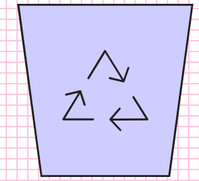
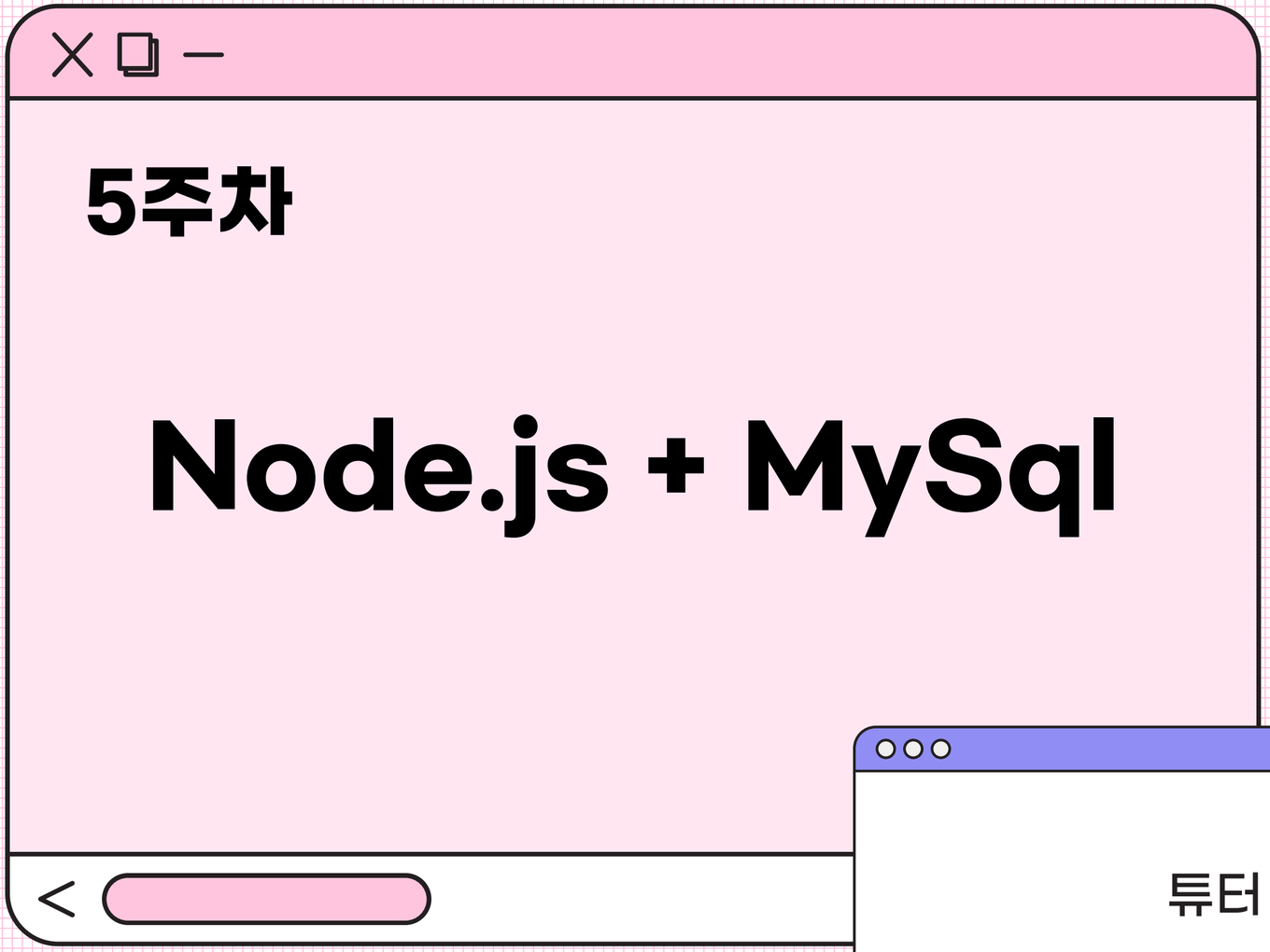
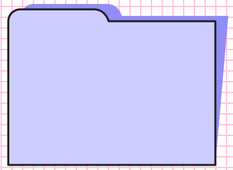
2022 OPEN YEAR ROUND 강의 자료

