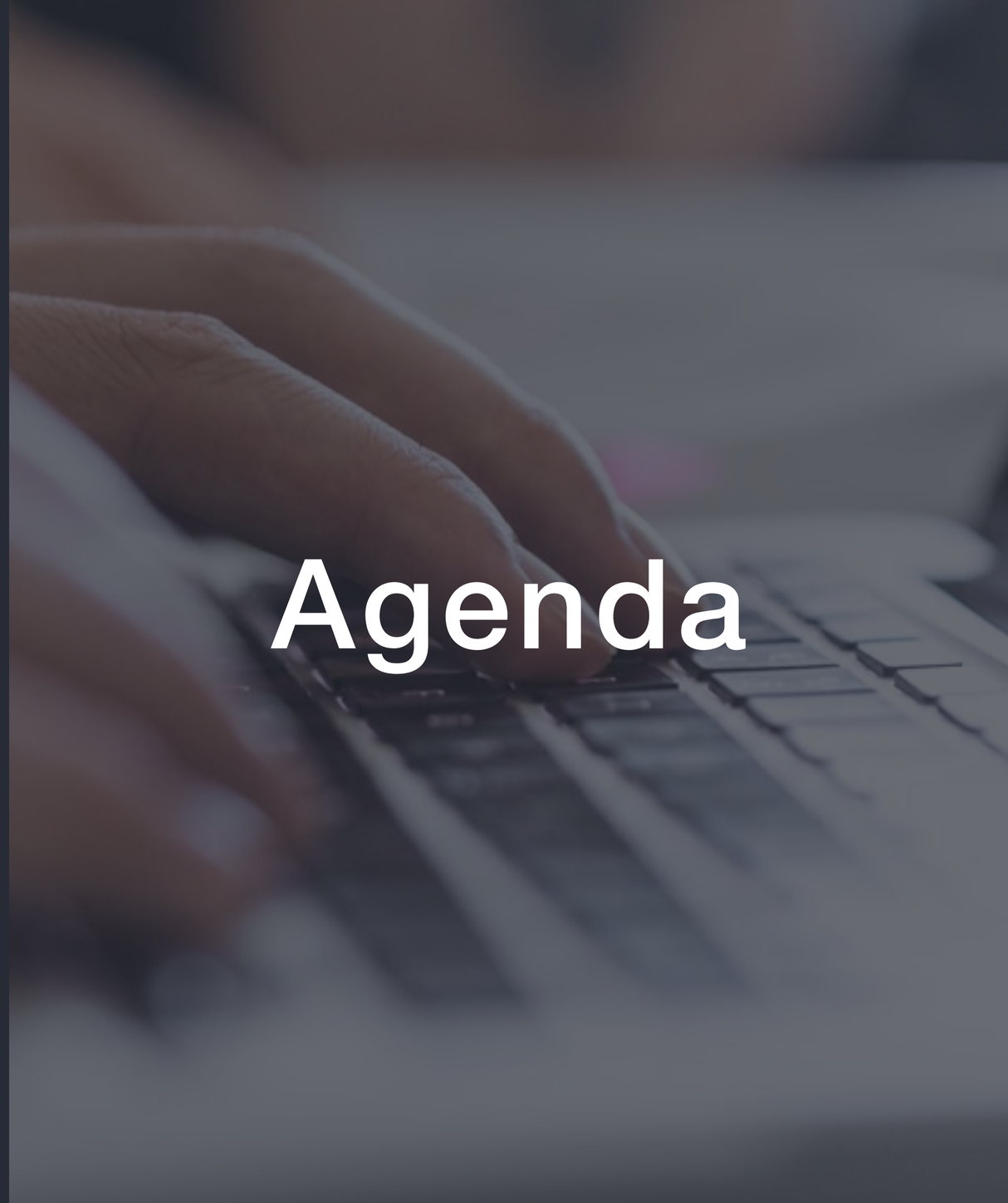
RACE.

ERHVERVSAKADEMI AARHUS
BUSINESS ACADEMY AARHUS

Agenda

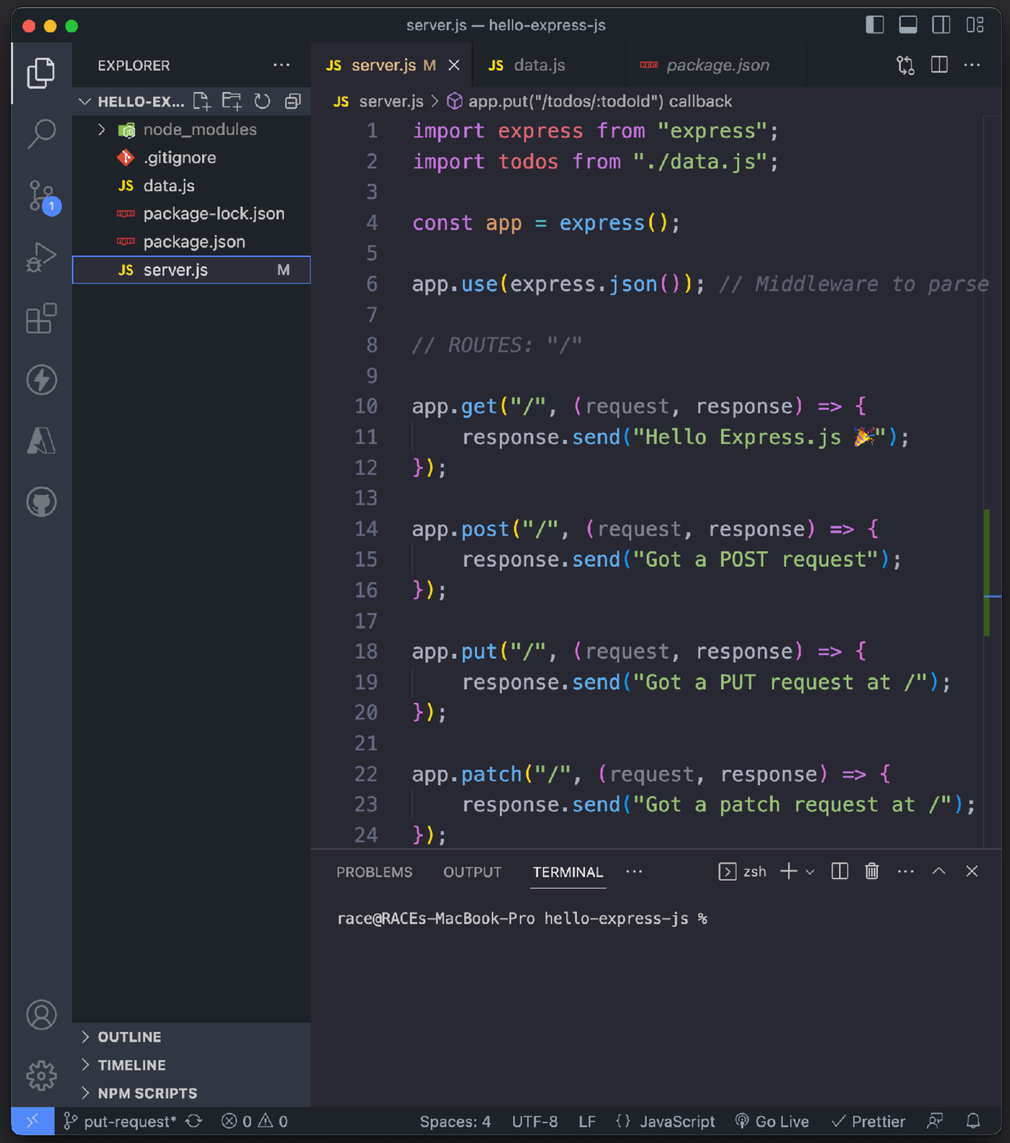
1. Node Runtime Environment
2. Node.js & Express.js
3. Node.js & MySQL
4. REST API
5. Node Contacts REST API med MySQL
6. Fejlhåndtering & beskeder



Opgaver

- Hello Node.js
- Getting Started with Express.js
- Node.js & MySQL

- Ekstraopgaver
  - Postman - Getting Started
  - Hello HTTP Module (uden Express.js)
  - Node.js File System
  - Introduction to Backend Development with Node.js & Express.js



“The cool thing about JavaScript is that you can create stuff that will put a smile on your face, as a developer.

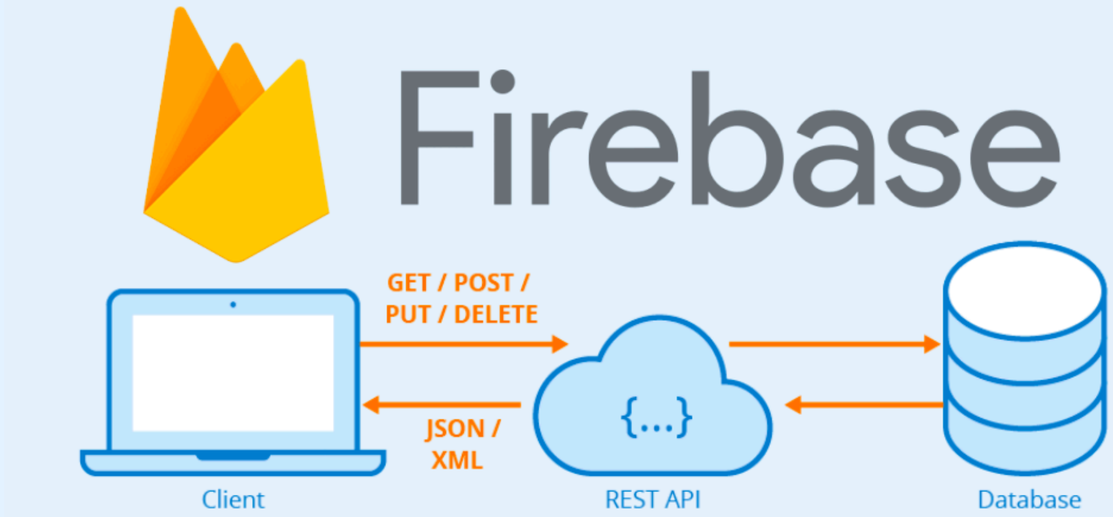
You can create stuff and it will visually look like something, and it'll do stuff, and it makes you feel good, **it makes you fall in love with programming.**”

Lex Fridman

Computer Scientist

https://www.youtube.com/watch?v=GLhyjVZp0cw&t=252s

When someone asks you how to get data from a database in a js code


Firebase
GET / POST /
PUT / DELETE
JSON /
XML
{...}
Client
REST API
Database


Rasmus
Firebase
made with mematic

Formål (første tre gange)

- At udvide vores forståelse og kompetencer indenfor backendudvikling med Node.js.
- Viden og forståelse for CRUD, REST, HTTP-metoder, request og response.
- Implementere REST API'er med Node.js og Express.js efter Best Practices.
- Kommunikere med databaser - først MySQL og dernæst MongDB.
- Praktisk tilgang: Vi skal arbejde med opgaver kombineret med mindre teorioplæg.

API with PHP & MySQL

FRONTEND

REST API

BACKEND

FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE

REST API with Node.js, Express & MongoDB

FRONTEND

REST API

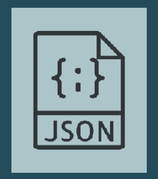
BACKEND

FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE

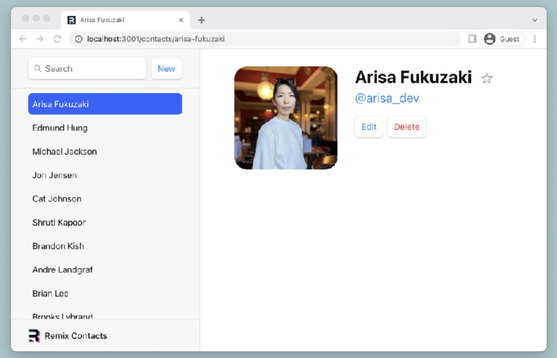


Contact App with REST API

FRONTEND

REST API

BACKEND



FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE

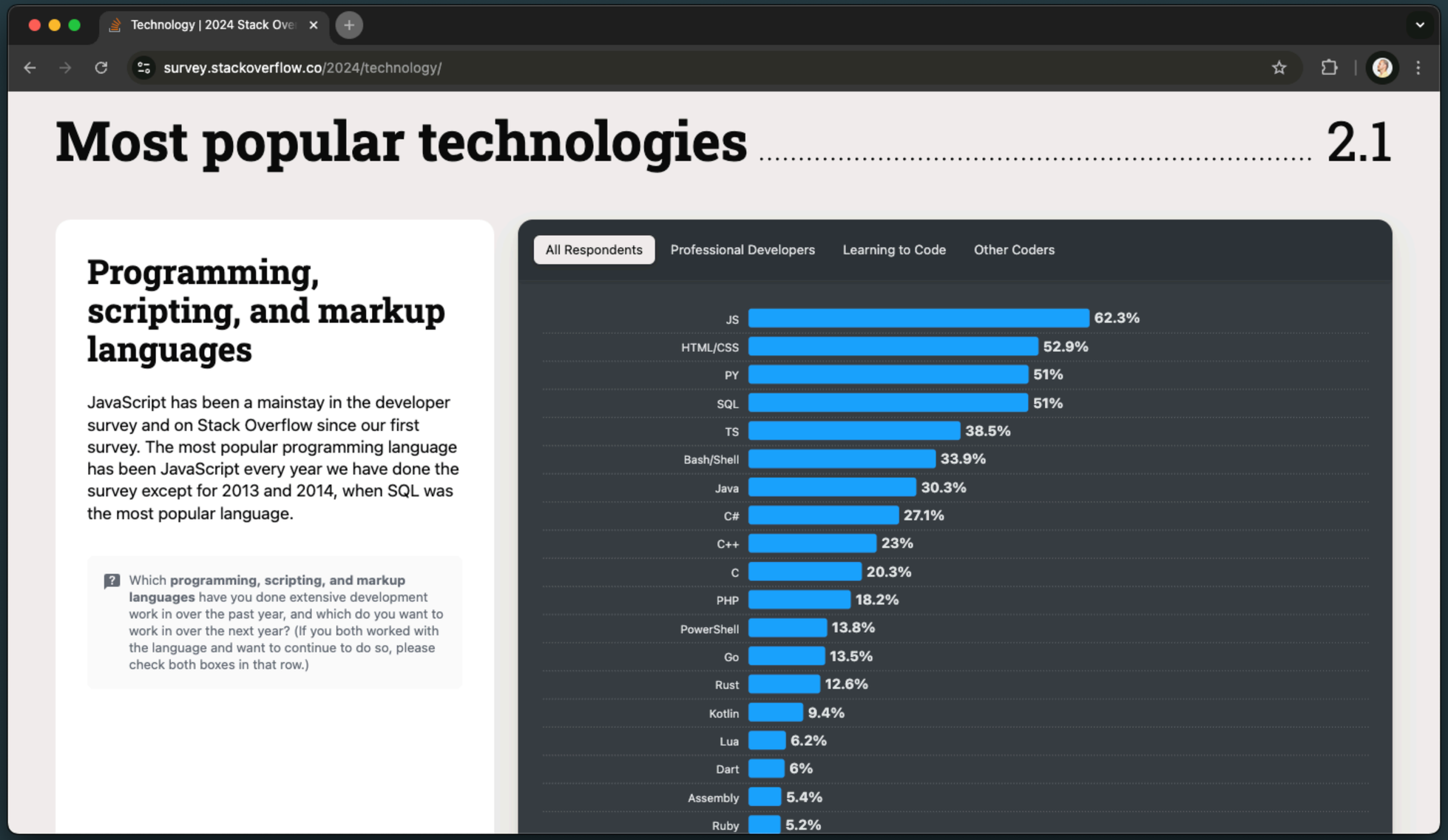
node

Express

mongoDB

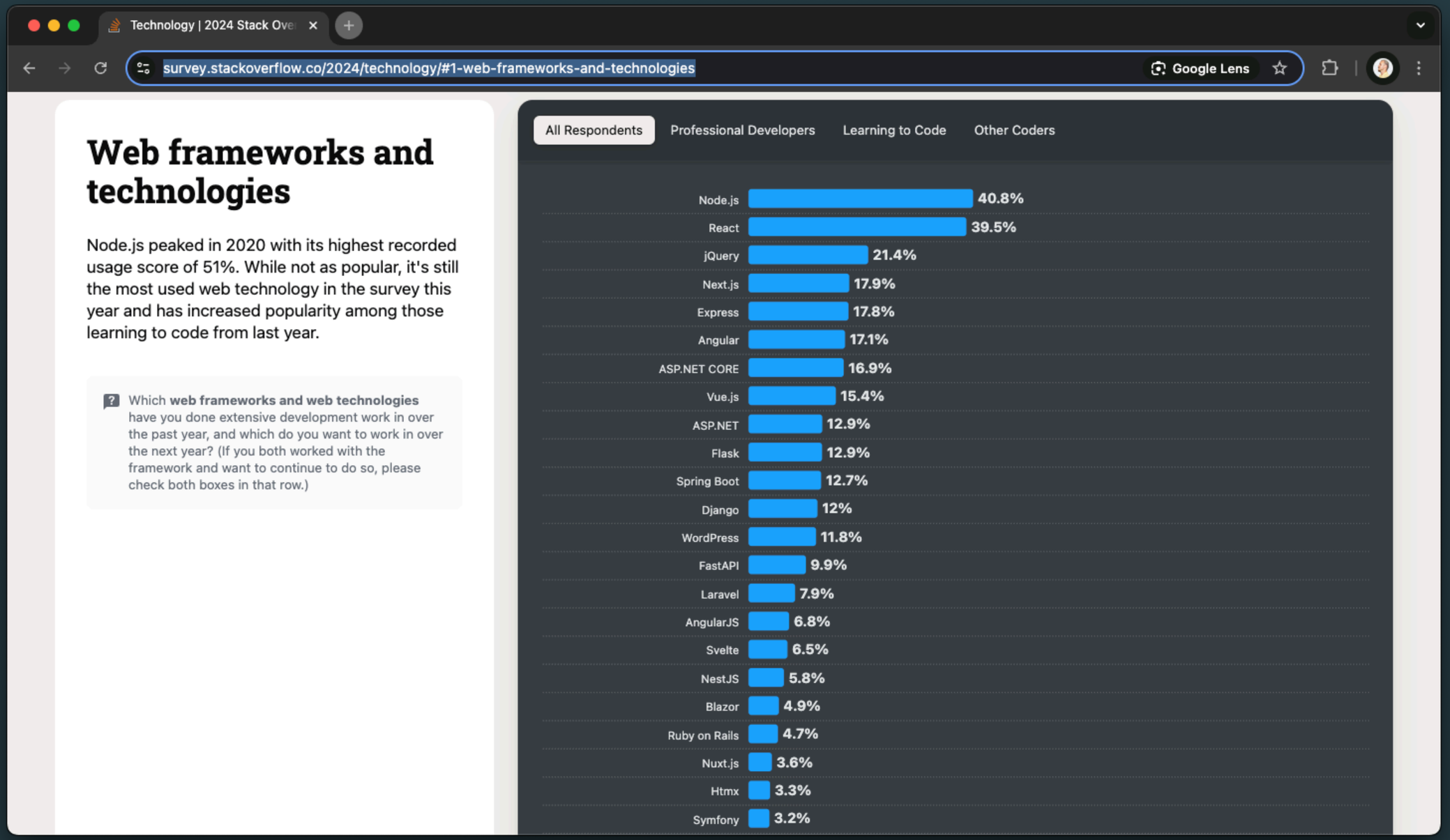
Code Every Day

Men hvorfor gør I det her ved os?

