

생활코딩

# Node.js 노드제이에스 프로그래밍

2024년 10월  
6주차



## 6주차 수업의 범위

---

- \* 입력 정보에 대한 보안
- \* 출력 정보에 대한 보안
- \* 두 테이블의 **join** : **author** 테이블과 **topic** 테이블
- \* 쿠키
- \* 로그인 화면

# 보안 이슈

- **querystring**에 경로가 포함된다면 상위 디렉토리까지 노출될 수 있음. → **path** 모듈로 경로를 지우고 파일명만 사용

특히 삭제할 때 주의 경로를 허락하면 원치 않는 경로의 파일을 지울 수 있음. 경로는 서버에서 책임 지도록

```
const express = require('express');
const app = express();
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');
var fs = require('fs');
var url = require('url');
var path = require('path');
app.get('/', (req, res) => {
  var _url = req.url;
  var queryData = url.parse(_url, true).query;
  console.log(queryData.id)
  // console.log(path.parse(queryData.id)); // 1.
  fs.readdir('./lib', function(error, filelist){
    console.log(filelist);
    fs.readFile(`lib/${queryData.id}`, 'utf8', function(error, description){ // 2. 이것을 주석처리 하고 1.을 실행
      // fs.readFile(`lib/${path.parse(queryData.id).base}`, 'utf8', function(error, description){ // 1.
        console.log(description)
        res.end(description)
      })
    })
  })
})

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

## 1) 사용자가 입력한 데이터를 조심해야 하는 이유

← → ↻ ⓘ localhost:3000/create

YouTube 지도

## WEB

- [CSS](#)
- [HTML](#)
- [JavaScript](#)
- [NodeJs-update](#)
- [노드제이에스](#)

<script>  
alert('merong');

제출

- ① 제목에 **XSS**
  - ② 내용에 다음과 같이 입력
- ```
<script>  
    alert('merong');  
</script>
```
- ③ 다음과 같은 결과
  - ④ **XSS** 클릭하면  
경고창

← → ↻ ⓘ localhost:3000

YouTube 지도

## WEB

- [CSS](#)
- [HTML](#)
- [JavaScript](#)
- [NodeJs-update](#)
- [XSS](#)
- [노드제이에스](#)

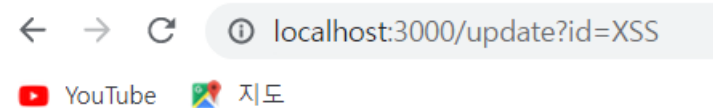
create update delete

## XSS

localhost:3000 내용:  
merong

확인

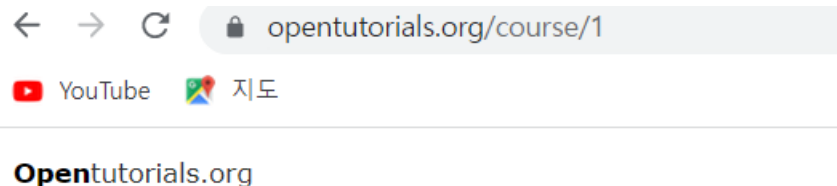
## 1) 사용자가 입력한 데이터를 조심해야 하는 이유

WEB

- [CSS](#)
- [HTML](#)
- [JavaScript](#)
- [NodeJs-update](#)
- [XSS](#)
- [노드제이에스](#)

[create update](#)

- ① **update** 화면에서 **alert**의 내용을 다음과 같이 수정  
`location.href = 'https://opentutorials.org/course/1';`
- ② **XSS**를 클릭할 때마다 다른 사이트로 넘어감.



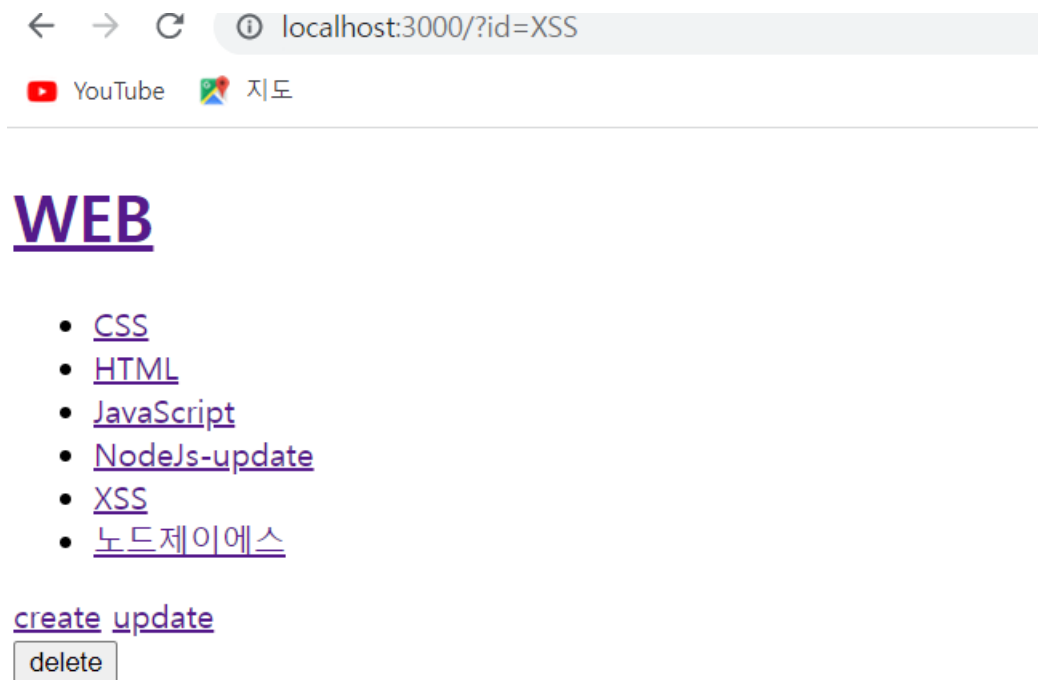
### 1) 사용자가 입력한 데이터를 조심해야 하는 이유

- 앞의 두 가지 경우 이외에도 사용자의 로그인 정보를 갈취하는 등 더 심각한 상황이 발생 가능.
- 사용자로부터 입력 받은 데이터를 바깥으로 출력할 때 그 데이터에서 발생할 수 있는 문제를 걸러내는 기능을 갖추고 있어야 함.
- 필터링 기능이 필요

① **<script>** 태그를 아예 지우는 방법

② **<script>** 태그를 그대로 노출되게 하는 방법

<를 &lt;로 >를 &gt;로



### XSS

