Unit 17

첫 웹 애플리케이션

UT-NodeJS / 03.24.2023

ut-nodejs.github.io

Contents / 내용

07. 첫 번째 웹 애플리케이션 만들기

```
. <-- 터미널에서 본 루트 폴더의 트리구조 (p. 120)
|____main.js
|____router.js
|____public
| |____css
| | |____bootstrap.css
| | |____style.css
| |____js
| | |____functions.js <-- 포함하지 않아도 된다
| |____img
| | |____ <-- 모든 페이지에서 하나의 이미지 포함 필수
|____views
| |____index.html
| |____courses.html
| |____contact.html
| |____thanks.html
| |____error.html
|____content-types.js
|____utils.js
|____package.json <-- npm init으로 만드십시오
|____package-lock.json <-- npm install로 만드십시오
```

07

캡스톤 프로젝트 1

첫 번째 웹 애플리케이션 만들기

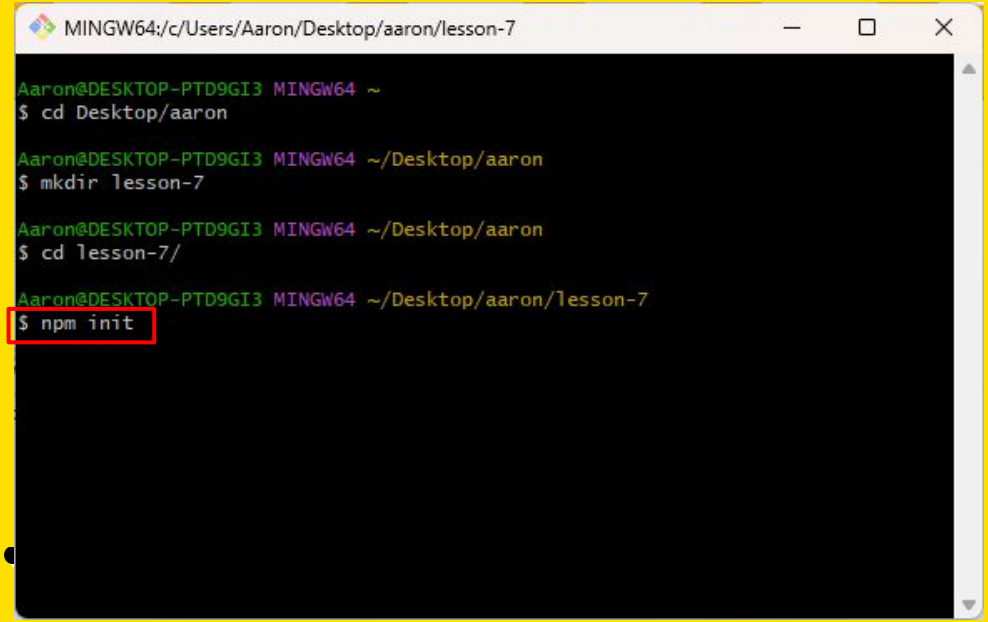
p. 117-129

첫 번째 웹 애플리케이션 만들기

이 애플리케이션을 만들기 위해 다음과 같은 단계를 거친다.

