KWEB Study: Week 4

1. Before Study
2. Express.js Intro
3. Routing
4. 실습 및 과제

Express.js

KWEB 2학기 준회원 스터디

Designed by M.G. BAE

Before Study

- 오랜만이네요, 여러분! ~~중간고사.. 죽어줘..~~ 이제 다시 Back-end와 서로 좀 더 알아갈 시간입니다 ㅎ

- 3주차까지 Node.js를 비롯해 비동기와 Modern Javascript, HTTP 등 많은 것들을 배웠죠?
  - ✓ 비동기는 특강까지 했으니 이제 비동기를 비롯해 앞의 내용들 잘 아실 거라 생각합니다.

- 4주차의 목표는 Node.js용 프레임워크인 Express.js에 대해 다뤄보도록 하겠습니다.
  - ✓ 처음 보시죠? 오늘부터 새롭게 시작하는 애입니다.

- 일단 Node.js LTS 버전이 10으로 올라갔으니 업그레이드부터 하고 시작할게요!

생각해봅시다.

- 우리가 사실 웹 서버를 구현하려면 이것저것 신경 써야할 것이 많습니다.
- URL에 포함된 parameter 파싱, 경로 변수 추출, 정적 파일 (CSS, Javascript) 요청 시에 해당 파일 로딩, Error 처리 등 생각해야 할 부분이 많습니다!
- 이걸 본인 손으로 일일이 다 구현..? → 가능은 하겠죠. 근데 힘들고 입문자라면 하기도 어렵습니다.
  - ✓ 실제로 이렇게 개발하기도 합니다.
- 그럼 어떻게 하냐..? → Framework를 사용해보자.

Node.js Framework

- Node.js에서 http 모듈만 사용해서 웹 서버를 구성하면 많은 것들을 직접 만들어야 합니다. 그러면 시간과 노력이 많이 들겠죠? 그래서 보다 간편하게 해줄 것이 필요합니다! → Node.js 용 Framework

- Node.js 용 Framework의 종류는 몇 가지 존재하지만 보통 Express.js, koa.js, Hapi 이 3가지를 사용합니다. 이러한 Framework를 사용하면, Node.js로 웹 서버를 만들 때 해야할 일이 줄어듭니다.

- Framework는 개발 규모와 개발자 성향, 성능 등 여러 기준에 따라 필요한 것을 사용하면 됩니다.
  - ✓ 사용하지 않는 것도 답이 될 수 있죠! 그렇지만 우리는 Node.js를 처음 사용하니 한 번 써봅시다.
  - ✓ (쓰지 않고 하면 너무 하드하잖니까요..ㅎ)

- 이번 학기 KWEB 준스에서는 Express.js를 사용합니다!

Express.js

- Express.js는 Node.js Web Application Framework로 여러 기능들을 제공합니다.
  - ✓ Framework → 어떠한 목적을 달성하기 위해 복잡하게 얽혀 있는 문제를 해결하기 위한 구조이며, 소프트웨어 개발에 있어 하나의 뼈대 역할을 합니다. ex) 1학기에 배운 Bootstrap 기억나시죠..?
  - ✓ 개발에 있어서 Framework와 Library가 도움을 줄 수 있는데 이에 대해선 좀 더 뒤에 다루도록 합시다!

- Express.js는 다른 여러 모듈과의 조합을 통해 라우팅, 오류처리, 파싱, 파일 로딩 등 여러 기능을 직접 개발하지 않고 사용할 수 있는 기반을 제공 합니다.

- 간단하게! Express.js를 이용하면 Node.js로 간편하게 웹 서버를 구축할 수 있습니다.

