

생활코딩

Node.js 노드제이에스 프로그래밍

2024.10.01



5주차 수업의 범위

- * 제목을 클릭하면 **topic** 테이블의 **descript** 필드 내용을 보여주는 기능
- * 글 생성 기능, **Form** 태그
- * 글 갱신 기능
- * 글 삭제 기능

localhost:3000/

초기 화면

WEB

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

[create](#)

Welcome

Node.js Start Page

빨간 박스 안의 내용을 main.js로부터 받게 됨

제목을 클릭하면
상세 페이지 출력



두 화면은 같은 ejs
파일을 이용

localhost:3000/page/1

상세 페이지 화면

WEB

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

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

MySQL

MySQL is Database Name.

제목을 클릭하여 나타나는
하단의 내용은
그 제목에 해당하는
descript 컬럼 내용

1. 제목을 클릭하면 세부 내용 **descript** 필드를 보여주는 기능 - **root**로 들어갈 때

프로그램

main.js

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

URL

 localhost:3000

화면

WEB

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

[create](#)**Welcome**

Node.js Start Page

상세 페이지에서는
생성, 갱신, 삭제 메뉴가
들어갈 부분

topic.js

```
home : (req, res) => {  
  
  db.query('SELECT * FROM topic', (error, topics) => {  
    var m = '<a href="#">create</a>'  
    var b = '<h2>welcome</h2><p>Node.js Start Page</p>'  
  
    var context = {list:topics,  
                   menu:m,  
                   body:b};  
    req.app.render('home', context, (err, html) => {  
      res.end(html)    })  
    });  
  },
```

1. 제목을 클릭하면 세부 내용 **descript** 필드를 보여주는 기능 - **root**로 들어갈 때

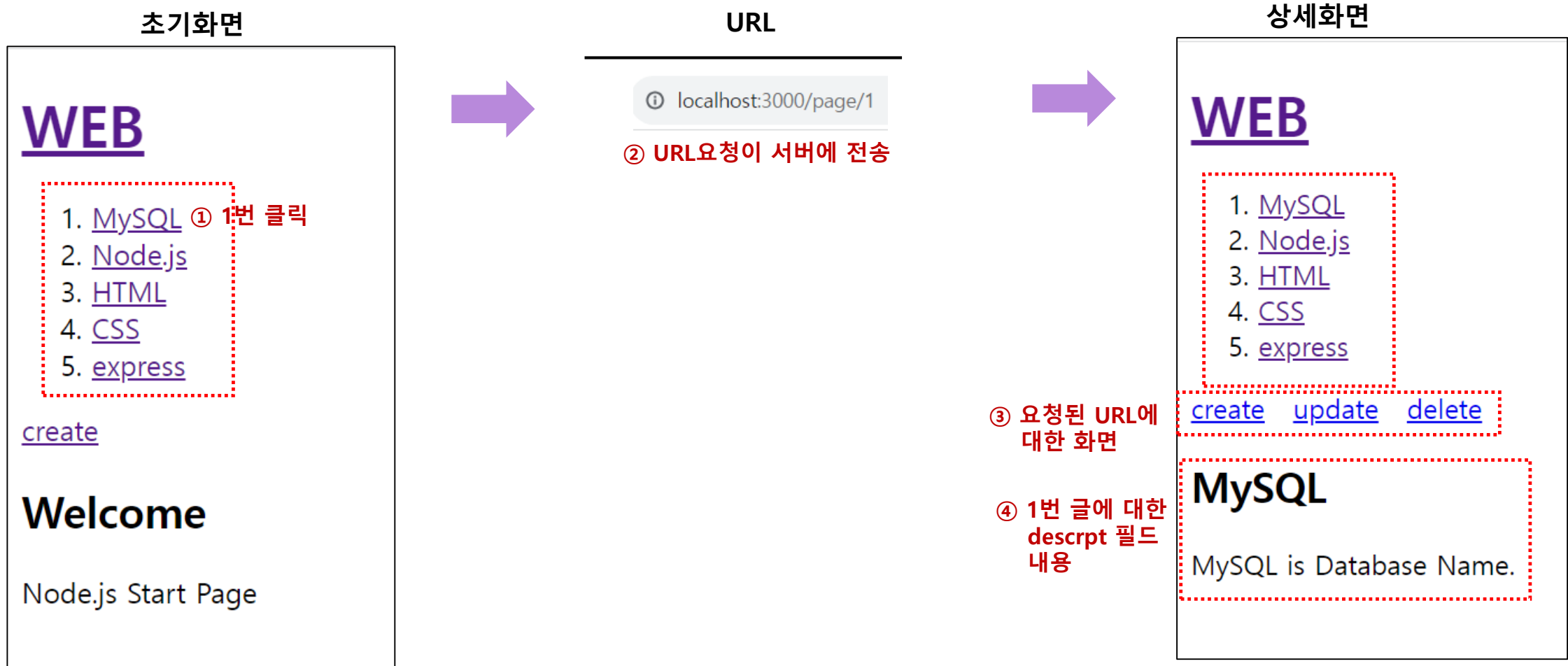
home.ejs

```
<!doctype html>
<html>
<head>
  <title>WEB1</title>
  <meta charset="utf-8">
  <link rel="icon" href="data:,">
</head>
<body>
  <h1><a href="/">WEB</a></h1>
  <ol type="1">
    <% var i = 0
      while(i < list.length)
        { %>
          <li><a href="/#"><%=list[i].title%></a></li>
        <% i += 1 } %>
    </ol>

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

  </body>
</html>
```

