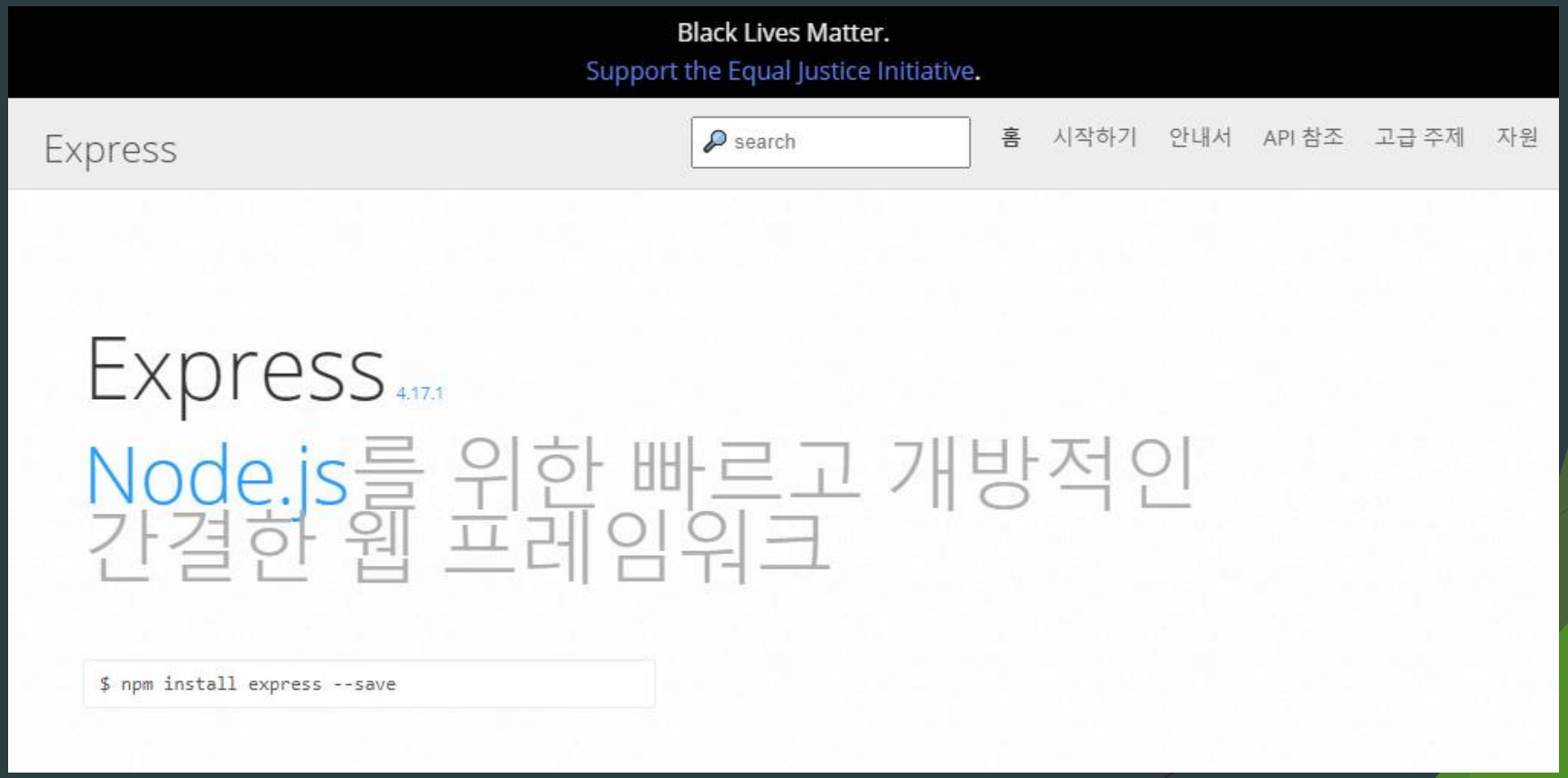
Node.js

express framework - 김근형 강사

express framework

- express framework : https://expressjs.com/ko/



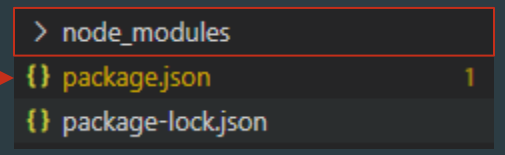
express framework

- express framework
  - Express.js, 또는 간단히 익스프레스(Express)는 Node.js를 위한 웹 프레임워크
  - 자유-오픈 소스 소프트웨어로 출시되었다.
  - 웹 애플리케이션, API 개발을 위해 설계되었다.
  - Node.js의 사실상의 표준 서버 프레임워크로 불리고 있음.
  - http를 통해 어렵게 구현해야 했던 웹 어플리케이션 환경을 쉽게 구축이 가능함

