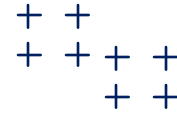
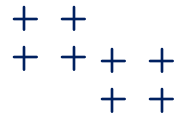


# 전북대 소중해유(You)

**Express 프레임워크 기초:**  
Express 설정과 웹 서버 구축



전북대학교  
SOFTWARE중심대학사업단

SW중심대학



정보통신기획평가원



# 목차

1/ Express란?

2/ Express 설정 및 세팅

3/ Express를 활용한 구현 예제

## Express란?

- Node.js를 위한 빠르고, 간결하고, 개방적이고, 확장성이 좋은 웹 프레임워크!!

Express 4.19.2

Node.js를 위한 빠르고 개방적인 간결한 웹 프레임워크

```
$ npm install express --save
```

### 웹 애플리케이션

Express는 웹 및 모바일 애플리케이션을 위한 일련의 강력한 기능을 제공하는 간결하고 유연한 Node.js 웹 애플리케이션 프레임워크입니다.

### API

자유롭게 활용할 수 있는 수많은 HTTP 유틸리티 메소드 및 미들웨어를 통해 쉽고 빠르게 강력한 API를 작성할 수 있습니다.

### 성능

Express는 기본적인 웹 애플리케이션 기능으로 구성된 얇은 계층을 제공하여, 여러분이 알고 있고 선호하는 Node.js 기능을 모호하게 만들지 않습니다.

## Express란? (주요 기능)

### 1. 라우팅(Routing)

- 라우팅 정의: 클라이언트 요청 URL에 따라 특정 코드 블록을 실행하는 기능
- HTTP 메서드 지원: GET, POST, PUT, DELETE 등 다양한 HTTP 메서드를 지원
- 동적 라우트: URL 매개변수를 사용하여 동적 라우트를 설정 가능

### 2. 미들웨어(Middleware)

- 미들웨어 정의: 요청과 응답 사이에서 특정 작업을 수행하는 코드
- 내장 미들웨어: `express.json()` [JSON 요청 처리], `express.urlencoded()` [URL 인코딩된 요청 처리]

## Express란? (주요 기능 - 미들웨어)

### 2. 미들웨어(Middleware)

- Express에서 요청(request)과 응답(response) 사이의 중간에 위치한 함수들로, 클라이언트의 요청을 처리하거나 응답을 보내기 전에 특정 작업을 수행할 수 있는 방법을 제공
- Express의 미들웨어는 웹 애플리케이션의 다양한 기능을 모듈화하고, 코드의 재사용성과 관리성을 높이는 데 중요한 역할
- 요청 처리:** 클라이언트의 요청을 처리하고, 필요에 따라 요청 객체(req)나 응답 객체(res)를 수정 가능
- 응답 처리:** 요청이 끝난 후 클라이언트에게 응답을 보내기 전에 특정 작업을 수행



- 경로별 미들웨어 적용:** 특정 경로에서만 동작하도록 설정할 수 있어, 애플리케이션의 특정 부분에만 미들웨어를 적용 가능
- 로깅, 인증, 오류 처리 등:** 로깅, 사용자 인증, 에러 처리, 정적 파일 제공 등 다양한 작업을 미들웨어를 통해 처리 가능

## Express란? (주요 기능)

### 3. 정적 파일 제공(Static File Serving)

- 정적 파일 서빙: Express는 `express.static` 미들웨어를 통해 CSS, 이미지, JS 파일 등 정적 파일을 제공 가능

### 4. 템플릿 엔진 지원(Template Engine)

- 템플릿 엔진 사용: HTML을 동적으로 생성하기 위해 여러 템플릿 엔진(Pug, EJS 등)을 지원
- 템플릿 파일 렌더링: 템플릿 파일을 작성하고, Express에서 렌더링하여 클라이언트에 전달 가능

## Express란? (주요 기능)

### 5. 폼 데이터 및 파일 업로드 처리

- 폼 데이터 처리: `express.urlencoded()` 미들웨어를 사용해 POST 요청의 폼 데이터를 처리
- 파일 업로드: 파일 업로드를 처리하기 위해 `multer`와 같은 추가 미들웨어를 사용 가능

### 6. RESTful API 구축

- CRUD API: Express는 다양한 HTTP 메서드를 사용해 쉽게 RESTful API를 구축 가능
- JSON 응답 처리: `res.json()`을 통해 JSON 형식의 응답을 쉽게 반환 가능

## Express란? (주요 기능)

- 8. 요청 매개변수 처리(Query Parameters and URL Parameters)
- 9. 쿠키 및 세션 관리(Cookies and Sessions)
- 10. 환경 변수 및 설정 관리(Configuration and Environment Variables)
- 11. 보안(Security)/ 확장성 및 모듈화(Scalability and Modularity)



## Express 설정 및 세팅

### • Express 설치

```
(base) ksl@MacStudio untitled2 % npm install express

added 64 packages, and audited 65 packages in 426ms

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
(base) ksl@MacStudio untitled2 %
```

1. “npm install express” 명령어를 통한 설치
2. 별다른 문제가 없다면 왼쪽 그림처럼 정상적으로 설치 됨!!

## Express 설정 및 세팅

### • Express 설치

```
(base) ksl@MacStudio ur
added 64 packages, and
12 packages are looking
run `npm fund` for de
found 0 vulnerabilities
(base) ksl@MacStudio ur
```

```
node_modules library root
> .bin
> accepts
> array-flatten
> body-parser
> bytes
> call-bind
> content-disposition
> content-type
> cookie
> cookie-signature
> debug
> define-data-property
> depd
> destroy
> ee-first
> encodeurl
> es-define-property
> es-errors
> escape-html
> etag
> express
```

#### 1. "npm install express" 명령어를 통한 설치

2. 별다른 문제가 없다면 왼쪽 그림처럼 정상적으로 설치 됨!!
3. 설치가 되면 node\_modules (패키지 관리) 파일 밑에 express 라이브러리가 설치된 것을 확인할 수 있음!

