## **Module 1 Assignment 2: Basic Dynamic Web Project using Node.js and EJS - Creating and Displaying User Information**

Saurabh Kale

IFT 458/554: Middleware Programming & Database Security

Dinesh Sthapit

Aug 26th, 2023

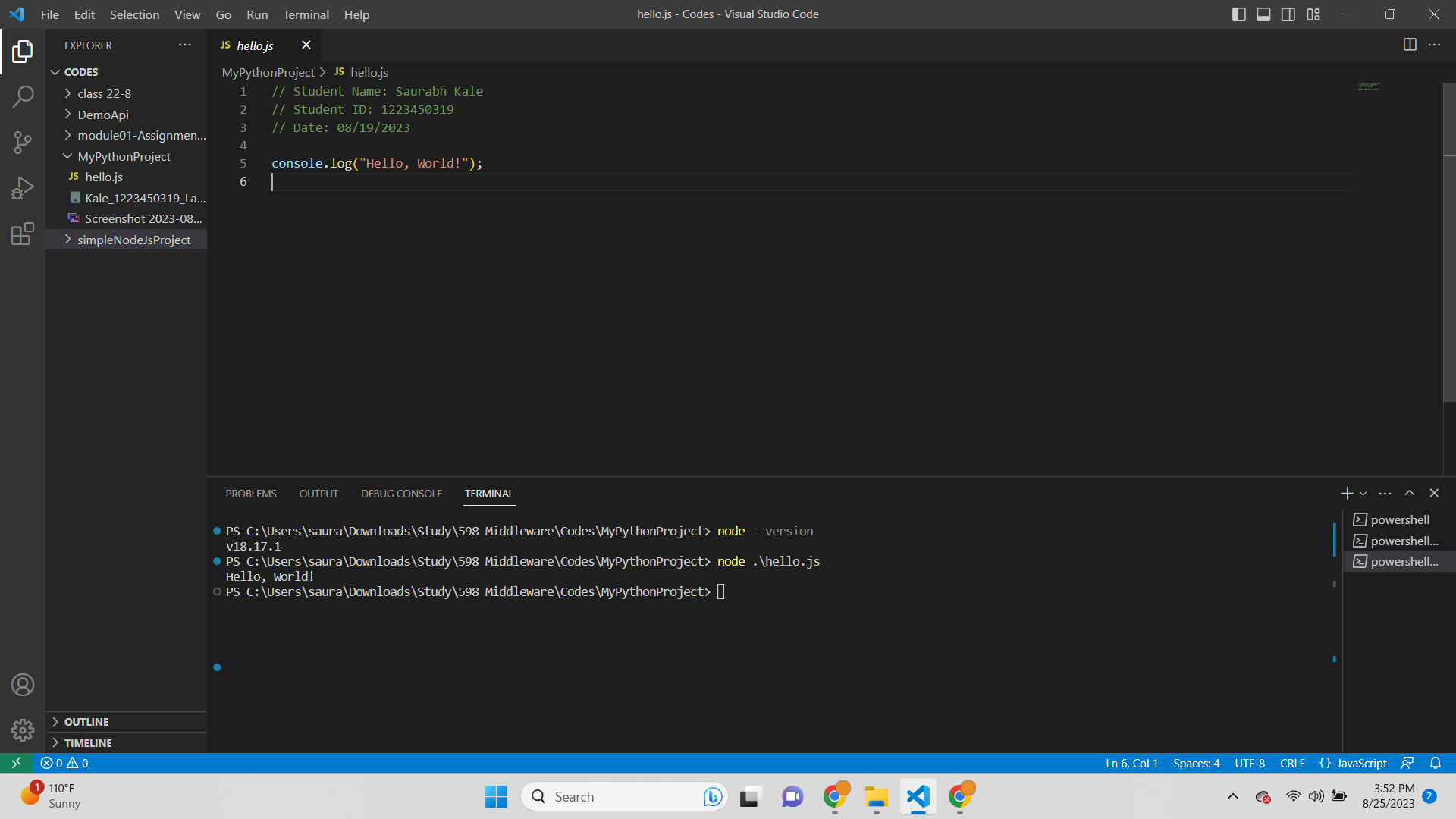
* **Summary of your understanding of Node.js**

Node.js is a powerful open-source JavaScript runtime that has revolutionized server-side programming. It enables developers to execute JavaScript code on servers, bringing the language's versatility from the browser to the backend. Built on Chrome's V8 JavaScript engine, Node.js employs an event-driven, non-blocking I/O model, enabling efficient handling of asynchronous operations.