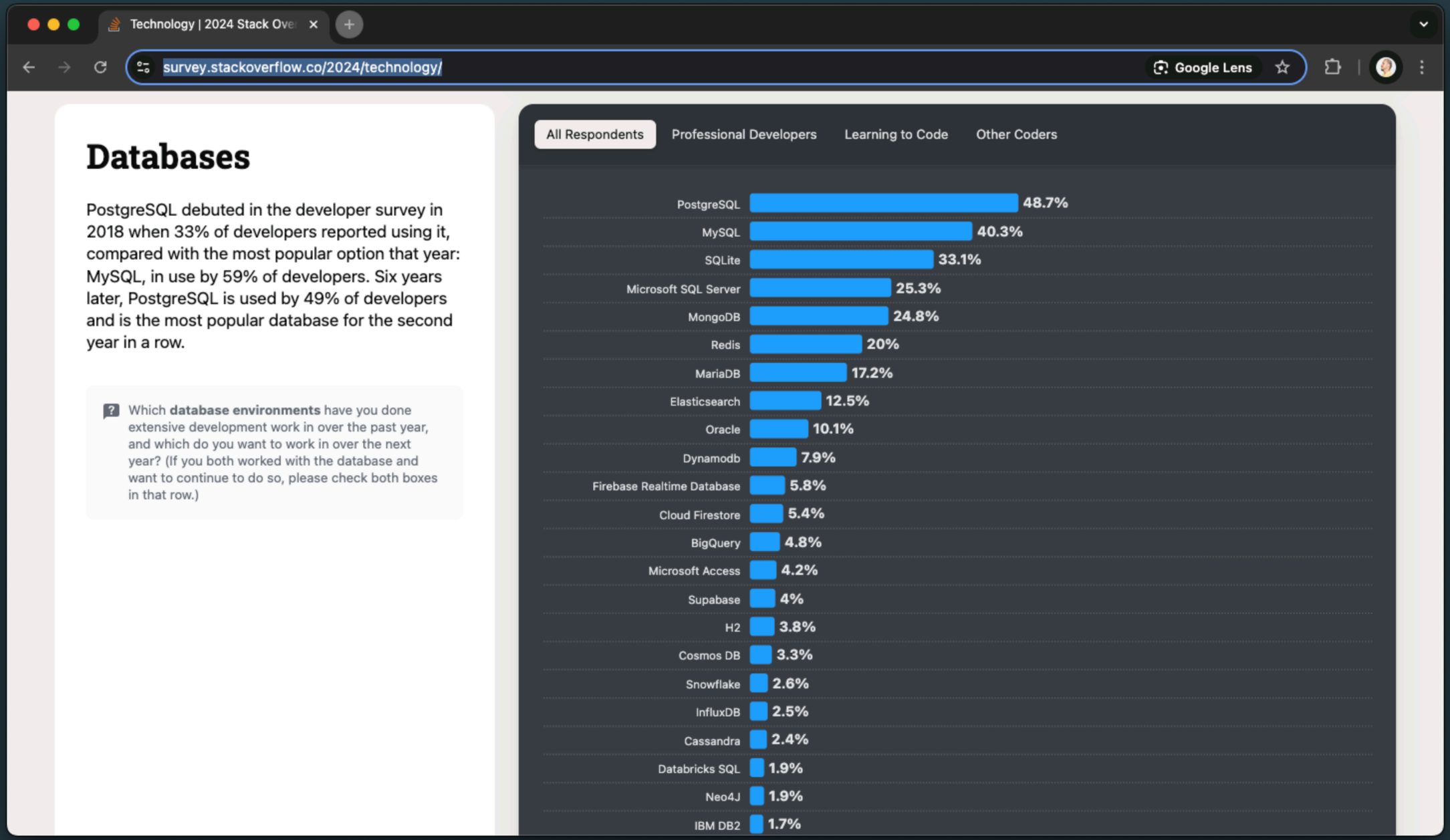


https://survey.stackoverflow.co/2024/technology/#1-programming-scripting-and-markup-languages

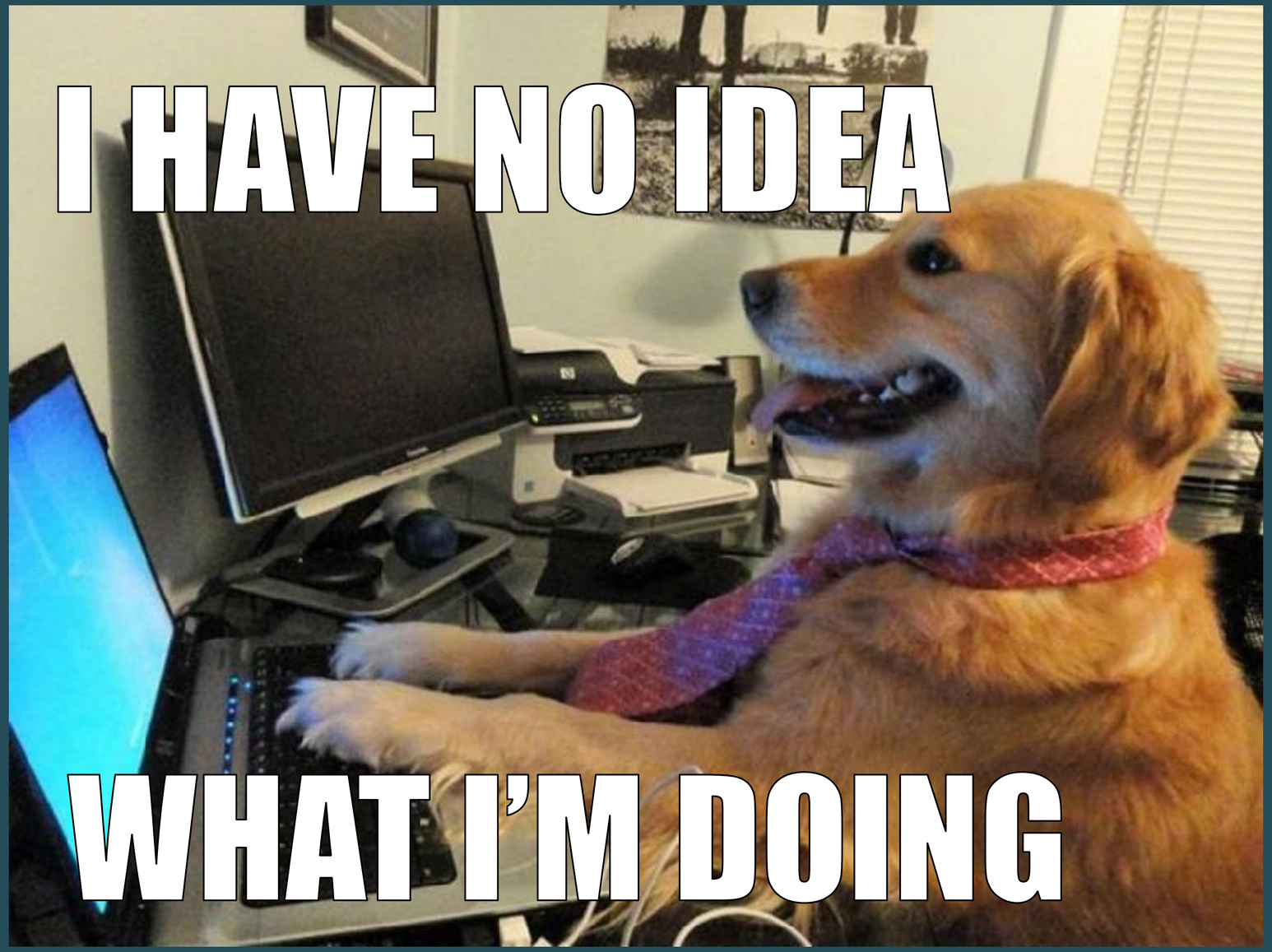

What is the world's most popular language?
A: French
B: English
C: Spanish
D: JavaScript

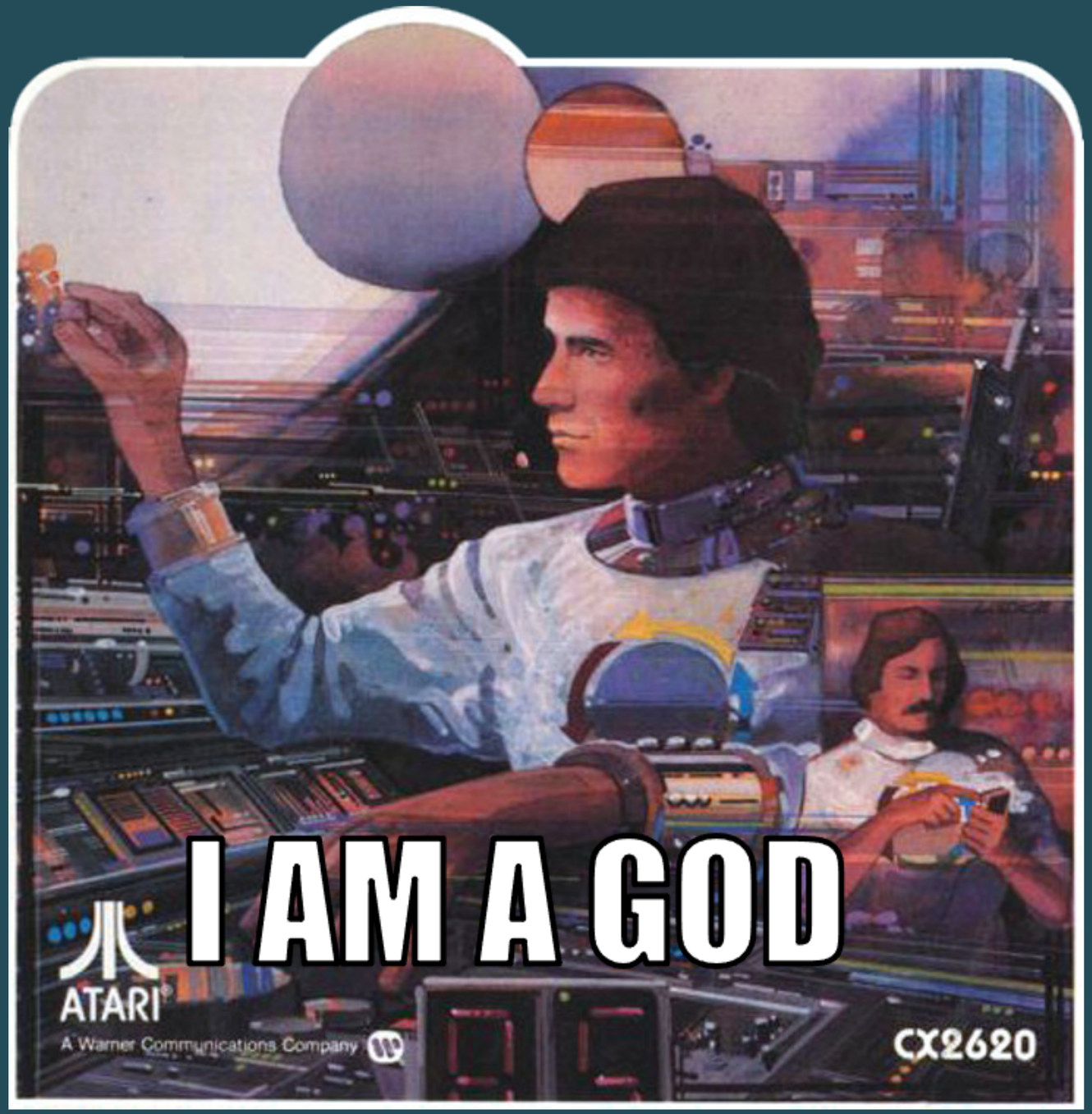


https://survey.stackoverflow.co/2024/technology/#1-web-frameworks-and-technologies

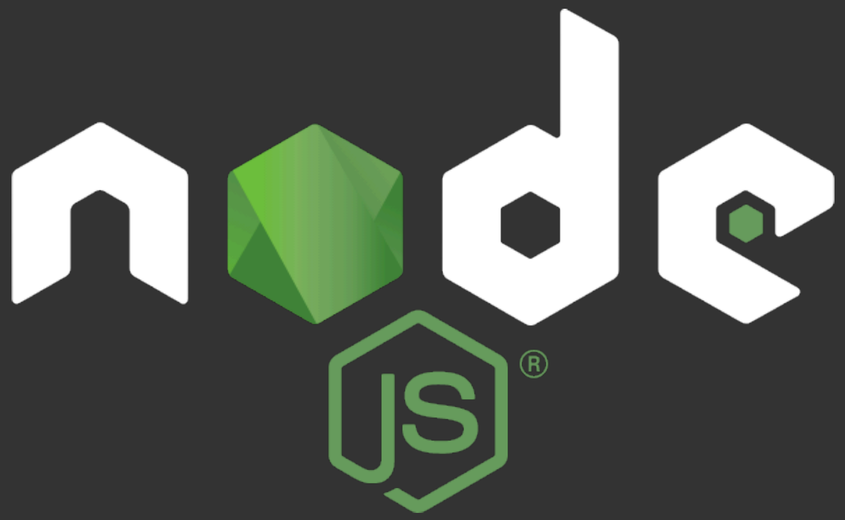


https://survey.stackoverflow.co/2024/technology/#1-databases

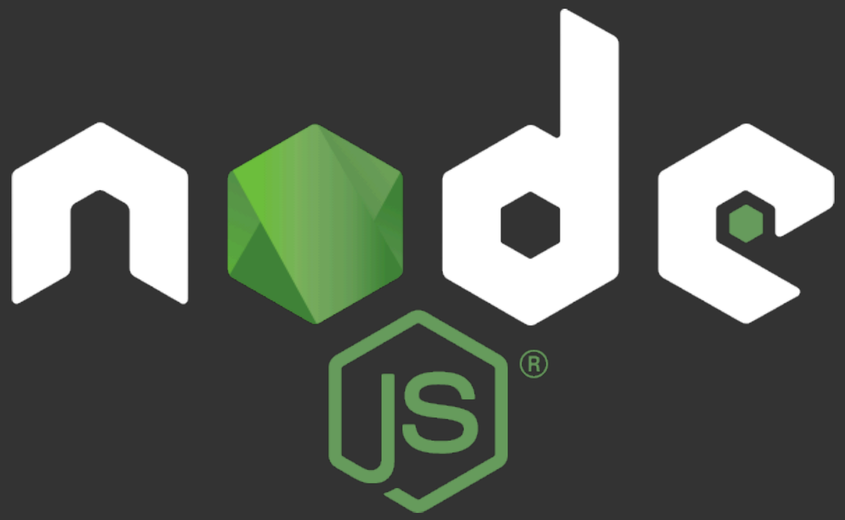

I HAVE NO IDEA
WHAT I'M DOING


I AM A GOD
ATARI
A Warner Communications Company
CX2620

Introduction to

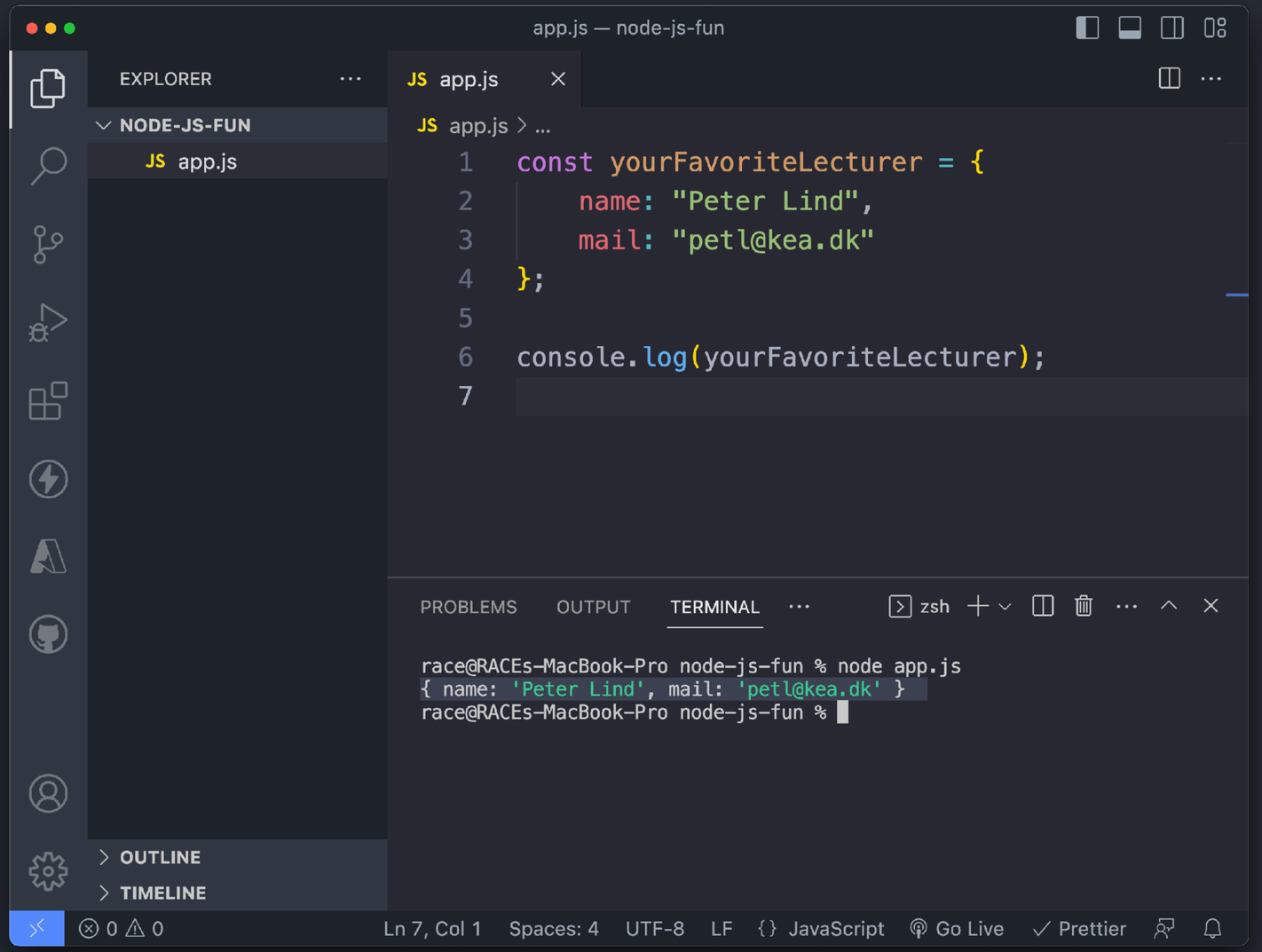


- Server-side JavaScript runtime environment.
- Executes JavaScript outside of the browser.