## Express 설정 및 세팅

### • Express 설치

```
1  const express = require('express');
2  const app = express();
3
4  app.get('/', (req, res) => {
5    res.send('Hello, World!');
6  });
7
8  const port = 3000;
9  app.listen(port, () => {
10    console.log(`Server running at http://localhost:${port}/`);
11  });
```

#### 1. "npm install express" 명령어를 통한 설치

2. 별다른 문제가 없다면 왼쪽 그림처럼 정상적으로 설치 됨!!
3. 설치가 되면 node\_modules (패키지 관리) 파일 밑에 express 라이브러리가 설치된 것을 확인할 수 있음!
4. 이후, javascript 파일을 만들어서 express를 Import 해서 사용 가능!
5. Import 방법: "**const express = require('express');**"
6. express application 사용 방법: "**const app = express();**"

## Express 설정 및 세팅

### • Express 설치

basicServer.js

```
1  const express = require('express');
2  const app = express();
3
4  app.get('/', (req, res) => {
5    res.send('Hello, World!');
6  });
7
8  const port = 3000;
9  app.listen(port, () => {
10    console.log(`Server running at http://localhost:${port}/`);
11  });
```

```
(base) ksl@MacStudio untitled2 % node basicServer.js
Server running at http://localhost:3000/
```

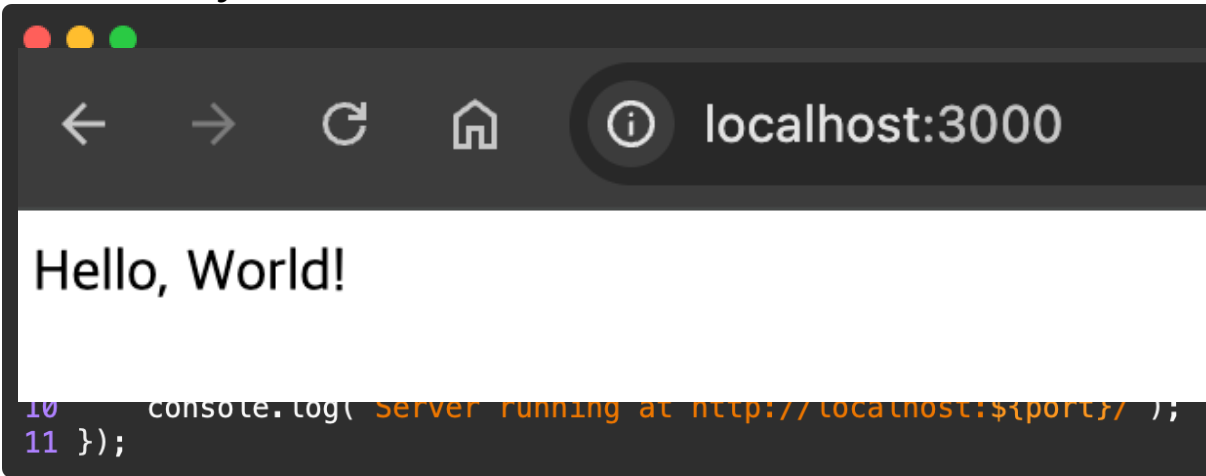
#### 1. "npm install express" 명령어를 통한 설치

2. 별다른 문제가 없다면 왼쪽 그림처럼 정상적으로 설치 됨!!
3. 설치가 되면 node\_modules (패키지 관리) 파일 밑에 express 라이브러리가 설치된 것을 확인할 수 있음!
4. 이후, javascript 파일을 만들어서 express를 Import 해서 사용 가능!
5. Import 방법: `"const express = require('express');"`
6. express application 사용 방법: `"const app = express();"`
7. 실행 방법: `node basicServer.js`

## Express 설정 및 세팅

### • Express 설치

basicServer.js



```
(base) ksl@MacStudio untitled2 % node basicServer.js  
Server running at http://localhost:3000/
```

#### 1. "npm install express" 명령어를 통한 설치

2. 별다른 문제가 없다면 왼쪽 그림처럼 정상적으로 설치 됨!!

3. 설치가 되면 node\_modules (패키지 관리) 파일 밑에 express 라이브러리가 설치된 것을 확인할 수 있음!

4. 이후, javascript 파일을 만들어서 express를 Import 해서 사용 가능!

