

# Introduction to

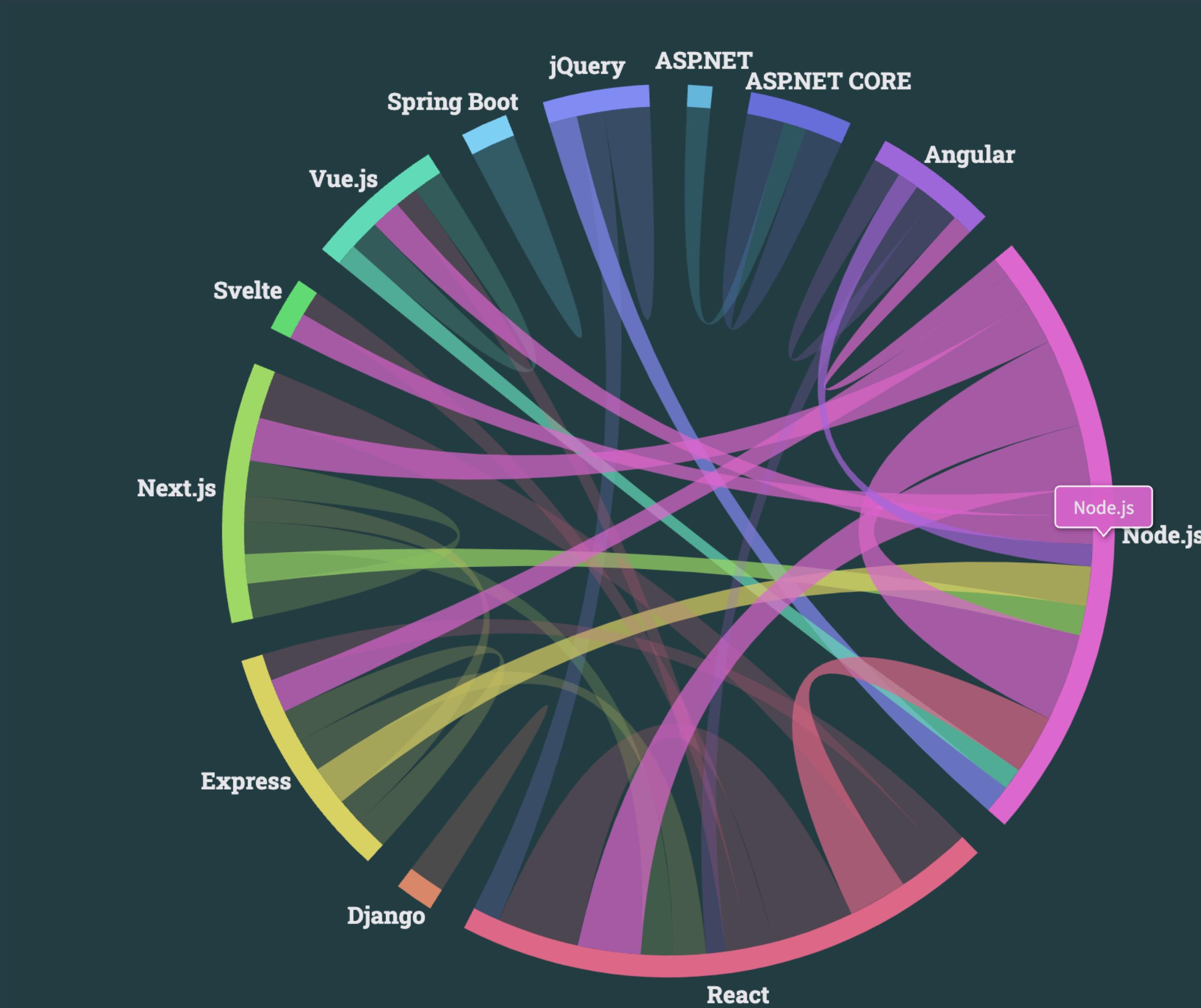


# Web frameworks and technologies



<https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe>

# Web frameworks and technologies

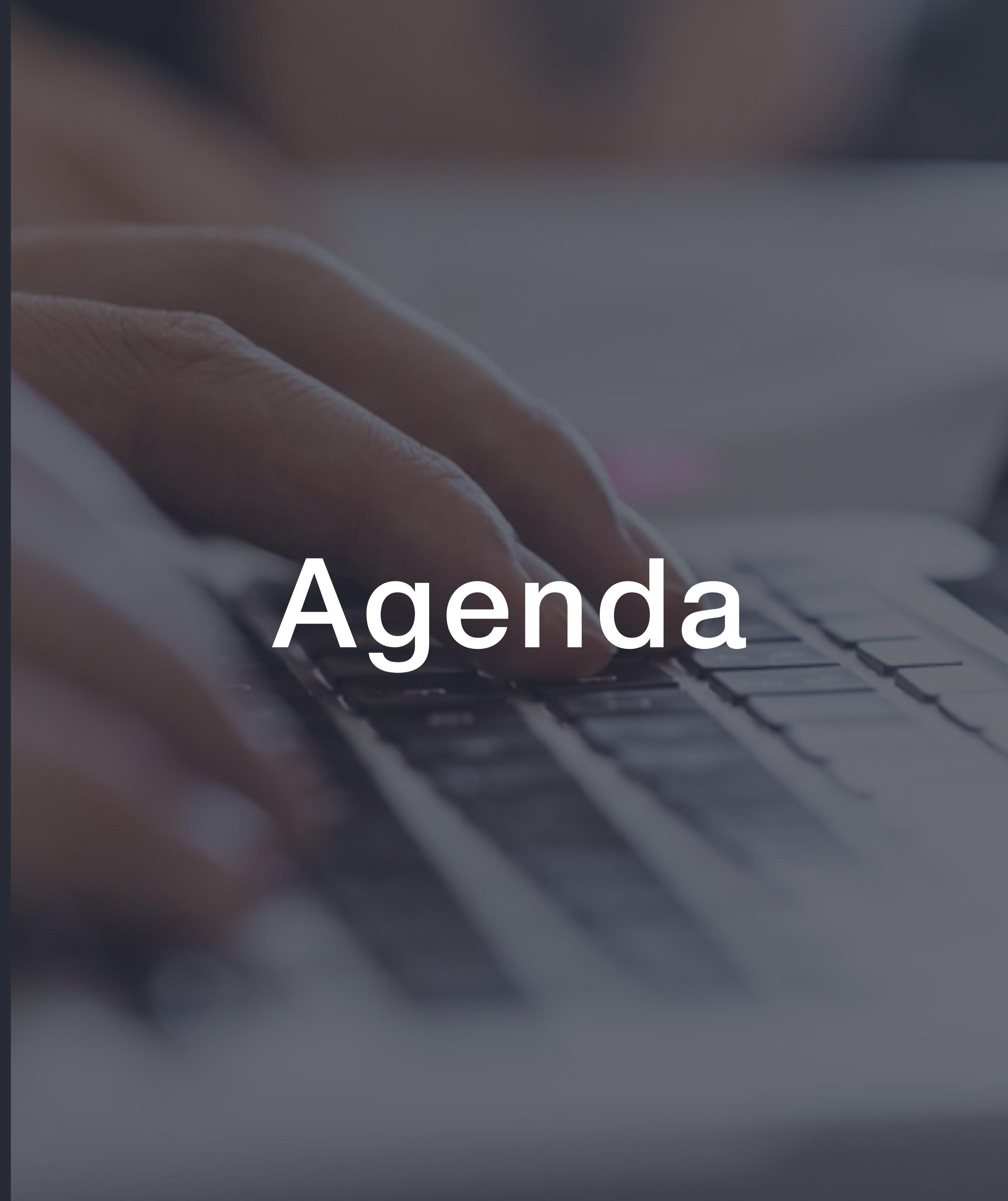


<https://survey.stackoverflow.co/2023/#section-worked-with-vs-want-to-work-with-web-frameworks-and-technologies>

# Formål

- At skabe en forståelse for hvad Node.js er og kan, samt introducere de koncepter og tools, der gør sig gældende for backendudvikling med JavaScript (Node.js).
- Praktisk tilgang: Vi skal arbejde med opgaver kombineret med mindre teorioplæg.

1. What's Node.js & Why Do We Need It?
2. Node.js Runtime Environment & Key Features
3. Node.js HTTP Module & Postman
4. Node.js File System
5. Wrap up (Libraries & Hand In)



# Agenda



- Server-side JavaScript runtime environment.
- Executes JavaScript outside of the browser.



- Imagine Node.js as a powerful engine that lets you run JavaScript code outside of a web browser.
- Instead of just using JavaScript for web pages, Node.js lets you use it on your computer or on a server.
- It's like having a super-fast computer that can understand and do all the smart things you do in a web browser with JavaScript, but now it can do it on your own computer or a server.

# Run JavaScript, not just in browsers



A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows a project folder named "NODE-JS-FUN" containing an "app.js" file. The main editor area displays the following code:

```
const yourFavoriteLecturer = {  
  name: "Peter Lind",  
  mail: "petl@kea.dk"  
};  
console.log(yourFavoriteLecturer);
```

The bottom right terminal window shows the output of running the script:

```
race@RACEs-MacBook-Pro node-js-fun % node app.js  
{ name: 'Peter Lind', mail: 'petl@kea.dk' }  
race@RACEs-MacBook-Pro node-js-fun %
```

The status bar at the bottom indicates the code is in "JavaScript" mode, has 0 errors and 0 warnings, and shows the current position as "Ln 7, Col 1".

- **JavaScript Beyond Browser**

Instead of being limited to doing things only on the internet, Node.js lets you use JavaScript to build programs and applications on your computer or server. You get to use the same language you know from web pages.

- **Fast and Efficient**

Node.js is built to be really fast and efficient. It can handle many things at once without getting slow. This is really useful when you're working with apps that need to do lots of things at the same time, like chat apps or real-time updates.

- **Don't Wait for Things**

Usually, when a computer does something, it has to wait until that task is finished before moving on to the next one. Node.js does things differently. It can start a task and then keep working on other things while it waits for the first task to finish.

- **Customize with Modules**

Think of modules as little building blocks of code. Node.js gives you lots of ready-made code that you can use in your programs. You don't have to invent everything from scratch. It makes it easy to add new features to your projects.

- **Used for Many Things**

People use Node.js to build all sorts of things, from simple web servers to complex applications. It's popular in modern web development because it allows fast and efficient communication between clients and servers.



# Key Points

# Øvelse

## Hello Node.js

The screenshot shows a dark-themed code editor interface. In the center, there's a code editor window titled "app.js — hello-node". The file contains the following code:

```
JS app.js
1 console.log("Hello, Node.js 🎉");
2 
```

