



## 웹 DB 구축 절차

### 1. 설치 - 웹 서버, **WAS** 서버, **DB** 서버

→ **Home Directory** 이해

### 2. 웹 서버(**WAS** 서버) 구동

### 3. 클라이언트(브라우저)로부터 요청 분석 및 처리

→ **URL** 을 통해 요청함으로 요청(**request**) 객체에 포함된 **URL** 획득 및 사용 방법 습득

→ **URL** 분류기 이해

→ **URL**에 질의 **keyword** 전송하고 요청 객체로부터 **keyword** 획득 방법 : **QueryString** 이해

### 4. 클라이언트(브라우저)에게 **HTML** 문서로 응답하는 방법

→ 응답(**response**) 객체에 보내줄 내용을 전달하여 브라우저에게 보내는 방법 습득 : **end** 메소드에 직접 **html** 작성

→ 응답해줄 **template** 문서 작성 하는 방법 습득 : **ejs**

. **html** 문서로의 **rendering** 개념 습득

. **template** 문서와 **business logic** 사이에 자료를 주고 받는 방법 습득 : **context**와 **form**

### 5. 요청에 대한 처리 부분(**business logic**) 작성하는 방법

→ **DB** 에 대한 **CRUD** 작성 : 동적 **web** 페이지 만들기



## 2주차 수업의 범위(웹 DB의 범위)

---

- \* **URL** 이해
- \* **URL** 분류기 이해
- \* **Home Directory** 이해
- \* **QueryString** 이해
- \* 동적 **web** 페이지 만들기 ( **end** 메소드에 직접 **html** 작성)
- \* **pm2** 설치
- \* **express** 설치 및 **express**를 이용한 웹서버 구축 및 동적 **web** 페이지 작성
- \* **ejs** 설치 및 작성

**http://opentutorials.org:3000/**

**http://opentutorials.org:3000/main**

**http://opentutorials.org:3000/main/side**

**http://opentutorials.org:3000/main?id=HTML&page=12**

1. **http** : 프로토콜

2. **opentutorials.org** 호스트(도메인 네임)

3. **3000** : 포트 번호

→ **URL**에 매핑 되는 **root** 디렉토리가 홈 디렉토리

→ **main.js**를 실행 시킨 디렉토리가 홈디렉토리가 됨

4. **main, main/side** : 다른 페이지를 불러오는 **URL** 부분, 요청의 종류

5. **?** : 이후에 나오는 것이 **query string**임을 알려줌

→ **query string** : 질의 단어, 조건 제시 기능

6. **id=HTML&page=12** : **querystring**, **&** 는 두 개의 변수를 연결

## 1. 사용자가 요청한 URL 구분 - main.js

```
var http = require('http');
var fs = require('fs'); // ① 파일 관련된 작업을 수행
var app = http.createServer(function(req, res) {
  var _url = req.url; // ② 사용자가 요청한 URL, request객체의 url 속성
  console.log(_url); // ③ _url을 console에 출력
  console.log(__dirname)
  if(req.url == '/') { // ④ req.url에 따라 _url 변경
    _url = '/index.html';
  }
  if(req.url == '/favicon.ico') { // ⑤ favicon.ico 5page 참조
    return res.writeHead(404); // 응답 헤더에 상태코드 추가 6page 참조
  }
  res.writeHead(200);
  res.end(fs.readFileSync(__dirname + _url)); // ⑥
});
app.listen(3000);
```

① fs 모듈 : 파일 관련된 작업을 수행하는 도구를 제공하는 모듈

파일을 읽고 쓰고 삭제하는 등의 기능

⑥ readFileSync : 매개변수로 넘겨진 파일을 읽는다.

\_\_dirname : \_\_는 js에서 기본적으로 제공하는 변수에 붙음

현재 파일이 위치한 폴더의 절대경로를 저장

현재 경로에 있는 \_url 파일의 내용을 읽어서 end 메소드의 인자로 넘겨줌.

