**[6장.익스프레스]**

**1. express 설치하기**

C:\materK1m\newmimac\test\_kes>**npm i -g express-generator**

C:\Users\Administrator\AppData\Roaming\npm\express -> C:\Users\Administrator\App

Data\Roaming\npm\node\_modules\express-generator\bin\express-cli.js

+ express-generator@4.16.0

updated 1 package in 0.708s

C:\materK1m\newmimac\test\_kes>

C:\materK1m\newmimac\test\_kes>

C:\materK1m\newmimac\test\_kes>**express learn-express --view=pug**

create : learn-express\

create : learn-express\public\

create : learn-express\public\javascripts\

create : learn-express\public\images\

create : learn-express\public\stylesheets\

create : learn-express\public\stylesheets\style.css

create : learn-express\routes\

create : learn-express\routes\index.js

create : learn-express\routes\users.js

create : learn-express\views\

create : learn-express\views\error.pug

create : learn-express\views\index.pug

create : learn-express\views\layout.pug

create : learn-express\app.js

create : learn-express\package.json

create : learn-express\bin\

create : learn-express\bin\www

change directory:

> cd learn-express

install dependencies:

> npm install

run the app:

> SET DEBUG=learn-express:\* & npm start

C:\materK1m\newmimac\test\_kes>**cd learn-express && npm i**

npm notice created a lockfile as package-lock.json. You should commit this file.

added 118 packages from 174 contributors and audited 247 packages in 8.416s

found 1 low severity vulnerability

run `npm audit fix` to fix them, or `npm audit` for details

C:\materK1m\newmimac\test\_kes\learn-express>npm start

> learn-express@0.0.0 start C:\materK1m\newmimac\test\_kes\learn-express

> node ./bin/www

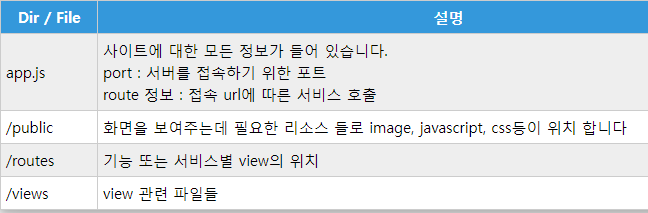
GET / 200 1797.699 ms - 170

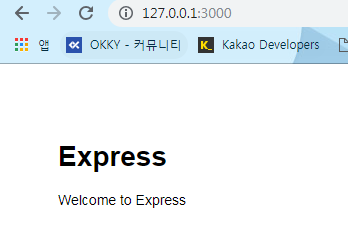
GET /stylesheets/style.css 200 17.973 ms - 111