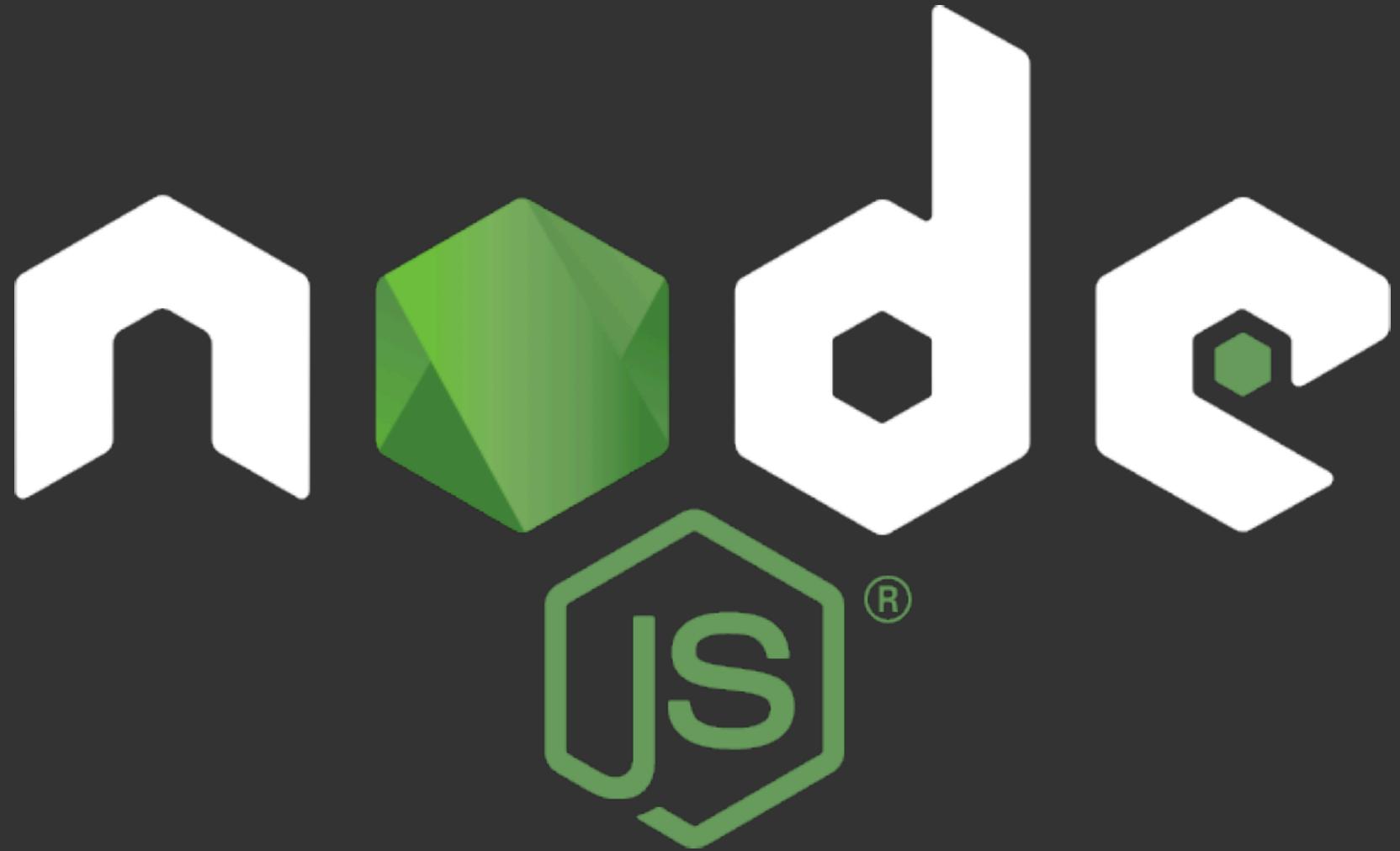
Below the code editor is a terminal window showing the output of running the script:

```
TERMINAL ... zsh + ⌂ ...
race@RACEs-MacBook-Pro hello-node % node app.js
Hello, Node.js 🎉
race@RACEs-MacBook-Pro hello-node % 
```

On the left side of the interface is an Explorer sidebar showing the project structure:

- HELLO-N... (expanded)
- .gitattributes
- .gitignore
- JS app.js** (selected)
- package.json

At the bottom of the interface, there are several status indicators and icons.



Why do we need it?



# Why do we need it?

- In pairs, discuss and investigate
- Look at code and precious Node projects
- Ask Google and ChatGPT



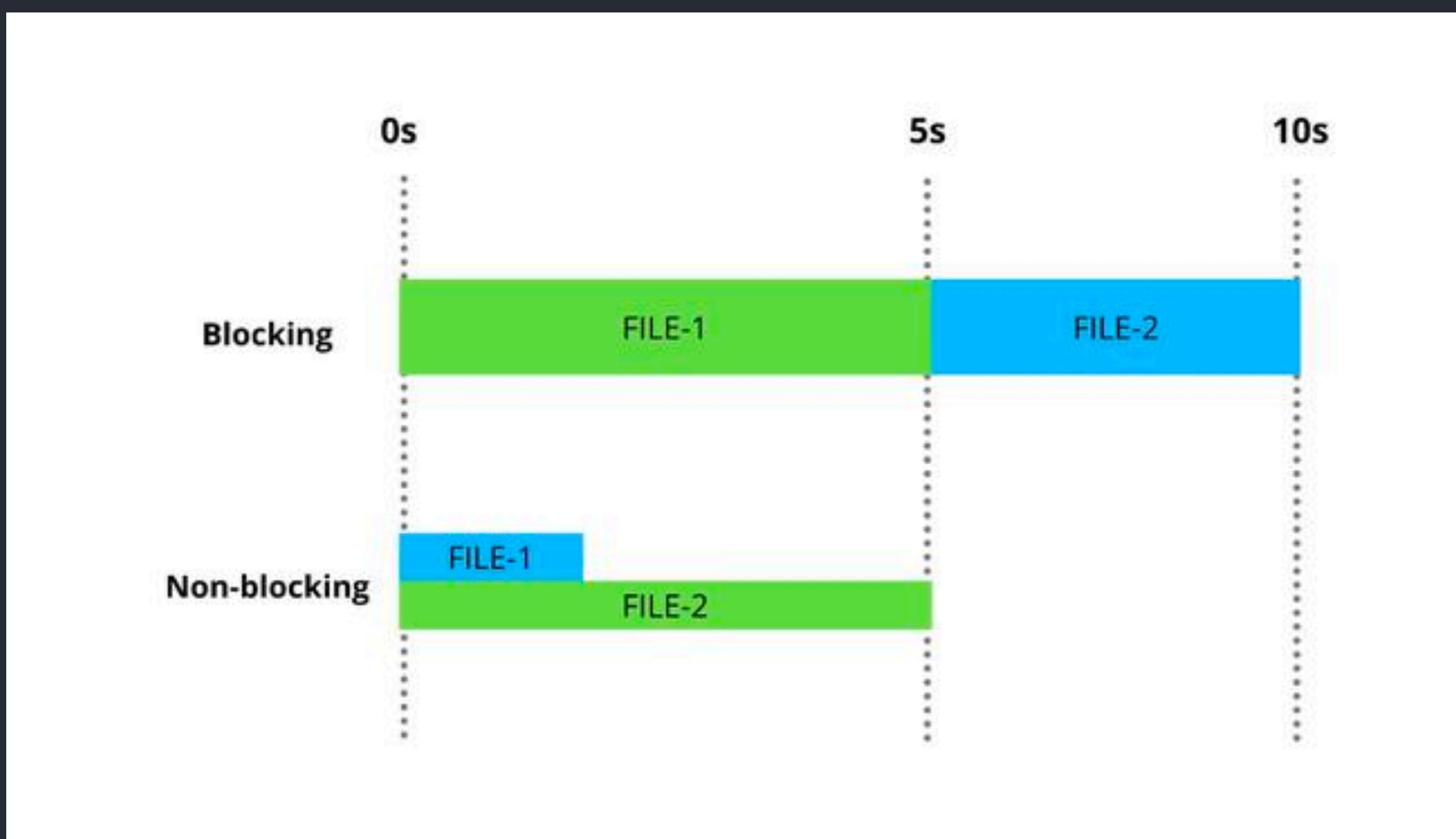
# Runtime Environment

- The Node.js runtime environment is an open-source, cross-platform JavaScript runtime built on Google Chrome's V8 JavaScript engine.
- It allows developers to execute JavaScript code on the server-side, outside of the browser, enabling them to build scalable and high-performance applications.
- Node.js provides a range of features and capabilities that are particularly well-suited for building real-time, networked, and event-driven applications.



## Runtime Environment

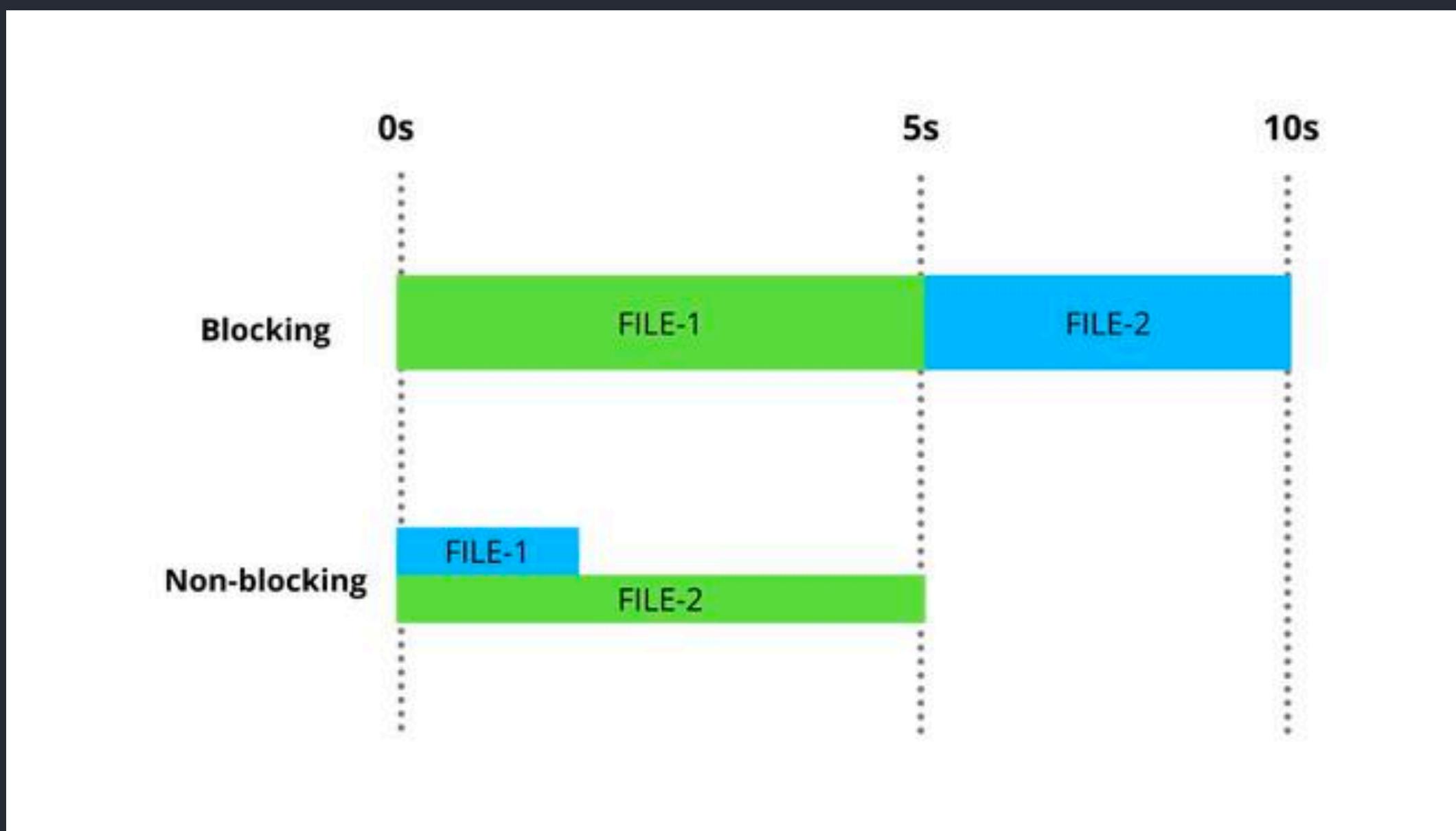
- **Non-blocking I/O**  
Node.js operates using an event-driven, non-blocking I/O model, which means that it can handle multiple concurrent connections without requiring a separate thread for each connection. This design choice enables efficient handling of I/O-bound tasks, making it suitable for applications that involve a lot of network communication.