- 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 🎉

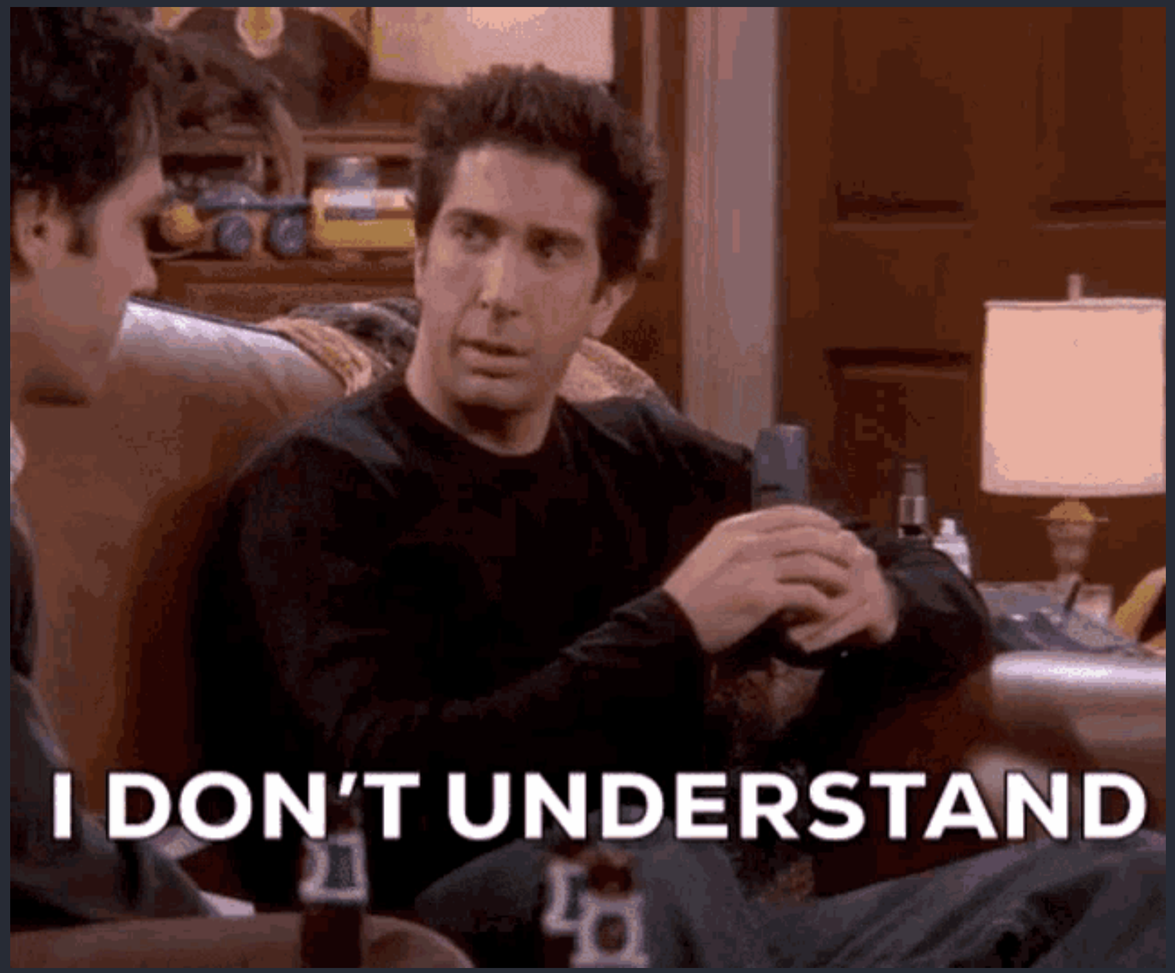


- **JavaScript Beyond Browser**
  Instead of being limited to doing things only on the internet, Node.js lets you use JavaScript to build programs and applications on your computer or server. You get to use the same language you know from web pages.
- **Fast and Efficient**
  Node.js is built to be really 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

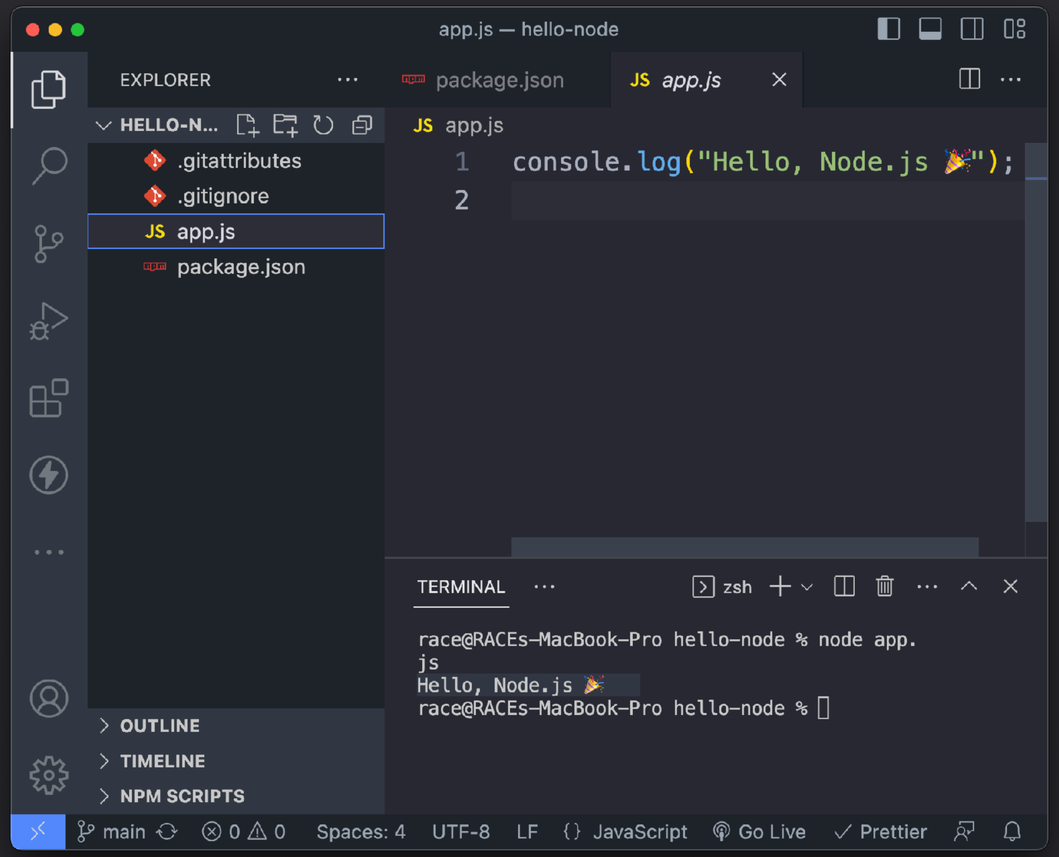
Please,

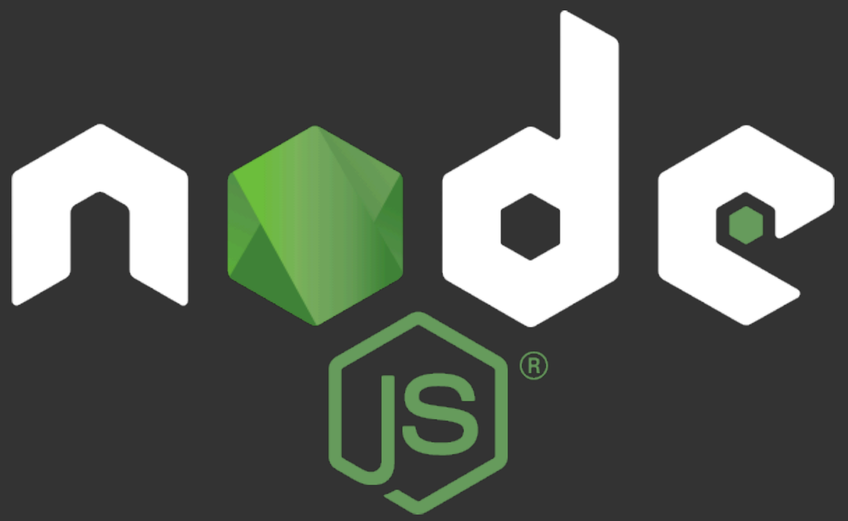
Do not try to understand everything!



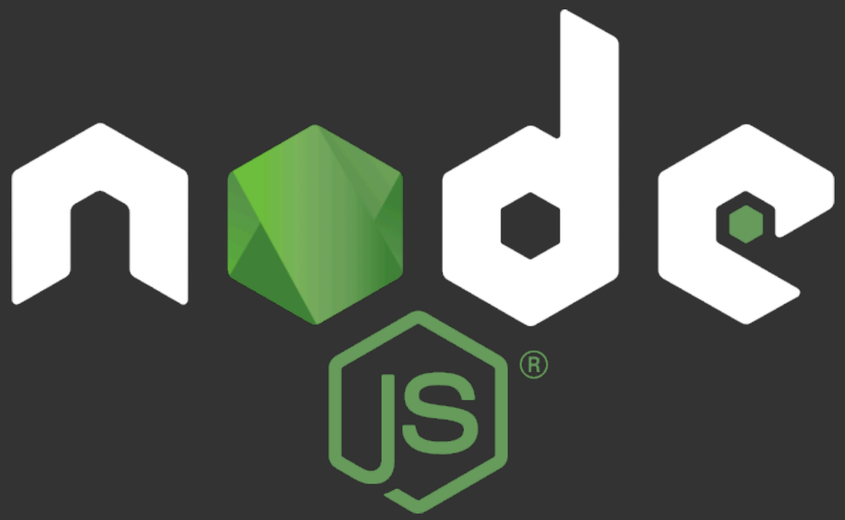
Øvelse

Hello Node.js





Why do we need it?

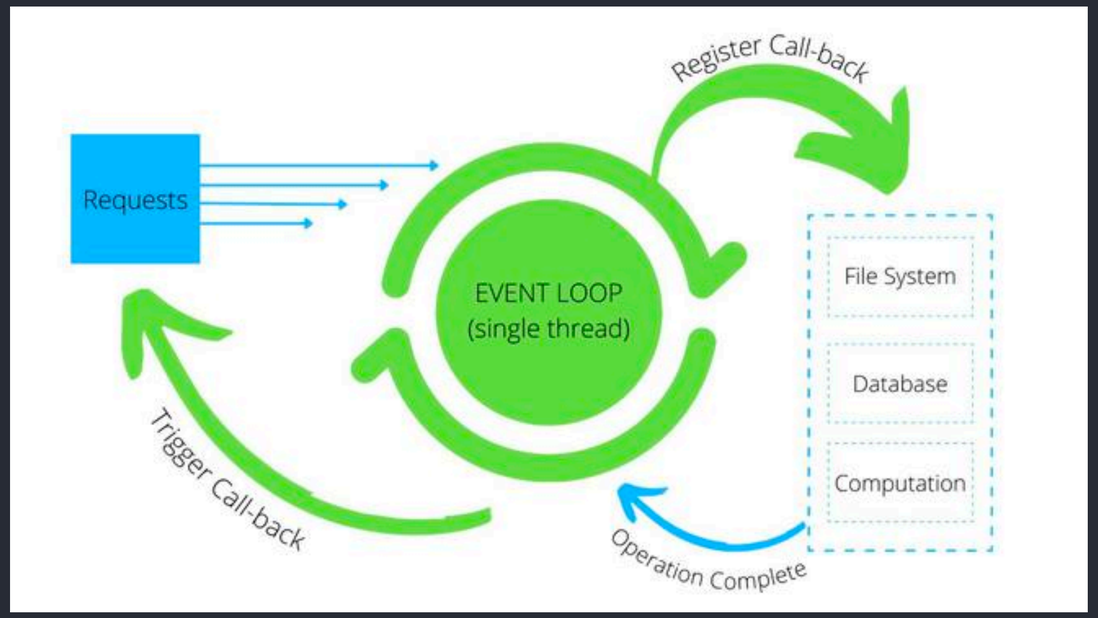


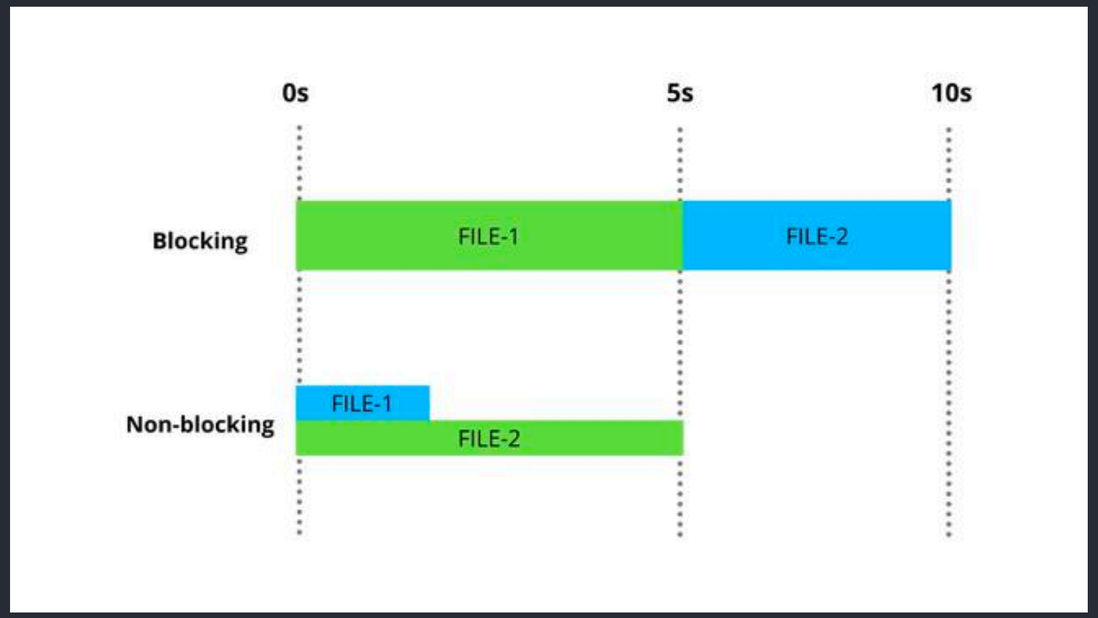
Why do we need it?

- In pairs, discuss and investigate
- Look at code and previous 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.



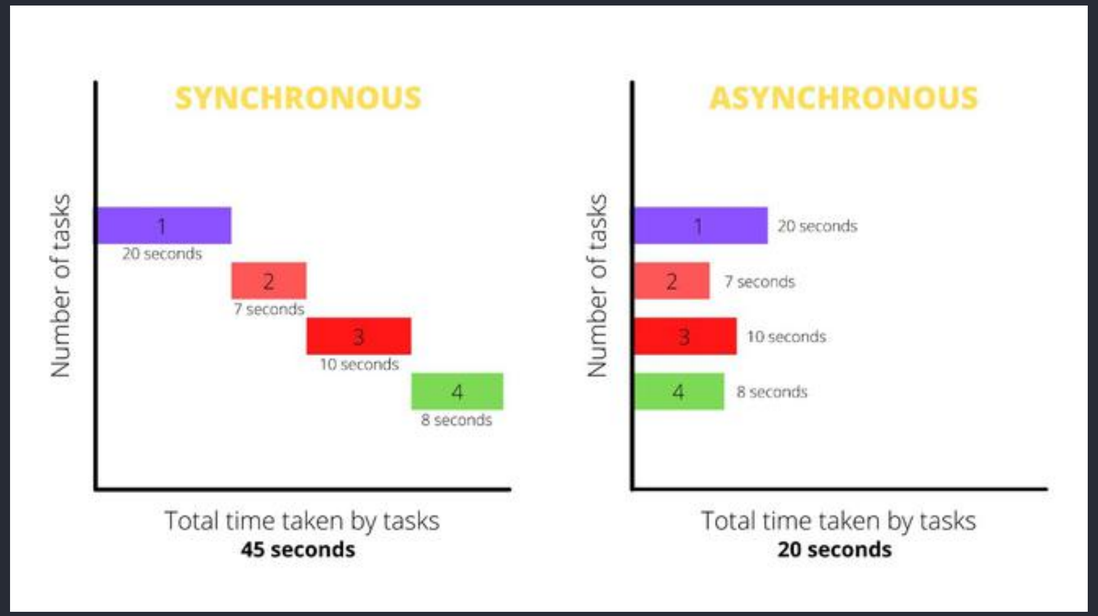


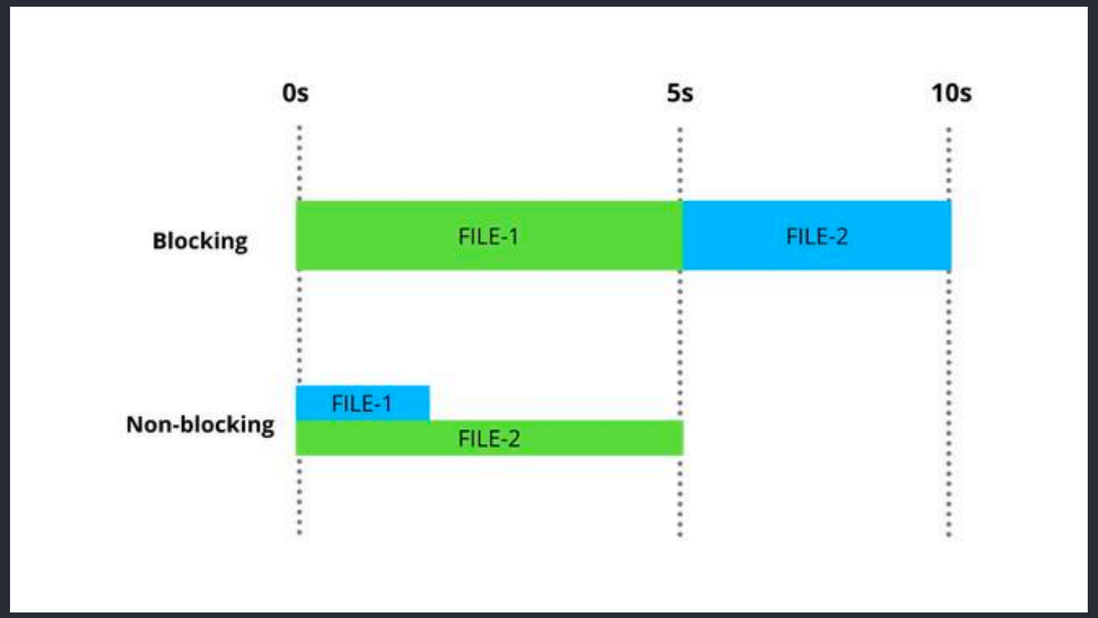
https://www.geeksforgeeks.org/how-node-js-overcome-the-problem-of-blocking-of-i-o-operations/

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.