express framework

- express framework 환경설정

```
{
    "name": "ExpressBasic",
    "version": "0.0.0",
    "private": true,
    "dependencies": {
        "express": "*"
    }
}
```



  - package.json 을 위와 같이 구성하고 install 실행을 하게 되면 같은 폴더에 node_modules 라는 패키지가 존재해야 한다.

express 실행

- express 기본적인 환경

```
// 기본적인 express 선언부
var express = require( 'express' );
var app = express();

// http.Server 객체의 listen과 동일
var server = app.listen([port], function()
{
    …서버 시작 시 수행 로직…
});
```

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

var server = app.listen(2000, function(){
    console.log('포트 2000번으로 서버 실행');
});
```

express 실행

- express 간단한 웹 예제

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

app.get("/", function(req, res){
    res.send("ROOT");
});

app.get("/test1", function(req, res){
    res.send("TEST");
});

var server = app.listen(2000, function(){
    console.log('포트 2000번으로 서버 실행');
});
```

localhost:2000

ROOT

localhost:2000/test1

TEST

express 기능

- express 메소드

메소드	설명
express.Router ([option])	새 라우터 개체를 만든다. options 매개 변수는 라우터의 동작을 지정한다.
express.static (root, [option])	정적으로 참조할 파일/폴더의 위치를 제공 한다. root 인수는 고정 제공하는 루트 디렉토리를 지정한다.

express 기능

- application 프로퍼티 및 이벤트,메소드

프로퍼티	설명
app.locals	애플리케이션 내의 로컬 변수처럼 사용이 가능하다.
app.mountpath	서브앱이 마운트가 되었을 때 여러 개의 url 패스의 패턴을 포함한다.

이벤트	설명
app.on('mount', callback(parent))	서브 앱이 마운트가 되었을 경우 실행되는 이벤트

express 기능

- application 프로퍼티 및 이벤트,메소드

메소드	설명
app.all(path, callback [, callback ...])	app.METHOD () 메서드와 거의 동일. 모든 요청을 라우트 한다.
app.delete(path, callback [, callback ...])	지정된 콜백 함수를 사용하여 HTTP DELETE 요청을 지정된 경로로 라우팅한다.
app.disable(name)	name에 대한 설정을 false로 바꾼다. app.set([name], false) 와 동일
app.disabled(name)	name에 대한 설정이 사용 못하게 막혀있는지 확인한다.
app.enable(name)	name에 대한 설정을 true로 바꾼다. app.set([name], false) 와 동일
app.enabled(name)	name에 대한 설정이 사용하도록 되어있는지 확인한다.
app.engine (ext, callback)	파일 확장자(ext)를 해당 모듈에 매핑시킨다.
app.get(name)	app 안에 name 이름으로 되어있는 값을 리턴한다.
app.get(path, callback [, callback ...])	지정된 콜백 함수를 사용하여 HTTP GET 요청을 지정된 경로로 라우팅한다.
app.listen(path, [callback])	주어진 경로에서 연결을 수신함. http.Server.listen() 과 동일
app.listen([port[, host[, backlog]]][, callback])	지정된 호스트 및 포트에서 연결을 바인드하고 수신한다. 이 메소드는 Node의 http.Server.listen () 과 동일하다

