사용자 인증을 통한 게시판 만들기

INTRODUCE

김시훈

- 세종대학교 컴퓨터공학과
- shuni93@naver.com

황성영

- 광운대학교 컴퓨터공학과
- sy930503@naver.com

INTRODUCE

강의자료

https://github.com/shuni4481/node_note

INDEX

LEARN

Http 모듈을 사용하여 서버 리스너 구동

BUT

라우팅, 세션관리 등 일일이 구현하기 힘듦

SO

Express, Koa, Hapi 등의 웹 프레임워크 사용

LEARN

Http 모듈을 사용하여 서버 리스너 구동

BUT

라우팅, 세션관리 등 일일이 구현하기 힘듦

SO

Express, Koa, Hapi 등의 웹 프레임워크 사용

Express

LEARN

1 빠르고 간편한 웹 프레임 워크

- Express는 Node.js를 위한 빠르고 간편한 웹 프레임워크

2 가장 많이 사용하는 Node.js의 웹 프레임 워크

- Hapi, Koa등 다양한 웹 프레임 워크가 있지만 현재까지 가장 많이 사용하는 웹 프레임 워크

3 Node.js의 핵심 모듈인 http와 connect 컴포넌트 기반

- 개발자들은 특정 프로젝트에 필요한 라이브러리를 어떤 것이든 자유롭게 선택할 수 있으며, 이는 개발자들에게 유연함과 수준 높은 맞춤식 구성을 보장(미들웨어 구조)

LEARN

```
$npm install express
$npm install express-generator
```

express 설치
express-generator 설치(애플리케이션 생성기 도구)

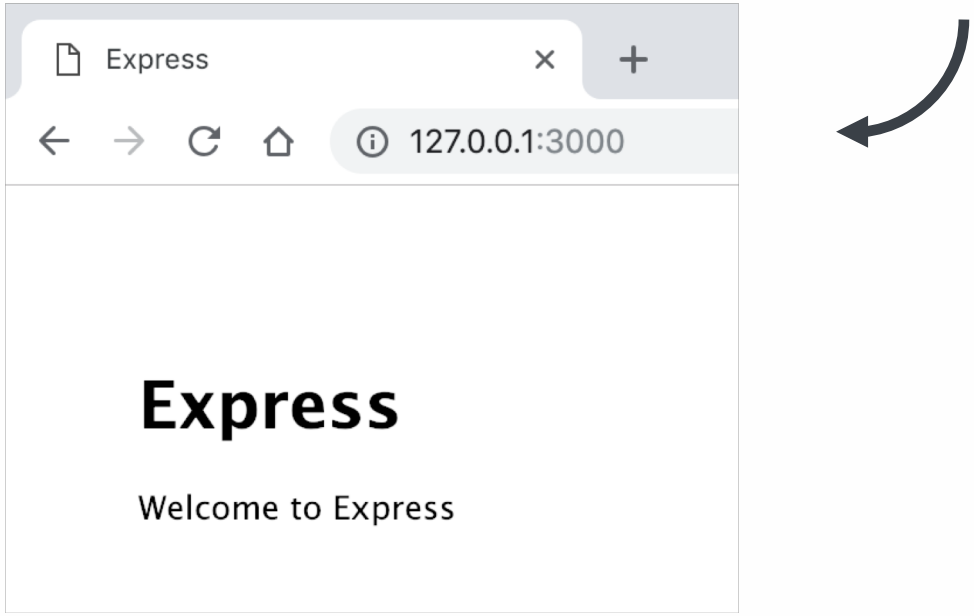
```
$cd nodejs
$express –e note
$cd note
```

note라는 프로젝트 생성

DOIT

```
$npm install
$npm start
```

note project에 생성된 package.json의 모듈 설치 후 서버 실행



LEARN

MVC ?