**[work]** end 메소드에 html을 직접 작성하시오.



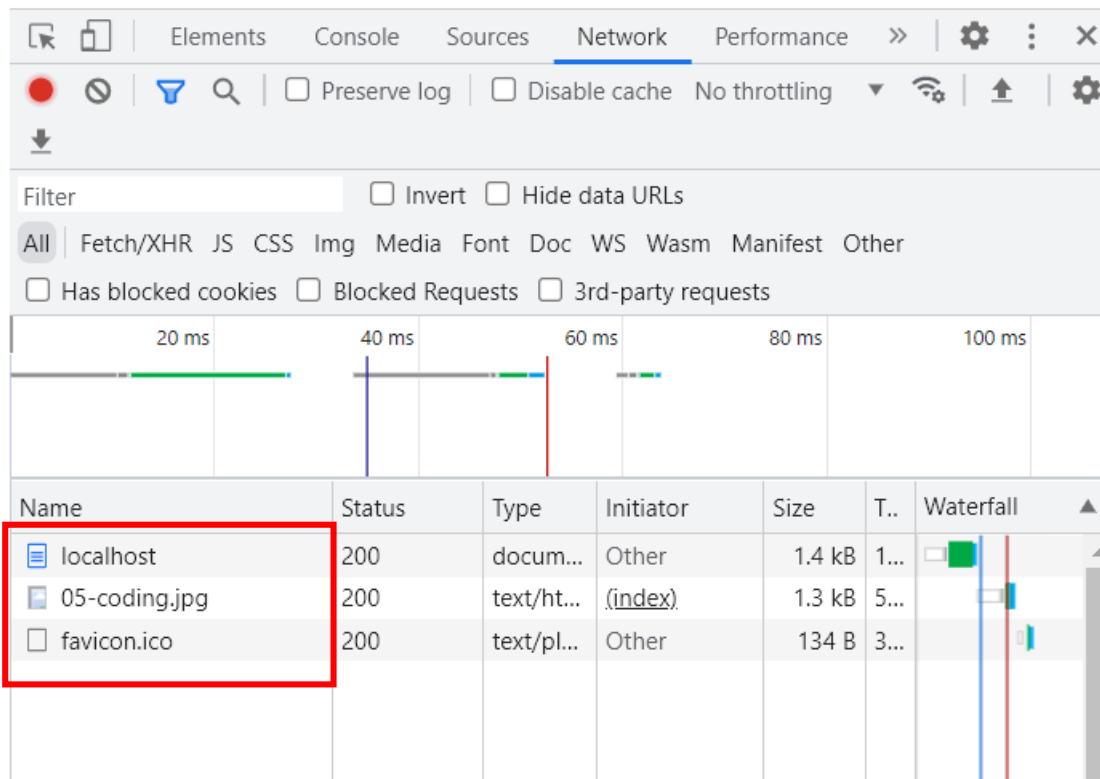
1) favorite icon의 약자

2) /favicon.ico

browser가 서버에게 자동으로 요청하는 URL → 127.0.0.1:3000/favicon.ico

3) 좌측 이미지가 favicon.ico  www.gachon.ac.kr

4) 오른쪽 마우스 버튼 → 검사메뉴 → Network 탭 : 브라우저가 요청하는 객체들을 볼 수 있음.





1) 클라이언트로부터의 **HTTP** 요청이 성공적으로 완료되었는지 알려줌

2) 응답은 **5**개의 그룹으로 나눔

① 정보를 제공하는 응답 : 100 ~ 103

② 성공적인 응답 : 200 ~ 208, 226

③ 리다이렉트 : 300 ~ 308

④ 클라이언트 에러 : 400 ~ 418 , 421 ~ 424 , 426 , 428 , 429 , 431 , 451

⑤ 서버 에러 : 500 ~ 508 , 510 , 511

3) 상태 코드는 다음 링크에 나와 있음

**<https://datatracker.ietf.org/doc/html/rfc2616#section-10>**

## 2. /main 경로가 붙은 URL 추가: if 문은 URL 분류기 역할

http://localhost:3000/main 추가

```
var http = require('http');
var fs = require('fs');
var app = http.createServer(function(req, res){

  var _url = req.url;
  console.log('요청받은 url :'+_url)
  console.log(__dirname)
  if(req.url == '/') {
    _url = '/index.html';
  }
  if(req.url == '/main'){    // ① 경로가 붙은 URL
    _url = '/main.html';
  }
  if(req.url == '/favicon.ico') {
    return res.writeHead(404);
  }
  res.writeHead(200);
  res.end(fs.readFileSync(__dirname + _url));
});
app.listen(3000)
```

[work]

