SOEN 287

Server side programming with Node.js (2) Basic functions

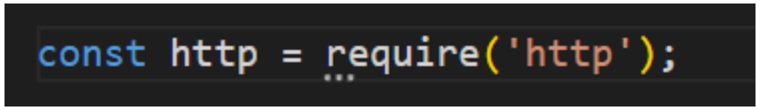
Dr. Yuhong Yan
CSE, Concordia University
Winter, 2024

Creating a Simple HTTP Server

const http = require('http');

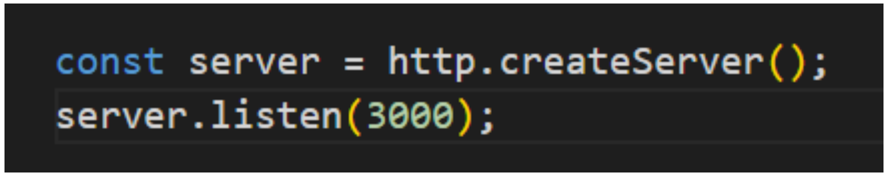
Node.js Simple HTTP Server

- Importing ‘http’ module:
  - We import the built-in Node.js 'http' module that provides functionality for creating web servers and handling HTTP requests.



```
const http = require('http');
```

- Create an HTTP server :
  - We create an HTTP server using the http.createServer() method.

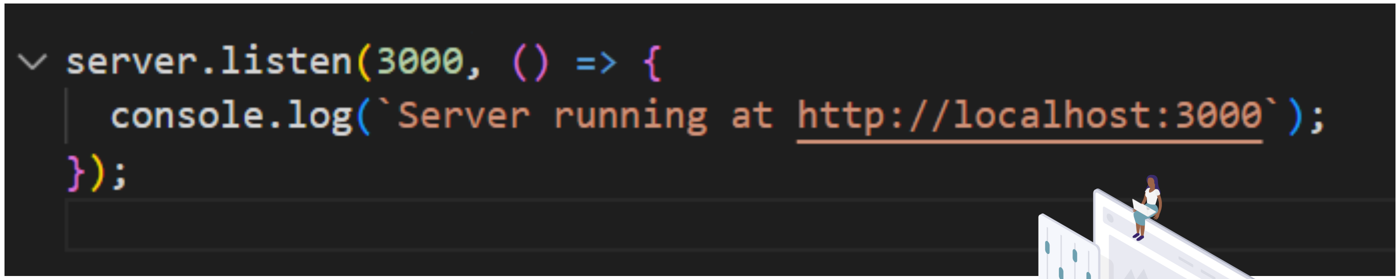


```
const server = http.createServer();
server.listen(3000);
```

const http = require('http');

Node.js HTTP Server

- **Check if server is running**

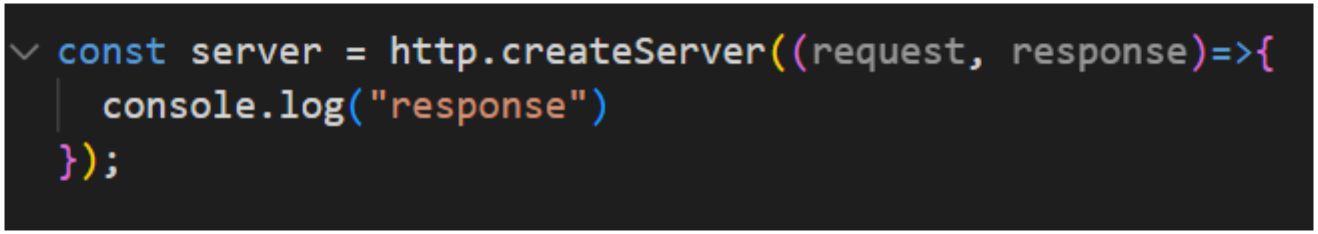


```
server.listen(3000, () => {
  console.log(`Server running at http://localhost:3000`);
});
```

const http = require('http');

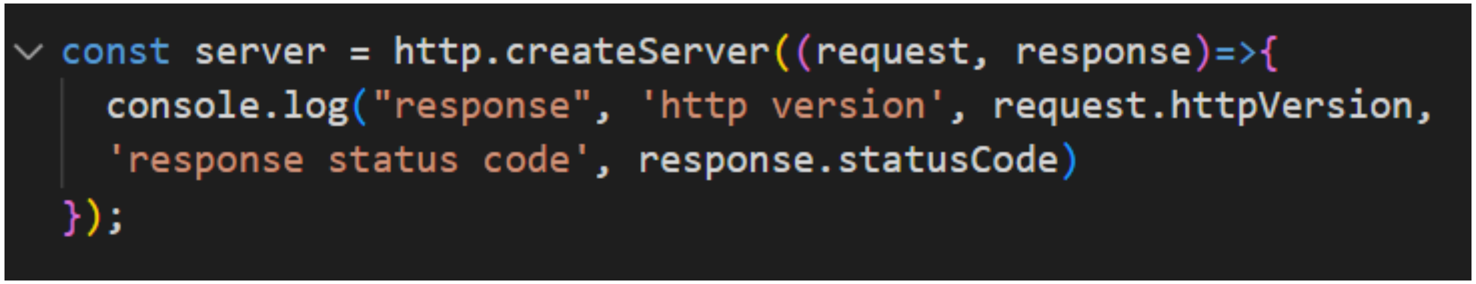
Node.js HTTP Server

- Configure the createServer()



```
const server = http.createServer((request, response)=>{
  console.log("response")
});
```

- Get request HTTP Version

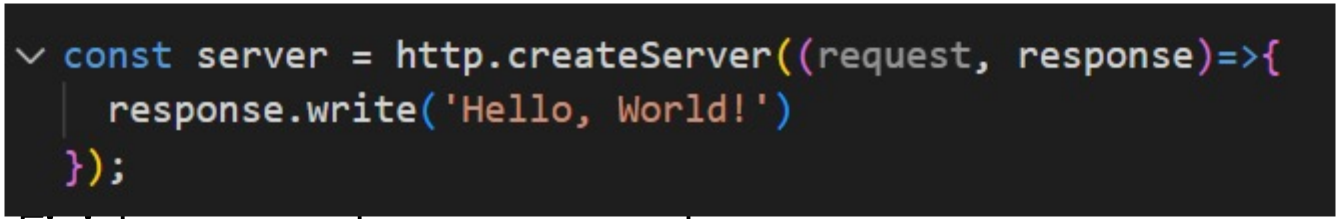


```
const server = http.createServer((request, response)=>{
  console.log("response", 'http version', request.httpVersion,
  'response status code', response.statusCode)
});
```

const http = require('http');

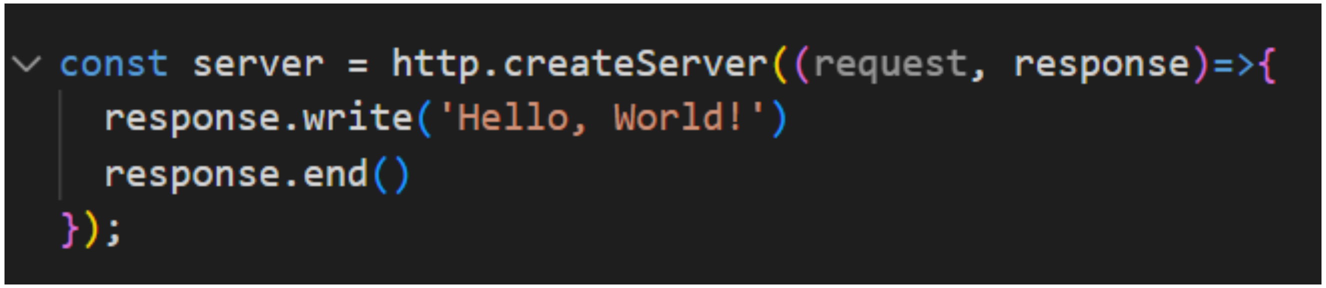
Node.js HTTP Server

- Send response by response.write()



```
const server = http.createServer((request, response)=>{
    response.write('Hello, World!')
});
```