LEARN

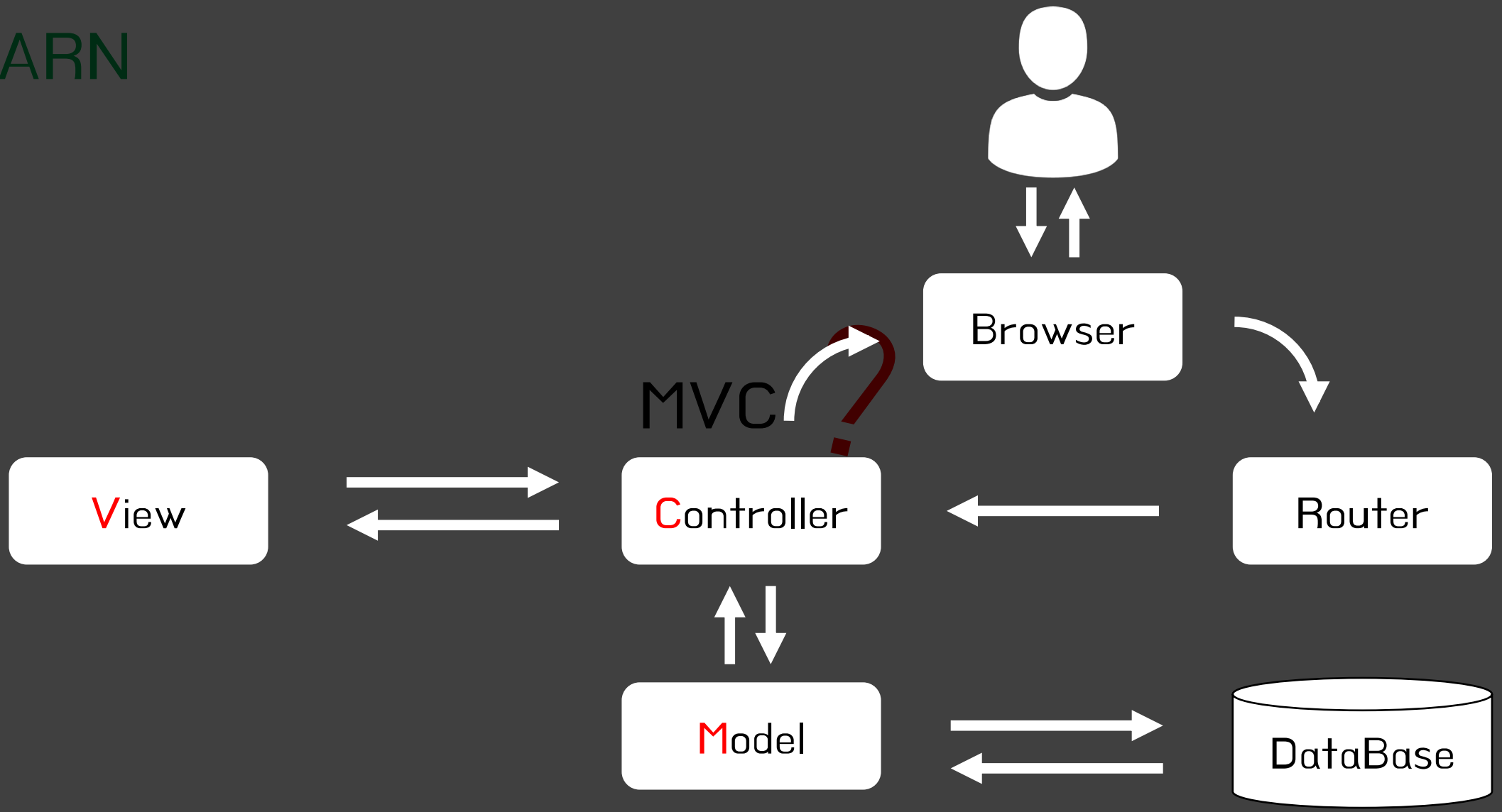
MVC ?