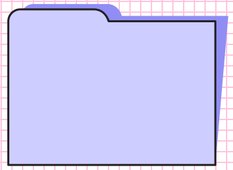




5주차

Node.js + MySql





튜터 : 정한이


Express.js

Express.js 설치

```
$ npm init -y
```

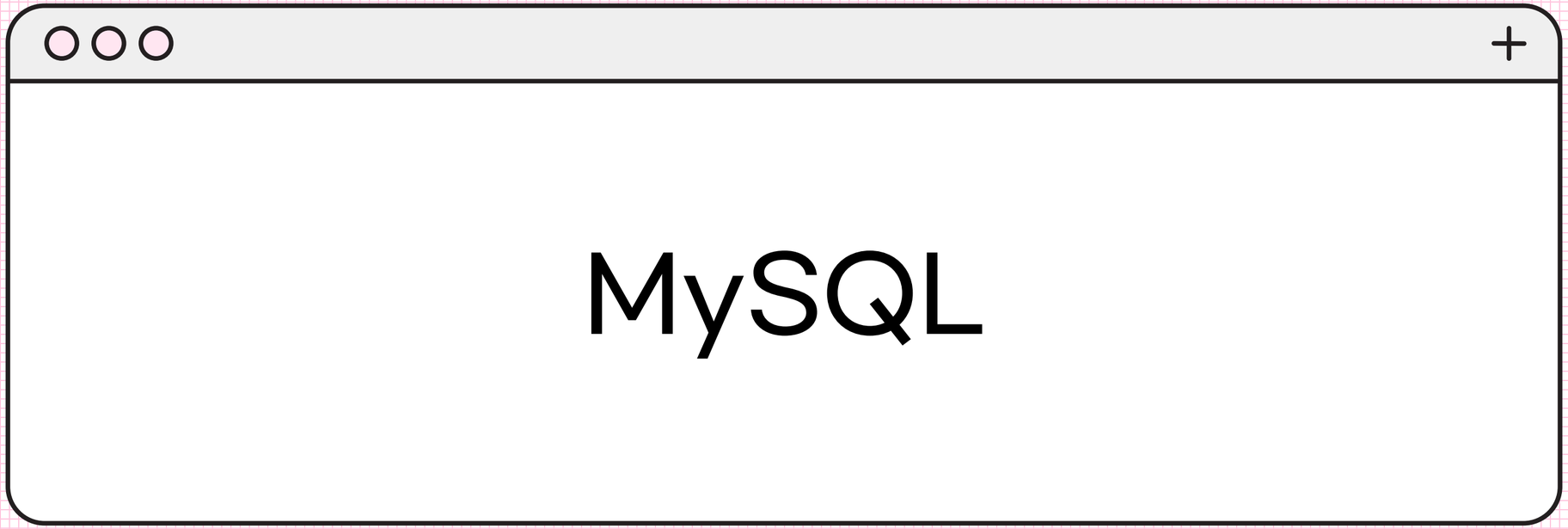
Express.js에서 사용하는 패키지 관리
package.json 파일 생성

```
$ npm install express
```

express.js 설치

Express.js 동작 확인

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(port, () => console.log('Example app listening on port 3000!'))
```


MySQL

MySQL 동작 확인

```
$ npm install mysql
```

mysql 드라이버 설치

```
const port = 3000

const mysql = require('mysql')

const con = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '[redacted]'
});

con.connect(function (err) {
    if (err) throw err;
    console.log('Connected');
});
```

app.js에서 MySQL 연결 확인

```
Example app listening on port 3000!
Connected
```

MySQL 데이터베이스 생성 및 연결

```
con.connect(function (err) {
    if (err) throw err;
    console.log('Connected');
    con.query('CREATE DATABASE express_db', function (err, result) {
        if (err) throw err;
        console.log('database created');
    });
});
```

데이터베이스 생성 시 CREATE DATABASE 명령어를 사용

Express.js는 sql문을 실행하기 위해 query 메소드를 사용