- Finish response by response.end

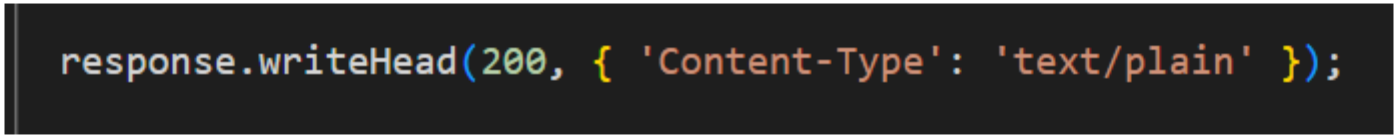


```
const server = http.createServer((request, response)=>{
    response.write('Hello, World!')
    response.end()
});
```

const http = require('http');

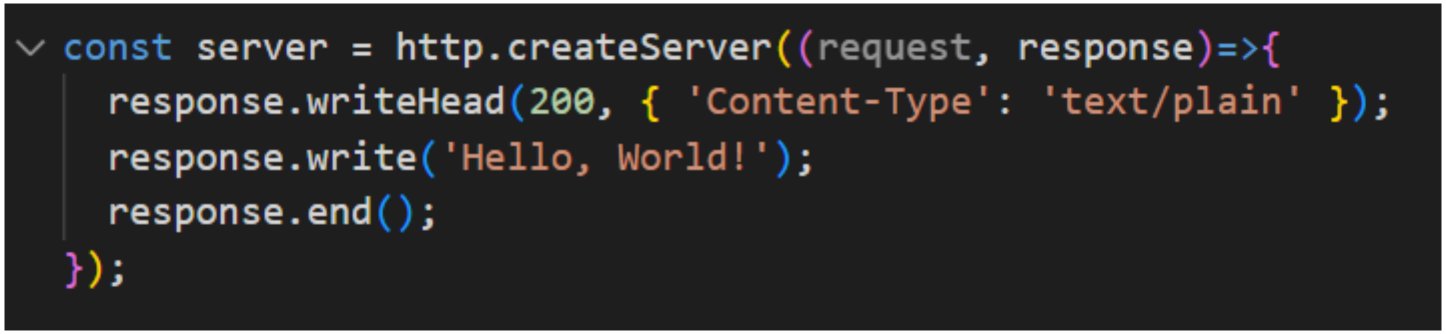
Node.js HTTP Server

- Send Header



```
response.writeHead(200, { 'Content-Type': 'text/plain' });
```

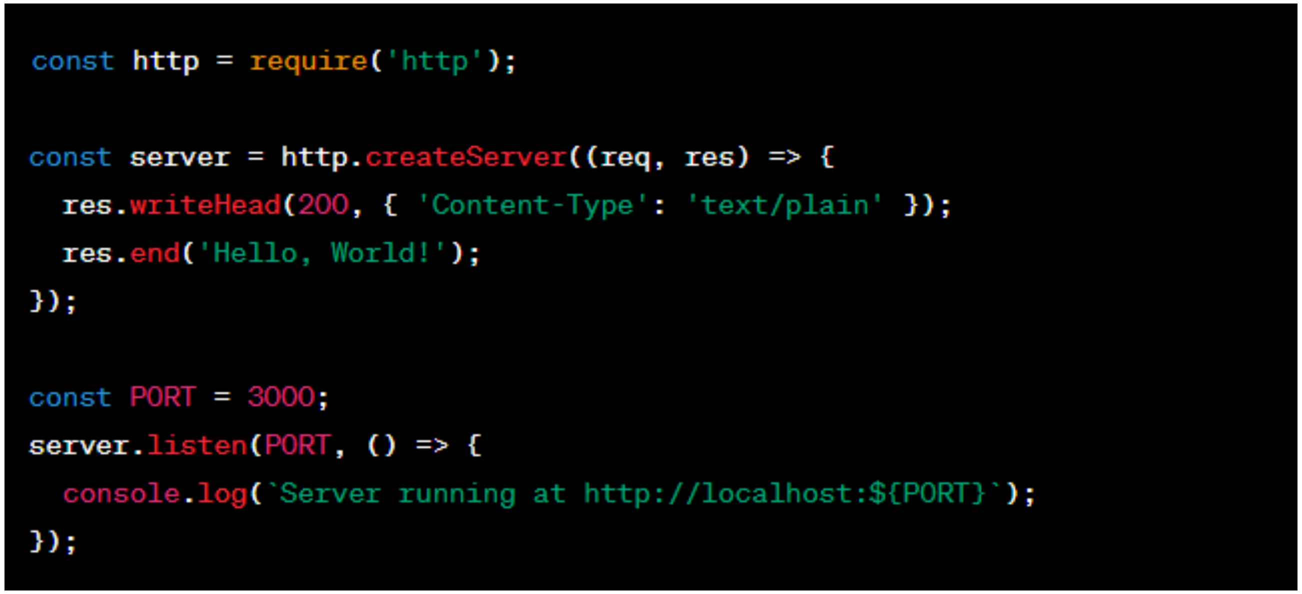
- Final Server configuration



```
const server = http.createServer((request, response)=>{
    response.writeHead(200, { 'Content-Type': 'text/plain' });
    response.write('Hello, World!');
    response.end();
});
```

createServer()