```
<script> location.href='https://opentutorials.org/course/1'; </script>
```

## 2) 외부 모듈 사용 준비

- 이 번 절의 두 가지 목적

- ① 보안 : 사용자가 입력한 정보에 오염된 정보가 있다면 소독하는 방법
- ② **npm**을 통해 다른 사람이 만든 모듈을 사용하는 것

- 외부 모듈 설치 : **sanitize-html** 모듈 설치

- ① 작업 폴더에서 다음 명령어 실행

> **npm install -S sanitize-html**

- ② **node\_modules** 디렉토리 생성 - 이 안에 여러 모듈 디렉토리가 생성. **sanitize-html** 생성을 확인

- ③ **package.json** 안에 `"sanitize-html": "^2.7.1",`



## 3) 외부 모듈(sanitize-html) 사용하기

package.json

```
"dependencies": {  
  "ejs": "^3.1.9",  
  "express": "^4.18.2",  
  "mysql": "^2.18.1",  
  "sanitize-html": "^2.11.0"  
},
```

dependencies : 본 응용은 여기에 정의된 모듈에 의존한다.

topic.js

```
const db = require('./db');  
var qs = require('querystring');  
var sanitizeHtml = require('sanitize-html');
```

sanitize-html 모듈 : 허용되지 않은 태그를 삭제

사용법은 인터넷을 찾아보세요. !!!!

3) 외부 모듈(sanitize-html) 사용하기 - **Client**가 보낸 내용을 읽어오는 부분, **DB** 내용 또는 **Client**가 입력한 내용 출력할 때도 사용

topic.js

```
create_process : (req,res) => {
  var body = '';
  req.on('data', (data)=> {
    body = body + data;
  });
  req.on('end', () => {
    var post = qs.parse(body);
    sanitizedTitile = sanitizeHtml(post.title)
    sanitizedDescription = sanitizeHtml(post.description)
    db.query(
      `INSERT INTO topic (title, descrpt, created)
      VALUES(?, ?, NOW())`,
      [sanitizedTitile, sanitizedDescription], (error, result)=> {
        if(error) {
          throw error;
        }

        //res.writeHead(302, {Location: `/page/${result.insertId}`});
        res.redirect(`/page/${result.insertId}`)
        res.end();
      }
    );
  });
},
```

※ sanitize-html 적용이  
필요한 나머지 부분에도  
적용하시오.

**author** 테이블 추가  
**topic**과 연결  
**topic** 화면 수정

## 1. 테이블 설계

- topic 테이블의 author\_id와 author 테이블의 id값이 서로 같다. author\_id는 foreign key

```
CREATE TABLE `author` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `name` varchar(20) NOT NULL,  
  `profile` varchar(200) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
);
```

※ 두 테이블을 연결하여 질문을 하려면 JOIN문 필요

```
CREATE TABLE `topic` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `title` varchar(30) NOT NULL,  
  `description` text,  
  `created` datetime NOT NULL,  
  `author_id` int(11) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
);
```

## 2. 두 테이블 JOIN

- topic 테이블의 author\_id와 author 테이블의 id값이 서로 같다.

topic.id 가 ? 인 글의 내용과 그 글을 쓴 저자의 이름과 profile을 나타내고 싶다면 다음 sql 문 사용

select \* from **topic left join author** on **topic.author\_id = author.id** where topic.id = ?

topic

|   | id   | title      | description                          | created             | author_id |
|---|------|------------|--------------------------------------|---------------------|-----------|
| ▶ | 1    | MySQL      | MySQL is...                          | 2018-01-01 12:10:11 | 1         |
|   | 2    | Orade      | Orade is ...                         | 2018-01-03 13:01:10 | 1         |
|   | 3    | SQL Server | SQL Server is ...                    | 2018-01-20 11:01:10 | 2         |
|   | 4    | PostgreSQL | PostgreSQL is ...                    | 2018-01-23 01:03:03 | 3         |
|   | 5    | MongoDB    | MongoDB is ...                       | 2018-01-30 12:31:03 | 1         |
|   | 6    | Nodejs is  | I feel Nodejs is easier than Django. | 2022-09-20 20:59:51 | 1         |
| • | NULL | NULL       | NULL                                 | NULL                | NULL      |

author

|   | id ▲ | name   | profile                   |
|---|------|--------|---------------------------|
| ▶ | 1    | egoing | developer                 |
|   | 2    | duru   | database administrator    |
|   | 3    | taeho  | data scientist, developer |
| • | NULL | NULL   | NULL                      |

### 3. 상세보기 화면

- 글목록에서 글을 선택했을 때 상세 보기 페이지에 작성자를 함께 표시

← → ↺ ⓘ localhost:3000/page/1

## WEB

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [날씨](#)
7. [XSS](#)

[create](#) [update](#) [delete](#)

## MySQL

MySQL is Database Name.

by Gu Han-Seok

## 4. topic.js

- 글목록에서 글을 선택했을 때 상세 보기 페이지에 작성자를 함께 표시

```

page : (req,res) => {
  var id = req.params.pageId;
  db.query('SELECT * FROM topic', (error,topics)=>{
    if(error){
      throw error;
    }
    db.query( `SELECT * FROM topic LEFT JOIN author ON topic.author_id = author.id WHERE topic.id =
    ${id}`, (error2, topic)=>{
      if(error2){
        throw error2;
      }

      var m = `<a href="/create">create</a>&nbsp;&nbsp;&nbsp;
               <a href="/update/${id}">update</a>&nbsp;&nbsp;&nbsp;
               <a href="/delete/${id}" onclick='if(confirm("정말로 삭제하시겠습니까?")==false){ return false }'>
               delete</a>`
      var b = `<h2>${topic[0].title}</h2>
               <p>${topic[0].descript}</p>
               <p>by ${topic[0].name}</p>`

      var context = {list:topics,
                     menu:m,
                     body:b};
      res.app.render('home', context, (err,html)=>{
        res.end(html)  })
    })
  });
}

```

수정

레코드 하나만 select 되므로 topic[0]

추가

## 04 글 생성 수정 – create할 때 저자를 선택할 수 있는 기능 추가

### 1. 글 생성 화면

← → ↻ localhost:3000

#### Topic List

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [날씨](#)
7. [XSS](#)

[create](#)

Welcome

Node.js Start Page



← → ↻ localhost:3000/create

#### Topic Create

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [날씨](#)
7. [XSS](#)

[create](#)

Gu Han-Seok ▼

제출

※ 작업순서

- ① 콤보 박스에 저자목록이 뜰 수 있도록 코드 수정
- ② 저자를 선택할 수 있는 콤보 박스가 나오도록 화면 구성
- ③ 제출 버튼을 클릭하면 insert 시 topic의 author\_id 필드도 입력 되도록 코드 수정



## 04 글 생성 수정 – create할 때 저자를 선택할 수 있는 기능 추가

### 2. topic.js

#### ① 콤보 박스 만들기 html 문장

html 문장

```
<select name="author">
  <option value="1">egoing</option>
  <option value="1">duru</option>
</select>
```

js 코드