```
const con = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '[redacted]',
    database: 'express_db'
});
```

DB 연결 옵션에 생성된 데이터베이스 정보를 추가하면

createConnection 메소드에서 데이터베이스 접속까지 실행

테이블 생성

```
con.connect(function (err) {
    if (err) throw err;
    console.log('Connected');
    /*
    con.query('CREATE DATABASE express_db', function (err, result) {
        if (err) throw err;
        console.log('database created');
    });
    */
    const sql = 'CREATE TABLE users (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,name VARCHAR(255) NOT NULL,email VARCHAR(255) NOT NULL)';
    con.query(sql, function(err, result) {
        if(err) throw err;
        console.log('table created');
    });
});
```

테이블 생성 시, CREATE TABLE 명령어를 이용

insert문으로 데이터 삽입

```
const sql = "INSERT INTO users(name, email) VALUES('hani', 'hani@test.com')"

con.query(sql, function (err, result, fields) {
    if (err) throw err;
    console.log(result)
});
```

```
OkPacket {
  fieldCount: 0,
  affectedRows: 1,
  insertId: 1,
  serverStatus: 2,
  warningCount: 0,
  message: '',
  protocol41: true,
  changedRows: 0
}
```

console.log로 result에 들어있는 정보를 확인하면
OKPacket 개체에서 리턴되는 것을 알 수 있음

select문으로 얻은 데이터 브라우저 반환

```
app.get('/', (request, response) => {
    const sql = "select*from users"
    con.query(sql, function (err, result, fields) {
        if (err) throw err;
        response.send(result)
    });
});
```

select문으로 얻은 데이터를 브라우저에 반환하는 소스 코드

입력 폼 작성

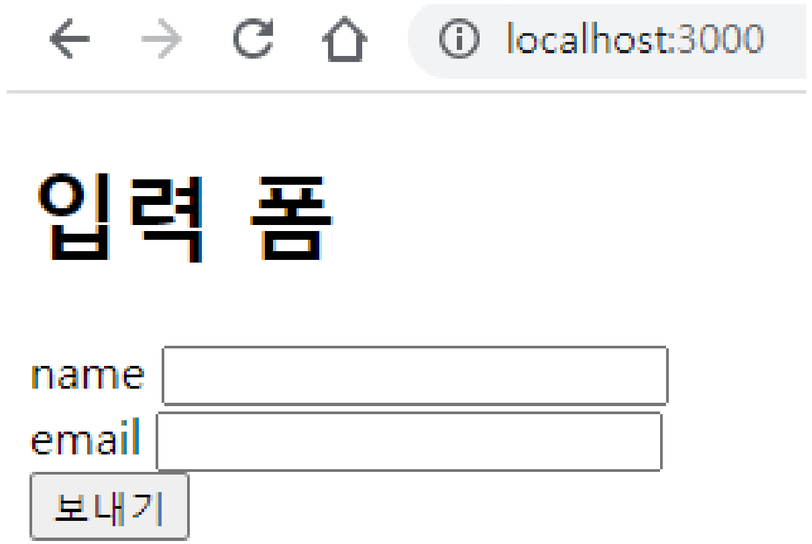
브라우저에서 입력한 데이터를 데이터베이스 테이블에 추가(insert)하는 방법

```html
<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="UTF-8">
    <title>입력 폼</title>
</head>

<body>
    <h1>입력 폼</h1>
    <form action="/" method="POST">
        name <input type="text" name="name"><br>
        email <input type="text" name="email"><br>
        <button type="submit">보내기</button>
    </form>
</body>

</html>
```

```
app.get('/', (req, res) =>
    res.sendFile(path.join(__dirname, 'html/form.html')))
```

입력폼이 표시되도록 app.js의 “/(루트)” 라우팅을 변경

sendFile 메소드를 사용하여 파일을 “/(루트)”에 접속한 브라우저에 전달



body-parser

```
>npm install body-parser
```

브라우저에서 입력한 name과 email은 POST 요청 결과로 express.js에 전송

전송된 name과 email을 express.js에서 처리하려면 body-parser 모듈이 필요

