하여 EJS를 설치함.

   npm install ejs

2. Express 애플리케이션에 EJS 설정하기 : Exrpess 애플리케이션을 생성하고 EJS를 사용하도록 설정.

```
const express = require('express');
const app = express();

// EJS를 템플릿 엔진으로 설정
app.set('view engine', 'ejs');
```

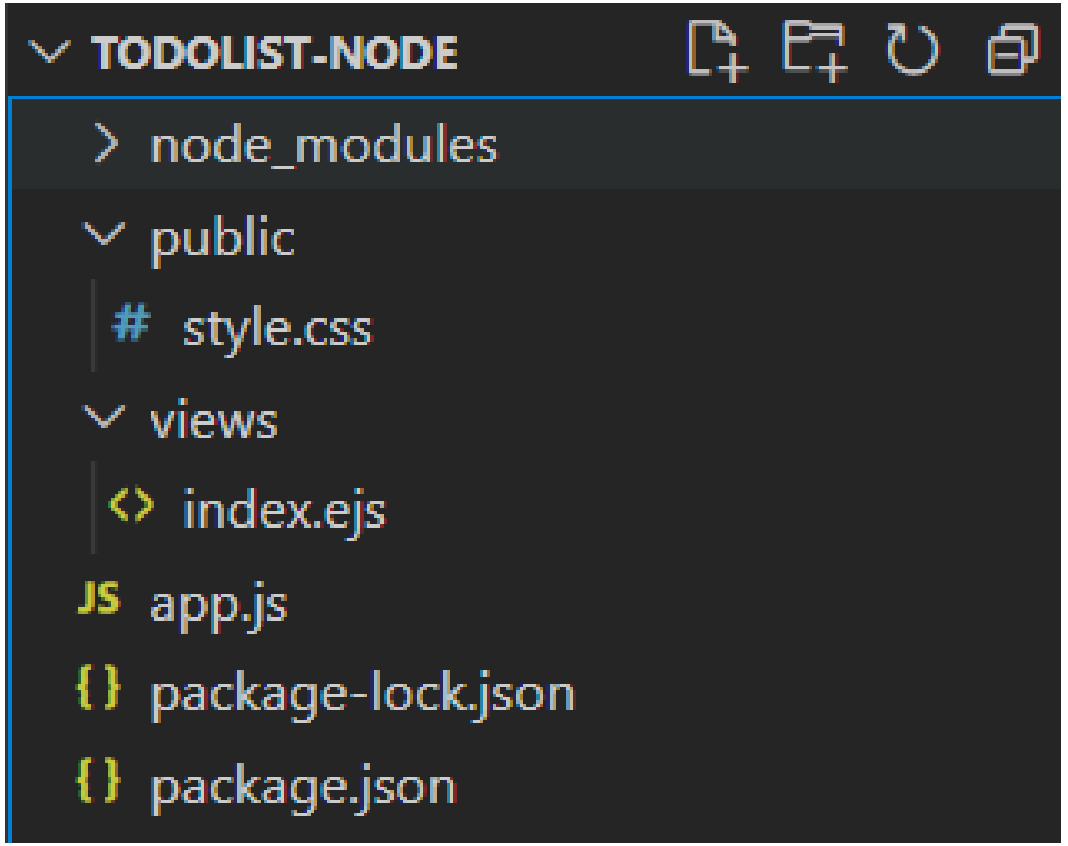
3. EJS 템플릿 작성하기 : .ejs 확장자를 가진 파일에 HTML과 JavaScript 코드를 함께 작성함.

```
<!-- views/index.ejs -->
<html>
<head>
    <title><%= title %></title>
</head>
<body>
    <h1>Hello, <%= name %>!</h1>
</body>
</html>
```

4. 템플릿 렌더링하기 : 라우팅을 설정하여 템플릿을 렌더링합니다.

```
app.get('/', (req, res) => {
    res.render('index', { title: 'Homepage', name: 'John' });
});
```

폴더 구조 설정



폴더 구조 이런식으로 짜주자.

Index.ejs와 style.css를 각각 public과 views라는 폴더에 넣어줘야 함.

서버 열기

App.js에 다음과 같이 입력한다.

```
const express = require("express");
const app = express();
const port = 3000;
```

- const express = require(“express”);

  express 모듈을 가져와 변수 express에 저장

- const app = express;

  express 앱을 생성, app 변수에 저장.