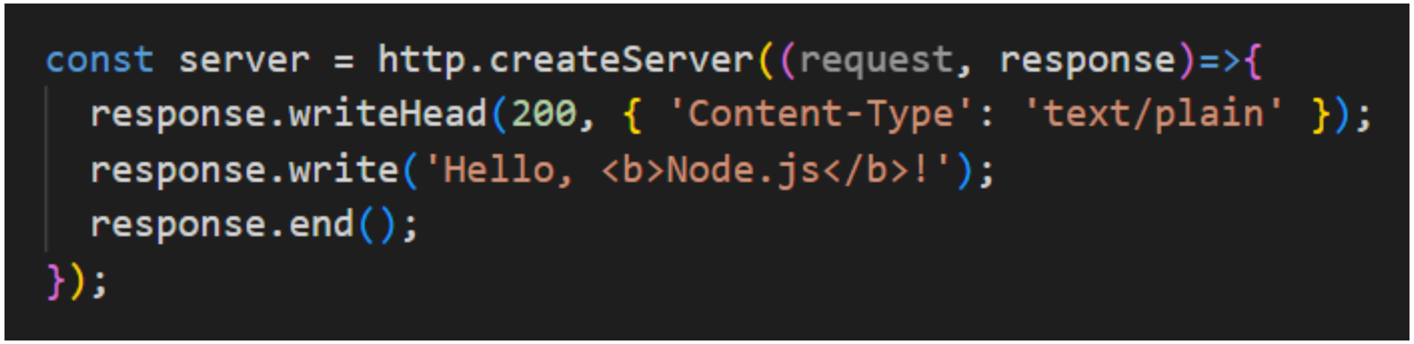
Using the 'http' module's 'createServer()' method to create a basic HTTP server.



```javascript
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, World!');
});

const PORT = 3000;
server.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

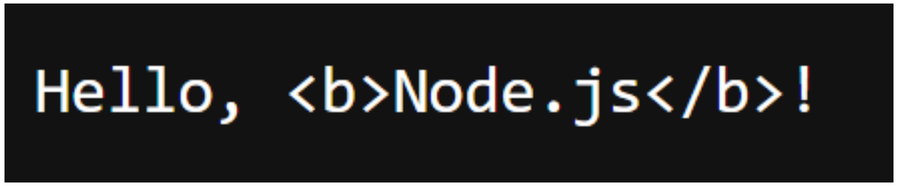
Response HTML

Send HTML in response.write()



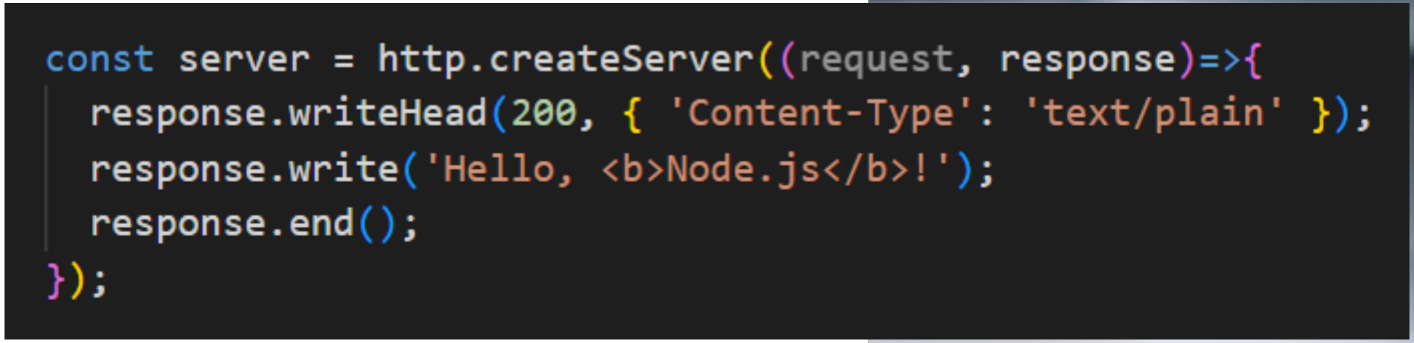
```javascript
const server = http.createServer((request, response)=>{
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  response.write('Hello, <b>Node.js</b>!');
  response.end();
});
```

Result:

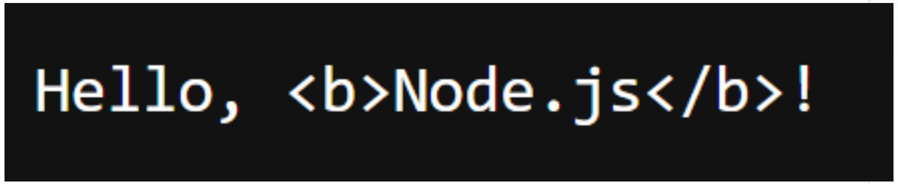


```
Hello, <b>Node.js</b>!
```

What is wrong?!



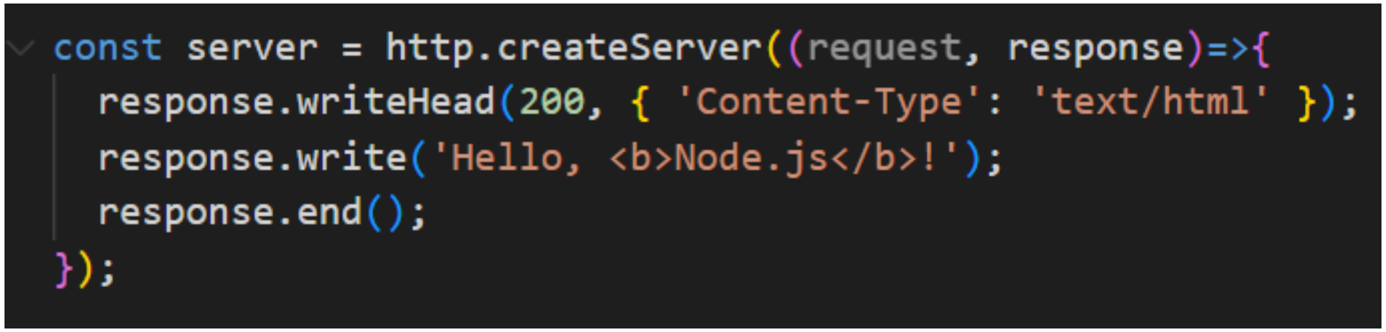
```
const server = http.createServer((request, response)=>{
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  response.write('Hello, <b>Node.js</b>!');
  response.end();
});
```



