# Express – 3

안녕하세요. Hokey Dokey입니다. 오늘의 할 일을 하기 전 spa와 mpa에 대해서 간단하게 집고 넘어가야 할 부분이 존재하기 때문에 템플릿 문서를 익히기 전에 두개의 차이점에 대해서 먼저 공부하도록 하겠습니다.

Mpa는 multiple page application의 약자로 클라이언트에서 어떤 페이지를 요청을 하게 되면 서버에서는 해당 페이지를 담은 문서를 전달하여 기존과는 전혀 다른 문서가 도착하는 application을 의미합니다. 그럼 Spa는 뭘까요 ? Spa는 single page application으로 같은 도메인내 페이지 이동을 할 때 axios같은 비동기 통신을 이용하여, 요청하는 데이터를 받아온 다음 dom 을 조작하여 페이지의 레이아웃 혹은 디자인을 변경하여, 페이지의 변화를 주는 application을 의미합니다.

각 application의 장,단점은 다음과 같습니다. 각 형태의 단점은 당연히 반대 application의 장점을 반대로 돌려놓은 것 입니다. 그렇기에 요즘은 axios같은 비동기 통신을 활용해서 mpa에서도 spa의 형태를 지원하는 hybrid application이 많이 사용되고 있습니다. 예를 들어 naver café페이지와 같이 하단의 탭을 누르면 페이지의 다운로드 없이 데이터만 받아와 page의 layout과 data 가 변경되지요.

전통적인 각 application의 형태에서 node.js는 spa에서 데이터만 주고 받는 서비스를 주로 제공하고, mpa에서는 데이터와 page를 동시에 전달하는 서비스를 제공합니다. 그렇다면 우리가 만들고 있는 hybrid형태는 어떻게 페이지를 전달해주어야 할까요 ? 정리해보면 3개의 시나리오가 존재합니다. Pu

먼저 static document를 제공하는 형태의 아주 아주 오래된 방식의 page제공 방식입니다. Html파일은 정적 문서이기 때문에, 서버에 저장되어 있는 html을 전달하게 되면 저장되어 있는 그대로 데이터와 레이아웃을 제공하게 됩니다. 데이터변화가 빈번히 존재하는 형태에서는 사용될 수가 없는 형태입니다. 주로 전통적인 인물 소개 블로그나, 회사 소개 사이트에서 사용된 형태입니다.

그 다음은 html과 css파일을 전송하고, 문서가 로드 완료되는 시점에 데이터를 전달받아 dom의 변화로 해당 데이터를 배치하는 형태의 방식이 존재합니다. 이런 경우 page로딩은 data가 없어서 빠를지 몰라도, data를 받고 dom의 변화를 끌어내는 동안 보이는 정보가 한정적이니 페이지는 매우 어색하고 두번의 통신과 dom 변화까지 겹쳐 느린 페이지 로드를 보여줍니다.

마지막으로는 dynamic document를 이용하는 겁니다. 2번째 방법의 문제점은 페이지를 다운로드 받은 즉시 데이터가 보여지지 않기 때문에 화면이 어색하다는 것인데, 이 dynamic document 즉 template문서라는 것을 활용해 서버에서는 이 템플릿에 들어있는 디자인 , 레이아웃에 따라 데이터를 배치해주고, html형태로 만들어 응답하는 방식입니다. 이런 방식을 사용하게 되면 페이지 전환 요청 시 한번의 페이지 전환은 존재하지만, 응답 받은 페이지는 랜딩 즉시 추가 데이터를 요청 할 필요 없이 데이터가 배치되어 페이지를 이용가능하게 되는 겁니다.

## Pug

Pug 문서는 dynamic문서를 만들기 위한 template document 입니다. Pug와 같은 템플릿 문서는 ejs가 있으며, pug 문서는 html과 비슷하게 생겼고, 태그를 닫는 것이 없어, 사용만 편해진다면 더욱 빠르게 문서를 제공할 수 있는 것이 특징입니다. Pug문서의 문법적 특징은 다음과 같습니다.

1. 열고 닫는 태그가 없다.
2. 들여쓰기로 태그 안과 밖의 text와 태그를 인식한다.
3. Html태그의 인라인 속성은 ()을 붙여 소괄호 내부에서 작성한다.
4. Id는 #id class는 .class를 붙여 사용하면 된다. (이 경우 자동으로 div로 인식)
5. 여러 줄인 경우 | 를 붙여 사용하면 된다.
6. 내부에 자바스크립트를 사용할 수 있다.
7. 내부에 반복문을 사용 할 수 있다.

이제 우리가 만들 페이지를 간단하게 보도록 하겠습니다. 화면 중앙에 글을 쓸수 있도록 배치하고 아래로 회원들의 모든 글들을 볼 수 있도록 만들었습니다. 더 복잡한 기능을 넣게 되면 시간이 많이 소모 됨으로 아주 간단한 page만 제공하도록 하겠습니다.