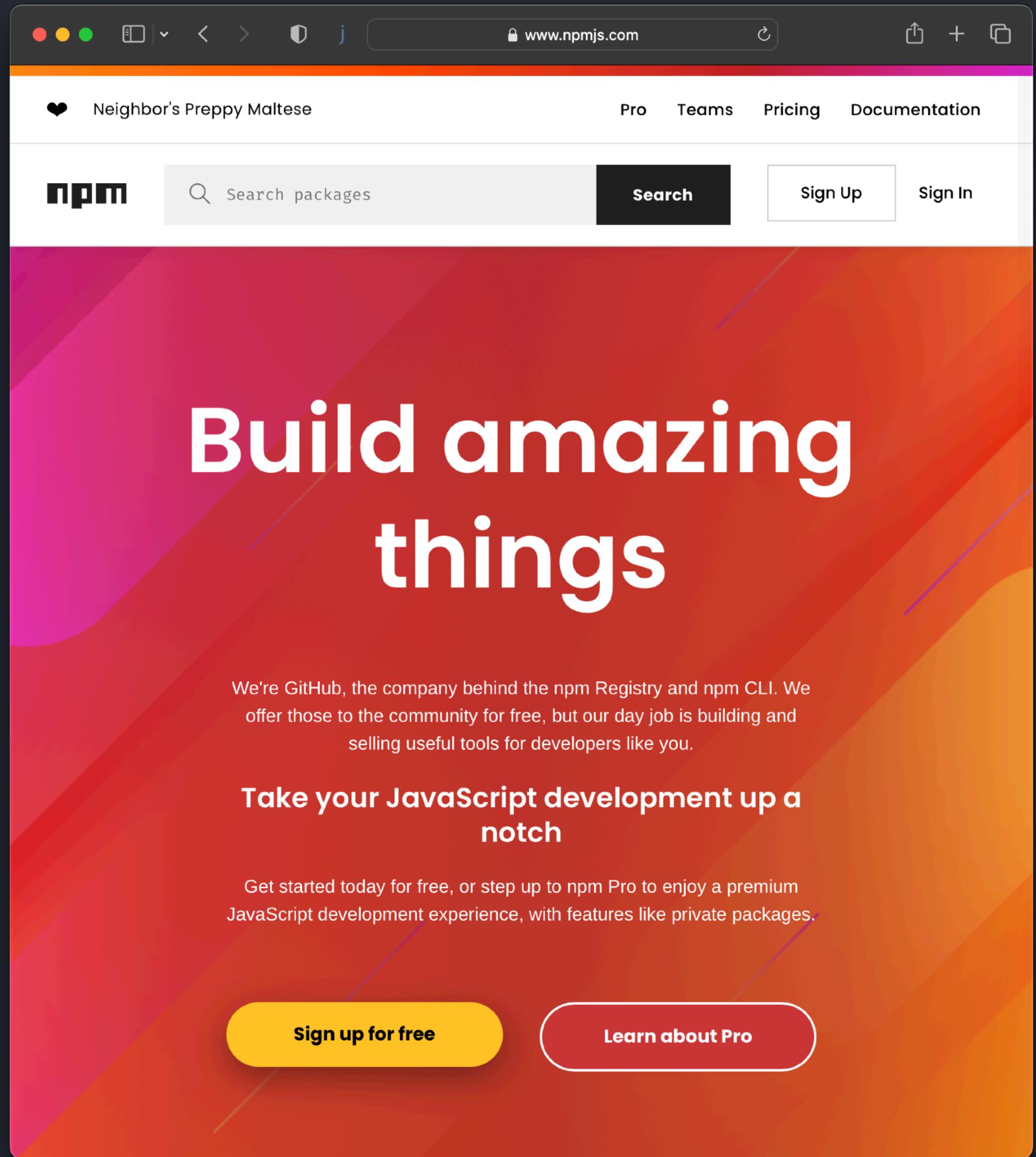




## Runtime Environment

- **Asynchronous Programming**  
Node.js heavily emphasizes asynchronous programming patterns. This allows developers to write code that doesn't block the execution of other tasks while waiting for certain operations to complete. This approach enhances the application's responsiveness and throughput.





## Runtime Environment

- **NPM (Node Package Manager)**

NPM is a powerful package manager that comes bundled with Node.js. It provides access to a vast ecosystem of open-source libraries and modules that can be easily integrated into Node.js applications, saving developers time and effort by avoiding the need to reinvent the wheel.



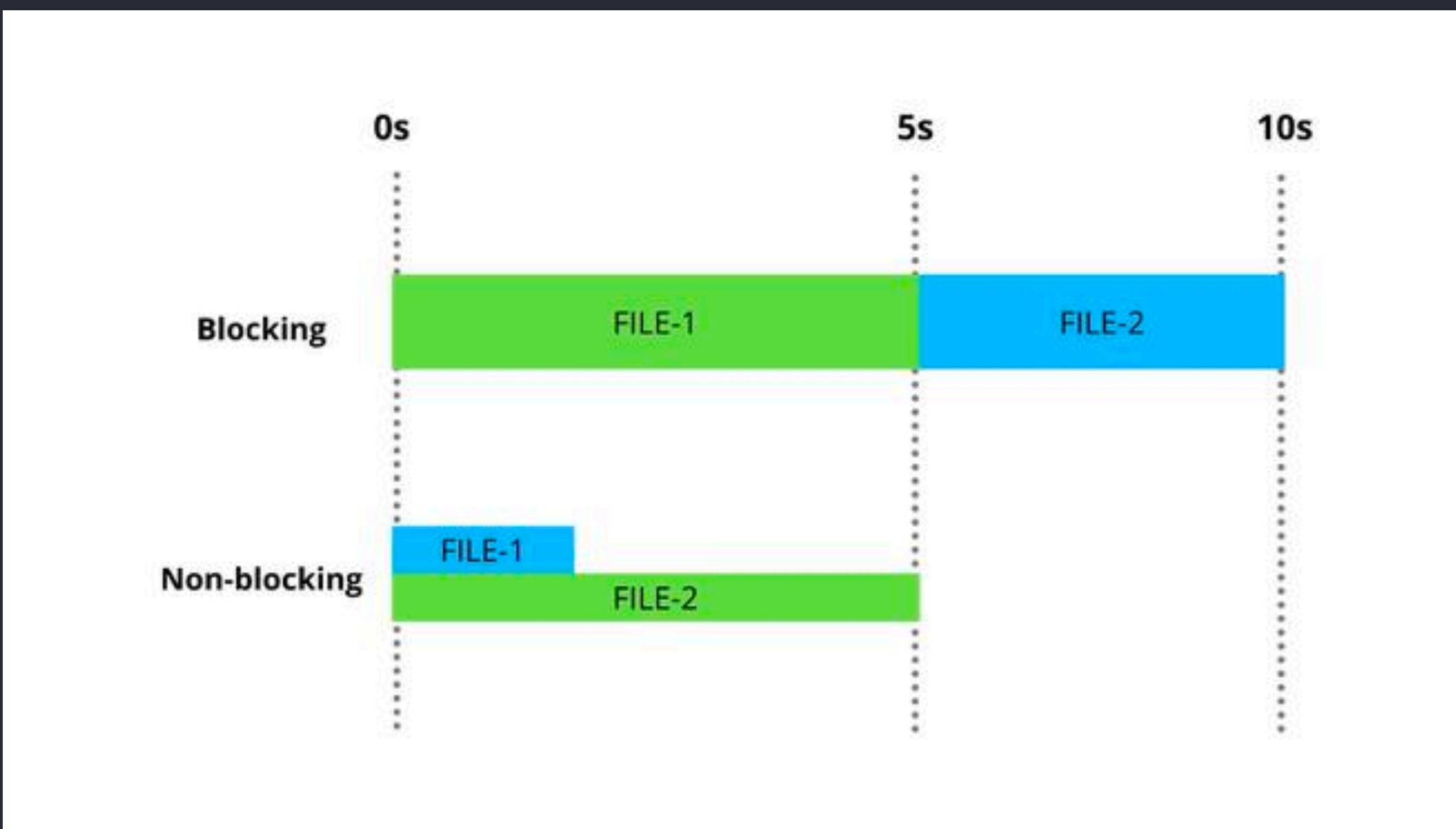
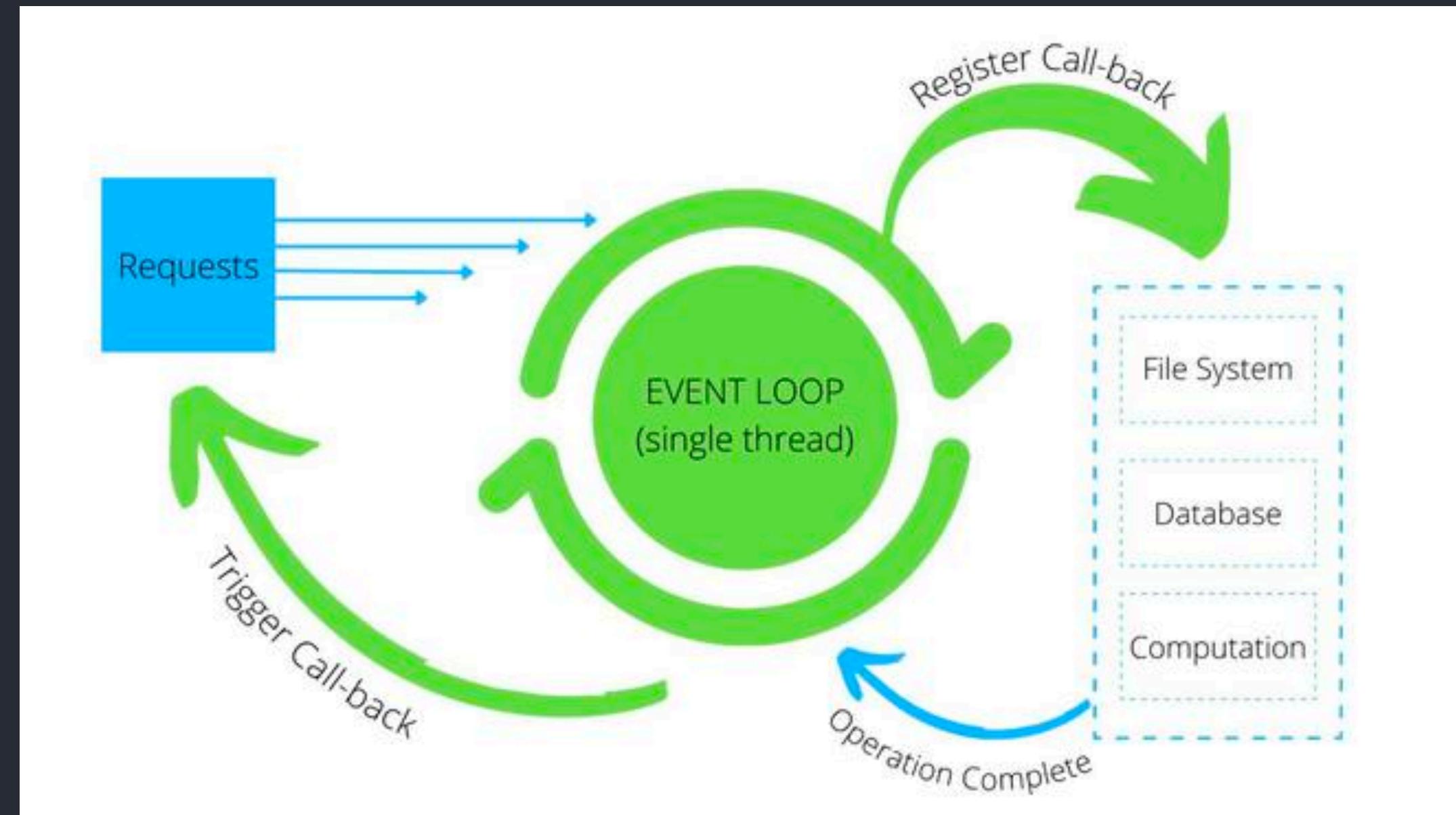
Runtime  
Environment



- **Server-side Capabilities**  
Node.js can function as a web server, handling incoming HTTP requests and serving responses. Developers can build web applications, APIs, and microservices using the same language for both client and server, which can streamline development and maintenance.