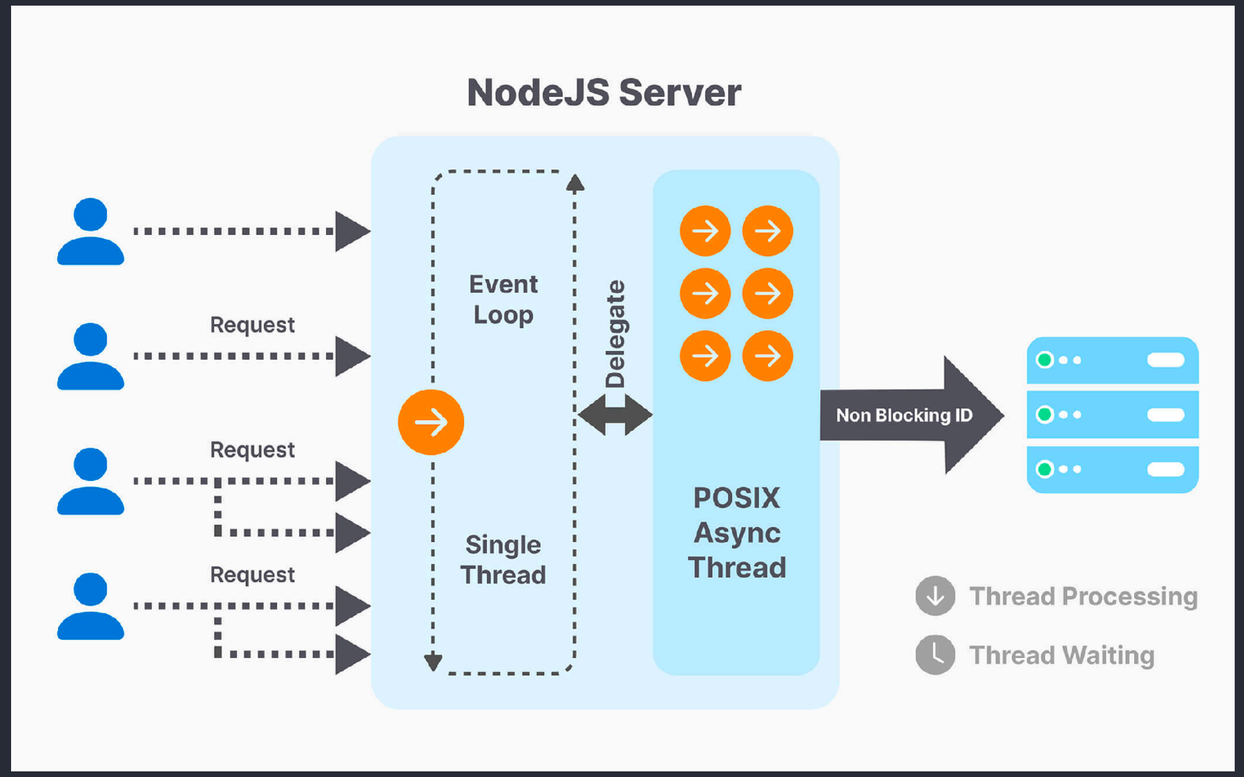


https://www.geeksforgeeks.org/how-node-js-overcome-the-problem-of-blocking-of-i-o-operations/

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.

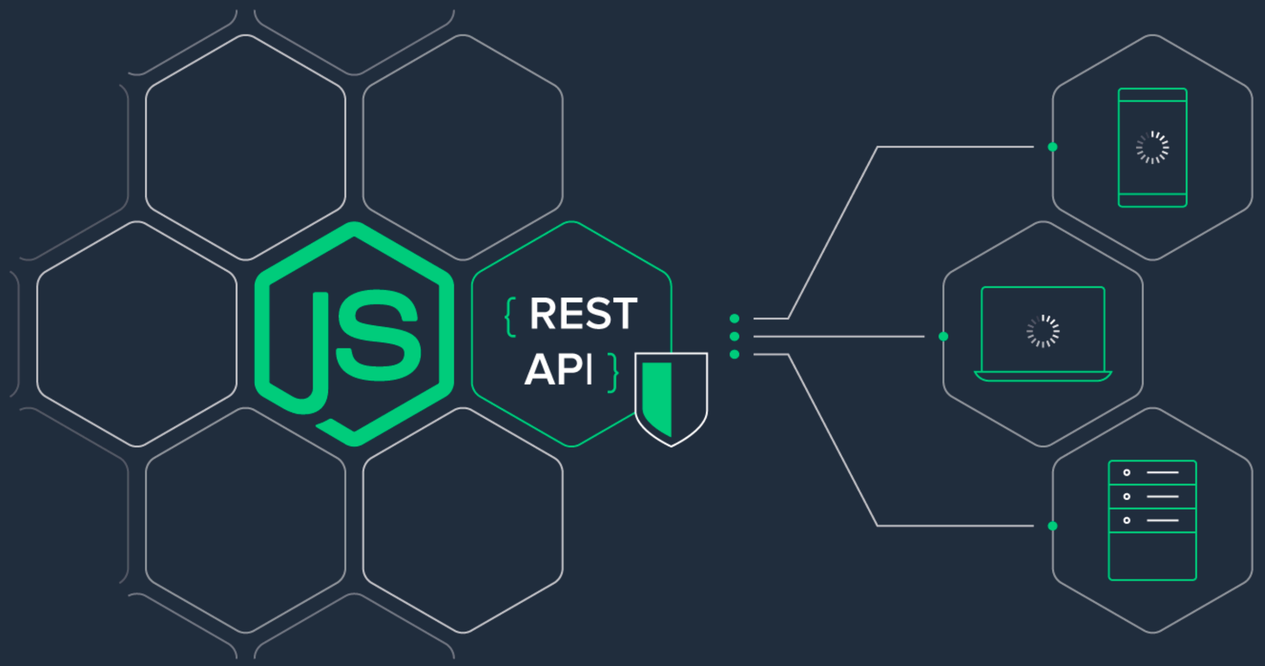


https://www.cygnismedia.com/blog/why-nodejs-is-the-top-choice-for-web-development/

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.

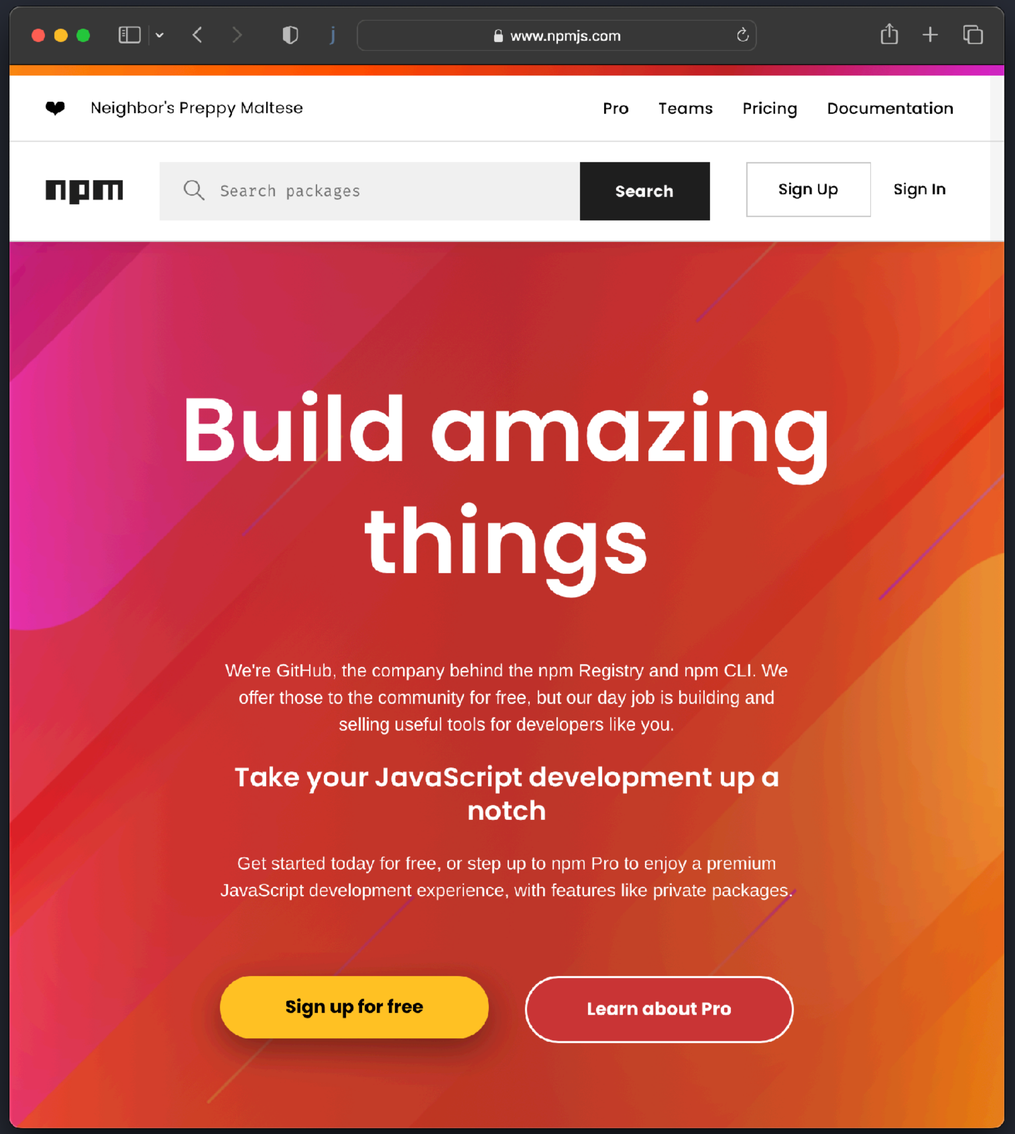


Runtime
Environment

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.


Remix

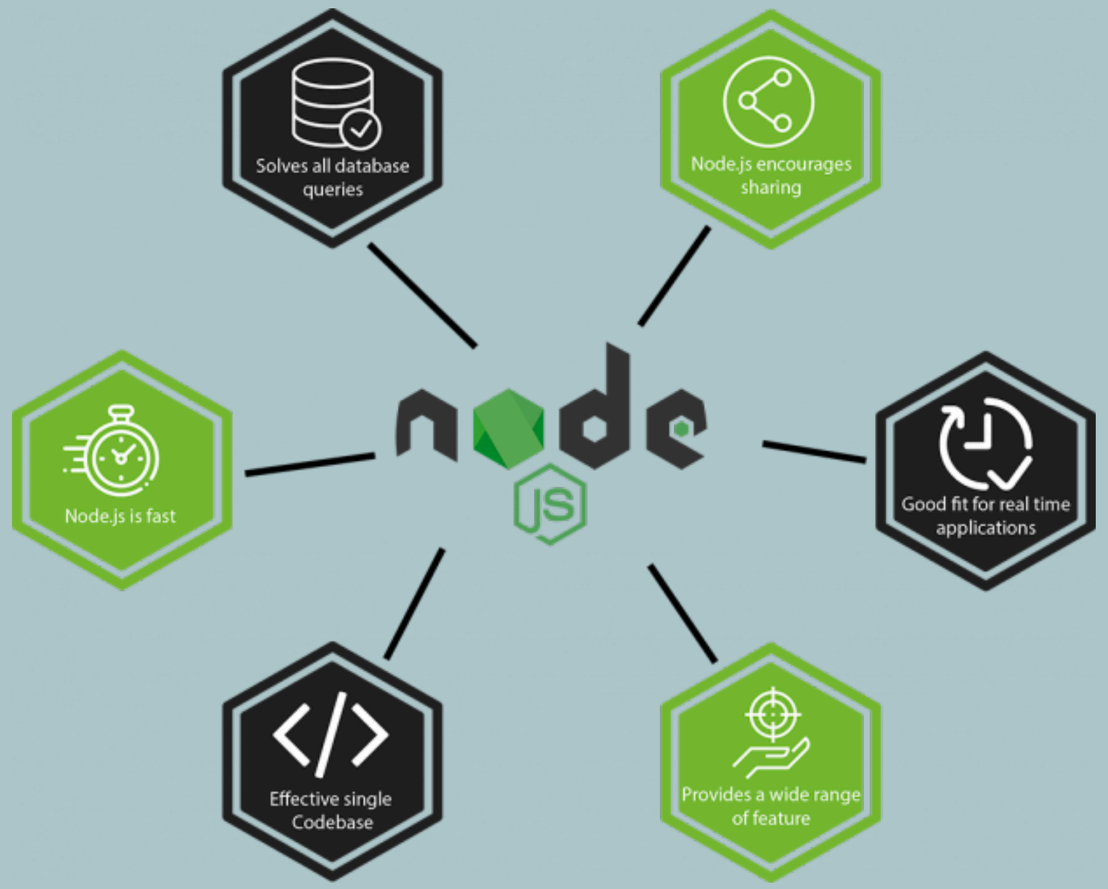


https://www.npmjs.com/

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

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

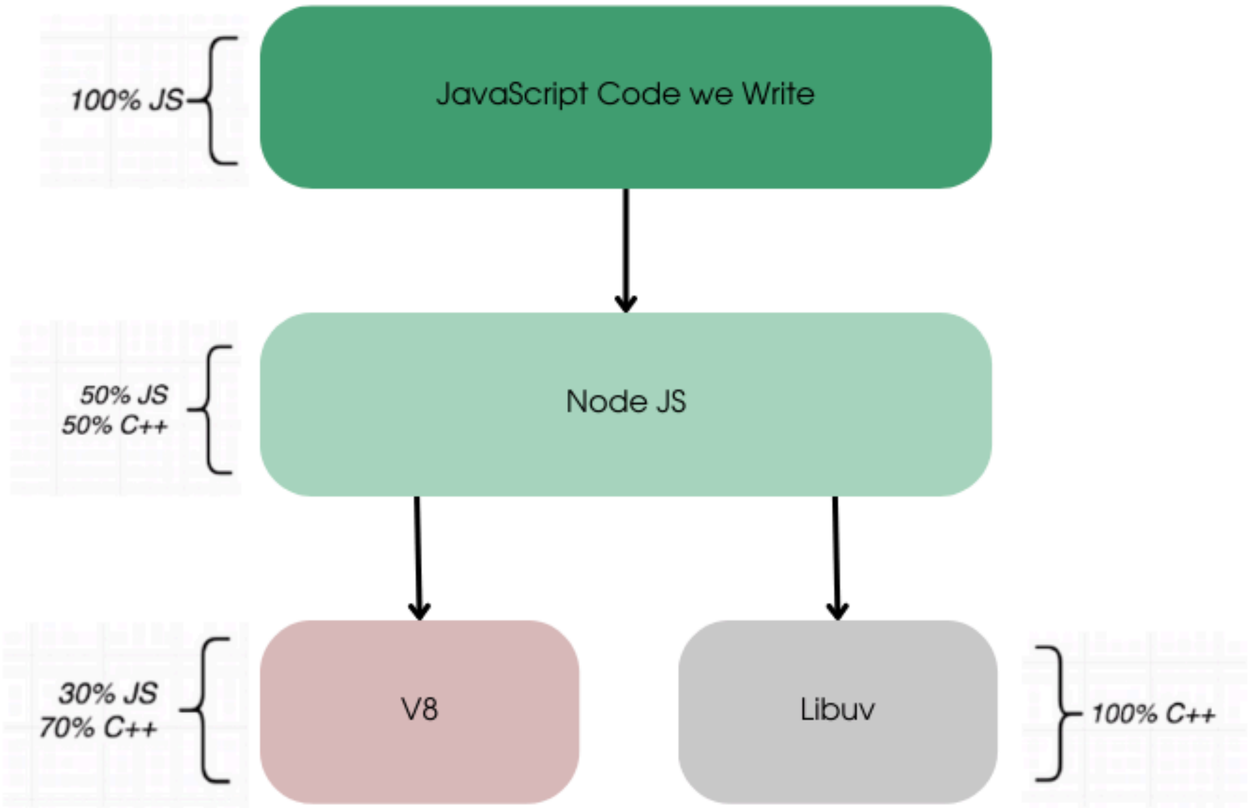
In summary, Node.js provides a runtime environment that allows developers to use JavaScript beyond the browser, enabling them to create high-performance, scalable, and real-time applications for various use cases.

... we can leverage our existing **JavaScript expertise** to develop backend applications, server apps, REST APIs, etc.

Runtime Environment

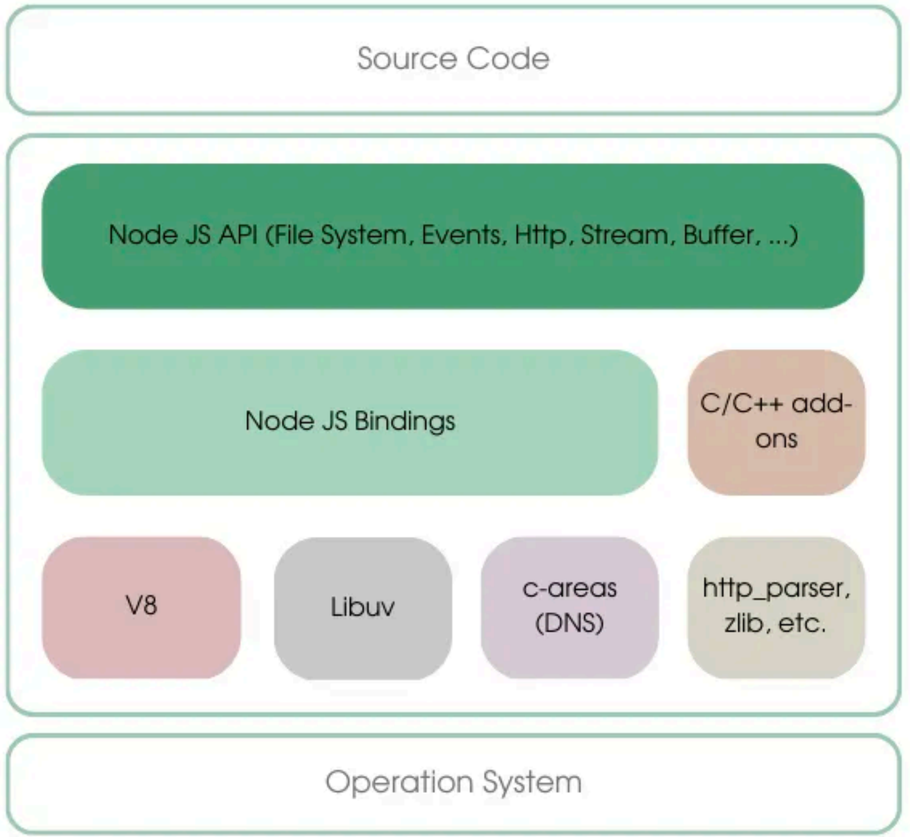
= Runtime Environment + JavaScript Library

Node.js Architecture



https://levelup.gitconnected.com/nodejs-runtime-environment-libuv-library-event-loop-thread-pool-5f5ecadc0318

Node.js Architecture



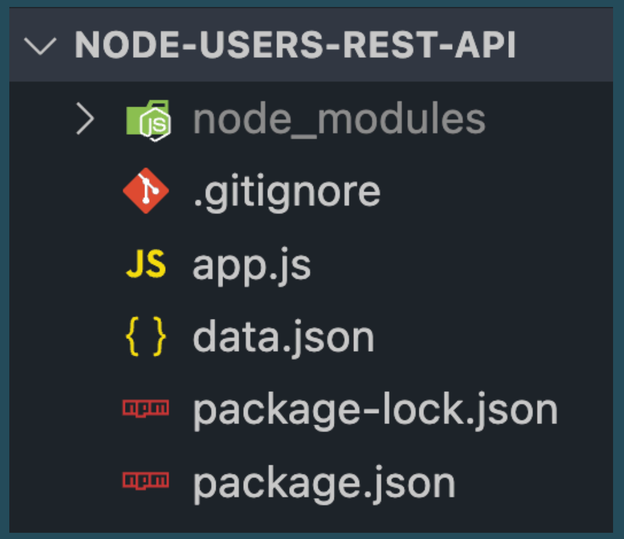
https://levelup.gitconnected.com/nodejs-runtime-environment-libuv-library-event-loop-thread-pool-5f5ecadc0318

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)