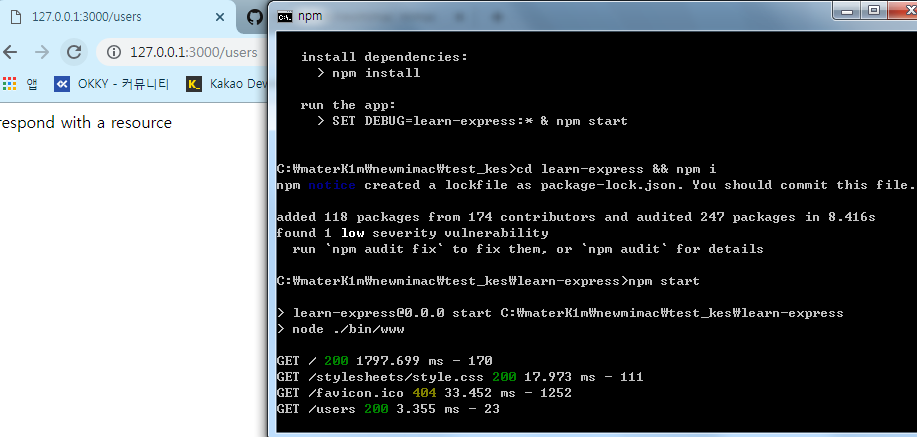
GET /favicon.ico 404 33.452 ms – 1252

**2. 익스프레스 구조 이해하기**

**[폴더 구조]**

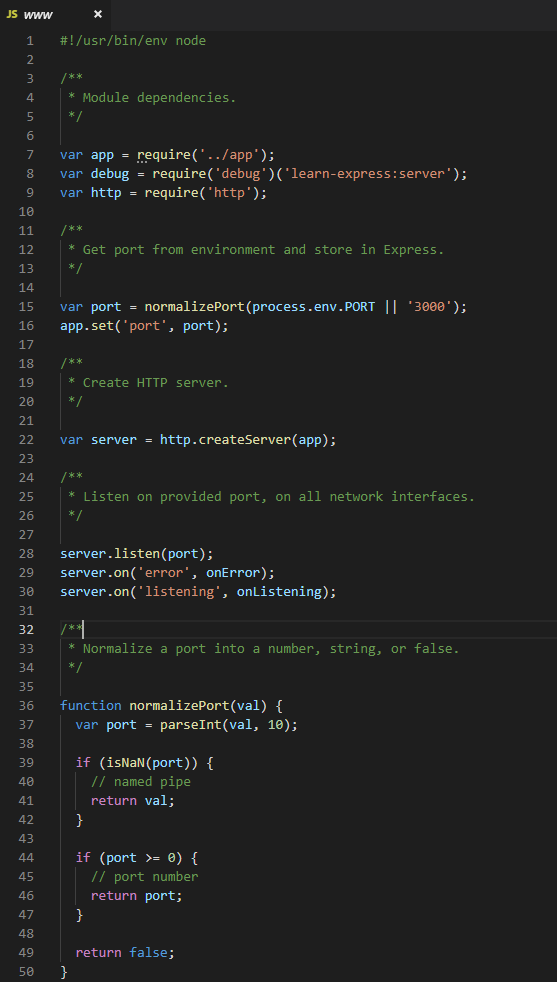






**[bin/www]**

http 모듈에 express 모듈을 연결하고, 포트를 지정하는 부분



var app = require('../app');

var debug = require('debug')('learn-express:server');

var http = require('http');

먼저, app, debug, http 모듈들을 가져옵니다.

var port = normalizePort(process.env.PORT || '3000');

app.set('port', port);

app.set(‘port’,port)로 서버가 실행될 포트를 설정합니다. process.env 객체에 PORT 속성이 있다면 그 값을 사용하고, 없다면 기본값으로 3000번 포트를 이용하도록 되어있습니다.

app.get(키)로 가져올 수 있습니다.

var server = http.createServer(app);

불러온 app 모듈을 넣어줍니다. app 모듈이 메서드의 콜백 함수 역할을 합니다.

server.listen(port);

server.on('error', onError);

server.on('listening', onListening);

listen을 하는 부분은 http 웹 서버와 동일합니다.

**[app.js]**



var app = express();

express 패키지를 호출하여 app변수 객체를 만들었습니다.

// view engine setup

app.set('views', path.join(\_\_dirname, 'views'));

app.set('view engine', 'pug');

app.set 메서드로 익스프레스 앱을 설정할 수 있습니다.

app.use(logger('dev'));

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

app.use(express.static(path.join(\_\_dirname, 'public')));

app.use('/', indexRouter);

app.use('/users', usersRouter);

// catch 404 and forward to error handler

app.use(function(req, res, next) {

next(createError(404));

});

// error handler

app.use(function(err, req, res, next) {

// set locals, only providing error in development

res.locals.message = err.message;

res.locals.error = req.app.get('env') === 'development' ? err : {};

// render the error page

res.status(err.status || 500);

res.render('error');

});

app.use로 시작하는 코드는 미들웨어를 연결하는 부분입니다.

module.exports = app;

마지막에는 app 객체 모듈로 만들었습니다. 이것이 bin/www 에서 사용된 app 모듈입니다.

**3. 미들웨어**

미들웨어는 익스프레스의 핵심입니다. 요청과 응답의 중간(middle, 미들)에 위치하여 미들웨어라고 부릅니다. 미들웨어는 요청과 응답을 조작하여 기능을 추가하기도하고, 나쁜 요청을 걸러내기도 합니다.