express 기능

- application 프로퍼티 및 이벤트,메소드

메소드	설명
app.METHOD(path, callback [, callback ...])	지정된 콜백 함수를 사용하여 HTTP GET, PUT, POST 요청을 지정된 경로로 라우팅한다.
app.param([name], callback)	라우팅 파라미터 콜백 트리거를 추가하는 데 사용된다. 일반적으로 경로 매개 변수와 관련하여 요청된 데이터의 존재를 확인하는 데 사용된다.
app.path()	앱의 표준 경로인 문자열을 반환한다.
app.post(path, callback [, callback ...])	정된 콜백 함수를 사용하여 HTTP POST 요청을 지정된 경로로 라우팅한다
app.put(path, callback [, callback ...])	지정된 콜백 함수를 사용하여 HTTP PUT 요청을 지정된 경로로 라우팅한다.
app.render(view, [locals], callback)	콜백 함수를 통해 뷰의 렌더링된 HTML을 반환한다.
app.route(path)	싱글 라우터 객체를 반환한다.
app.set (name, value)	name에 value을 할당한다. 원하는 값을 저장할 수 있지만 특정 이름을 사용하여 서버의 동작을 구성할 수 있다.
app.use([path,] callback [, callback...])	지정된 경로에 지정된 미들웨어 함수를 마운트한다. 요청 된 경로의 기준이 일치 할 때 미들웨어 함수가 실행된다.

express 기능

- request 프로퍼티 및 이벤트,메소드

프로퍼티	설명
req.app	자신을 호출한 애플리케이션의 인스턴스값이 들어있다.
req.baseUrl	라우터 인스턴스가 마운트 된 URL 경로
req.body	본문에 제출 된 데이터의 키-값 쌍을 포함한다
req.cookies	쿠키값을 포함하고 있는 객체
req.fresh	캐싱 사양에 따른 상태 값을 갖고있다.
req.hostname	HTTP 헤더 에서 파생 된 호스트 이름을 포함한다.
req.ip	요청의 원격 IP 주소를 포함한다
req.method	요청의 HTTP 메소드에 해당하는 문자열을 포함한다
req.originalUrl	node.js의 req.url과 동일
req.params	명명 된 경로 파라미터에 매핑 된 속성을 포함하는 객체
req.path	url의 경로 부분을 포함한다.
req.protocol	프로토콜을 포함한다.
req.query	경로의 각 쿼리 문자열 매개 변수에 대한 속성을 포함하는 개체
req. route	현재 일치하는 경로, 문자열을 포함

express 기능

- request 프로퍼티 및 이벤트,메소드

메서드	설명
req.accepts (types)	HTTP 헤더 필드를 기반으로 지정된 콘텐츠 유형이 허용되는지 확인한다
req.acceptsCharsets (charset [, ...])	Accept-CharsetHTTP 헤더 필드 에 따라 지정된 문자 집합의 첫 번째 허용 된 문자 집합을 반환한다.
req.acceptsEncodings (encoding [, ...])	Accept-EncodingHTTP 헤더 필드를 기반으로 지정된 인코딩의 첫 번째 승인 된 인코딩을 반환한다.
req.acceptsLanguages (lang [, ...])	요청의 Accept-LanguageHTTP 헤더 필드 에 따라 지정된 언어의 첫 번째 허용 언어를 반환한다
req.get (field)	지정된 HTTP 요청 헤더 필드를 반환한다.
req.is(type)	"Content-Type"HTTP 헤더 필드가 type매개 변수에 지정된 MIME 유형과 일치하는 경우 일치하는 컨텐츠 유형을 리턴한다

express 기능

- response 프로퍼티 및 이벤트,메소드

프로퍼티	설명
res.app	자신을 호출한 애플리케이션의 인스턴스값이 들어있다.
res.headersSent	앱이 응답을 위해 HTTP 헤더를 전송했는지 여부를 나타내는 부울 속성
res.locals	app.locals와 동일속성. 단 차이점은 요청 경로 이름, 인증 된 사용자, 사용자 설정 등과 같은 요청 수준 정보를 노출하는 데 유용하다.

