NodeJS 2일차

Modules

=> 모듈의 도움을 받아서 서버를 동작시킬 수 있다.

NPM : Node Package Modules

URL Module

var url = require("url");

var urlString = "http://localhost:3000/trackback/1234-1234-1111";

var parsedObject = url.parse(urlString);

console.log(parsedObject);

=> URL 요청 정보를 가지고 있습니다.

QueryString Module

=> url객체를 뒤의 trackback/1234-1234-1111 가지고와서 정보를 알 수 있게 해준다.

File System Module

Node의 Javascript차이점

파일시스템에 쓰고 지우는 접근이 가능하다

Node에 파일시스템은

readFile : 파일을 비동기적으로 읽는다..

readFileSync : 파일을 동기적으로 읽는다.

writeFile : 파일을 비동기적으로 쓴다.

wrtieFileSync : 파일을 동기적으로 쓴다

동기 => 업무처리방식에서 업무가 끝나지 않고 다음 것으로 넘어오지 않으면 그 업무가 끝날 떄까지 기다렸다가 업무를 처리하는 방식

비동기 => 업무처리에서 그전의 업무가 처리되지않아도 그 다음 업무를 처리할 수 있는 방식

파일을 동기로 읽는 방법

var fs = require("fs");

var text = fs.readFileSync("textFile.txt","utf-8");

console.log(text);

파일을 비동기로 읽는 방법

var fs = require("fs");

fs.readFile("textFile.txt","utf-8",function(error,data{

console.log(data):

}

http 모듈

http.createServer => http를 생성해준다.

http는 기본적으로 한가지의 페이지를 보여줄 수 있다.

response.writeHead(200, body에 실려질 타입);

=> 200의 요청은 요청이 잘왔다는 것이다.

response.end(body에 내용을 보여준다)

response.end("<h1>Hello World!</h1>");

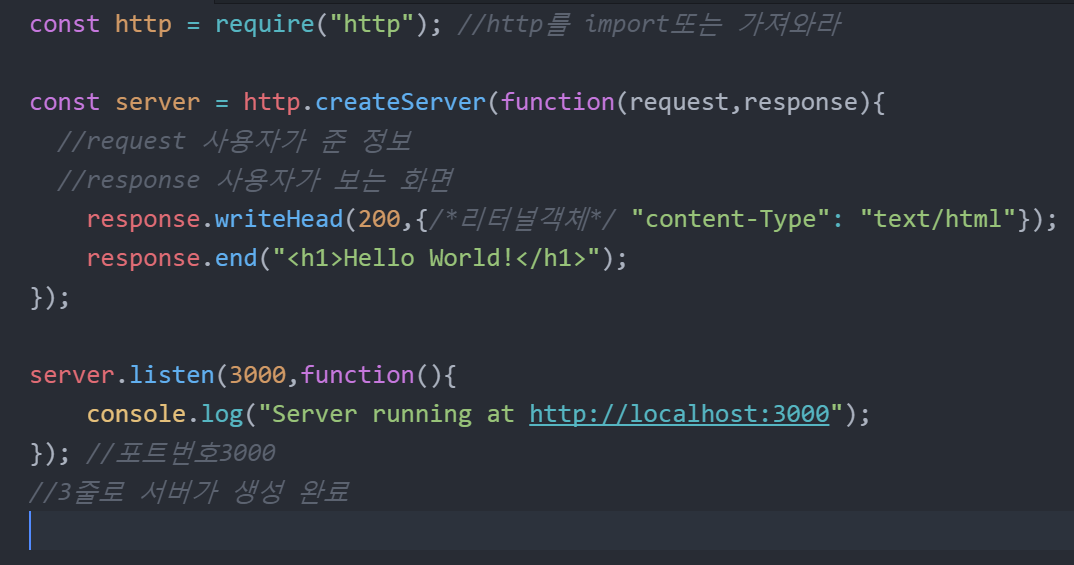
httpServer 실행

server.listen(3000,function(){

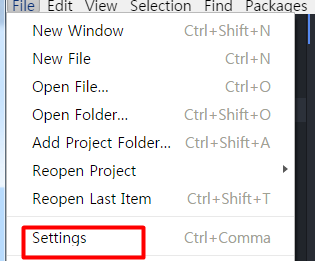
console.log("Server running"):

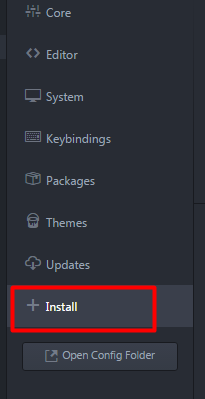
} => 서버가 응답이 잘되면 3000, 잘되었을떄 후처리로 function을 동작시킨다.