```
Hello, <b>Node.js</b>!
```

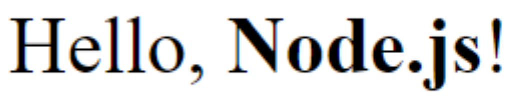
Response in HTML

Send HTML in response.write() with correct header

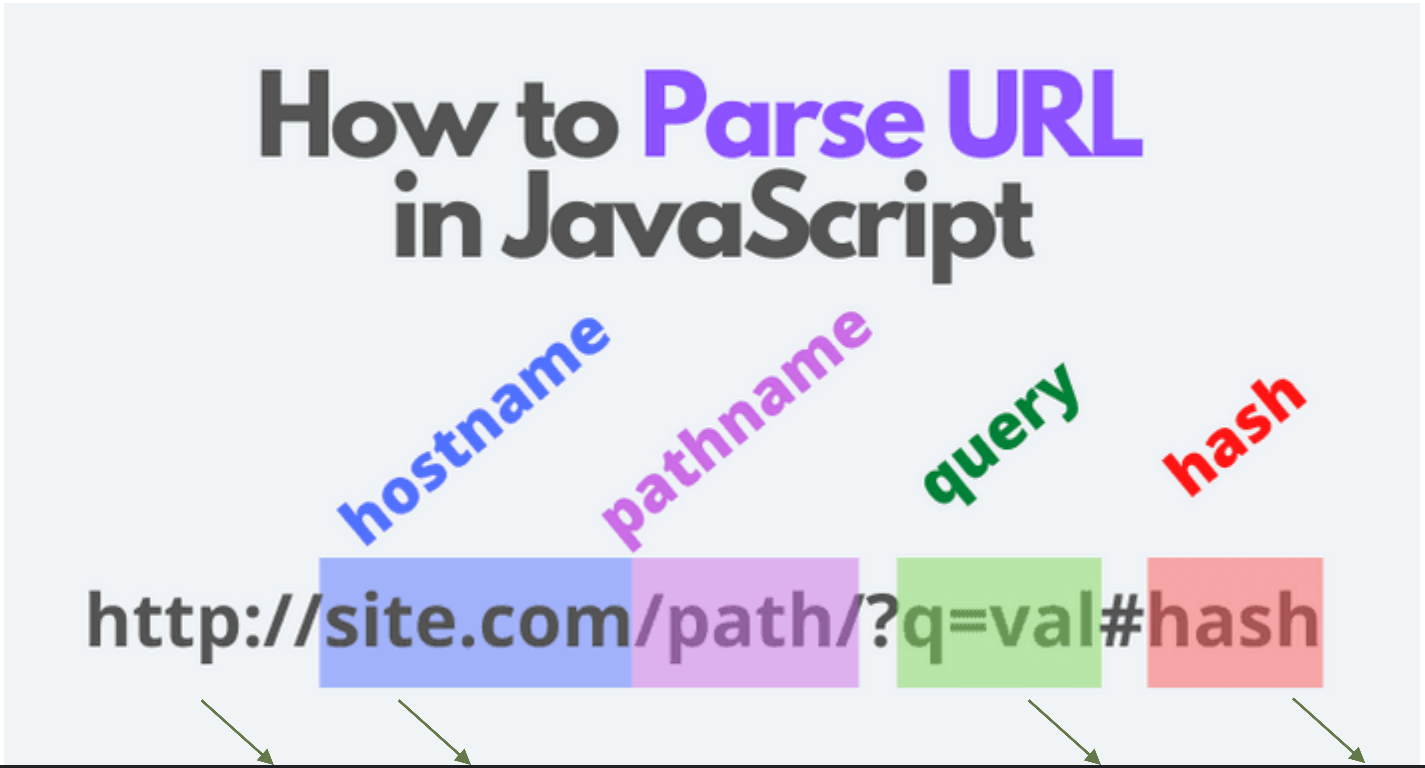


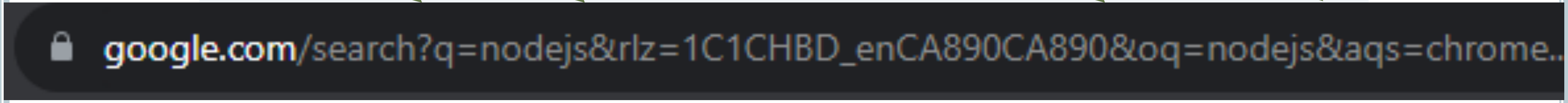
```
const server = http.createServer((request, response)=>{
    response.writeHead(200, { 'Content-Type': 'text/html' });
    response.write('Hello, <b>Node.js</b>!');
    response.end();
});
```

Result:



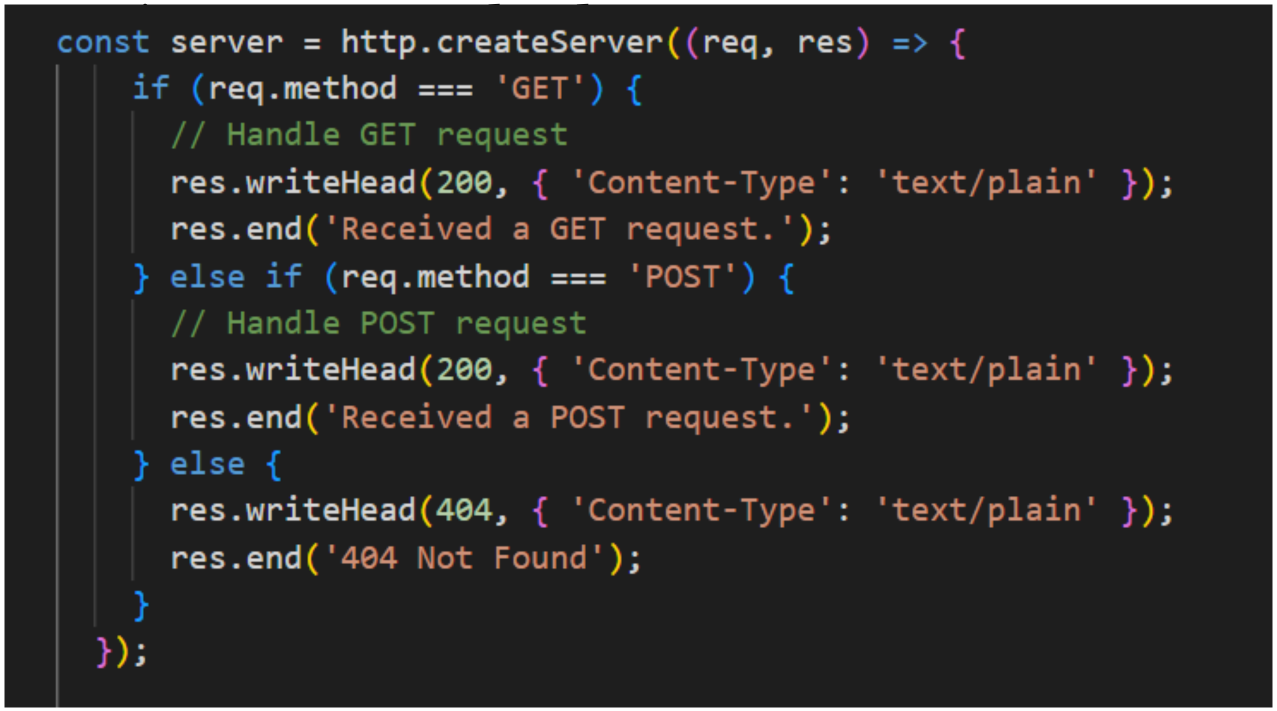
Hello, **Node.js**!




google.com/search?q=nodejs&rlz=1C1CHBD_enCA890CA890&oq=nodejs&aqs=chrome..

Request.method()

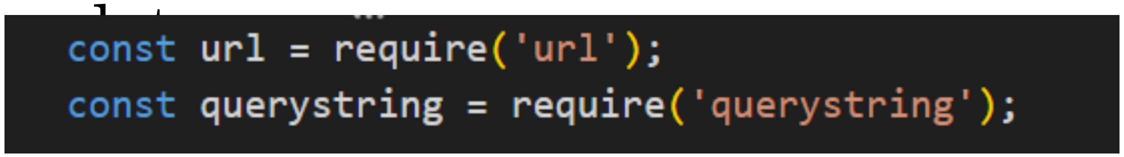
- Implementing routes for different HTTP methods



```
const server = http.createServer((req, res) => {
    if (req.method === 'GET') {
      // Handle GET request
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      res.end('Received a GET request.');
    } else if (req.method === 'POST') {
      // Handle POST request
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      res.end('Received a POST request.');
    } else {
      res.writeHead(404, { 'Content-Type': 'text/plain' });
      res.end('404 Not Found');
    }
  });
```

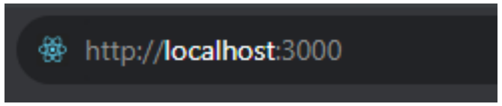
Parsing Request Data

- Utilizing 'url' and 'querystring' modules to extract

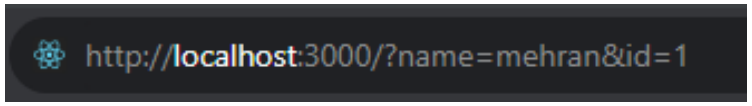


```javascript
const url = require('url');
const querystring = require('querystring');
```

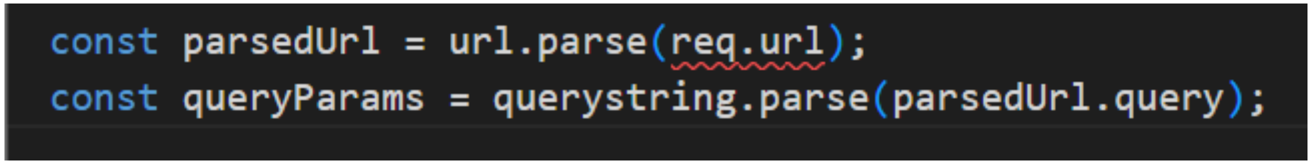
- Parsing url