/1 → 1.html

/2 → 2.html

/3 → 3.html

### 3. URL에서 쿼리 스트링 추출

`http://localhost/?id=HTML` : 주소의 마지막에 있는 **querystring**값에 따라 다른 콘텐츠를 보여줌

```
var http = require('http');
var fs = require('fs');
var urlm = require('url');    // ① url 모듈 요청, url 모듈에 담긴 기능 사용 가능
var app = http.createServer(function(req, res) {
  var _url = req.url;
  var queryData = urlm.parse(_url, true).query // ② url에서 querystring 문자열만 추출
  console.log(_url);
  console.log(queryData);    // ③ queryData.id로 바꾸어서도 실행 시켜보기

  if(req.url == '/') {
    _url = '/index.html';
  }
  if(req.url == '/favicon.ico') {
    return res.writeHead(404);
  }
  res.writeHead(200);
  res.end(fs.readFileSync(__dirname + _url));
});
app.listen(3000);
```

② 객체를 return {id : 'HTML'}

[입력]

← → ↻ ⓘ localhost:3000/?id=HTML

YouTube 지도

[③ 결과]

```
[Object: null prototype] { id: 'HTML' }
/?id=HTML
```

**[work1]** parse 메소드의 true 인자를 삭제하고 실행하여 결과를 보시오.

**[work2]** parse 메소드의 path, pathname 속성의 값을 출력하시오.



#### 4. 요청 값에 따라 다르게 응답하기

```
var http = require('http');
var fs = require('fs');
var urlm = require('url'); // ① url 모듈 요청, url 모듈에 담긴 기능 사용 가능
var app = http.createServer(function(req, res) {
  var _url = request.url;
  var queryData = urlm.parse(_url, true).query // ② url에서 querystring 문자열만 추출
  console.log(_url);
  console.log(queryData); // ③ queryData.id로 바꾸어서도 실행 시켜보기

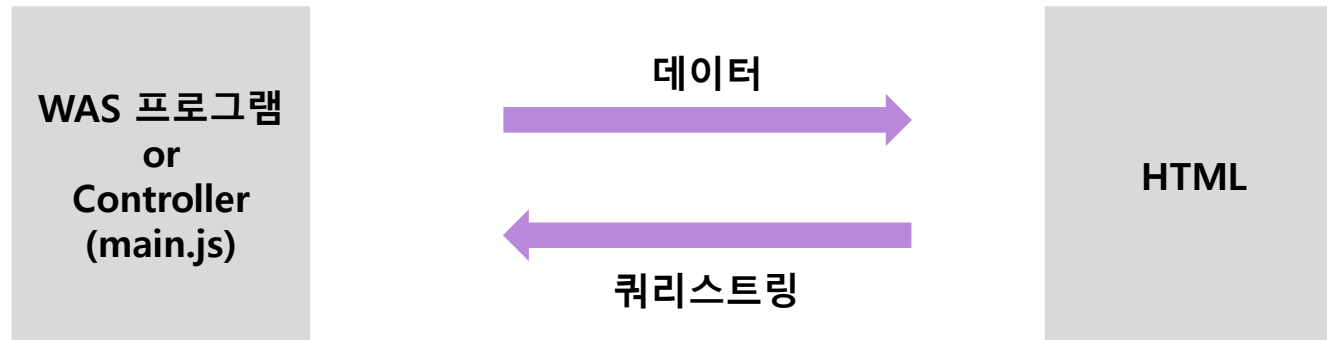
  if(req.url == '/') {
    _url = '/index.html';
  }
  if(req.url == '/favicon.ico') {
    return res.writeHead(404);
  }
  res.writeHead(200);
  res.end(queryData.id); // ④ 추가
  //res.end(fs.readFileSync(__dirname + _url)); // ⑤ 주석처리
});
app.listen(3000);
```



## 동적 페이지를 작성하기 위한 WAS와 HTML간의 자료 주고 받는 방법

\* response 객체에 HTML 문서를 전달해주는 방법

- ① end 메소드에 직접 html 작성
- ② end 메소드에 fs 모듈을 이용하여 html 파일의 내용을 읽어옴
- ③ template 언어 이용