```
var i = 0;
var tag = '';
while( i < authors.length) {
  tag += `<option value=${authors[i].id}>${authors[i].name}</option>`;
  i++;
}
```

## 2. topic.js

## ② create 메소드 수정

```

create : (req,res) => {
  db.query(`SELECT * FROM topic`, (error,topics)=>{
    if(error){
      throw error;
    }
    db.query(`SELECT * FROM author`,(err, authors)=>{
      var i = 0;
      var tag = '';
      while(i<authors.length)
      {
        tag+= `<option value="${authors[i].id}">${authors[i].name}</option>`;
        i++;
      }

      var context = { list:topics,
        menu: `<a href="/create">create</a>`,
        body: `<form action="/create_process" method="post">
          <p><input type="text" name="title" placeholder="title"></p>
          <p><textarea name="description" placeholder="description"></textarea></p>
          <p><select name="author">
            ${tag}
          </select></p>
          <p><input type="submit"></p></form>`

        콤보박스 생성
        select 태그
      };

      req.app.render('home',context, (err, html) => {
        res.end(html);
      }); // render 종료
    }); // 두번째 query 메소드 괄호
  }); // 첫번째 query 메소드 괄호
},

```

저자 목록을 가져오는 sql 문장 추가

콤보박스를 채울 저자 자료

## 2. topic.js

③ create\_process 메소드 수정 : submit 버튼을 눌렀을 때 그 정보를 받아서 처리하는 부분을 수정

```
create_process : (req,res)=>{
  var body = '';
  req.on('data', (data)=>{
    body += data;
  });
  req.on('end',()=>{
    var post = qs.parse(body);
    var sanitizedTitle = sanitizeHtml(post.title);
    var sanitizedDescription = sanitizeHtml(post.description)
    var sanitizedAuthor = sanitizeHtml(post.author) //추가
    db.query(`insert into topic (title, descrpt, created,author_id) values(?,?,now(),?)`, //수정
      [sanitizedTitle, sanitizedDescription,sanitizedAuthor],(error, result)=>{ //수정
        if(error){
          throw error
        }
        // res.writeHead(302, {Location: `/page/${result.insertId}`});
        res.redirect(`/page/${result.insertId}`)
        res.end();
      }); //첫번째 query 종료
  })
},
```

## 1. update 화면

- /update 부분 수정 → update 메소드 수정

← → ↻ localhost:3000/update/1

### Topic Update

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [날씨](#)
7. [XSS](#)

[create](#) [update](#)

Gu Han-Seok ▼

Gu Han-Seok

Ha Bong-Soo

Sin You-Jeong

Jung Ha-Yoon

Lee Kyung-Chang

Park Ki-Soo

```

update : (req,res)=>{
  var id = req.params.pageId;
  db.query('select * from topic', (error, topics)=>{
    if(error){ throw error }
    db.query(`select * from topic where id = ?`,[id],(error2, topic)=>{
      if(error2){ throw error2 }
      db.query(`SELECT * FROM author`, (error3, authors) => {
        if(error3){ throw error3 }

        var i = 0;
        var tag = '';
        while ( i < authors.length) {
          var selected = '';
          if(authors[i].id === topic[0].author_id){ selected = 'selected'; }
          tag += `<option value="${authors[i].id}" ${selected}>${authors[i].name}</option>`;
          i++; }

        var m = `<a href="/create">create</a>&nbsp;&nbsp;&nbsp;<a href="/update/${topic[0].id}">update</a>`
        var b = `<form action="/update_process" method="post">
          <input type="hidden" name="id" value="${id}">
          <p><input type="text" name="title" placeholder="title" value="${topic[0].title}"></p>
          <p><textarea name="description" placeholder="description">${topic[0].descript}</textarea></p>
          <p><select name="author">
            ${tag}
          </select></p>
          <p><input type="submit"></p>
        </form>`

        var context = {list : topics,
          menu : m,
          body : b };
        res.render('home',context,(err,html)=>{
          res.end(html)
        }); //render 종료
      }); //세번째 query 종료
    }); //두번째 query 종료
  }); //첫번째 query 종료
}

```

**2. topic.js**

① update 메소드 수정

저자 목록이 보여야 하니까 author 테이블 select

콤보박스를 채울 저자 자료  
수정하려는 글의  
작성자가 콤보  
박스에 보여야  
함

콤보박스 생성  
select 태그

## 2. topic.js

### ② 저자 콤보 박스 목록 만들기

주의 : update 할 글의 저자가 디폴트로 나와야 함

```
<select name="author">  
  <option value="1">egoing</option>  
  <option value="1" selected>duru</option>  
</select>
```

## 2. topic.js

③ update\_process : 제출 버튼 클릭하면 update, author\_id도 변경할 수 있도록 코드 수정

```
update_process : (req,res)=>{
  var body = '';
  req.on('data', (data)=>{
    body += data;
  });
  req.on('end',()=>{
    var post = qs.parse(body);
    var sanitizedTitle = sanitizeHtml(post.title);
    var sanitizedDescription = sanitizeHtml(post.description)
    var sanitizedAuthor = sanitizeHtml(post.author) //추가
    db.query(`update topic set title = ?, descrpt = ?, author_id = ? where id = ?`,
      [sanitizedTitle, sanitizedDescription, sanitizedAuthor, post.id],(error, result)=>{
        if(error){ throw error
        }
        res.writeHead(302, {Location: `/page/${post.id}`}); // redirection
        res.end();
      }); //첫번째 query 종료  })  },
```