5. Import 방법: `"const express = require('express');"`

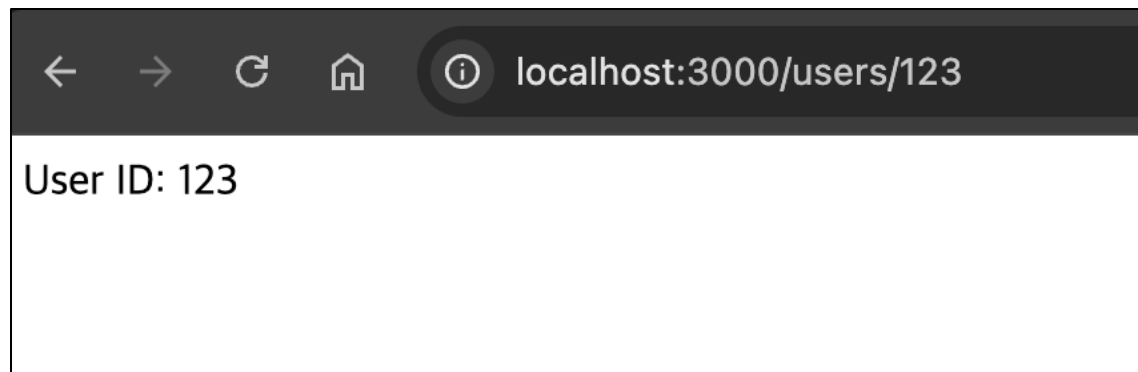
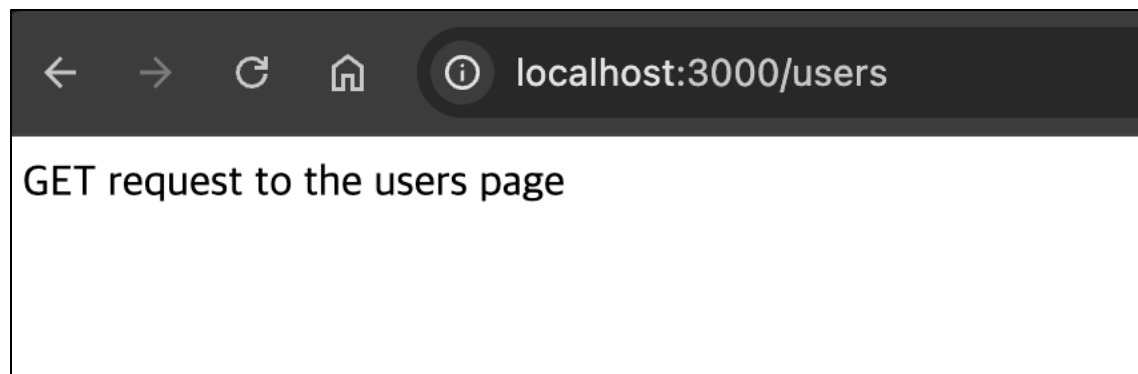
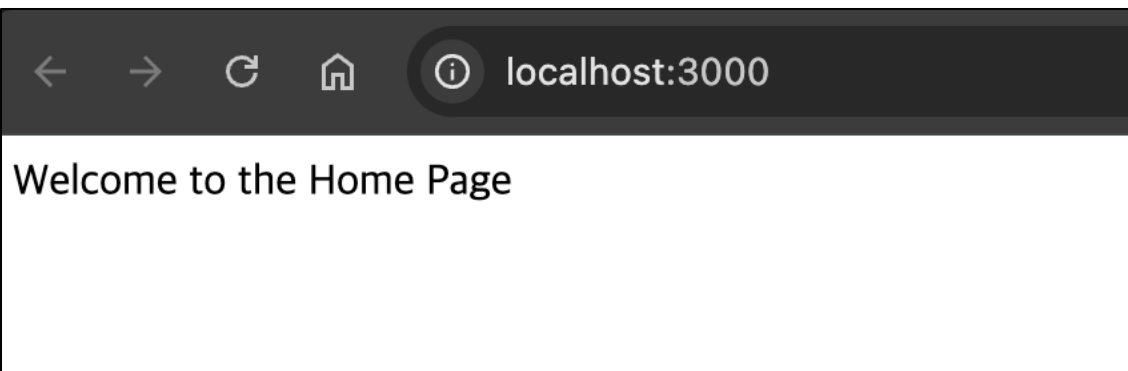
6. express application 사용 방법: `"const app = express();"`

7. 실행 방법: `node basicServer.js`

8. 웹 브라우저에서 <http://localhost:3000>으로 접속하면 확인 가능!