미들웨어는 주로 app.use와 함께 사용됩니다.

app.use(logger('dev'));

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

app.use(express.static(path.join(\_\_dirname, 'public')));

app.use('/', indexRouter);

app.use('/users', usersRouter);

// catch 404 and forward to error handler

app.use(function(req, res, next) {

next(createError(404));

});

// error handler

app.use(function(err, req, res, next) {

// set locals, only providing error in development

res.locals.message = err.message;

res.locals.error = req.app.get('env') === 'development' ? err : {};

// render the error page

res.status(err.status || 500);

res.render('error');

});

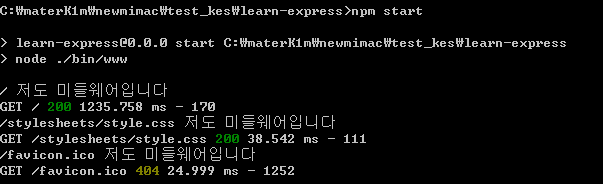
* 커스텀 미들웨어

app.use(function(req, res, next){

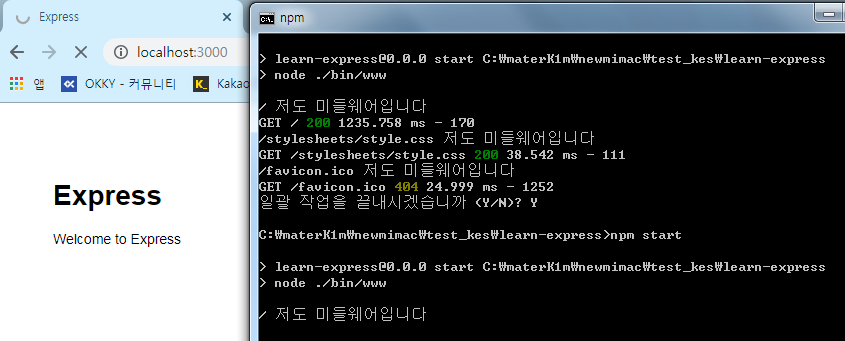
console.log(req.url, '저도 미들웨어입니다');

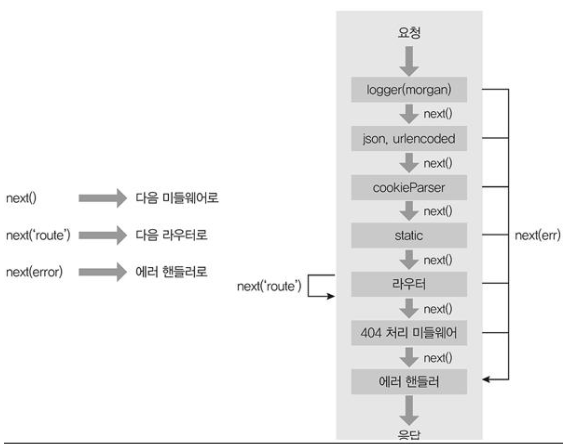
next();

});



next()를 안 넣으면 요청의 흐름이 끊김





* morgan

var logger = require('morgan');

app.use(logger('dev'));

함수의 인자로 dev 대신 short, common, combined 등을 줄 수 있다.

dev인 경우 GET / 200 51.267 ms – 1539 의 의미는 순서대로 HTTP요청(GET) 주소(/) HTTP상태코드(200) 응답속도(51.357ms) – 응답바이트(1539)입니다.

보통 개발 시에는 short나 dev를 많이 쓰고, 배포시에는 common이나 combined를 많이 사용합니다.