```
const express = require('express')
const app = express()
const port = 3000
const path = require('path')
const bodyParser = require('body-parser')
```

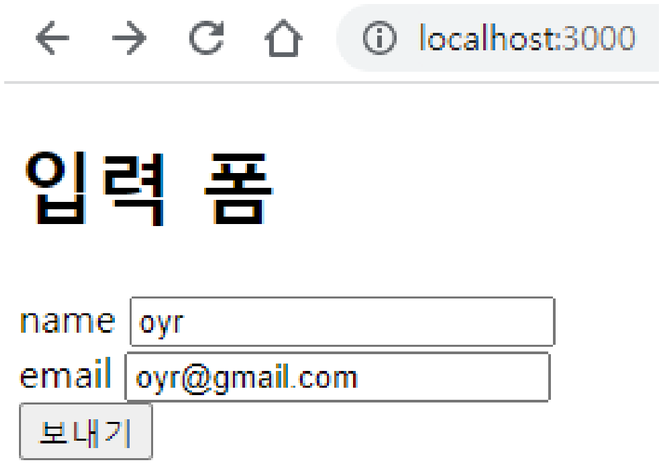
설치가 완료되면 body-parser 로드

```
app.use(bodyParser.urlencoded({ extended: true }));

app.get('/', (req, res) =>
    res.sendFile(path.join(__dirname, 'html/form.html')))

app.post('/', (req, res) => res.send(req.body))
```

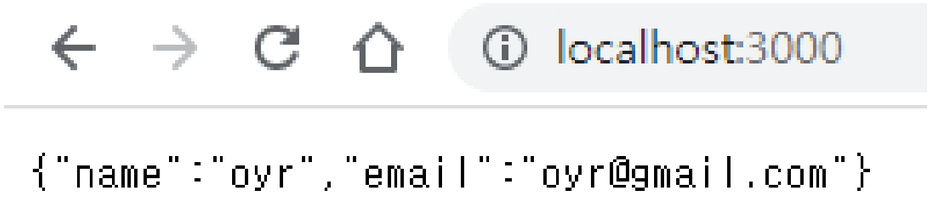
입력 폼에서 POST 요청으로 데이터를 얻을 수 있는지 확인하기 위해 라우팅을 설정

req.body에 받은 데이터가 저장됨

body-parser



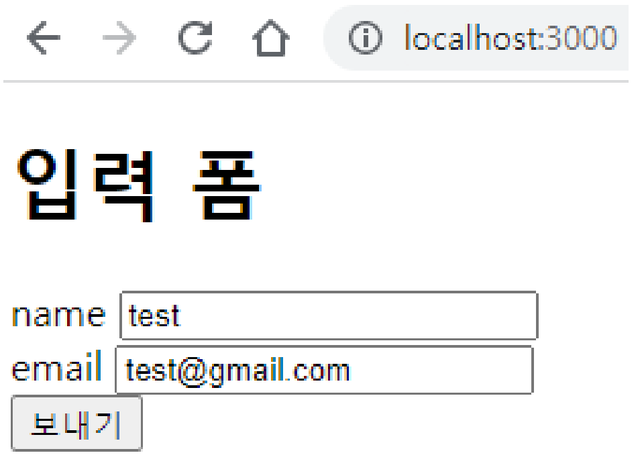
입력폼에 정보를 입력하고 보내기 버튼을 누름

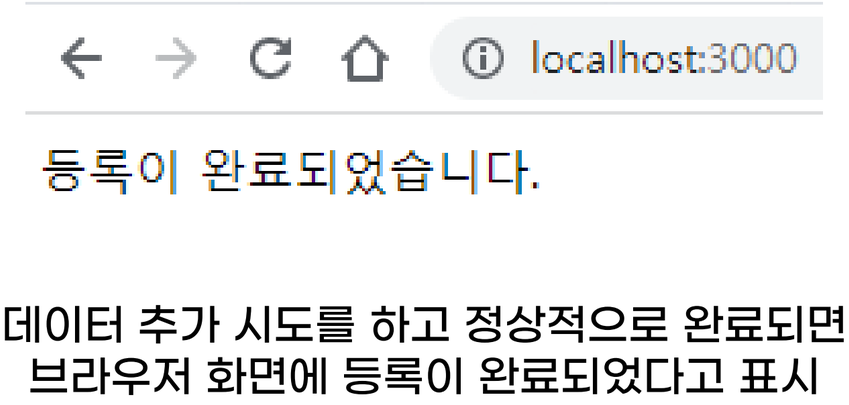


express.js에서는 POST 요청으로 데이터를 받고
res.send에서 req.body를 브라우저로 보내기 때문에
브라우저는 다음과 같이 req.body안의 값을 표시

폼에서 보내온 데이터 insert

