



Dev/nodejs

nodejs - express 프로젝트와 MySQL 연동

_overload | 2020. 3. 17. 22:06

kakao**AdFit**



nodejs - express 프레임워크를 이용한 웹 애플리케이션 구축 편에서 예고했던 Express 프레임워크에서의 MySQL 연동 방법에 대해 포스팅하겠습니다.

연동 방법에 대한 예제로 MySQL을 이용한 메모 조회, 추가, 수정, 삭제 기능을 포함하는 간단한 메모 웹 애플리케이션을 제작해 보도록 하겠습니다.

본 포스팅은 nodejs, npm 그리고 express-generator 패키지가 global 설치되어 있음을 전제로 하고 있습니다.

혹시 이 중에서 준비되어있지 않은 것이 있다면 이전 포스팅을 참고해 주시길 바랍니다.



nodejs - express 프레임워크를 이용한 웹 어플리케이션 구축

dev-overload.tistory.com

1. MySQL 설치

MySQL은 Linux, Windows가 서로 설치하는 방식이 다릅니다.

본 블로그에서 미리 각각 설치하는 방법에 대한 포스팅을 기록 해 두었으니 아래 링크를 참고해 주세요.

Windows에서 Mysql 설치

2020/03/12 - [Dev/etc] - Windows 환경에서의 MySql5.1~ 설치



Windows 환경에서의 MySql5.1~ 설치

dev-overload.tistory.com

Linux에서 Mysql 설치

2020/03/12 - [Dev/etc] - Linux 환경에서의 MySql5.1~ 설치와 root 계정 접속 세팅



Linux 환경에서의 MySql5.1~ 설치와 root 계정 접속 세팅

dev-overload.tistory.com

2. MySQL - 테스트용 데이터 삽입

Linux, Windows에서 각각 터미널, cmd를 열어 mysql에 접속 해 예제에 사용될 더미 데이터베이스, 테이블 데이터를 미리 삽입합니다.

아래에 미리 데이터를 준비 해 두었으니 이를 활용해 주세요.

```
$ mysql -u root -p
```

E: overload 구독하기



```
# 데이터베이스 생성
```

```
mysql> CREATE DATABASE NODE_DB;
```

```
# 사용할 데이터베이스 지정
```

```
mysql> USE NODE_DB;
```

```
# 테이블 생성
```

```
mysql> CREATE TABLE MEMOS(  
  -> ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  -> CONTENT VARCHAR(500) CHARACTER SET UTF8 NOT NULL,  
  -> CREATED_AT TIMESTAMP NULL DEFAULT NULL,  
  -> UPDATED_AT TIMESTAMP NULL DEFAULT NULL);
```

```
# 데이터 삽입
```

```
mysql> INSERT INTO MEMOS(CONTENT, CREATED_AT, UPDATED_AT)  
  -> VALUES('TEXT 1', NOW(), NOW()),  
  -> ('TEXT 2', NOW(), NOW()),  
  -> ('TEXT 3', NOW(), NOW()),  
  -> ('TEXT 4', NOW(), NOW());
```

```
# 데이터 조회
```

```
mysql> SELECT * FROM MEMOS;
```

ID	CONTENT	CREATED_AT	UPDATED_AT
1	TEXT 1	2020-03-17 20:34:17	2020-03-17 20:34:17
2	TEXT 2	2020-03-17 20:34:17	2020-03-17 20:34:17
3	TEXT 3	2020-03-17 20:34:17	2020-03-17 20:34:17
4	TEXT 4	2020-03-17 20:34:17	2020-03-17 20:34:17

```
# 변경 사항 적용
```

```
mysql> FLUSH PRIVILEGES;
```

```
# mysql 접속 종료
```

```
mysql> EXIT;
```

참고로 본 예제에서는 MySQL과 관련한 데이터베이스, 테이블, 테이블 컬럼 명은 모두 대문자로 통일하겠습니다.

3. Express 프로젝트 생성

Express 프로젝트는 html과 사용법이 유사한 ejs를 view를 사용해 **node-simple-memo**라는 이름으로 생성하겠습니다.

생성하고자 하는 경로에서 아래의 명령어로 프로젝트를 생성하고 npm 패키지를 인스톨하겠습니다.