- 더 궁금하시면 소개는! 공식 홈페이지에 더 자세히 나와있습니다.
  - ✓ URL: https://expressjs.com/
  - ✓ GitHub: https://github.com/expressjs/express

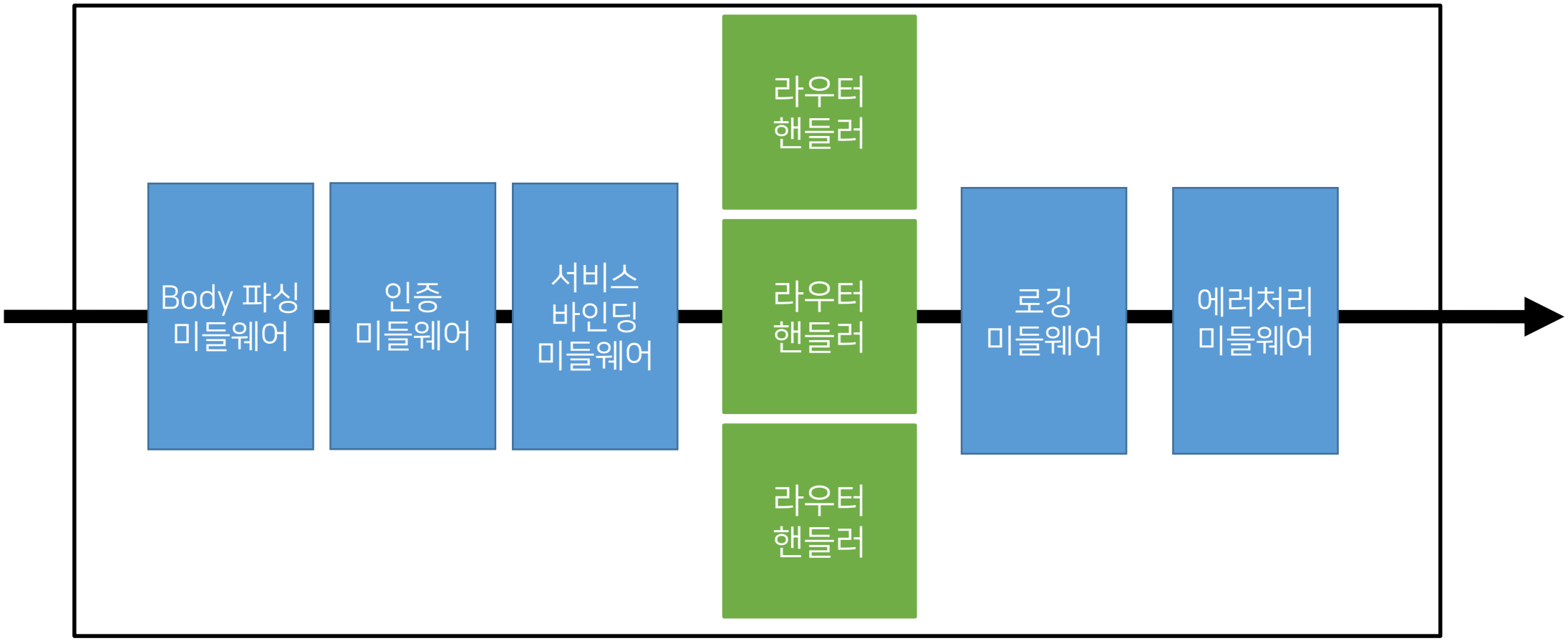
Why Express.js..?

- 솔직히 원래는 koa.js를 다루려고 했습니다. 훨씬 Modern하고 깔끔하며, 인터넷에서 찾을 수 있는 코드도 웬만하면 지금도 다 쓸 수 있는 코드들입니다.

- But, Express.js가 많은 것들을 기본으로 처리해주기 때문에 입문용으로 적합하기도 하고 레퍼런스가 많기도 해서 다시 Express.js를 고르게 되었습니다.
  - ✓ 제 개인 취향은 koa.js 입니다 ㅎ

- 다시 But, 예전부터 존재해왔고 방대한 예제들이 있으므로! 지금 쓰지 말아야할 코드들이 존재합니다.
  - ✓ 이 코드들은 잘 선별해서 필요한 것만 사용하도록 해야합니다.