# App – 동적인 웹 페이지 만들기(① end 메소드에 직접 html)

## 1. 요청 값에 따라 다르게 응답하기 - 프로그램

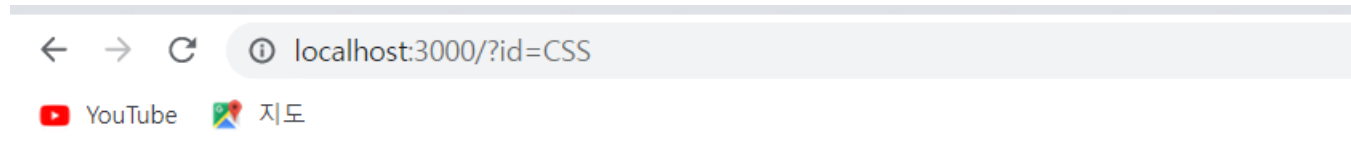
```
var http = require('http');
var fs = require('fs');
var urlm = require('url');
var app = http.createServer(function(req, res) {
  var _url = req.url;
  var queryData = urlm.parse(_url, true).query
  console.log(_url);
  console.log(queryData.id);

  if(req.url == '/') {
    // url = '/05-index.html'; ① 주석처리
  }
  if(req.url == '/favicon.ico') {
    return res.writeHead(404);
  }
  res.writeHead(200);

  // ② var template 추가
  var template = `
<!doctype html>
<html>
<head>
  <title>WEB1 - ${queryData.id}</title> // ③ HTML에서 변수사용
  <meta charset="utf-8">
</head>
<body>
  <h1><a href="05-index.html">WEB</a></h1>
```

```
<ol>
  <li><a href="1.html">HTML</a></li>
  <li><a href="2.html">CSS</a></li>
  <li><a href="3.html">JavaScript</a></li>
</ol>
<h2>${queryData.id}</h2> // ④ HTML에서 변수사용
<p><a href="https://www.w3.org/TR/html5/" target="_blank"
title="html5 specification">Hypertext Markup Language (HTML)</a> is
the standard markup language for <strong>creating <u>web</u>
pages</strong> and web applications.web browsers receive HTML
documents from a web server or from local storage and render them into
multimedia web pages. HTML describes the structure of a web page
semantically and originally included cues for the appearance of the
document.
  
</p><p style="margin-top:45px;">HTML elements are the building
blocks of HTML pages. with HTML constructs, images and other objects,
such as interactive forms, may be embedded into the rendered page. It
provides a means to create structured documents by denoting structural
semantics for text such as headings, paragraphs, lists, links, quotes
and other items. HTML elements are delineated by tags, written using
angle brackets.
  </p>
</body>
</html> ` ;
res.end(template); // ⑤ 수정
});
app.listen(3000);
```

## 1. 요청 값에 따라 다르게 응답하기 - 결과



### WEB

1. [HTML](#)
2. [CSS](#)
3. [JavaScript](#)

### CSS

[Hypertext Markup Language \(HTML\)](#) is the standard markup language for **creating web pages** and rendering them into multimedia web pages. HTML describes the structure of a web page semantically.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects, such as interactive forms, can be embedded within the structured documents by denoting structural semantics for text such as headings, paragraphs, and lists.

## App – 동적인 웹 페이지 만들기(① end 메소드에 직접 html)

## 2. 프로그램 수정하기

main.js

```

var http = require('http');
var fs = require('fs');
var urlm = require('url');
var app = http.createServer(function(req, res) {
  var _url = req.url;
  var queryData = urlm.parse(_url, true).query
  console.log(queryData.id);
  var title = queryData.id // ① title 변수에 저장
  console.log(_url);
  if(_url == '/') {
    title = 'welcome'; // ② undefined 방지
    // url = '/05-index.html'; ③ 주석처리
  }
  if(_url == '/favicon.ico') {
    return res.writeHead(404);
  }
  res.writeHead(200);

  var template = `
<!doctype html>
<html>
<head>
  <title>WEB1 - ${title}</title> // ④ title 로 수정
  <meta charset="utf-8">
</head>
<body>
  <h1><a href="/">WEB</a></h1> // ⑤ href값을 /로 수정

```

```

<ol>
  <li><a href="/?id=HTML">HTML</a></li> // ⑥ href값 바꾸기
  <li><a href="/?id=CSS">CSS</a></li>
  <li><a href="/?id=JavaScript">JavaScript</a></li>
</ol>
<h2>${title}</h2>
<p><a href="https://www.w3.org/TR/html5/" target="_blank"
title="html5 specification">Hypertext Markup Language (HTML)</a> is
the standard markup language for <strong>creating <u>web</u>
pages</strong> and web applications. Web browsers receive HTML
documents from a web server or from local storage and render them into
multimedia web pages. HTML describes the structure of a web page
semantically and originally included cues for the appearance of the
document.

</p><p style="margin-top:45px;">HTML elements are the building
blocks of HTML pages. With HTML constructs, images and other objects,
such as interactive forms, may be embedded into the rendered page. It
provides a means to create structured documents by denoting structural
semantics for text such as headings, paragraphs, lists, links, quotes
and other items. HTML elements are delineated by tags, written using
angle brackets.
</p>
</body>
</html> ` ;
res.end(template);

});
app.listen(3000);

```



### 3. 장점

- 1) 목록이 있는 순서를 목록이 없는 순서로 바꿀 경우 **1.html ~ 3.html**을 모두 바꾸어야 하지만 앞선 프로그램에서는 그럴 필요가 없음.