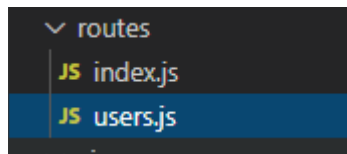
```
$ express --view=ejs node-simple-memo  
$ cd node-simple-memo  
$ npm install
```

패키지 인스톨이 완료되면 npm start 명령어로 서버를 실행 한 다음, localhost:3000으로 접속 해 프로젝트가 제대로 실행되는지 테스트도 잊지 않고 해 줍니다.

E: overload 구독하기



여기서 routes폴더에서 기본적으로 생성해 주는 파일 중 사용하지 않을 users.js파일은 지워줍니다.
예제 프로젝트의 메인 페이지는 index.js를 그대로 활용할 것이니 index.js는 남겨 주세요.



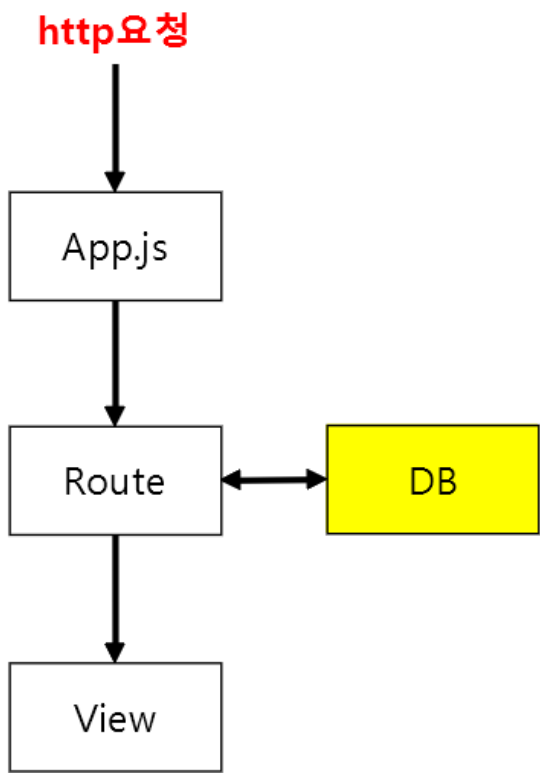
app.js에서 /users 라우터로 연결해주는 require 코드와 바인딩 코드도 지워 줍니다.

```
// 지워줄 코드

var usersRouter = require('./routes/users'); // 8번 줄에 있습니다.

app.use('/users', usersRouter); // 23번 줄에 있습니다.
```

4. DB 접속 스크립트 작성



스크립트 작성에 앞서, 본 예제에서 구성할 프로세스는 위와 같습니다.

http 요청을 받을 때 Route에서 DB 스크립트를 요청해 MySQL에 접속하고 필요한 데이터 조회, 삽입, 수정, 삭제 등을 수행한 후 결과와 데이터를 Route로 반환합니다.

E: overload 구독하기



Route는 반환받은 결과를 필요에 따라 View에 출력하는 형태로 구현할 예정입니다.

이를 위해 db 접속을 전담할 db.js를 생성해야 합니다.

db.js 파일을 생성하기에 앞서, MySQL과 통신할 node 모듈을 아래의 명령어로 설치합니다.

```
$ npm install --save mysql
```

이제 MySQL과 통신할 준비가 되었습니다.

프로젝트 루트 경로에 db.js 파일을 생성하고 아래의 코드를 작성합니다.

```
// db.js

const mysql = require('mysql');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'mysql 계정명',
  password: 'mysql 패스워드',
  port: 3306,
  database: 'NODE_DB'
});
```

const mysql = require('mysql'); 구문은 npm으로 설치한 mysql 모듈을 사용하겠다는 선언입니다.

user, password, port는 본인이 mysql 설치할 때 설정한 내용을 입력해 주세요.

database 파라미터는 2번 항목에서 생성한 데이터베이스 이름입니다.

5. Memo 조회 기능 구현

Memo 조회는 '/' 즉 localhost:3000으로 접속했을 때 렌더링 하는 페이지에서 출력해 주겠습니다.

routes/index.js 파일의 router.get('/', ... 코드 안에 아래의 코드를 작성합니다.

```
// routes/index.js

...

const db = require('../db'); // db 모듈 추가

/* GET home page. */
router.get('/', function(req, res, next) {
  db.getAllMemos((rows) =>{
    res.render('index', { rows: rows });
  });
});
```

```
});
```

```
module.exports = router;
```

E: overload 구독하기



const db = require('../db'); 객체는 4번 항목에서 작성한 db.js를 의미합니다.

db객체에서 getAllMemos 함수를 호출해 db에서 전체 메모를 조회합니다.

