Documentation of building a web browser IDE

Before building an IDE, it best to understand that the web browser already know how to read and run Html, CSS and JavaScript. It injects and executes them safely inside the browser.

The core idea of this IDE is:

1. *Collect code as text from editors (HTML, CSS, JS)*
2. *Combine the code*
3. *Run it inside an isolated environment*
4. *Display the result in an output panel*

For better understanding on how the computer reads the code, ach editor is just a text editor **component**:

- HTML editor → stores HTML as a string
- CSS editor → stores CSS as a string
- JavaScript editor → stores JS as a string

Example (conceptually):

```
const htmlCode = "<h1>Hello</h1>";

const cssCode = "h1 { color: red; }";

const jsCode = "console.log('Hello IDE')";
```

As for running the code, the safest approach is to use an iframe, because it isolates user code from the main app, preventing breaking your IDE, and again mimics a real browser environment.

The main app wil look like this with the three components passed inside of it:

```
<!DOCTYPE html>
        <html>
                <head>
                        <style>
                                /* CSS here */
                        </style>
                </head>
                <body>
                        <!-- HTML here -->
                        <script>
                                // JavaScript here
                        </script>
                </body>
        </html>
```

We will then inject it into an iframe using:

- srcDoc (recommended)
- OR document.write() (old way)

Then the browser renders it instantly.

The output component simply:

- Displays the iframe
- Captures console logs and errors (optional advanced feature)

We willl override console.log inside the iframe to send messages back to React using postMessage.

In short, here are our tasks: We will allocate each other sections when working on each task.

Task	Where it runs
Run HTML/CSS/JS	Browser (iframe)
Syntax highlighting	Frontend (React)
Linting	Frontend or Backend
Security checks	Backend
Saving files	Backend (Node.js)

Frontend Libraries are needed in our text editors for:

- Syntax highlighting
- Auto-complete
- Error hints

1. Monaco Editor (Recommended)

- npm install @monaco-editor/react

It supports: HTML, CSS, JavaScript, IntelliSense, Themes

2. ESLint (JavaScript analysis)

For: Syntax errors, Bad practices

- npm install eslint

3. Prettier

For: Formatting

- npm install prettier\

4. HTML & CSS Linters
- npm install htmlhint stylelint

Backend libraries needed to execute the code for main app:

(Note: Backend should **NOT run user JavaScript directly** (security risk))

Instead it should handle:

- Authentication (JWT, sessions)
- Saving recent files
- User projects
- Versioning
- Database access

We will need this packages:

- npm install express jsonwebtoken bcrypt cors

(Note: We will need sequelize for PostgreSQL connection)

4. Summary: How our IDE works internally

Flow:

1. User types code in editors
2. React stores code in state
3. React builds a combined HTML document
4. iframe renders the result
5. Output component displays it
6. Node.js stores files and manages authentication

Summary: How the Web-Based IDE Works

This IDE is a web application built using **React for the frontend** and **Node.js for the backend**. It runs entirely in the browser and allows users to write and test **HTML, CSS, and JavaScript** in real time.

How it works

The browser already knows how to read and execute HTML, CSS, and JavaScript. The IDE takes advantage of this by treating each editor as a **text editor**. The HTML, CSS, and JavaScript code are stored as plain text, then combined into one complete HTML document. This document is executed inside an **iframe**, which isolates the user's code from the main application and keeps the IDE safe and stable. The iframe renders the output instantly, just like a normal browser page.