Node.js operates through an event loop, allowing it to manage multiple tasks simultaneously without blocking the main execution thread. This approach ensures scalability and responsiveness, making it an ideal choice for real-time applications and microservices architectures. Its single-threaded design, although unconventional, is remarkably efficient, as it can manage numerous connections concurrently without the overhead of traditional multi-threading.

Its relevance in server-side programming is remarkable. With Node.js, developers can use JavaScript throughout the entire development stack, ensuring code consistency and easing the learning curve. The Node Package Manager (npm) offers a vast repository of modules and libraries, expediting development. Its applications span real-time chat applications, online gaming, and streaming services due to its ability to handle high traffic loads efficiently. Companies like Netflix, LinkedIn, and PayPal have embraced Node.js for its scalability and performance.(Documentation | Node.js)

* **Screenshot of Successful Installation of Node.js on local Machine-**

****

* **A brief explanation of the additional functionalities Node.js offers over standard browser-side JavaScript.**

Node.js extends the capabilities of standard browser-side JavaScript by providing a runtime environment for server-side applications. While browser-side JavaScript focuses on enhancing user interfaces, Node.js introduces functionalities tailored for server-side programming. This runtime environment facilitates server-side scripting, network communications, and file system access, expanding the language's versatility.

One notable feature is non-blocking I/O, which enables Node.js to efficiently handle asynchronous operations, such as handling multiple client requests concurrently without blocking the execution thread. This is in contrast to browser JavaScript, which is primarily single-threaded and focuses on responding to user interactions.

Node.js also introduces a wealth of modules through its package manager, npm. These modules cover a wide array of functionalities, from building web servers to handling data storage, authentication, and more. This rich ecosystem accelerates development by offering pre-built solutions to common challenges.

Additionally, Node.js enables developers to use JavaScript consistently throughout the entire development stack, from frontend to backend. This unified language approach streamlines development and encourages code reusability.(Documentation | Node.js)

* **The Node.js script you created, along with a description of what it does and a screenshot of the script's output.**

**Script-**

const fs = require('fs');

try {

const data = fs.readFileSync(`${\_\_dirname}/samples\_document\_txt\_sample1.txt`, 'utf8');

console.log(data);

} catch (err) {

console.error(err);