```
http://localhost:3000
```



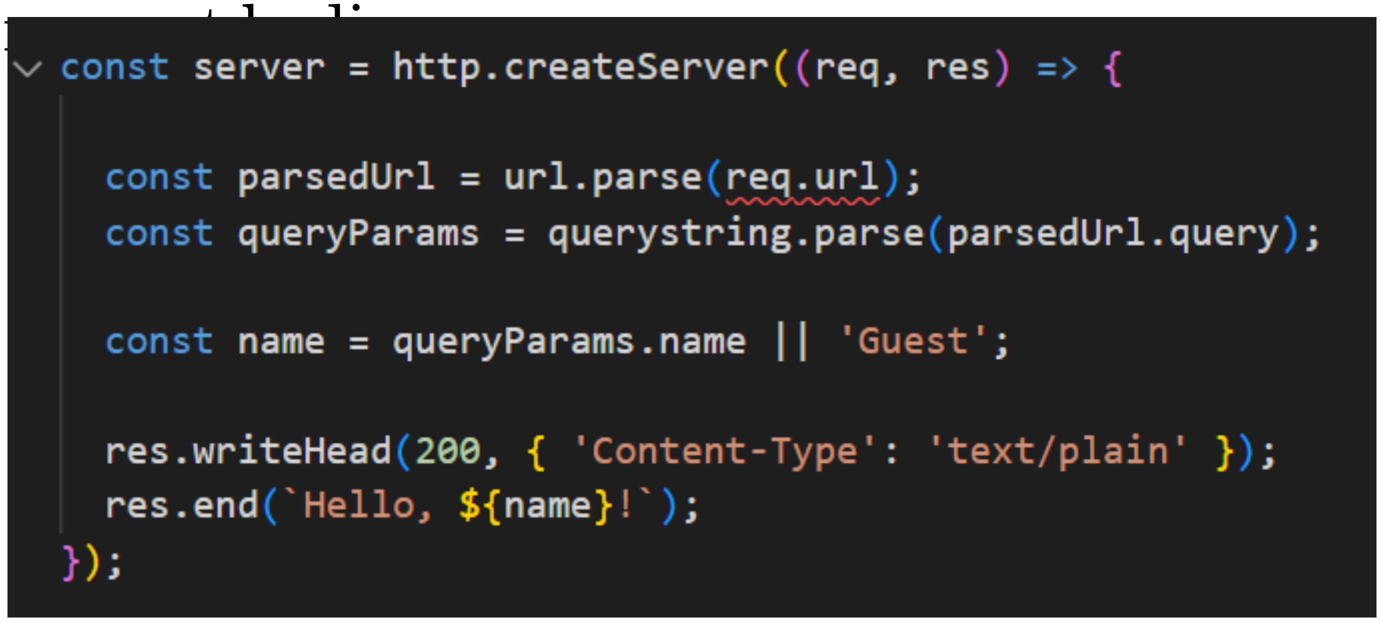
```
http://localhost:3000/?name=mehran&id=1
```



```javascript
const parsedUrl = url.parse(req.url);
const queryParams = querystring.parse(parsedUrl.query);
```

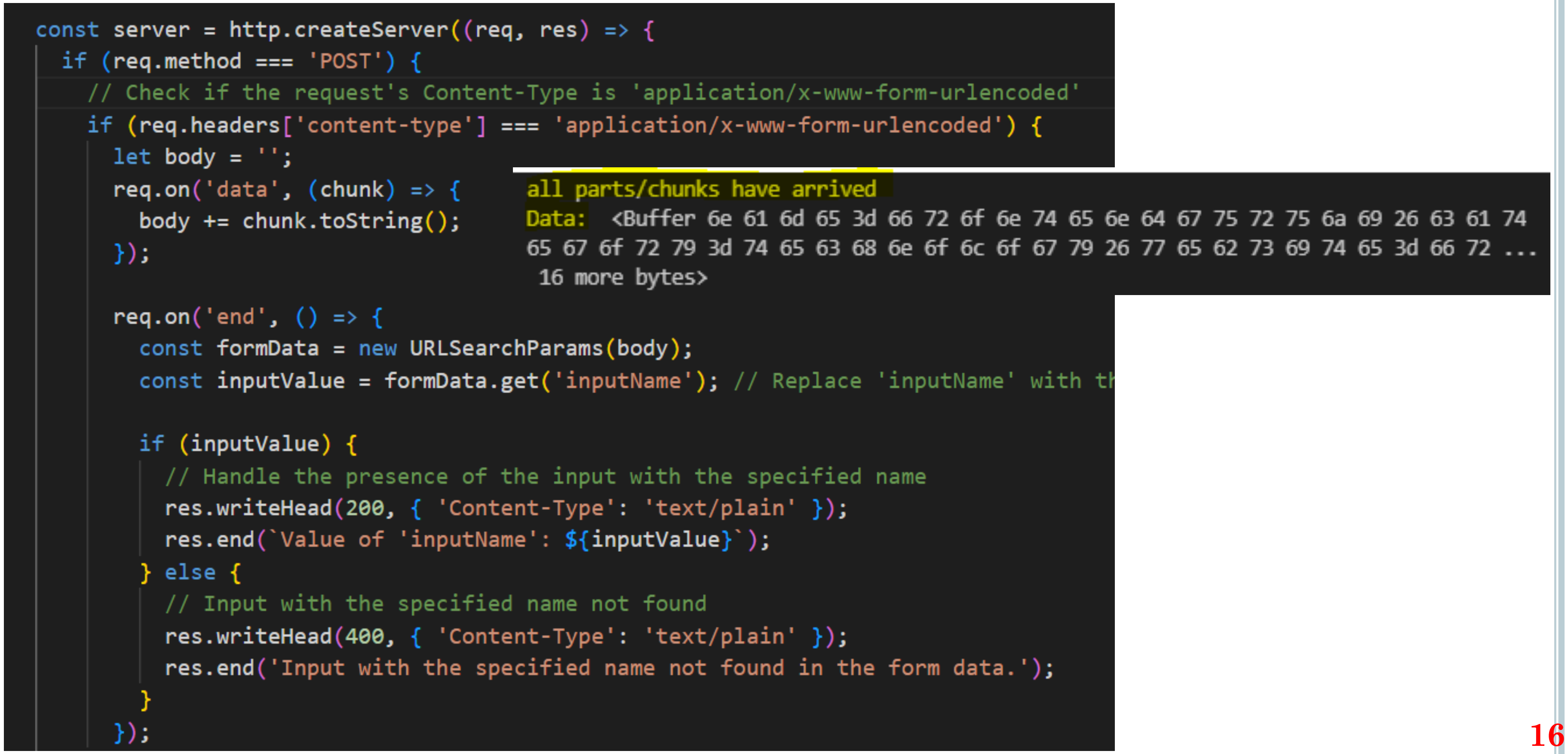
Parsing Request Data

- how to parse request data like URL parameters and



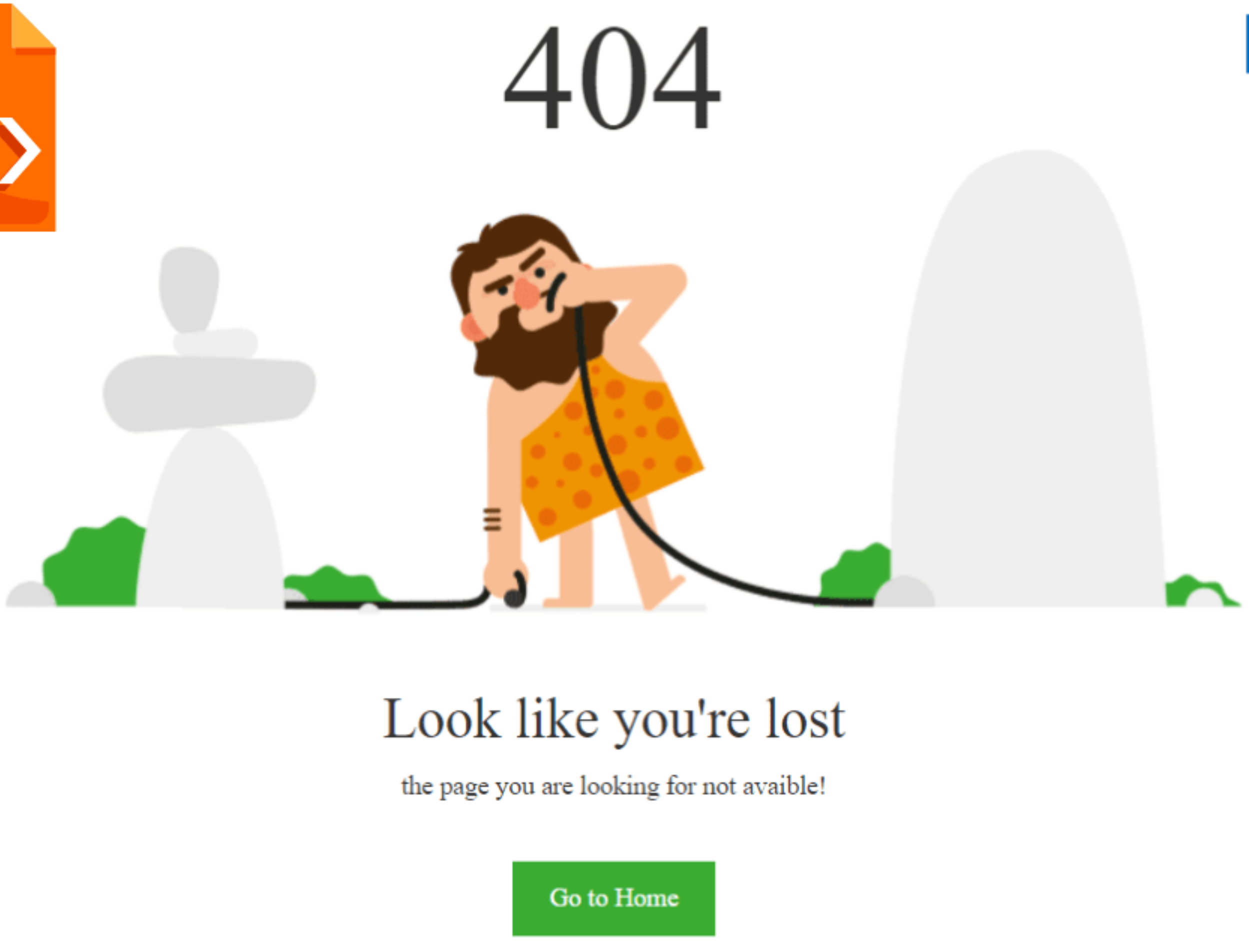
```
const server = http.createServer((req, res) => {