파일이나 데이터베이스에 로그를 남길 수 있는데 이때 모듈은 Winston

* body-parser

요청의 본문을 해석해주는 미들웨어. 보통 폼데이터나 AJAX 요청의 데이터를 처리한다.

var cookieParser = require('cookie-parser');

app.use(bodyParser.json());

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

* cookie-parser

쿠키를 해석해줌.

var cookieParser = require('cookie-parser');

app.use(cookieParser());

* static

정적인 파일들을 제공

app.use(express.static(path.join(\_\_dirname, 'public')));

함수의 인자로 정적인 파일들이 담겨있는 폴더를 지정하면 됨

* express-session

세션관리용 미들웨어

npm i express-session

var session = require('express-session');

app.use(cookieParser('secret code'));

app.use(session({

resave : false,

saveUninitialized : false,

secret : 'secret code',

cookie : {

httpOnly : true,

secure : false,

},

}));

express-session은 인자로 세션에 대한 설정을 받습니다. resave는 요청이 왓을 때 세션에 수정사항이 생기지 않더라도 세션을 다시 저장할지에 대한 설정이고, saveUninitialize는 세션에 저장할 내역이 없더라도 세션을 저장할지에 대한 설정입니다. 보통 방문자를 추적할 때 사용됩니다. secret 항목은 cookie-parser의 비밀키와 같은 역할을 합니다.

express-session은 세션 관리시 클라이언트에 쿠키를 보냅니다. 이를 세션쿠키라고 부릅니다. cookie-parser의 scret과 같게 설정해야 합니다.

cookie 옵션은 세션 쿠키에 대한 설정입니다. maxAge, domain, path, expires, sameSite, httpOnly, secure 등 일반적인 쿠키 옵션이 모두 제공됩니다.

httpOnly -> 클라이언트에서 쿠키를 확인하지 못하도록

secure는 false로 해서 https가 아닌 환경에서도 사용할 수 있게

store -> 데이터베이스에 연결하여 세션을 유지 (Redis)

세션 삭제 -> req.session.destroy()

* connect-flash

일회성 메시지들을 웹브라우저에 나타낼 때 좋다

npm i connect-flash

var flash = require('connect-flash');

app.use(flash);

req.flash(키, 값)

req.flash(키)

[routes/user.js]

router.get('/flash', function(req, res){

req.session.message = '세션 메시지';

req.flash('message', 'flash 메시지');

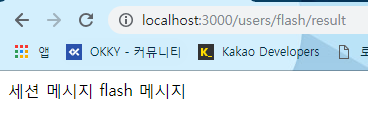
res.redirect('/user/flash/result');

});

router.get('/flash/result', function(req,res){

res.send(`${req.session.message} ${req.flash('message')}`);

});



4. Router 객체로 라우팅 분리하기

[app.js]

var indexRouter = require('./routes/index');

var usersRouter = require('./routes/users');

app.use('/', indexRouter);

app.use('/users', usersRouter);

[routes/index.js]

var express = require('express');

var router = express.Router();

/\* GET home page. \*/

router.get('/', function(req, res, next) {

res.render('index', { title: 'Express' });

});

module.exports = router;

[routes/user.js]

var express = require('express');

var router = express.Router();

/\* GET users listing. \*/

router.get('/', function(req, res, next) {

res.send('respond with a resource');

});

router.get('/flash', function(req, res) {

req.session.message = '세션 메시지';

req.flash('message', 'flash 메시지');

res.redirect('/users/flash/result');

});

router.get('/flash/result', function(req, res) {

res.send(`${req.session.message} ${req.flash('message')}`);

});

module.exports = router;

**5. 템플릿 엔진 사용하기**

**pug에 대한 설명**

* 참고 : <https://jeong-pro.tistory.com/65>
* 참고 : <https://flaviocopes.com/pug/>

**ejs에 대한 설명**

app.set('view engine', 'ejs');

npm i ejs

* 참고 : <https://www.npmjs.com/package/ejs>

**nunjucks에 대한 설명**

npm i nunjucks

* 참고 : <http://blog.naver.com/PostView.nhn?blogId=goodleedw&logNo=221058000621&redirect=Dlog&widgetTypeCall=true&directAccess=false>
* 참고 : <https://dohoons.com/blog/1376/>

**6. 에러 처리 미들웨어**

// error handler

app.use(function(err, req, res, next) {

// set locals, only providing error in development

res.locals.message = err.message;

res.locals.error = req.app.get('env') === 'development' ? err : {};

// render the error page

res.status(err.status || 500);

res.render('error');

});

[view/error.pug]

extends layout

block content

h1= message

h2= error.status

pre #{error.stack}