## Runtime Environment

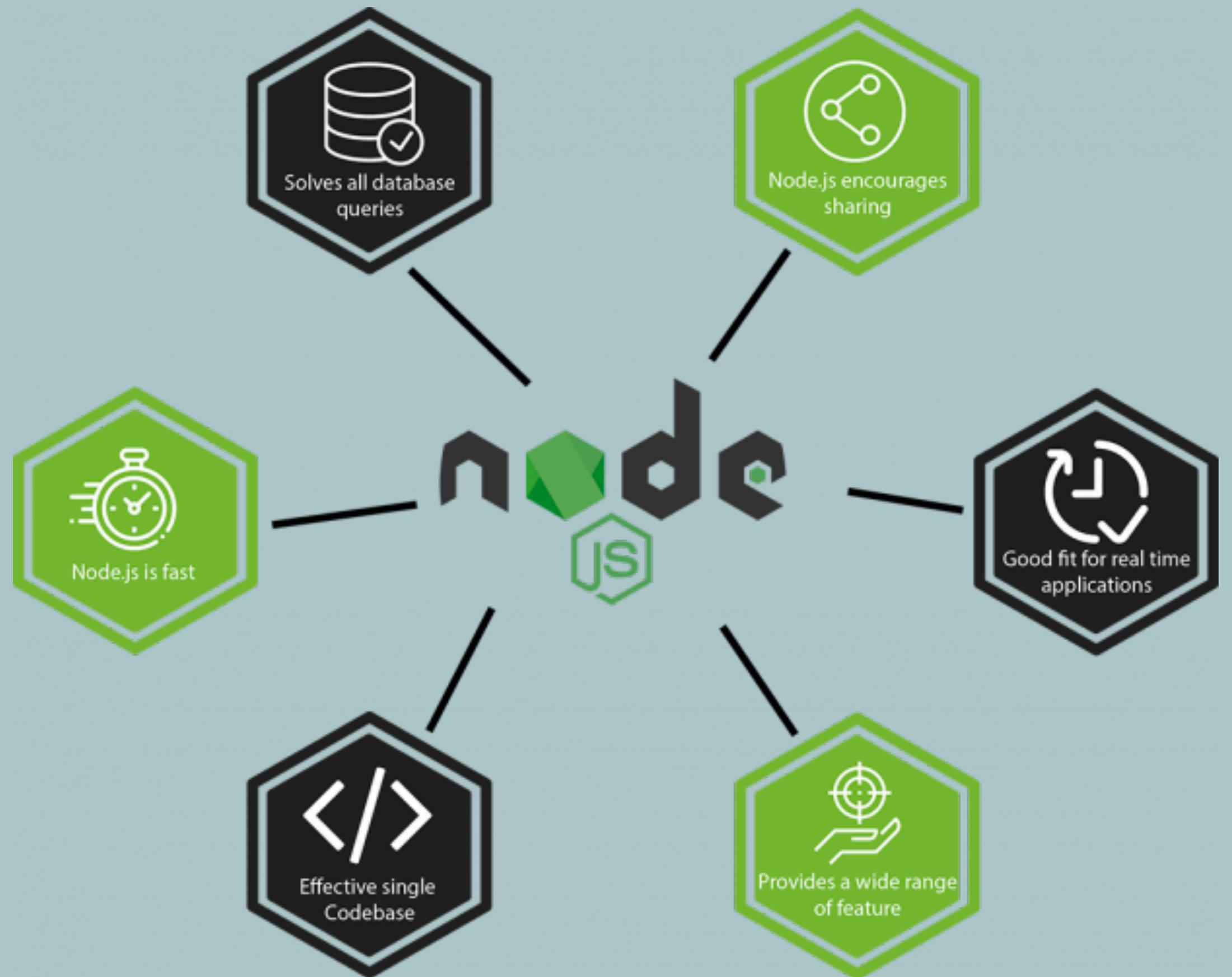


- **Scalability**

Due to its non-blocking architecture, Node.js is well-suited for building scalable applications that can handle a large number of concurrent connections. This makes it a popular choice for real-time applications like chat applications, online gaming, and streaming services.



## Runtime Environment



- **Community and Ecosystem**

Node.js has a large and active community that continuously contributes to its growth. The ecosystem includes various tools, frameworks, and libraries that extend its capabilities and make development more efficient.



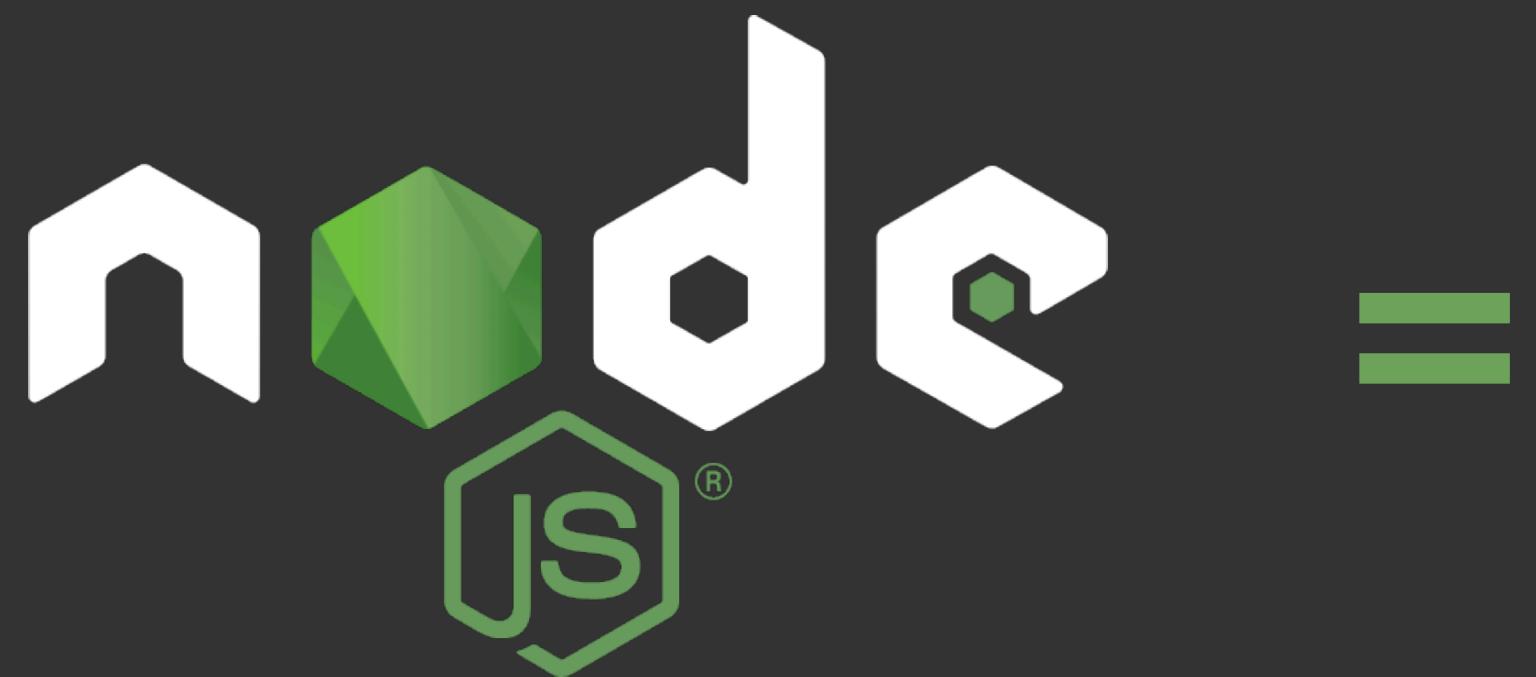
# Runtime Environment

- In summary, Node.js provides a runtime environment that allows developers to use JavaScript beyond the browser, enabling them to create high-performance, scalable, and real-time applications for various use cases.

... we can use our  
existing **JavaScript**  
**expertise** to develop  
backend applications,  
servers, REST APIs, etc.