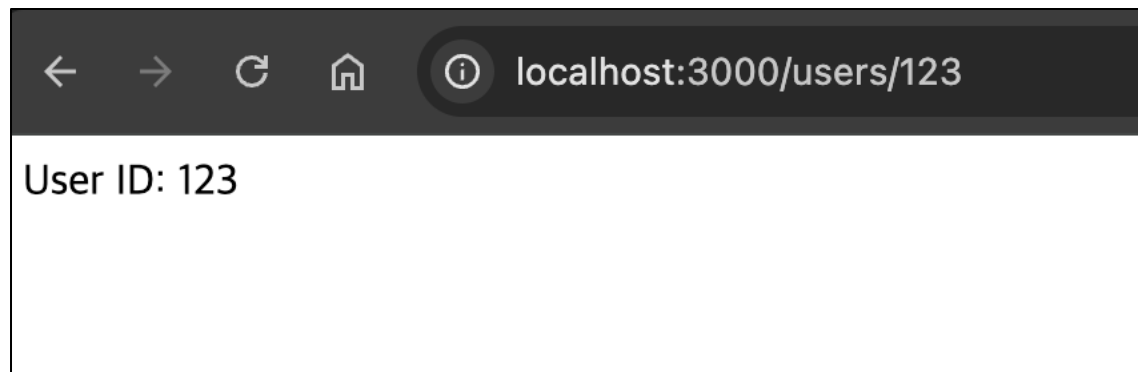
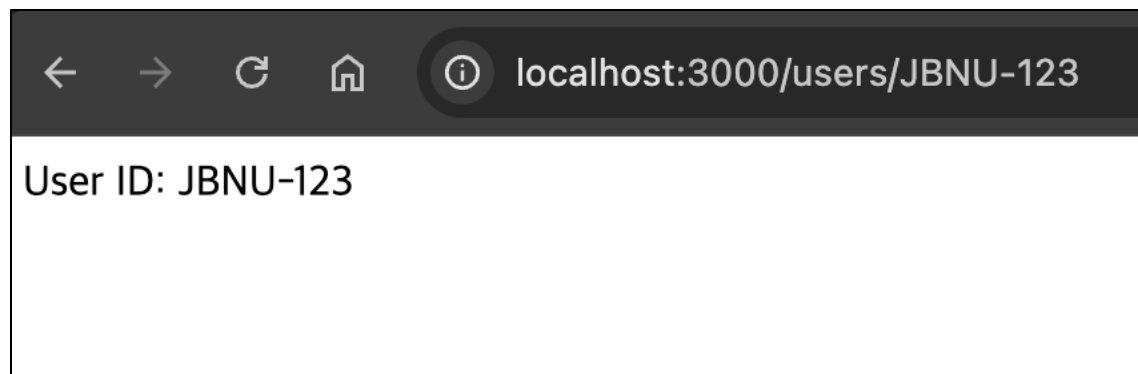
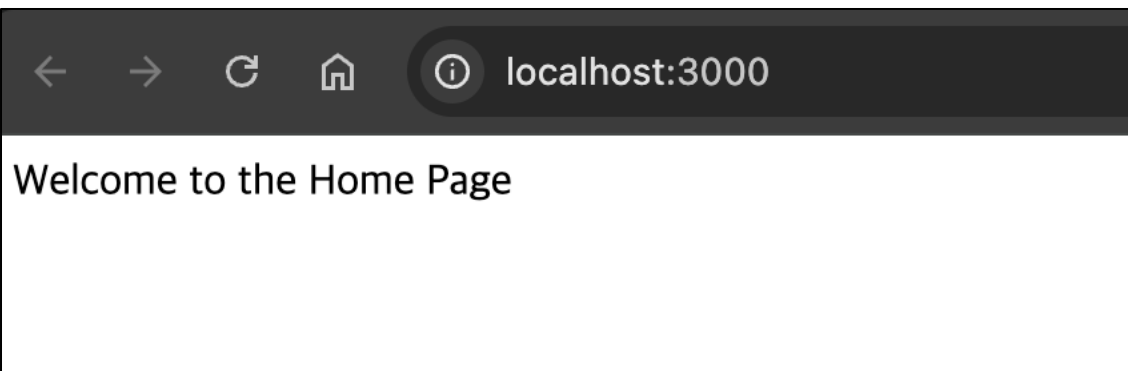
## Express를 활용한 구현 예제 - 라우팅(Routing)

```
1 const express = require('express');
2 const app = express();
3
4 // 기본 GET 요청 라우트
5 app.get('/', (req, res) => {
6   res.send('Welcome to the Home Page');
7 });
8
9 // GET 요청 라우트
10 app.get('/users', (req, res) => {
11   res.send('GET request to the users page');
12 });
13
14 // POST 요청 라우트
15 app.post('/users', (req, res) => {
16   res.send('POST request to the users page');
17 });
18
19 // 동적 라우팅
20 app.get('/users/:id', (req, res) => {
21   res.send(`User ID: ${req.params.id}`);
22 });
23
24 const port = 3000;
25 app.listen(port, () => {
26   console.log(`Server running at http://localhost:${port}/`);
27 });
```



## Express를 활용한 구현 예제 - 라우팅(Routing)

```
1 const express = require('express');
2 const app = express();
3
4 // 기본 GET 요청 라우트
5 app.get('/', (req, res) => {
6   res.send('Welcome to the Home Page');
7 });
8
9 // GET 요청 라우트
10 app.get('/users', (req, res) => {
11   res.send('GET request to the users page');
12 });
13
14 // POST 요청 라우트
15 app.post('/users', (req, res) => {
16   res.send('POST request to the users page');
17 });
18
19 // 동적 라우팅
20 app.get('/users/:id', (req, res) => {
21   res.send(`User ID: ${req.params.id}`);
22 });
23
24 const port = 3000;
25 app.listen(port, () => {
26   console.log(`Server running at http://localhost:${port}/`);
27 });
```



## \* POSTMAN (<https://www.postman.com>)

### Docume APIs together

Over 30 million developers use Postman. Get started by signing up or downloading the desktop app.

[Sign Up for Free](#)

Download the desktop app for



The screenshot displays the Postman web interface. The top navigation bar includes links for Product, Pricing, Enterprise, Resources and Support, and Public API Network, along with a search bar and buttons for Contact Sales, Sign In, and Sign Up for Free. A yellow banner at the top of the workspace contains the Korean text: "클릭하면 이전 페이지로 가고 누르고 있으면 방문 기록이 나타납니다." (Click to go to the previous page, and press and hold to show visit history).

The main workspace shows a REST client setup for the Notion API. The collection is named "Notion API" and contains a folder "Databases" with a request "GET Retrieve a database". The request is configured with the URL "https://api.notion.com/v1/databases/id" and a query parameter "id" with the value "({DATABASE\_ID})". The response is displayed in the "Body" tab, showing a JSON object with the following structure:

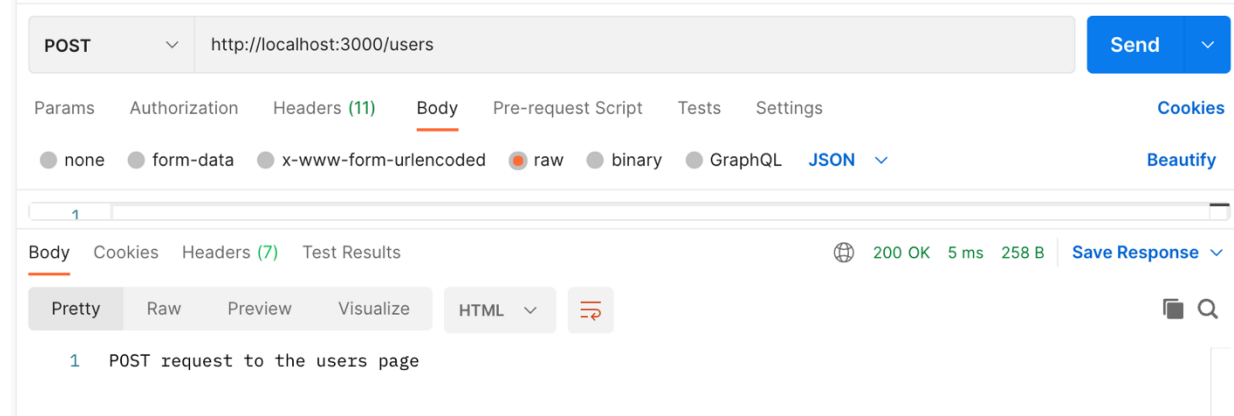
```
1 {
2   "Publisher": {
3     "id": "%3E%24Pb",
4     "name": "Publisher",
5     "type": "select",
6     "select": {
7       "options": [
8         {
9           "id": "c5ee489a-f307-4176-99ee-6e424fa89afa",
10          "name": "NYT",
11          "color": "default"
12        }
13      ]
14    }
15  }
16 }
```

The right-hand panel shows the "Documentation" tab, which provides information about the API endpoint, including its description, authorization requirements, and request headers.



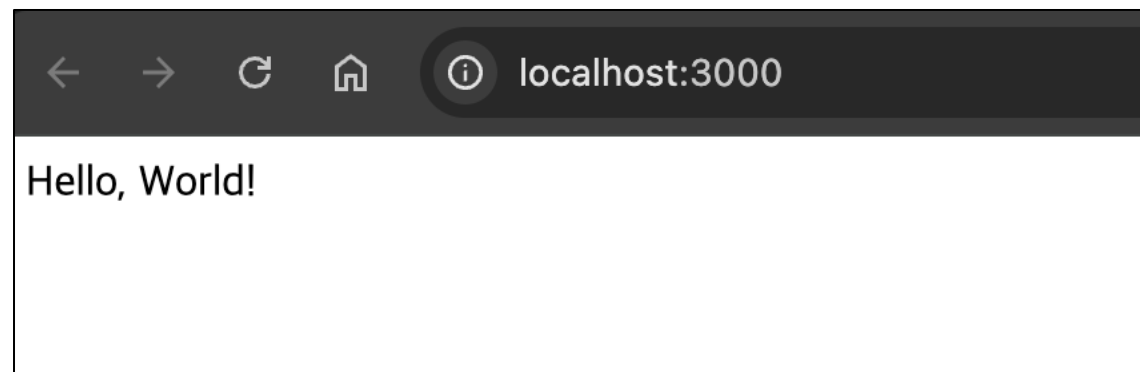
## Express를 활용한 구현 예제 - 라우팅(Routing)

```
1 const express = require('express');
2 const app = express();
3
4 // 기본 GET 요청 라우트
5 app.get('/', (req, res) => {
6   res.send('Welcome to the Home Page');
7 });
8
9 // GET 요청 라우트
10 app.get('/users', (req, res) => {
11   res.send('GET request to the users page');
12 });
13
14 // POST 요청 라우트
15 app.post('/users', (req, res) => {
16   res.send('POST request to the users page');
17 });
18
19 // 동적 라우팅
20 app.get('/users/:id', (req, res) => {
21   res.send(`User ID: ${req.params.id}`);
22 });
23
24 const port = 3000;
25 app.listen(port, () => {
26   console.log(`Server running at http://localhost:${port}/`);
27 });
```



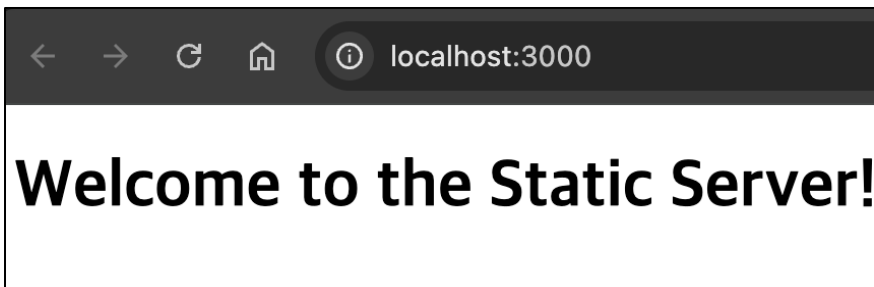
## Express를 활용한 구현 예제 - 미들웨어

```
1 const express = require('express');
2 const app = express();
3
4 // 사용자 정의 미들웨어
5 app.use((req, res, next) => {
6   console.log('mid Time:', Date.now());
7   next();
8 });
9
10 app.get('/', (req, res) => {
11   console.log('app Time:', Date.now());
12   res.send('Hello, World!');
13 });
14
15 const port = 3000;
16 app.listen(port, () => {
17   console.log(`Server running at http://localhost:${port}/`);
18 });
19
```



```
Server running at http://localhost:3000/
mid Time: 1723566481306
app Time: 1723566481307
```

## Express를 활용한 구현 예제 - 정적 파일 제공



staticServer.js

```
1 const express = require('express');
2 const app = express();
3
4 // 정적 파일 제공
5 app.use(express.static('public'));
6
7 app.get('/', (req, res) => {
8   res.send('Welcome to the Static Server!');
9 });
10
11 const port = 3000;
12 app.listen(port, () => {
13   console.log(`Server running at http://localhost:${port}/`);
14 });
15
```