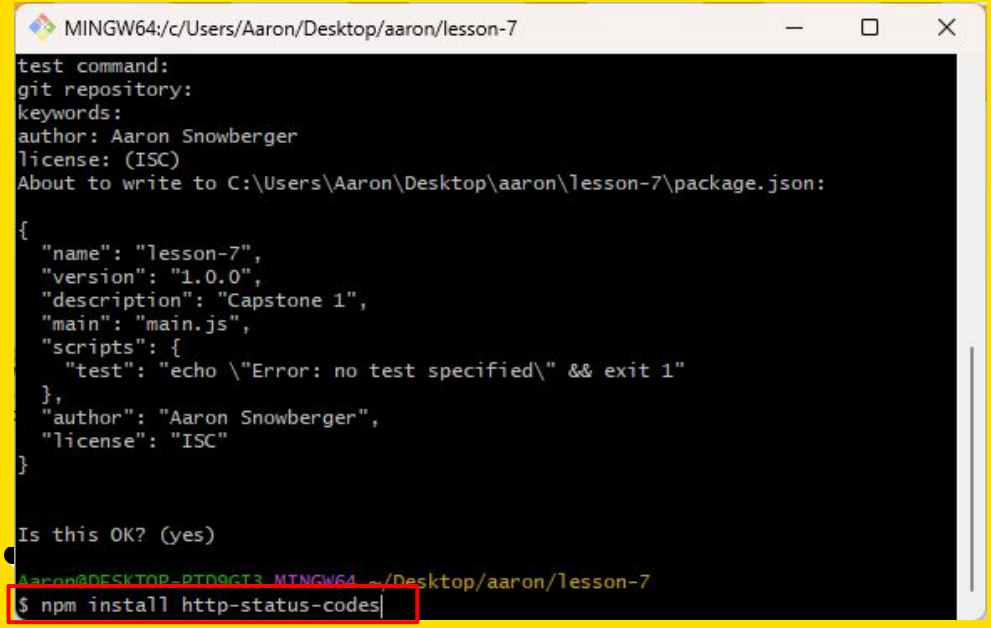
1. 애플리케이션 package.json의 초기화 (`npm init`)
2. 프로젝트 디렉터리 구조 셋업 (p. 120)
3. main.js에 애플리케이션 로직 만들기
4. 3 개의 view를 생성한다. 각 view는 다음과 같은 독립적인 기능을 제공하는 클릭 가능한 이미지를 갖고 있다.
    - Index (home) (`index.html`)
    - Courses (`courses.html`)
    - Contact (`contact.html`)
    - Thanks (`thanks.html`)
    - Error (`error.html`)
5. 에셋의 추가
6. 애플리케이션 라우트 구성
7. 애플리케이션 에러 처리
8. 애플리케이션 실행

① 애플리케이션의 초기화




```
cd ~/Desktop
cd user-folder
mkdir lesson-7
cd lesson-7
npm init
```

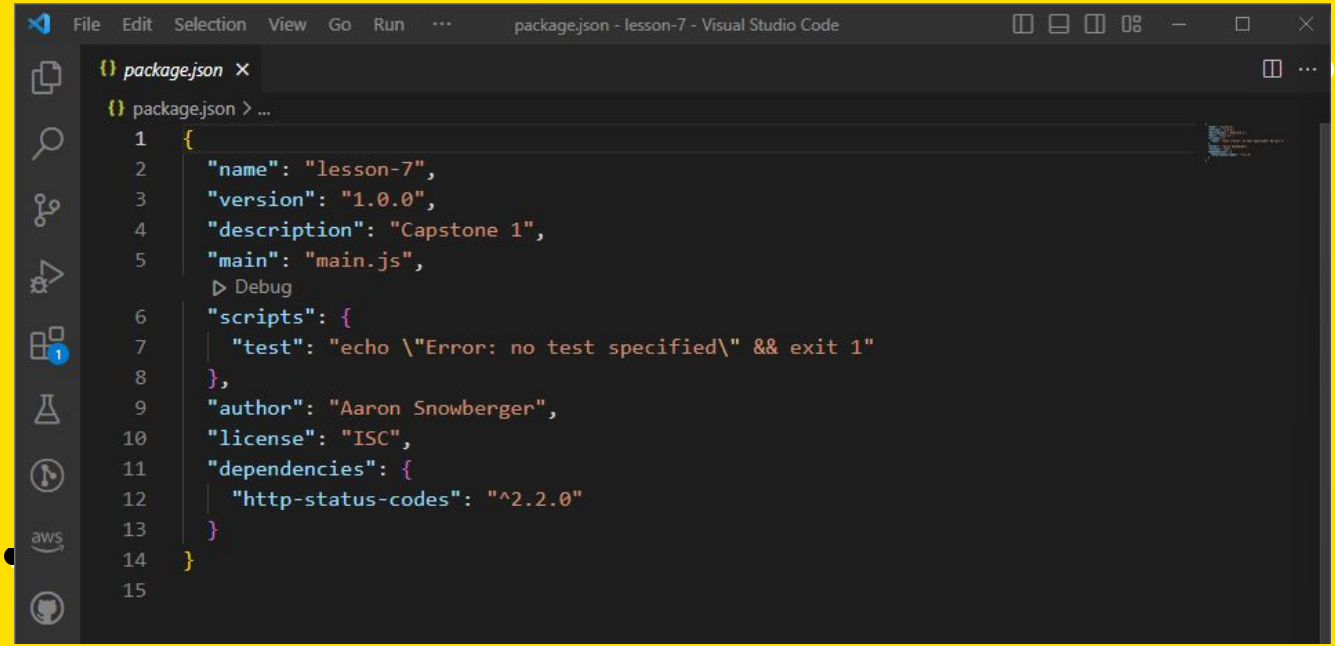
① 애플리케이션의 초기화




```
npm install
http-status-codes

code .
```