**Runtime  
Environment**



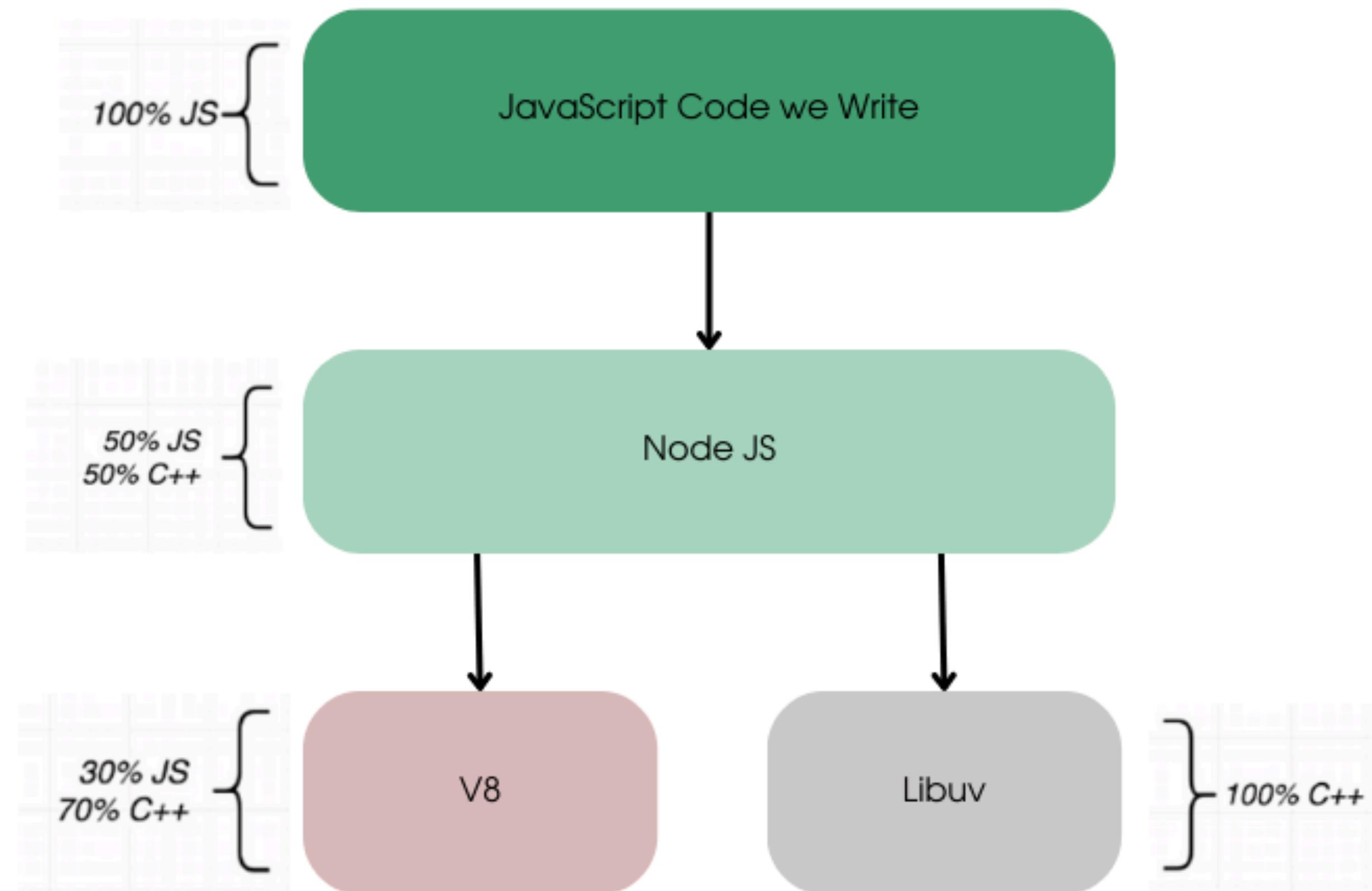
=

Runtime  
Environment

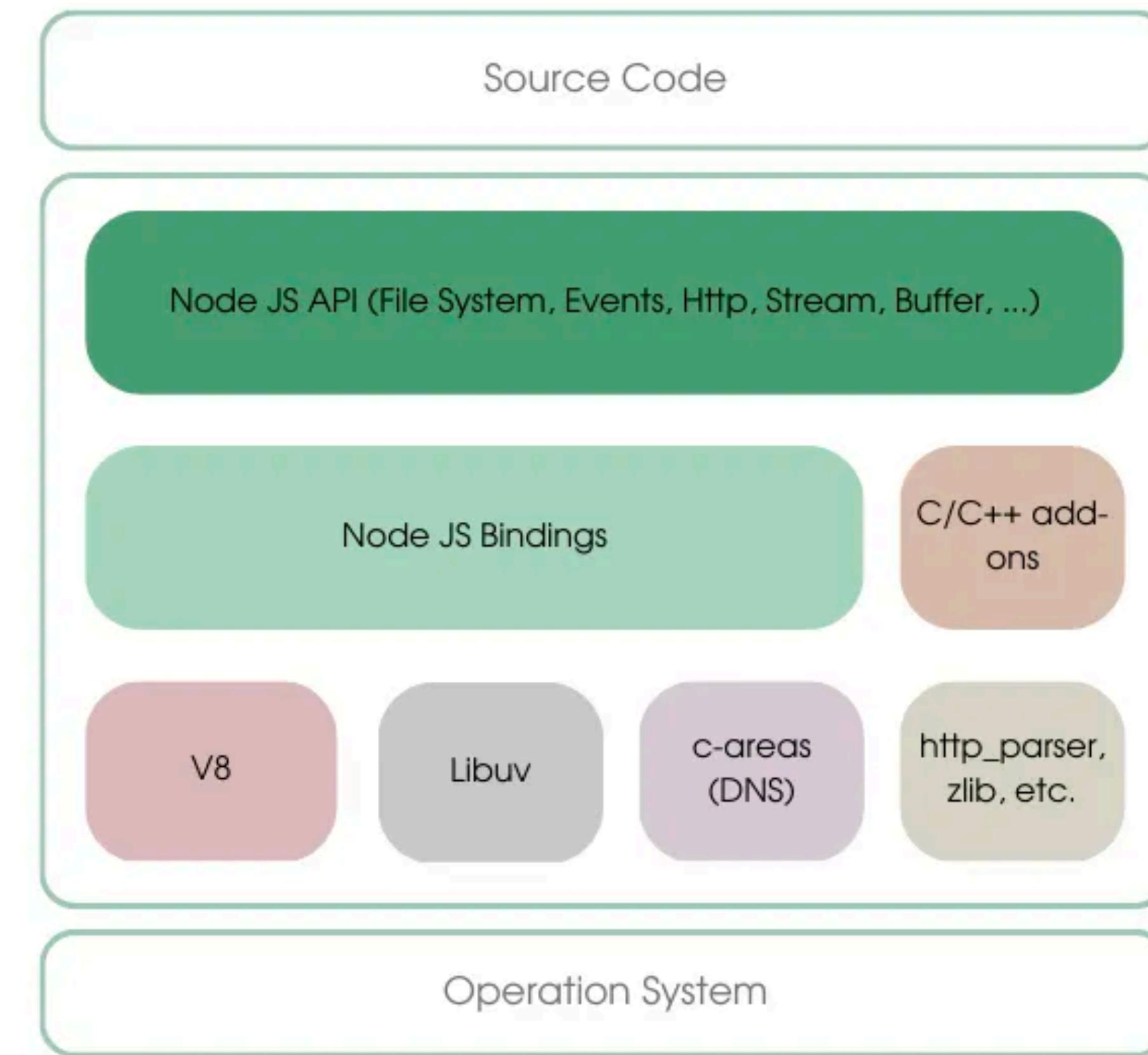
+

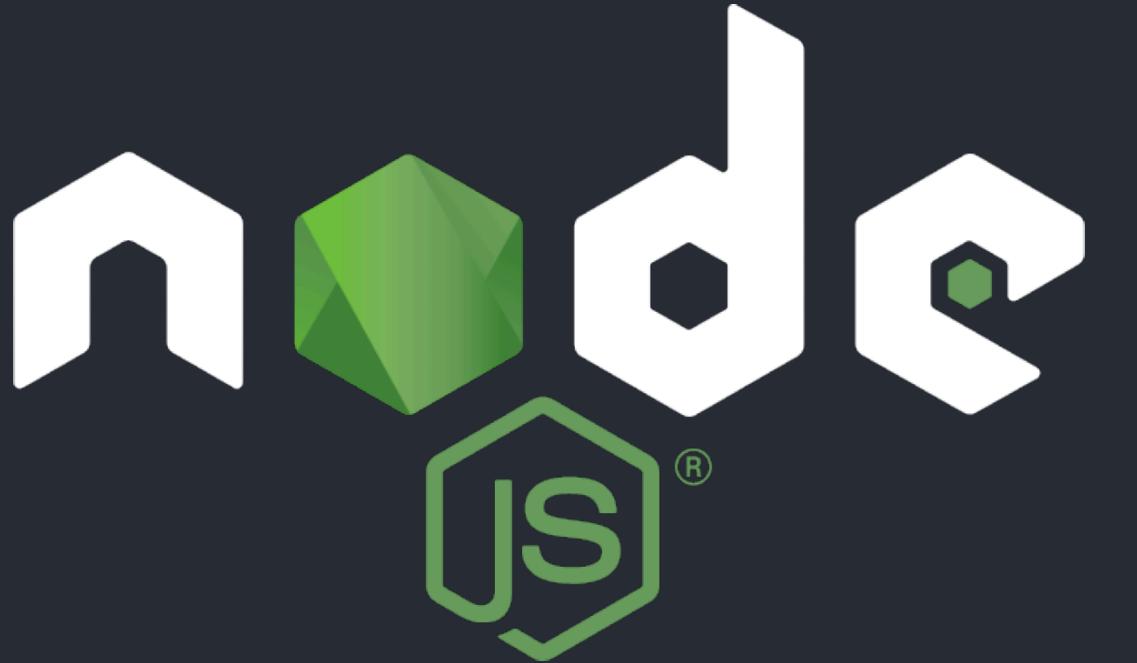
JavaScript  
Library

# Node.js Architecture



# Node.js Architecture





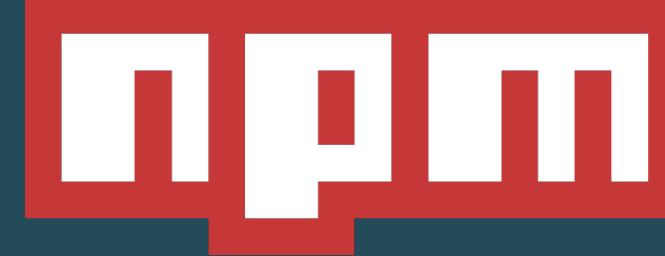
Node.js also comes with a toolbox called NPM, which has all sorts of ready-made tools (like Lego blocks) that developers can use to build their programs. Instead of building everything from scratch, you can use these tools to save time and make your app even better.

# npm is a package manager for Node.js packages (JS modules)

- Node.js packages contains all the files you need for a module.
- Modules are JavaScript libraries you can include in your project.

# My Project

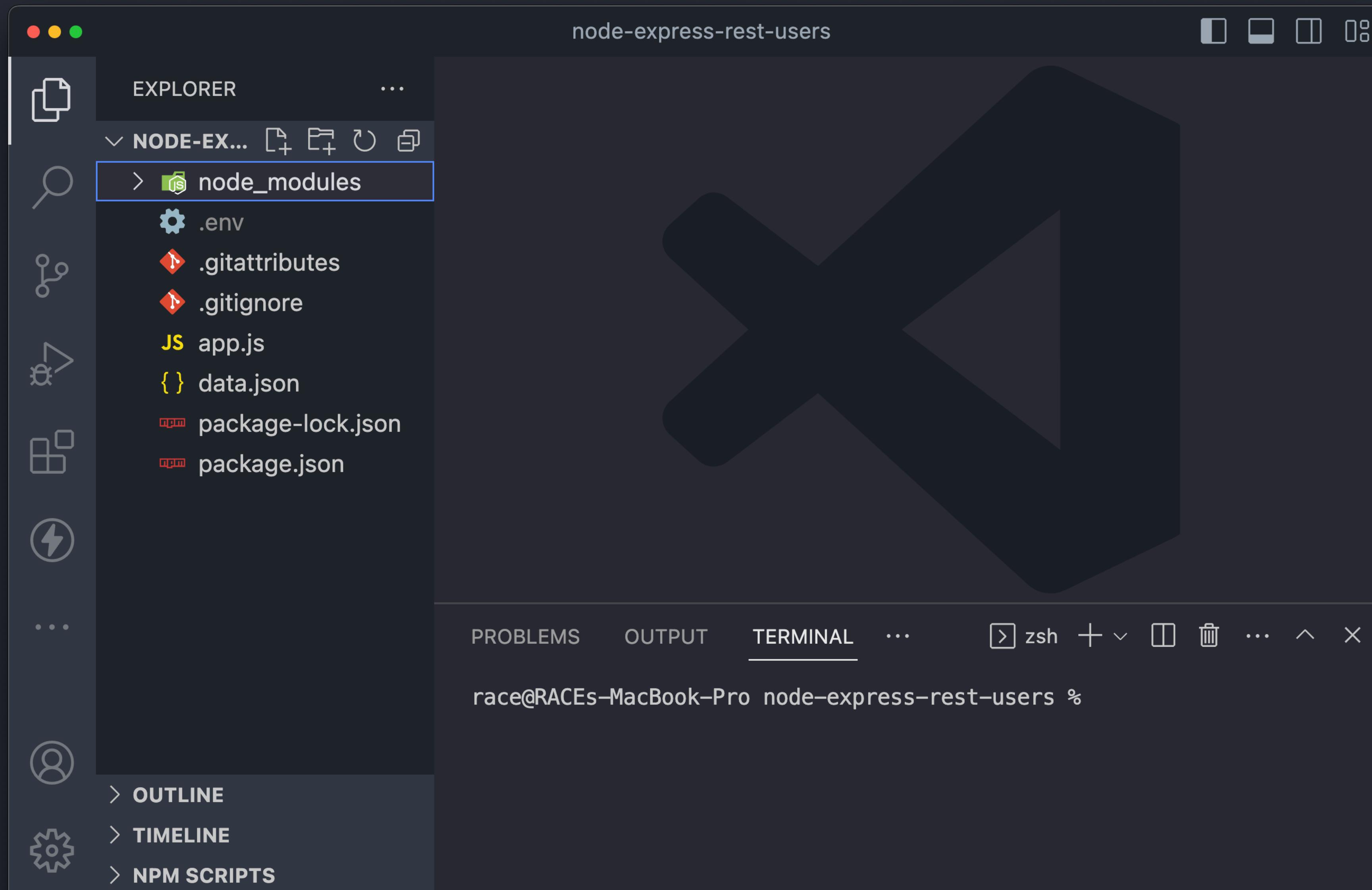
```
✓ NODE-USERS-REST-API
  > node_modules
  .gitignore
  app.js
  data.json
  package-lock.json
  package.json
```



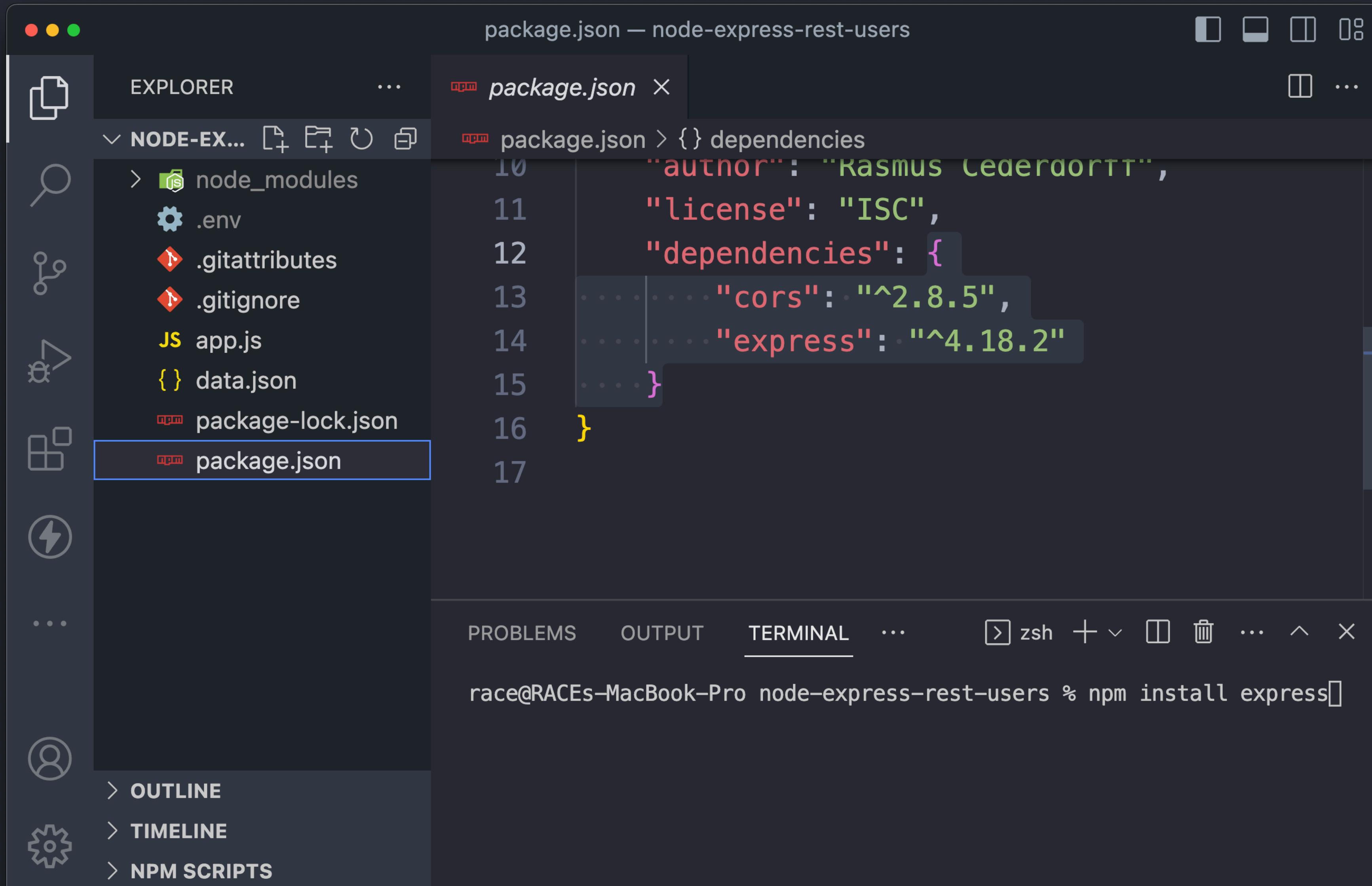
CORS

And many other  
JavaScript libraries

# node\_modules



# npm install { package-name }



A screenshot of the Visual Studio Code (VS Code) interface. The title bar shows "package.json — node-express-rest-users". The Explorer sidebar on the left shows a project structure with files like node\_modules, .env, .gitattributes, .gitignore, app.js, data.json, package-lock.json, and package.json. The package.json file is selected in the Explorer. The main editor area displays the following JSON code:

```
package.json — node-express-rest-users
package.json X
{
  "name": "node-express-rest-users",
  "version": "1.0.0",
  "author": "Rasmus Læderortz",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.18.2"
  }
}
```

The "dependencies" section is highlighted with a pink background. Below the editor, the status bar shows the terminal command: "race@RACEs-MacBook-Pro node-express-rest-users % npm install express".