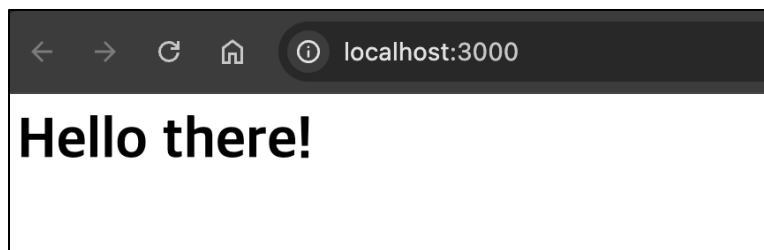
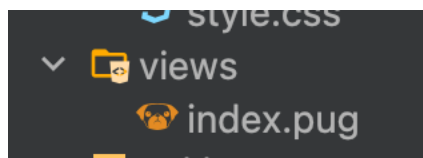
index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Static Server</title>
7   <link rel="stylesheet" href="styles.css">
8 </head>
9 <body>
10 <h1>Welcome to the Static Server!</h1>
11 </body>
12 </html>
13
```

style.css

```
1 body {
2   background-color: #f0f0f0;
3   font-family: Arial, sans-serif;
4 }
5
6 h1 {
7   color: #333;
8   text-align: center;
9 }
10
```

## Express를 활용한 구현 예제 - 템플릿 엔진 사용



templateExample.js

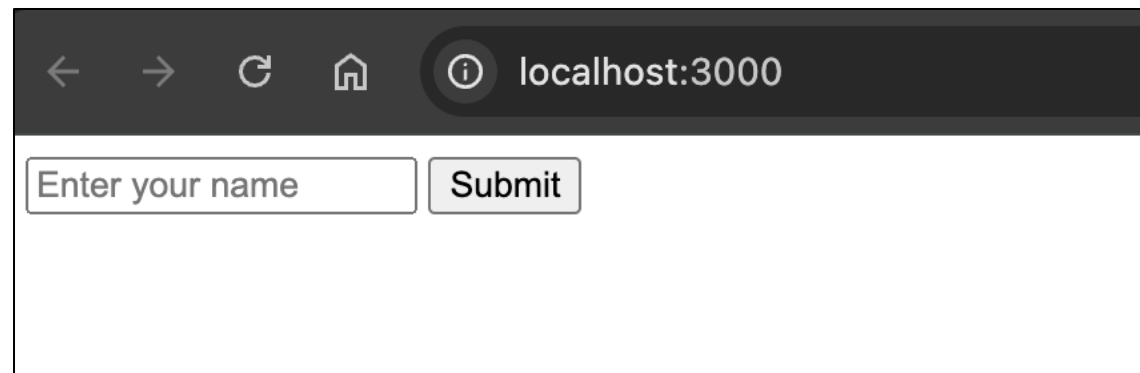
```
1  const express = require('express');
2  const app = express();
3
4  app.set('views engine', 'pug');
5  app.set('views', './views');
6
7  app.get('/', (req, res) => {
8    res.render('index', { title: 'Express', message: 'Hello there!' });
9  });
10
11 const port = 3000;
12 app.listen(port, () => {
13   console.log(`Server running at http://localhost:${port}/`);
14 });
15
```

index.pug

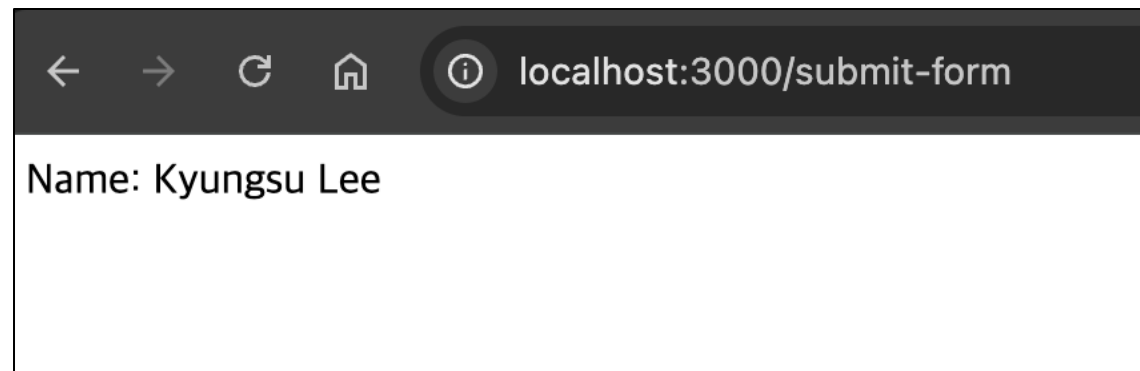
```
1  html
2    head
3      title= title
4    body
5      h1= message
6
```

## Express를 활용한 구현 예제 – 폼 (Form) 데이터 처리