  const parsedUrl = url.parse(req.url);
  const queryParams = querystring.parse(parsedUrl.query);

  const name = queryParams.name || 'Guest';

  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end(`Hello, ${name}!`);
});
```

Parsing Request Data - POST



```
const server = http.createServer((req, res) => {
  if (req.method === 'POST') {
    // Check if the request's Content-Type is 'application/x-www-form-urlencoded'
    if (req.headers['content-type'] === 'application/x-www-form-urlencoded') {
      let body = '';
      req.on('data', (chunk) => {
        body += chunk.toString();
      });

      req.on('end', () => {
        const formData = new URLSearchParams(body);
        const inputValue = formData.get('inputName'); // Replace 'inputName' with th

        if (inputValue) {
          // Handle the presence of the input with the specified name
          res.writeHead(200, { 'Content-Type': 'text/plain' });
          res.end(`Value of 'inputName': ${inputValue}`);
        } else {
          // Input with the specified name not found
          res.writeHead(400, { 'Content-Type': 'text/plain' });
          res.end('Input with the specified name not found in the form data.');
        }
      });
```

```
all parts/chunks have arrived
Data:  <Buffer 6e 61 6d 65 3d 66 72 6f 6e 74 65 6e 64 67 75 72 75 6a 69 26 63 61 74
65 67 6f 72 79 3d 74 65 63 68 6e 6f 6c 6f 67 79 26 77 65 62 73 69 74 65 3d 66 72 ...
 16 more bytes>
```



404

Look like you're lost

the page you are looking for not avaible!

HTML Codes

200 Series

200 OK: The request was successful, and the server has returned the requested data.

201 Created: The request was successful, and a new resource has been created as a result.

400 Series

400 Bad Request: The server cannot process the request due to a client error

401 Unauthorized: The client must authenticate itself to get the requested response

403 Forbidden: The client does not have permission to access the requested resource.

404 Not Found: The requested resource could not be found on the server.

500 Series

500 Internal Server Error: The server encountered a situation it doesn't know how to handle (generic error).

502 Bad Gateway: received an invalid response

503 Service Unavailable: The server is currently unable to handle the request due to maintenance or overload.

504 Gateway Timeout

Express.js: A Powerful Node.js Framework

- **Express.js** is a popular and robust web framework for Node.js.
- It simplifies the process of building web applications and APIs by providing a feature-rich set of tools and utilities.

What makes Express.js valuable?

Minimalistic and Flexible:

- Express.js follows a minimalist approach, allowing developers to choose the components they need.
- It provides essential features while leaving the rest to optional middleware.

Easy Routing:

- Express.js offers a straightforward and intuitive way to define routes for handling various HTTP requests (GET, POST, etc.).
- Routing helps organize application logic and improves code readability.

What makes Express.js valuable?

Middleware Support:

- Middleware functions can be added to the request-response cycle to perform various tasks like authentication, logging, and error handling.
- Middleware enhances modularity and reusability.

Template Engines:

- Express.js supports various template engines (e.g., EJS, Handlebars) for rendering dynamic HTML pages.
- This simplifies the process of generating HTML with data from the server.

Weekly Downloads

26,310,036

Whoa! That's a big number!

const http = require('http');

Step-by-Step Guide: Getting Started with Express.js

- Importing Express:
  - To use Express.js, first, we need to install it and then import it into our project.
  - Install Express.js using npm:
  -