}

**Description-**

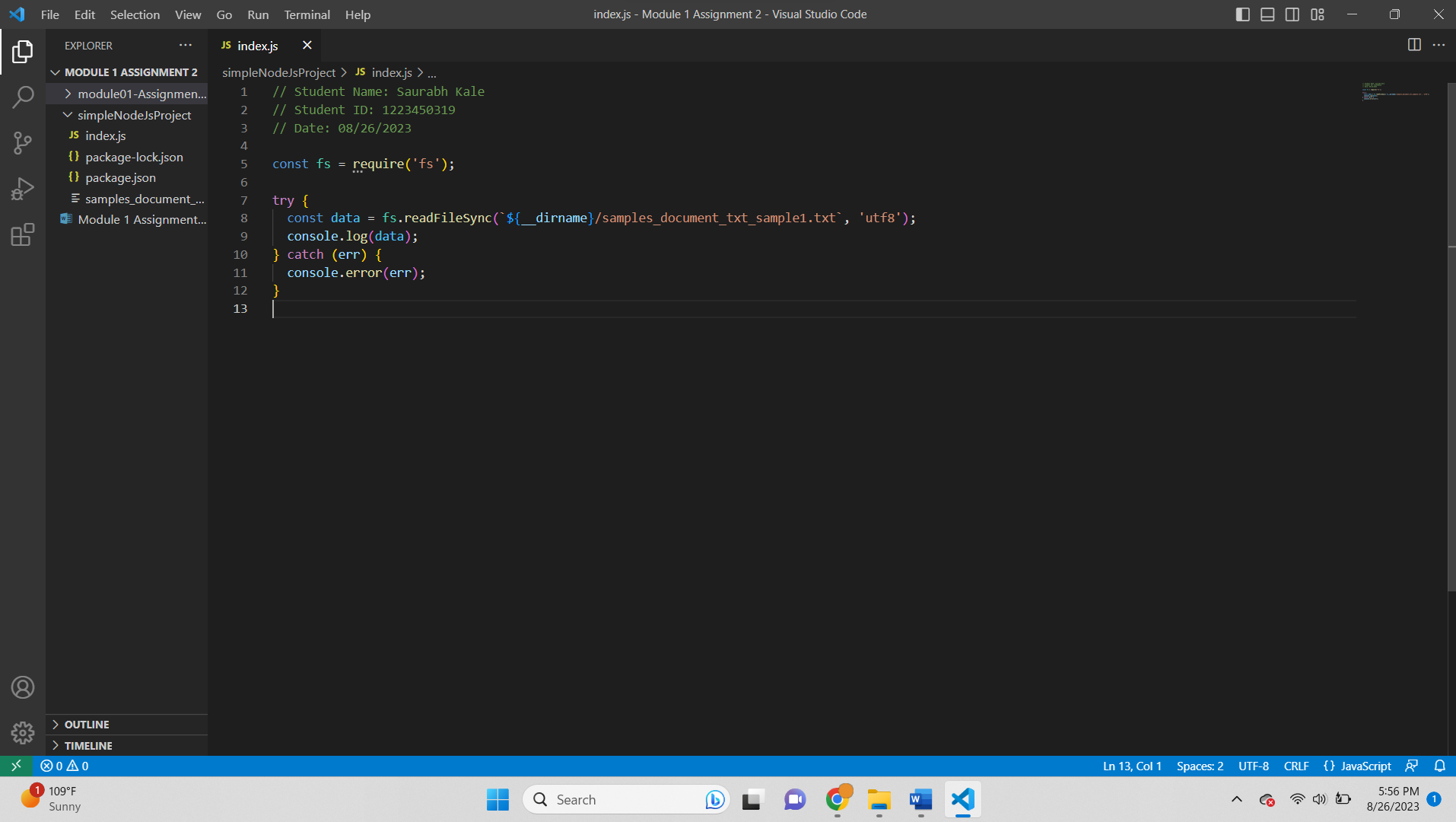
The provided code snippet demonstrates the usage of the fs (File System) module in Node.js for reading the contents of a text file synchronously. First, the fs module is imported using the require function. The code then attempts to read the content of a specific text file named "samples\_document\_txt\_sample1.txt" using the readFileSync method. The \_\_dirname variable represents the current directory of the script, ensuring the correct path to the file.

If the file is successfully read, its contents are stored in the data variable and subsequently printed to the console using console.log(data). In case of an error during the file reading process, the catch block is triggered, and the error is displayed using console.error(err).

It's important to note that the synchronous nature of readFileSync means that the program will wait for the file reading to complete before moving on, potentially causing blocking behavior in applications with high concurrency requirements.

This code showcases a basic example of reading a file synchronously in Node.js and handling potential errors that may occur during the process.

**Screenshot-**



* **Your reflection on the experience of setting up and working with Node.js.**

Setting up and working with Node.js has been an enlightening and empowering experience.The process began with installing Node.js and npm, which turned out to be straightforward thanks to the well-documented instructions provided on the official website. The Node Package Manager (npm) provided an extensive library of modules, simplifying the integration of third-party functionalities into my projects. Moreover, the ability to utilize JavaScript both on the client and server sides introduced a remarkable coherence to my projects. Code reusability and a shared language made transitioning between frontend and backend development smoother. One notable aspect was the vast community support surrounding Node.js. Online forums, tutorials, and an abundance of open-source projects provided invaluable resources. This collaborative atmosphere boosted my learning curve and problem-solving capabilities.

**Req 4: Research HTTP Status numbers and provide a brief description of each of the status values.**

* Informational responses (100 – 199):

These status codes indicate provisional responses from the server. They inform the client that the request has been received and understood, and the server is continuing to process it. These codes are often used for communication between the client and server headers.

* Successful responses (200 – 299):

Status codes in this range indicate that the client's request was successfully received, understood, and accepted by the server. These codes indicate that the requested action has been taken or that the request has been successfully processed.

* Redirection messages (300 – 399):

Redirection status codes indicate that the client needs to take additional action to complete the request. These codes typically occur when a resource has been moved, and the client is redirected to a new location.

* Client error responses (400 – 499):

These status codes indicate that the client's request cannot be fulfilled or processed due to an error on the client's side. Common reasons include bad syntax in the request, unauthorized access, or requesting a non-existent resource.

