Express -4

오늘은 글쓰기와 글 불러오기를 끝으로 Express강의를 마치려고 합니다. 현재 수정 삭제 보안등을 신경 쓰시려면 현재 많은 것을 변경해야 합니다. 모든 것을 염두하고 멋진 문서를 만들려면 많은 시간이 필요하기에 이번 강의는 혼자서 페이지를 구상하며 시나리오를 그려보고 현재 더 필요한 것은 강의가 끝난 후 외부 라이브러리를 사용하거나 알고리즘을 개발하며 더 멋진 app을 만들어보길 바랍니다.

Moment

이번에는 maxAge가 아닌 Expires를 사용할 예정인데 터미널에서 node 를 키신후 현재 시간을 확인 하기위해 new Date객체를 새로 만들어 보면 시간이 조금 다르게 설정된 것을 알 수 있습니다. 이는 현재의 node date객체가 한국으로 설정되어 있지 않기 때문인데요. 이를 해결하는 방법은 외부 라이브러리 moment를 사용하면 매우 쉽게 해결 할 수 있습니다.

Express 디렉토리에서 npm install -s moment를 해주시길 바랍니다. 그런 다음 해당 라우터에서 moment 객체를 부릅니다. 사실 여기까지 하면 왠만하면 서버시간은 정상적으로 돌아오지만 좀 더 확실하게 하기 위해 time-zone을 설정 하도록 하겠습니다. Npm install moment-timezone해주시고

Moment에 타임존 세팅을 하도록 하겠습니다.

const moment = require('moment');

require('moment-timezone');

moment.tz.setDefault('Asia/Seoul');

console.log(moment())

하니 잘 되는 것을 확인 할 수 있습니다.

Moment객체는 Date객체보다 훨씬 편한 기능들을 제공 합니다. 예를 들어 6시간을 date객체에 추가 시킨 시간을 적용시키려면 다음과 같이 적용해야 하지만

var tDate = new Date('2018-07-10 12:30')

tDate.setHours(tDate.getHours()+6)

moment객체에서는 moment().add(6,‘hours’) 처럼 누구나 알기 쉬운 방법으로 계산 할 수 있는 기능과 여러 기본 포맷이 존재하여 편리하게 시간에 관한 작업을 할 수 있습니다.

1시간 짜리 Expires를 만들기 위해 필요한 포맷은 GMT시간을 표기한

('ddd, DD MMM YYYY HH:mm:ss')+' GMT';

입니다. 이를 예를 들면 Thu, 30 Jul 2020 02:27:45 GMT 입니다.

이제 라우터에서 expires 시간을 만들도록 하겠습니다.

const expires = moment().add(1,'hours').format('ddd, DD MMM YYYY HH:mm:ss')+' GMT';

그리고 쿠키 옵션에 maxAge대신 Expires를 넣어 작동시켜보도록 하겠습니다.

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

Expires : expires

}).send("성공")

이제 정상적으로 쿠키가 보내지는 것을 볼 수 있습니다. Expires옵션이 존재하면 영구적인 쿠키로 보고, 서버에서 cookie의 Expires 확인하고, 시나리오를 달리 할 수 있습니다. 만약 1시간 짜리 쿠키를 자동으로 삭제하고 싶다면 1000 \* 60 \* 60 의 계산으로 옵션을 주면 됩니다. 그러나 이 Expires옵션을 사용하여 cookie를 받아오게 된다면 우리가 set한 cookie의 옵션은 request당시 같이 오지 않습니다. 그렇기에 이런 경우 session을 사용하여 쿠키에서는 session의 key값을 저장하고 요청 시 쿠키 속 key값을 이용하여 session이 가지고 있어야 하기 때문에 서버 입장에서는 용량 부담이 존재합니다. 또한 로그인 이후 로그아웃을 하지 않은 경우 파일 혹은 데이터가 그대로 남아있기 때문에 서버에서는 장시간 접속을 하지 않은 session들을 확인하여, 삭제하여야 하는 부담도 존재합니다. 이런 문제들을 해결하기 위해 jwt 토큰이라는 것이 존재합니다. 오늘 강의 마지막에 cookie와 session의 문제를 jwt 토큰에 대한 간단한 사용법을 통하여 해결 해보도록 하겠습니다.

## 로그인 유지

현재 login을 할 수 있는 도메인/ 과 main을 서로 움직일 수 있는 것이 매우 부자연스럽습니다. 그러므로 간단하게 로그인 유지를 할 수 있도록 index페이지에 미들웨어를 추가하여 로그인 유지를 할 수 있도록 하겠습니다.