Model - 백그라운드에서 동작하는 로직을 처리(DB)

View - 사용자가 보게 될 결과 화면(웹 페이지)

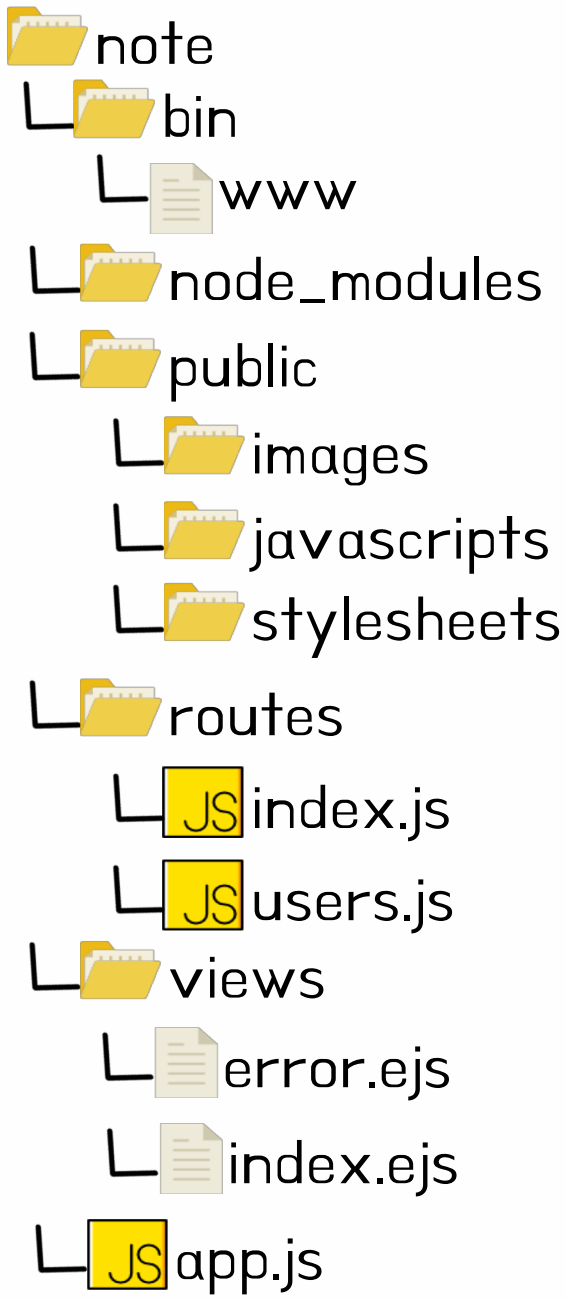
Controller - 사용자의 입력 처리와 흐름 제어(중간자)

LEARN



LEARN

기본적인 프로젝트 구조



LEARN

index.ejs

```
<title><%= title %></title>
```

JS index.js

```
/* GET home page. */
router.get('/', function(req, res, next) {
        res.render('index', { title: 'Express' });
});
```

<%= 객체 %>

라우팅은 URI(또는 경로) 및 특정한 HTTP 요청 메소드(GET, POST 등)인 특정 엔드포인트에 대한 클라이언트 요청에 애플리케이션이 응답하는 방법을 결정하는 것을 말합니다.

LEARN

HTTP 요청 메소드

METHOD	역할
GET	GET을 통해 해당 리소스를 조회 및 정보 참조
POST	POST를 통해 해당 URL을 요청하면 리소스 생성
PUT	PUT을 통해 해당 리소스를 수정
DELETE	DELETE를 통해 해당 리소스를 삭제

JS app.js

```
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
                                        .
                                        .
app.use('/', indexRouter);
app.use('/users', usersRouter);
```