2. 제목을 클릭하면 세부 내용 **descript** 필드를 보여주는 기능 - 제목 클릭했을 때



2. 제목을 클릭하면 세부 내용 **descript** 필드를 보여주는 기능 - 제목 클릭했을 때

main.js

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

[작업 순서]

- ① 제목을 클릭했을 때의 요청 URL을 main.js에 추가 : 요청 URL은 /page/글번호 임.
- ② home.ejs의 title 목록 부분의 제목 <a href> 링크를 수정
- ③ 추가된 URL 요청이 왔을 때 처리하는 프로그램을 topic.js에 page 메소드로 작성

2. 제목을 클릭하면 세부 내용 **descript** 필드를 보여주는 기능 - 제목 클릭했을 때

home.ejs

```
<!doctype html>
<html>
<head>
  <title>WEB1</title>
  <meta charset="utf-8">
  <link rel="icon" href="data:,">
</head>
<body>
  <h1><a href="/">WEB</a></h1>
  <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>

    <%- control%>
    <%- body%>

  </body>
</html>
```

- 제목의 링크 부분 수정
- list에는 DB로 부터 검색된 row들이 객체 배열로 저장되어 있으므로 첫번째 row는 list[0]임
- 각 row에는 id, title, descript, created 필드가 있음.
- 이 필드를 읽을 때에는 list[인덱스].id, list[인덱스].title과 같이 읽어옴.
- id와 title은 객체의 속성이 됨.

2. 제목을 클릭하면 세부 내용 **descript** 필드를 보여주는 기능 - 제목 클릭했을 때

topic.js

```
page : (req,res) => {
  var id = req.params.pageId; ① "/page/<%=list[i].id%>" 요청 시 글 번호가 pageId에 저장되고 req.params.pageId로 읽어올 수 있다
  db.query('SELECT * FROM topic', (error,topics)=>{ ② 글 목록을 위한 query
    if(error){
      throw error;
    }
    db.query(`SELECT * FROM topic WHERE id = ${id}`,(error2, topic)=>{ ③ 선택된 글의 descript 필드를 읽기 위한 query
      if(error2){
        throw error2;
        where 절로 id가 url에 있는 id와 같다는 조건 부여
      }

      var m = <a href="#">create</a>&nbsp;&nbsp;&nbsp;<a href="#">update</a>&nbsp;&nbsp;&nbsp;<a href="#">delete</a>
      var b = <h2>${topic[0].title}</h2><p>${topic[0].descript}</p>

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

2. 제목을 클릭하면 세부 내용 **descript** 필드를 보여주는 기능 - 제목 클릭했을 때

id	title	descript	created	author_id
1	MySQL	MySQL is Database Name.	2023-09-20 00:00:00	1
2	Node.js	Node.js is runtime of javascript	2023-09-20 00:00:00	1
3	HTML	HTML is Hyper Text Markup Language	2023-09-20 00:00:00	1