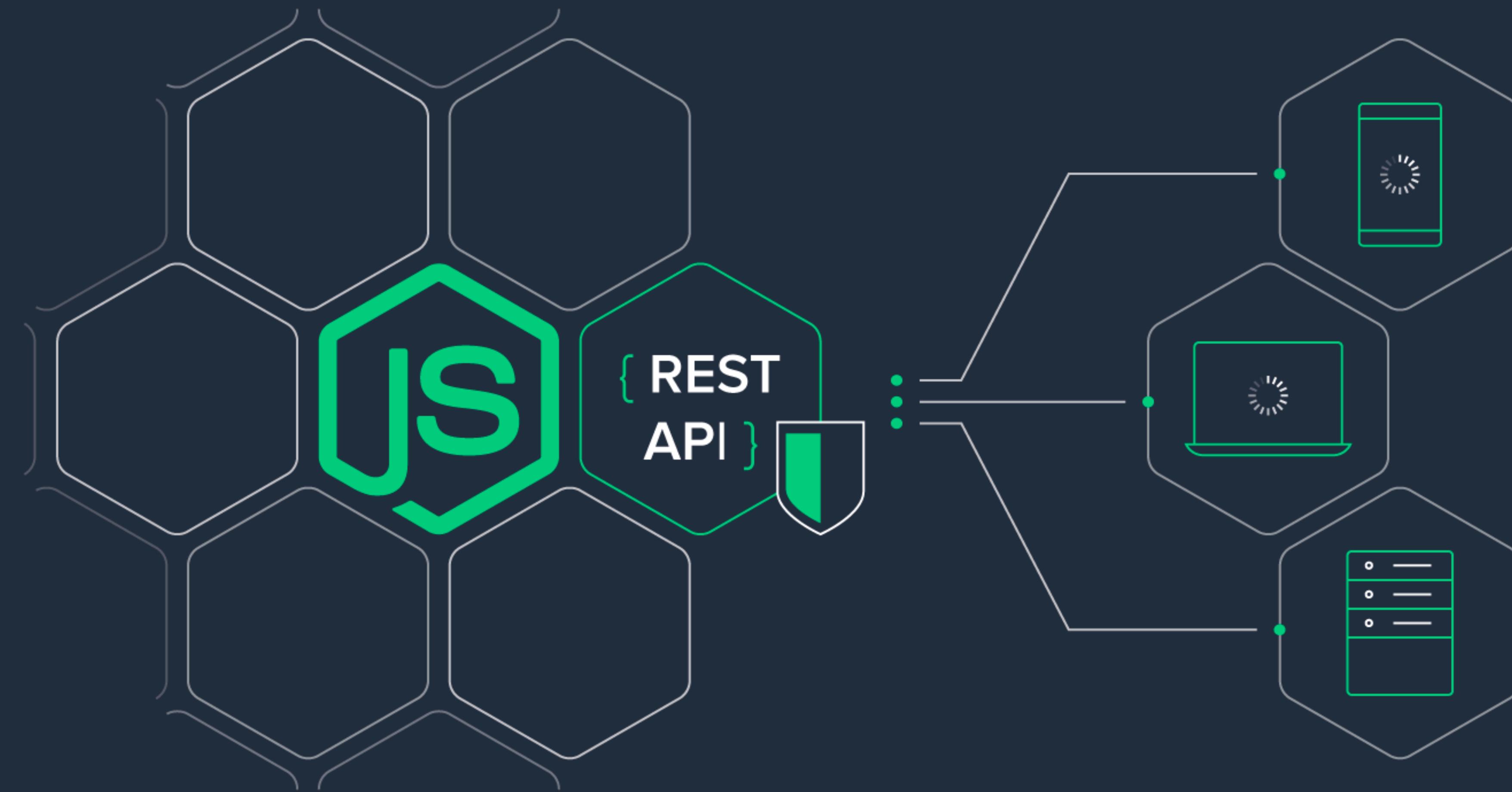
# npm consists of

1. Website to discover packages
2. Command line interface (cli)
3. A registry - database with JS software/  
libraries/ node modules

<https://docs.npmjs.com/about-npm/>



# Node.js HTTP Module



The screenshot shows a web browser window with the title "Node.js HTTP Module". The URL in the address bar is "w3schools.com/nodejs/nodejs\_http.asp". The page content is titled "The Built-in HTTP Module". It explains that Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP). It shows how to include the HTTP module using the `require()` method:

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

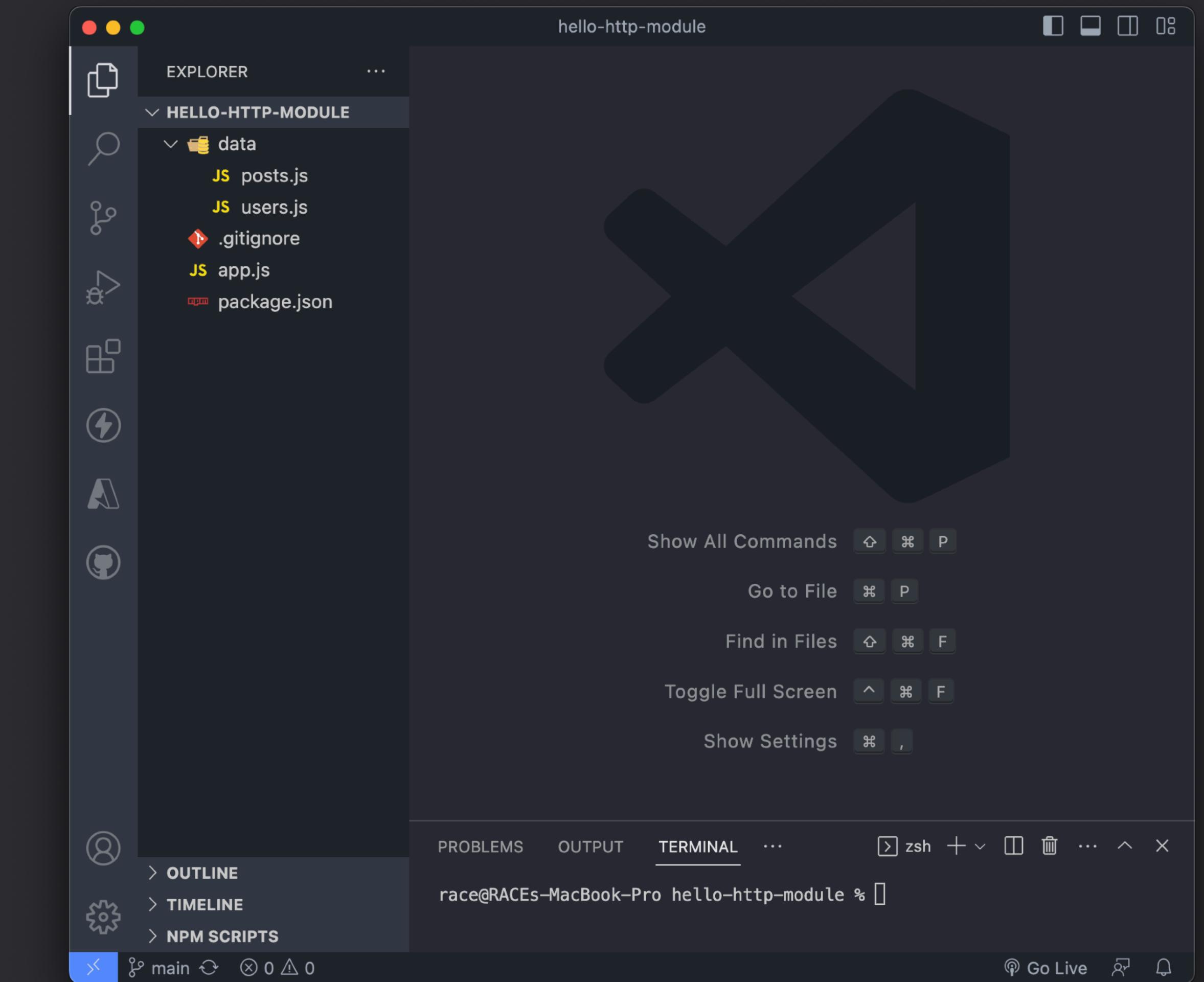
Below this, there is a section titled "Node.js as a Web Server" which states that the HTTP module can create an HTTP server that listens to server ports and gives a response back to the client. It suggests using the `createServer()` method to create an HTTP server:

```
Use the createServer() method to create an HTTP server:
```

```
// Import the built-in Node.js 'http' module.  
import http from "http";  
  
// Here, we create an HTTP server using 'createServer' function.  
// Takes a callback function that is called when a page is requested.  
const app = http.createServer((request, response) => {  
    // Set status code and headers for the response.  
    response.statusCode = 200;  
    response.setHeader("Content-Type", "text/plain");  
    // Send a message as the response.  
    response.end("Working with HTTP Module and routing");  
});  
  
// Here, we define the port number the server should listen on.  
const port = 3000;  
  
// Finally, we start the server by calling the 'listen' function.  
app.listen(port, () => {  
    // Once the server is started, we display a message in the terminal.  
    console.log(`Server is running at http://localhost:\${port}`);  
});
```