/cookie 의 미들웨어를 조금 고쳐 보도록 하겠습니다.

module.exports = {

default : function (req,res,next){

if(req.cookies.express){

next();

}else{

next(new Error("비정상 적인 경로입니다."))

}

},

mainAccess : function (req,res,next){

if(req.cookies.express){

res.redirect('/main');

}else{

next();

}

}

}

/main 라우터 또한 변경해야 합니다.

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

이제 / 인덱스 라우터에서 mainAccess를 추가하도록 하겠습니다.

const mainAccess = require('./middleware/cookie').mainAccess;

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

이제 로그인 한 상태로 /에 접속하면 main으로 redirect되는 것을 확인 할 수 있습니다.

## 글 쓰기

회원가입을 진행하였다면 글 쓰기는 별거 없다는 것을 알게 될 것입니다.

Main.pug 파일에 상단에는 axios를 연결하고 마지막 줄에는 새로운 스크립트를 연결하도록 합니다.

/main.pug

script(src="http://localhost:3000/javascripts/main.js")

그리고 서버에서 public/javascript/main.js 파일을 생성해주시기 바랍니다. 그리고 새로운 버튼을 클릭하면 post할 수 있도록 이벤트 리스너를 연결합니다.

(

function(button,editor){

button.addEventListener('click',()=>{

if(editor.value.length < 10){

alert("글은 최소 10자리 이상되어야 합니다.");

return;

}

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

article : editor.value

})

})

}

)(document.getElementById("write-button"),document.getElementById("editor"))

이제 article router를 구성해보도록 하겠습니다.

var express = require('express');

var router = express.Router();

const isValidAccess = require('./middleware/cookie').default;

const mysql = require('../db/mysql').init();

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

mysql.query(`call new\_article('${req.body.article}','${req.cookies.express}')`,(err,result)=>{

if(err){

next(new Error("글 쓰기 생성에 실패하였습니다."));

return;

}

res.send(result[0]);

})

})

module.exports = router;

우리는 새로운 procedure new\_article을 만들고 파라메터로 글과 id를 받아와 글과 id 그리고 시간, 번호, 글 id 를 매겨 저장할 예정입니다. 아직 라우터를 연결하지 않고 article 테이블을 만들고 procedure를 만들도록 하겠습니다.

전자기기, 모니터, 컴퓨터, 남자이(가) 표시된 사진

자동 생성된 설명

테이블이 완성되었으면 새로운 procedure를 만들어 주도록 하겠습니다.

CREATE DEFINER=`root`@`localhost` PROCEDURE `new\_article`(article varchar (300), id varchar(45))

BEGIN

declare article\_id varchar(45);

set article\_id = uuid();

insert into article(article\_id,writer,article) value(article\_id,id,article);

select No,writer,article,date from article where article.article\_id = article\_id;

END

시험을 해보니 우리가 원하는데로 잘 나오는 듯 합니다. 그럼 이제 라우터를 연결해보도록 하겠습니다.

./app.js

var articleRouter = require('./routes/article');

app.use('/article',articleRouter);

이제 /main 화면에서 글을 작성하고 , 데이터베이스를 확인해보니 잘 작동하는 것을 알 수 있습니다.

그럼 이제 화면에서 해당 글을 볼 수 있도록 하겠습니다.

/main.js

const displayArticle = (articles) => {

const articleContainer = document.getElementById('article-container')

function makeArticleBox({No,date,writer,article}){

const articleBox = document.createElement('div');

articleBox.className = "articleBox";

const articleFlexContainer = document.createElement('div');

articleFlexContainer.className = "article-flex-container";

const headContainer = document.createElement('div');

headContainer.className = "head-flex-container";

const noDiv = document.createElement('div');

noDiv.className = "no";

noDiv.innerText = No;

const dateDiv = document.createElement('div');

dateDiv.className = 'date';

dateDiv.innerText = date;

const writerDiv = document.createElement('div');

writerDiv.className = "writer";

writerDiv.innerText = writer;

const articleDiv = document.createElement('div');

articleDiv.className = "article";

articleDiv.innerText = article;

headContainer.appendChild(noDiv);

headContainer.appendChild(dateDiv);

headContainer.appendChild(writerDiv);

articleFlexContainer.appendChild(headContainer);

articleFlexContainer.appendChild(articleDiv);

articleBox.appendChild(articleFlexContainer);

return articleBox;