```
topic    { id : 1,  
          title : 'MySQL',  
          descript : 'MySQL is Database Name.',  
          created : '2023-09-20 00:00:00',  
          author_id : 1  
        },  
        ...
```

* **create**를 클릭하면 **/create** 요청이 서버에 전송되고 글을 입력할 수 있는 **html** 코드를 **b**에 저장해 **ejs**에 전달
글 입력 후에 제출 버튼을 클릭하면 글이 **DB**에 저장되고 입력 화면의 상세 내용 페이지로 **redirect**

← → ↻ localhost:3000

WEB

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

[create](#)

Welcome

Node.js Start Page



← → ↻ localhost:3000/create

WEB

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

[create](#)

딥러닝을 수행하는 플랫폼 이다.

제출



← → ↻ localhost:3000/page/8

WEB

1. [MySQL](#)
2. [Node.js](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [GAN](#)
7. [트랜스포머](#)
8. [케라스](#)

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

케라스

딥러닝을 수행하는 플랫폼 이다.

1. create의 링크를 /#에서 /create로 수정하기

`localhost:3000/#``localhost:3000/create`

WEB

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

[create](#)

Welcome

Node.js Start Page

2. create 링크를 /create로 수정하기 위해서는 topic.js에서 변수 c를 수정한다.

topic.js

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

```
page : (req,res) => {  
    .....  
    var c = `<a href="/create">create</a>&nbsp;&nbsp;&nbsp;<a  
href="#">update</a>&nbsp;&nbsp;&nbsp;<a href="#">delete</a>`  
    var b = `<h2>${topic[0].title}</h2><p>${topic[0].descript}</p>`
```

3. 새로운 URL /create가 생성되었으므로 URL 분류기를 main.js에 추가한다.

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

4. topic 모듈에 /create 요청을 처리하는 create 메소드를 작성한다.

4. topic 모듈에 /create 요청을 처리하는 create 메소드를 작성한다.

topic.js

```
create : (req,res)=>{
  db.query('select * from topic', (error, topics)=>{
    if(error){
      throw error
    }
    var b = `

<form action="/create_process" method="post">
        <p><input type="text" name="title" placeholder="title"></p>
        <p><textarea name="description" placeholder="description"></textarea></p>
        <p><input type="submit"></p>
      </form>`
    var context = {list : topics,
      menu : '<a href="/create">create</a>',
      body : b };
    res.render('home',context,(err,html)=>{
      res.end(html)
    }); //render 종료
  }); //첫번째 query 종료
},


```

※ form tag

- * **action** - **submit** 버튼을 클릭하면 데이터를 전달할 주소(URL). 이 주소에 연결된 프로그램이 **form** 태그 내부의 데이터를 전달 받아서 처리함
- * **method** - **form** 태그 내부의 데이터를 전달하는 방법 **Get** 방식과 **Post** 방식이 있음
- * **name** - 각 데이터의 이름 또는 데이터를 담는 변수라고 생각해도 무방
action에 있는 주소에 연결된 프로그램은 이 이름으로 데이터를 제어

```
<form action="/process_create" method="post">  
  <p><input type="text" name="title"></p>  
  <p>  
    <textarea name="description"></textarea>  
  </p>  
  <p>  
    <input type="submit">  
  </p>  
</form>
```


※ GET과 POST 방식

- GET, POST
 - GET과 POST는 HTTP프로토콜을 이용해서 서버에 무언가를 전달할 때 사용하는 방식
 - GET은 주소줄에 값이 ?뒤에 쌍으로 이어 붙고 POST는 숨겨져서(body안에) 보내진다.
 - GET은 URL에 이어 붙기 때문에 길이제한이 있어서 많은 양의 데이터는 보내기 어렵고 POST는 많은 양의 보내기에도 적합하다.(용량제한은 있음)
 - 즉 `http://url/bbslist.html?id=5&pagenum=2` 같이 하는 것이 GET방식
 - id를 넘겨서 게시판의 리스트를 가져온다고 하면 당연히 GET을 쓸 것이고 글을 작성한다고 하면 POST를 작성하는 것이 일반적.
 - 전달해야 될 양이 많을 경우에는 고민 없이 POST를 사용. 양이 많지 않은 경우에는 GET도 되고 POST도 됨.
 - GET은 가져오는 것이고 POST는 수행하는 것입니다.
 - GET은 Select적인 성향, POST는 서버의 값이나 상태를 바꾸기 위해서 사용
 - GET은 서버에서 어떤 데이터를 가져와서 보여준다거나 하는 용도이지 서버의 값이나 상태 등을 바꾸지 않음
게시판의 리스트라던지 글 보기 기능 같은 것이 이에 해당
 - 글쓰기를 하면 글의 내용이 DB에 저장이 되고 수정을 하면 DB 값이 수정. 이럴 경우에 POST를 사용

topic 테이블에 글 생성 – Create

※ form tag 작동 원리

① 제출 버튼을 클릭하면

② action 속성에 있는 URL이 서버에 요청된다.