```
app.post('/', (req, res) => {
    const sql = "INSERT INTO users SET ?"

    con.query(sql, req.body, function (err, result, fields) {
        if (err) throw err;
        console.log(result);
        res.send('등록이 완료되었습니다.');
    });
});
```





데이터 추가 시도를 하고 정상적으로 완료되면
브라우저 화면에 등록이 완료되었다고 표시

ejs 설정

```
npm install ejs
```

HTML 및 javascript를 함께 작성할 수 있는
템플릿 엔진 ejs를 이용

```
const express = require('express')
const app = express()
const port = 3000
const path = require('path')
const bodyParser = require('body-parser')
const ejs = require('ejs')
```

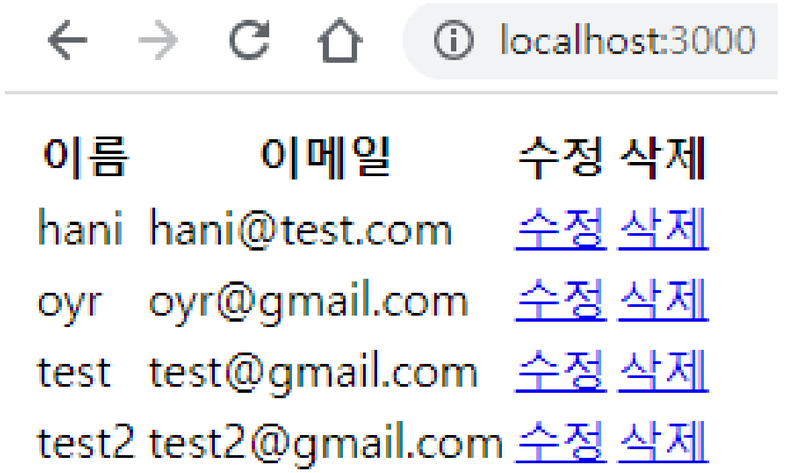
```
app.set('view engine', 'ejs');
```

```
app.get('/', (req, res) => {
    const sql = "select * from users";
    con.query(sql, function (err, result, fields) {
        if (err) throw err;
        res.render('index', { users: result });
    });
});
```

“/(루트)” 접근이 발생하면 select 문이 실행되고,
실행 결과는 ejs 파일에 전달

ejs

```ejs
<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="UTF-8">
    <title>User List</title>
</head>

<body>
    <p><a href="/create">사용자 추가</a></p>
    <table>
        <tr>
            <th>이름</th>
            <th>이메일</th>
            <th>수정</th>
            <th>삭제</th>
        </tr>
        <% users.forEach(function (value) { %>
        <tr>
            <td><%= value.name %></td>
            <td><%= value.email %></td>
            <td><a href="/edit/<%= value.id %>">수정</td>
                <td><a href="/delete/<%= value.id %>">삭제</td>
        </tr>
        <% }); %>
    </table>
</body>