}

const div = document.createElement('div');

articles.map((article)=>{

div.appendChild(makeArticleBox(article))

})

if(articleContainer.childNodes[0]){

articleContainer.insertBefore(div,articleContainer.childNodes[0]);

}else{

articleContainer.appendChild(div);

}

}

(

function(button,editor){

button.addEventListener('click',()=>{

if(editor.value.length < 10){

alert("글은 최소 10자리 이상되어야 합니다.");

return;

}

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

article : editor.value

}).then((res)=> displayArticle(res.data))

})

}

)(document.getElementById("write-button"),document.getElementById("editor"))

./main.css

\*{

margin: 0;

padding : 0;

background-origin: 0;

}

.head-bar{

width: 100%;

position: fixed;

height: 50px;

background-color: rgb(255, 194, 119);

}

.head-bar h1{

display: inline-block;

line-height: 50px;

margin-left: 15px;

}

.search-box{

display: inline-block;

position: absolute;

top: 25%;

right : 15px;

}

.main-container{

box-sizing: border-box;

padding-top: 50px;

padding-bottom: 100px;

min-height: 100vh;

height: 100%;

background: bisque;

*/\** padding-bottom: 100px;

min-height: 100vh;

height: 100%;

추후 추가됨

*\*/*

}

.editor-container , .articleBox{

box-sizing: border-box;

padding: 20px;

width: 400px;

height: 300px;

background: burlywood;

margin: 15px auto;

}

#editor{

width: 100%;

height: 80%;

}

#write-button{

width: 100%;

height: 40px;

}

*/\** //article box *\*/*

.article-flex-container{

display: flex;

height: 100%;

flex-direction: column;

}

.article-flex-container .article{

flex-grow: 1;

border: 1px solid gray;

border-top: none;

padding: 15px;

background-color: blanchedalmond;

}

.head-flex-container{

display: flex;

background-color: blanchedalmond;

text-align: center;

border: 1px solid gray;

}

.head-flex-container .no{

flex-grow: 2;

}

.head-flex-container .date{

flex-grow: 5;

}

.head-flex-container .writer{

flex-grow: 2;

}

이제 최초 main 페이지 render에 우리가 작성한 글들을 미리 넣어보는 작업을 시작하겠습니다.

Mysql로가서 get\_article이라는 모든 글을 가져오는 프로시듀어를 생성하도록 하겠습니다.

CREATE DEFINER = `root`@`localhost` PROCEDURE`get\_article`()

BEGIN

select No, writer, article, date from article order by date;

END

그리고 main페이지에서 다음과 같이 호출할 준비를 합니다.

var express = require('express');

const { json } = require('express');

const mysql = require('../db/mysql').init();

var router = express.Router();

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

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

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

mysql.query('call get\_article()', (err, result) => {

if(err){

next(new Error("문제가 발생하였습니다."));

return;

}

console.log(result)

res.render('main')

})

});

module.exports = router;

이제 pug템플릿을 고쳐볼까요 ? 익스프레스의 퍼그는 render메서드를 사용할 때 전달인자로 자바스크립트 데이터 포맷을 보내, 퍼그에서 해당 내용을 사용 할 수 있습니다. 또한 반복문을 지원하기도 합니다. 퍼그에서 사용 할 수 있는 반복문은 each문과 while문입니다. 우리가 mysql의 결과로 받는 result는 배열이기 때문에 굳이 변수를 하나 더 써가며 while문을 사용하지 않고 편하게 each문으로 반복문을 사용 할 수 있게 되었습니다.

articleContainer에 내용을 추가 하도록 하겠습니다. 우리가 만들었던 그 양식 그대로 사용하면 됩니다. 다만 each문 에서 요소를 사용하기 위해서는 들여쓰기를 해야하니 주의해주세요. 현재 result의 결과를 확인해보면 article은 객체입니다. Article에서 접근 연산자를 사용하기 위해서는 #{}을 사용하여 현재 작성하는 변수가 텍스트가 아니라 자바스크립트 문을 사용하는 것임을 알립니다.

#article-container

div

each article in articleList

.articleBox

.article-flex-container

.head-flex-container

.no #{article.No}

.date #{article.date}

.writer #{article.writer}

.article #{article.article}

이제 articleList로 해당 배열을 넘겨 보도록 하겠습니다.

res.render('main', {

articleList : result[0]

})

이제 확인을 해보니 gmt타임까지 나오는 모습을 볼 수 있는데 이를 좀 해결하기 위해 mysql의 쿼리를 좀 변경 하도록 하겠습니다.

select No,writer,article,date\_format(article.date,'%Y-%m-%d %T') as date

date\_format 함수를 사용하여 우리가 원하는 포맷대로 뽑아 올 수 있도록 합니다.