How it is built

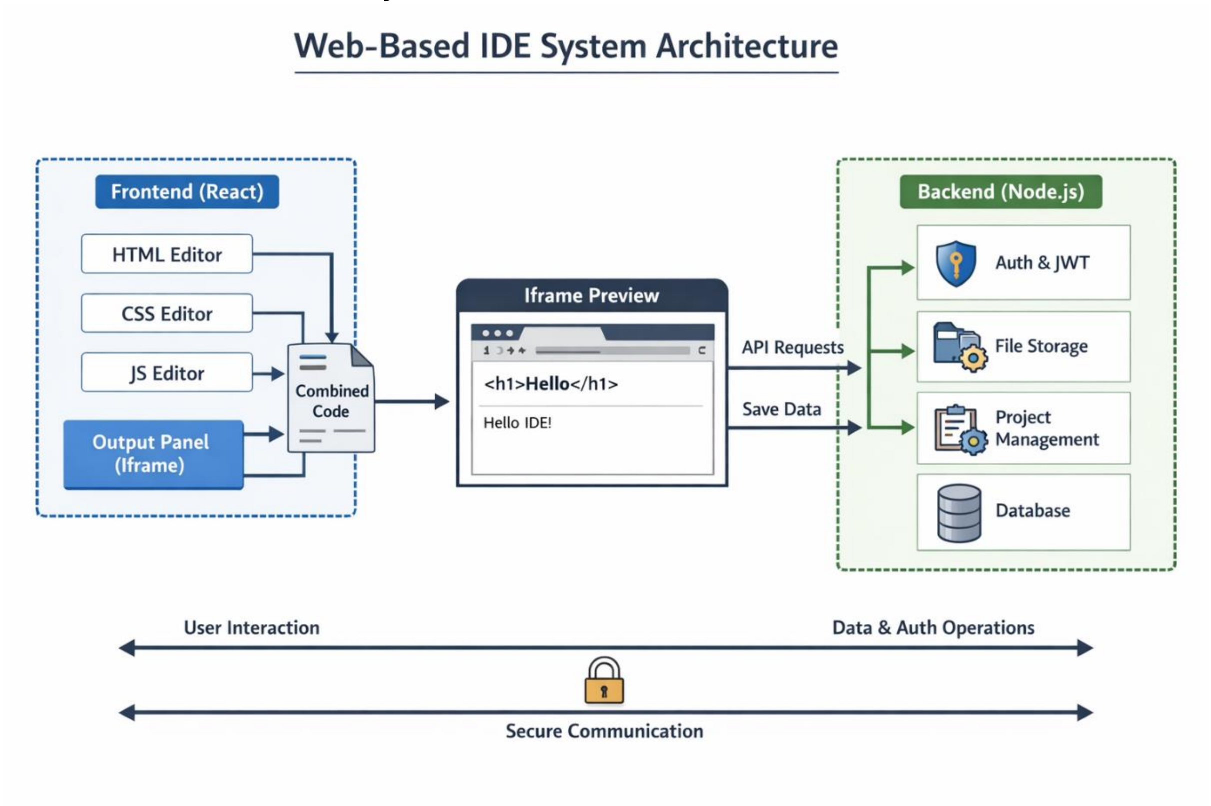
The frontend is built with **React**, which manages the editors, application state, and the output view. A code editor library like **Monaco Editor** is used to provide syntax highlighting, auto-completion, and error hints. Linting and formatting tools such as **ESLint, Prettier, HTMLHint, and Stylelint** help analyze code quality.

The backend is built with **Node.js and Express**. It does not execute user code for security reasons. Instead, it handles **authentication, saving recent files, managing projects, versioning, and database access**. Authentication is handled using **JWT**, and data is stored in a database such as **PostgreSQL using Sequelize**.

How to use the IDE

A user logs in, opens or creates a project, and writes code in the HTML, CSS, and JavaScript editors. As they type, the IDE updates the iframe automatically and displays the output. The user can save files, reopen recent work, and continue from where they left off.

Preview of the web based IDE System Architecture


Web-Based IDE System Architecture
Frontend (React)
HTML Editor
CSS Editor
JS Editor
Output Panel (Iframe)
Combined Code
Iframe Preview
<h1>Hello</h1>
Hello IDE!
API Requests
Save Data
Backend (Node.js)
Auth & JWT
File Storage
Project Management
Database
User Interaction
Data & Auth Operations
Secure Communication