### 4. 단점

- 1) 아직 내용은 동적으로 바뀌지 않았음

## Node.js – 패키지 매니저와 PM2

### 1) 패키지 매니저 - npm

- \* 패키지를 설치, 업데이트, 삭제하는 등 관리하는 데 도움을 주는 프로그램

### 2) PM2 설치

- \* **Node.js**로 만든 프로세스를 관리해주는 프로그램
- \* 프로그램을 감시하고 있다가 의도하지 않게 꺼지거나 소스가 변경될 때 자동으로 재시동
- \* 터미널에서 **npm**을 이용하여 설치

```
PS C:\작업들\5_강의자료\2022-2학기\웹DB프로그래밍\programs\01> npm install pm2 -g
```

### 3) PM2 실행

- \* **PM2** 동작 확인
- \* 만약 다음과 같은 오류가 난다면 **Windows Powershell**을 관리자로 실행한 다음

```
PS C:\작업들\5_강의자료\2022-2학기\웹DB프로그래밍\programs\01> pm2 monit
pm2 : 이 시스템에서 스크립트를 실행할 수 없으므로 C:\Users\WBH\AppData\Roaming\npm\pm2.ps1 파일을 로드할 수 없습니다. 자세한 내용은 about_Execution_Policies(https://go.microsoft.com/fwlink/?LinkID=135170)를 참조하십시오.
위치 줄:1 문자:1
+ pm2 monit
+ ~~~
+ CategoryInfo          : 보안 오류: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
```

**Get-ExecutionPolicy** 를 입력하여 **restricted**라고 확인될 경우

**set-executionpolicy remotesigned** 라고 입력하고 **y**를 입력

[참조] <https://connieya.tistory.com/68>



## Node.js – 패키지 매니저와 PM2

### 3) PM2 실행

#### \* PM2 동작 확인

```
PS C:\작업들\5_강의자료\2022-2학기\웹DB프로그래밍\programs\01> pm2 start 26-main-5.js
[PM2] Starting C:\작업들\5_강의자료\2022-2학기\웹DB프로그래밍\programs\01\26-main-5.js in fork_mode (1 instance)
[PM2] Done.
```

id	name	namespace	version	mode	pid	uptime	⌚	status	cpu	mem	user	watching
0	26-main-5	default	N/A	fork	45632	0s	0	online	0%	32.0mb	WBH	disabled

프로그램 이름

실행중

프로세스를 중단할 때

```
PS C:\작업들\5_강의자료\2022-2학기\웹DB프로그래밍\programs\01> pm2 stop 26-main-5
[PM2] Applying action stopProcessId on app [26-main-5](ids: [ 0 ])
[PM2] [26-main-5](0) ✓
```

id	name	namespace	version	mode	pid	uptime	⌚	status	cpu	mem	user	watching
0	26-main-5	default	N/A	fork	0	0	0	stopped	0%	0b	WBH	disabled

### 3) PM2 실행

\* 프로세스 감시

```
PS C:\작업들\5_강의자료\2022-2학기\웹DB프로그래밍\programs\01> pm2 monit
```

The terminal output of the `pm2 monit` command is displayed in a dark-themed window. It is divided into four main sections: Process List, 26-main-5 Logs, Custom Metrics, and Metadata.

**Process List**

Process Name	Mem	CPU
[ 0 ] 26-main-5	29 MB	0

**26-main-5 Logs**

(Log content is empty in this view)

**Custom Metrics**

Used Heap Size	6.96 MiB
Heap Usage	86.69%
Heap Size	8.03 MiB

**Metadata**

App Name	26-main-5
Namespace	default
Version	N/A

left/right: switch boards | up/down/mouse: scroll | Ctrl-C: exit

To go further check out <https://pm2.io/>

## Node.js – 패키지 매니저와 PM2

### 3) PM2 실행

\* 소스 파일 감시 - 소스를 고치고 프로그램을 재시동했던 방식을 자동화 할 수 있음.

```
PS C:\작업들\5_강의자료\2022-2학기\웹DB프로그래밍\programs\01> pm2 start 26-main-5.js --watch
[PM2] Applying action restartProcessId on app [26-main-5](ids: [ 0 ])
[PM2] [26-main-5](0) ✓
[PM2] Process successfully started
```

id	name	namespace	version	mode	pid	uptime	σ	status	cpu	mem	user	watching
0	26-main-5	default	N/A	fork	46804	0s	0	online	0%	32.5mb	WBH	enabled

\* 실행 로그를 확인 - 오류 발생 등의 로그를 확인할 수 있음.