</html>
```



브라우저에서 “/(루트)”에 접속하면,
지금까지 테이블에 추가한 사용자 정보 목록이 표시됨

delete

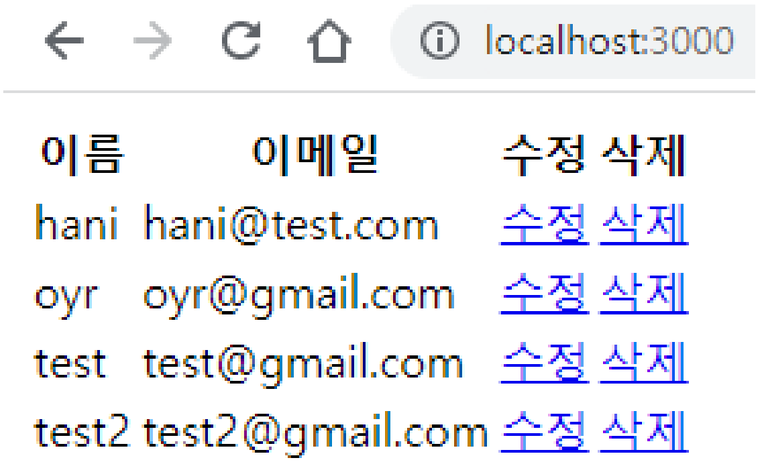
라우팅 /delete/:id 를 추가

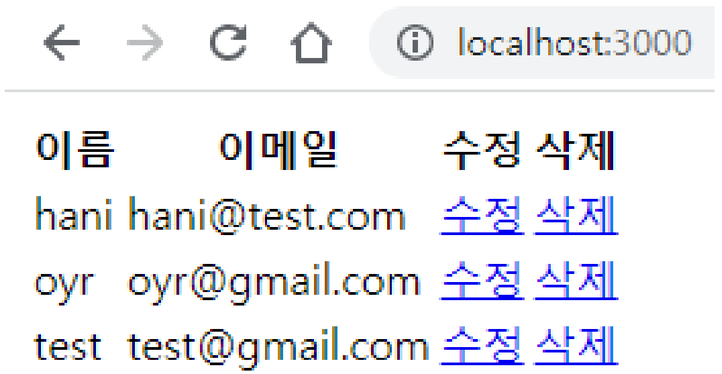
:id에 들어오는 값은 브라우저에서 보낸 req.params.id에서 얻을 수 있음

WHERE절의 id 값에 req.params.id를 설정하면 그 id를 가진 행이 삭제됨

```
app.get('/delete/:id', (req, res) => {
    const sql = "DELETE FROM users WHERE id =?";
    con.query(sql, [req.params.id], function (err, result, fields) {
        if (err) throw err;
        console.log(result)
        res.redirect('/');
    })
});
```

```
<td><a href="/delete/<%= value.id %>">삭제</td>
```





update

```
app.post('/update/:id', (req, res) => {
    const sql = "UPDATE users SET ? WHERE id =" + req.params.id;
    con.query(sql, req.body, function (err, result, fields) {
        if (err) throw err;
        console.log(result);
        res.redirect('/');
    });
});
```

라우팅에 /update/:id를 추가
delete와 마찬가지로 :id에 들어가는 값은 req.params.id에서 얻음
WHERE 절에서 id 값에 req.params.id를 설정하고
해당 id의 행을 업데이트함
업데이트 내용은 insert 문에서 사용한 것처럼 req.body를 사용함