```
<form>
  <input type="hidden" name="id"> post.id
  <input type="text" name="title"> post.title
  <textarea name = "description"> post.description
  <select name="author"> post.author
  <input type="submit">
</form>
```

### 3. 정리

① submit 버튼을 클릭했을 때 수행하는 /update\_process 처리

The screenshot shows a web browser at localhost:3000/update?id=1. The page has a header with 'WEB' and a list of database links: MySQL, Oracle, SQL Server, PostgreSQL, MongoDB, and Nodejs.is. Below this is a 'create update' section with a text input containing 'MySQL', a description input containing 'MySQL is...', and a dropdown menu currently showing 'egoing'. A '제출' (Submit) button is at the bottom. A red dashed box highlights the 'egoing' dropdown menu.

This screenshot is identical to the previous one but includes red arrows pointing to specific elements: from the 'MySQL' text input to 'post.title', from the 'MySQL is...' description input to 'post.description', and from the 'taeho' dropdown menu to 'post.author'. The dropdown menu now shows 'taeho'. A red arrow points from the '제출' button to the text '클릭' (Click), which then points to the URL '/update\_process'.



**author** 추가 기능 구현

## 1. 저자 초기 화면 프로그램 - 화면구성



## 1. 저자 생성 프로그램 추가·화면구성

127.0.0.1:3000/author

Gmail 지도 번역 PYRASIS.COM: 가장...

## WEB TOPIC 테이블

[author](#)

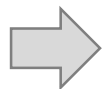
1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [React](#)
7. [지구온난화](#)

김서울	scientist	<a href="#">update</a>	<a href="#">delete</a>
이부산	columnist	<a href="#">update</a>	<a href="#">delete</a>

왕보현

연구자

저자생성 → 클릭



127.0.0.1:3000/author

Gmail 지도 번역 PYRASIS.COM: 가장...

## WEB TOPIC 테이블

[author](#)

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [React](#)
7. [지구온난화](#)

김서울	scientist	<a href="#">update</a>	<a href="#">delete</a>
이부산	columnist	<a href="#">update</a>	<a href="#">delete</a>
왕보현	연구자	<a href="#">update</a>	<a href="#">delete</a>

name

profile

저자생성

## 1. 저자 삭제 프로그램 추가·화면구성

← → ↻ 127.0.0.1:3000/author

Gmail 지도 번역 PYRASIS.COM: 가장...

## WEB TOPIC 테이블

[author](#)

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [React](#)
7. [지구온난화](#)

김서울	scientist	<a href="#">update</a>	<a href="#">delete</a>
이부산	columnist	<a href="#">update</a>	<a href="#">delete</a>
왕보현	연구자	<a href="#">update</a>	<a href="#">delete</a>

name

profile

저자생성



← → ↻ 127.0.0.1:3000/author

Gmail 지도 번역 PYRASIS.COM: 가장...

## WEB TOPIC 테이블

[author](#)

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [React](#)
7. [지구온난화](#)

김서울	scientist	<a href="#">update</a>	<a href="#">delete</a>
이부산	columnist	<a href="#">update</a>	<a href="#">delete</a>

name

profile

저자생성

## 1. 저자 갱신 프로그램 추가 - 화면구성

127.0.0.1:3000/author/update/1

Gmail 지도 번역 PYRASIS.COM: 가장...

## WEB TOPIC 테이블

[author](#)

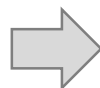
1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [React](#)
7. [지구온난화](#)

김서울	scientist	<a href="#">update</a>	<a href="#">delete</a>
이부산	columnist	<a href="#">update</a>	<a href="#">delete</a>

김서울

scientist

수정



127.0.0.1:3000/author/update/1

Gmail 지도 번역 PYRASIS.COM: 가장...

## WEB TOPIC 테이블

[author](#)

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [React](#)
7. [지구온난화](#)

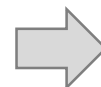
김서울	scientist	<a href="#">update</a>	<a href="#">delete</a>
이부산	columnist	<a href="#">update</a>	<a href="#">delete</a>

김서울

computer engineer

수정

클릭



127.0.0.1:3000/author

Gmail 지도 번역 PYRASIS.COM: 가장...

## WEB TOPIC 테이블

[author](#)

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [React](#)
7. [지구온난화](#)

김서울	computer engineer	<a href="#">update</a>	<a href="#">delete</a>
이부산	columnist	<a href="#">update</a>	<a href="#">delete</a>

name

profile

저자생성

## 2. 저자 생성 갱신 삭제 프로그램 추가

main.js에 다음 코드 추가 → 저자관련 외부모듈을 author.js로 작성

```
app.get('/author', (req, res) => {
  author.create(req, res);
})

app.post('/author/create_process', (req, res) => {
  author.create_process(req, res);
})

app.get('/author/update', (req, res) => {
  author.update(req, res);
})

app.post('/author/update_process', (req, res) => {
  author.update_process(req, res);
})

app.get('/author/delete', (req, res) => {
  author.delete_process(req, res);
})
```

1. '/'로 요청하면 topic 목록 위에 author 링크를 만들어야 함.

## WEB TOPIC 테이블

[author](#)

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [React](#)
7. [지구온난화](#)

[create](#)

## Welcome

Node.js Start Page

### ① home.ejs 수정

```
<body>
  <h1><a href="/">WEB TOPIC 테이블</a></h1>
  <a href="/author">author</a> 추가 또는 수정
```

## 2. author 클릭하면 요청되는 URL인 /author에 대한 처리

① main.js 파일에 /author에 대한 처리

```
app.get('/author', (req, res) => {
    author.create(req, res);
})
```

### WEB TOPIC 테이블

[author](#)

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [React](#)
7. [지구온난화](#)

김서울	scientist	<a href="#">update</a>	<a href="#">delete</a>
이부산	columnist	<a href="#">update</a>	<a href="#">delete</a>




```
<body>
  <h1><a href="/"><%=title%></a></h1>
  <a href="/author">author</a>
  <ol type="1">
    <% var i = 0
      while(i < list.length)
        { %>
          <li><a href="/page/<%=list[i].id%>"><%=list[i].title%></a></li>
        <% i += 1 } %>
    </ol>

    <%- menu%>
    <%- body%>

  </body>
```

**menu**에 저자 목록을 출력

**body**에 저자 입력 form 출력



## author.js 파일의 create 메소드 작성

```

create: (req, res) => {
  db.query('select * from topic', (err, topics) => {
    if(err){ throw err}
    db.query('select * from author', (err2, authors) => {
      if (err2) { throw err2 }
      var i = 0;
      var tag = '<table border="1" style="border-collapse: collapse;">'
      for ( i=0; i<authors.length; i++) {
        tag += `<tr><td>${authors[i].name}</td><td>${authors[i].profile}</td>
                  <td><a href="/author/update/${authors[i].id}">update</a></td>
                  <td><a href="/author/delete/${authors[i].id}" onclick=
                    'if(confirm("정말로 삭제하시겠습니까?")==false){return false}'>delete</a></td></tr>`
      }
      tag += '</table>'

      var b = `<form action="/author/create_process" method='post'>
                <p><input type='text' name='name' placeholder='name'></p>
                <p><input type='text' name='profile' placeholder='profile'></p>
                <p><input type='submit' value='저자생성'></p>
              </form>`

      var context = { list: topics,
                      menu: tag,
                      body: b }
      res.render('home', context, (err, html) => res.end(html))
    })
  })
},

```

김서울	scientist	<a href="#">update</a>	<a href="#">delete</a>
이부산	columnist	<a href="#">update</a>	<a href="#">delete</a>

name
profile
생성

```
app.post('/author/create_process', (req, res) => {
  author.create_process(req, res);
})
```

1. `/author/create_process` 처리를 위한 메소드 `create_process`를 `author.js` 파일에 추가 → `topic.js`의 `create_process` 메소드 이용, `querystring` 모듈 추가

```
var qs = require('querystring');
```

```
create_process : (req, res) => {
  var body = '';
  req.on('data', (data) => {
    body += data;
  });
  req.on('end', () => {
    var post = qs.parse(body);
    var sanitized_name = sanitizeHtml(post.name);
    var sanitized_profile = sanitizeHtml(post.profile);
    db.query(`insert into author (name, profile)
              values(?,?)`,
            [sanitized_name, sanitized_profile], (error, result) => {
              if (error) {
                throw error;
              }
              res.redirect(`/author`)
              res.end();
            });
    // 첫 번째 query 종료
  });
},
```

1. **home.ejs** 에서 다음 코드를 화면마다 적당한 제목을 보여 줄 수 있도록 **<%=title%>** 로 수정하고 필요한 부분을 모두 수정하시오.