이제 main으로 접속 할 때 모든 인원들이 작성한 글들을 뽑아 올 수 있도록 만들었습니다.

## 글 불러오기

글 불러오기 까지 했으나 모든 article을 가져오게 되면 서버측 부담과 사용자 측면에서도 원치 않는 데이터까지 불러오게 되므로 10개씩 끊어서 가져오도록 하겠습니다.

/main.pug

button(id="get-article-button") 예전 글 불러오기

/main.css

#get-article-button{

display: block;

height: 25px;

width: 400px;

margin : 0 auto;

margin-top: 15px;

}

Mysql get\_article

그리고 mysql쿼리를 수정하여 10개씩 가져오도록 하겠습니다.

select No,writer,article,date\_format(article.date,'%Y-%m-%d %T') as date from article order by date DESC **limit 10;**

이제 버튼에 onclick옵션을 추가해주고 router를 조금 건들여 보겠습니다.

/main.pug

onclick=getArticleFunc

/main.js

res.render('main', {

articleList : result[0],

getArticleFunc : `getArticle(${result[0].length ? result[0][result[0].length -1].No : 0})`

})

이제 콘솔창을 확인해보니 버튼에 우리가 입력한 함수가 설정된 것을 볼 수 있습니다.

스크립트 파일에 getArticle을 추가 하도록 하겠습니다.

function getArticle(oldestNo){

axios.get(`http://localhost:3000/article?from=${oldestNo}&limit=10`).then((res)=>{

console.log(res)

})

}

?뒤에 있는 것은 쿼리라는 것으로 get메서드는 바디가 없기 때문에 추가 적인 정보를 담고 싶을때 이렇게 url에 부가적인 내용을 담습니다.

이제 get article 라우터를 추가 하도록 하겠습니다.

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

console.log(req.query);

res.send("ok")

})

이제 우리가 보낸 쿼리가 정상적으로 도착하는 것을 확인 할 수 있습니다. 다시 get\_article 프로시듀어를 변경 하도록 하겠습니다.

CREATE DEFINER = `root`@`localhost` PROCEDURE`get\_article`(in isMain boolean, in \_from int,in \_limit int)

BEGIN

if isMain

then select No, writer, article, date\_format(article.date, '%Y-%m-%d %T') as date from article order by date DESC limit 10;

else

select No, writer, article, date\_format(article.date, '%Y-%m-%d %T') as date from article where article.No < \_from order by date DESC limit \_limit;

END if;

END

IsMain인 경우는 접속 당시 최신정보 10개만 보내주면 되고 아닌 경우는 from 과 limit까지만 전송 하면 됩니다.

main router에 쿼리에 전달인자 isMain과 더미 정보를 추가합니다.

mysql.query('call get\_article(true,1,1)', (err, result) => {

if(err){

next(new Error("문제가 발생하였습니다."));

return;

}

res.render('main', {

articleList : result[0],

getArticleFunc : `getArticle(${result[0][result[0].length -1].No || 0})`

})

})

/article

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

mysql.query(`call get\_article(false,'${Number(req.query.from)}','${Number(req.query.limit)}')`, (err, result) => {

if(err){

next(new Error("문제가 발생하였습니다."));

return;

}

res.send(result[0])

})

})

Get article 라우터를 완료합니다.

Main.js 스크립트 파일 변경

함수의 첫번째 전달인자에 따라 앞에 넣을지 뒤에 넣을지 변경 하겠습니다.

const displayArticle = (isLatest,articles) => {

…

if(isLatest){

if(articleContainer.childNodes[0]){

articleContainer.insertBefore(div,articleContainer.childNodes[0]);

}else{

articleContainer.appendChild(div);

}

}else{

articleContainer.insertBefore(div,articleContainer.childNodes[articleContainer.childNodes.length -1]);

}

}

글쓰기 버튼 함수 전달인자 추가

button.addEventListener('click',()=>{

if(editor.value.length < 10){

alert("글은 최소 10자리 이상되어야 합니다.");

return;

}

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

article : editor.value

}).then((res)=> displayArticle(true,res.data))

})

getArticle 변경

init = null

last = false

function getArticle(oldestNo) {

if ((!init || oldestNo === init)) {

const button = document.getElementById('get-article-button');

axios.get(`http://localhost:3000/article?from=${oldestNo}&limit=${oldestNo > 10 ? 10 : oldestNo +1}`).then((res) => {

if (last) {

button.removeEventListener('click', getArticle)