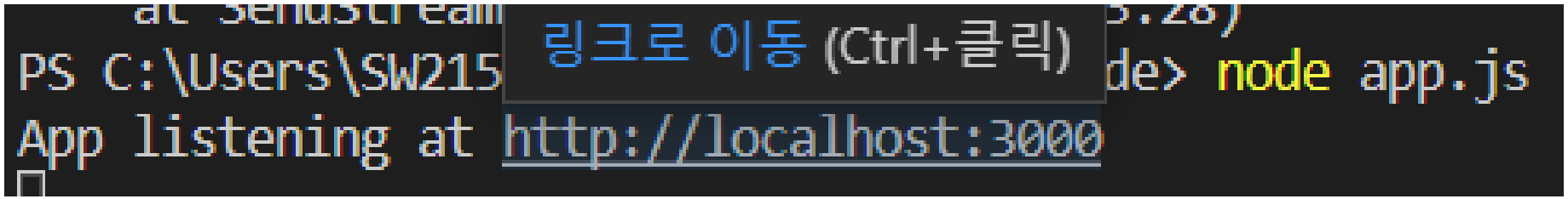
- const port = 3000;

  앱이 실행될 포트 번호를 설정.

서버 열기

```
app.listen(port, () => {
  console.log(`App listening at http://localhost:${port}`);
});
```

- 설정해둔 포트 번호(3000)으로 서버를 열어줌.



- 콘솔창의 링크를 crtl+클릭해서 열어둔 서버를 실행시키자.

Html, css연동

```
app.use(express.urlencoded({ extended: true }));
app.use(express.static("public"));
app.set("view engine", "ejs");
```

- app.use(express.urlencoder({extended : true});

  body-parse 설정, qs 모듈을 사용(중첩된 객체나 배열을 파싱할 수 있게 함.)

- app.use(express.static("public"));

  "public" 디렉토리에서 HTML, CSS, JS 파일과 같은 정적파일을 제공.

- app.set("view engine", "ejs");

  뷰 엔진을 ejs로 설정.(서버 측에서 HTML 페이지를 생성하는데 사용되는 템플릿 엔진)

body-parser

- Node.js의 Express 애플리케이션에서 요청(Request) 본문(Body)을 쉽게 처리할 수 있도록 도와주는 미들웨어입니다.
- 클라이언트에서 서버로 데이터를 전송할 때, 그 데이터를 읽고 사용할 수 있도록 파싱해주는 역할.

  npm install body-parser

body-parser

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const port = 3000;

// body-parser 설정
app.use(bodyParser.json()); // JSON 데이터 파싱
app.use(bodyParser.urlencoded({ extended: true })); // 폼 데이터 파싱

// 루트 경로('/')에 대한 GET 요청을 처리하는 라우트
app.get('/', (req, res) => {
    res.send(`
        <h1>POST 요청 테스트</h1>
        <form action="/submit" method="POST">
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" required>
            <br><br>
            <label for="age">Age:</label>
            <input type="number" id="age" name="age" required>
            <br><br>
            <button type="submit">Submit</button>
        </form>
    `);
});

// /submit 경로에 대한 POST 요청을 처리하는 라우트
app.post('/submit', (req, res) => {
    const { name, age } = req.body; // 요청 본문에서 name과 age를 추출
    res.send(`<h2>Received Data:</h2><p>Name: ${name}</p><p>Age: ${age}</p>`);
});

app.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
});
```

Welcome to the body-parser example!

Try sending a POST request to /submit

배열에 입력값 넣기

- app.js에 코드를 작성합니다.

```
let todos = [];
app.get("/", (req,res) =>{
    res.render("index", {todos : todos})
});
```

- 입력받은 할일들을 담아둘 배열 하나 선언해준 뒤 index.ejs에 todos(선언한 배열)라는 객체를 전달.

```
<form action="/addTask" method="post">
  <input type="text" name="add" placeholder="할일을 입력하세요">
  <button>할 일 추가</button>
</form>
```