JS index.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
        res.render('index', { title: 'Express' });
});

module.exports = router;
```

JS users.js

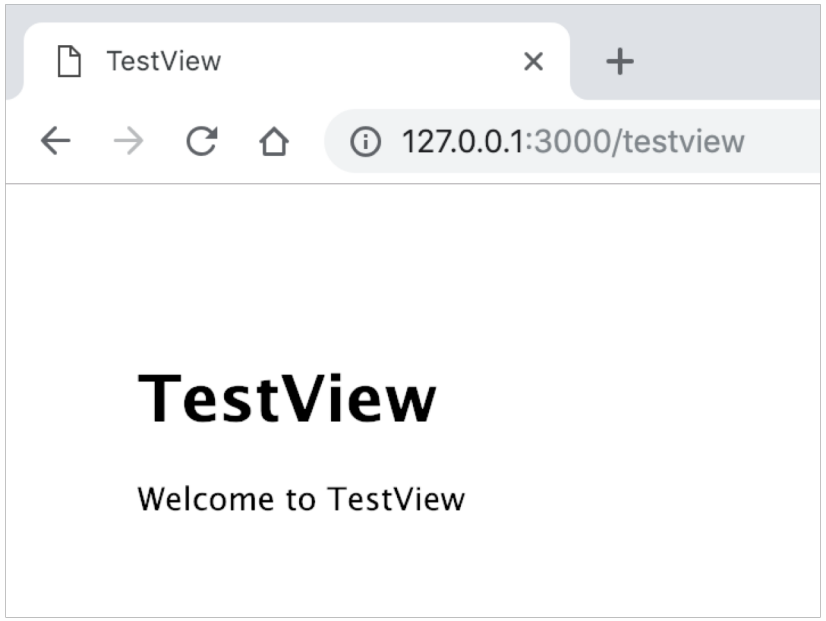
```
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res, next) {
        res.send('respond with a resource');
});

module.exports = router;
```

DOIT

/testview라는 경로로 접속했을 때,
기본 view title을 TestView로 전송하여 띄우기



LEARN

URL을 이용하여 데이터 전송

http://localhost:3000/users?number=4

쿼리스트링

http://expressjs.com/ko/api.hml#reqquery

LEARN

URL을 이용하여 데이터 전송

JS users.js

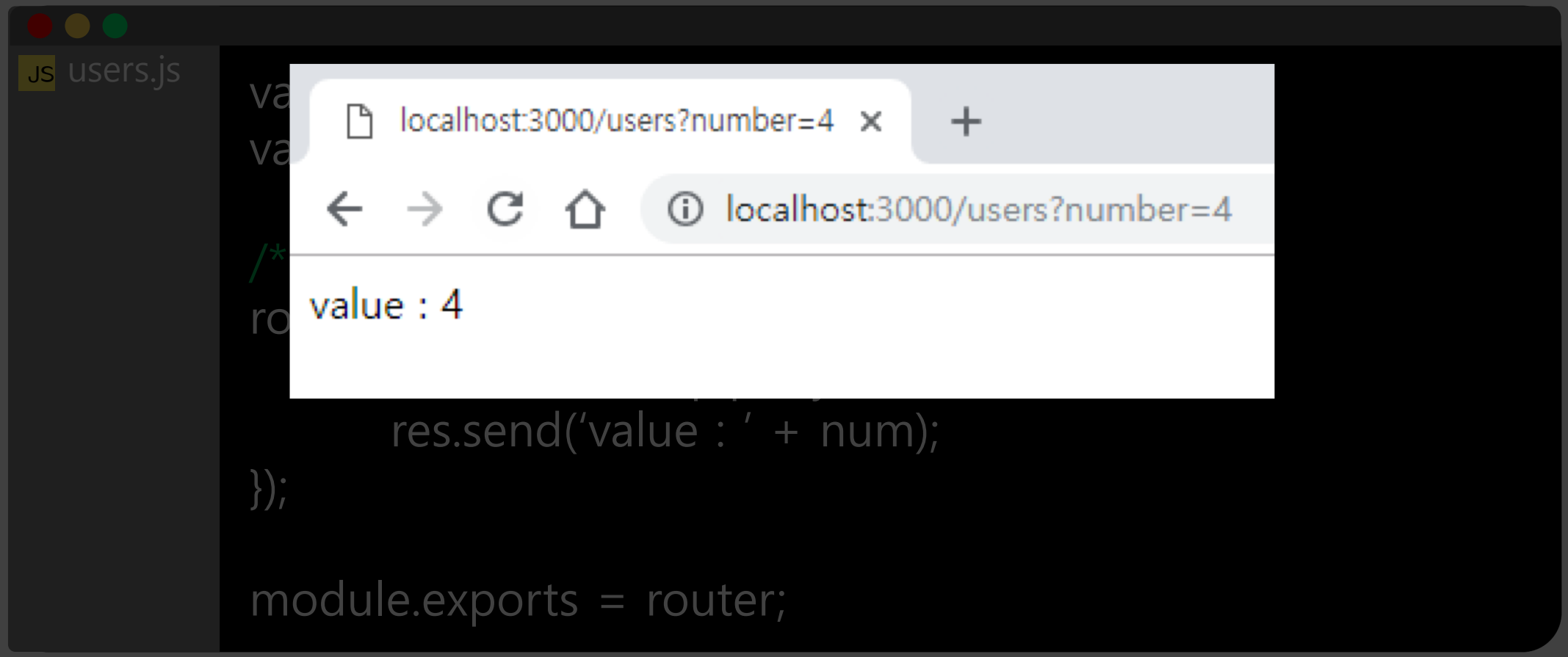
```
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res, next) {
        var num = req.query.number;
        res.send('value : ' + num);
});