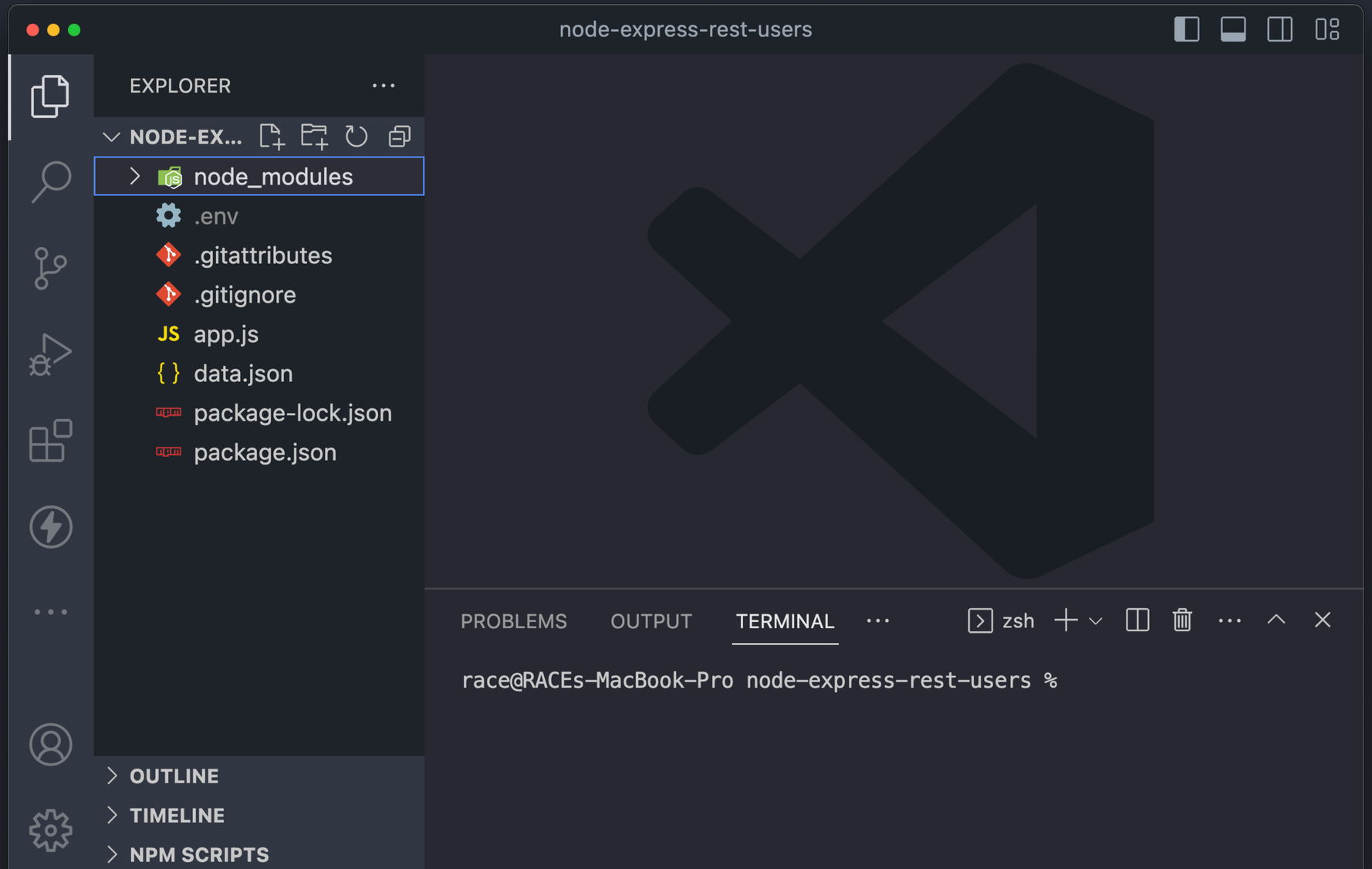
- Node.js packages contains all the files you need for a module.
- Modules are JavaScript libraries you can include in your project.

My Project

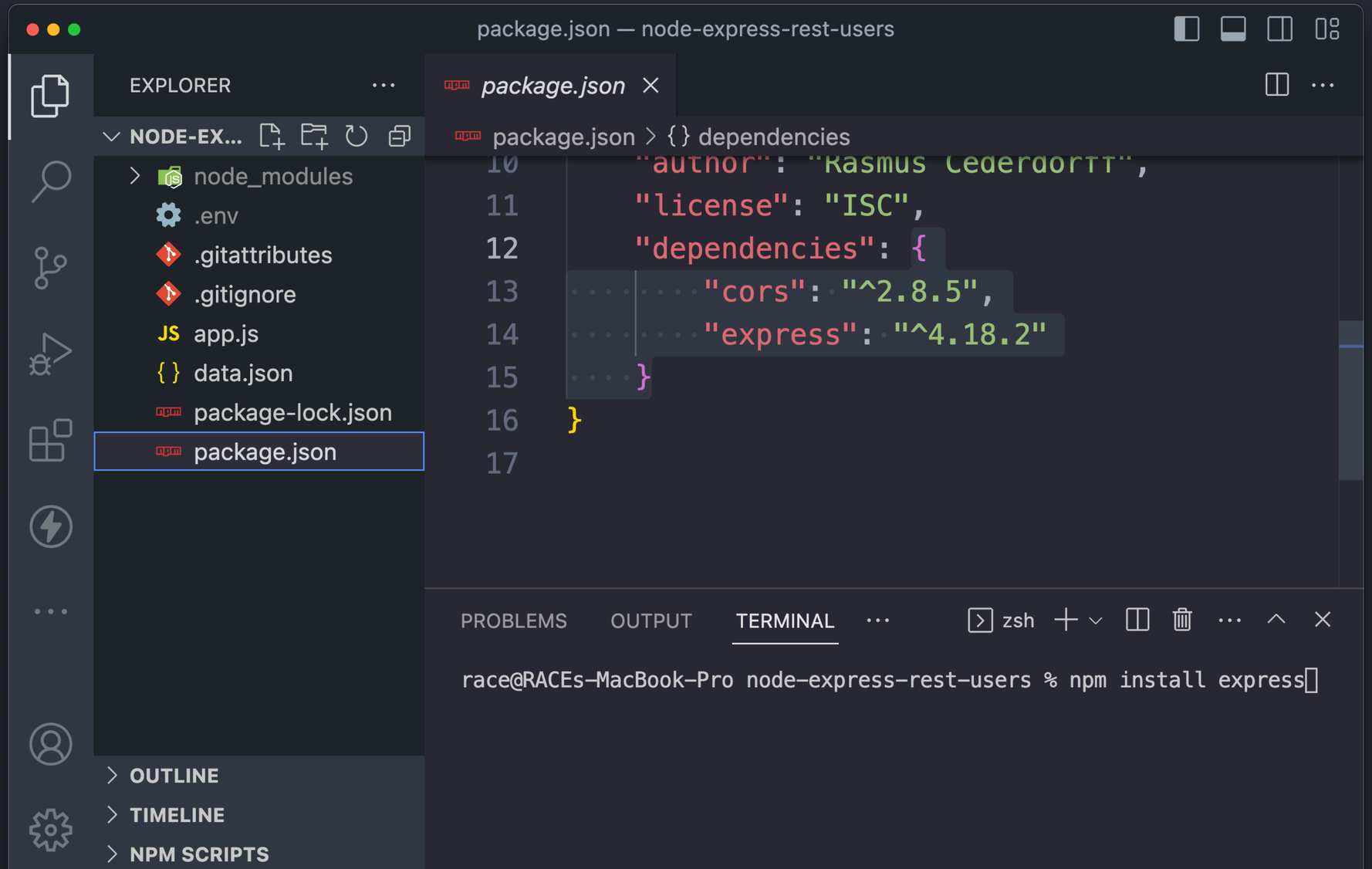

NODE-USERS-REST-API
node_modules
.gitignore
app.js
data.json
package-lock.json
package.json


npm
Express
JS
mysql2
CORS
Remix
Mongoose
And many other
JavaScript libraries

node_modules



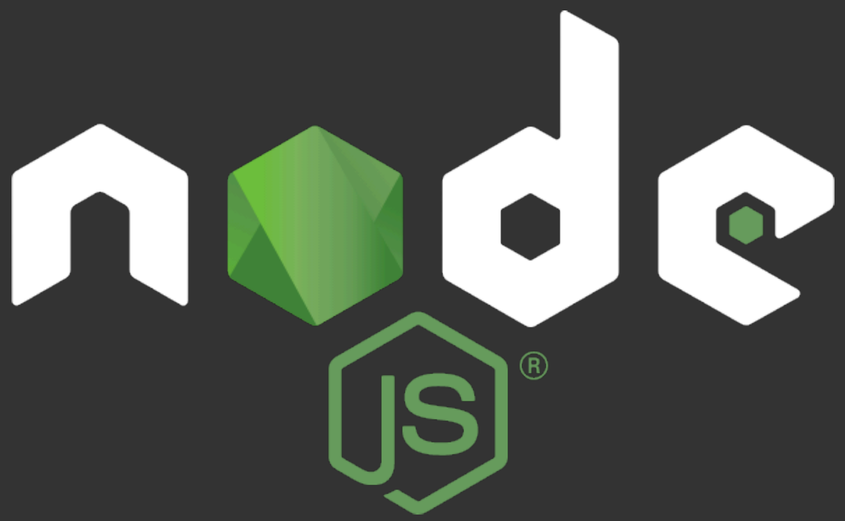
npm install {package-name}



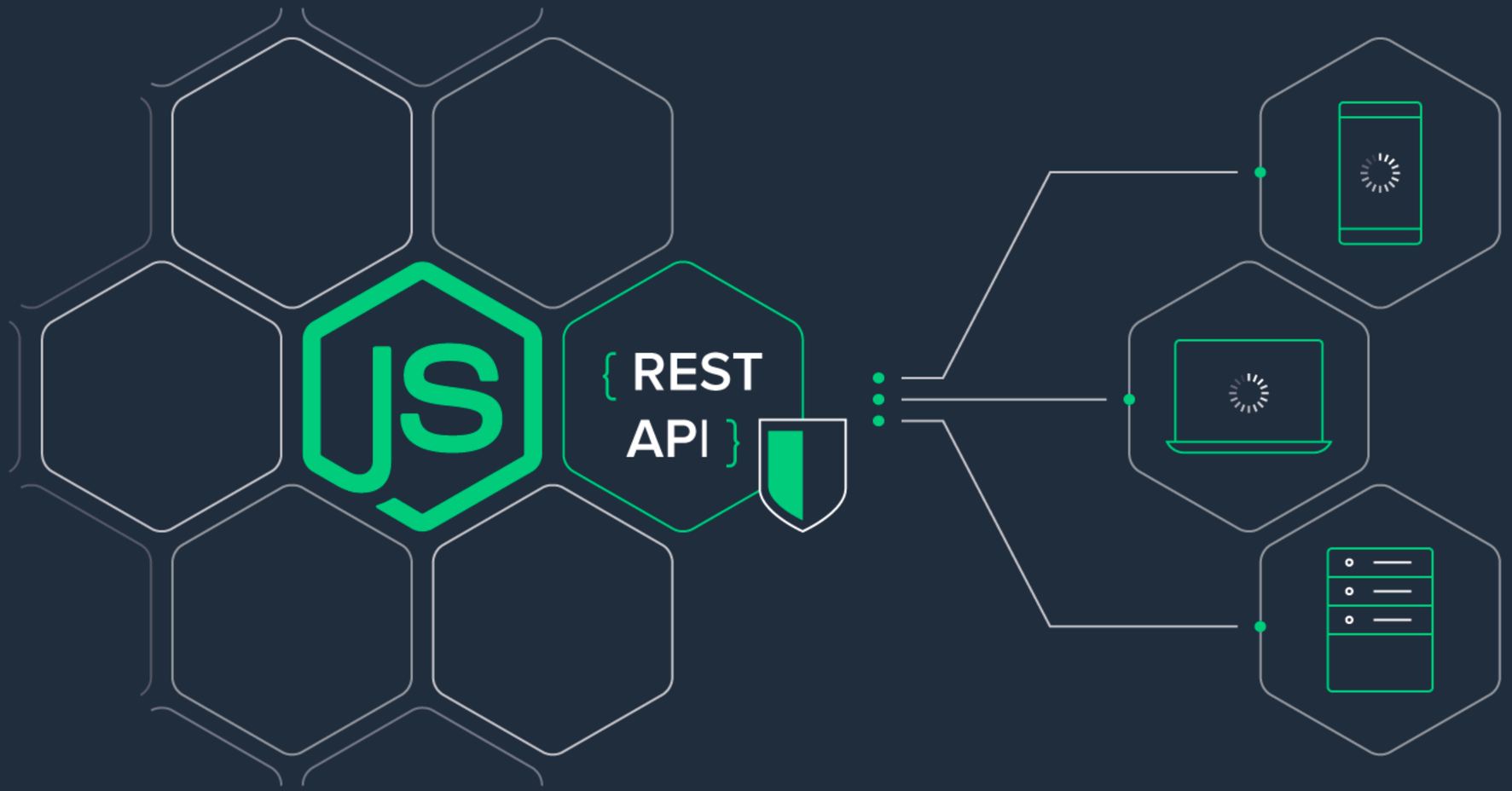
npm consists of

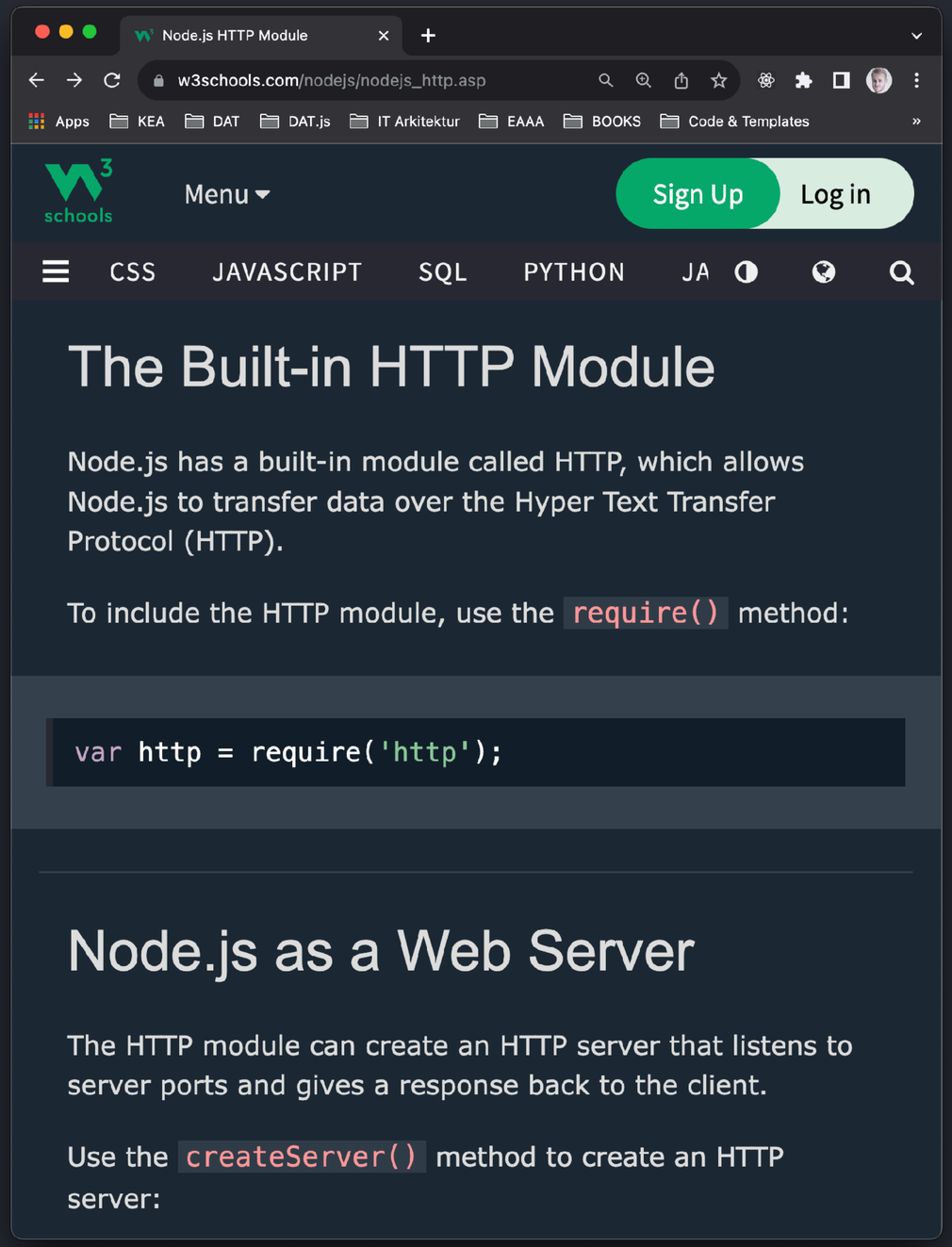
1. Website to discover packages
2. Command line interface (CLI)
3. A registry - database with JS software/ libraries (node modules)