```
1 const express = require('express');
2 const app = express();
3
4 app.use(express.urlencoded({ extended: true }));
5
6 app.get('/', (req, res) => {
7   res.send(`
8     <form method="POST" action="/submit-form">
9       <input type="text" name="name" placeholder="Enter your name" />
10      <button type="submit">Submit</button>
11    </form>
12  `);
13 });
14
15 app.post('/submit-form', (req, res) => {
16   res.send(`Name: ${req.body.name}`);
17 });
18
19 const port = 3000;
20 app.listen(port, () => {
21   console.log(`Server running at http://localhost:${port}/`);
22 });
23
```



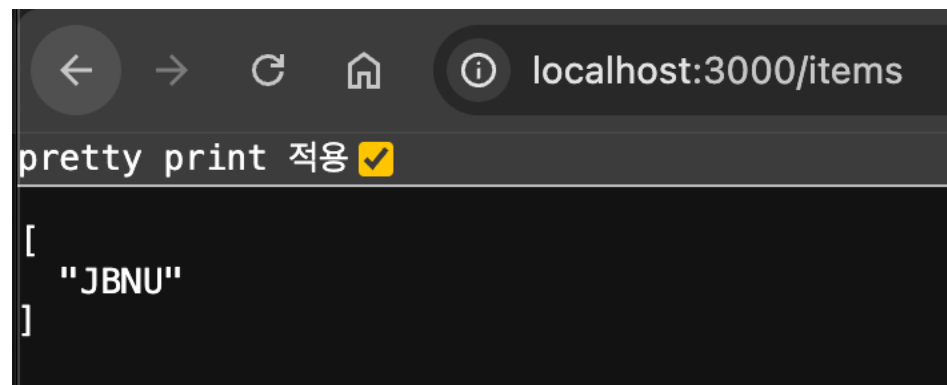
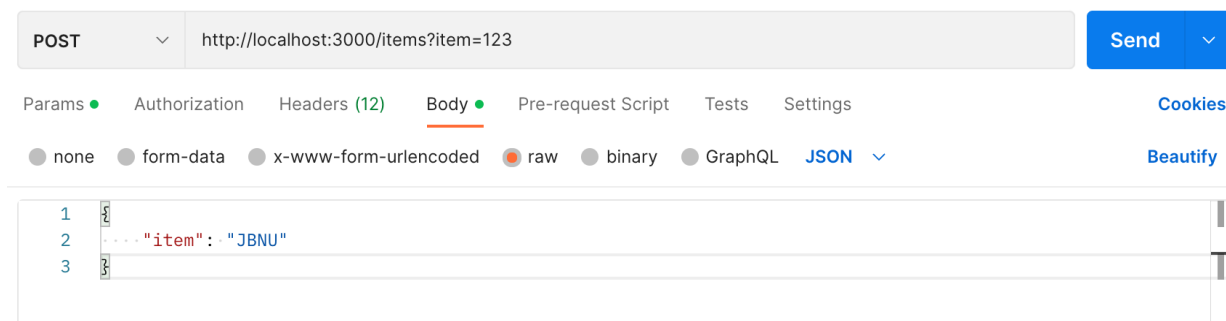
A screenshot of a web browser window. The address bar shows 'localhost:3000'. The page content consists of a text input field with the placeholder text 'Enter your name' and a 'Submit' button to its right.



A screenshot of a web browser window. The address bar shows 'localhost:3000/submit-form'. The page content displays the text 'Name: Kyungsu Lee'.

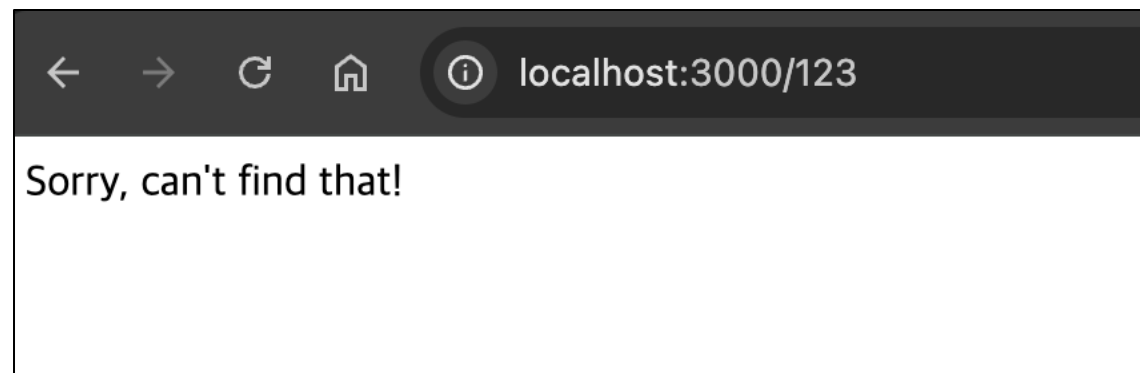
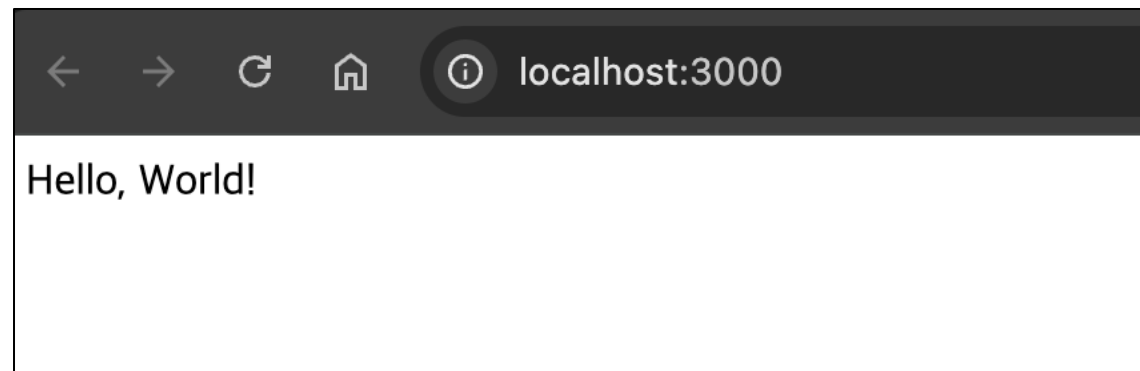
## Express를 활용한 구현 예제 – 간단한 REST API 구축