module.exports = router;
```

LEARN

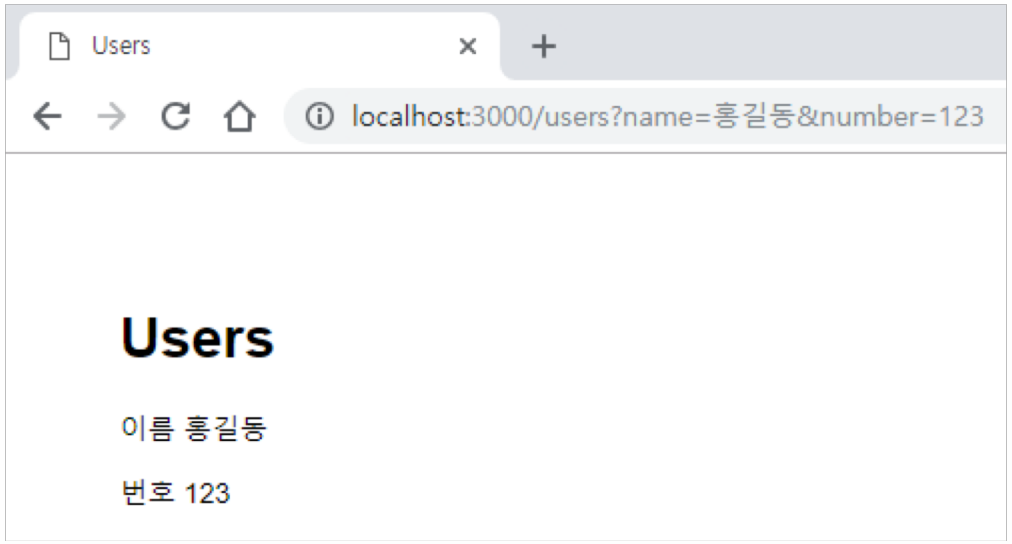
URL을 이용하여 데이터 전송



DOIT

http://localhost:3000/users?name=홍길동&number=123

위의 url을 입력 했을 때, users.ejs 에서 홍길동, 123을 출력하도록 코드를 작성하시오.



Semantic URL

http://localhost:3000/users?name=홍길동&number=123

vs

http://localhost:3000/users/홍길동/123

Semantic URL

JS users.js