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



Node.js HTTP Module


JS
{ REST
API }

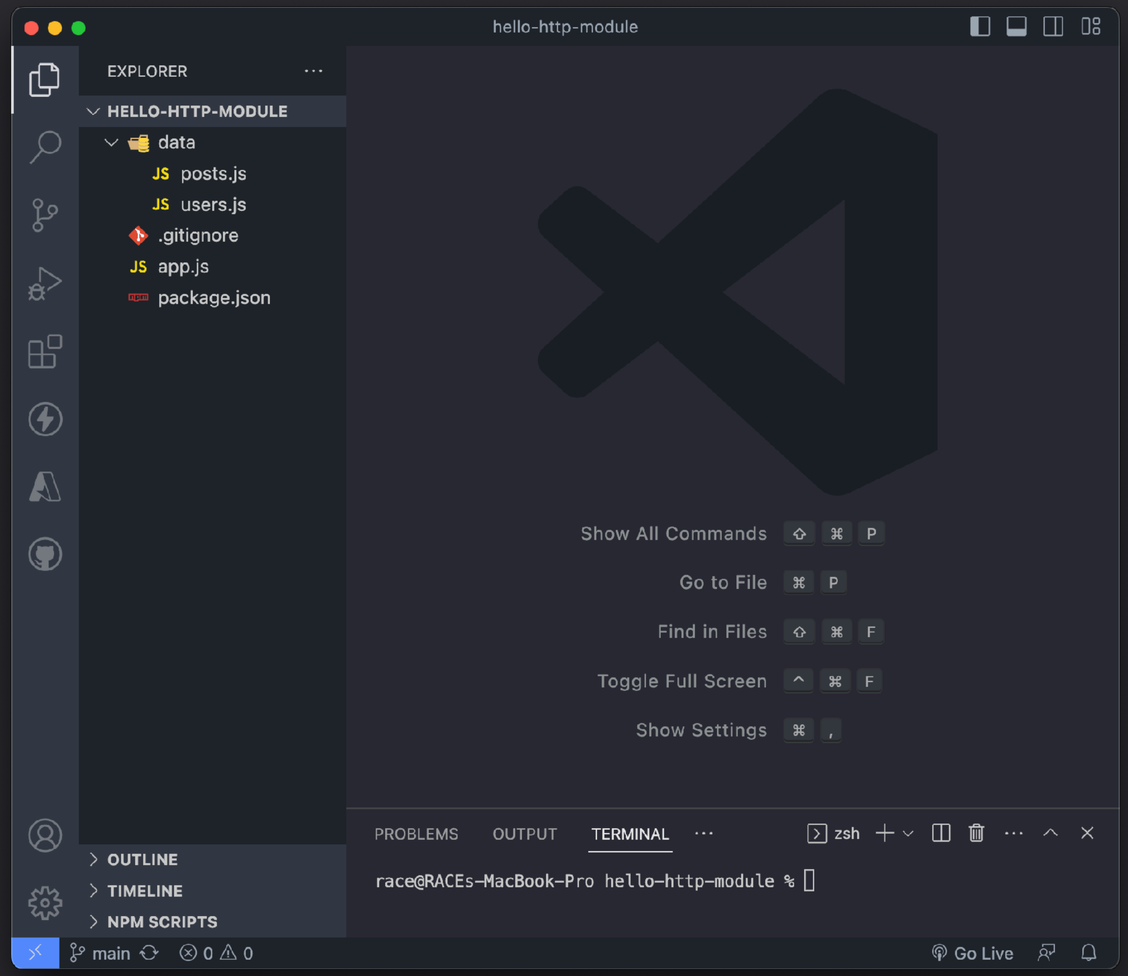


https://www.w3schools.com/nodejs/nodejs_http.asp

```
// 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

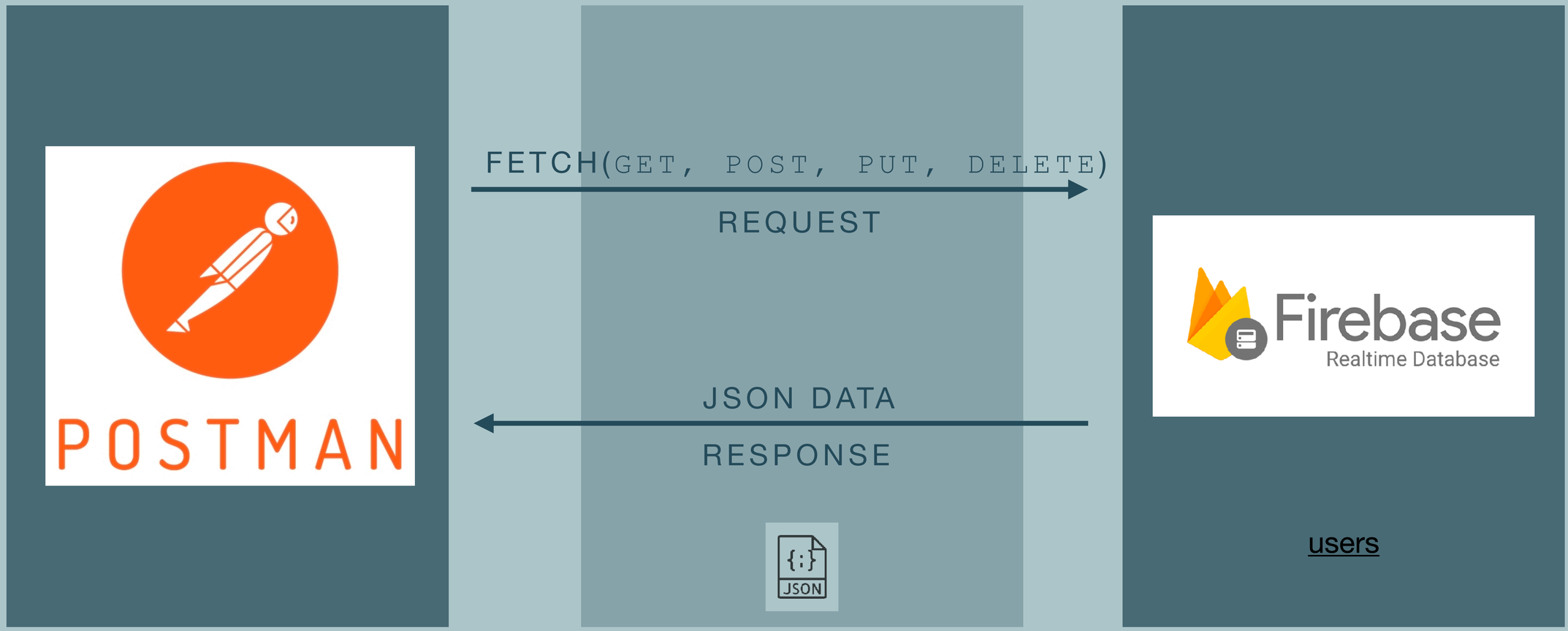
- Postman is a software tool for working with APIs.
- We can use it as client to test our backend API’s and to test APIs in general.
- It helps developers send requests and receive responses from APIs.
- Postman tests API functionality and documents how to use APIs.
- It simplifies API development and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

Web Development

POSTMAN

FIREBASE REST API

DATA SOURCE



CLIENT

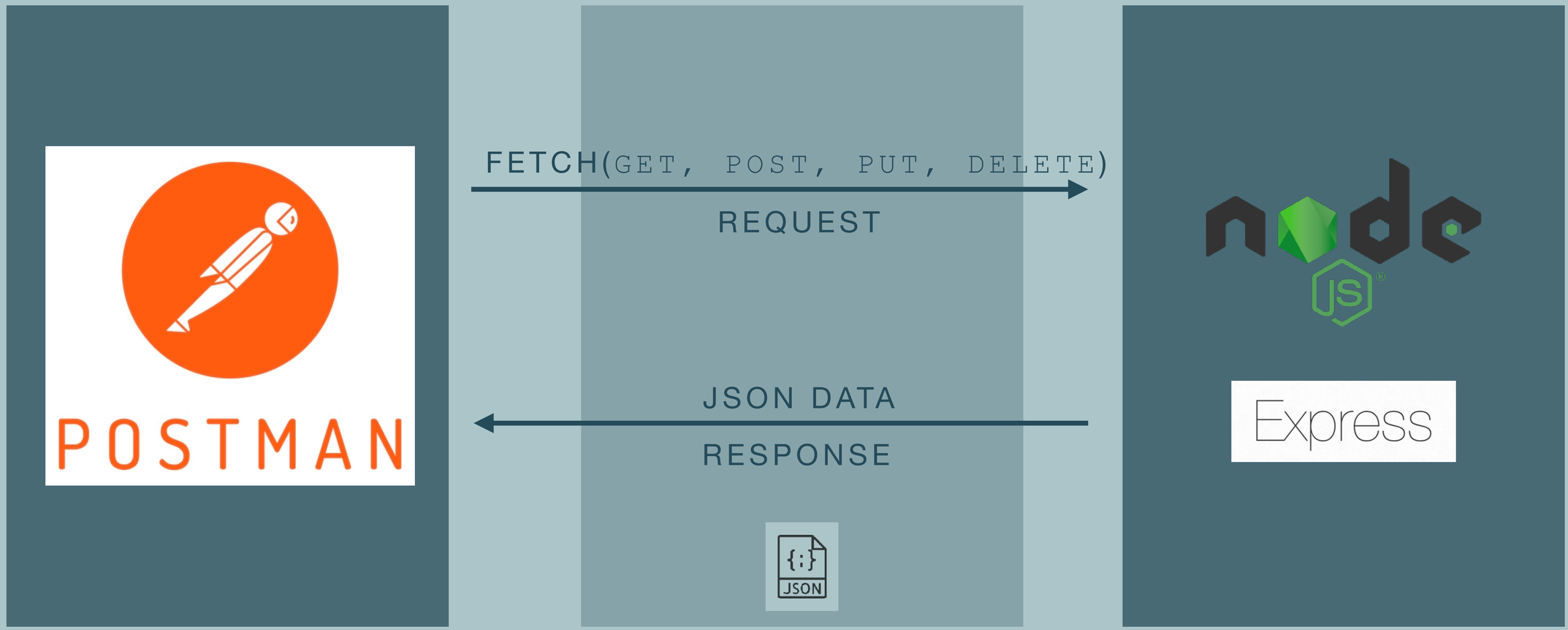
SERVER

Web Development

POSTMAN

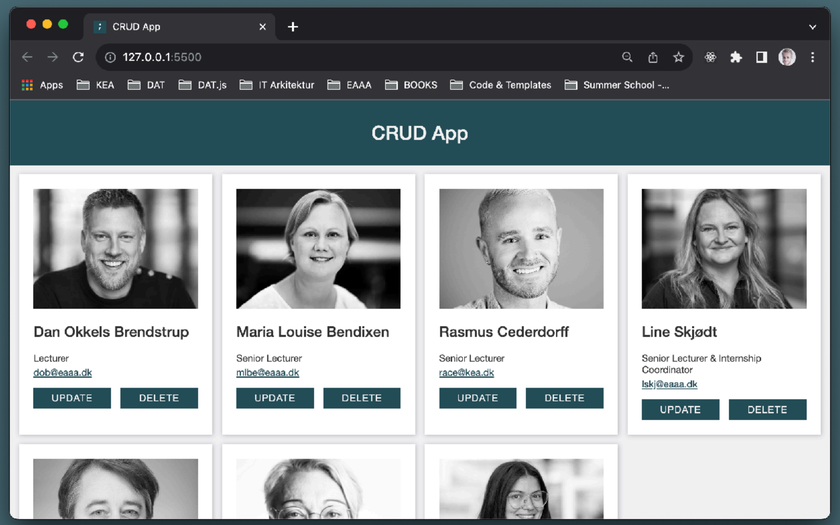
FIREBASE REST API

DATA SOURCE

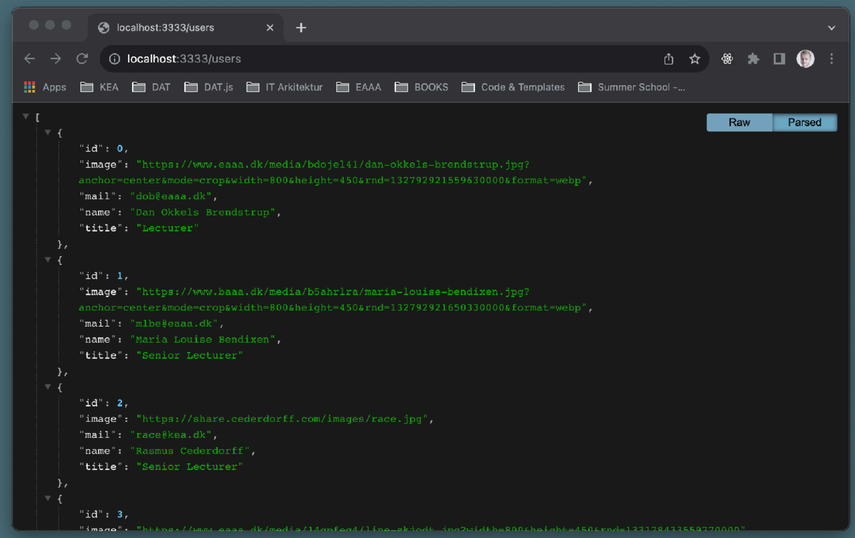


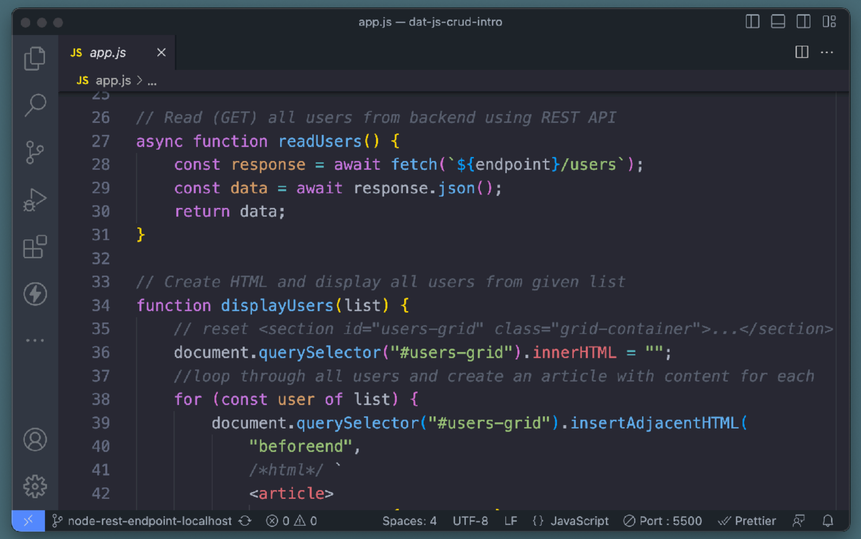
CLIENT

SERVER

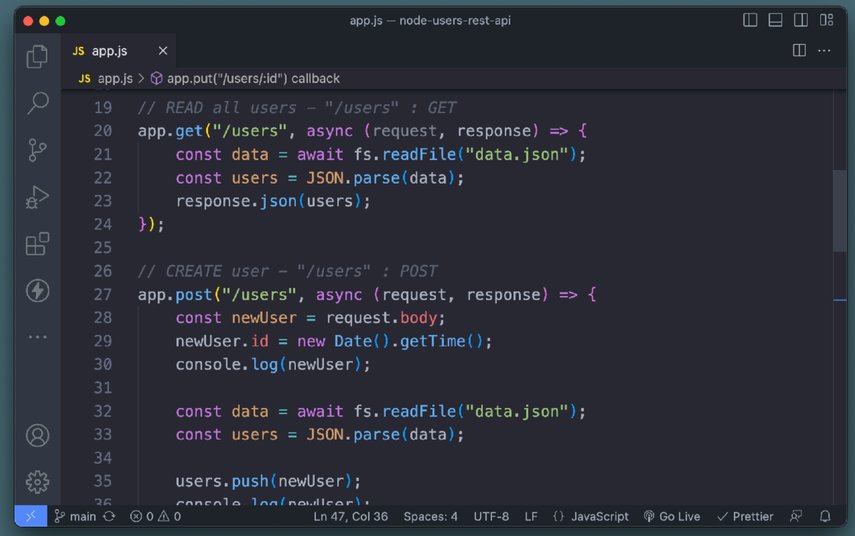


REQUEST



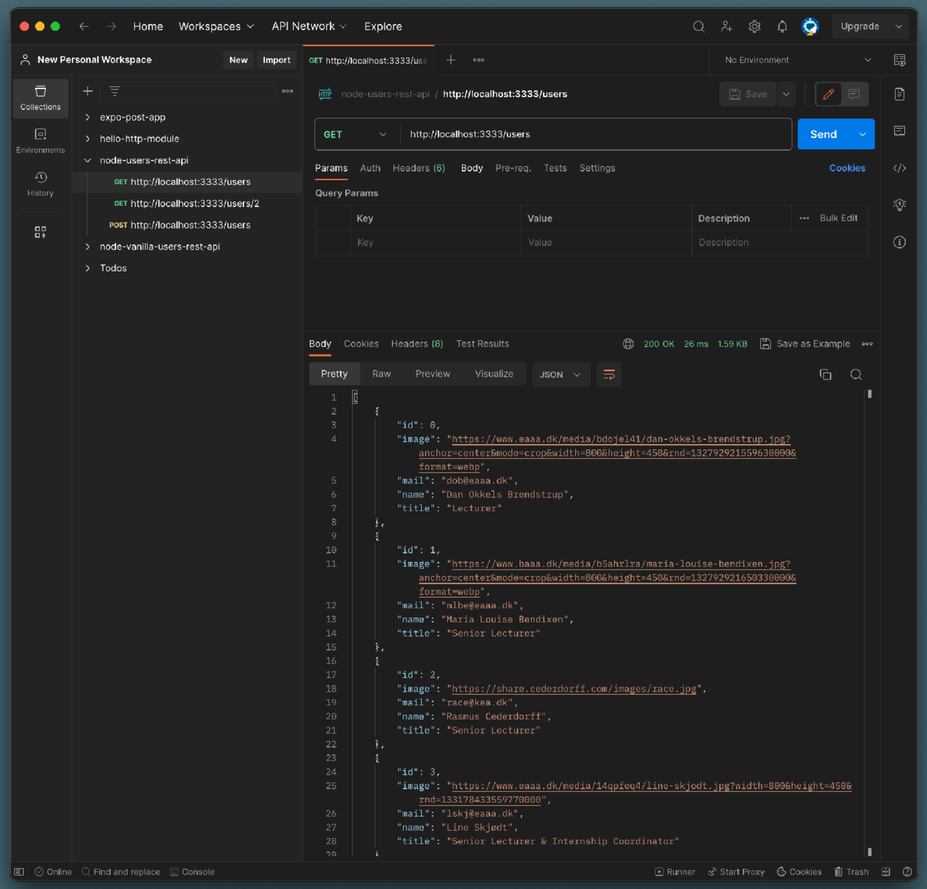


RESPONSE



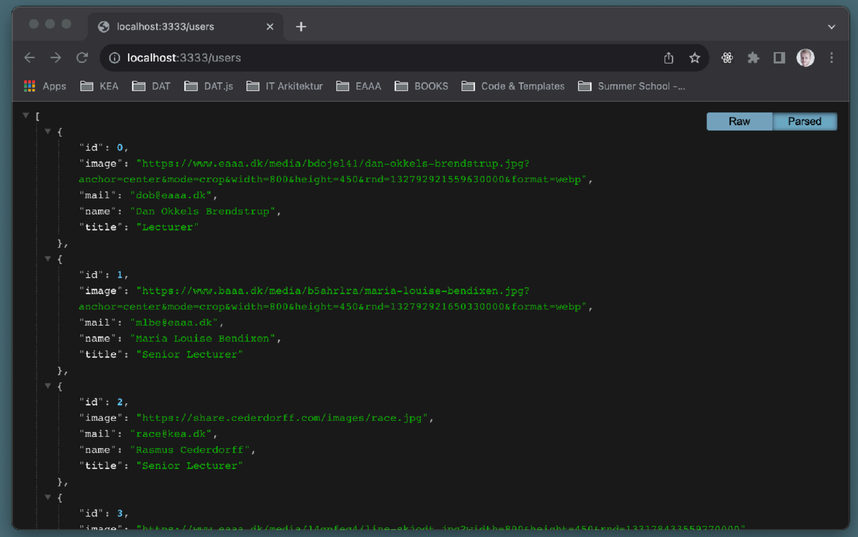
Frontend

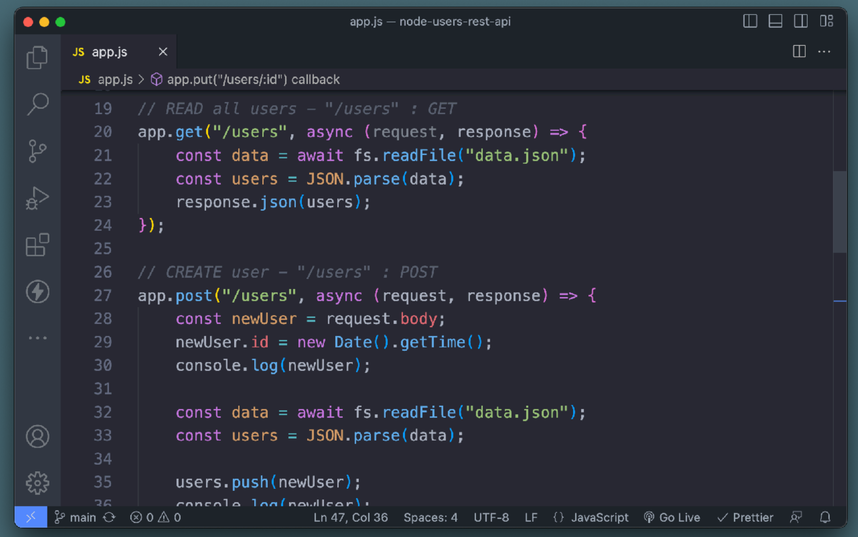
Backend



REQUEST

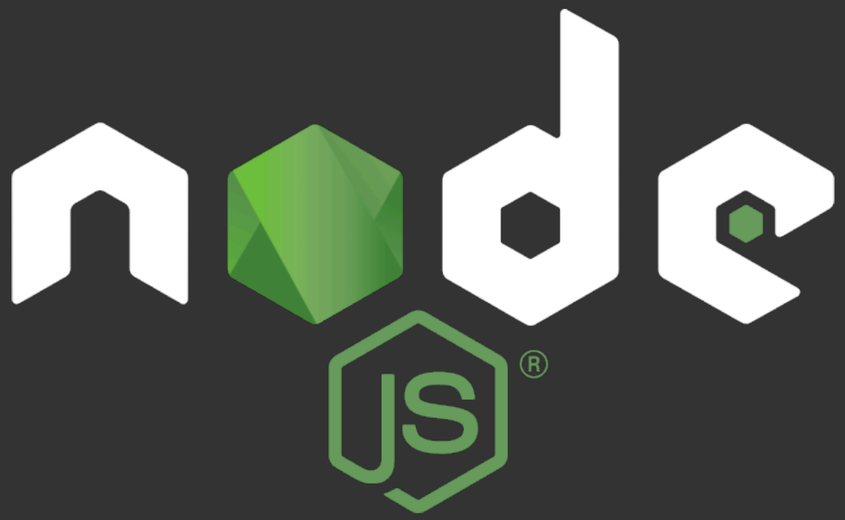
RESPONSE





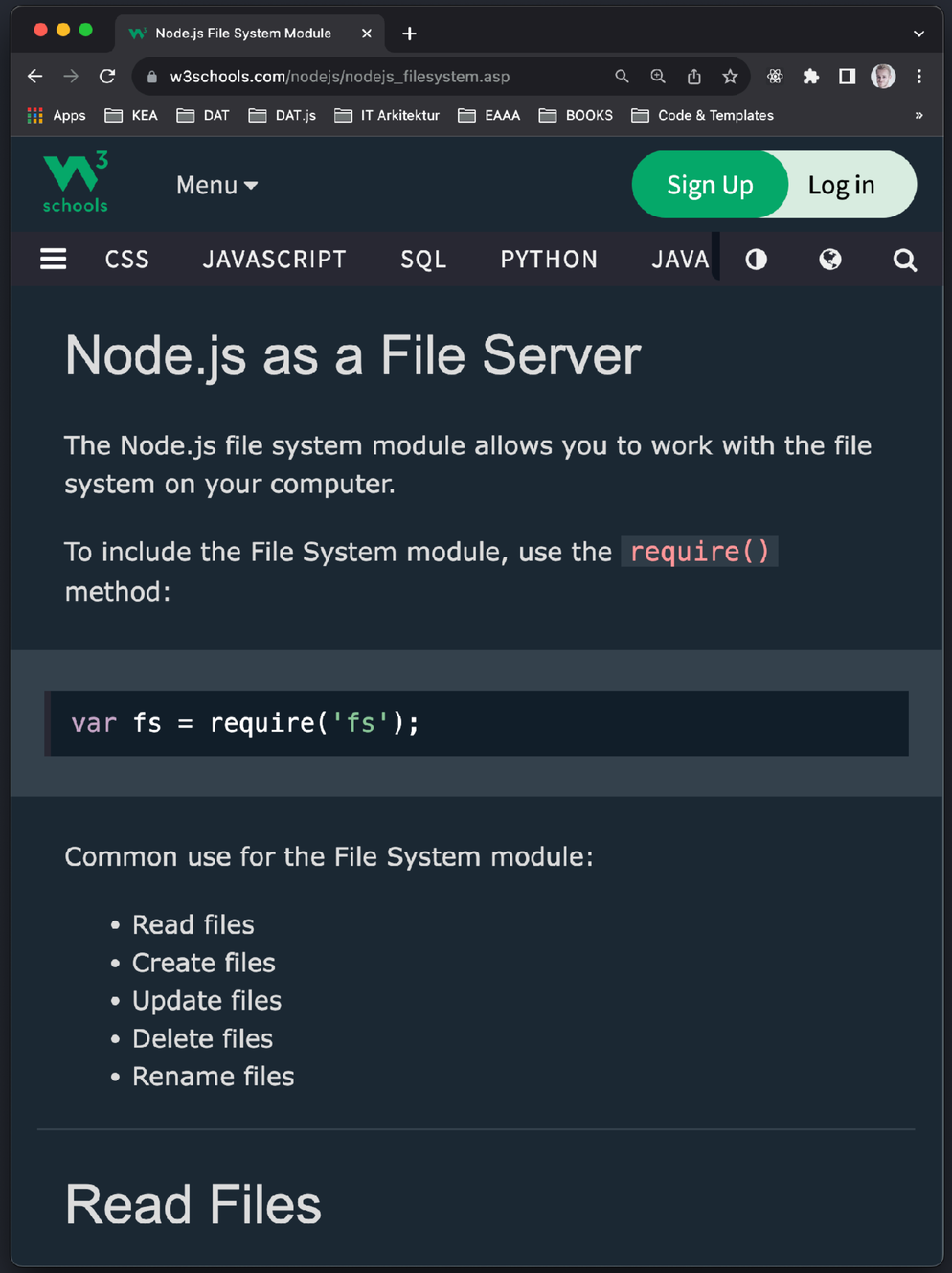
Client

Backend



Node.js File System

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

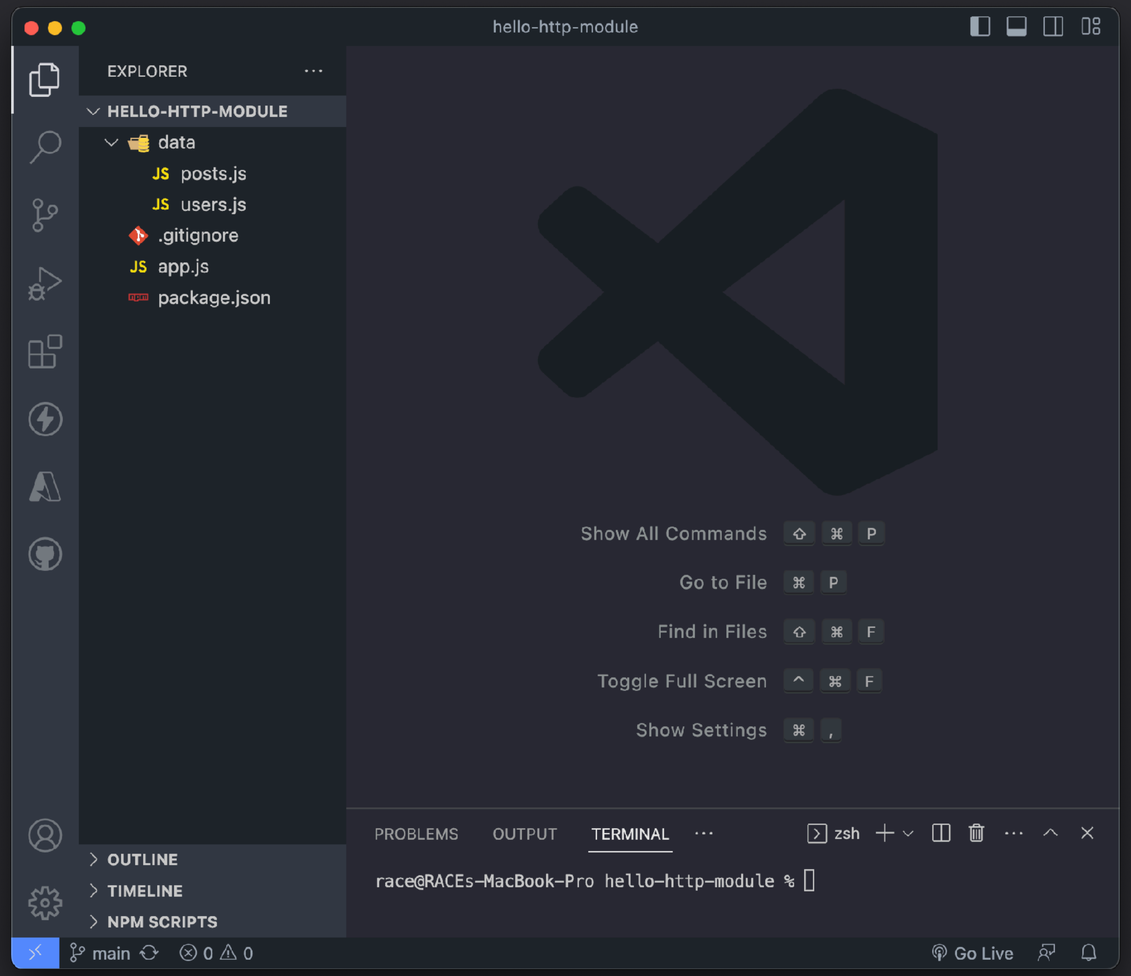


https://www.w3schools.com/nodejs/nodejs_filesystem.asp