① 애플리케이션의 초기화



② 애플리케이션 디렉터리 구조의 이해

```
. <-- 터미널에서 본 루트 폴더의 트리구조 (p. 120)
|____main.js
|____router.js
|____public
|  |____css
|  |  |____bootstrap.css
|  |  |____style.css
|  |____js
|  |  |____functions.js
|  |____img
|  |  |____
|____views
|  |____index.html
|  |____courses.html
|  |____contact.html
|  |____thanks.html
|  |____error.html
|____content-types.js
|____utils.js
|____package.json
|____package-lock.json
```

프로젝트 디렉터리의 최상위(루트)에는 main.js, package.json 그리고 router.js 파일이 있다. HTML 콘텐츠들은 .html 확장자로 존재하며 views 폴더에 위치한다.

애플리케이션 서버는 view 폴더에 있는 HTML 파일을 갖고 응답할 것이다. 이 파일들을 지원하는 역할을 하는 에셋은 public 폴더에 위치하게 된다.

먼저 bootstrap.css 파일을 다운로드해 public 폴더 내 css 폴더에 저장한다. 또 style.css 파일도 생성해 이 프로젝트에서 원하는 스타일이 생겼을 때 대응할 수 있도록 한다.

③ router.js 및 main.js 파일의 생성

require 모듈이 있는 main.js

이제 사이트에 메인 애플리케이션 로직을 사이트에 추가해야 하며 이를 통해 3000번 포트로 서비스를 보여주게 된다. 라우트 관리도 별도의 파일에서 할 것이다. 따라서 fs 모듈과 함께 파일을 요청해야 하며 이를 통해 정적 파일들을 서비스할 수 있게 된다.

Listing 7.3 require module이 있는 main.js

```
const port = 3000,
  http = require("http"),
  httpStatus = require("http-status-codes"),
  router = require("./router"),
  contentTypes = require("./content-types"),
  utils = require("./utils");
```

필요 모듈의 임포트

③ router.js 및 main.js 파일의 생성

content-types.js

이 애플리케이션은 로컬 모듈들을 생성하기 전까지는 동작하지 않을 것이다. 로컬 모듈은 Listing 7 .4에 있는 코드를 이용한 content-types.js 파일 생성으로 제작을 시작할 것이다. 이 파일에서는 응답에서 사용될 header 값들과 파일 타입을 매핑하는 객체를 익스포트하고 있다. 나중에 contentTypes.html을 사용해 main.js 내에 있는 HTML 콘텐츠 타입에 액세스할 것이다.