```
router.get('/:name/:number', function(req, res, next) {
        var name = req.params.name;
        var number = req.params.number;
        res.render('users', {name: name, number: number});
});

module.exports = router;
```

GET

- 클라이언트로가 입력한 query의 이름과 값이 결합되어 스트링 형태로 서버에 전달

- 한번의 데이터 요청 시, 제한이 없음

- URL에 입력한 정보가 노출

- 서버에서 어떤 데이터를 가져와서 보여줄 때 사용

LEARN

POST

- 클라이언트에서 데이터를 인코딩(encoding) 후,
서버 측에서 디코딩(decoding) 해서 사용

- 한번의 데이터 요청 시, 제한이 없음

- URL에 입력한 정보가 노출되지 않음

- 서버상의 데이터 값이나 상태를 바꾸기 위해서 사용

users.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <title>Users</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <form action="/test" method="GET">
        <h1>users</h1>
        <p>이름<input type="text" name="name"></p>
        <p>번호<input type="text" name="number"></p>
        <p><input type="submit"></p>
    </form>
  </body>
</html>
```

LEARN

GET

JS test.js

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

GET / POST

test.js

```
router.get('/', function(req, res, next) {
    var name =   req.query.name;
    var number = req.query.number;
    res.send("Get₩n" + name + "₩n" + number);
});

router.post('/', function(req, res, next) {
    var name =   req.body.name;
    var number = req.body.number;
    res.send("post₩n" + name + "₩n" + number);
});
```

HTTP URI(Uniform Resource Identifier)를 통해 자원(Resource)을 명시하고, HTTP Method(POST, GET, PUT, DELETE)를 통해 해당 자원에 대한 CRUD Operation을 적용하는 것을 의미

REST API 구성요소

Resource
Method
Representation

CRUD	HTTP	Route
User들의 목록을 표시	GET	/user
User 하나의 내용을 표시	GET	/user/:id
User를 생성	POST	/user
User를 수정	PUT	/user/:id
User를 삭제	DELETE	/user/:id

https://meetup.toast.com/posts/92

JSON

```
{
  “firstName”: “Hwang”,
  “lastName”: “SeongYoung”,
  “email”: sy930503@naver.com
  “hobby”: [“soccer”, “game”]
}
```

경량(Lightweight)의 DATA-교환 형식

JSON 표현식은 사람과 기계 모두 이해하기 쉬움

최근에는 JSON이 XML을 대체해서 데이터 전송 등에 많이 사용

특정 언어에 종속되지 않으며, 대부분의 프로그래밍 언어에서 JSON 포맷의 데이터를 핸들링 할 수 있는 라이브러리를 제공

LEARN

POSTMAN

https://www.getpostman.com

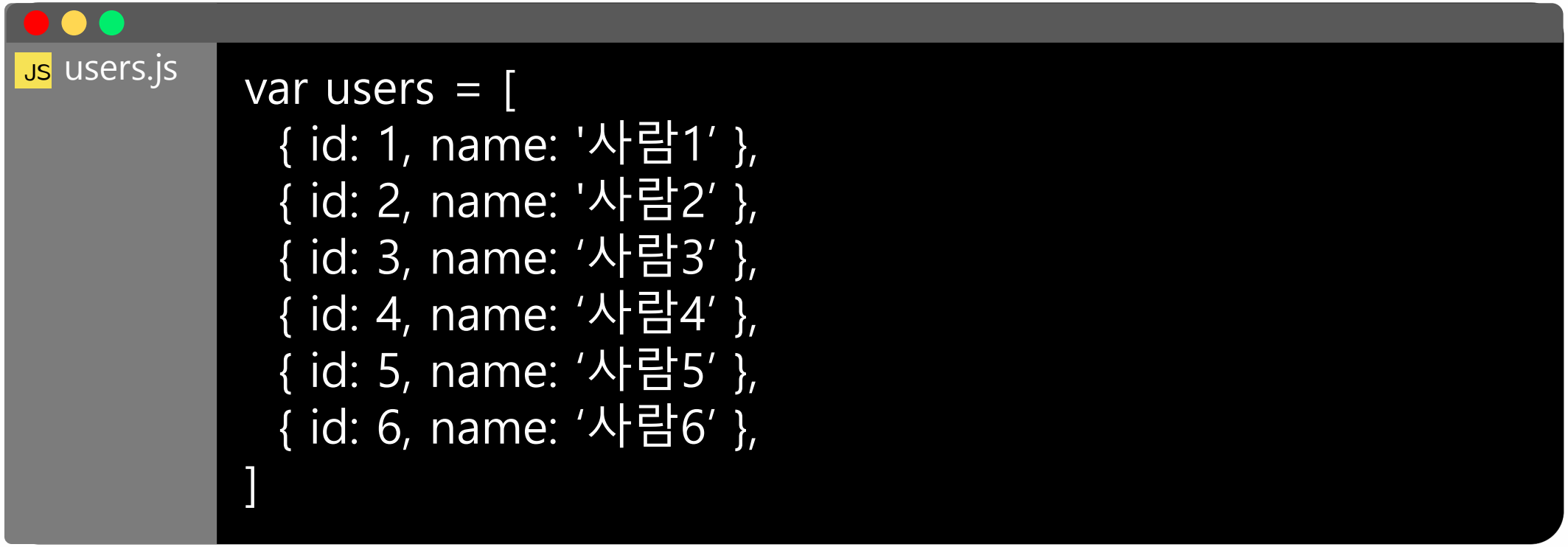
API 개발을 빠르고 쉽게,
개발된 API를 테스트할 수 있고,
팀원들 간 공유를 할 수 있게 해주는 플랫폼