```
<h1><a href="/">WEB TOPIC 테이블</a></h1>
```

쿠키

\* 쿠키의 목적 - 개인화 된 웹 페이지를 보여주는 것에 관심

- 1) 인터넷 쇼핑을 할 때 장바구니에 물건을 담으면 다음에 방문할 때도 장바구니에 그대로 물건이 담겨 있는 것.
- 2) 로그인해서 한번 인증하면 그 다음에 접속할 때는 인증할 필요 없이 웹 사이트를 이용할 수 있는 것.
- 3) 쿠키 도입

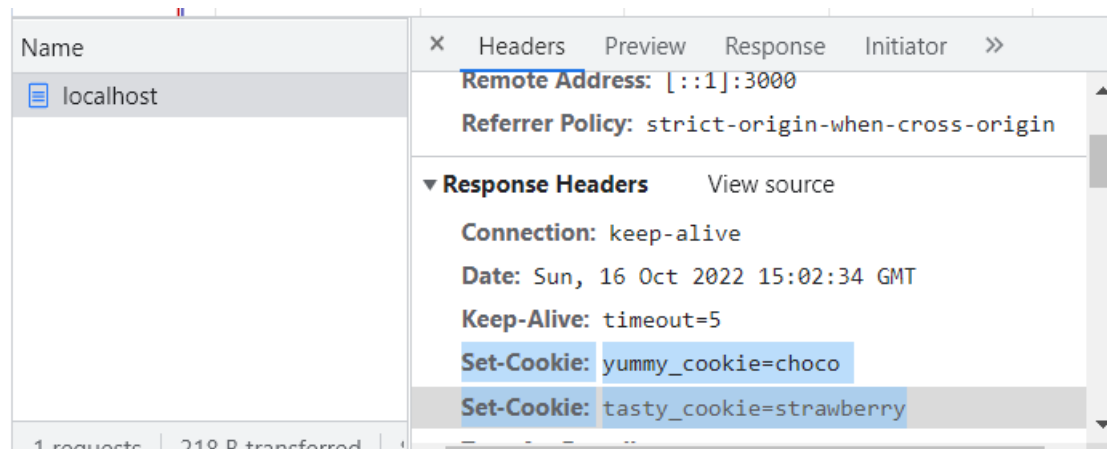
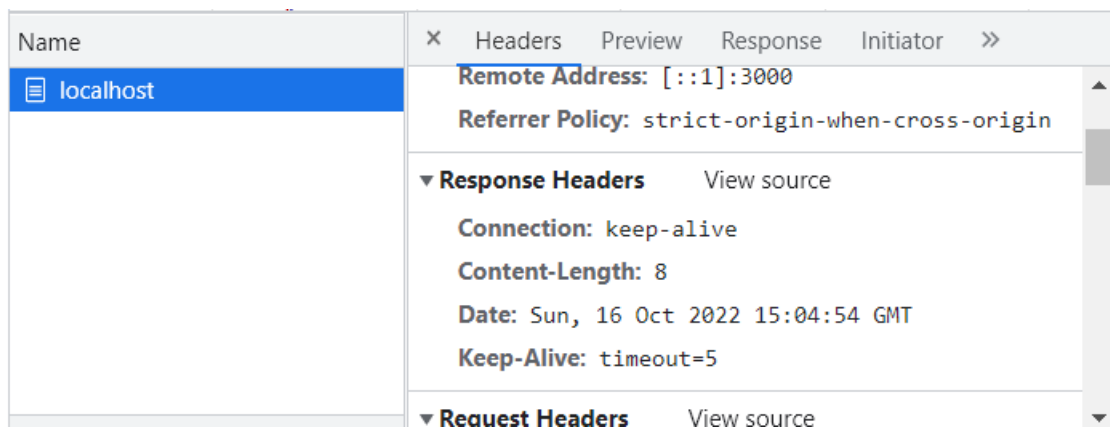
## 1. 쿠키의 기능

- ① 웹 브라우저와 웹서버가 주고 받는 정보
- ② 세션 관리(인증) : 서버에 저장해야 할 정보를 관리
- ③ 개인화 : 사용자 선호, 테마 등의 설정
- ④ 트래킹 : 사용자 행동을 기록하고 분석하는 용도
- ⑤ 쿠키 설정  
Set-Cookie : <cookie-name>=<cookie-value>
- ⑥ 쿠키 예제  
HTTP/1.0 200 OK      **응답메시지**  
Content-type: text/html  
Set-Cookie: yummy\_cookie=choco      **쿠키 생성**  
Set-Cookie: tasty\_cookie=strawberry

## 2. 쿠키 생성

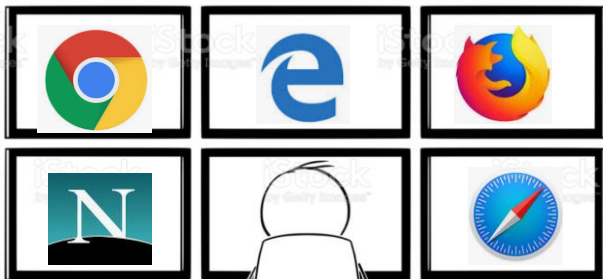
```
const express = require('express') ;
const app = express() ;
app.get('/',function(req, res){
  res.writeHead(200, {
    'Set-Cookie':['yummy_cookie=choco','tasty_cookie=strawberry']
  });
  res.end('Cookie!!');
});
app.listen(3000, () => console.log('Cookie Test'))
```

요청header에는 cookie가 없음. 두번째 요청부터 생성



오른쪽 마우스 버튼 → 검사 → 네트워크

클라이언트



'/' 요청

cookie를 response 객체의 header를  
통해 전송

쿠키저장

'/' 요청 시 request header에 쿠키전송

브라우저 닫으면 요청 쿠키 삭제

웹서버

```
const express = require('express') ;
const app = express() ;
app.get('/',function(req, res){
  res.writeHead(200, {
    'Set-
    Cookie':['yummy_cookie=choco','tasty_cookie=strawberry']
  });
  res.end('Cookie!!');
});
app.listen(3000, () =>
  console.log('Cookie Test'))
```



## 2. 쿠키 생성