Listing 7.4 content-types.js에서의 객체 매핑

```
module.exports = {                    ← 콘텐츠 타입 매핑 객체의 익스포트
  html: {
    "Content-Type": "text/html"
  },
  text: {
    "Content-Type": "text/plain"
  },
  js: {
    "Content-Type": "text/js"
  },
  jpg: {
    "Content-Type": "image/jpg"
  },
  png: {
    "Content-Type": "image/png"
  },
  css: {
    "Content-Type": "text/css"
  }
};
```

③ router.js 및 main.js 파일의 생성

utils.js

다음으로 새로운 utils 모듈에 있는 파일 콘텐츠를 읽기 위해 사용할 함수들을 준비한다. utils에서 Listing 7.5의 코드를 추가했다. 이 모듈에서 getFile 함수를 포함하는 객체를 익스포트했다. 이 함수는 제공된 위치의 파일을 찾는다. 찾는 파일이 없다면 바로 에러 페이지를출력한다.

Listing 7.5 util.js에서의 유틸리티 함수

```
const fs = require("fs"),
  httpStatus = require("http-status-codes"),
  contentTypes = require("./content-types");

module.exports = {
  getFile: (file, res) => {
    fs.readFile(`./${file}`, (error, data) => {
      if (error) {
        res.writeHead(httpStatus.INTERNAL_SERVER_ERROR, contentTypes.html);
        res.end("There was an error serving content!");
      }
      res.end(data);
    });
  }
};
```

getFile에서 사용할 모듈들의 임포트

파일을 읽고 응답을 돌려주기 위한 함수의 익스포트

③ router.js 및 main.js 파일의 생성

router.js

Router 모듈은 routes 객체를 포함하고 있다 routes 객체는 키-값 쌍으로 get 함수를 통한 GET 요청 그리고 post 함수를 통한 POST 요청을 매핑한다. Handle 함수는 main.js 에서 createServer의 콜백 함수로 간주된다. get과 post 함수는 콜백 함수와 URL을 취하며 routes 객체에서 서로 매핑한다. 라우트를 찾지 못한다면 utils 모듈 내 사용자 정의 함수인 getFile 함수를 사용해 에러 페이지를 표시할 것이다.

Listing 7.6 router.js에서의 라우트 처리

```
const httpStatus = require("http-status-codes"),
  contentTypes = require("./content-types"),
  utils = require("./utils");

const routes = {
  "GET": {},
  "POST": {}
};

exports.handle = (req, res) => {
  try {
    routes[req.method][req.url](req, res);
  } catch (e) {
    res.writeHead(httpStatus.OK, contentTypes.html);
    utils.getFile("views/error.html", res);
  }
};

exports.get = (url, action) => {
  routes["GET"][url] = action;
};

exports.post = (url, action) => {
  routes["POST"][url] = action;
};
```

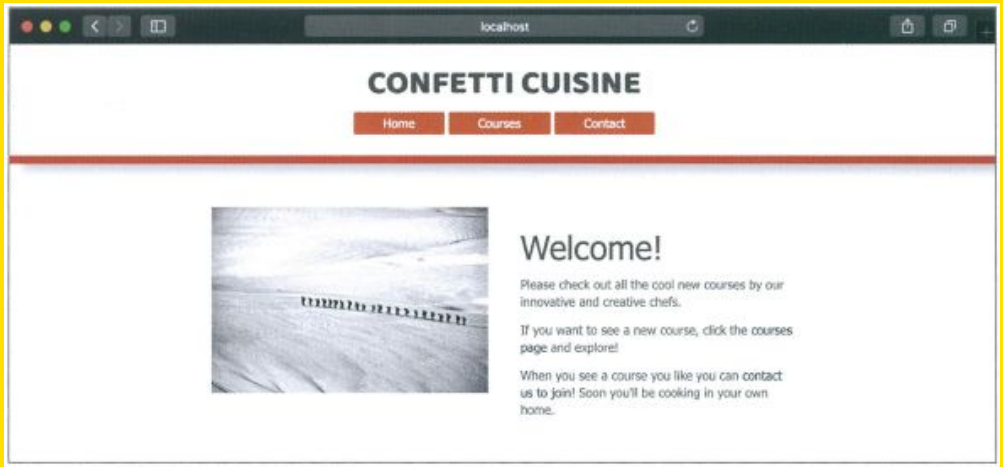
라우트 함수를 위한 routes 객체 생성

요청을 처리하기 위한 handle 함수를 생성한다.