```
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);
```

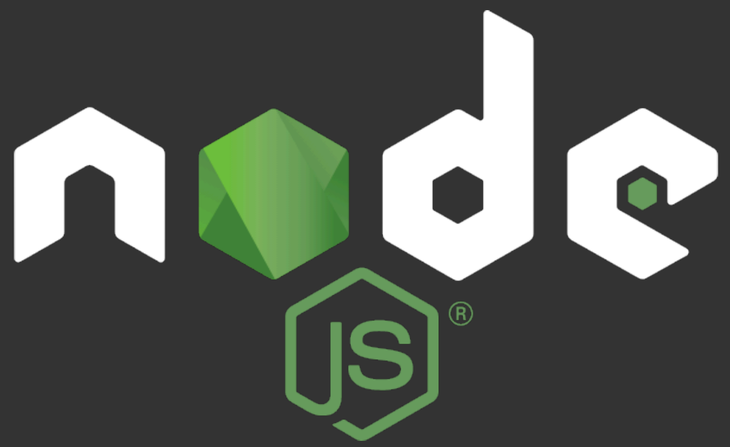
Øvelse

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



With your own words

- What’s Node.js?
- Important features of Node.js
- What did we build?
- What improvements can be made?
- How can we simplify the implementation of the different concepts?

Introduction to

Express

- A popular, minimalist web application framework for building APIs and web applications using Node.js.
- It simplifies the process of building web applications and APIs by providing a set of tools and utilities.

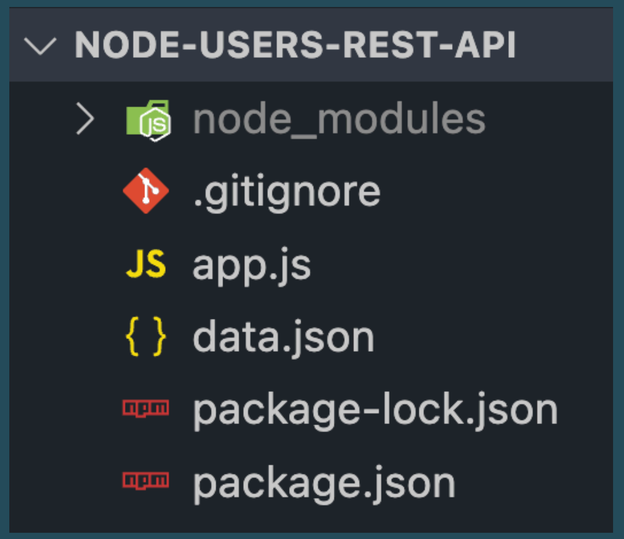
Express

A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications

Express

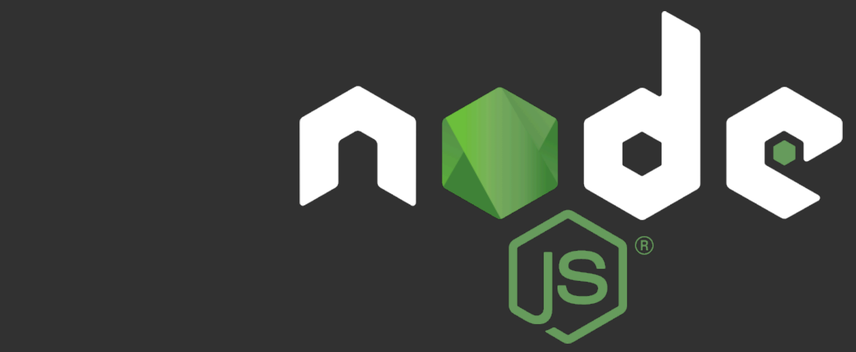
Express.js is known for its flexibility and ease of use, making it an ideal choice for developers who want to build web applications quickly and efficiently. With its built-in middleware functions, routing system, and template engines, Express.js simplifies the process of building complex web applications.

My Project


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

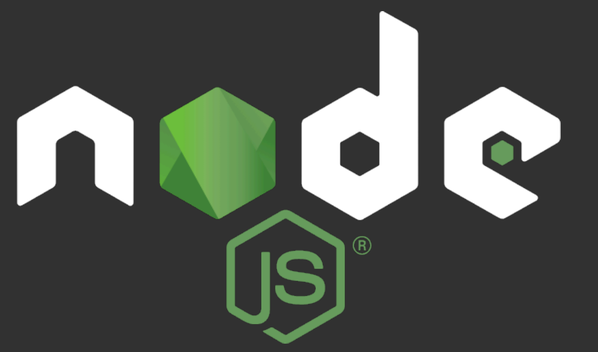

npm
Express
JS
mysql2
CORS
Remix
Mongoose
And many other
JavaScript libraries

Why Express.js?



HTTP Module

—>



+

express

Simple GET Request

```
const app = http.createServer(async (request, response) => {
    // READ all users - "/users" : GET
    if (request.url === "/users" && request.method === "GET") {
        const data = await fs.readFile("data.json");
        response.writeHead(200, { "Content-Type": "application/json" });
        response.end(data);
    }
```

```
// READ all users - "/users" : GET
app.get("/users", async (request, response) => {
    const data = await fs.readFile("data.json");
    const users = JSON.parse(data);
    response.json(users);
});
```

“Simple” POST Request

```
const app = http.createSer (async (request  esponse) => {
    // "/users" : POST
    if (request.url === "/users" && request.method === "POST") {
        // get the data sent along
        const newUser = JSON.parse(await getReqData(request));
        // user object with generated dummy id and parsed userData
        const user = {
            id: new Date().getTime(),
            ...newUser
        };
        const data = await fs.readFile("data.json");
        const users = JSON.parse(data);
        users.push(user);
        const usersJSON = JSON.stringify(users);
        await fs.writeFile("data.json", usersJSON);

        // set the status code and content-type
        response.writeHead(200, { "Content-Type": "application/json" });
        //send the user
        response.end(usersJSON);
    }
});

async function getReqData(req) {
    return new Promise((resolve, reject) => {
        let body = "";
        req.on("data", chunk => (body += chunk));
        req.on("end", () => resolve(body));
        req.on("error", reject);
    });
}
```

```
// CREATE user - "/users" : POST
app.post("/users", async (request, response) => {
    const newUser = request.body;
    newUser.id = new Date().getTime();

    const data = await fs.readFile("data.json");
    const users = JSON.parse(data);