그리고 res.render('index', {rows: rows}); 함수를 실행하는데, 이 선언의 의미는 index.ejs 파일에 rows라는 데이터를 포함해 렌더링 하겠다는 의미입니다.

db 스크립트에는 getAllMemos라는 함수가 없기 때문에 db.js에 해당 함수를 선언해줍니다.
아래의 함수를 추가해 주세요.

```
// db.js

...

function getAllMemos(callback){
  connection.query(`SELECT * FROM MEMOS ORDER BY ID DESC`, (err, rows, fields) => {
    if(err) throw err;
    callback(rows);
  });
}

module.exports = {
  getAllMemos
}
```

여기서 module.exports을 선언해 주었는데, 외부에서 require를 통해 추가한 스크립트의 함수를 참조할 때 선언해주어야 해당 함수를 참조할 수 있습니다.

여기서 db 관련 코드는 callback 형태로 구현해 주어야 db를 조회한 후에 온전히 view로 데이터를 전달해 줄 수 있습니다.

이제 기존에 views/index.ejs 파일에 작성되어있던 내용을 지우고 라우터에서 전달해 주는 데이터를 받는 준비를 해야 합니다.

아래의 코드를 views/index.ejs에 작성해 주세요.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Simple memo</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1>Simple Memo</h1>
    <input type="button" value="new Memo" onclick="javascript:location.href='/newMemo'">
    <table border='1'>
      <tr>
        <th>Memo</th>
```

<th>Date</th>
<th>Edit</th>
</tr>
<% rows.forEach((row, index)=>{ %>
 <tr>
 <td><%=row.CONTENT%></td>
 <td><%=row.UPDATED_AT%></td>
 <td>
 <input type="button" onclick="javascript:location.href='/updateMemo?id=<%=row.ID%>' "
value="Edit">
 <input type="button" onclick="javascript:location.href='/deleteMemo?id=<%=row.ID%>' "
value="Del">
 </td>
 </tr>
<% }) %>
</table>
</body>
</html>

E: overload 구독하기

index.ejs 에 표시할 데이터는 content, update_at로 합니다.

여기서, 일반적인 html 파일에서는 볼 수 없는 <%= %>, <% %>형태의 코드가 보이는데, 이를 특수 태그라고 합니다.

특수 태그의 종류는 여러 가지가 있지만, 본 포스팅에서는 2가지만 사용하도록 하겠습니다.

위 두 가지 특수 태그의 차이는 아래와 같습니다.

특수 태그	기능
<% %>	javascript 코드 제어, 실행 (출력 없음)
<%= %>	javascript 변수 값 출력

특수 태그에서 foreach에서 참조하고 있는 rows는 routers/index.js에서 전달해준 rows 객체를 의미합니다.

rows에는 db에서 조회한 메모 데이터를 담고 있으며, rows를 순회하며 각 row를 table 형태로 출력해 주는 형태로 작성합니다.

위 ejs 파일 코드에서 각 input 태그의 onclick 속성에서 location.href를 통해 주소 요청을 하고 있습니다.

앞으로 주고받을 라우트 요청에 대해 미리 선언해 두었습니다.

요청 명에 대한 정리는 아래와 같습니다.

라우트 요청	기능
/newMemo	새 메모 작성 페이지 이동
/updateMemo	기존 메모 수정 페이지 이동 (파라미터 id를 받는다.)
/deleteMemo	메모 삭제 요청 (파라미터 id를 받는다.)

routes/index.js에 선언해 줄 내용이니 기억해 주세요.

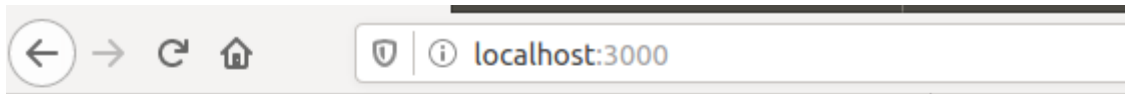
E: overload 구독하기



잘 선언되었는지 확인하기 위해 서버를 재시작해서 테스트해봅니다.

서버가 살아 있다면 Ctrl + C를 통해 중단하고 npm start로 재시작합니다.

브라우저에서 localhost:3000으로 접속해 봅니다.



Simple Memo

new Memo

Memo	Date	Edit	
TEXT 4	Wed Mar 18 2020 15:53:18 GMT+0900 (KST)	Edit	Del
TEXT 3	Wed Mar 18 2020 15:53:18 GMT+0900 (KST)	Edit	Del
TEXT 2	Wed Mar 18 2020 15:53:18 GMT+0900 (KST)	Edit	Del
TEXT 1	Wed Mar 18 2020 15:53:18 GMT+0900 (KST)	Edit	Del

위와 같은 화면이 출력되면 정상적으로 작성된 것입니다.

출력해주는 데이터는 2번 항목에서 MEMOS 테이블을 생성할 때 넣어준 데이터입니다.

그런데 GMT +0900 (KST)라는 문구까지 같이 출력되어 불필요하게 컬럼 너비가 넓어진 느낌이 듭니다.

GMT +0900 (KST)가 의미하는 것은, 세계 표준 시에서 한국 표준시는 +9시간이다 라는 의미입니다.

여기서 년/월/일/시/분/초 제외한 정보는 출력하지 않도록 db.js connection 객체에 내용 하나를 추가합니다.