앉아있는, 테이블, 화면, 모니터이(가) 표시된 사진

자동 생성된 설명

우리가 만들 페이지 html코드 입니다. Pug로 먼저 레이아웃을 짜게 되면 익숙치가 않기 때문에 좀 더 직관적인 html로 먼저 구성하고 pug로 구성 하는 것이 시간 절약에 도움 되는 것 같습니다.

Css또한 다음과 같이 구성하고 pug page 를 살펴보도록 하겠습니다.

스크린샷, 앉아있는, 화면, 모니터이(가) 표시된 사진

자동 생성된 설명

(Views/main.pug)

Pug는 html 태그와 1:1로 대응되는 구조로 작성되어 있으니 doc 명령어로 html구조를 먼저 만들어주고 보면 태그는 우리가 알고 있는 형태로 작성되나, inline으로 작성한 속성들이 ()안에서 정의되는 것을 알게 됩니다.

Div는 class혹은 id만 입력하게 되면 div로 인식하게 됩니다. 태그에 소괄호를 열어 id 속성과 class를 작성할 수 있지만, 태그 옆에 #혹은 .을 적는 것으로 태그에 class와 id를 입력할 수 있습니다 .이제 제대로 작동하는지 확인하기 위해 /main router를 만들어 주도록 하겠습니다.

/router/main.js

var express = require('express');

var router = express.Router();

/\* GET home page. \*/

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

res.render('main')

});

module.exports = router;

res.render메서드에 첫번째 인자로 문자열을 작성하게 되면 해당 스트링에 맞는 템플릿 파일을 찾아 응답으로 전달하게 됩니다.

이제 pug를 사용하기 위해 npm install pug -s로 pug를 express에서 사용 할 수 있도록 합니다. App.js의 상단부로 가서 jade를 pug로 고쳐주시길 바랍니다. Jade는 pug템플릿의 옛 이름입니다.

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

var mainRouter = require('./routes/main');

…

app.use('/main',mainRouter);

이제 localhost:3000/main으로 접속해보면 우리가 만든 템플릿 파일을 볼 수 있습니다.

## Cookie

현재 저희 페이지는 개나 소나 main페이지로 접속이 가능합니다. 그러나 우리가 원하는 바는 로그인 과정을 거친 검증된 회원들만 main페이지로 접속을 하게 하고 싶습니다. 이 방법을 우리는 cookie라는 작은 데이터를 이용하여 가능하게 끔 할 예정입니다.

Cookie는 작은 데이터입니다. 쿠키는 서버에서 만들어 응답의 header에 넣고, 클라이언트는 서버에 요청할 때 마다 cookie를 전달하여 서버는 이 쿠키가 유효한 쿠키인지 확인 하고, 유효의 여부에 따라 올바른 응답을 합니다.

로그인 이후 main page를 요청하는 시나리오는 다음과 같습니다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

3번째 방법은 세션이라는 것을 사용하는 방법입니다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

서버에 객체를 하나 만든 다음 로그인시 객체에 클라이언트의 ip명으로 property 객체를 만들고 요청자에 대한 id 를 작성하여 추후 요청이 들어왔을 경우 session에서 요청자의 ip주소에 해당하는 프로퍼티가 있는지를 확인한 후 검증된 사용자인지를 확인합니다. 이 방법은 서버에서 로그인 정보까지 보관 하기 때문에 사용자가 증가함에 따라 메모리 부담이 증가하게 됩니다.

우리가 사용할 방법은 1번 시나리오로 cookie를 담아 main으로 요청 합니다. 이후 cookie가 유효한지 확인하고, 데이터와 페이지를 제공하는 형식입니다.

## Login 만들기

데이터 베이스에 login이라는 procedure를 생성합니다.

CREATE DEFINER=`root`@`localhost` PROCEDURE `login`(in id varchar(45), in password varchar(45))

BEGIN

select count(\*) as isValid from account where account.id = id AND account.pw = password;

END

그리고 POST /service/login 라우터를 테스트용도로만 만들어줍시다.

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

mysql.query(`call jasmin\_express.login('${req.body.id}','${req.body.pw}')`,(err,result)=>{

if(err){

res.status(403).render('error');

return;

}

if(result[0][0].isValid){

res.send("성공");

}else{

res.send("비밀번호와 아이디를 확인해주세요");

}

})

})

그리고 login.html을 수정하여 id와 password를 보낼 수 있도록 각 input에 id를 부여 하고 axios cdn을 연결합니다.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<link rel="stylesheet" href="http://localhost:3000/main/login.css">

<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

</head>

<body>

<div class="main-container">

<div class="login-box">

<h1 class="login-h1">SNS</h1>

<div class="login-input-container">

<div class="login-input-box">

<input id="login-id" class="input" type="text" placeholder="ID를 입력해주세요.">

</div>

<div class="login-input-box">

<input id="login-pw" class="input" type="password" placeholder="Password를 입력해주세요.">

</div>

</div>

<div class="login-tool-container login-input-container">

<button id="login-button" class="login-tool-button login-input-box login-button input">로그인</button>

<a href="http://localhost:3000/service/join"><button class="login-tool-button login-input-box input">회원가입</button></a>

</div>

</div>

</div>

</body>

<script src="http://localhost:3000/main/login.js"></script>

</html>

마지막에 스크립트를 연결하여 버튼을 누르면 id와 pw가 서버로 전송 될 수 있도록 만듭니다.