    users.push(newUser);
    fs.writeFile("data.json", JSON.stringify(users));
    response.json(users);
});
```

- **Single-page applications (SPAs)**
  Express.js creates SPAs that load all resources on a single page, ensuring a seamless user experience.
- **Mobile applications**
  Express.js builds mobile apps using web technologies like HTML, CSS, and JavaScript.
- **RESTful APIs**
  Express.js crafts RESTful APIs for data manipulation and access by other applications.
- **Server-side rendering**
  Express.js enables server-side rendering, rendering pages on the server before sending to the client.
- **Real-time applications**
  Express.js constructs real-time apps, facilitating ongoing client-server communication.
- **Microservices**
  Express.js constructs microservices, small independent units combined to create larger applications.

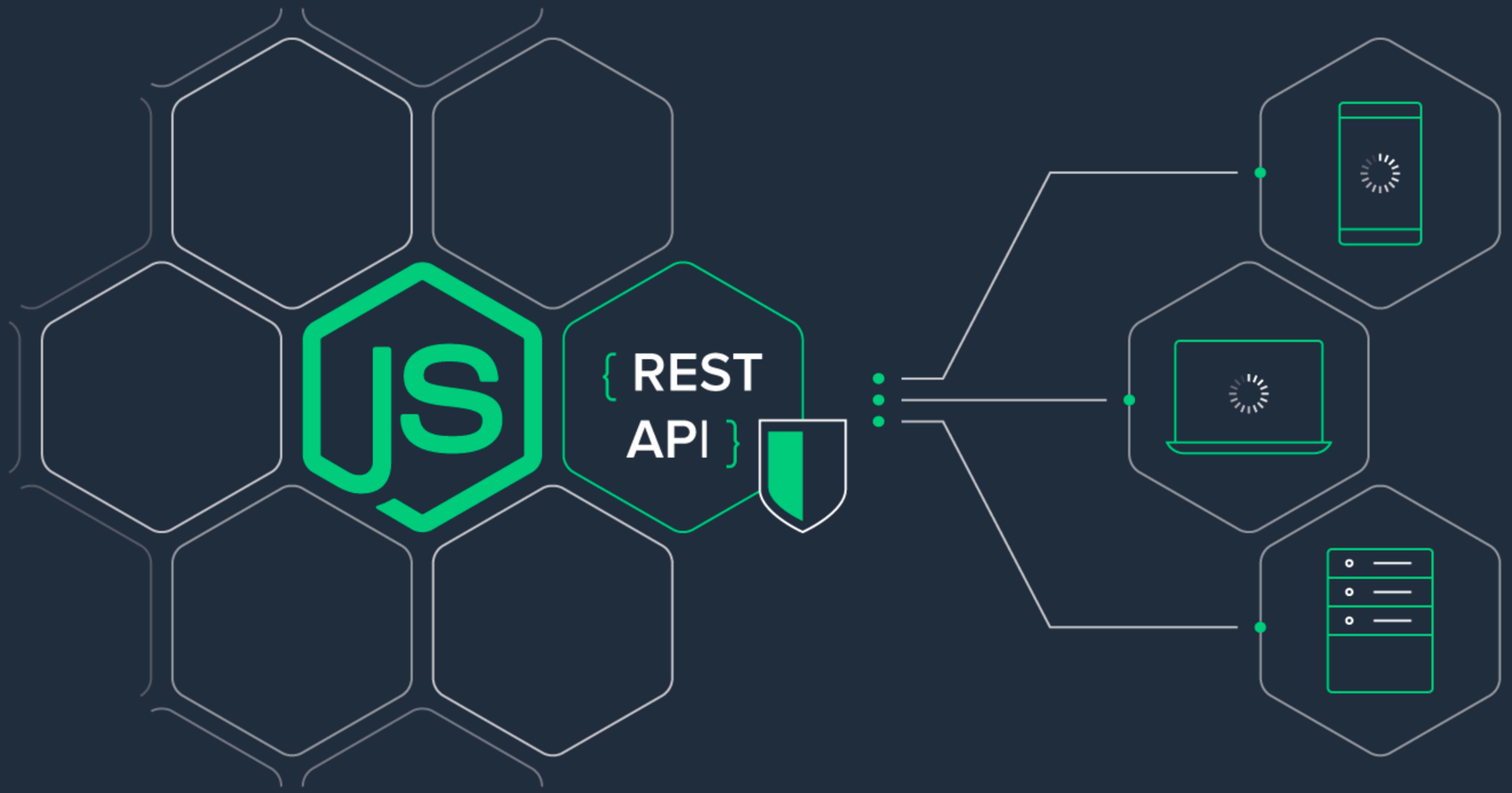
What is it used for?

- **Minimalistic Framework**
  Express is a lightweight and minimal web application framework for Node.js.
- **Routing**
  It offers a powerful routing system to handle different HTTP methods and URL patterns.
- **Middleware**
  Express uses middleware to handle various tasks, such as authentication, logging, and data parsing.
- **HTTP Utility Methods**
  Provides easy-to-use methods for interacting with the HTTP protocol.
- **Extensibility**
  Allows integration of third-party middleware and extensions for added functionality.
- **Static File Serving**
  Simplifies serving static files like CSS, JavaScript, and images.
- **Error Handling**
  Offers error handling mechanisms for a better user experience.
- **Flexible for APIs**
  Suitable for building RESTful APIs with its routing and middleware capabilities.
- **Easy to Use**
  Express.js is easy to use and requires minimal configuration, making it ideal for developers who want to build web applications quickly.

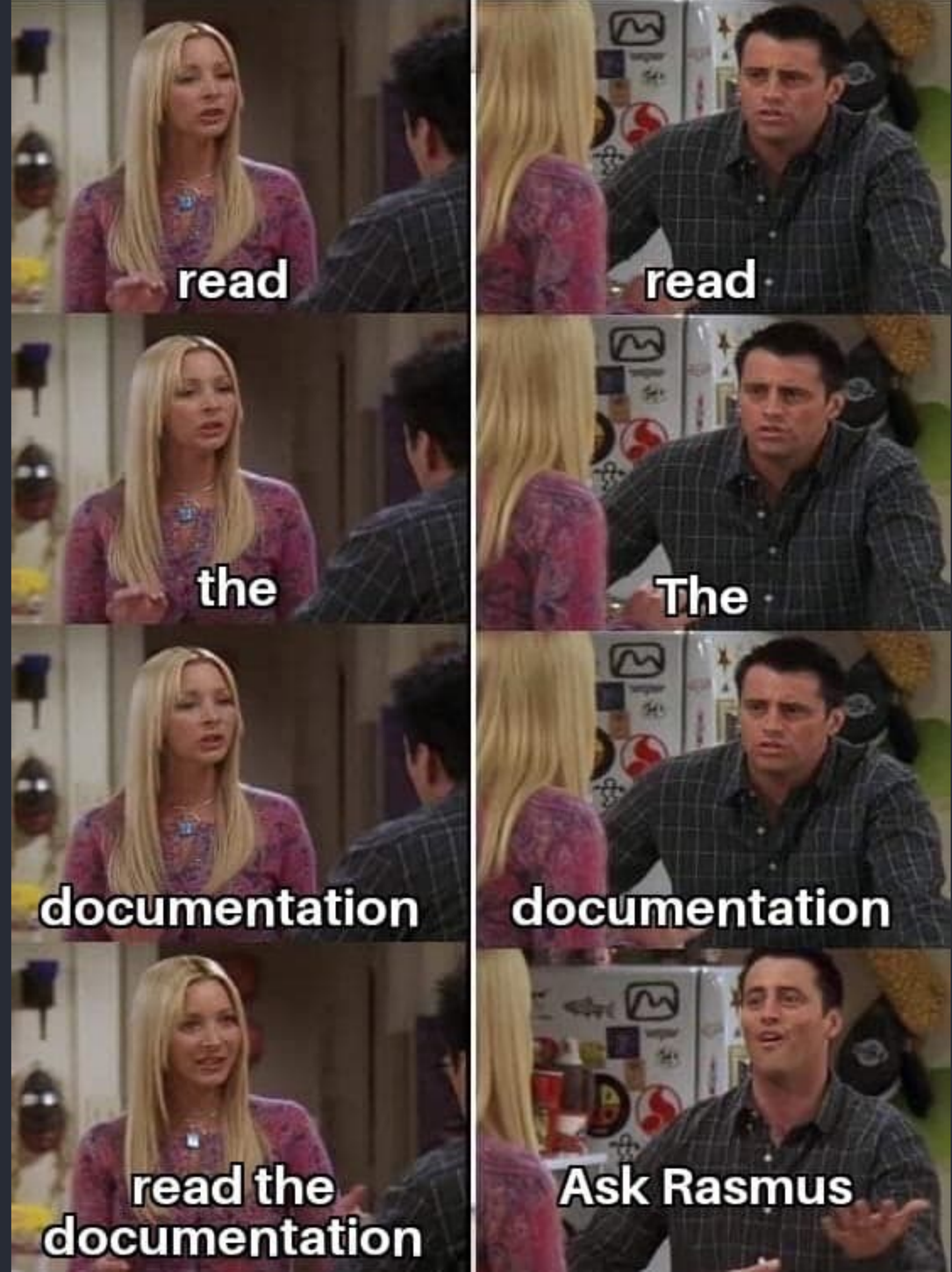
Express JS

Key Features

Streamlines and simplifies the implementation of REST API



But RACE, how do you know all that stuff?



Routing

What’s a Router?

What’s a Router?

“It is the piece of software in charge to organize the states of the application, switching between different views.”

Routes organize how your application handles different requests, making your code structured and maintainable.

https://medium.com/@fro_g/routing-in-javascript-d552ff4d2921

Server

Client

Incoming request URL

Router

GET “/posts”

POST “/posts”

PUT “/posts”

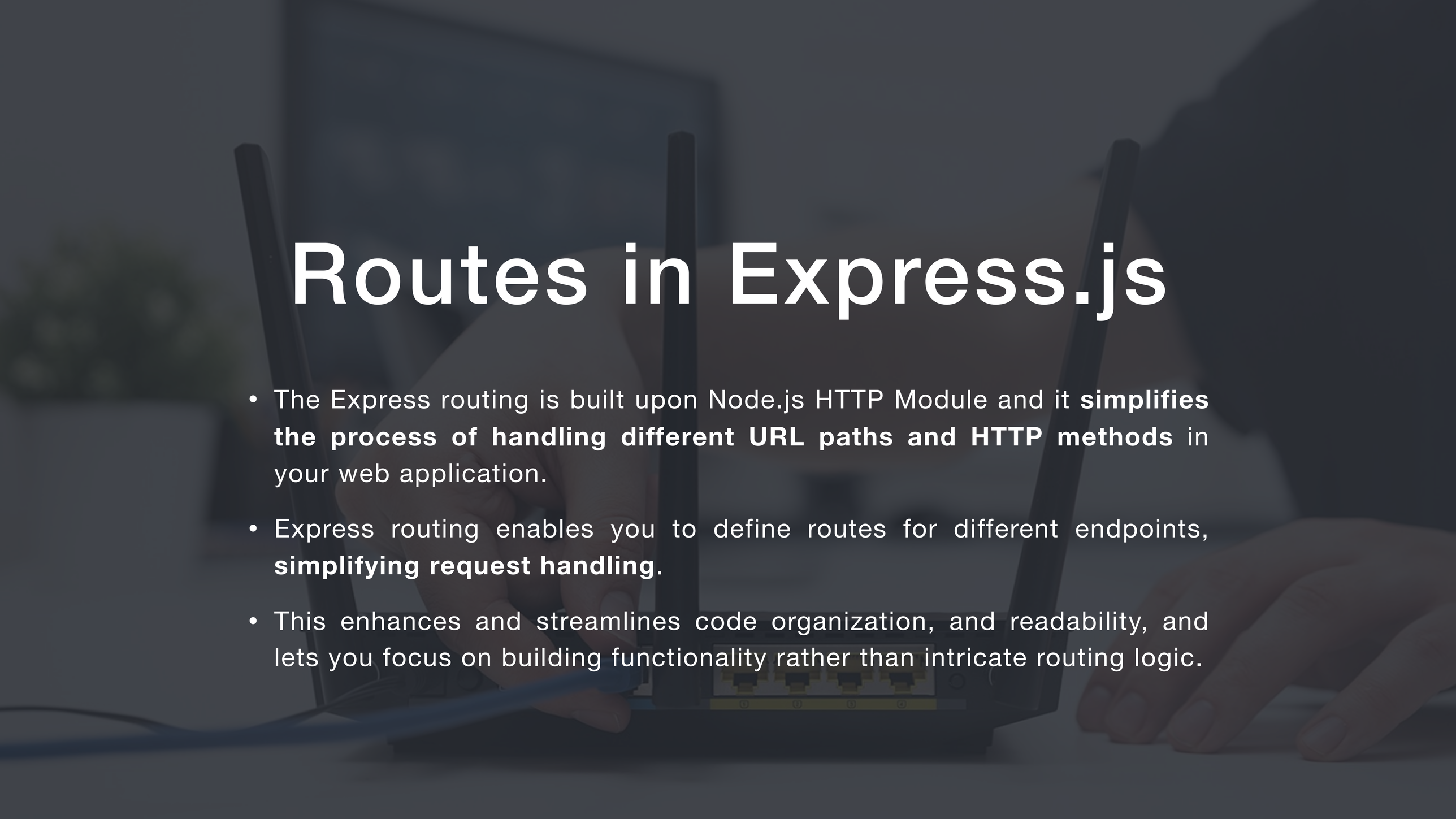
DELETE “/posts”

Based on the request URL and method, the router decides what to do

```
app.get("/posts", (req, res) => {
    // return all posts
});

app.post("/posts", (req, res) => {
    const post = req.body;
    // Create a new post based on th
});

app.put("/posts/:id", (req, res) =>
    const postId = req.params.id;
    const post = req.body;
    // Find and update the post with
});

app.delete("/posts/:id", (req, res)
    const postId = req.params.id;
    // Find and delete the post with
});
```

Routes in Express.js

- Routes are a fundamental concept that helps you define **how your web application responds to different types of HTTP requests** at different URLs.
- Think of routes as the paths your application can take based on the URLs that users request.



Routes in Express.js

- The Express routing is built upon Node.js HTTP Module and it **simplifies the process of handling different URL paths and HTTP methods** in your web application.
- Express routing enables you to define routes for different endpoints, **simplifying request handling**.
- This enhances and streamlines code organization, and readability, and lets you focus on building functionality rather than intricate routing logic.

Routes in Express.js

- Routes determine how your application responds to specific URLs and HTTP methods.
- Each route combines an HTTP method (like GET, POST) with a URL path.
- When a client request matches a route, a corresponding route handler function is executed to process the request and send a response.

```
import express from "express";
const app = express();

// Define routes using app.METHOD(PATH, HANDLER)
app.get("/posts", (req, res) => {
    // return all posts
});

app.post("/posts", (req, res) => {
    const post = req.body;
    // Create a new post based on the data in req.body
});

app.put("/posts/:id", (req, res) => {
    const postId = req.params.id;
    const post = req.body;
    // Find and update the post with postId and data from req.body
});

app.delete("/posts/:id", (req, res) => {
    const postId = req.params.id;
    // Find and delete the post with postId
});

// ...and so on

app.listen(3000, () => {
    console.log("Server is running on port 3000");
});
```

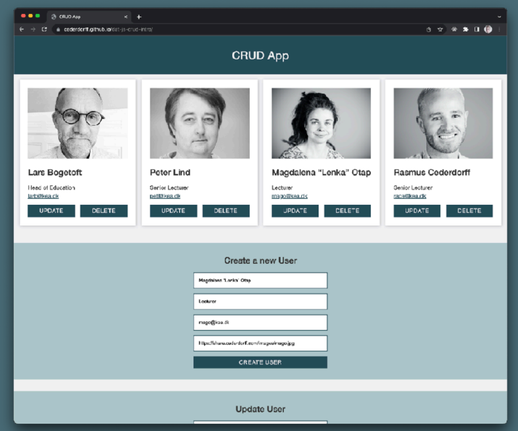
Middleware, JSON Request and Response

Web Development

WEB APP

REST API

DATA SOURCE



FETCH(GET, POST, PUT, DELETE)

REQUEST {JSON DATA}

JSON

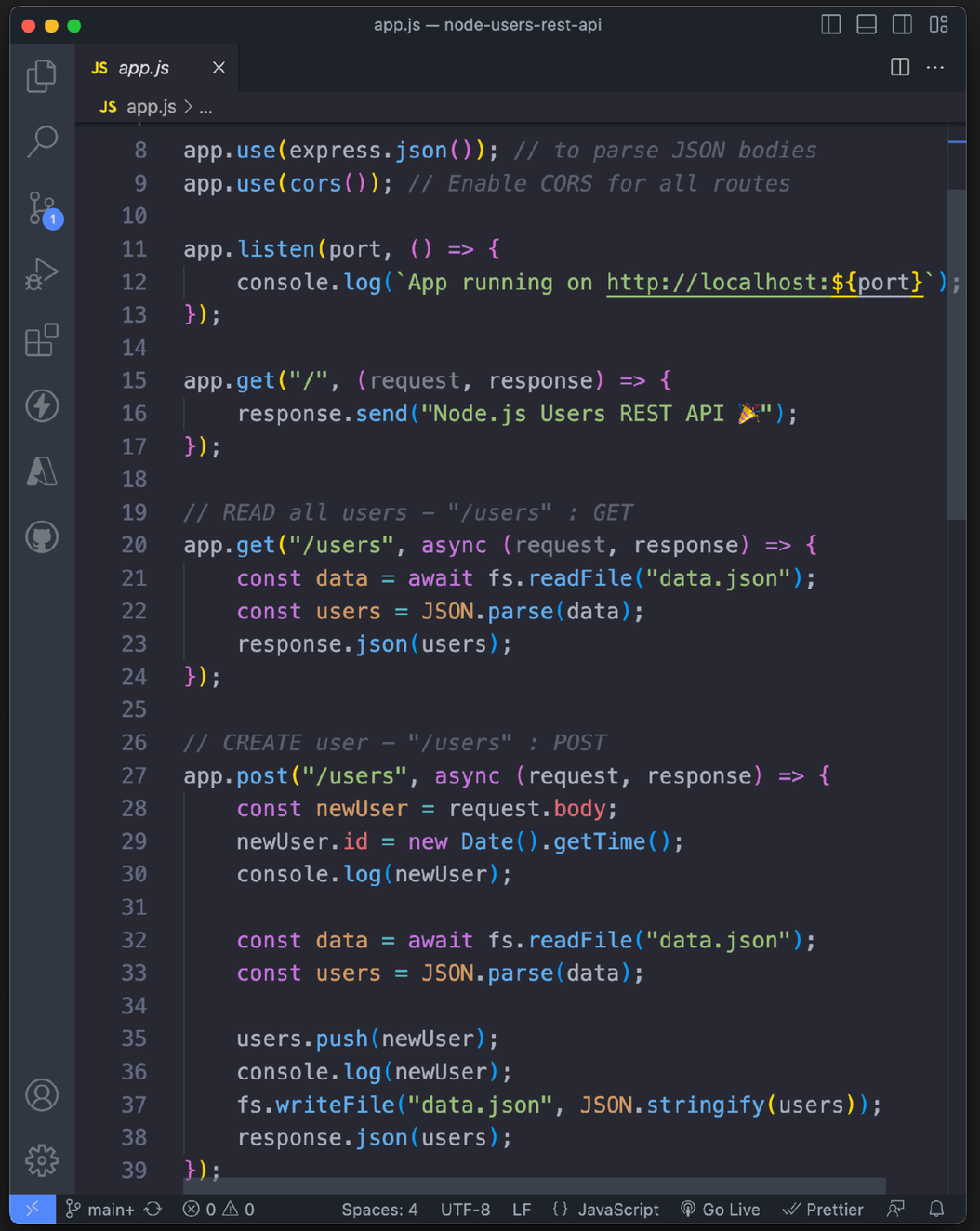
RESPONSE {JSON DATA}

Express

CLIENT

SERVER

- In Express.js, middleware refers to functions that are **executed during the request-response cycle** of a web application. These functions have access to the request and response objects and can modify them, perform tasks, or control the flow of the application.
- Express.js allows you to add middleware at the application level using `app.use()` or at specific route levels. Middleware functions are executed in the order they are added, so the order matters.
- For instance, using `app.use(express.json())` is a middleware that automatically parses incoming JSON data from requests, making it accessible as JavaScript objects in your route handlers. This simplifies handling JSON data.
- Similarly, `app.use(cors())` is middleware that enables Cross-Origin Resource Sharing (CORS), allowing your server to respond to requests from different domains. This is essential for security when your frontend and backend are on separate domains.
- In both cases, middleware helps simplify and enhance the functionality of your Express application by handling common tasks automatically before your route handlers are executed.

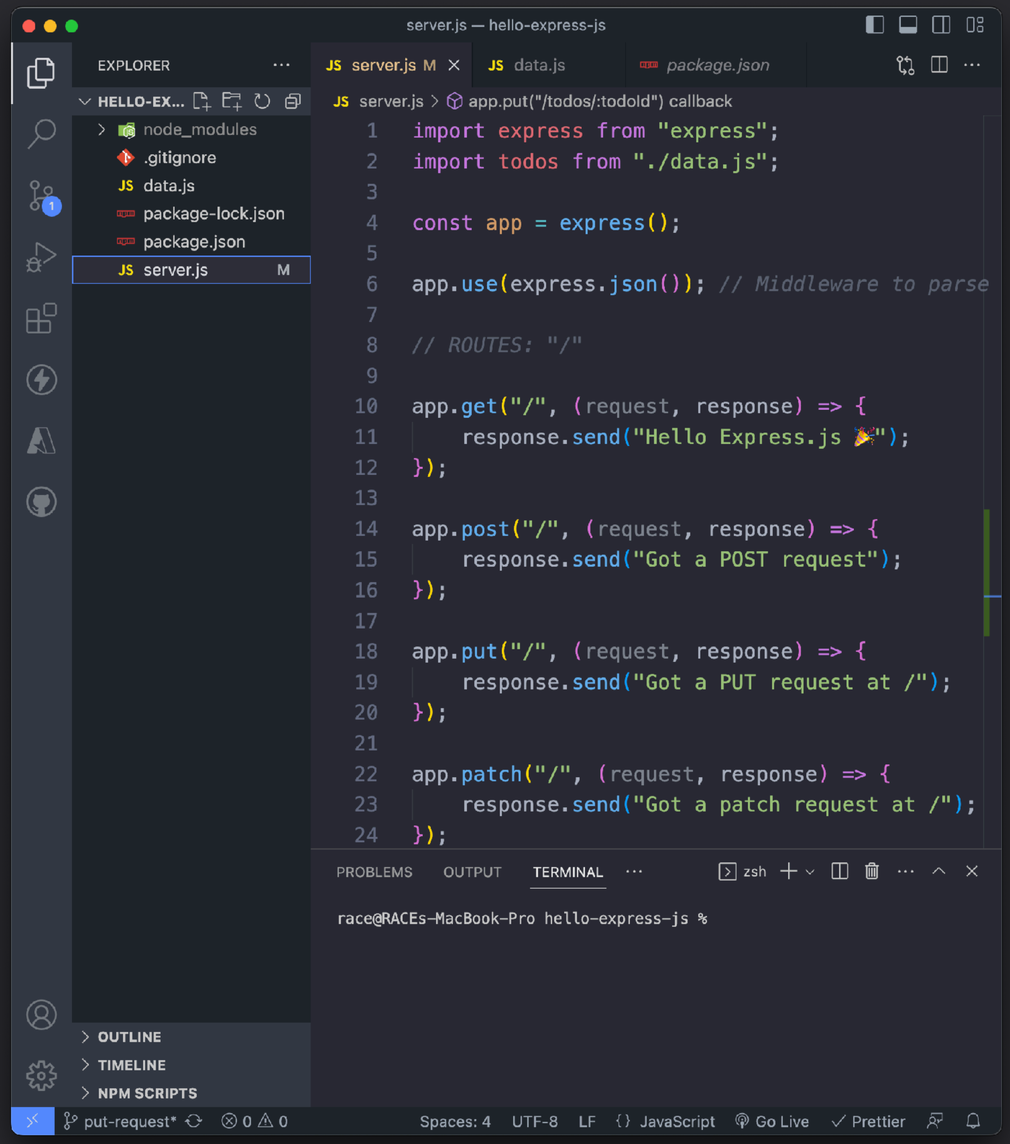


100 SECONDS OF

CROSS-ORIGIN RESOURCE SHARING

Exercises

- Getting Started with Express.js
- Node.js og MySQL



Code Every Day