Node.js

방명록 프로젝트 - 김근형 강사

구성

- DB 세팅

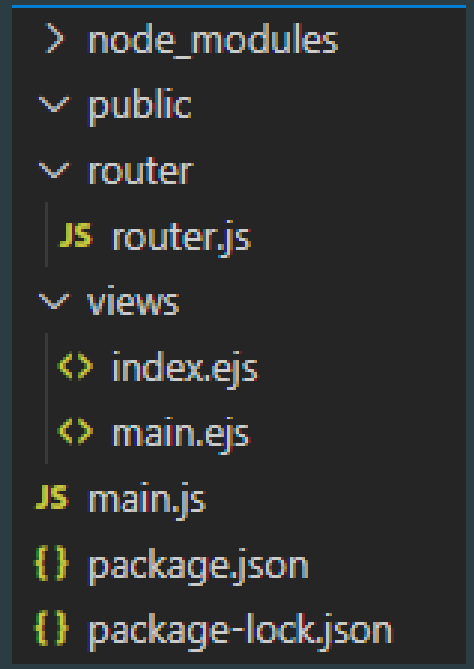
```
create database GuestBookDB default character set utf8 collate utf8_general_ci;

use GuestBookDB;

create table guestbook(
    idx int primary key auto_increment,
    guestbook_name varchar(50) not null,
    guestbook_content varchar(500) not null
);
```

구성

- 전체적인 파일 위치 및 package.json & main.js 구성




```json
{
    "name": "guestbook",
    "version": "0.0.0",
    "private": true,
    "dependencies": {
        "express": "*",
        "ejs": "*",
        "body-parser": "*",
        "cookie-parser": "*",
        "express-session": "*",
        "mysql2": "*"
    }
}
```

```js
var express = require('express');
var ejs = require("ejs");
var cookieParser = require("cookie-parser");
var session = require("express-session");

var app = express();

app.set("views",__dirname + "/views");
app.set("view engine", "ejs");
app.engine("ejs", ejs.renderFile);
app.use(cookieParser())
app.use(session({
    secret : "abcdefg", // 이 데이터를 이용해서 암호화를 하게 된다.
    resave : false,  // 세션의 정보를 다시 저장하는지 여부
    saveUninitialized : true  // 초기화 값을 저장할지 안할지 여부
}));

app.use(express.static("public"));

var router1 = require('./router/router')(app);

var server = app.listen(2000, function(){
    console.log('포트 2000번으로 서버 실행');
});
```

구성

- router.js 구성

```
var bodyParser = require("body-parser");
var urlencodedParser = bodyParser.urlencoded({extended:false});
var mysql = require("mysql2");

var conn_info = {
    host : "localhost",
    port : 3306,
    user : "root",
    password : "1234",
    database : "GuestBookDB"
}

module.exports = function(app){
    app.get("/", function(req,res){
        res.render("index.ejs");
    });

    app.post("/login", urlencodedParser, function(req,res){
        var user_name = req.body.user_name;
        //res.send(user_name);

        req.session.user_name = user_name;
        // 서버 main으로 재요청
        res.redirect("main");
    });
```

```
app.get("/main",function(req, res){
    var user_name = req.session.user_name;

    var conn = mysql.createConnection(conn_info);
    var sql = "select guestbook_name, guestbook_content "+
              "from guestbook order by idx desc";

    conn.query(sql, function(err, rows){
        var render_data = {
            name : user_name,
            rows : rows
        }

        res.render("main.ejs",render_data);
    });
});

app.post("/save_guestbook", urlencodedParser, function(req,res){
    var user_name = req.session.user_name;
    var content = req.body.content;

    var conn = mysql.createConnection(conn_info);
    var sql = "insert into guestbook(guestbook_name, "+
              "guestbook_content) values(? , ?)";
    var input_data = [user_name, content];

    conn.query(sql, input_data, function(err){
        conn.end()
        res.redirect("main");
    });
});
```

구성

- ejb 웹페이지 구성

index.ejb

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>로그인 페이지</h1>
    <form action="login" method="POST">
        <label>이름 : <input type="text" name="user_name"></label><br>
        <button type="submit">로그인</button>
    </form>
</body>
</html>
```

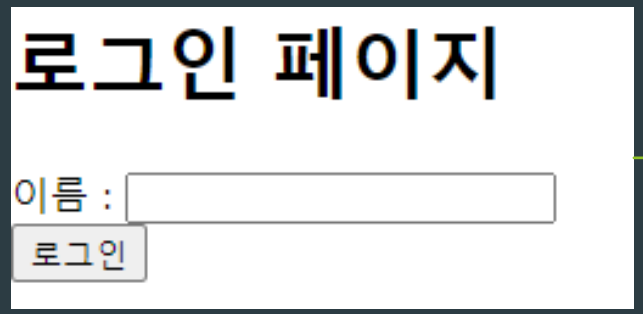
main.ejb

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1><%=name%> 님이 입장하셨습니다.</h1>
    <form action="save_guestbook" method="POST">
        <textarea name="content" style="width:300px; height:50px"></textarea>
        <button type="submit">저장</button>
    </form>

    <% for(var obj of rows){ %>
        <p>작성자 : <%=obj.guestbook_name %></p>
        <p>내용 : <%=obj.guestbook_content %></p>
        <hr/>
    <% }%>
</body>
</html>
```