DOIT

JS users.js

```
router.get('/:id', function(req, res){
  // 코드 작성!
});
router.post('/', function(req, res){
  // 코드 작성!
});
router.put('/:id', function(req, res){
  // 코드 작성!
});
router.delete('/:id', function(req, res){
  // 코드 작성!
});
```

DOIT


JS users.js
var users = [
{ id: 1, name: '사람1' },
{ id: 2, name: '사람2' },
{ id: 3, name: '사람3' },
{ id: 4, name: '사람4' },
{ id: 5, name: '사람5' },
{ id: 6, name: '사람6' },
]

DOIT

JS users.js

```
router.get('/', (req, res) => res.json(users));
router.get('/:id', function(req, res) {
        var id = parseInt(req.params.id, 10);

        if(!id)
                return res.status(400).json({error: 'Incorrect id'});

        var user = users.filter(user => user.id === id)[0]

        if (!user)
                return res.status(404).json({error: 'Unknown user'});

        return res.json(user);
});
```

DOIT

JS users.js

GET http://localhost:3000/users Params Send

Authorization Headers Body Pre-request Script Tests

Type No Auth

Body Cookies Headers (6) Test Results Status: 200 OK

Pretty Raw Preview JSON


```
[
    {
        "id": 1,
        "name": "사람1"
    },
    {
        "id": 2,
        "name": "사람2"
    },
    {
        "id": 3,
        "name": "사람3"
    },
    {
        "id": 4,
        "name": "사람4"
    },
    {
        "id": 5,
        "name": "사람5"
    },
    {
        "id": 6,
        "name": "사람6"
    }
]
```