* Server error responses (500 – 599):

Server error status codes indicate that the server encountered an error while trying to fulfill a valid request. These codes suggest that the problem is on the server's end, such as server misconfigurations or issues with backend processes.

**Req 5: Research HTTP Response headers and provide a brief description of each**

* UI Elements for HTTP Response Headers:

These elements are part of a user interface (UI) designed to manage and configure HTTP response headers. They include input fields, checkboxes, dropdowns, and buttons that allow users to define how the server responds to client requests. Users can control aspects such as caching, security policies, and communication standards by manipulating these UI elements.

* Feature Page Elements:

Feature page elements refer to the components of a webpage or application screen dedicated to a specific feature or functionality, in this case, managing HTTP response headers. These elements present information related to response headers, such as the list of headers, their values, and options to modify them. They can also include navigation elements for users to access related features or settings.

* Actions Pane Elements:

The actions pane elements are interactive components within the UI that offer options for performing specific tasks related to HTTP response headers. This might include buttons for adding new headers, editing existing ones, saving changes, or discarding modifications. The actions pane provides a focused area where users can trigger actions without navigating through various screens.

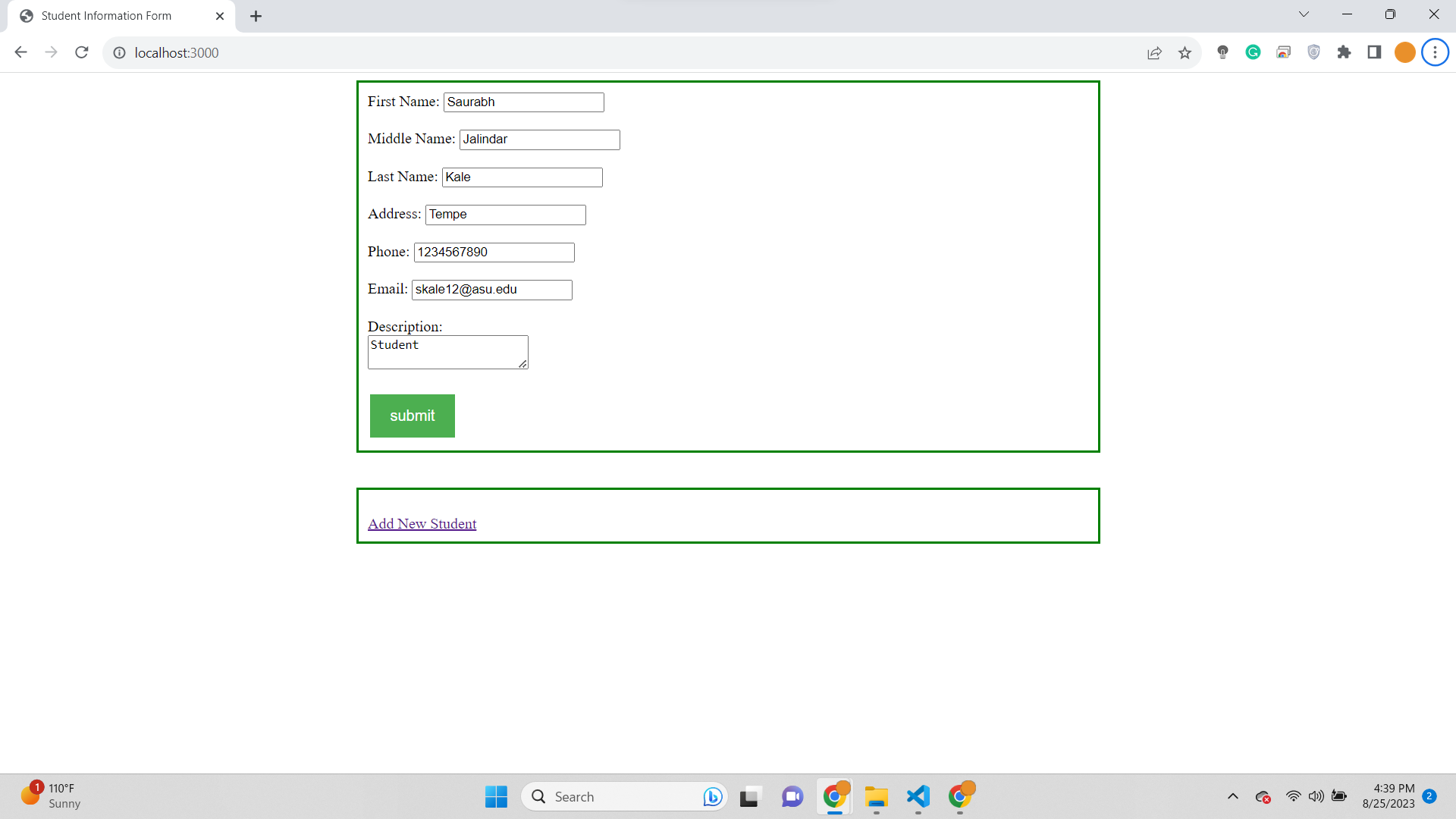
* Add or Edit Custom HTTP Response Header Dialog Box:

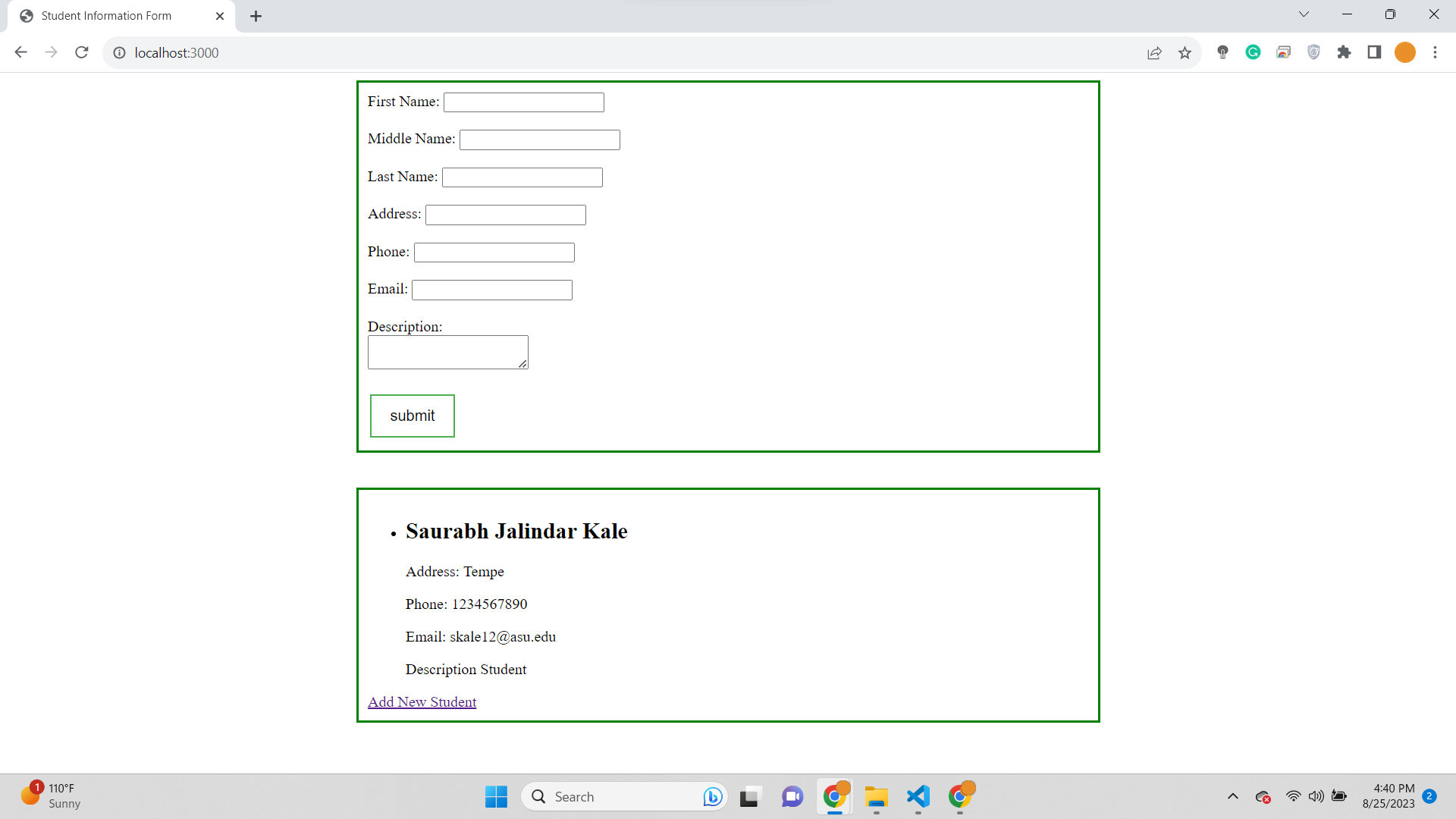
This dialog box is presented to users when they want to add a new custom HTTP response header or modify an existing one. It includes input fields to define the header's name and value, as well as options to specify the scope and behavior of the header. This dialog box ensures that users can manage response headers with precision and clarity.