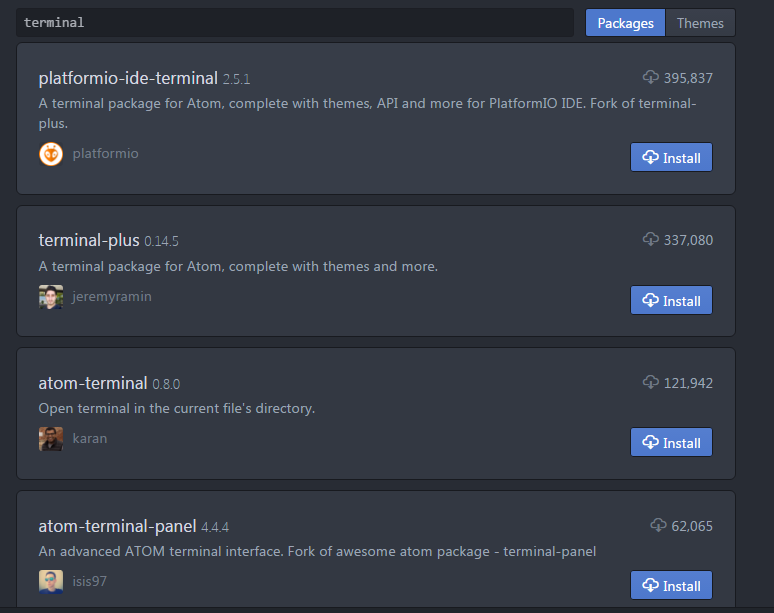
**서버 생성 및 실행** 실습



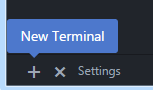
서버를 작동시킬 수 있는 atom에 플러그인 설치하기

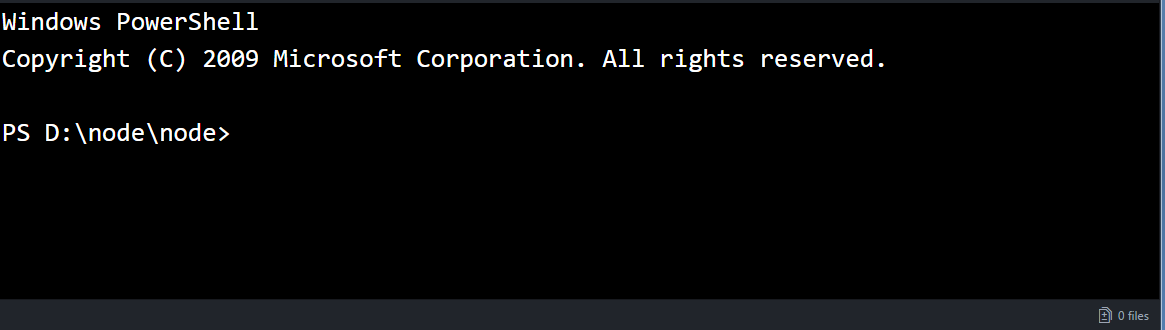






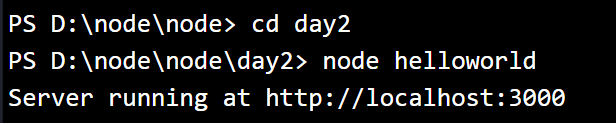
설치완료후 왼쪽하단





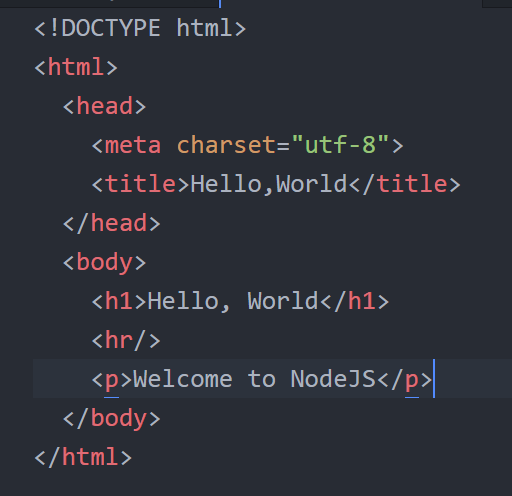
cmd 창을 이용할 수 있다.