라우트 함수를 매핑하기 위한 get과 post 함수를 생성한다.

④ 뷰 페이지 생성

뷰 페이지는 클라이언트 측에서 만들어지며 사용자 경험을 만들거나 망칠 수 있다. 이 애플리케이션의 복잡성을 줄이기 위해 각 페이지에 대해 비슷한 템플릿을 사용한다. 각 HTML 페이지의 맨 위에는 HTML 레이아웃, 헤드, 곧 생성될 사용자 정의 스타일시트에 대한 링크 및 페이지 탐색 기능이 있어야 한다. bootstrap.css를 사용하기 때문에 <link rel="stylesheet" href="/bootstrap.css">를 HTML 페이지 내 head 태그마다 추가해줘야 한다. 동일한 작업을 사용자 정의 스타일시트인 style.css에도 해준다.



④ 뷰 페이지 생성

필요한 페이지

- index.html
- courses.html
- contact.html
- thanks.html
- error.html

Listing 7.7 contacts.html 중 홈페이지 라우트에의 포스팅 폼

```
<form class="contact-form" action="/" method="post">
  <input type="email" name="email" required>
  <input class="button" type="submit" value="submit">
</form>
```

name 필드를 홈페이지로 제출하기 위한 폼 구성

⑤ 에셋 추가

이 애플리케이션을 위해 각 뷰 페이지에 사용될 사용자 정의 스타일을 생성했다 style.css 에 사이트에서 사용될 요소들의 색상 차원 위치 변경에 대한 정보들이 담기며, 이는 public/css 폴더에 bootstrap.css와 함께 위치한다.

이 파일들이 저장되면 뷰 페이지들은 시작 시 색상과 구성을 가지게 된다. 클라이언트 측 JavaScript를 쓰기로 결정했다면 .js 파일도 필요하며 이는 public/js 폴더에 저장될 것이다. 이 파일들은 <script> 태그에서 링크로 읽어들일 것이다. 마지막으로 이미지들은 public/images 폴더에 저장될 것이다. 이 이미지들은 HTML 뷰 페이지에서 읽어들일 것이다.

⑥ main.js의 router 모듈로 라우트 등록

이 퍼즐의 마지막 조각인 라우트는 아주 중요하다.

라우트들을 핸들링하기 위해 router.js 파일을 이미 만들었다.
하지만 이 라우트들은 등록돼야 한다.

라우트 등록을 정리하면 다음과 같다.

- 요청이 GET인가 POST인가
- URL 라우트
- 돌려줄파일 이름
- HTTP 상태 코드
- 돌려줄 파일 타입 (content type)

각 콜백 함수에서, 응답으로 보낼 콘텐츠 타입을 지정해야 하며 fs 모듈을 사용해 뷰 페이지와 에셋을 응답 콘텐츠로 읽어들여야 한다.

⑥ main.js의 router 모듈로 라우트 등록

Listing 7.8 main.js의 router 모듈로 라우트 등록

```
router.get("/", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.html);
  utils.getFile("views/index.html", res);
});

router.get("/courses.html", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.html);
  utils.getFile("views/courses.html", res);
});

router.get("/contact.html", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.html);
  utils.getFile("views/contact.html", res);
});

router.post("/", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.html);
  utils.getFile("views/thanks.html", res);
});
```

웹 페이지와 에셋을 위한 라우트 목록 추가

```
router.get("/graph.png", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.png);
  utils.getFile("public/images/graph.png", res);
});
router.get("/people.jpg", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.jpg);
  utils.getFile("public/images/people.jpg", res);
});
router.get("/product.jpg", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.jpg);
  utils.getFile("public/images/product.jpg", res);
});
```

```
router.get("/confetti_cuisine.css", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.css);
  utils.getFile("public/css/confetti_cuisine.css", res);
});
router.get("/bootstrap.css", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.css);
  utils.getFile("public/css/bootstrap.css", res);
});
router.get("/confetti_cuisine.js", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.js);
  utils.getFile("public/js/confetti_cuisine.js", res);
});
```

서버 시작

```
http.createServer(router.handle).listen(port);
console.log(`The server has started and is listening on port number: ${port}`);
```

[노트] POST 라우트에 주목하라 이는 contact. html 페이지에서 전달되는 폼을 처리한다 다른 HTML 페이지로 응답하는 대신 이 라우트는 “Thank you for supporting the product" HTML 페이지로 응답한다.