```
npm install express
```

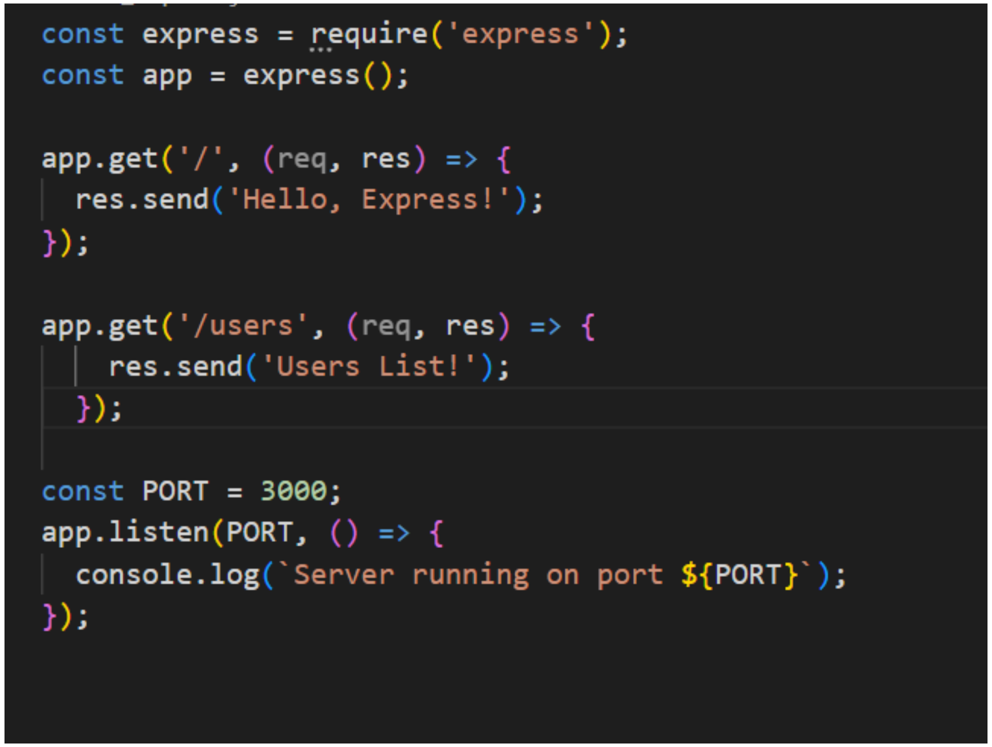
- Import Express into your Node.js file:

```
const express = require('express');
const app = express();
```

const http = require('http');

Setting Up Routes

- Define routes to handle GET HTTP methods and URLs.

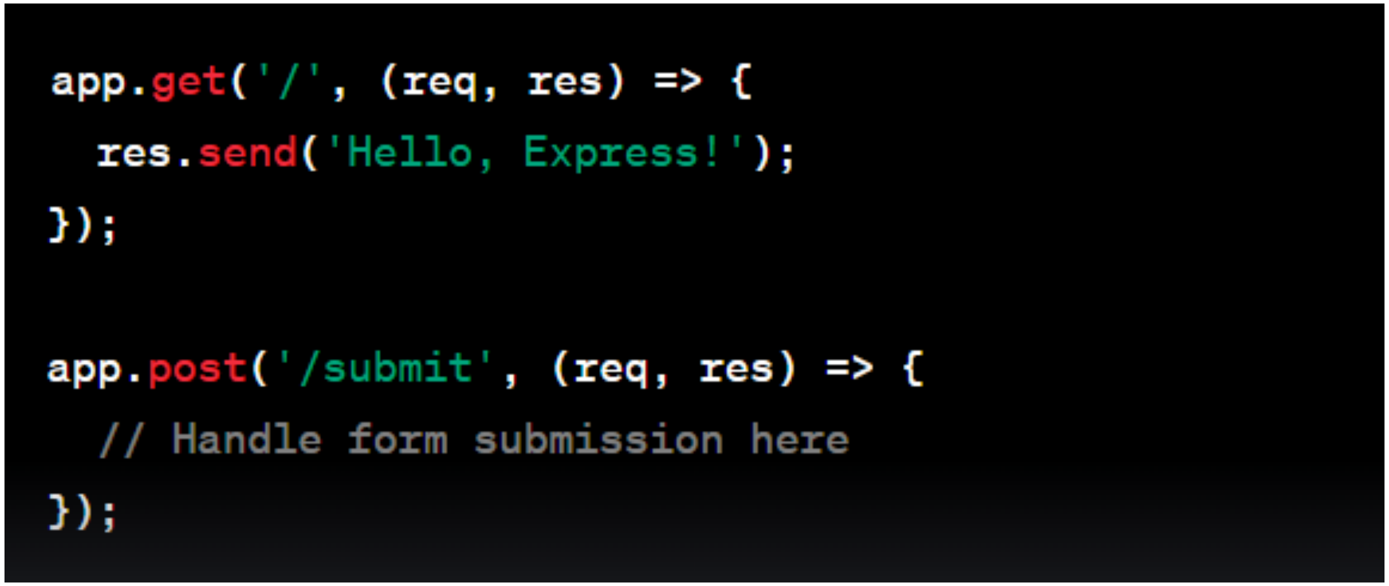


```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello, Express!');
});

app.get('/users', (req, res) => {
    res.send('Users List!');
  });

const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

const http = require('http');

Setting Up Routes

- Define routes to handle different HTTP methods and URLs.
  - Use app.get(), app.post(), etc., to specify how the server should respond to each request.



```
app.get('/', (req, res) => {
  res.send('Hello, Express!');
});

app.post('/submit', (req, res) => {
  // Handle form submission here
});
```

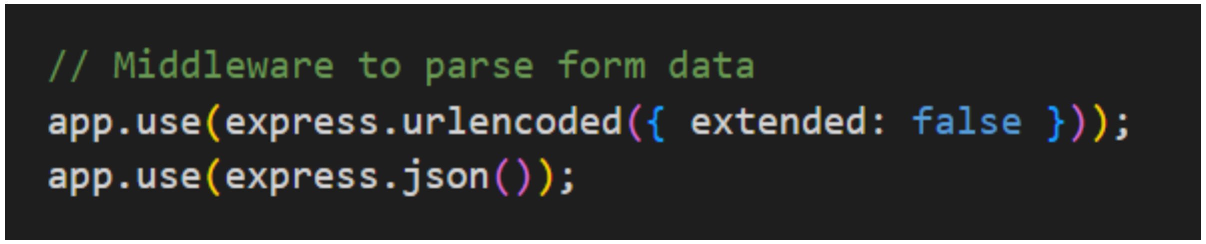
Parsing Form Data with Express Framework

https://jaydevs.com/top-companies-using-react-js/

const http = require('http');

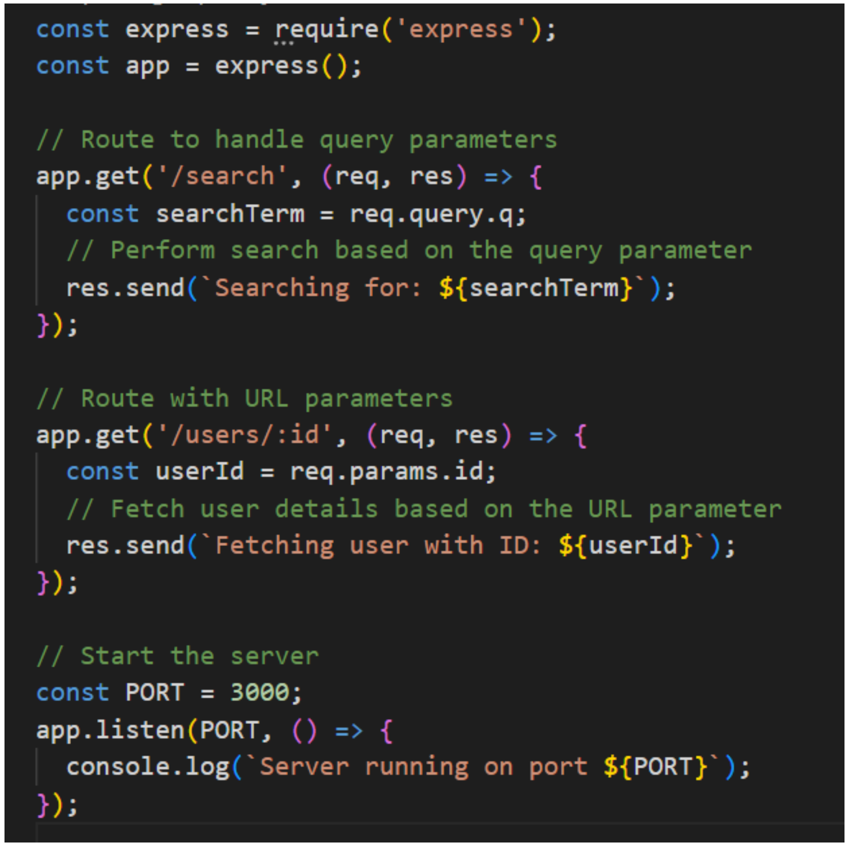
Step-by-Step Guide: Getting Started with Express.js