서버 실행 테스트



**helloworld.html 생성**

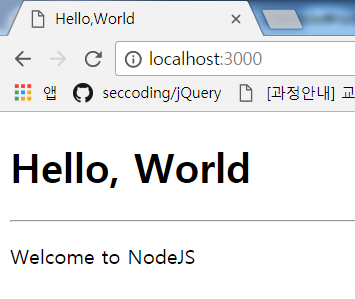
html 입력 후 엔터치면 html작성문서 자동으로 생성



**helloworld.js 수정 파일시스템방식으로**



**실행결과**



**=> 문제점 다른페이지로 이동을 못한다.**

**=> 다른모듈의 힘을 빌려서 여러페이지를 이용 할 수 있도록 해준다.**

**=> URL을 이용 var url = require("url");**

**URL 모듈**

helloworld.js 수정

const http = require("http"); //http를 import또는 가져와라

const fs = require("fs"); //파일시스템 모듈 생성

const url = require("url");

const server = http.createServer(function(request,response){

//request 사용자가 준 정보

//response 사용자가 보는 화면

const urlInfo = url.parse(request.url);

console.log(urlInfo);

const pathName = urlInfo.pathName;

console.log(pathName);

if(pathName == "/index"){

fs.readFile("helloworld.html","utf-8",function(error,data){

response.writeHead(200,{/\*리터널객체\*/ "content-Type": "text/html"});

response.end(data);

});//현재폴더기준위치

}

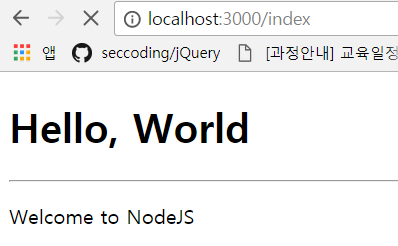
});

server.listen(3000,function(){

console.log("Server running at http://localhost:3000");

}); //포트번호3000

//3줄로 서버가 생성 완료



요청 url이 없을 때 오류 페이지 나오도록 추가

helloworld.js

const http = require("http"); //http를 import또는 가져와라

const fs = require("fs"); //파일시스템 모듈 생성

const url = require("url");

const server = http.createServer(function(request,response){

//request 사용자가 준 정보

//response 사용자가 보는 화면

const urlInfo = url.parse(request.url);

console.log(urlInfo);

const pathName = urlInfo.pathName;

console.log(pathName);

if(pathName == "/index"){

fs.readFile("helloworld.html","utf-8",function(error,data){

response.writeHead(200,{/\*리터널객체\*/ "content-Type": "text/html"});

response.end(data);

});//현재폴더기준위치

}

else{

response.writeHead(404,{"content-Type": "text/html"});

response.end();

}

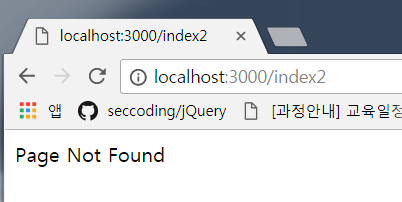
});

server.listen(3000,function(){

console.log("Server running at http://localhost:3000");

}); //포트번호3000

//3줄로 서버가 생성 완료



웹어플리케이션에서 요청정보가 중요하다. GET방식/POST방식

request.on('data') => data안에는 페이지에서 전송한 form데이터

login.html 생성



**helloworld.js 수정**

const http = require("http"); //http를 import또는 가져와라

const fs = require("fs"); //파일시스템 모듈 생성

const url = require("url");

const server = http.createServer(function(request,response){

//request 사용자가 준 정보

//response 사용자가 보는 화면

const urlInfo = url.parse(request.url);

console.log(urlInfo);

const pathName = urlInfo.pathname;

console.log(pathName);

if(pathName == "/index"){

fs.readFile("helloworld.html","utf-8",function(error,data){

response.writeHead(200,{/\*리터널객체\*/ "Content-Type": "text/html"});

response.end(data);

});//현재폴더기준위치

}//if

else if(pathName == "/login"){

if (request.method=="GET") {

fs.readFile("login.html","utf-8",function(error,data){

response.writeHead(200,{/\*리터널객체\*/ "Content-Type": "text/html"});

response.end(data);

});

}

else if(request.method=="POST"){

request.on("data",function(data){

console.log(data);

response.writeHead(200,{/\*리터널객체\*/ "Content-Type": "text/html"});

response.end(data);

});//request에 데이터를 바인딩하겠다.