⑦ 애플리케이션 에러 처리

```js
// utils.js
"use strict";

/**
 * listing 7.5 (p. 123)
 */

const fs = require("fs"),
  httpStatus = require("http-status-codes"),
  contentTypes = require("./content-types"); // getFile에서 사용할 모듈들의 임포트

// 파일을 읽고 응답을 돌려주기 위한 함수의 익스포트
module.exports = {
  getFile: (file, res) => {
    fs.readFile(`./${file}`, (error, data) => {
      if (error) {
        res.writeHead(httpStatus.INTERNAL_SERVER_ERROR, contentTypes.html);
        res.end("There was an error serving content!");
      }
      res.end(data);
    });
  },
};

```

```js
// router.js
"use strict";

/**
 * listing 7.6 (p. 123-124)
 */

const httpStatus = require("http-status-codes"),
  contentTypes = require("./content-types"),
  utils = require("./utils");

// 라우트 함수를 위한 routes 객체 생성
const routes = {
  GET: {},
  POST: {},
};

// 요청을 처리하기 위한 handle 함수를 생성한다
exports.handle = (req, res) => {
  try {
    routes[req.method][req.url](req, res);
  } catch (e) {
    res.writeHead(httpStatus.OK, contentTypes.html);
    utils.getFile("views/error.html", res);
  }
};
```

8 실행

index.html

CONFETTI CUISINE

Home Courses Contact

<nav>

<header>

Welcome!

Please check out all the cool new courses by our innovative and creative chefs.

If you want to see a new course, click the courses page and explore!

When you see a course you like you can contact us to join! Soon you'll be cooking in your own home.

<footer>개인 정보</footer>

필요한 페이지 콘텐츠

- <header>
- <nav>
- <img>
- <p>
- <footer>

⑧ 실행 courses.html

CONFETTI CUISINE

Home Courses Contact <nav>

<header>

Learn to cook cutting edge food

Our Courses:

- Bread Making
- Pasta Milan
- Dark Chocolate Delight

Contact Us today!

<footer>개인 정보</footer>

필요한 페이지 콘텐츠

- <header>
- <nav>
- <img>
- <p>
- <footer>

⑧ 실행

contact.html

CONFETTI CUISINE

Home Courses Contact <nav>

<header>

Reach out to us!

Enter your email if you're interested to learn more:

submit

We'll get back to you soon!

<footer>개인 정보</footer>

필요한 페이지 콘텐츠

- <header>
- <nav>
- <img>
- <p>
- <footer>

⑧ 실행 thanks.html

CONFETTI CUISINE

Home Courses Contact <nav>

<header>

Thank you for submitting!

Go to home!

<footer>개인 정보</footer>

필요한 페이지 콘텐츠

- <header>
- <nav>
- <img>
- <p>
- <footer>

⑧ 실행

error.html

CONFETTI CUISINE

Home Courses Contact

<nav>

<header>

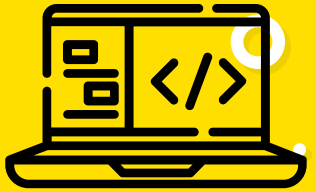
Oops! Something went wrong, or the page you are looking for is not available.

Go to home!

필요한 페이지 콘텐츠

- <header>
- <nav>
- <img>
- <p>
- <footer>

<footer>개인 정보</footer>



Coding 과제!

캡스톤 프로젝트

첫 번째 웹 애플리케이션 만들기

p. 117-129

과제 타임!

한번 해 보자!~