```
app.post("/addTask", (req, res) => {
  todos.push(req.body.add);
  res.redirect("/");
});
```

- "/addTask" 경로에 대한 POST요청을 처리하는 핸들러 함수를 정의 후, req.body.add -> input에 입력한 값. 여기서 add는 input태그의 name에 해당하는 값.

배열 출력

- index.ejs에 입력해주세요

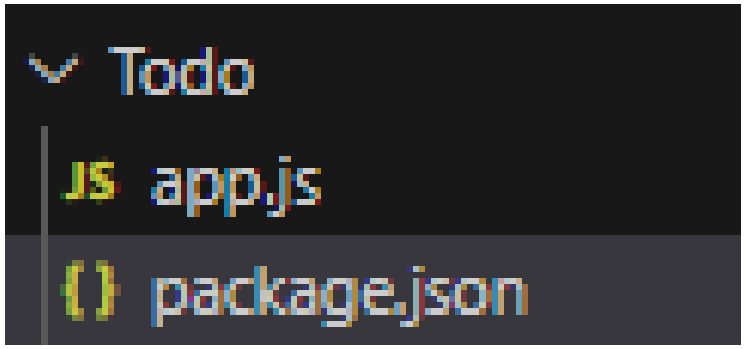
```
<ul id="todo-ul">
    <% for (let i = 0; i < todos.length; i++) { %>
        <li>
            <%= todos[i] %>
            <form action="/deleteTask" method="POST" style="display:inline;">
                <input type="hidden" name="del" value="<%= i %>">
                <button type="submit">삭제</button>
            </form>
        </li>
    <% } %>
</ul>
```

- 리스트 형식으로 입력받은 값을 화면에 출력할 것이다. Ejs에서 <% %>로 자바스크립트 구문을 추가함.

Node.js, MongoDB로 TODO 구체적으로 만들기

1. Todo 폴더를 생성하고, 메인 파일인 app.js를 생성해준다.

: Terminal 에서 Todo 폴더 경로로 이동한 후 npm init을 해주면 package.json 파일이 자동으로 생성됨.




```
{
  "name": "todo",
  "version": "1.0.0",
  "description": "todo list",
  "main": "app.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "youngpyo",
  "license": "MIT"
}
```

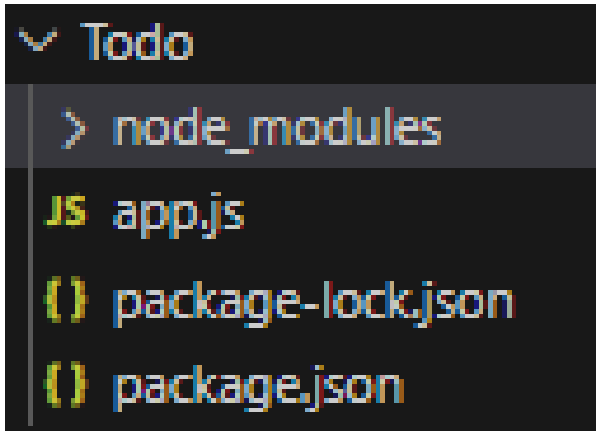
"MIT 라이선스"는 소프트웨어 라이선스 중 하나로, 자유롭고 개방적인 라이선스로 잘 알려져 있음.

Node.js, MongoDB로 TODO 구체적으로 만들기

2. npm install로 dependency 추가

Terminal에서 npm install express body-parse ejs mongoose path moment-timezone

(반드시 Todo 폴더 안에서 해야함.)




```
{
  "name": "todo",
  "version": "1.0.0",
  "description": "todo list",
  "main": "app.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "youngpyo",
  "license": "MIT",
  "dependencies": {
    "body-parse": "^0.1.0",
    "ejs": "^3.1.10",
    "express": "^4.19.2",
    "moment-timezone": "^0.5.45",
    "mongoose": "^8.5.3",
    "path": "^0.12.7"
  }
}
```

Node.js, MongoDB로 TODO 구체적으로 만들기

3. app.js에 필요한 모듈들을 import한다.

```
// Import Modules
const express = require("express");
const mongoose = require("mongoose");
const path = require("path");
const bodyParser = require("body-parser");
```

4. 변수 app에 express 서버를 만들어 줍니다.

```
// Make Express Servers
const app = express(); // 3000: Todo
```

Node.js, MongoDB로 TODO 구체적으로 만들기

5, app(서버) 세팅 및 디렉토리 생성

```
// Server Setting - View, Static Files, Body Parser
app.set("view engine", "ejs"); //express 서버에서 jsp처럼 쓰는 ejs파일을 뷰 엔진으로 설정
app.set("views", path.join(__dirname, 'views'));

app.use("/public", express.static(__dirname + '/public'));

app.use(bodyParser.urlencoded({extended: false}));
app.use(bodyParser.json());
```

뷰 엔진 설정

Express에서 EJS(Embedded JavaScript)를 뷰 엔진으로 설정.

‘views’ 디렉토리 경로를 설정

Node.js, MongoDB로 TODO 구체적으로 만들기

5, app(서버) 세팅 및 디렉토리 생성