}

}//else if

else{

response.writeHead(404,{"Content-Type": "text/html"});

response.end("Page Not Found");

}//else

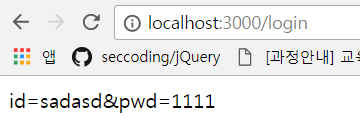
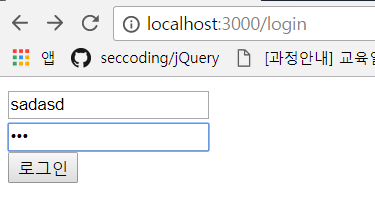
});

server.listen(3000,function(){

console.log("Server running at http://localhost:3000");

}); //포트번호3000

//3줄로 서버가 생성 완료



Template Engine => JSP와 동일하다고 생각하면 된다.

서버가 전달한 페이지와 함께 데이터를 같이 보여준다.

EJS

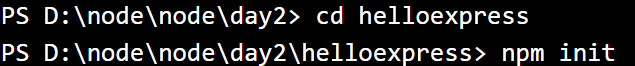
**Express Framework => Node.js를 사용한다는 Express를 사용한다와 같다.**

HelloExpress(Empty Project)를 만들고 npm init명령 수행

npm 프로젝트 셋팅하기

json파일형태로 프로젝트가 생성되면 npm프로젝트가 셋팅되어진것이다.

helloexpress 폴더생성



엔터키로 하면서 기본셋팅되어져있는 상태를 이용한다.

=> 기본값으로 설정되어서 npm프로젝트가 생성된다. package.json생성되어진다.





node\_modules 폴더가 생성되어진다.

package.json이 변경되어져있다.





=> 모든 어플리케이션을 다 설치

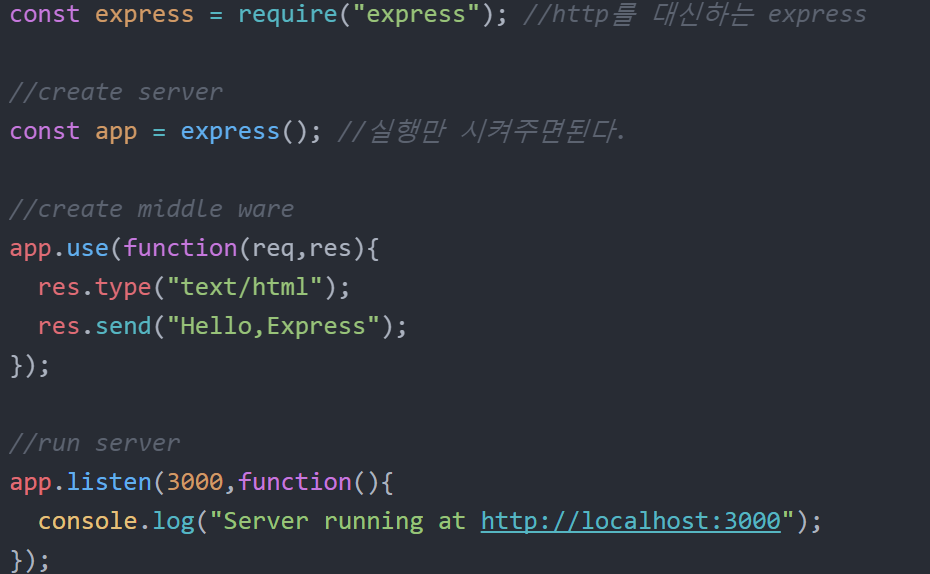


--save 는 package.json에 ejs를 넣어주어라 의미를 가지고있다.

Hello, Express 실습

helloexpress 폴더에 index.js 파일생성 package.json에 main부분에 쓰여있는 js파일을 일치시켜주어야한다.

**index.js**



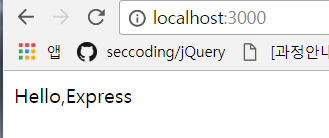
package.json script부븐을

"scripts": {

"start" : "node index"

},

변경시켜주고 npm start 작동시켜주면 서버가 작동되어진다.



미들웨어 : 요청과 응답 사이에서 동작하는 모듈을 의미한다.

aap.use([미들웨어 함수]); 사용한다.