```
PS C:\작업들\5_강의자료\2022-2학기\웹DB프로그래밍\programs\01> pm2 log
[TAILING] Tailing last 15 lines for [all] processes (change the value with --lines option)
C:\Users\WBH\.pm2\pm2.log last 15 lines:
PM2      | 2022-08-20T22:07:49: PM2 log: pid=46804 msg=process killed
PM2      | 2022-08-20T22:07:49: PM2 log: App [26-main-5:0] starting in -fork mode-
PM2      | 2022-08-20T22:07:49: PM2 log: App [26-main-5:0] online
PM2      | 2022-08-20T22:08:04: PM2 log: Change detected on path 26-main-5.js for app 26-main-5 - restarting
PM2      | 2022-08-20T22:08:04: PM2 log: Stopping app:26-main-5 id:0
PM2      | 2022-08-20T22:08:06: PM2 log: App [26-main-5:0] exited with code [1] via signal [SIGINT]
PM2      | 2022-08-20T22:08:06: PM2 log: pid=47436 msg=process killed
PM2      | 2022-08-20T22:08:06: PM2 log: App [26-main-5:0] starting in -fork mode-
PM2      | 2022-08-20T22:08:06: PM2 log: App [26-main-5:0] online
PM2      | 2022-08-20T22:08:11: PM2 log: Change detected on path 26-main-5.js for app 26-main-5 - restarting
PM2      | 2022-08-20T22:08:11: PM2 log: Stopping app:26-main-5 id:0
```

1. **Express : Node.js** 위에서 동작하는 웹 프레임워크

2. 반복적으로 등장하는 기능을 처리할 때 더 적은 코드로 처리할 수 있도록 효율성 증가

반복적인 일 - **URL** 파라미터를 통해 전달된 데이터로 작업

정적인 자바스크립트 파일, 이미지 파일 등을 제공하는 기능

로그인 기능, 보안 기능 등

3. 3장 목표 -**template** 언어 적용

1) 보안, 안정성, 성능, 동시성을 **DB**가 제어함

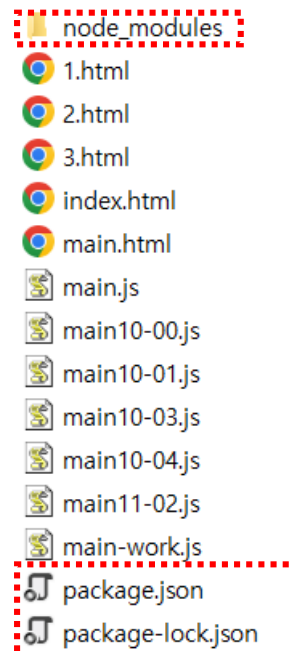
## 1. Express 설치 및 Hello World 실행하기

① <https://expressjs.com> 들어가서 명령어 copy

② 터미널에서 express 설치

```
03> npm install express --save
```

③ 우측과 같이 node\_modules 폴더 생성  
package.json과 package-lock.json 파일 생성



## 1. Express 설치 및 Hello World 실행하기

④ main.js 파일에 다음을 코딩

```
const express = require('express') // ① express 모듈을 import. const에 의해 express 변수는 값이 바뀌지 않음
const app = express() // ② express() 함수에 의해 Application 객체를 app에 저장
app.get('/', (req, res) => res.send('Hello world')) // ③
app.get('/author', (req, res) => res.send('/author'))
app.listen(3000, () => console.log('Example app listening on port 3000')) // ④
```

① http 모듈의 요청과 응답 객체에 추가 기능을 부여하였음.

② 마치 http 모듈에서 반환된 객체의 createServer 메소드와 유사

③ get(path, callback) : 사용자가 해당 경로를 요청하면 callback 실행

get 메소드는 라우팅(URL 분류기) 기능을 수행 : 요청된 경로마다 응답해 주는 기능

더 이상 if else로 URL 분류할 필요 없음

res.send 메소드 : express에서 res 객체에 send 메소드 추가. end 메소드와 같은 기능

④ 웹의 요청을 수신, 이벤트 루프

## 2. Express로 동적 페이지 만들기 - 13page 프로그램과 같음