```
// db.js

const mysql = require('mysql');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'root',
  port: 3306,
  database: 'NODE_DB',
  dateStrings: 'date' // 추가하는 내용
});
```


이제 서버를 재시작하고 적용되었는지 확인합니다.

E: overload 구독하기



Simple Memo

new Memo		
Memo	Date	Edit
TEXT 4	2020-03-17 13:36:38	Edit Del
TEXT 3	2020-03-17 13:36:38	Edit Del
TEXT 2	2020-03-17 13:36:38	Edit Del
TEXT 1	2020-03-17 13:36:38	Edit Del

년/월/일/시/분/초를 제외한 정보가 없어진 것을 확인할 수 있습니다.

6. 신규 메모 작성 페이지 구현

index.ejs 파일 내 신규 메모 작성 버튼은 아래와 같습니다.

```
<input type="button" value="new Memo" onclick="javascript:location.href='/newMemo'">
```

location.href를 통해 /newMemo를 요청하고 있습니다.

이제 routes/index.js에서 이 /newMemo 요청을 받을 코드를 선언해 주겠습니다.

아래의 코드를 작성해 주세요.

```
// routes/index.js

...

router.get('/newMemo', function(req, res, next){
  res.render('newMemo');
});

module.exports = router;
```

res.render를 통해 newMemo.ejs를 렌더링 합니다.

E: overload 구독하기



신규 메모 추가 페이지 요청 시에는 특별히 필요한 데이터가 없기 때문에 전달하는 파라미터는 지정하지 않습니다.

이제 newMemo.ejs를 작성하겠습니다. 아래의 코드를 작성 해 주세요.

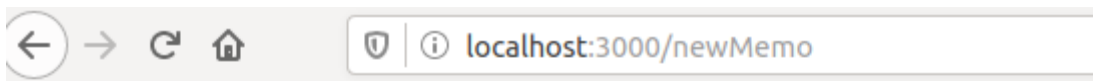
```
<!DOCTYPE html>
<html lang='utf-8'>
  <meta charset="utf-8">
  <head>
    <title>new Memo Add</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1>new Memo Add</h1>
    <form method="POST" action="/store" >
      <textarea name="content"></textarea>
      <input type="submit" value="submit">
      <input type="button" value="back" onclick="javascript:location.href='/'">
    </form>
  </body>
</html>
```

위 코드는 신규 메모 등록을 위한 작성 페이지입니다.

작성 후, submit 버튼을 통해 데이터를 /store로 post 방식 요청을 보내고 있습니다.

그리고 작성을 취소할 때는 다시 메인 페이지로 이동하도록 back 버튼의 onclick 이벤트로 javascript:location.href='/' 선언도 추가 해 두었습니다.

메인 페이지에서 new Memo 버튼을 눌러 방금 작성한 페이지로 이동되는지 확인합니다.



new Memo Add

submitback

출력하는 화면과 주소 표시줄을 보니 정상적으로 작성되었음을 확인했습니다.

이제 submit 버튼을 눌러 db에 신규 메모가 추가되도록 처리해야 합니다.

하지만 그전에 입력받은 데이터가 올바른 데이터인지 검증해줄 검증 모듈을 추가하고 간단한 검증 코드를 구성해야 합니다.



7. 신규 메모 등록 및 데이터 유효성 검사 처리

express 프레임워크에서 사용되는 유효성 검증을 위한 node 모듈은 여러 가지가 있지만, 본 예제에서는 express-validator라는 모듈을 사용하도록 하겠습니다.

터미널 및 cmd에서 아래의 명령어로 모듈을 설치합니다.