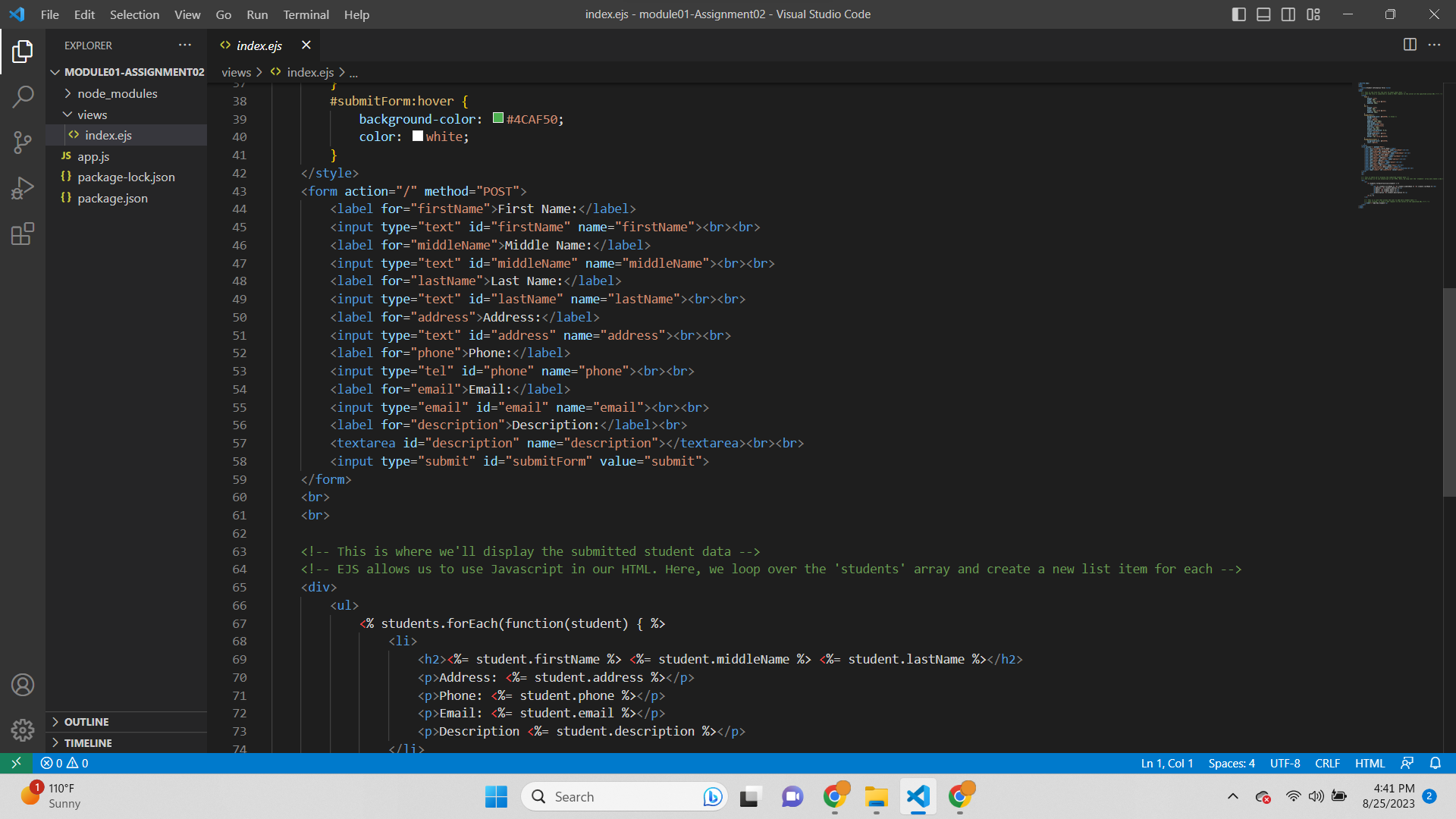
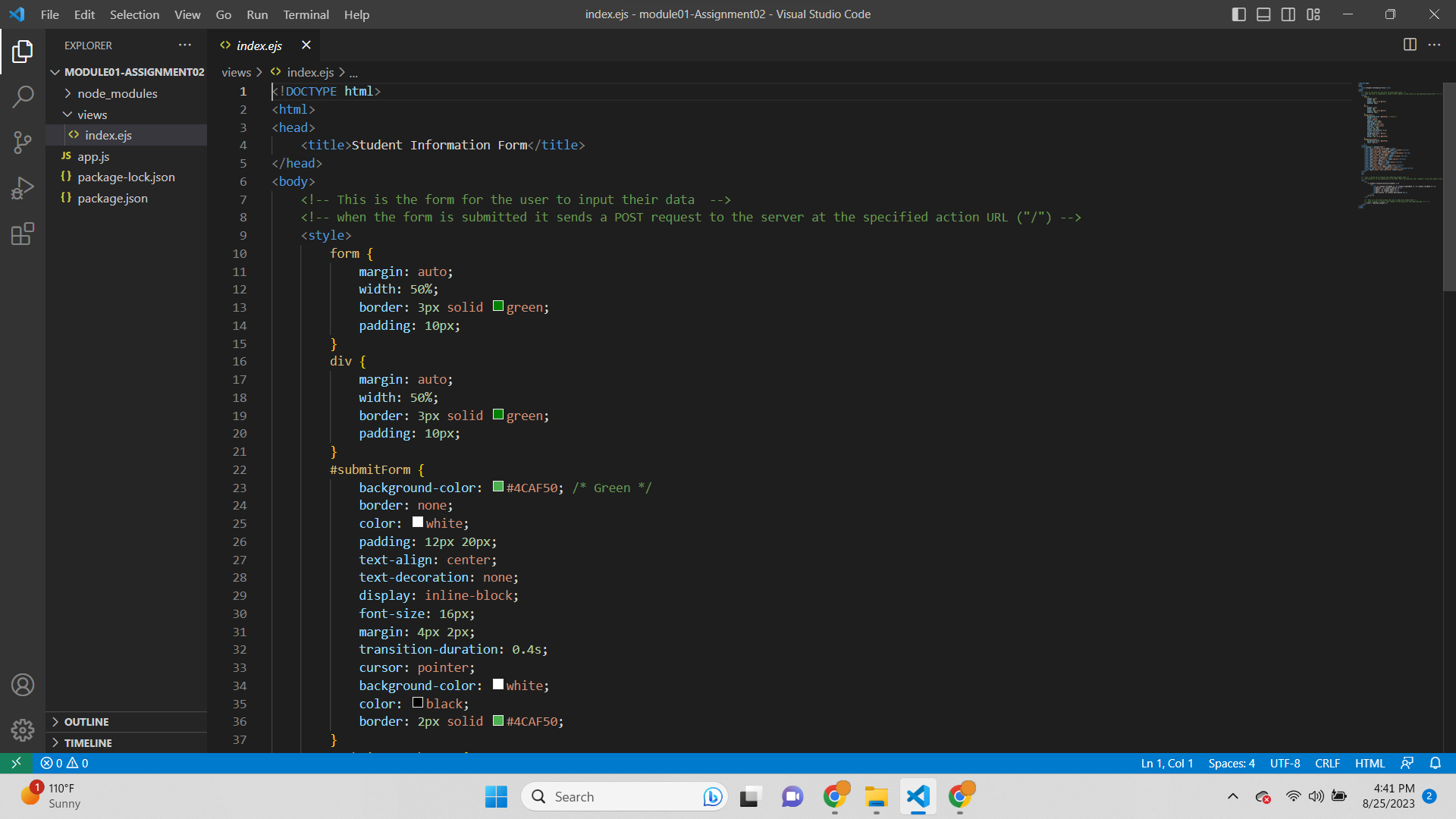
* Set Common HTTP Response Headers Dialog Box:

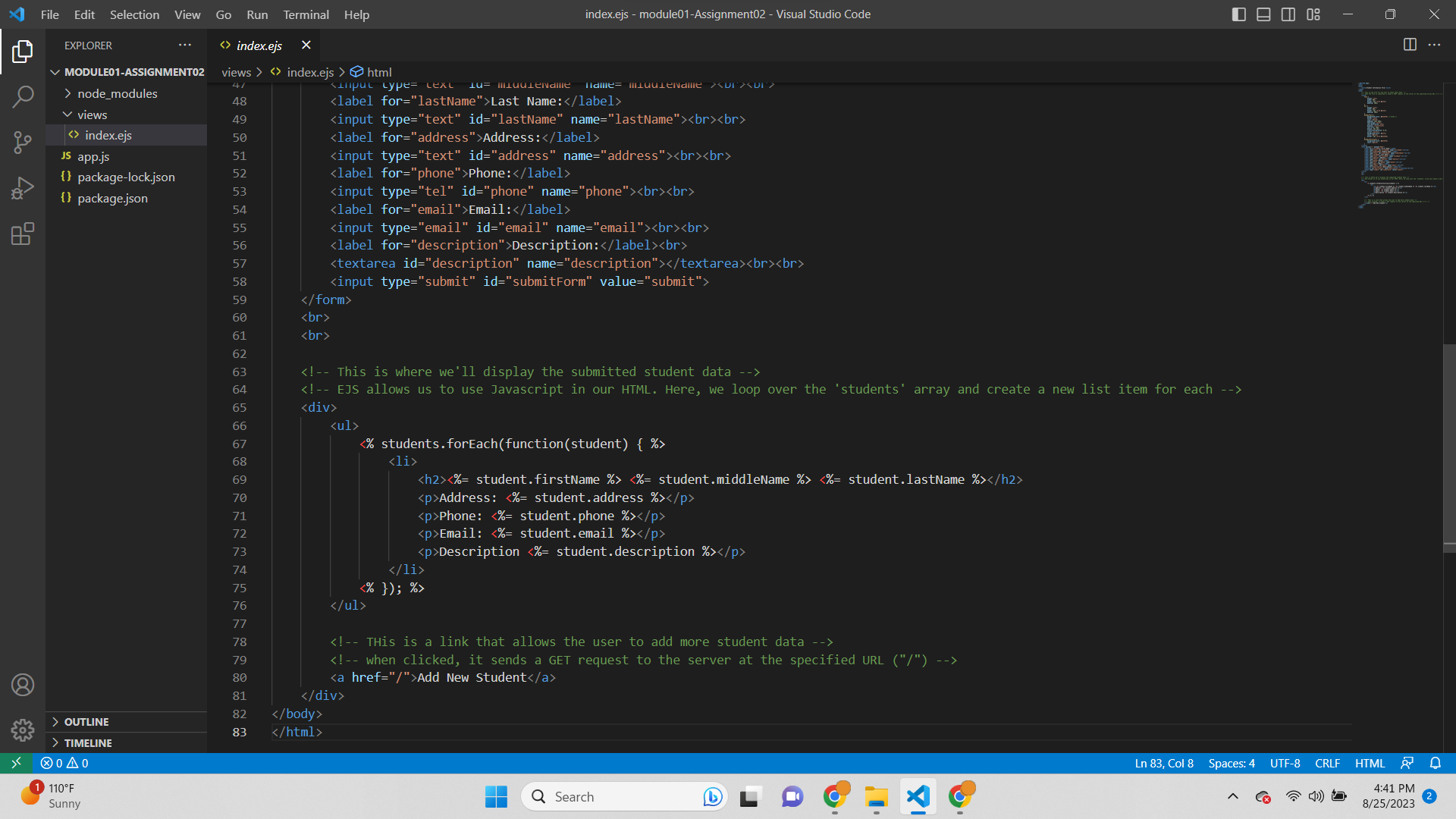
This dialog box serves as a consolidated interface for setting frequently used or standard HTTP response headers. It presents a list of common headers along with input fields to specify their values. Users can conveniently configure headers like "Content-Type" or "Cache-Control" without needing to remember specific syntax or options.

Output Screenshot-





Code Screenshot-



A computer screen with text on it

Description automatically generated

A computer screen shot of a program

Description automatically generated

**References-**

Documentation | Node.js. Node.js. https://nodejs.org/en/docs