```
app.get('/edit/:id', (req, res) => {
    const sql = "SELECT *FROM users WHERE id =?";
    con.query(sql, [req.params.id], function (err, result, fields) {
        if (err) throw err;
        res.render('edit', { user: result });
    });
});
```

업데이트 화면에 라우팅 /edit/:id를 추가
select문 및 id를 이용하여 업데이트할 행의 현재 값을 가져옴
브라우저에 표시할 폼은 취득한 값을 전달할 ejs 파일을 사용하여 만듦
ejs파일에 변수 user를 사용하여 얻은 값을 전달

update

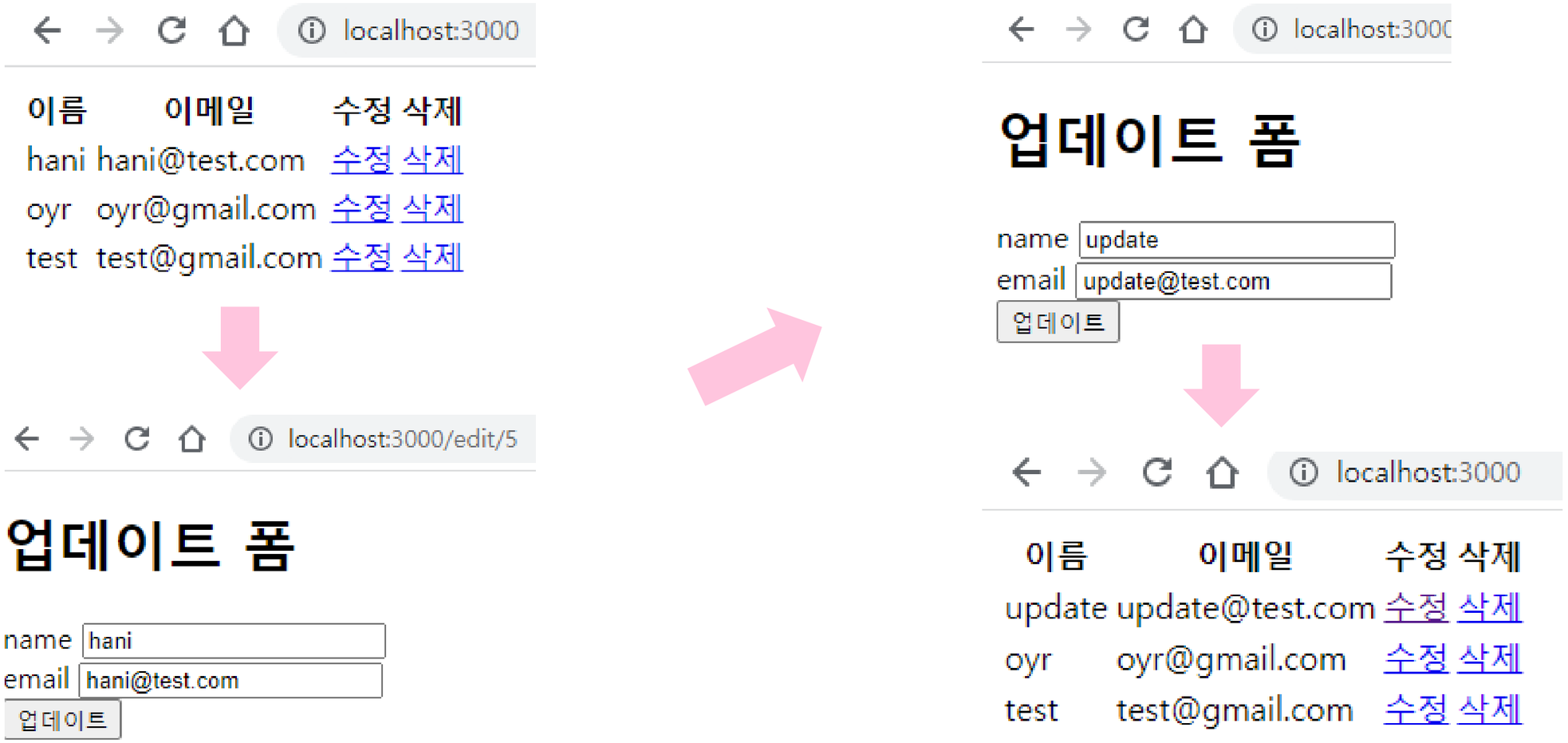
```ejs
<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="UTF-8">
    <title>User Edit</title>
</head>

<body>
    <h1>업데이트 폼</h1>
    <form action="/update/<%= user[0].id %>" method="POST">
        name <input type="text" name="name" value="<%= user[0].name %>"><br>
        email <input type="text" name="email" value="<%= user[0].email %>"><br>
        <button type="submit">업데이트</button>
    </form>
</body>

</html>
```

edit.ejs에선 select 문에서 취득한 데이터를 갖는 변수 user를 이용

input 요소의 value에 name, email의 현재 값을 확인하고,

user[0].name, user[0].email을 각각 설정

update


localhost:3000
이름 이메일 수정 삭제
hani hani@test.com 수정 삭제
oyr oyr@gmail.com 수정 삭제
test test@gmail.com 수정 삭제
localhost:3000/edit/5
업데이트 폼
name hani
email hani@test.com
업데이트
localhost:3000
업데이트 폼
name update
email update@test.com
업데이트
localhost:3000
이름 이메일 수정 삭제
update update@test.com 수정 삭제
oyr oyr@gmail.com 수정 삭제
test test@gmail.com 수정 삭제

감사합니다!