# Øvelse

## Hello HTTP Module



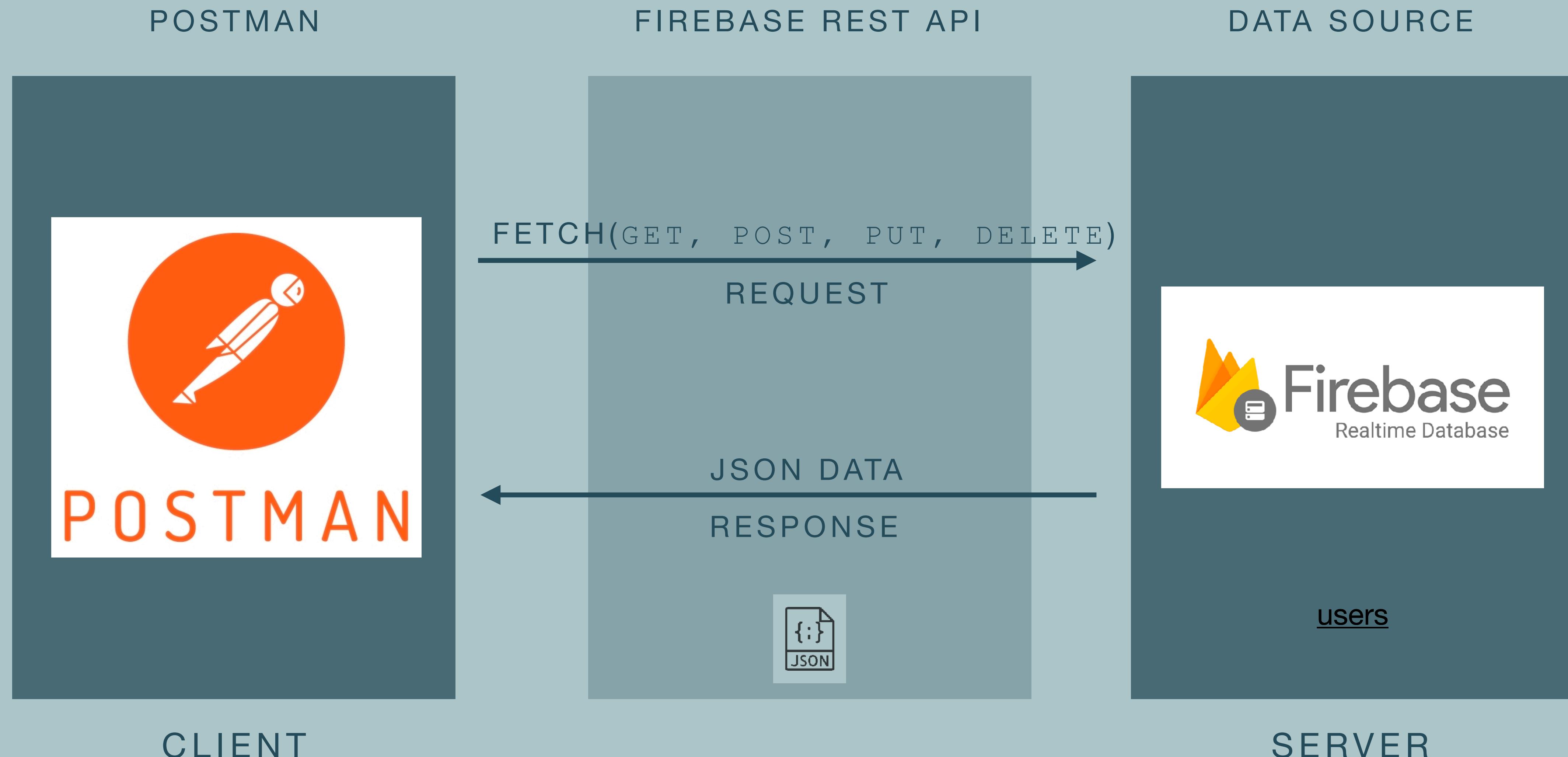


# POSTMAN

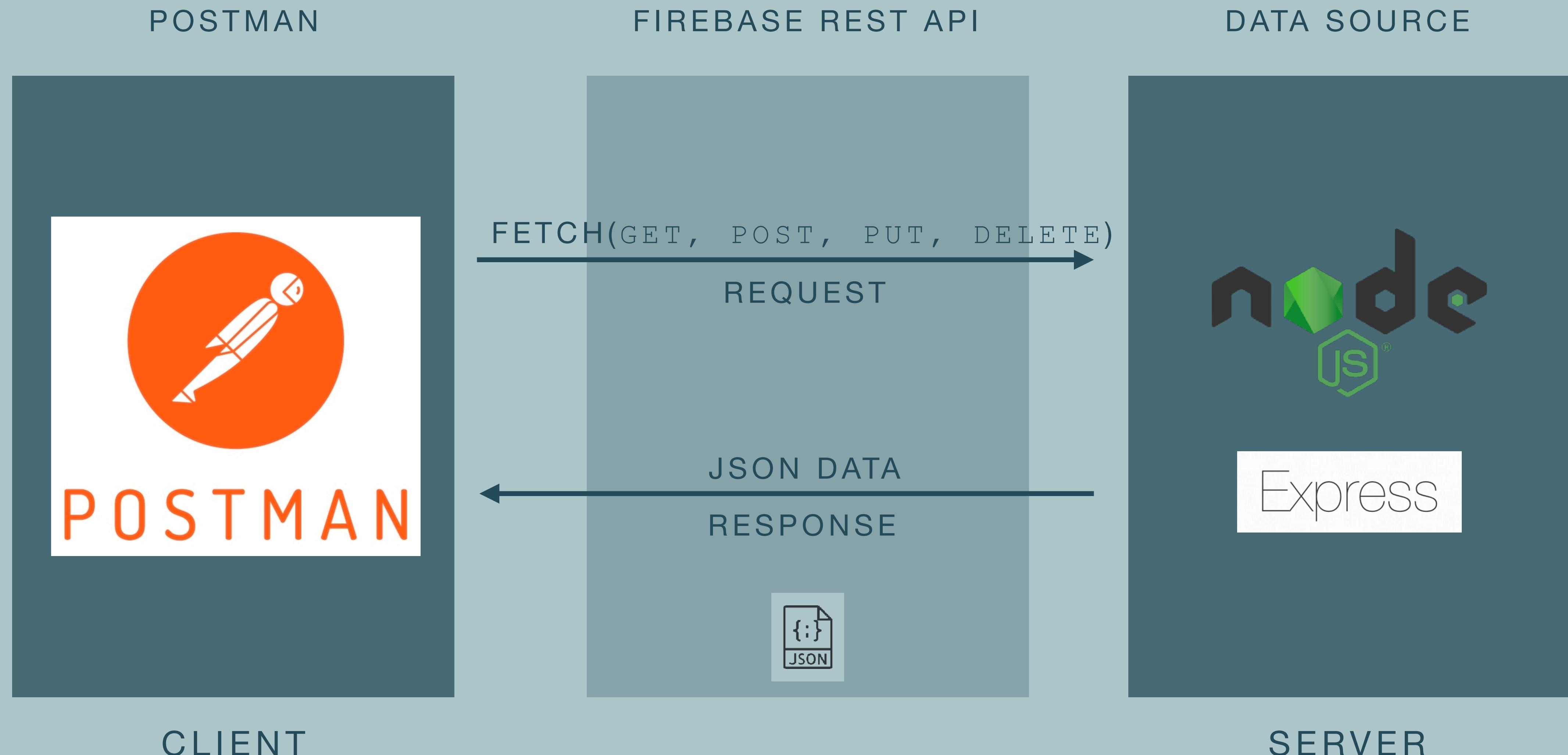
## Getting Started with Postman

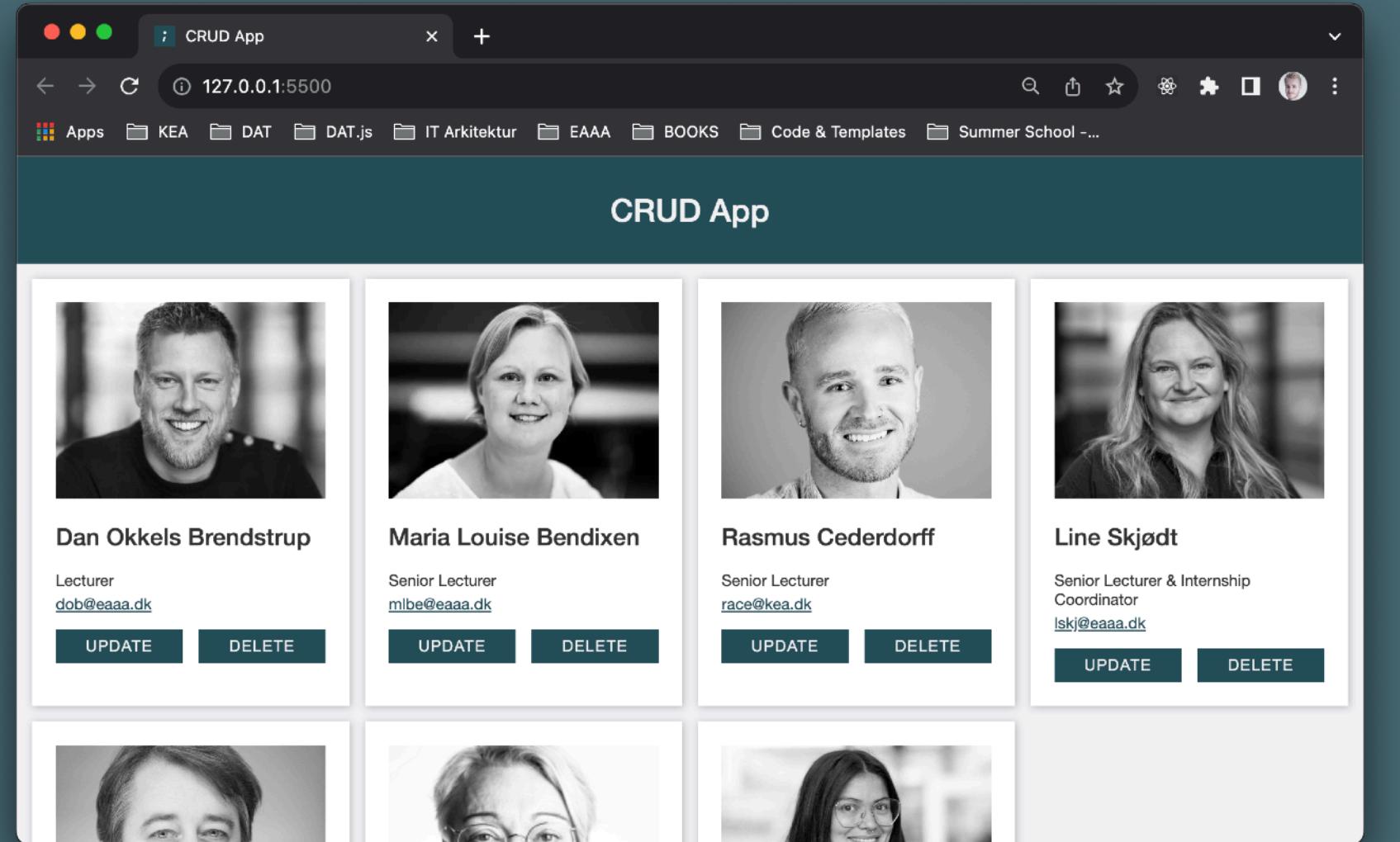
- Postman is a software tool for working with APIs.
- We can use it as client to test our backend API's and to test APIs in general.
- It helps developers send requests and receive responses from APIs.
- Postman tests API functionality and documents how to use APIs.
- It simplifies API development and testing.
- Developers collaborate using Postman to work together effectively.

# Web Development



# Web Development

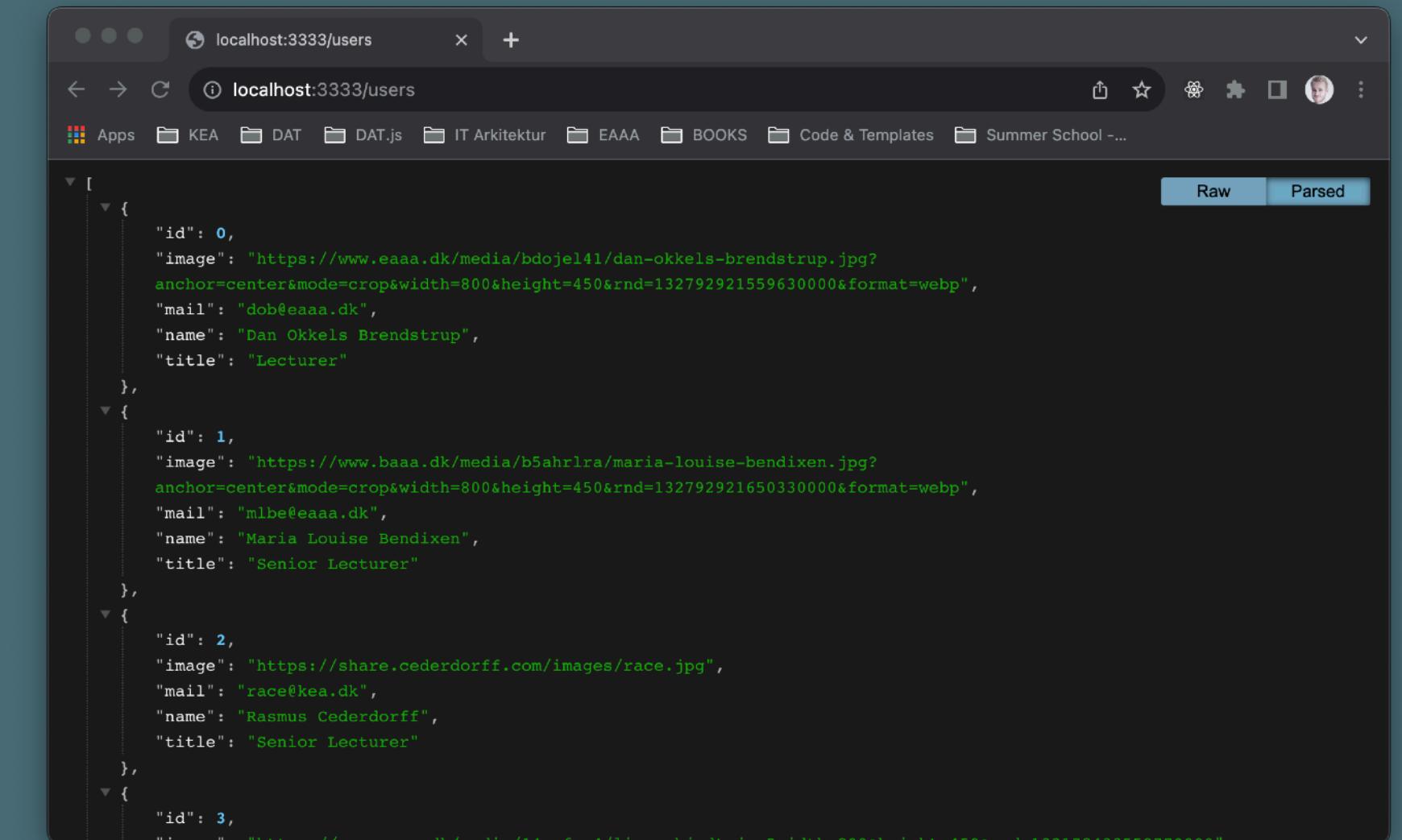




```
app.js — dat-js-crud-intro
JS app.js > ...
25
26 // Read (GET) all users from backend using REST API
27 async function readUsers() {
28   const response = await fetch(`${endpoint}/users`);
29   const data = await response.json();
30   return data;
31 }
32
33 // Create HTML and display all users from given list
34 function displayUsers(list) {
35   // reset <section id="users-grid" class="grid-container">...</section>
36   document.querySelector("#users-grid").innerHTML = "";
37   //loop through all users and create an article with content for each
38   for (const user of list) {
39     document.querySelector("#users-grid").insertAdjacentHTML(
40       "beforeend",
41       /*html*/
42       <article>
```

