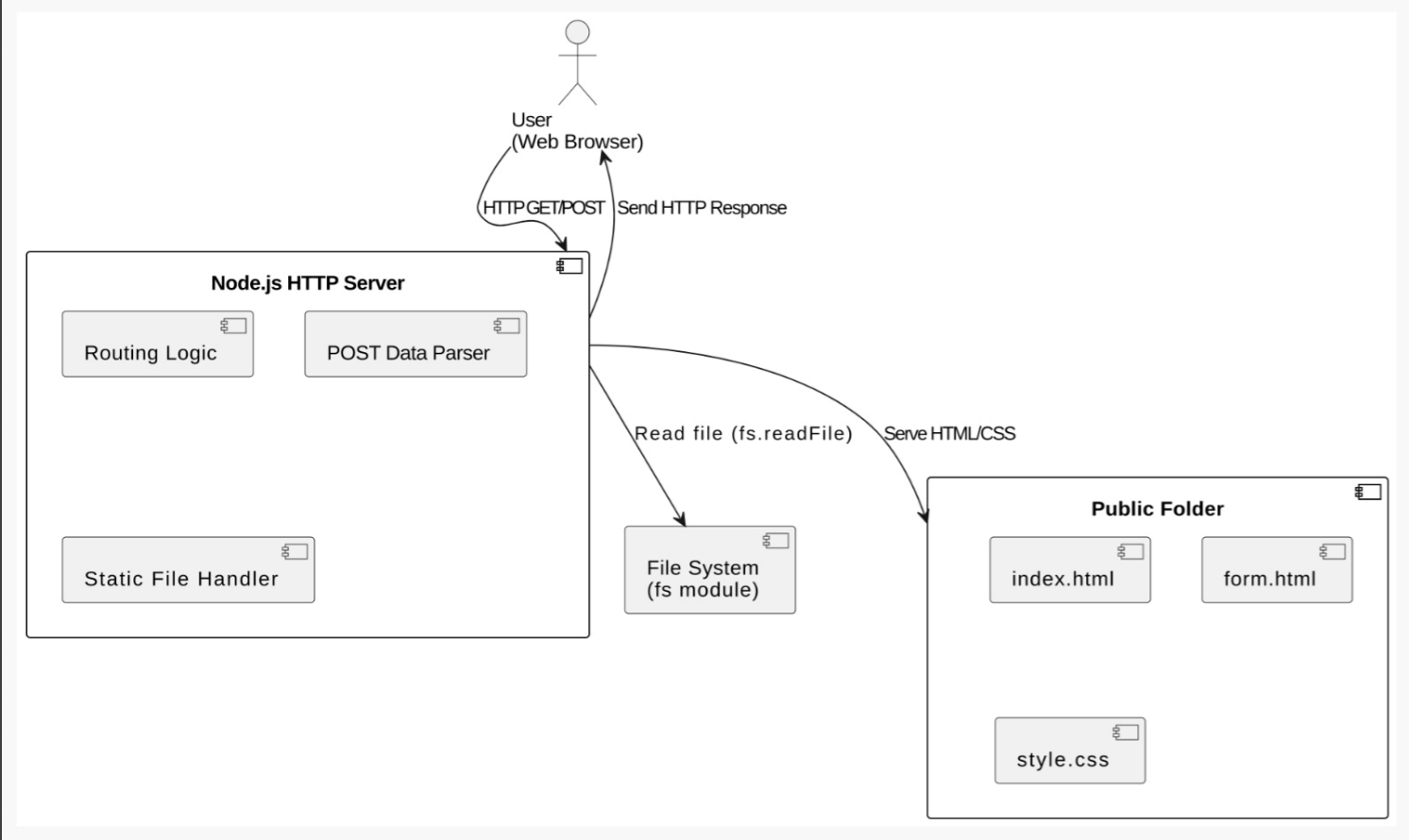
**Simple HTTP Web Server using Node.js**

**Team Information**

* Syed Muhammad Irtiza — 22K-4638
* Muhammad Sadiq – 22K-4303

**Abstract**

In the modern web ecosystem, web servers are foundational components that facilitate communication between users and applications over the internet. This project presents the design and implementation of a lightweight HTTP server using Node.js that can efficiently handle HTTP GET and POST requests and serve static HTML content. It aims to provide a hands-on understanding of how web servers operate at a low level without the abstraction provided by high-level frameworks. The final deliverable is a functional, minimalistic web server capable of processing form inputs, rendering pages, and managing client requests—laying the groundwork for more complex server-side applications.



**Introduction**

With the rise of full-stack web development, developers often rely heavily on sophisticated libraries and frameworks that abstract core functionality. However, gaining a clear understanding of how HTTP requests are handled natively is crucial for robust backend development. This project delves into the construction of a basic HTTP server using the built-in http module in Node.js, emphasizing core concepts like request routing, status codes, MIME types, and form data processing.

The project environment involves a server.js file that bootstraps the HTTP server, a public/ directory that holds static files (HTML, CSS), and logic to handle POST requests from HTML forms. The implementation focuses on clarity, modularity, and adherence to web standards.

**Objectives**

The primary objectives of this project are as follows:

1. Develop a basic HTTP server using Node.js that listens on a specific port.
2. Serve static content including HTML pages from a designated directory.
3. Handle GET and POST requests manually without using Express.js or similar frameworks.
4. Parse form data from POST requests and display it dynamically.
5. Implement error handling for undefined routes (404 page).
6. Provide a clean, readable codebase and maintainable folder structure.

**Problem Statement**

Many beginner-level developers lack clarity on how data flows between the frontend and backend during an HTTP request lifecycle. High-level tools like Express.js and Flask simplify server creation, but they also obscure the underlying mechanics. This project attempts to bridge that gap by constructing a raw HTTP server that reveals and reinforces fundamental backend concepts such as routing, request parsing, header management, and asynchronous processing.

**Methodology**

The methodology adopted for this project is agile and iterative:

**Project Setup:**

* Initialize a Node.js project with a clear directory structure including source code, static assets, and documentation.

**Server Development:**

* Use Node.js built-in http and fs modules.
* Serve files based on routes such as /, /about, or /submit.
* Implement logic to handle Content-Type headers based on file extensions.

**Form Handling:**

* Parse data from POST requests using buffer streams.
* Display submitted data on a result page.

**Testing & Validation:**

* Manually test with multiple browsers.
* Handle edge cases like 404 responses and unsupported request methods.

**Documentation:**

* Include comprehensive comments in the code.
* Prepare detailed project report.

**Team Contributions**

**Syed Muhammad Irtiza – 22K-4638**

* Project setup and repository initialization
* Development of the server.js core logic
* Routing implementation and MIME handling
* Writing 404 response and modular functions

**Muhammad Sadiq – 22K-4303**

* Creation of HTML forms and static files in /public
* Implementation of form data parsing in POST requests
* Displaying parsed data on a dynamic response page
* Drafting the initial version of the report and user guide

**References**

* Node.js Documentation – https://nodejs.org/en/docs/
* Mozilla Developer Network (MDN) – https://developer.mozilla.org/
* W3Schools HTML Forms – https://www.w3schools.com/html/html\_forms.asp
* HTTP Status Codes – https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
* Node.js Tutorial (TutorialsPoint) – https://www.tutorialspoint.com/nodejs/nodejs\_web\_module.html