```
// Server Setting - View, Static Files, Body Parser
app.set("view engine", "ejs"); //express 서버에서 jsp처럼 쓰는 ejs파일을 뷰 엔진으로 설정
app.set("views", path.join(__dirname, 'views'));

app.use("/public", express.static(__dirname + '/public'));

app.use(bodyParser.urlencoded({extended: false}));
app.use(bodyParser.json());
```

정적 파일 제공

Express에서 정적 파일(이미지, CSS, JavaScript 파일 등)을 제공하기 위한 설정.

express.static은 지정된 디렉토리의 파일을 직접 제공할 수 있게 함.

Node.js, MongoDB로 TODO 구체적으로 만들기

5, app(서버) 세팅 및 디렉토리 생성

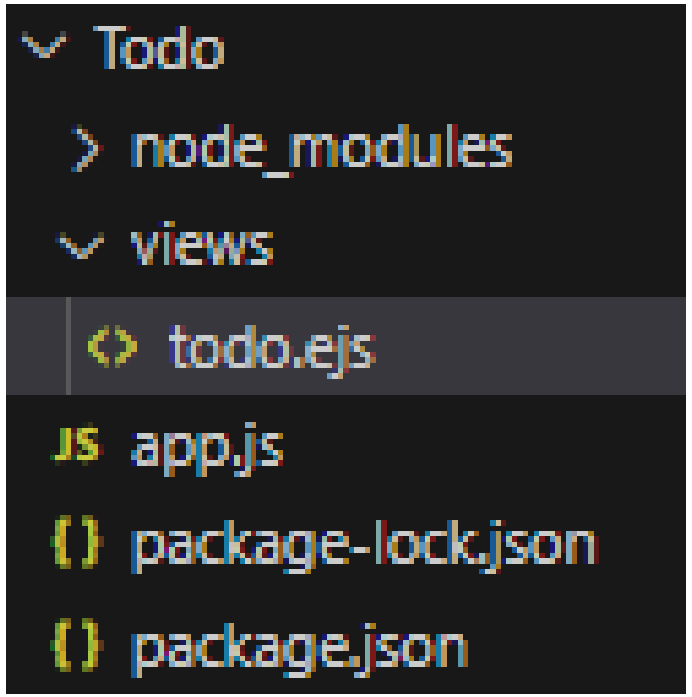
```
// Server Setting - View, Static Files, Body Parser
app.set("view engine", "ejs"); //express 서버에서 jsp처럼 쓰는 ejs파일을 뷰 엔진으로 설정
app.set("views", path.join(__dirname, 'views'));

app.use("/public", express.static(__dirname + '/public'));