```
const express = require('express') ;
const app = express() ;
app.get('/',function(req, res){
  //res.writeHead(200, {
  //  'Set-Cookie':['yummy_cookie=choco','tasty_cookie=strawberry']
  //})
  res.end('Cookie!!');
}) ;
app.listen(3000, () => console.log('Cookie Test'))
```

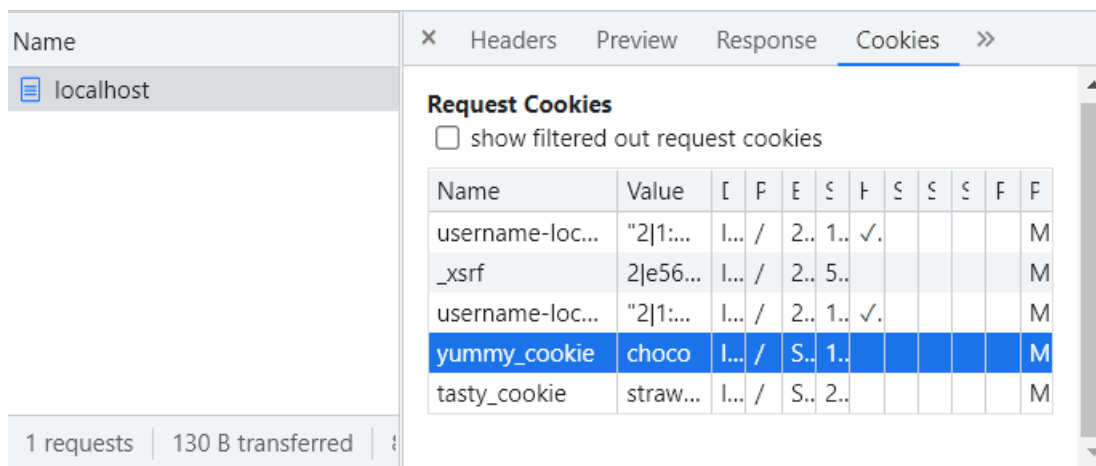
주석처리하고 다시 실행 시키면 response header에는 없고 request header에 생성  
브라우저에서 새로 고침 할 때마다 Set-Cookie로 저장된 쿠키값을 Cookie라는 헤더값을 통해 서버로 전송

Name	× Headers Preview Response Initiator >>
localhost	Remote Address: [::1]:3000 Referrer Policy: strict-origin-when-cross-origin ▼ Response Headers View source Connection: keep-alive Content-Length: 8 Date: Sun, 16 Oct 2022 15:04:54 GMT Keep-Alive: timeout=5 ▼ Request Headers View source

Name	× Headers Preview Response Initiator >>
localhost	Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7 Cache-Control: max-age=0 Connection: keep-alive Cookie: yummy_cookie=choco; tasty_cookie=strawberry Host: localhost:3000 sec-ch-ua: "Chromium";v="106", "Google Chrome";v="106", "Not;A=Brand";v="99" 1 requests   130 B transferred

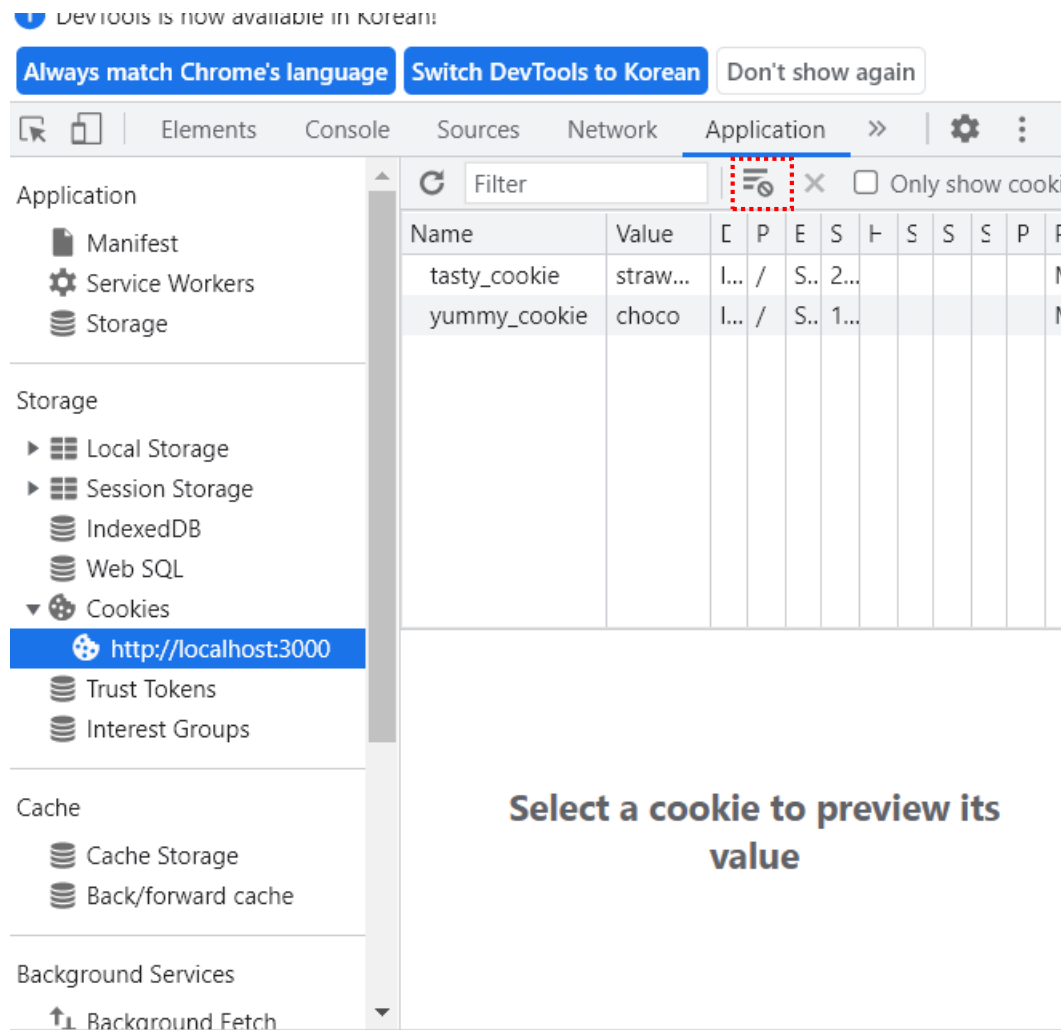
## 2. 쿠키 생성

현재 웹 브라우저에 어떤 쿠키가 있는지 보려면 웹 브라우저의 검사 창에서 [Cookies] 탭을 클릭



### 3. 쿠키 삭제

Application 탭에서 Cookie메뉴 클릭하고 빨간 박스의 아이콘을 클릭 하면 삭제



1. 생성한 쿠키를 웹 브라우저가 다시 웹 서버 쪽으로 전송(요청)했을 때 그것을 웹 애플리케이션에서 알 수 있는 방법

```
const express = require('express') ;
const app = express() ;
app.get('/', (req, res) => {
  console.log(req.headers.cookie);
  response.end('Cookie!!');
}) ;
app.listen(3000, () => console.log('Cookie Test'))
```

```
PS C:\Users\WBH\nodejs\prj\cookie> node .\cookie.js
Cookie Test
yummy_cookie=choco; tasty_cookie=strawberry
```

## 2. 쿠키 관리 모듈 설치: 문자열로 되어 있는 쿠키를 객체로 만드는 모듈

```
PS C:\Users\WBH\nodejs\prj> npm install -s cookie
```

## 3. cookie 모듈 추가 : 쿠키를 객체화 함.