Express.js 처리과정 예시



Middleware

- 처음 보는 개념이 있죠? → Middleware는 뭐야 대체……

- Middleware: 요청에 대한 응답 과정 중간에 껴서 어떠한 동작을 해주는 프로그램

- Express.js에서 Middleware는 요청을 처리하는 도중 다음과 같은 역할을 수행합니다.
  - ✓ 모든 코드를 실행 (ex. 로깅)
  - ✓ 요청 및 응답 오브젝트에 대한 변경을 실행 (ex. 파싱)
  - ✓ 요청-응답 주기를 종료 (ex, res.end())
  - ✓ 스택 내의 그 다음 Middleware를 호출 (ex. next())

- 각각의 Middleware는 next() 메소드를 호출하여 그 다음 미들웨어가 처리할 수 있도록 순서를 넘길 수 있습니다. → Middleware 여러 개를 사용할 수 있다는 뜻이죠?

Middleware 형태

- Middleware는 3개의 인자를 가지는 함수 형태입니다.

```
function (req, res, next) {
    …
    next()
}
```

- Express.js는 Middleware인지, Error 처리인지, Controller 인지를 parameter 개수로 구분 합니다.
  - ✓ Controller는 MVC 같은 소프트웨어 디자인 패턴을 소개할 때 더 언급하겠습니다.

- 대부분의 Middleware는 자신의 Logic을 실행 후 next() 메소드를 통해 실행 권한을 넘깁니다.
  - ✓ 즉, next() 가 나오는 것이 다음 Middleware를 시작해라는 trigger 입니다.

Express.js 웹 서버 예시

- Node.js 웹 서버 Example 보다 더 간단하죠?
  - ✓ Express도 모듈 형태로 제공되니 yarn add express / npm install express –save 명령어로 사용가능!

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

Express Server Object

- 앞의 코드에서 const app = express(); 를 살펴봅시다.

- 여기서 app 객체는 express() 메소드 호출로 만들어지는 익스프레스 서버 객체입니다!
  - ✓ 그냥 간단하게 서버를 생성할 수 있는 객체입니다.

- Method in Express server object
  - ✓ set(name, value) → 서버 설정을 위한 속성을 지정합니다. set() 메소드로 지정한 속성은 get() 메소드로 꺼내어 확인할 수 있습니다.
  - ✓ get(name) → 서버 설정을 위해 지정한 속성을 꺼내 옵니다.
  - ✓ use([path,] function [,function…]) → 미들웨어 함수를 사용합니다.
  - ✓ get([path,] function) → 특정 패스로 요청된 정보를 처리합니다.

res & req 객체

- Express에서 사용하는 요청 객체와 응답 객체는 http 모듈에서 사용하는 객체들과 같지만 아래 몇가지 메소드가 추가되어 있습니다.
  - ✓ send([body]) → 클라이언트에 응답 데이터를 보냅니다. (HTML 문서, Buffer 객체, JSON, etc.)
  - ✓ status(code) → HTTP 상태 코드를 반환합니다. 상태 코드는 end()나 send()같은 전송 메소드를 추가로 호출해야 전송할 수 있습니다.
  - ✓ sendStatus(statusCode) → HTTP 상태 코드를 반환합니다. 상태 코드는 상태 메시지와 함께 전송됩니다.
  - ✓ rediret([status,] path) → 웹 페이지 경로를 강제로 이동시킵니다.
  - ✓ render(view [, locals][, callback]) → 뷰 엔진을 사용해 문서를 만든 후 전송합니다.

- 요청 객체에 추가된 헤더와 파라미터도 있습니다!
  - ✓ req.query: GET 방식 전송 요청 파라미터 확인
  - ✓ req.body: POST 방식 전송 요청 파라미터 확인
  - ✓ req.header(name): 헤더를 확인합니다.