express 기능

- response 프로퍼티 및 이벤트,메소드

메소드	설명
res.append(field [, value])	지정된 value를 HTTP response header에 세팅한다.
res.attachment([filename])	HTTP 응답헤더의 Content-Disposition 필드를 “attachment” 로 설정한다.
res.cookie(name, value [, options])	쿠키 이름과 값을 설정한다. value 값은 문자열이나 객체가 되어야 한다.
res.clearCookie(name [, options])	해당 name에 있는 쿠키 값을 지운다
res.download(path [, filename] [, fn])	path의 " 첨부 파일”을 전송한다
res.end([data] [, encoding])	응답 프로세스를 종료한다.
res.get(field)	필드에서 지정한 응답 헤더를 반환한다.
res.json([body])	JSON 타입으로 응답을 보낸다.
res.jsonp([body])	JSONP 지원과 함께 JSON 응답을 보낸다.
res.location (path)	응답 LocationHTTP 헤더를 지정된 path매개변수로 설정한다 .
res.redirect([status,] path)	지정된 URL에서 파생 된 URL로 리디렉션한다.
res.render(view [, locals] [, callback])	사용자가 전송한 데이터를 그대로 html 문자열로 사용자에게 전송해준다.
res.send([body])	응답을 전송한다.

express 기능

- response 프로퍼티 및 이벤트,메소드

메소드	설명
res.sendFile(path [, options] [, fn])	해당 경로에 존재하는 파일을 전송한다.
res.sendStatus (statusCode)	상태코드를 보낸다.
res.set(field [, value])	해당 response's HTTP header의 field의 값을 설정한다.
res.status(code)	응답 객체의 상태코드를 설정한다.

express 기능

- route 프로퍼티 및 이벤트,메소드

메소드	설명
router.all(path, [callback, ...] callback)	router.METHOD()와 동일
router.METHOD(path, [callback, ...] callback)	익스프레스의 라우팅 기능을 제공한다. (router.get(), router.post(), router.put()...)
router.param (name, callback)	라우팅 매개 변수에 콜백 트리거를 추가한다.
router.route (path)	단일 경로의 인스턴스를 반환한다.
router.use([path], [function, ...] function)	옵션 마운트 경로와 함께 지정된 미들웨어 기능을 사용할때 쓰는 기능

Routing

- Route 를 이용한 라우팅 예제

```
var express = require('express');
var app = express();
var router1 = require('./router/router1')(app)
var router2 = require('./router/router2')(app)

var server = app.listen(2000, function(){
    console.log('포트 2000번으로 서버 실행');
});
```

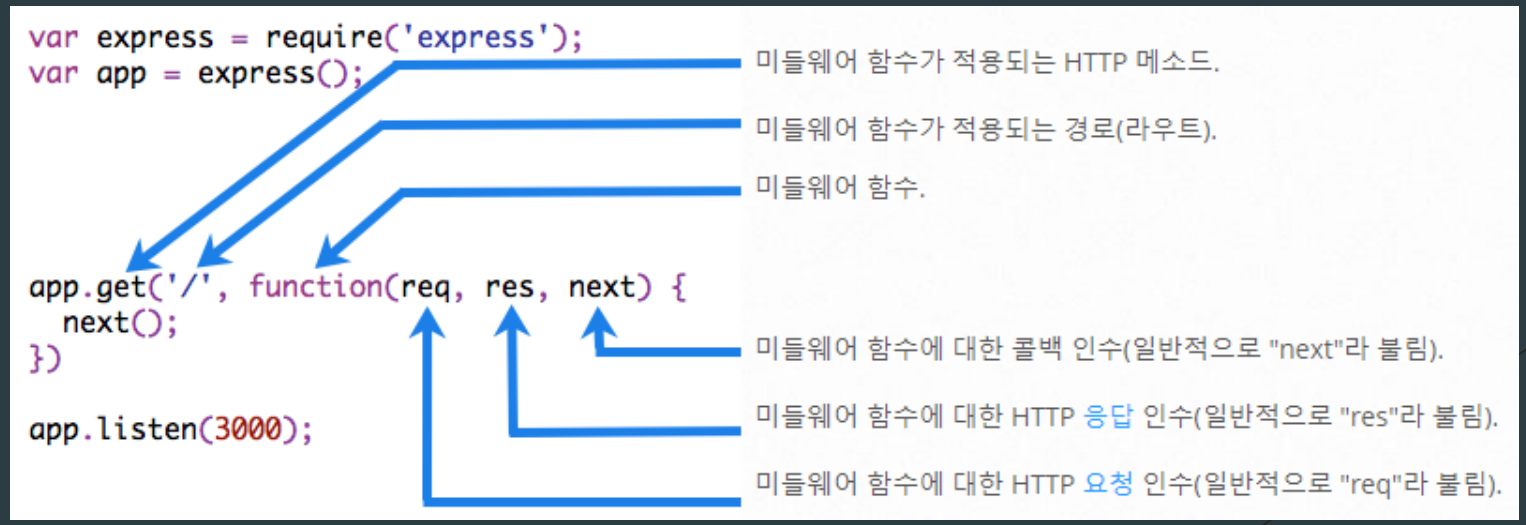
router 폴더

```
module.exports = function(app){
    app.get("/", function(req, res){
        res.send("ROOT");
    });

    app.get("/test1", function(req, res){
        res.send("TEST1");
    });
};
```

```
module.exports = function(app){
    app.get("/test2", function(req, res){
        res.send("TEST2");
    });

    app.get("/test3", function(req, res){
        res.send("TEST3");
    });
};
```