구성

- 실행 결과


로그인 페이지
이름 :
로그인


김근형 님이 입장하셨습니다.
저장
작성자 : 김근형
내용 : 안녕하세요
작성자 : 김근형
내용 : ㅂㅁㄴㅇㅈ
작성자 : 김근형
내용 : ㅋㅌㅊㅋㅌㅊ
작성자 : 김근형
내용 : ㅂㅈㄷㅂㅈㄷ

redirect

- redirect
  - redirect 함수는 라우터 함수 내부에서 쓰이는 기능
  - 한 경로에서 다른 경로로 다시 get 방식으로 호출하려 할 경우 해당 기능을 쓴다.
  - 이전 기능을 재활용 하고자 할 경우 사용하는 경우가 많다.

```
app.post("/login", urlencodedParser, function(req,res){
    var user_name = req.body.user_name;
    //res.send(user_name);

    req.session.user_name = user_name;
    // 서버 main으로 재요청
    res.redirect("main");
});
```

```
app.get("/main",function(req, res){
    var user_name = req.session.user_name;

    var conn = mysql.createConnection(conn_info);
    var sql = "select guestbook_name, guestbook_content "+
              "from guestbook order by idx desc";

    conn.query(sql, function(err, rows){
        var render_data = {
            name : user_name,
            rows : rows
        }

        res.render("main.ejs",render_data);
    });
});
```

Error handling

- error handling
  - 웹페이지를 이용하다 에러가 생긴 경우가 종종 존재한다.
  - 웹 페이지에서는 관련 에러들을 처리해야 하며 그러기 위해서 에러에 관련된 처리들을 해주지 않으면 안되는데 이를 에러 핸들링이라 한다.
  - 에러 핸들링을 하기 위해서는 **status**라는 값이 존재하는데 **status** 는 숫자를 받으며 각 고유의 숫자는 사용자에게 보여질 요청에 대한 상태 값을 의미한다.
  - 아래와 같은 상태 코드들이 존재한다.

값	설명
100~199	정보성 상태 코드
200~299	성공 상태 코드
300~399	리다이렉션 상태 코드
400~499	클라이언트 에러 상태 코드
500~599	서버 에러 상태 코드

Error handling

- error handling - 예제

```
app.use(express.static("public"));

var router1 = require('./router/router')(app);

app.use( ( req , res, _ ) => {
    res.status(404).render('common/404.ejs');
});

app.use( (err, req, res,  _ ) => {
    console.log(err.toString());
    res.status(500).render('common/500.ejs',{message : err.message});
});

var server = app.listen(2000, function(){
    console.log('포트 2000번으로 서버 실행');
});
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>404 에러</h1>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>500에러</h1>
    <h3><%=message%></h3>
</body>
</html>
```

404 에러

templating

- 템플릿 활용
  - 웹페이지를 개발하다 보면 반복되는 부분이 발생하는 경우가 있다.
  - 이 경우 단순히 값이나 자바 스크립트로 처리하기에는 상당히 어려운 점이 있는데 이런 부분을 **ejs**에서 제공하는 **include** 라는 기능을 통해 해결이 가능하다.
  - **include** 는 다음 아래와 같이 활용이 가능하다.

```
<%-include('가져올 파일 경로')%>
```

templating

- 템플릿 활용 예제 -1

template1.ejs

```ejs
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <%-include('./inc/header.ejs')%>
    <h1>template1.ejs</h1>
</body>
</html>
```

./inc/header.ejs

```ejs
<h1>header.ejs</h1>
<h3>헤더에서 가져온 데이터</h3>
```

templating

- 템플릿 활용 예제 -2

router.js

```
app.get("/template2", (req, res) =>{

    var render_data = {
        hd : '이것은 헤더에 전달한 데이터입니다',
        content : '이것은 바디에 전달한 데이터입니다'
    }

    res.render("template2.ejs", render_data);
});
```

header2.ejs

```
<h1>header2.ejs</h1>
<p><%=hd%></p>
```

template2.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <%-include('./inc/header2.ejs')%>
    <h1>template2.ejs</h1>
    <%=content%>
</body>
</html>
```