```
$ npm install --save express-validator
```

이제 routes/index.js에 설치한 모듈을 선언해 주고 /store 라우팅 요청을 받을 코드를 선언해 줍니다.

```
// routes/index.js

...
const {check, validationResult} = require('express-validator');

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

...

router.post('/store', [check('content').isByteLength({min:1, max:500})], function(req, res, next) {
  let errs = validationResult(req);
  if(errs['errors'].length > 0){ //화면에 에러 출력하기 위함
    res.render('newMemo', {errs:errs['errors']});
  }else{
    let param = JSON.parse(JSON.stringify(req.body));
    db.insertMemo(param['content'], () =>{
      res.redirect('/');
    });
  }
});

...

```

express-validator 모듈에서 check와 validationResult 함수를 사용합니다.

check 함수는 유효성 검사할 데이터를 특정하고 조건을 걸 수 있고, validationResult는 에러가 발생하면 해당 에러 내용을 반환해 주는 역할을 합니다.

check('content').isByteLength({min:1, max:500}); 는 content의 문자열의 바이트 길이가 1 이상 500 이하 인지를 검사합니다.

바이트 길이를 500 이하로 지정하는 이유는 테이블을 생성할 때 content 칼럼의 데이터 타입을 varchar(500)으로 했기 때문에 500바이트를 넘으면 데이터가 들어가지 않고 에러를 뱉기 때문에 이러한 장치를 마련해 주었습니다.

참고로 영문 한 글자는 1바이트, 한글 한 글자는 2바이트입니다.

유효성 검사에서 걸릴 경우, validationResult 함수에서 반환해 주는 데이터는 아래와 같습니다.

```
Result {
  formatter: [Function: formatter],
  errors: [
    {
      value: '~ 입력한 데이터 ~',
      msg: 'Invalid value',
      param: 'content',
      location: 'body'
    }
  ]
}
```

E: overload 구독하기



유효성 검사에서 걸리면 newMemo.ejs를 다시 렌더링 해 위 반환 데이터에서 msg 요소의 내용을 출력해 줍니다. 그럼 newMemo.ejs에서 errs 객체를 받을 수 있도록 내용을 약간 수정해줍니다.

```
<!DOCTYPE html>
<html lang='utf-8'>
  <meta charset="utf-8">
  <head>
    <title>new Memo Add</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1>new Memo Add</h1>
    <% if(typeof errs !== "undefined") { %>
      <% errs.forEach((err)=>{%>
        <h4 style="color: red;"><%=err.param%> : <%=err.msg%></h4>
      <% }) %>
    <% } %>
    <form method="POST" action="/store" >
      <textarea name="content"></textarea>
      <input type="submit" value="submit">
      <input type="button" value="back" onclick="javascript:location.href='/'">
    </form>
  </body>
</html>
```

특수 태그를 이용해 erres라는 데이터를 route에서 보내 주었을 경우, 화면에 붉은 텍스트로 경고 문구를 띄워주는 코드입니다.

이제, 유효성 검사를 통과했을 경우 db에 데이터를 넣어주는 코드를 작성합니다. db.js에 아래의 함수를 추가해 주세요.

```
// db.js
...

function insertMemo(content, callback){
  connection.query(`INSERT INTO MEMOS (CONTENT, CREATED_AT, UPDATED_AT) VALUES ('${content}',NOW(),NOW())`, (err, result) =>{
    if (err) throw err;
    callback();
  });
}
```

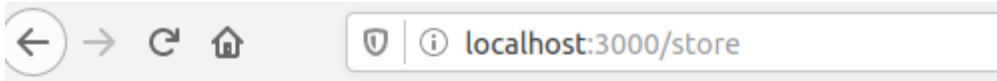
```
}  
  