Middleware

- Middleware
  - 클라이언트에게 요청이 오고 그 요청을 보내기 위해 응답하려는 중간(미들)에 목적에 맞게 처리를 하는 함수들이다.
  - 예를 들어서 요청-응답 도중에 시간을 콘솔 창에 남기고 싶으면 미들웨어 함수를 중간에 넣어서 표시를 한 뒤에 계속해서 다음 미들웨어들을 처리할 수 있다.
  - 미들웨어의 사용 위치와 구성은 다음과 같다.



Middleware

- Middleware 역할
  - 모든 코드를 실행.
  - 요청 및 응답 오브젝트에 대한 변경을 실행.
  - 요청-응답 주기를 종료.
  - 스택 내의 그 다음 미들웨어를 호출.

Middleware

- Middleware 예제 - 1

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

var myLogger1 = function (req, res, next) {
    console.log('LOGGED1');
    next();
};

app.get("/test1", myLogger1, function(req, res){
    res.send("test1");
});

var server = app.listen(2000, function(){
    console.log('포트 2000번으로 서버 실행');
});
```

```
포트 2000번으로 서버 실행
LOGGED1
```

Middleware

- Middleware 예제 - 2
  - 두개 이상의 미들웨어를 실행 시킬 경우 함수를 늘여 쓰면 된다.

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

var myLogger1 = function (req, res, next) {
    console.log('LOGGED1');
    next();
};

var myLogger2 = function (req, res, next) {
    console.log('LOGGED2');
    next();
};

app.get("/test1", myLogger1, function(req, res){
    res.send("test1");
});

app.get("/test2", myLogger1, myLogger2, function(req, res){
    res.send("test2");
});

var server = app.listen(2000, function(){
    console.log('포트 2000번으로 서버 실행');
});
```

```
포트 2000번으로 서버 실행
LOGGED1
LOGGED2
```

localhost:2000/test2

Middleware

- Middleware 예제 - 3
  - express에서는 use()라는 메서드를 제공하는데 이를 통해 미들웨어를 전역으로 설정이 가능하다.

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

var myLogger1 = function (req, res, next) {
    console.log('LOGGED1');
    next();
};

var myLogger2 = function (req, res, next) {
    console.log('LOGGED2');
    next();
};

app.use(myLogger1);
app.use(myLogger2);

app.get("/test1", function(req, res){
    res.send("test1");
});

var server = app.listen(2000, function(){
    console.log('포트 2000번으로 서버 실행');
});
```

Middleware

- Middleware 예제 - 4
  - 라우터를 통한 미들웨어 설정

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

var myLogger1 = function (req, res, next) {
    console.log('LOGGED1');
    next();
};

app.use(myLogger1);

var router1 = require('./router/router3')(app)

var server = app.listen(2000, function(){
    console.log('포트 2000번으로 서버 실행');
});
```

```
var myLogger2 = function (req, res, next) {
    console.log('LOGGED2');
    next();
};

module.exports = function(app){
    app.get("/", function(req, res){
        res.send("ROOT");
    });

    app.get("/test1", function(req, res){
        res.send("TEST1");
    });

    app.get("/test2", myLogger2,function(req, res){
        res.send("TEST2");
    });
};
```

```
포트 2000번으로 서버 실행
LOGGED1
LOGGED1
LOGGED2
```