```
1 const express = require('express');
2 const app = express();
3
4 app.use(express.json());
5
6 let items = [];
7
8 app.get('/items', (req, res) => {
9   res.json(items);
10 });
11
12 app.post('/items', (req, res) => {
13   const newItem = req.body.item;
14   items.push(newItem);
15   res.status(201).json({ message: 'Item added successfully', item: newItem });
16 });
17
18 app.put('/items/:id', (req, res) => {
19   const id = parseInt(req.params.id);
20   items[id] = req.body.item;
21   res.json({ message: 'Item updated successfully', item: items[id] });
22 });
23
24 app.delete('/items/:id', (req, res) => {
25   const id = parseInt(req.params.id);
26   const deletedItem = items.splice(id, 1);
27   res.json({ message: 'Item deleted successfully', item: deletedItem });
28 });
29
30 const port = 3000;
31 app.listen(port, () => {
32   console.log(`API server running at http://localhost:${port}/`);
33 });
34
```



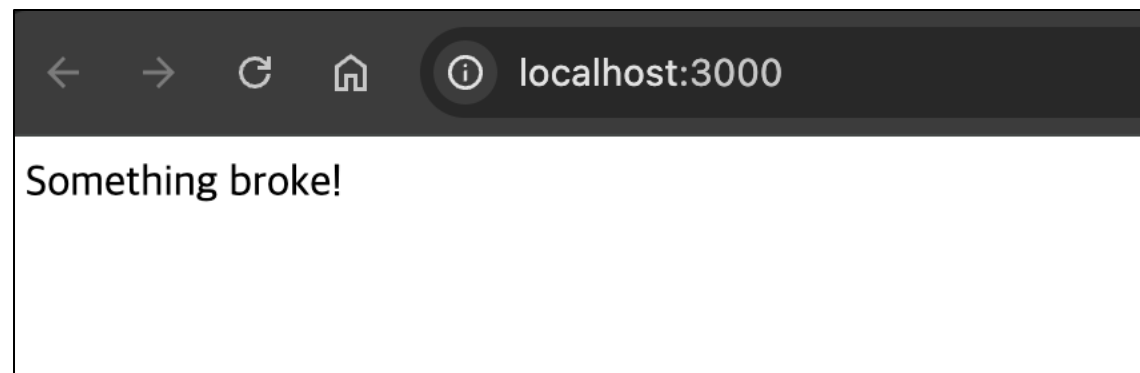
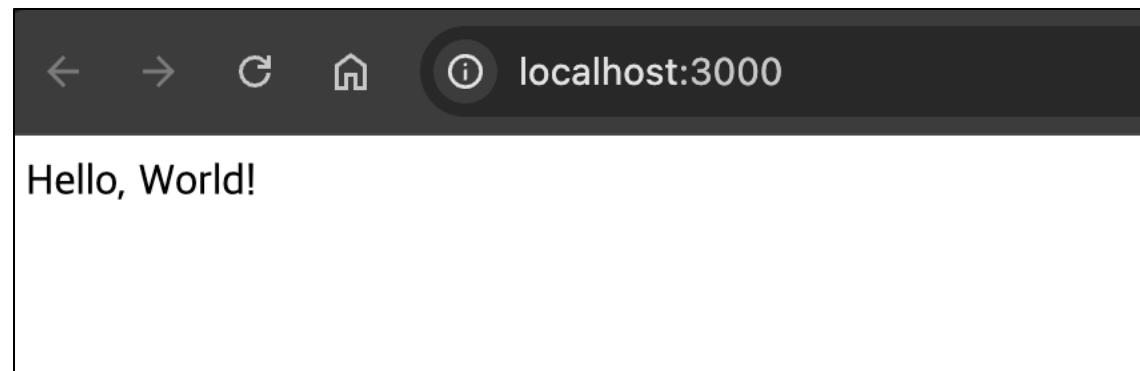
## Express를 활용한 구현 예제 – 에러 처리

```
1  const express = require('express');
2  const app = express();
3
4  // 기본 라우트
5  app.get('/', (req, res) => {
6    res.send('Hello, World!');
7  });
8
9  // 404 에러 처리 미들웨어
10 app.use((req, res, next) => {
11   res.status(404).send("Sorry, can't find that!");
12 });
13
14 // 일반 에러 처리 미들웨어
15 app.use((err, req, res, next) => {
16   console.error(err.stack);
17   res.status(500).send('Something broke!');
18 });
19
20 const port = 3000;
21 app.listen(port, () => {
22   console.log(`Server running at http://localhost:${port}/`);
23 });
24
```



## Express를 활용한 구현 예제 – 에러 처리

```
1  const express = require('express');
2  const app = express();
3
4  // 기본 라우트
5  app.get('/', (req, res) => {
6    throw new Error('Test Error');
7    res.send('Hello, World!');
8  });
9
10 // 404 에러 처리 미들웨어
11 app.use((req, res, next) => {
12   res.status(404).send("Sorry, can't find that!");
13 });
14
15 // 일반 에러 처리 미들웨어
16 app.use((err, req, res, next) => {
17   console.error(err.stack);
18   res.status(500).send('Something broke!');
19 });
20
21 const port = 3000;
22 app.listen(port, () => {
23   console.log(`Server running at http://localhost:${port}/`);
24 });
25
```







# 감사합니다.

- 본 온라인 콘텐츠는 2024년도 과학기술 정보통신부 및 정보통신기획평가원의 'SW중심대학사업' 지원을 받아 제작되었습니다.
- 본 결과물의 내용을 전재할 수 없으며, 인용(재사용)할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원이 지원한 'SW중심대학'의 결과물이라는 출처를 밝혀야 합니다.