module.exports = {  
  getAllMemos,  
  insertMemo  
}
```

E: overload 구독하기



module.exports에 insertMemo 함수를 등록하는 것도 잊지 말아 주세요.

이제 서버를 재시작하고 테스트를 해 봅시다.



new Memo Add

content : Invalid value

submit

back

데이터 유효성 검사에 적합하지 않으면 붉은 글씨를 출력해줍니다.

Simple Memo

new Memo

Memo	Date	Edit	
테스트 메모	2020-03-18 15:53:58	Edit	Del
TEXT 4	2020-03-18 15:53:18	Edit	Del
TEXT 3	2020-03-18 15:53:18	Edit	Del
TEXT 2	2020-03-18 15:53:18	Edit	Del
TEXT 1	2020-03-18 15:53:18	Edit	Del

유효성 검사에 적합하면 데이터를 삽입하고 메인 페이지로 이동합니다.

7. 기존 메모 수정 페이지 구현

기존 메모 수정 기능은 파라미터를 가지고 라우트 요청을 보내는 것과 DB에서 특정 ID를 조회하는 것 외에는 전반적으로 같습니다.

routes/index.js에 /updateMemo 요청을 받는 라우트 코드를 작성합니다.

```
// routes/index.js

....

router.get('/updateMemo', (req, res) =>{
  let id = req.query.id;

  db.getMemoById(id, (row) =>{
    if(typeof id === 'undefined' || row.length <= 0){
      res.status(404).json({error: 'undefined memo'});
    }else{
      res.render('updateMemo', {row: row[0]});
    }
  })
})
```

```
});  
});  
...
```

E: overload 구독하기



/updateMemo 라우트 요청을 보내는 코드를 다시 한번 상기하겠습니다.

```
<input type="button" onclick="javascript:location.href='/updateMemo?id=<%=row.ID%>'" value="Edit">
```

views/index.ejs에서 rows를 순회하면서 각 row의 id값을 파라미터로 /updateMemo 요청을 보내고 있습니다. 따라서 routes/index.js에서는 req.query.id를 통해 전달받은 파라미터 값 id를 변수 let id에 저장합니다.

그리고 db.getMemoById를 거쳐 해당 id값을 가지는 로우를 db에서 조회해서 updateMemo.ejs를 렌더링하고 있습니다.

여기서, 해당하는 id가 없거나 파라미터를 보내지 않는 요청 (올바르지 못한 요청)에 대해서는 404 스테이터를 갖고 에러 페이지를 반환합니다.

이는 브라우저 주소창에서 직접 접근할 때 발생할 오류에 대해 상정한 가벼운 조치입니다.

이제 db.js에 getMemoById 함수를 작성합니다.

```
// db.js  
  
...  
  
function getMemoById(id, callback){  
  connection.query(`SELECT * FROM MEMOS WHERE ID =${id}`, (err,row, fields) =>{  
    if(err) throw err;  
    callback(row);  
  });  
}  
  
module.exports = {  
  getAllMemos,  
  insertMemo,  
  getMemoById,  
}
```

이제 views 디렉토리 안에 updateMemo.ejs 파일을 생성하고 아래의 내용을 작성합니다.

```
<!DOCTYPE html>  
<html lang='utf-8'>  
  <meta charset="utf-8">  
  <head>  
    <title>update Memo Add</title>
```

```

<link rel='stylesheet' href='/stylesheets/style.css' />
</head>
<body>
  <h1>update Memo Add</h1>
  <% if(typeof errs !== "undefined") { %>
    <% errs.forEach((err)=>{%>
      <h4 style="color: red;"><%=err.param%> : <%=err.msg%></h4>
    <% }) %>
  <% } %>
  <form method="POST" action="/updateMemo" >
    <input type="hidden" name="id" value="<%=row.ID%>">
    <textarea name="content"><%=row.CONTENT%></textarea>
    <input type="submit" value="submit">
    <input type="button" value="back" onclick="javascript:location.href='/'">
  </form>
</body>
</html>

```

E: overload 구독하기



updateMemo.ejs에서도 수정할 데이터가 유효성에 맞지 않으면 다시 입력하도록 해야 하기 때문에
에러 문구를 출력해주는 코드를 미리 넣어 두었습니다.

form 태그 안에 id값은 input type=hidden으로 보내고 있는데, 이는 수정할 데이터의 id 값을 특정하기 위함입니다.
그리고 textarea에는 <%=row.CONTENT %>를 통해 기존에 작성한 메모 내용을 출력해 줍니다.

form 태그를 보면, 요청을 /updateMemo에 post 방식으로 보내고 있습니다.

이제, routes/index.js에 post 방식으로 /updateMemo 요청을 받는 코드를 작성합니다.