}

if (res.data.length < 10) {

last = true;

}

displayArticle(false, res.data)

init = res.data[res.data.length - 1].No;

button.addEventListener('click', () => getArticle(res.data[res.data.length - 1].No))

})

}

}

여차저차 여기까지 twice calling 문제랑 부정적인 사이드 이펙트에 삽질을 하다가 이런 생각이 들더군요. 혹시 pug에서 script를 동적으로 만들 수는 없을까 … pug 공식 문서랑 이것 저것 찾아봤는데 마땅한 내용을 찾을 수 없더군요 . 그래서 혹시나 하고 script 태그를 head에 열고 dummy변수를 마음대로 입력했습니다. 그런데 되네요 ??? 그럼 oldestNo를 보관하기 어려워 inline으로 보관하고 이벤트 리스너를 지웠다 생성했다 하지 않아도 깔끔하게 이벤트를 설정 할 수 있게 되었습니다.

Main.js에 script를. 추가 하도록 하겠습니다.

Pug main.js

script.

(

function (getOldButton) {

let latestNo = #{\_latestNo};

getOldButton.addEventListener('click',getArticle = ()=>{

axios.get(`http://localhost:3000/article?from=${latestNo}&limit=${latestNo > 10 ? 10 : latestNo +1}`).then((res) => {

if (res.data.length < 10) {

getOldButton.removeEventListener('click',getArticle)

}

displayArticle(false, res.data);

latestNo = res.data.length ? res.data[res.data.length - 1].No : 0;

})

})

}

)(document.getElementById('get-article-button'));

이제 새로운 글을 받을 수 있도록 만드는 버튼 또한 만들어 보겠습니다.

기존 버튼의 id를 get-old-articles 로 변경하고 필요한 부분에 모두 적용하겠습니다.

Script 변경

script.

(

function (getOldButton) {

let latestNo = #{\_latestNo};

getOldButton.addEventListener('click',getArticle = ()=>{

axios.get(`http://localhost:3000/article?from=${latestNo}&limit=${latestNo > 10 ? 10 : latestNo +1}`).then((res) => {

if (res.data.length < 10) {

getOldButton.removeEventListener('click',getArticle)

}

displayArticle(false, res.data);

latestNo = res.data.length ? res.data[res.data.length - 1].No : 0;

})

})

}

)(document.getElementById('get-old-articles'));

Button 추가 및 id변경

button(class="get-article-button",id="get-latest-articles") 새로운 글 불러오기

#article-container

div

each article in articleList

.articleBox

.article-flex-container

.head-flex-container

.no #{article.No}

.date #{article.date}

.writer #{article.writer}

.article #{article.article}

button(class="get-article-button",id="get-old-articles") 예전 글 불러오기

main.css변경

.get-article-button{

display: block;

height: 25px;

width: 400px;

margin : 0 auto;

margin-top: 15px;

}

새로운 글을 받을때 중복된 게시물은 받으면 안되니까 현재 화면에 떠 있는 최신 게시글 전 까지의 최신 글을 불러와야만 합니다.

Main 라우터에서 최신정보의 번호를 pug 변수에 할당하도록 합니다.

res.render('main', {

articleList : result[0],

\_latestNo : result[0].length ? result[0][0].No : 0,

\_oldestNo : result[0].length ? result[0][result[0].length -1].No : 0,

getArticleFunc : `getArticle(latestNo)`

})

Main.pug 설정

Script 변수 추가

let latestNo = #{\_latestNo};

procedure 변경

CREATE DEFINER=`root`@`localhost` PROCEDURE `get\_article`(in isMain boolean, in \_from int, in \_to int, in \_limit int)

BEGIN

if isMain

then select No,writer,article,date\_format(article.date,'%Y-%m-%d %T') as date from article order by date DESC limit 10;

else

if \_to is null

then select No,writer,article,date\_format(article.date,'%Y-%m-%d %T') as date from article where article.No < \_from order by date DESC limit \_limit;

elseif \_from is null

then select No,writer,article,date\_format(article.date,'%Y-%m-%d %T') as date from article where article.No > \_from order by date DESC;

else

select No,writer,article,date\_format(article.date,'%Y-%m-%d %T') as date from article where article.No <= \_from and article.No > \_to order by date DESC;

END if;

END if;

END

Main.js 라우터 변경

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

mysql.query(`call get\_article(false,${!isNaN(req.query.from) ? Number(req.query.from) : 'null'},

${!isNaN(req.query.to) ? Number(req.query.to) : 'null'},

${!isNaN(req.query.limit) ? Number(req.query.limit) : 'null'})`, (err, result) => {

if(err){

console.log(err);

next(new Error("문제가 발생하였습니다."));

return;

}

res.send(result[0])

})

})

module.exports = router;

main.pug의 script변경 (새로운 글 쓰기 이벤트 연결)

script.