```
<form action="/create_process" method="post">
  <p><input type="text" name="title" placeholder="title"></p>
  <p><textarea name="description"
placeholder="description"></textarea></p>
  <p><input type="submit"></p>
</form>
```

③ 그러므로 action 속성에 있는 URL에 대한 분류기를
main.js에 추가해야 하고
topic.js에 /create_process URL 요청을 처리하는 메소드를 추가해야 한다.

④ /create_process URL 요청을 처리하는 메소드에 form태그 안에 있는
input 태그나 textarea 태그 내용이 name 속성에 정의된 이름을
이용하여 전달된다.

topic 테이블에 글 생성 - Create

※ form tag 자료가 전송되는 것을 확인

WEB

- [CSS](#)
- [HTML](#)
- [JavaScript](#)

create

① 글작성

제목: node.js

④ 제출

뒤로	Alt+왼쪽 화살표
앞으로	Alt+오른쪽 화살표
새로고침	Ctrl+R
다른 이름으로 저장...	Ctrl+S
인쇄	Ctrl+P
전송...	
Google Lens로 이미지 검색	
이 페이지의 QR 코드 생성	
Google에서 이미지 설명 가져오기	
페이지 소스 보기	Ctrl+U

② 검사

DevTools is now available in Korean!

Always match Chrome's language Switch DevTools to Korean Don't show again

Elements Console Sources **Network** Performance >> | Settings | More

Filter [] [] Invert [] Hide data URLs

All | Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other

[] Has blocked cookies [] Blocked Requests [] 3rd-party requests

Name	Stat..	Type	Initiator	Size	Time	Waterfall
create_process	404	doc...	Other	157 B	6 ms	

⑤ create_process

- ① input 박스에 글 입력
- ② 오른쪽 마우스 버튼을 클릭하고 검사 메뉴 클릭
- ③ Network 메뉴 클릭
- ④ 제출 버튼
- ⑤ create_process 클릭
- ⑥ Payload 클릭
- ⑦ 확인

⑥ Payload

Name	Headers	Payload	Preview	Response	>>
create_process		<p>Form Data view source view URL-encoded</p> <p>title: 글작성</p> <p>description: 제목: node.js</p>			

⑦ 확인

5. /create_process 분류기를 추가한다.

main.js

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

form 태그에서 /create_process 요청은 post 방식으로 하였기 때문에 app객체의 get 메소드가 아닌 post 메소드를 호출해야 한다.

6. /create_process 요청을 처리하는 create_process 메소드를 topic 모듈에 추가한다.

topic.js

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

```
create_process : (req,res) => {
  var body = '';
  req.on('data', (data)=> {
    body = body + data;
  });
  req.on('end', () => {
    var post = qs.parse(body);
    db.query(`
      INSERT INTO topic (title, descrpt, created)
      VALUES(?, ?, NOW())`,
      [post.title, post.description], (error, result)=> {
        if(error) {
          throw error;
        }
        res.writeHead(302, {Location: `/page/${result.insertId}`});
        res.end();
      }
    );
  });
}
```

6. /create_process 요청을 처리하는 create_process 메소드를 topic 모듈에 추가한다.

topic.js

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

querystring 모듈 : url의 쿼리 스트링을 해석하고 포매팅 할 수 있다.