- Express Middleware for Form Data:
  - express.**urlencoded**(): Middleware for parsing form data with 'Content-Type: application/x-www-form-urlencoded'.
  - express.**json**(): Middleware for parsing JSON data with 'Content-Type: application/json'.

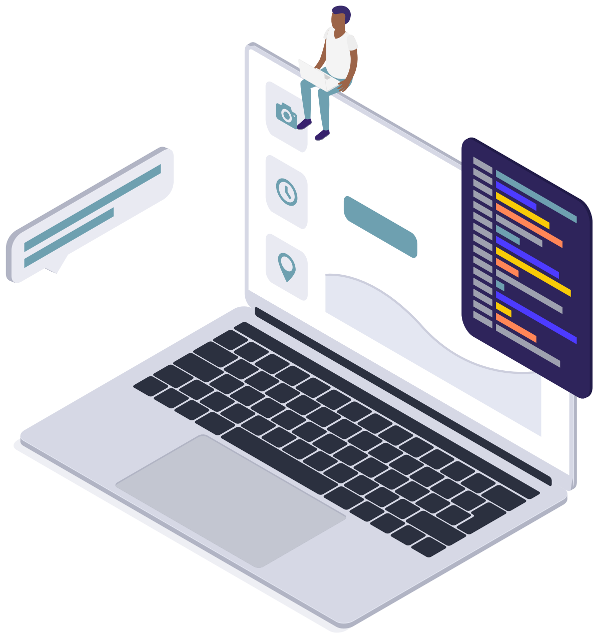


```
// Middleware to parse form data
app.use(express.urlencoded({ extended: false }));
app.use(express.json());
```

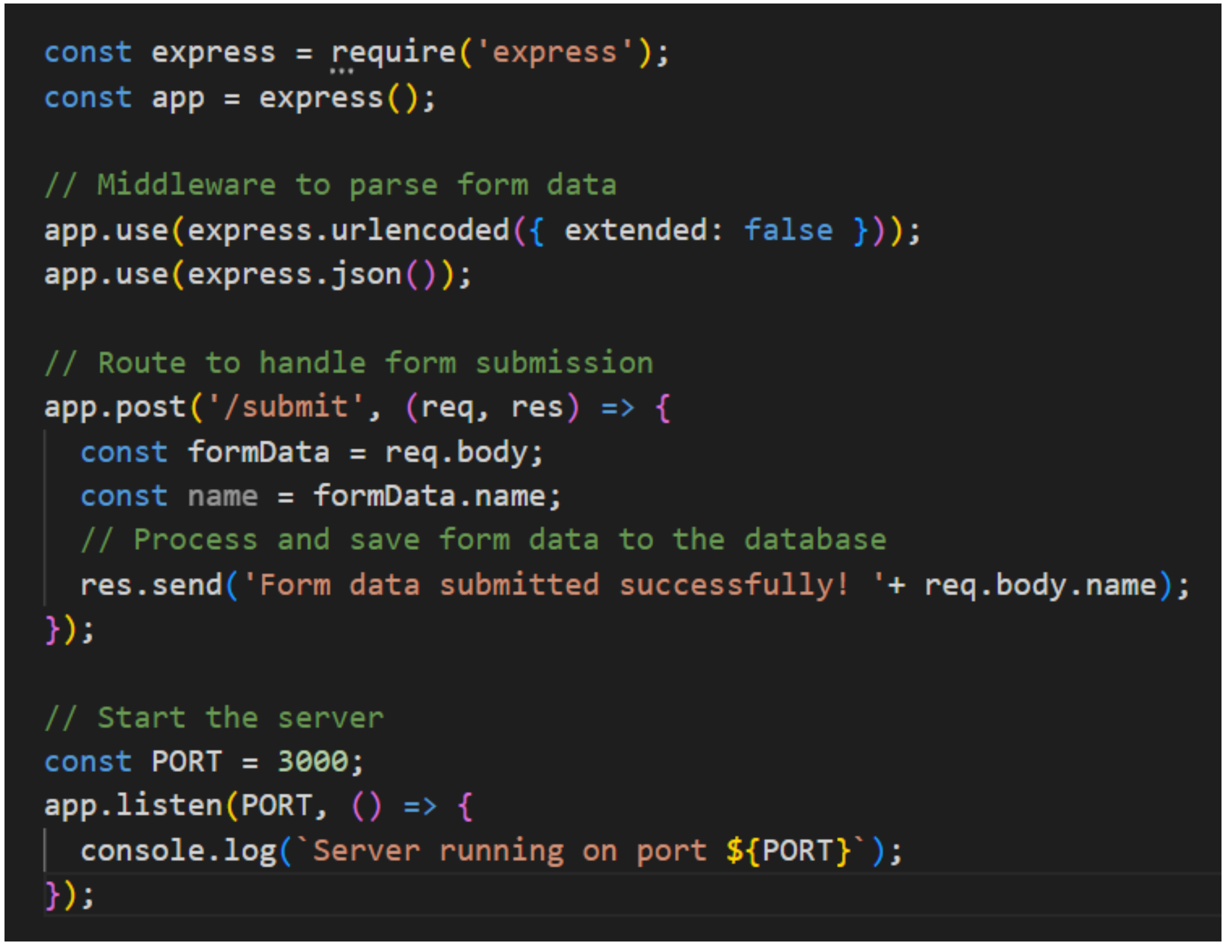
Step-by-Step Guide: : Parse GET with Express.js



```
const express = require('express');
const app = express();

// Route to handle query parameters
app.get('/search', (req, res) => {
  const searchTerm = req.query.q;
  // Perform search based on the query parameter
  res.send(`Searching for: ${searchTerm}`);
});

// Route with URL parameters
app.get('/users/:id', (req, res) => {
  const userId = req.params.id;
  // Fetch user details based on the URL parameter
  res.send(`Fetching user with ID: ${userId}`);
});

// Start the server
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```



Step-by-Step Guide: Submit form with Express.js



```
const express = require('express');
const app = express();

// Middleware to parse form data
app.use(express.urlencoded({ extended: false }));
app.use(express.json());

// Route to handle form submission
app.post('/submit', (req, res) => {
  const formData = req.body;
  const name = formData.name;
  // Process and save form data to the database
  res.send('Form data submitted successfully! '+ req.body.name);
});

// Start the server
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Want big impact?

Act boldly, leave a mark.

The End