/main/login.js

(

function (){document.getElementById('login-button').addEventListener('click',()=>{

const id = document.getElementById("login-id").value;

const pw = document.getElementById("login-pw").value;

if(!id){

alert("id를 입력해주세요");

return;

}

if(!pw){

alert("pw를 입력해주세요");

return;

}

axios.post('http://localhost:3000/service/login',{

id,

pw

}).then((res)=>{

alert(`${res.data}`)

})

})}

)();

이제 회원가입한 아이디와 비밀번호가 유효한지 확인 할 수 있습니다.

## Cookie

이제 cookie를 한번 넘겨 보도록 하겠습니다. Res객체에는 cookie라는 메서드가 존재하는데 이 쿠키의 메서드의 파라메터는 첫번째로 쿠키 네임, 쿠키 value, 쿠키 옵션입니다. 서버는 이 쿠키를 전달하고 인증된 인원에 한하여 특정 path의 접근을 제한하고 싶은 경우 요청에서 cookie가 있는지를 확인하는데, 이때 쿠키 네임이 사용됩니다. 서버에서 전달한 쿠키가 존재하는 경우 해당 옵션을 확인하여 해당 쿠키가 만료가 되었는지 https 프로토콜에 사용되는 쿠키인지 확인 합니다. 이런 저런 확인을 하여 만료되지 않은 쿠키라면 접속을 허가 합니다. Cookie value의 경우에는 다양한 경로로 사용되는데, 만약 get 메서드로 /service/user\_info 와 같은 특정 페이지를 요청하게 되면 body를 사용할 수 없어 url에 요청하는 사용자의 id를 담을 때도 있습니다. (/service/user\_info?id=”id”) 이런 경우 id가 외부에 노출되어 보안이 약해지는 문제도 존재하고, url이 엄청 복잡해집니다. 그렇지만 cookie를 사용하면 id부분을 제거하고 해당 라우터에서 쿠키를 확인하여 해당 쿠키의 밸류 부분에서 id를 추출하여 로그인 한 유저를 특정 할 수 있습니다.

쿠키의 옵션은 다음과 같습니다.

먼저 max age 와 expires는 쿠키의 만료 기간을 설정하는 옵션입니다. 만약 쿠키의 max age 와 expires 가 존재 하지 않다면 브라우저는 쿠키를 삭제해버립니다.

Max age는 쿠키가 저장된 시점부터 종료되기 시작하는 시간을 정의하며 밀리 세컨드로 정의되어 있어, 주로 짧은 주기를 가진 쿠키에 사용됩니다. 예를 들어 우리가 핸드폰 인증을 받기 위해 버튼을 눌러 3분동안 인증을 거쳐야하는 인증 요청에 사용됩니다.

Expires 또한 max age처럼 쿠키의 만료 시간을 정의하는데, Numb 타입이 아닌 string타입으로 만료되는 시점의 GMT time을 의미합니다. 로그인 유지 버튼에 사용됩니다.

Path는 설정한 경로나 그 하위 경로에서만 쿠키 정보를 볼 수 있게 합니다. 예를 들어 /service 로그인을 옵션으로 주었다면 / 혹은 /service/login에서만 사용할 수 있게 합니다. 옵션을 주지 않았다면 / 이 기본 값으로 모든 경로에서 사용 가능합니다.

Signed옵션은 cookie가 서버에서 암호화되서 내려온 것을 의미합니다.

이제 간단하게 로그인에서 cookie를 클라이언트에게 보내보고 사용해보도록 하겠습니다.

if(result[0][0].isValid){

res.cookie('express',`${req.body.id}`,{

maxAge : 10000

}).send("성공");

}else{

res.send("비밀번호와 아이디를 확인해주세요");

}

})

로그인 후 크롬의 콘솔창을 열어 Application의 storage 탭의 cookies로 이동해보면 우리가 보낸 쿠키가 하나 존재 하는 것을 알 수 있습니다.

스크린샷, 앉아있는, 플레이어, 모니터이(가) 표시된 사진

자동 생성된 설명

이를 이용하여, 간단하게 /main router의 토큰 검증 미들웨어를 사용하여, 토큰이 발행되었는 경우에만 접속을 할 수 있도록 해보자 .

/routes/middleware/token.js 생성

module.exports = function (req,res,next){

if(req.cookies.express){

next();

}else{

next(new Error("비정상 적인 경로입니다."))

}

}

/routes/main변경

var express = require('express');

var router = express.Router();

var isValidAccess = require('./middleware/cookie');

*/\** GET home page. *\*/*

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

res.render('main')

});

module.exports = router;

이제 main 으로 요청을 하면 에러 페이지에 막히게 됩니다. 빠르게 로그인 후 접속 해보면 페이지가 렌더 됩니다 . 이제 마무리로 login시 바로 main으로 redirect되게 바꾸겠습니다.