```
const express = require('express');
const app = express();
var urlm = require('url');

app.get('/', (req, res) => {
  var _url = req.url;
  title = 'welcome';
  var queryData = urlm.parse(_url, true).query
  console.log(queryData.id);
  var title = queryData.id ;

  var template = `
<!doctype html>
<html>
<head>
<title>WEB1 - ${title}</title>
<meta charset="utf-8">
</head>
<body>
<h1><a href="/">WEB</a></h1>
<ol>
```

```
<li><a href="/?id=HTML">HTML</a></li>
<li><a href="/?id=CSS">CSS</a></li>
<li><a href="/?id=JavaScript">JavaScript</a></li>
</ol>
<h2>${title}</h2>
<p>Test</p>
</body>
</html> ` ;
  res.send(template);
}) ;
app.get('/favicon.ico', (req, res) => res.writeHead(404));
app.listen(3000, () => console.log('Example app listening on port 3000')) ;
```

[work] /로 요청할 때 title이 undefined 가 안 나오도록 수정 하시오.

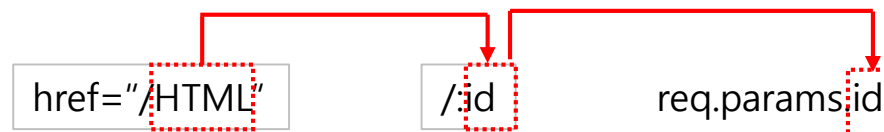
### 3. Semantic URL 사용 : ?가 붙은 **querystring**이 아닌 매개변수 개념으로 값을 서버에 전달

```
const express = require('express');
const app = express();
// var urlm = require('url');

app.get('/:id', (req, res) => {
  // var _url = req.url;
  var id = req.params.id;
  title = 'welcome';
  // var queryData = urlm.parse(_url, true).query
  console.log(id);
  var title = id;

  var template = `
<!doctype html>
<html>
<head>
<title>WEB1 - ${title}</title>
<meta charset="utf-8">
</head>
<body>
```

```
<h1><a href="/">WEB</a></h1>
<ol>
<li><a href="/HTML">HTML</a></li>
<li><a href="/CSS">CSS</a></li>
<li><a href="/JavaScript">JavaScript</a></li>
</ol>
<h2>${title}</h2>
<p>Test</p>
</body>
</html> `;
  res.send(template);
}) ;
app.get('/favicon.ico', (req, res) => res.writeHead(404));
app.listen(3000, () => console.log('Example app listening on port3000')) ;
```



[work] / 요청이 정상적으로 나오도록 코드를 수정하시오.



## 1. template 언어 사용 - ejs 설치

```
03> npm install ejs --save
```

## 2. template 언어 문법

표기(ejs 태그)	의미
<% js 문법 %>	흐름 제어문
<%= %>	변수 값
<%- %>	변수 내용을 태그로 인식
<%- include(view의 상대주소) %>	다른 view파일을 불러 옴.

## ejs 사용 ( ③ template 언어 사용)

### 3. ejs : main.js가 있는 폴더에 views 폴더 생성 후 ejs 파일을 views에 저장

main.js

```
const express = require('express');
const app = express();
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');

app.get('/', (req, res) => {
  var context = {title: 'welcome-1'};
  res.render('home', context, (err, html) => {
    res.end(html) })
})

app.get('/:id', (req, res) => {
  var id = req.params.id;

  var context = {title: id};
  res.render('home', context, (err, html) => {
    res.end(html) })
});

app.get('/favicon.ico', (req, res) => res.writeHead(404));
app.listen(3000, () => console.log('Example app listening on port 3000')) ;
```

① ejs 엔진을 사용하기 위한 코드

②

home.ejs

```
<!doctype html>
<html>
<head>
  <title>WEB1 - <%=title%></title>
  <meta charset="utf-8">
</head>
<body>
  <h1><a href="/">WEB</a></h1>
  <ol>
    <li><a href="/HTML">HTML</a></li>
    <li><a href="/CSS">CSS</a></li>
    <li><a href="/JS">JavaScript</a></li>
  </ol>
  <h2><%=title%></h2>
  <p>Test</p>
</body>
</html>
```

② res.render(view [, locals] [, callback]) : locals 위치에 있는 자료를 view에 넘겨주고 view를 html로 만들어서 클라이언트에 보내는 함수

### 3. ejs : main.js가 있는 폴더에 **views** 폴더 생성 후 **ejs** 파일을 **views**에 저장

- ① app.set(key, value) : set은 settings라는 json에 필드를 추가.  
console.log(app) 하면 다음과 같이 출력