```
var express = require('express') ;
var app = express() ;
var cookie = require('cookie');
app.get('/', function(req, res){
  // res.writeHead(200, {
  //   'Set-Cookie': ['yummy_cookie=choco', 'tasty_cookie=strawberry']
  // });
  console.log(req.headers.cookie);
  var cookies = cookie.parse(req.headers.cookie);
  console.log(cookies);
  res.end('Cookie!!');
});
app.listen(3000, () => console.log('Cookie Test'))
```

```
yummy_cookie=choco; tasty_cookie=strawberry
{ yummy_cookie: 'choco', tasty_cookie: 'strawberry' }
```

## 4. 쿠키를 없앴을 때를 대비한 코드

```
var express = require('express') ;
var app = express() ;
var cookie = require('cookie');
app.get('/',function(req, res){
  // res.writeHead(200, {
  //   'Set-Cookie':['yummy_cookie=choco','tasty_cookie=strawberry']
  // });
  console.log(req.headers.cookie);
  if(req.headers.cookie!==undefined){
    var cookies = cookie.parse(req.headers.cookie);
  }
  console.log(cookies.yummy_cookie);
  res.end('Cookie!!');
});
app.listen(3000, () => console.log('Cookie Test'))
```

## 11 세션(Session)과 영구 쿠키(Permanent 쿠키)

### 1. 쿠키를 언제까지 살아 있게 할 것인가?

- ① 세션 쿠키 – 웹 브라우저가 켜져 있는 동안에만 유효한 쿠키  
브라우저를 종료했다가 다시 실행하면 사라짐  
앞선 절의 yummy\_cookie는 세션 쿠키
- ② 영구 쿠키 – 웹 브라우저를 종료했다가 다시 실행해도 쿠키가 살아 있음.  
세션 쿠키에 Max-Age나 Expires 같은 옵션을 설정하면 영구 쿠키가 됨.  
Max-Age : 쿠키가 현재부터 얼마 동안 유효한지 지정(상대적)  
Expires : 쿠키를 언제 해지할지 지정(절대적)

## 11 세션(Session)과 영구 쿠키(Permanent 쿠키)

### 2. Permanent 쿠키를 지정 후 프로그램 실행 -> 검사 메뉴에서 확인

```
var express = require('express') ;
var app = express() ;
var cookie = require('cookie');
app.get('/',function(req, res){
  res.writeHead(200, {
    'Set-Cookie': ['yummy_cookie=choco',
                  'tasty_cookie=strawberry',
                  'Permanent=cookies;Max-Age=${60*60*24*30}` //초단위 지정으로 30일동안 유효
                  ]
    //영구 쿠키 지정 시 역따옴표 사용
  });
  console.log(req.headers.cookie);
  if(req.headers.cookie != undefined){
    var cookies = cookie.parse(req.headers.cookie);
  }
  console.log(cookies.yummy_cookie);
  console.log(cookies.Permanent);
  response.end('Cookie!!');
});
app.listen(3000, () => console.log('Cookie Test'))
```



## 11 세션(Session)과 영구 쿠키(Persistent 쿠키)

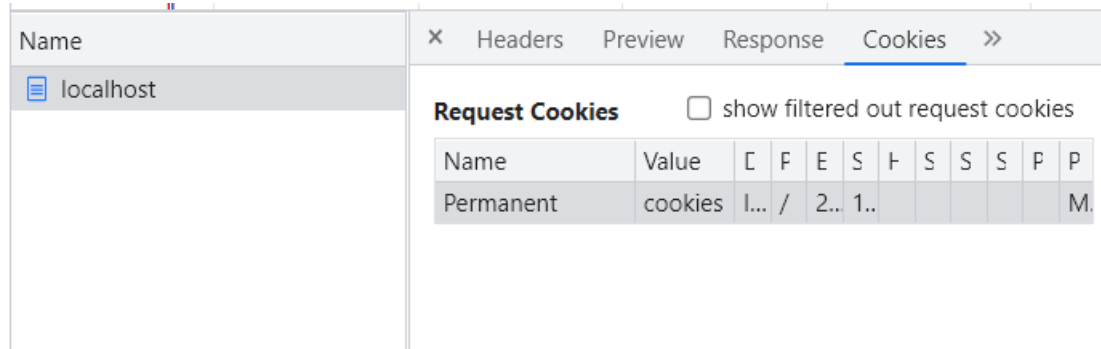
### 3. 쿠키 지정 부분 주석 처리 후 실행

main.js

```
var express = require('express') ;
var app = express() ;
var cookie = require('cookie');
app.get('/',function(req, res){
  // res.writeHead(200, {
  //   'Set-Cookie':['yummy_cookie=choco',
  //                 'tasty_cookie=strawberry',
  //                 `Permanent=cookies;Max-Age=${60*60*24*30}` //초단위 지정으로 30일동안 유효
  //   ]
  // });
  console.log(req.headers.cookie);
  if(req.headers.cookie != undefined){
    var cookies = cookie.parse(req.headers.cookie);
  }
  console.log(cookies.yummy_cookie);
  console.log(cookies.Permanent);
  response.end('Cookie!!');
});
app.listen(3000, () => console.log('Cookie Test'))
```

## 11 세션(Session)과 영구 쿠키(Persistent 쿠키)

4. 브라우저를 닫았다가 다시 열어서 실행 → **Permanent**만 남아 있음



1. 보안과 관련된 쿠키 옵션
2. **Secure** : 웹 브라우저와 웹 서버가 **HTTPS** 프로토콜로 통신하는 경우에만 쿠키를 전송
3. **HttpOnly** : **HTTP** 프로토콜로 통신하는 경우에도 쿠키를 전송하지만 자바스크립트로는 쿠키값을 가져올 수 없게 하는 옵션

1. 특정 디렉터리에서만 쿠키가 활성화되게 하고 싶을 때 사용하는 옵션이 **Path**
2. 어떤 서브 도메인에서도 생성되는 쿠키를 만들 수 있는 옵션이 **Domain**

# 쿠키를 이용한 인증

학습을 위해서만 사용  
현업에서는 인증용도로 쿠키를 사용하지 않음

- 이메일과 비밀번호를 입력하고 **submit** 버튼을 누르면 로그인 되고 로그아웃 링크를 누르면 로그아웃되는 프로그램
- 로그인 하면 **email, nickname, password** 등의 쿠키가 생성됨
- 쿠키 값은 브라우저에 노출이 됨으로 실제 현장에서는 로그인 **ID, PW**에 쿠키를 사용하지는 않음.

## 1. 로그인 링크 만들기

home.ejs