```
req.on('data', (data)=> {  
  body = body + data;  
});
```

request객체의 on메소드 : 특정 이벤트를 listen할 수 있게 해주는 기능

첫번째 data인자 – data 이벤트

두번째 콜백함수 인자 – data 이벤트가 발생하면 실행하는 함수

data 이벤트 – 클라이언트로부터 데이터를 수신할 때마다 발생하는 이벤트

클라이언트는 데이터(예: form 태그의 데이터 등등)를 보낼 때 여러 개로 나누어 보냄

body = body + data : 여러 개로 나누어진 data가 수신 될 때마다 body 변수에 연결하여 원래 크기의 데이터로 만든다.

6. /create_process 요청을 처리하는 create_process 메소드를 topic 모듈에 추가한다.

topic.js

```
req.on('end', () => {
```

end 이벤트 : data 수신이 끝나면 발생하는 이벤트

```
var post = qs.parse(body);
```

querystring의 parse 메소드 : querystring을 분석하여 객체로 반환해 준다. 여기에서는 form 태그에 포함된 input태그나 textarea 태그의 내용을 name 속성에 정의된 이름으로 객체를 생성

```
{ title : '   ~~~', description : '~~~~~'}
```

```
db.query(`
  INSERT INTO topic (title, descrpt, created)
  VALUES(?, ?, NOW())`,
  [post.title, post.description], (error, result)=> {
```

Insert 문 안의 ? : 파라메타로 넘겨질 값의 위치

두번째 인자인 [post.title, post.description] : insert 문 안의 ? 위치에 차례대로 들어갈 값. 보안을 위해 이런 표기법 추천

result : insert문에서 콜백함수의 두번째 인자에는 insert된 row의 순번

6. /create_process 요청을 처리하는 create_process 메소드를 topic 모듈에 추가한다.

topic.js

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

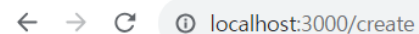
Location 속성에 저장된 URL로 redirect

express에서는 res.redirect 메소드를 사용할 수 있다.

```
res.redirect(`/page/${result.insertId}`)
```


7. 정리

[클라이언트]

WEB

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

create

딥러닝을 수행하는 플
랫폼 이다.

[서버]

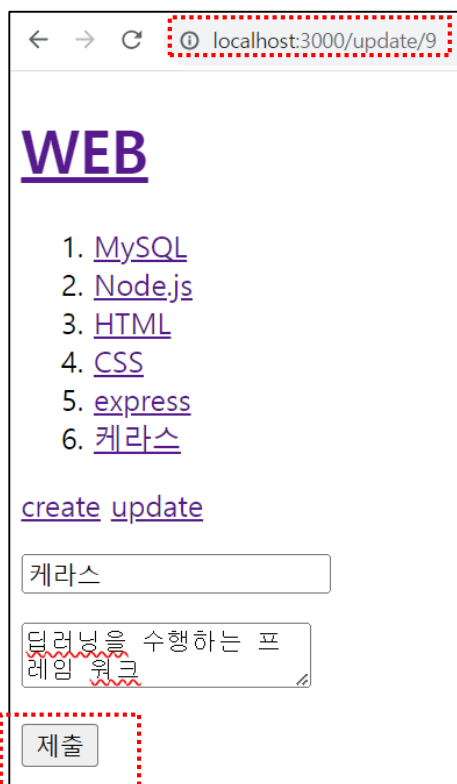
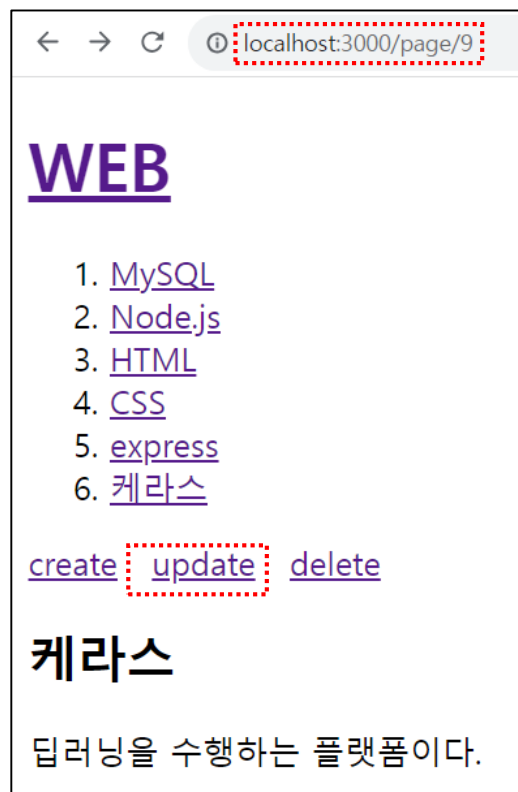
/create_process URL 요청

create_process 메소드 호출

create_process 메소드에 전달
되어 post 변수에 저장

상세 페이지로 redirection

* **update**를 클릭하면 **/update** 요청이 서버에 전송되고 현재 상세 내용에 보여지는 글을 수정할 수 있는 **html** 코드를 **b**에 저장해 **ejs**에 전달. 글 수정 후에 제출 버튼을 클릭하면 글이 **DB**에 수정되고 상세 내용 페이지로 **redirect**



- 상세 화면에서 update를 클릭하면 update 화면이 나오도록 update의 링크를 /#에서 /update/id로 수정하기.
update는 현재 상세내용을 보고 있는 글을 수정해야 하기 때문에 url에 현재글의 id를 함께 넘겨 주어야 한다.

localhost:3000/page/10



localhost:3000/update/10

topic.js