app.use(bodyParser.urlencoded({extended: false}));
app.use(bodyParser.json());
```

바디 파서 설정

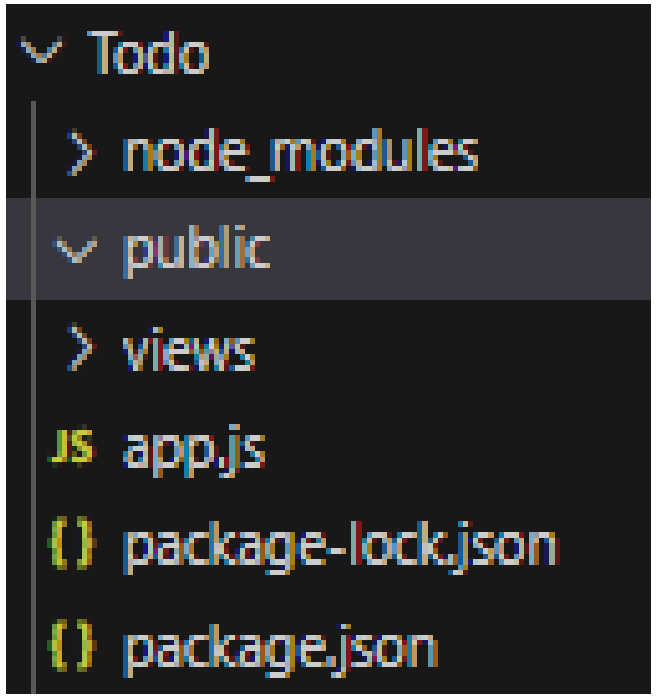
body-parser 미들웨어를 사용하여 URL-encoded 데이터를 처리함. 주로 HTML 폼 데이터를 파싱할 때 사용됨. extended: false는 쿼리스트링 모듈을 사용하여 파싱된 데이터를 단순 객체 형태로 만들도록 설정하는 것. app.use(bodyParser.json());로 JSON 형식의 데이터를 처리

Node.js, MongoDB로 TODO 구체적으로 만들기

ejs engine으로 설정해주고, ejs 파일을 넣어둘 views 폴더를 views로 설정해줄건데, 이때 views 폴더와 todo.ejs 파일도 같이 생성해주자.



정적 파일(fonts, images, css)를 저장할 public 폴더도 생성해주고 app.use로 정적 파일 경로를 설정해주자.

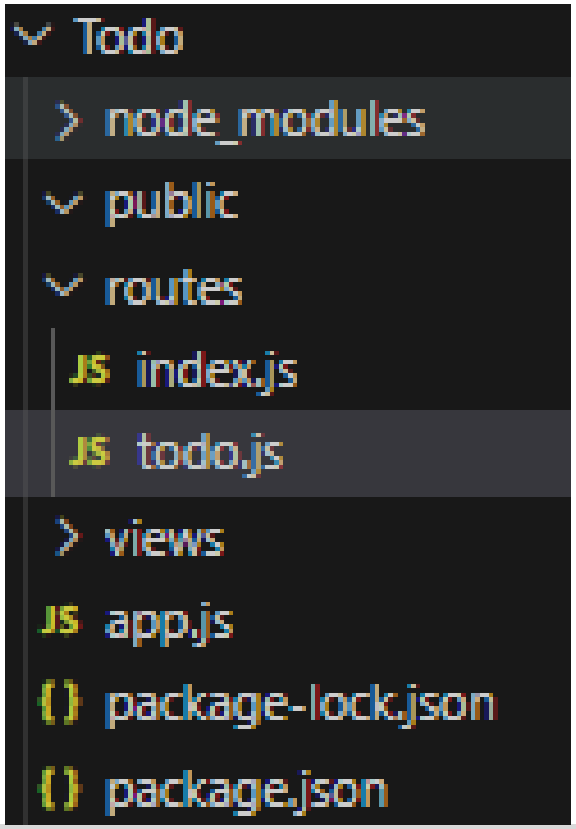


Node.js, MongoDB로 TODO 구체적으로 만들기

6, 라우터 설정

url에 따라 페이지를 보여주는 내비게이션 / 클라이언트로부터 요청을 받고, 이를 처리한 후 적절한 응답을 반환하는 역할

app.js에서 정의할 수도 있지만 기능별로 모듈화 해두고 app.js에 불러와서 쓰는 형태가 확장성이 좋기 때문에 routes 폴더를 생성해서 index.js 파일을 만들어주고 기능별 라우터를 묶어줌.



app.js

```
// Router Setting
const router = require("./routes/index");
app.use(router);

// 라우터가 유효한지 확인하기 위해 로그 출력
console.log(router);  // 유효한 라우터 객체가 출력되어야 함
```

Node.js, MongoDB로 TODO 구체적으로 만들기

7. MongoDB 연결, 서버오픈

app.js

```
// Connect to DB
/*
    connect error 발생 시
    mongodb://[아이디]:[비밀번호]@localhost:27017/[db이름]?authSource=[아이디] 로 변경
    ex) mongodb://admin:1234@localhost:27017/node?authSource=admin
*/
async function startServer() {
    try {
        await mongoose.connect("mongodb://localhost:27017/node", {
            useNewUrlParser: true,
            useUnifiedTopology: true,
        });
        console.log("mongoDB Connected!");

        app.listen(3000, () => {
            console.log("Server listening on port 3000!");
        });
    } catch (err) {
        console.error("mongoDB Connection Error!", err);
    }
}

startServer();
```

routes/index.js

```
// routes/index.js
const express = require("express");
const router = express.Router();

// 빈 라우터 객체를 내보내기
module.exports = router;
```

Node.js, MongoDB로 TODO 구체적으로 만들기

8. routes/todo.js 파일 작성

```
//Import modules
const express = require("express");
const app = express();
const router = express.Router();

// Controller 를 불러와서 exports 메소드 사용
const controller = require("../controllers/todo");

// Main
router.get('/', controller.get); // http://localhost:3000/todo/

// Write
router.post('/write', controller.write); // http://localhost:3000/todo/write

// Edit
router.get("/edit/:id", controller.edit);

// Update
router.post("/update/:id", controller.update);

// Remove
router.get("/remove/:id", controller.remove);
```

Node.js, MongoDB로 TODO 구체적으로 만들기

9. Controllers 폴더 생성, todo.js 파일 생성

Thank you.

웹 프로그래밍
ryp1662@gmail.com