```
settings: {  
  'x-powered-by': true,  
  etag: 'weak',  
  'etag fn': [Function: generateETag],  
  env: 'development',  
  'query parser': 'extended',  
  'query parser fn': [Function: parseExtendedQueryString],  
  'subdomain offset': 2,  
  'trust proxy': false,  
  'trust proxy fn': [Function: trustNone],  
  view: [Function: View],  
  views: 'C:\\Users\\owner\\nodejs\\202402\\WEEK02\\views',  
  'jsonp callback name': 'callback',  
  'view engine': 'ejs',  
}
```

- ② res.render(view [, locals] [, callback]) : locals 위치에 있는 자료를 view에 넘겨주고  
view를 html로 만들어서 클라이언트에 보내는 함수

## ejs 사용 ( ③ template 언어 사용)

### 4. ejس 작동 메커니즘

① node 프로그램에서 html로 변수의 값을 전달하므로 변수의 이름과 개수가 일치해야 함.

#### main.js

```
app.get('/:id',(req, res)=>{
  var id = req.params.id;
  var context = {title:id};
  res.render('home', context, (err,html)=>{
    res.end(html)  } )
});

app.get('/favicon.ico',(req, res)=>res.writeHead(404));
app.listen(3000, () => console.log('Example app listening on port 3000')) ;
```

① node에서 title과 ejس에서 title이 일치  
id가 HTML 이라고 가정  
context 객체의 title 멤버변수에 HTML 저장

변수명: 값

#### home.ejs

```
<!doctype html>
<html>
<head>
<title>WEB1 - <%=title%></title>
<meta charset="utf-8">
</head>
<body>
<h1><a href="/">WEB</a></h1>
<ol>
<li><a href="/HTML">HTML</a></li>
<li><a href="/CSS">CSS</a></li>
<li><a
href="/JavaScript">JavaScript</a></li>
</ol>
<h2><%=title%></h2>
<p>Test</p>
</body>
</html>
```

## ejs 사용 ( ③ template 언어 사용)

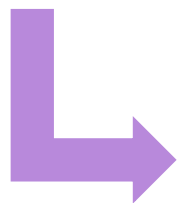
### 4. ejس 작동 메커니즘

② render 함수가 context 객체의 멤버 변수와 같은 home.ejs의 변수에 값을 넘겨주어 순수한 html을 생성

main.js

```
app.get('/:id',(req, res)=>{
  var id = req.params.id;
  var context = {title:id};

  res.render('home', context,
    (err,html)=>{
      res.end(html)
    })
});
```



```
var context = {title : 'HTML'};
```

home.ejs

```
<!doctype html>
<html>
<head>
<title>WEB1 - <%=title%></title>
<meta charset="utf-8">
</head>
<body>
<h1><a href="/">WEB</a></h1>
<ol>
<li><a href="/HTML">HTML</a></li>
<li><a href="/CSS">CSS</a></li>
<li><a
href="/JavaScript">JavaScript</a></li>
</ol>
<h2><%=title%></h2>
<p>Test</p>
</body>
</html>
```

순수한 HTML 파일 생성


```
<!doctype html>
<html>
<head>
<title>WEB1 - HTML</title>
<meta charset="utf-8">
</head>
<body>
<h1><a href="/">WEB</a></h1>
<ol>
<li><a href="/HTML">HTML</a></li>
<li><a href="/CSS">CSS</a></li>
<li><a
href="/JavaScript">JavaScript</a>
</li>
</ol>
<h2>HTML</h2>
<p>Test</p>
</body>
</html>
```

※ rendering : ejs 파일에 변수들의 값이나 프로그램들을 모두 컴파일 하여 완전한 html 문서로 만드는 작업.

※ rendering을 하는 메소드가 render() 메소드 이다.

③ render의 세번째 인자인 콜백함수에 ②번 단계에서 생성된 html 문서를 두번째 인자로 넘겨 주고 콜백함수에서는 response 객체의 end 메소드를 호출하여 HTML 문서를 browser에서 전송한다.

```
app.get('/:id',(req, res)=>{  
  var id = req.params.id;  
  var context = {title:id};  
  
  res.render('home', context, (err,html)=>{  
    res.end(html)  
  })  
});
```



The diagram illustrates the data flow in the provided code. A red dotted arrow originates from the 'html' parameter in the callback function of `res.render()` and points to the `html` argument of `res.end(html)`. Another red dotted arrow points from the text 'HTML' to the `html` argument of `res.end(html)`, indicating that the rendered HTML string is passed to the `end` method for transmission.