```

page : (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;
            }

            var m = `<a
href="/create">create</a>&nbsp;&nbsp;&nbsp;<a
href="/update/${topic[0].id}">update</a>&nbsp;&nbsp;&nbsp;<a
href="#">delete</a>
            var b =
`<h2>${topic[0].title}</h2><p>${topic[0].descript}</p>`

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

```

2. update 링크를 /update/id로 수정하기 위해서는 topic.js에서 page메소드의 변수 m을 수정한다.

topic.js

```
var m = `
```

3. 새로운 URL /update/id가 생성되었으므로 URL 분류기를 main.js에 추가한다.

main.js

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

4. topic 모듈에 /update/:pageId 요청을 처리하는 update 메소드를 작성한다.

topic 테이블의 글 갱신 - Update

4. topic 모듈에 /update/:pageId 요청을 처리하는 update 메소드를 작성한다. page 메소드와 유사하지만 menu와 body에 저장하는 값이 다름

```
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  
            }  
            var m = ``  
            var b = `                <input type="hidden" name="id" value="\${topic\[0\].id}"> ① 어떤 글이 update 되는지 알려주기 위해  
                <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><input type="submit"></p>  
            </form>`  
  
            var context = {list : topics,  
                            menu : m,  
                            body : b };  
            res.render\('home08',context,\(err,html\)=>{  
                res.end\(html\)  
            }\); //render 종료  
        }\); //두번째 query 종료  
    }\); //첫번째 query 종료  
},
```

3. []

4. CS

5. ex

6. 케

create u

케라스

답러닝을 레임 워

제출

① update할 글은 id로 찾아야 하기 때문에 hidden으로 정보를 서버에 전달

※ 클라이언트가 서버에 자료를 보내는 방법은 form 태그와 querystring

topic 테이블의 글 갱신 - Update

5. 내용을 update하고 제출 버튼을 클릭하면 `action="/update_process"` 부분이 요청됨
 새로운 URL이 추가되면 `main.js`에 URL 분류기 추가하고 `topic.js`에 추가된 URL 요청에 대한
 처리 프로그램으로 구성된 메소드 추가해야 함.

← → ↻ ⓘ localhost:3000/update/9

WEB

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

[create update](#)

케라스

딥러닝을 수행하는 프레임 워크

제출

```
body : `<form action="/update_process" method="post">
      <input type="hidden" name="id" value="${topic[0].id}">
      <p><input type="text" name="title" placeholder="title"
value="${topic[0].title}"></p>
      <p><textarea name="description"
placeholder="description">${topic[0].descript}</textarea></p>
      <p><input type="submit"></p>
</form>`
```

제출 버튼을 클릭하면 `/update_process` URL을 처리하는 메소드에 form 태그에 포함된 내용들이
 다음 객체 형식으로 전달