Error 처리

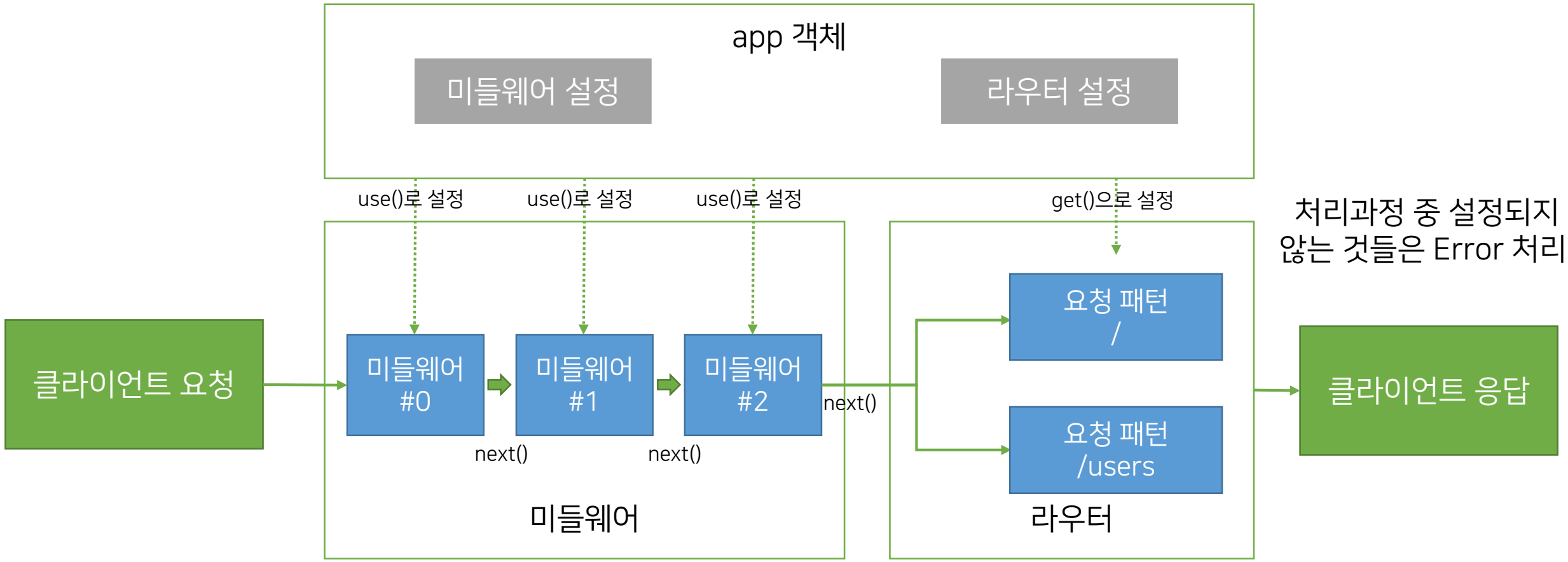
- Express의 Error 처리 함수는 4개의 인자를 가지며 자신보다 앞선 미들웨어나 컨트롤러에서 예외처리 되지 않는 예외가 존재하면 호출 됩니다.

```
function (err, req, res, next) {
    …
    next()
}
```

- 이 함수는 Error의 내용을 확인하고 응답을 수정하거나 next를 실행시켜 실행권한을 넘깁니다.

- app.use((req,res,next) => { next(createError(404)); }); 이런 식으로 Error가 발생하면 위와 같은 형태의 함수가 Error를 처리하게 됩니다.
  - ✓ 이 코드 위에서 처리하지 못한 것들이 전부 Error가 되겠죠?

Express.js Middleware 처리 예시



Routing

- 어 근데 그림에 못보던게 있죠? → Router는 또 뭐야 대체..

- Routing: URI (or path)와 특정 HTTP request method (GET, POST, etc.) 같은 특정한 endpoint에 대한 Client의 요청에 대해 Application이 어떻게 응답할지 설정하는 것
  - ✓ 컴퓨터 네트워크에서 Routing과 Router는 네트워크 단위에서 설명한 것이고, 우리는 웹 서버에서 말하는 겁니다.
  - ✓ Router → 각 Router에는 하나 이상의 handler 함수가 있을 수 있으며, 정의해둔 것과 일치 할 때 실행됩니다.