Express와 함께 쓰이면 좋은 미들웨어

router : 페이지 라우팅을 지원

static : 정적 자원을 서비스 할떄 사용

morgan : 요청이 들어올떄마다 로그를 기록

cooke parser : 요청 쿠키를 추출

body parser : POST 요청 데이터를 추출

connect multiparty : multipart/form-data 요청 데이터를 추출

express-session : 세션을 쉽게 사용 할 수 있도록 지원 세션을 만든다.

Express 모듈에서 Request 객체는 속성 메소드 지원

params : 라우팅 매개변수를 추출 => Path Params

query : 요청 매개변수를 추출 => Form/Query Params

blog.naver.com/post/1

=> url의 특정 변수 : 1 , 1번글을 읽어라 params

=> query 는 ?id=100

Page Router 방식

미들웨어를 사용하면 하나의 url만 처리가능

여러가지 url을 제공하려면 page Router를 사용해야한다.

**index.js**

const express = require("express"); //http를 대신하는 express

//create server

const app = express(); //실행만 시켜주면된다.

//create middle ware

app.use(function(req,res){

res.type("text/html");

res.send("Hello,Express");

});

//app는 서버

app.get("/index",function(req,res){

//파일시스템가져와서 읽어준다.

fs.readFile("helloworld.html","utf-8",function(error,data){

res.type("text/html");

res.send(data);

});//현재폴더기준위치

});

app.get("/login",function(req,res){

fs.readFile("login.html","utf-8",function(error,data){

res.type("text/html");

res.send(data);

});//현재폴더기준위치

});

app.post("/login",function(req,res){

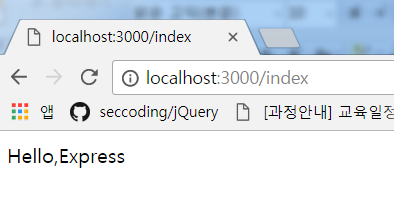
});

//run server

app.listen(3000,function(){

console.log("Server running at http://localhost:3000");

});



미들웨어의 순서의 중요성

=>서버를 요청할 때 미들웨어를 먼저 찾기 때문에 app.use를 먼저 실행시킨다.

순서를 바꿔주어야한다.

app.use를 app.listen위에 옮겨주어야한다.

//create middle ware

app.use(function(req,res){

res.type("text/html");

res.send("Hello,Express");

});

//run server

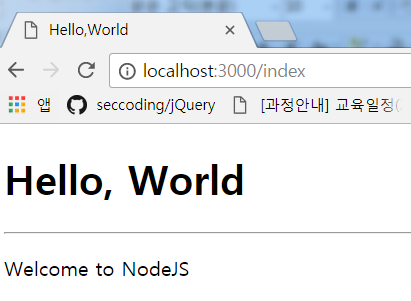
app.listen(3000,function(){

console.log("Server running at http://localhost:3000");

});

const fs = require("fs");

지정해주어야 파일시스템이 잘작동되어진다.



board폴더생성 => npm init 설치

package.json 스크립트 변경

"scripts": {

"start" : "node index"

},

npm install --save express

npm install --save ejs 설치



**board폴더에 view폴더 생성**

**index.js**

const express = require("express");

const app = express(); //서버생성

const fs = require("fs");

const ejs = require("ejs");

const bodyparser = require("body-parser"); //form형식 넘길떄 이용

// 사용자화면에 보여줄 데이터

const mockData = [

{ subject : "Mock Subject", writer : "Mock Writer", content : "Mock Content"}

];

app.get("/:id",function(req,res){

let id = req.params.id; //params뒤에 붙는것이 :id이다

const item = mockData[id]; //위에서 지정한 데이터를 얻어와서 item에 넣어준다

fs.readFile("view/detail.html","utf-8",function(error,data){

res.type("text/html");

res.send(ejs.render(data,item /\* 리터널만 적어줄수있다.\*/));

//item을 data에 보내주겟다 페이지를 보여줄떄 item도 같이 보여줘리

});

});

app.listen(3000,function(){

console.log("Server running at http://localhost:3000");

});

**view폴더에 detail.html생성**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title><%= subject %></title>

<!-- 전달받은 데이터를 받아라-->

</head>

<body>

<h1><%= subject%> </h1>

<h3><%= writer%></h3>

<hr/>

<p><%= content%></p>

</body></html>