```
{ id : ?,
  title : '~~~~',
  description : '~~~~~'
}
```

6. main.js에 /update_process에 대한 URL 분류기 추가. form 태그의 method 속성이 post 이므로 app의 post 메소드 호출

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

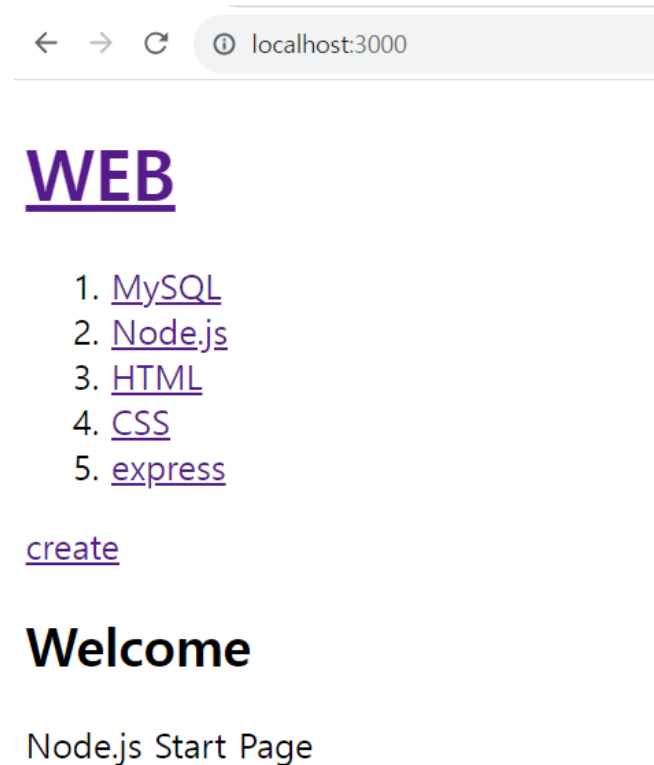
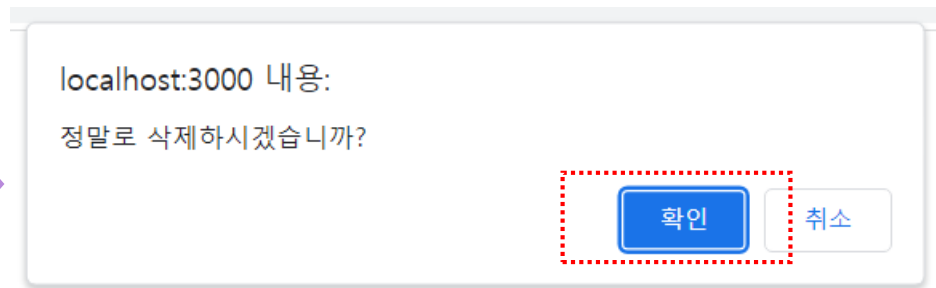
7. topic.js에 /update_process 요청에 대한 처리 메소드 update_process 생성

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

① `var post = qs.parse(body);`

post에 클라이언트로부터 전달된 form태그에 대한 객체가 저장

* 글 상세 화면에서 **delete**를 클릭하면 삭제 확인 메시지가 나오고 확인을 클릭하면 삭제되고 취소를 클릭하면 삭제 되지 않는다. 삭제 후 /경로로 간다.



1. page 메소드의 m 변수에 저장된 **delete** 링크를 수정한다.

```
var c = `<a href="/create">create</a>&nbsp;&nbsp;&nbsp;<a href="/update/${topic[0].id}">update</a>&nbsp;&nbsp;&nbsp;<a  
href="/delete/${topic[0].id}" ① onclick='if(confirm("정말로 삭제하시겠습니까?")==false){ return false }'>delete</a>`
```

①

②

① 서버에게 삭제할 글의 id를 알려 주어야 함으로 /delete URL에 id를 추가하여 /delete/id 형태로 요청

② `onclick='if(confirm("정말로 삭제하시겠습니까?")==false){ return false }'`

링크를 클릭하면 클릭 이벤트가 발생하며 클릭 이벤트를 처리하는 문장
삭제하지 않겠다고 응답하면 return false에 의해 링크가 작동 안됨.

2. `/delete/id` URL 에 대한 분류기를 `main.js`에 추가

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

3. /delete/id URL 에 대한 처리 메소드 `delete_process`를 `topic.js`에 추가

```
delete_process : (req, res) => {  
  id = req.params.pageId ;  
  db.query('DELETE FROM topic WHERE id = ?', [id], (error, result) => {  
    if(error) {  
      throw error;  
    }  
    res.writeHead(302, {Location: `/`});  
    res.end();  
  });  
}
```

※ 모든 자료를 다 지워서 테이블에 자료가 없을 때를 위한 코드 작성하기