```

// routes/index.js

...

router.post('/updateMemo', [check('content').isLength({min:1, max:500})], (req, res) =>{
  let errs = validationResult(req);

  let param = JSON.parse(JSON.stringify(req.body));
  let id = param['id'];
  let content = param['content'];

  if(errs['errors'].length > 0){ //화면에 에러 출력하기 위함

    db.getMemoById(id, (row)=>{ //유효성 검사에 적합하지 않으면 정보를 다시 조회 후, updateMemo
      페이지를 다시 랜더링한다.
      res.render('updateMemo',{row:row[0], errs:errs['errors']});
    });
  }else{
    db.updateMemoById(id, content, () =>{
      res.redirect('/');
    });
  }
});

...

```


post 방식 /updateMemo에서는 받아야 할 데이터가 복수이므로 req.body를 json 형태로 가공하여 param 변수에 할당하고, param 안에 있는 id와 변경된 메모 내용을 갖고 있는 content를 별도의 변수에 저장합니다. E: overload 구독하기

error 처리에서 신규 메모 등록 때와는 다르게, 기존의 메모 내용을 가지고 렌더링을 해야 하기 때문에, 페이지를 렌더링 하기 전에 db.getMemoById를 호출 해 다시 해당 id의 데이터를 조회합니다.

만약 유효성 검사를 통과하면 db.updateMemoById를 호출 해 데이터를 업데이트하고 메인 페이지로 이동합니다.

db.js에 db.updateMemoById 함수를 작성합니다.

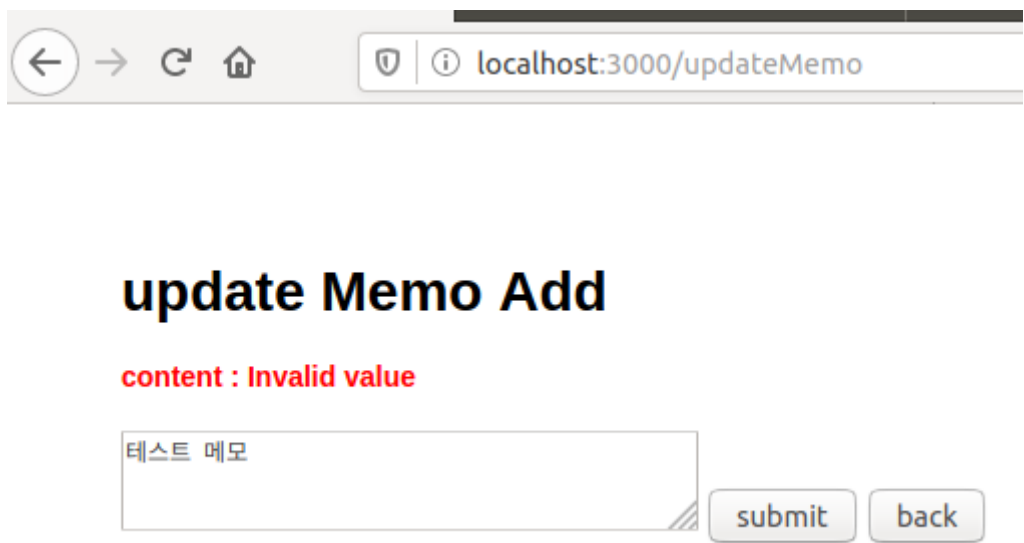
```
// db.js

...

function updateMemoById(id, content , callback){
  connection.query(`UPDATE MEMOS SET CONTENT='${content}', UPDATED_AT=NOW() WHERE ID=${id}`, (
err, result) => {
    if(err) throw err;
    callback();
  });
}

module.exports = {
  getAllMemos,
  insertMemo,
  getMemoById,
  updateMemoById,
}
```

이제, 서버를 다시 시작하고 메모 수정 기능이 동작하는지 테스트합니다.



유효성 검사를 통과하지 못하면 붉은색 문구와 함께, db에서 정보를 다시 조회 해 textarea에 기존 메모 내용을 출력하는 것을 확인했습니다.

Memo	Date	Edit
테스트 메모 (수정)	2020-03-18 16:48:57	Edit Del
TEXT 4	2020-03-18 15:53:18	Edit Del
TEXT 3	2020-03-18 15:53:18	Edit Del
TEXT 2	2020-03-18 15:53:18	Edit Del
TEXT 1	2020-03-18 15:53:18	Edit Del

유효성 검사를 통과했을 때의 처리도 정상적으로 완료되는 것을 확인할 수 있습니다.

8. 메모 삭제 기능 구현

메모 삭제는 매우 간단합니다.

우선, `views/index.ejs`에서 삭제 요청을 보내는 버튼을 다시 한번 살펴보겠습니다.