```
<body>
  <a href="/login">login</a>
  <h1><a href="/"><%=title%></a></h1>
  <a href="/author">저자관리</a>
  <ol type="1">
```



수시로 바뀌어야 하니까

```
<body>
  <%=lg%>
  <h1><a href="/"><%=title%></a></h1>
```



## 1. 로그인 링크 만들기

main.js에 /login URL 분류기 추가

```
app.get('/login',(req,res)=>{  
  topic.login(req,res);  
})
```



## 1. 로그인 링크 만들기

topic.js에 home 메소드 수정 – home.ejs의 login 변수에 넘겨줄 값 생성

```
home : (req,res)=>{
  db.query('select * from topic', (error, topics)=>{
    var login = '<a href="/login">login</a>'
    var m = '<a href="/create">create</a>'
    var b = '<h2> Welcome </h2><p>Node.js Start Page </p>'

    if (topics.length == 0){
      b = '<h2> Welcome </h2><p>자료가 없으니 create 링크를 이용하여 자료를 입력하세요 </p>'
    }

    var context = {login: login,
                  list : topics,
                  menu : m,
                  body : b };
    res.render('home55',context,(err,html)=>{
      res.end(html)
    });
  });
},
```

## 2. login 을 클릭하면 다음과 같은 화면이 나오도록 **topic.js**를 수정하기

← → ↻ localhost:3000/login

[login](#)

## Topic List

[저자관리](#)

1. [MySQL](#)
2. [NodeJs](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [날씨](#)
7. [XSS](#)

[create](#)

email

password

### 3. 제출 버튼을 클릭하면 로그인 표시가 로그아웃으로 바뀌면서 쿠키를 생성하는 프로그램 작성하기

← → ↻ localhost:3000

[logout](#)

## Topic List

[저자관리](#)

1. [MySQL](#)
2. [NodeJs](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [날씨](#)
7. [XSS](#)

[create](#)

## Welcome

Node.js Start Page

main.js에 다음 코드 추가하기

```
app.post('/login_process',(req,res)=>{  
    topic.login_process(req,res);  
  
})
```

### 3. 제출 버튼을 클릭하면 로그인이 로그아웃 되면서 쿠키를 생성하는 프로그램 작성하기

topic.js에 login\_process 메소드 추가하기

```
login_process : (req, res) => {  
  var body = '';  
  req.on('data', (data)=> {  
    body = body + data;  
  });  
  req.on('end', () => {  
    var post = qs.parse(body);  
    if(post.email==='bhwang99@gachon.ac.kr' && post.password === '123456'){  
      res.writeHead(302, {  
        'Set-Cookie': [  
          `email = ${post.email}`,  
          `password=${post.password}`,  
          `nickname='egoing'`, Location: '/' });  
      res.end();  
    }  
    else {  
      res.end('who?');  
    }  
  });  
},
```

쿠키 생성코드

### 3. 제출 버튼을 클릭하면 로그인이 로그아웃 되면서 쿠키를 생성하는 프로그램 작성하기

topic.js에 home 메소드 수정하기

```
home : (req,res) => {  
  var isOwner = false;  
  var cookies = {};  
  if(req.headers.cookie)  
    { cookies= cookie.parse(req.headers.cookie); }  
  if(cookies.email=== 'bhwang99@gachon.ac.kr' && cookies.password === '123456')  
    { isOwner = true; }  
  
  db.query('SELECT * FROM topic', (error,topics)=>{  
    var login = ''  
    if (isOwner){ login = `<a href="/logout_process">logout</a>` }  
    else { login = `<a href="/login">login</a>` }  
    console.log(login);  
    var m = `<a href="/create">create</a>`  
    var b = `<h2>welcome</h2><p>Node.js Start Page</p>`  
  
    var context = { login : login,  
                    title: 'Topic List',  
                    list:topics,  
                    menu:m,  
                    body:b};  
    req.app.render('home', context, (err,html)=>{  
      res.end(html) })  
  });  
},
```

## 3. 제출 버튼을 클릭하면 로그인이 로그아웃 되면서 쿠키를 생성하는 프로그램 작성하기

topic.js 함수화 하기

```
function authIsOwner(req,res) {
  var isOwner = false;
  var cookies = {};
  if(req.headers.cookie)
    { cookies= cookie.parse(req.headers.cookie); }
  if(cookies.email=== 'bhwang99@gachon.ac.kr' &&
cookies.password === '123456')
    { isOwner = true; }
  return isOwner
}

module.exports = {
```

```
home : (req,res) => {

  db.query('SELECT * FROM topic', (error,topics)=>{
    var login = ''
    var isOwner = authIsOwner(req,res);
    if (isOwner){ login = `<a
href="/logout_process">logout</a>` }
    else { login = `<a href="/login">login</a>` }
    console.log(login);
    var m = `<a href="/create">create</a>`
    var b = `<h2>welcome</h2><p>Node.js Start
Page</p>`

    var context = { login : login,
                    title: 'Topic List',
                    list:topics,
                    menu:m,
                    body:b};
    req.app.render('home', context, (err,html)=>{
      res.end(html) })
  });
},
```

#### 4. 로그아웃 처리

main.js 에  
logout\_process추가

```
app.get('/logout_process', (req, res) => {  
  topic.logout_process(req, res);  
})
```

topic.js 에  
logout\_process추가

```
logout_process : (req, res) => {  
  res.writeHead(302, {  
    'Set-Cookie': [  
      `email = ; Max-Age=0`,  
      `password = ; Max-Age=0`,  
      `nickname = ; Max-Age=0`],  
    Location: '/'  
  });  
  res.end();  
},
```

## 5. 접근 제어

로그인한 사람만 생성, 갱신, 삭제 가능하도록

```
create : (req,res) => {  
  if(authIsOwner(req,res) === false){  
    res.end(`<script type='text/javascript'>alert("Login required ~~~")  
    <!--  
    setTimeout("location.href='http://localhost:3000/'",1000);  
    //-->  
    </script>`)  
  }  
  db.query(`SELECT * FROM topic`, (error,topics)=>{  
    추가 {
```

갱신과 삭제에도 적용하시오.



## 6. login 링크 만들어 주는 함수

```
function authStatusUI(req,res) {  
    var login = '<a href="/login">login</a>';  
    if (authIsOwner(req,res)){ login = '<a href="/logout_process">logout</a>' }  
    return login;  
}
```

적용 가능한 부분을 찾아서 적용하시오.