- 우리는 개발 시에 서버에게 어떤 메소드, 어떤 경로일때 어떤 로직을 실행할지 알려주어야 합니다.
  → 네, 구현을 다 해줘야 한다고요. 네.

- 다시 말하면, 특정 http 요청에 응답할 내용들을 미리 다 만들어 놓아야한다는 말이죠!
  - ✓ 각각의 경우에 무엇을 할지는 개발자가 직접 http 프로토콜의 헤더를 참고해 정해주어야 합니다.
  - ✓ HTTP Request Method에 대해서는 다음주에 Restful API에 대해 다룰 때 자세히 살펴봅시다.

Routing in Express.js

- Express.js에서 라우터는 Express 가 켜지면서 정의된 URL path 맵핑에 따라서 적절한 기능을 순서대로 실행 합니다. (라우터에 등록되는 경로는 정규식을 포함 할 수 있습니다.)

- 실행되는 기능에는 미들웨어(parser, Error 처리, etc.), 컨트롤러(핸들러를 대체로 컨트롤러라함!)가 포함 됩니다. → Controller에 대해선 나중에 다뤄봅시다.
  - ✓ app.use('/path/A', middlewareA); → /path/A 에 대해서 middlewareA를 실행
  - ✓ app.get('/board', boardController); → /board 에 대해서 boardController를 실행
  - ✓ app.use(errorControl); → 전체 path에 대해서 에러처리 미들웨어를 사용

- 라우터에 등록된 데이터는 Express bootstrap 과정에서 정의된 순서대로 실행됩니다.

Review! Express.js Server

- 제가 3주차에서 Postman 사용해볼 때, 서버 예시 코드로 줬던 것입니다.
  - ~~✓ 전혀 새롭나요 여러분..? 진짜 그렇다면.. 흑흑.. ㅋㅋㅋㅋㅋㅋ~~

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

app.get('/', (req, res) => res.send('Hello World!'));
app.post('/', (req, res) => res.send('Got a POST request'));
app.put('/', (req, res) => res.send('Got a PUT request'));
app.delete('/', (req, res) => res.send('Got a DELETE request'));

app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```

- 이제 이게 Router 설정이 되어있었다는 것을 좀 아시겠죠?
  - ✓ 이 예제는 endpoint가 '/' 하나 이며, Method만 분리되어 있습니다.

실습

Express.js

실습

- 네, 오늘 벌써 이론이 끝났어요. 뭐가 없죠? ㅋㅋㅋㅋ 중간고사 후 첫 스터디라 이론은 짧게 준비했습니다.

- 이번주 실습은 하나 배웠으니 그거 써야죠? 3가지 주제로 Express.js를 사용해볼 예정입니다.
  - ✓ **실습 1 – Express.js 웹 서버**: Express.js를 이용한 간단한 웹 서버 실행해보기
  - ✓ **실습 2 – Middleware 사용해보기**: Express.js의 Middleware 사용해보기
  - ✓ **실습 3 – Express.js로 html 띄워보기**: ejs를 Express.js와 함께 사용해 html 파일 띄워보기

- 실습도 딱히 어려운 건 없습니다. ~~(제 생각만 그렇나요? ㅋㅋㅋ)~~ 가볍게 Express.js의 기초 사용법 정도에 대해 코드로 실행해볼 예정입니다.

- Express.js는 오늘 완전히 본다 보단 오늘부터 계속 실습을 진행하면서 조금씩 알아가시면 됩니다! 오늘은 Express.js와 함께 남은 시간동안 놀아봅시다~

실습 1 – Express.js 웹 서버

- 첫번째 실습을 간단하게 아까 본 Express.js 웹 서버 예제 코드를 실행시켜볼 것입니다.
  - ✓ 더불어, express 모듈을 사용하는 법도!