```
<input type="button" onclick="javascript:location.href='/deleteMemo?id=<%=row.ID%>'" value="Del">
```

파라미터 `id`값을 가지고 `/deleteMemo`라는 이름으로 라우트 요청을 보냅니다.

`routes/index.js`에 `/deleteMemo` 요청을 받는 라우트를 선언합니다.

```
// routes/index.js

...

router.get('/deleteMemo', (req, res) =>{
  let id = req.query.id;
  db.deleteMemoById(id, () =>{
    res.redirect('/');
  });
});
```

```
});
```

```
..
```

E: overload 구독하기



req.query.id를 통해 파라미터 id의 값을 let id에 할당합니다.

그리고 db.deleteMemoById를 통해 해당 id값을 갖는 로우를 삭제하고 메인 페이지로 이동합니다.

이제 db.js에 deleteMemoById 함수를 선언합니다.

```
// db.js

...

function deleteMemoById(id, callback){
  connection.query(`DELETE FROM MEMOS WHERE ID=${id}`, (err, result) =>{
    if(err) throw err;
    callback();
  });
}

module.exports = {
  getAllMemos,
  insertMemo,
  getMemoById,
  updateMemoById,
  deleteMemoById
}
```

마지막으로 데이터 삭제까지 정상 동작하는지 확인합니다.

해당 예제 프로젝트는 github에 등록 해 두었으니 필요하신 분들은 참고해 주세요.

```
$ git clone https://github.com/overload-dev/node-simple-memo.git
```

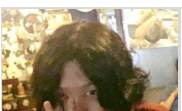
위 리파지토리를 받으면 제일 먼저 README에 작성된 내용을 mysql에 세팅하고 db.js의 connection의 user, password, port 내용을 본인의 환경에 맞게 세팅하셔야 합니다.

첨언하자면, 본 프로젝트의 내용은 Express 프레임워크의 전반적인 흐름에 대한 예시용으로 작성되어 각 상황에 대한 예외 처리가 매우 부실한 편입니다.

본 프로젝트를 받아서 이러한 부분을 어떻게 보완할지에 대한 연구가 더해진다면 Express 프레임워크를 익히는데 도움이 되지 않을까 싶습니다.

본 포스팅의 예시 프로젝트를 github에 따로 마련 해 두었습니다.

github.com/overload-dev/node-simple-memo



overload-dev/node-simple-memo



이상으로 nodejs - express 프레임워크에서의 mysql 연동 포스팅을 마치겠습니다.

kakaoAdFit

2

구독하기

카카오스토리

트위터

페이스북

'Dev > nodejs' 카테고리의 다른 글

nodejs - express 프로젝트와 MySQL 연동 (2)	2020.03.17
nodejs - express 프레임워크를 이용한 웹 어플리케이션 구축 (0)	2020.03.11

태그 #Express, #linux, #MYSQL, #Nodejs, #ubuntu, #windows

'Dev/nodejs' Related Articles



nodejs - express 프레임워크를 이용한 웹 어...



야조 | 2021.10.15 12:33

E: overload 구독하기



쓰기

현재 열심히 읽으면서 구조 파악하고 따라하고 있습니다..초보자라 이것저것 어렵네요 그래도 도움이 많이 되었어요
고맙습니다



_overload | 2021.10.15 13:49 | 신고

댓글주소 수정/삭제

도움이 되셨다니 다행입니다. :)

궁금한 점 있으시다면 언제든지 말씀 주세요. 힘 닿는 부분까지 같이 고민해보겠습니다 ㅎㅎ

이름

암호



Secret

여러분의 소중한 댓글을 입력해주세요.

댓글달기

<

1

...

38

39

40

41

42

43

44

45

46

...

48

>