DOIT

JS users.js

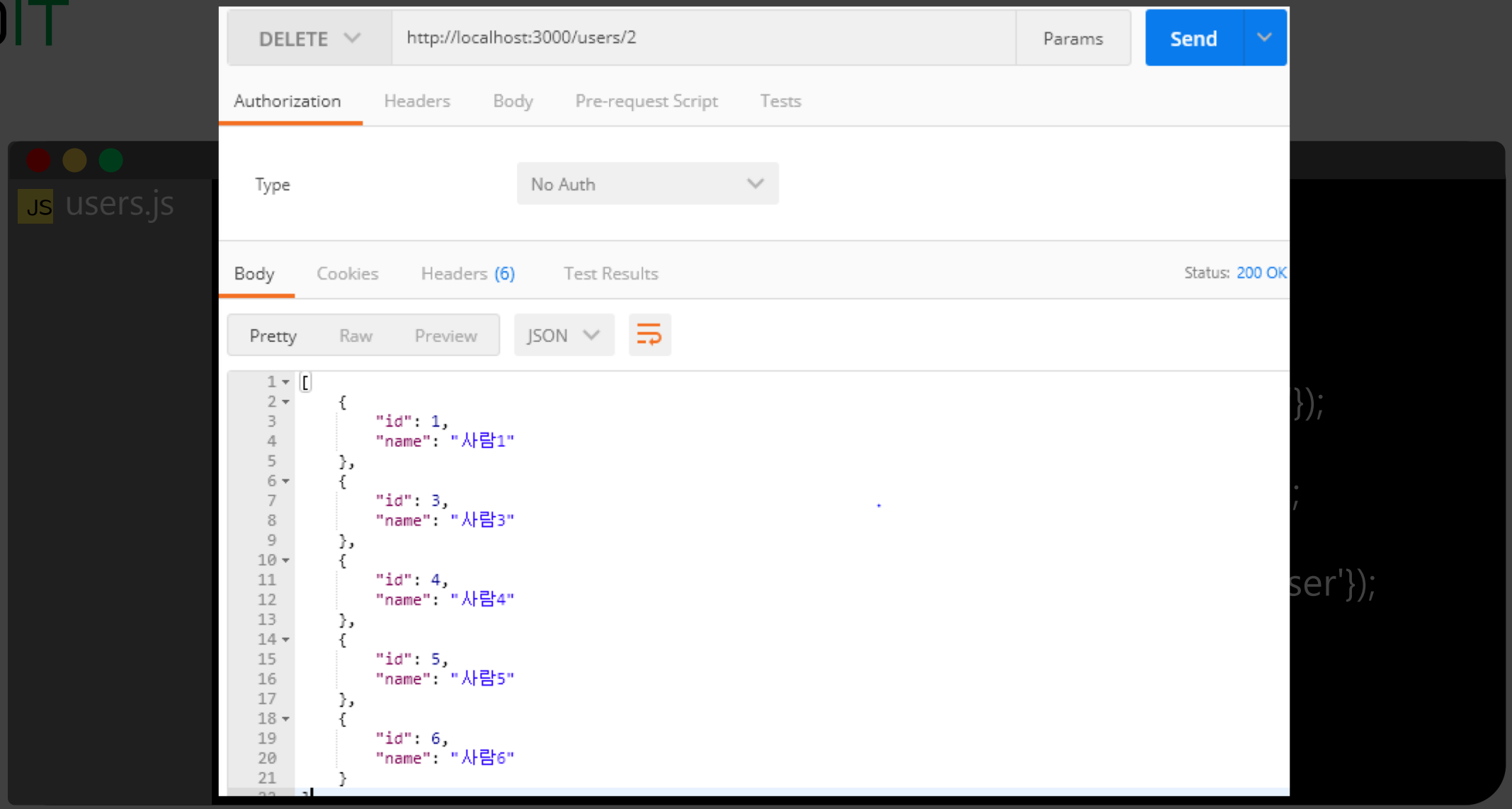
```
router.delete('/:id', function(req, res){
        var id = parseInt(req.params.id, 10);

        if (!id)
                return res.status(400).json({error: 'Incorrect id'});

        var userIdx = users.findIndex(user => user.id === id);
        if (userIdx === -1)
                return res.status(404).json({error: 'Unknown user'});

        users.splice(userIdx, 1);
        return res.json(users);
});
```

DOIT


DELETE
http://localhost:3000/users/2
Params
Send
Authorization
Headers
Body
Pre-request Script
Tests
Type
No Auth
Body
Cookies
Headers (6)
Test Results
Status: 200 OK
Pretty
Raw
Preview
JSON
[
{
"id": 1,
"name": "사람1"
},
{
"id": 3,
"name": "사람3"
},
{
"id": 4,
"name": "사람4"
},
{
"id": 5,
"name": "사람5"
},
{
"id": 6,
"name": "사람6"
}
users.js

THANK YOU