Frontend

REQUEST



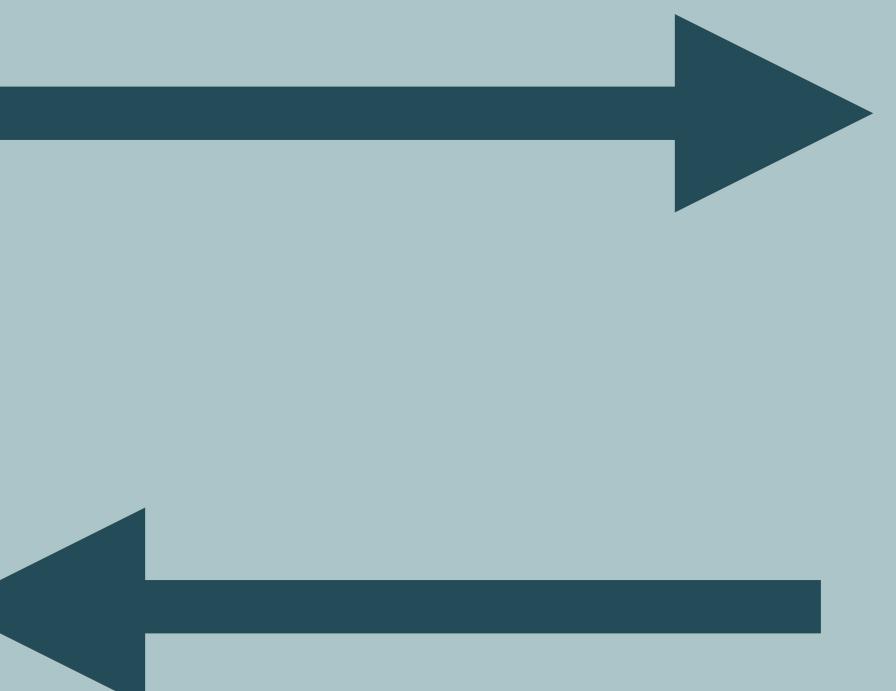
```
app.js — node-users-rest-api
JS app.js > ⚡ app.put("/users/:id") callback
19 // READ all users - "/users" : GET
20 app.get("/users", async (request, response) => {
21   const data = await fs.readFile("data.json");
22   const users = JSON.parse(data);
23   response.json(users);
24 });
25
26 // CREATE user - "/users" : POST
27 app.post("/users", async (request, response) => {
28   const newUser = request.body;
29   newUser.id = new Date().getTime();
30   console.log(newUser);
31
32   const data = await fs.readFile("data.json");
33   const users = JSON.parse(data);
34
35   users.push(newUser);
36   console.log(newUser);
```

Backend

The screenshot shows a browser-based API testing interface. On the left, a sidebar lists various workspace collections and environments. In the main area, a request is being made to `http://localhost:3333/users`. The request method is `GET`, and the URL is `http://localhost:3333/users`. Below the request, there are tabs for `Params`, `Auth`, `Headers (8)`, `Body`, `Pre-req.`, `Tests`, and `Settings`. The `Body` tab is selected, showing a table with columns `Key`, `Value`, `Description`, and `...`. The response section shows a `200 OK` status with a `26 ms` latency and `1.59 KB` size. The response body is displayed in `Pretty` format, showing a JSON array of user objects. The JSON is as follows:

```
1
2 [
3   {
4     "id": 0,
5     "image": "https://www.eaaa.dk/media/bdoje141/dan-okkels-brendstrup.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921559630000&format=webp",
6     "mail": "dob@eaaa.dk",
7     "name": "Dan Okkels Brendstrup",
8     "title": "Lecturer"
9   },
10  {
11    "id": 1,
12    "image": "https://www.baaa.dk/media/b5ahrlra/maria-louise-bendixen.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921650330000&format=webp",
13    "mail": "mlbe@eaaa.dk",
14    "name": "Maria Louise Bendixen",
15    "title": "Senior Lecturer"
16  },
17  {
18    "id": 2,
19    "image": "https://share.cederdorff.com/images/race.jpg",
20    "mail": "race@kea.dk",
21    "name": "Rasmus Cederdorff",
22    "title": "Senior Lecturer"
23  },
24  {
25    "id": 3,
26    "image": "https://www.eaaa.dk/media/14qpfeq4/line-skjodt.jpg?width=800&height=450&rnd=13317843359770000",
27    "mail": "lskj@eaaa.dk",
28    "name": "Line Skjodt",
29    "title": "Senior Lecturer & Internship Coordinator"
30  }
31 ]
```

REQUEST



RESPONSE

The screenshot shows a browser window displaying the JSON response received from the API. The response is identical to the one shown in the testing tool. Below the browser, a code editor is open with a file named `app.js`. The code contains JavaScript functions for handling API requests. The `get` function handles a GET request to `/users`, reading data from a `data.json` file and sending it back as a JSON response. The `post` function handles a POST request to `/users`, reading the new user data from the request body, creating a new user object with a timestamp, and adding it to a `users` array in the `data.json` file. The code editor interface includes tabs for `main` and `0`, and various status indicators at the bottom.

```
19 // READ all users - "/users" : GET
20 app.get("/users", async (request, response) => {
21   const data = await fs.readFile("data.json");
22   const users = JSON.parse(data);
23   response.json(users);
24 });
25
26 // CREATE user - "/users" : POST
27 app.post("/users", async (request, response) => {
28   const newUser = request.body;
29   newUser.id = new Date().getTime();
30   console.log(newUser);
31
32   const data = await fs.readFile("data.json");
33   const users = JSON.parse(data);
34
35   users.push(newUser);
36   console.log(newUser);
37 }
```

Backend

Client



# Node.js File System

Allows us to work with the file system on  
the computer or server

The screenshot shows a web browser window with the title "Node.js File System Module". The URL in the address bar is "w3schools.com/nodejs/nodejs\_filesystem.asp". The page content is from w3schools.com and discusses the Node.js file system module. It includes a code example for reading and writing JSON files using fs.promises.

## Node.js as a File Server

The Node.js file system module allows you to work with the file system on your computer.

To include the File System module, use the `require()` method:

```
var fs = require('fs');
```

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

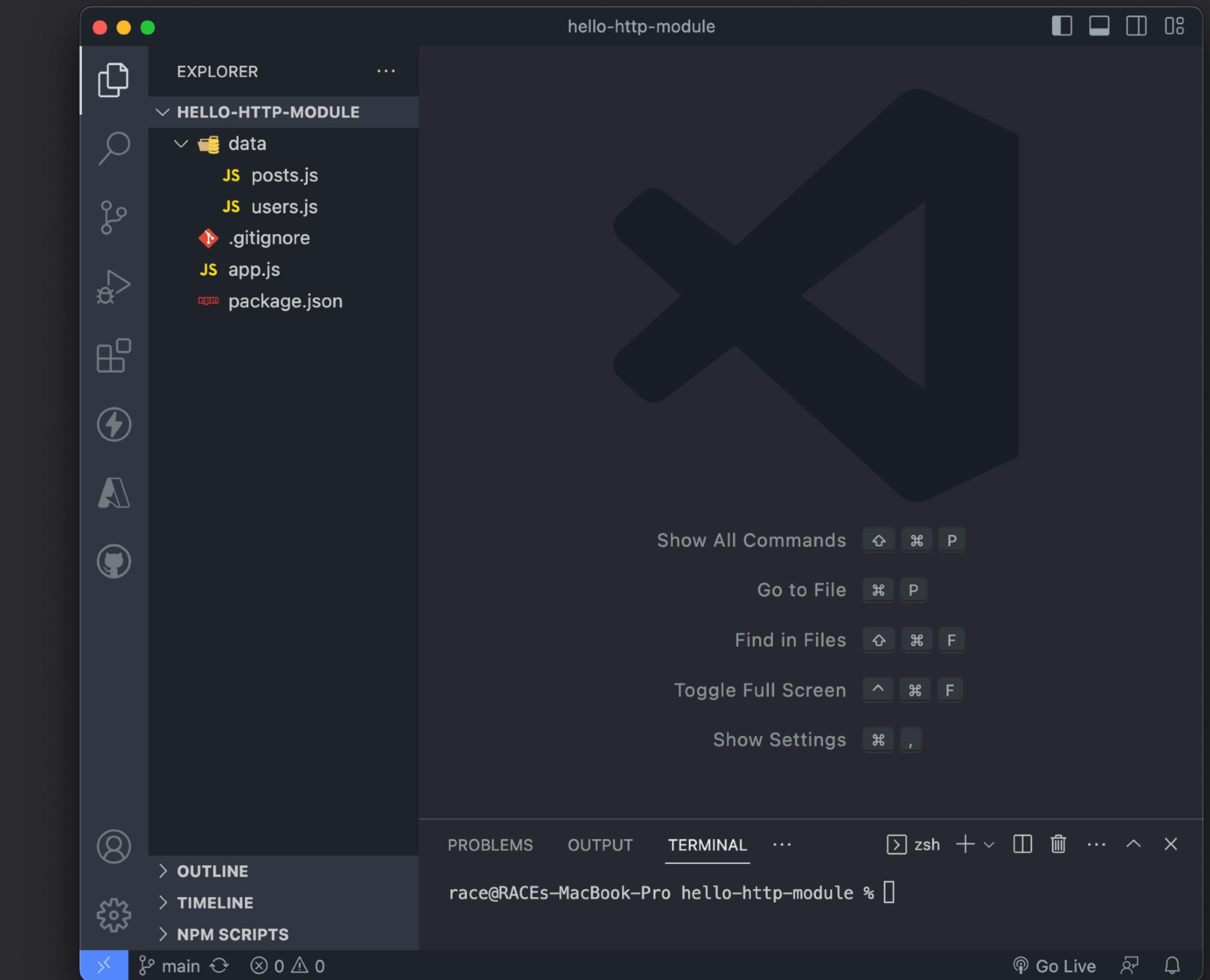
## Read Files

```
import fs from "fs/promises";

// Læs fra JSON
const json = await fs.readFile("data/users.json");
console.log(json);
// Parse til JavaScript
const users = JSON.parse(json);
console.log(users);
// Tilføj "user" til "users"
users.push({ name: "Peter Lind", title: "Senior Lecturer" });
// Konverter users til JSON igen
const usersJSON = JSON.stringify(users);
// Skriv til JSON-fil
await fs.writeFile("data/users.json", usersJSON);
```

# Øvelse

## Node.js File System



# Wrap Up

- NPM & JavaScript Libraries
  - Node.js NPM
  - Node.js Upload Files
  - Node.js Send an Email
  - Brug ChatGPT og Google
- Bunden forudsætning: Udvikl REST API med Node.js & Express (Se Fronter)



# Code every Day