(

function (getOldButton,getLatestButton) {

let oldestNo = #{\_oldestNo};

let latestNo = #{\_latestNo};

getLatestButton.addEventListener('click',()=>{

axios.get(`http://localhost:3000/article?from=null&to=${latestNo}&limit=null`).then((res) => {

displayArticle(true, res.data);

latestNo = res.data.length ? res.data[0].No : latestNo;

})

})

getOldButton.addEventListener('click',getArticle = ()=>{

axios.get(`http://localhost:3000/article?from=${oldestNo}&to=null&limit=${oldestNo > 10 ? 10 : latestNo +1}`).then((res) => {

if (res.data.length < 10) {

getOldButton.removeEventListener('click',getArticle)

}

displayArticle(false, res.data);

oldestNo = res.data.length ? res.data[res.data.length - 1].No : 0;

})

})

}

)(document.getElementById('get-old-articles'),document.getElementById('get-latest-articles'));

이제 두개의 브라우저를 이용하여 한쪽에서 글을 써보고 다른 한쪽에서 새로운 글 읽기 버튼을 클릭 해보시면 위쪽에 새로운 글을 얻어 올 수 있습니다.

아직까지 문제가 있는데 누군가가 글을 쓰고 내가 글을 새로 쓴다면 싱크가 어긋나서 프론트 화면에 글 순서가 꼬이게 됩니다. 마지막으로 이를 해결 해봅시다.

우리가 할 전략은 post로 latestNo까지 함께 담아 보낸 후 다시 get article로 리다이렉트 하는 전략입니다. 굳이 그럴 필요 있냐고 하겠지만, restAPI를 명확히 하기 위해서는 필요 합니다 .

New\_article 프로시듀어 변경

CREATE DEFINER=`root`@`localhost` PROCEDURE `new\_article`(article varchar (300), id varchar(45))

BEGIN

declare article\_id varchar(45);

set article\_id = uuid();

insert into article(article\_id,writer,article) value(article\_id,id,article);

select No from article where article.article\_id = article\_id;

END

Main.pug script 변경 (latestNo 변수를 사용하기 위해 main.js에서 wrtiebutton evnet옮김)

script.

(

function (getOldButton,getLatestButton,writeButton,editor) {

let oldestNo = #{\_oldestNo};

let latestNo = #{\_latestNo};

writeButton.addEventListener('click',() => {

if (editor.value.length < 10) {

alert("글은 최소 10자리 이상되어야 합니다.");

return;

}

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

article: editor.value,

latestNo : latestNo

}).then((res) => {

displayArticle(true, res.data);

latestNo = res.data.length ? res.data[0].No : latestNo;

})

})

getLatestButton.addEventListener('click',()=>{

axios.get(`http://localhost:3000/article?from=null&to=${latestNo}&limit=null`).then((res) => {

displayArticle(true, res.data);

latestNo = res.data.length ? res.data[0].No : latestNo;

})

})

getOldButton.addEventListener('click',getArticle = ()=>{

axios.get(`http://localhost:3000/article?from=${oldestNo}&to=null&limit=${oldestNo > 10 ? 10 : latestNo +1}`).then((res) => {

if (res.data.length < 10) {

getOldButton.removeEventListener('click',getArticle)

}

displayArticle(false, res.data);

oldestNo = res.data.length ? res.data[res.data.length - 1].No : 0;

})

})

}

)(document.getElementById('get-old-articles'),

document.getElementById('get-latest-articles'),

document.getElementById("write-button"),

document.getElementById("editor"));

article router변경

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

mysql.query(`call new\_article('${req.body.article}','${req.cookies.express}')`, (err, result) => {

if (err) {

next(new Error("글 생성에 실패하였습니다."));

return;

}

res.redirect(`http://localhost:3000/article?from=${result[0][0].No}&to=${req.body.latestNo}&limit=null`)

})

})

이제 두개의 브라우저에서 확인 해보니 정상적으로 동작 하는 것을 확인 할 수 있습니다.