- 일단 yarn init 명령어로 프로젝트를 하나 생성해봅시다.
  - ✓ 이제 이 정도는 그냥 하셔야죠? ㅎ

- 그리고! yarn add express 명령어를 통해 express 모듈을 설치해봅시다.
  - ✓ Express.js는 언급한대로! Framework로 Node.js의 모듈 형태로 제공됩니다.

- (캡쳐본을 드릴까 하다가 이제 이 정도는 혼자 할 수 있으셔야 합니다. 모르시겠으면 찾아가며 해보세요!)

실습 1 – Express.js 웹 서버

- 기본 세팅이 완료되셨으면 거기에 아무 js 파일이나 만들고 아래 코드를 입력하고 yarn script를 통해 실행시켜봅시다.

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

- 잘 실행되시죠? console 창과 브라우저에 표시되는 내용들을 각각 확인해봅시다.

실습 2 – Middleware 사용해보기

- 짠! 가볍게 실습 1이 끝났습니다. 쉽죠? 기본적으로 Node.js 용 Framework이므로 Node.js와 사용방법이 유사합니다. + Javascript를 사용하기 때문이죠!

- 이번엔! Express의 대표적인 Middleware인 body-parser, Error 처리를 어떻게 적용하는지 봅시다.

- 잠깐, body-parser! → Client의 HTTP 요청 중 POST 요청의 body 데이터에 접근하기 위한 모듈
  - ✓ yarn add body-parser 명령어로 먼저 추가해야겠죠?

실습 2 – Middleware 사용해보기

- yarn 프로젝트로 파일을 생성하고 실행!
- Postman으로 각각의 endpoint에 대해 HTTP 요청을 해봅시다.
- Endpoint 목록
  - ✓ GET /
  - ✓ POST /
  - ✓ POST /body
- 결과가 어떻게 될까요?

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

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

app.get('/', (req, res) => { res.send(`KWEB week4`); });

app.post('/', (req, res) => {
    console.log(`${req.method} ${req.url}
        ${Object.keys(req.headers).map(k => `${k}: ${req.headers[k]}`).join('\n')}`);
    res.redirect('/');
});

app.post('/body', (req, res) => { res.send(req.body); });

app.listen(port, () => console.log(`KWEB week4 - Practice 2`));
```

실습 2 – Middleware 사용해보기

- 마지막으로 Error처리까지 해봅시다!
- 앞의 코드에서 빨간 부분을 추가해봅시다!
- 그러고 GET /body 에 접속하면 어떻게 될까요?
- 밑의 빨간 부분과 같은 방식으로 Error를 처리할 수 있습니다.
  - ✓ Status Code 500번 기억하시죠? ㅎ

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

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

app.get('/', (req, res) => { res.send(`KWEB week4`); });
app.post('/', (req, res) => {
    console.log(`${req.method} ${req.url}
        ${Object.keys(req.headers).map(k => `${k}: ${req.headers[k]}`).join('\n')}`);
    res.redirect('/');
});

app.get('/body', (req, res) => { throw new Error('TEST!'); });
app.post('/body', (req, res) => { res.send(req.body); });

app.use(function (err, req, res, next) {
    console.log(err);
    res.statusCode = 500;
    res.send('error');
});

app.listen(port, () => console.log(`KWEB week4 - Practice 2`));
```

실습 3 – Express.js로 html 띄워보기

- 놀랍게도 Express.js의 코드를 얼렁뚱땅 제법 살펴봤습니다.
  - ✓ 사실 Express.js 자체도 중요하지만 거기서 사용할 수 있는 Middleware에 대해 많이 알아보셔야 해요!

- Express.js는 뭔가 더 배운다기 보단 테크닉이기 때문에 오늘은 Toy Example들만 살펴보았습니다.

- 오늘의 마지막 실습으론 View Engine을 이용해 HTML 파일을 드디어! 띄워봅시다!

잠깐! View Engine

- View Engine: 뷰 템플릿을 사용해 결과 웹 문서를 자동으로 생성한 후 응답을 보내는 역할을 수행합니다.
  - ✓ 당연히 보내는 형식이 필요하겠죠? 그 형식의 바로 밑에서 설명하는 View Template입니다.

- View Template: 클라이언트에 응답을 보낼 때 사용하려고 미리 만들어 놓은 웹 문서의 원형

- View Engine의 사용으로 손쉽게 서버의 응답 결과를 View로 넘겨줄 수 있습니다.
  - ✓ 근데 사실 요새는 Front-end 서버를 분리해서 React.js나 Angular 같은 걸로 띄우므로 별 필요 없.. 쿨럭쿨럭
  - ✓ Back-end는 요새 API 서버로만 만드는 것도 개발 트렌드입니다.

- Node.js에는 ejs나 pug 같은 여러 개의 뷰 엔진이 존재합니다.
  - ✓ 이번 준스에서 사용하는 경우에는 HTML과 가장 똑같은 ejs만 사용할 예정입니다.

- 잘 와닿지 않죠? 지금 ejs 한 번 써보고 생각해봅시다!

실습 3 – Express.js로 html 띄워보기

- 별로 거창한 건 아니고 한 줄 추가하면 ejs를 쓸 수 있습니다. → app.set('view engine', 'ejs');
  - ✓ 물론! ejs를 당연히 설치했어야 겠죠? 외장 모듈입니다!

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

const path = require('path');
const cal = require('./cal');

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.get('/', (req, res) => res.render('index'));
app.get('/caculator', (req, res) => res.render('cal', { x: cal.add(1,2), y: cal.mod(19247,423), E: cal.getE() }));

app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```

실습 3 – Express.js로 html 띄워보기

- 앞 코드의 index와 cal은 ejs 파일입니다. → index.ejs / cal.ejs

- 아래 두 파일을 views 라는 폴더를 만들고 안에 만들어줍시다. → **왜 vies 폴더에 넣어줘야할까요..?**
  - ✓ index.ejs에는 아무거나 입력해보세요! 기본적으로 html 파일과 똑같이 쓰시면 됩니다.
  - ✓ cal.ejs에는 아무거나 쓰고 body 태그 내에 아래 태그를 쓰고 출력해봅시다.

```
<div>x의 값은 <%=x%>입니다.</div>
<div>y의 값은 <%=y%>입니다.</div>
<div>e의 값은 <%=E%>입니다.</div>
```

- 이런 방식으로 서버에서 앞단으로 서버에서 연산한 값을 렌더링할 수 있습니다! → res.render 메소드!

실습 3 – Express.js로 html 띄워보기

- 빨간 박스 부분이 서버사이드에서 ejs로 값을 넘겨주는 부분이고 ejs에서는 <%=x%>, <%-y%> 형태로 넘겨받을 수 있습니다. → <% … %> 내부에는 Javascript 사용할 수 있습니다!

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

const path = require('path');
const cal = require('./cal');

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.get('/', (req, res) => res.render('index'));
app.get('/caculator', (req, res) => res.render('cal', { x: cal.add(1,2), y: cal.mod(19247,423), E: cal.getE() }));

app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```

과제 (~ 11/16 23:59:59)

- 과제기한: 2018년 11월 16일 금요일 자정까지

- 4주차 과제는 Node.js로 Pure하게 구현된 Routing 코드를 express 모듈을 활용하는 것으로 바꾸는 것입니다!
  - ✓ 기한 내에는 충분히 하실 수 있을겁니다!

- 아래 URL에 코드가 있으니 여기서 가져가세요!
  - ✓ URL: https://github.com/baemingun/kweb_week4_hw.git

- 제출방법: 위 과제를 완료한 후, GitHub에 업로드하고 GitHub Repository 주소를 준회원 톡방에 올리시면 됩니다.

That’s all for today!