

## Day 18 - Node.js Core Modules

### Challenge 4: File System (fs) Module

In this challenge, the Node.js fs module with promises is used to take user input from the command line and store it in a file named feedback.txt. The same file is then read asynchronously and its contents are displayed on the console to confirm successful file operations.

#### Output:

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files: challenge4\_fs.js, challenge5\_http.js, challenge6\_events.js, feedback.txt, index.html, and package.json. The main editor area displays the challenge4\_fs.js file content:

```
D:\Wipro Training\OlympusAssignments\Day18CodingSolutionNodeJS\challenge4_fs.js
1 const fs = require('fs').promises;
2 const path = require('path');
3
4 (async () => {
5   try {
6     const userInput = process.argv[2];
7
8     if (!userInput) {
9       console.log("Please pass input text. Example:");
10      console.log("node challenge4_fs.js \"Your feedback here\"");
11      return;
12    }
13
14    const filePath = path.join(__dirname, 'feedback.txt');
15
16    await fs.writeFile(filePath, userInput, 'utf8');
17    console.log('Data written successfully.');
18
19    console.log('Reading file...');
20    const content = await fs.readFile(filePath, 'utf8');
21    console.log(content);
22
23  } catch (err) {
24    console.error('Error:', err);
25  }
26 })();
```

The Terminal tab at the bottom shows the command `node challenge4_fs.js "Node.js is awesome!"` being run, followed by the output: "Data written successfully.", "Reading file...", and "Node.js is awesome!".

## Challenge 5: HTTP Module

This challenge demonstrates creating a basic HTTP server using Node.js without Express. The server handles multiple routes and serves static HTML content when available, while also supporting graceful shutdown.

### Output:



## Challenge 6: Events Module

In this challenge, the `EventEmitter` class is used to simulate event-driven behavior. Custom events such as user login, logout, and session expiration are emitted dynamically to represent a simple notification system.

### Output:

The screenshot shows a dark-themed instance of Visual Studio Code. In the Explorer sidebar, there are several files listed under the folder "DAY18CODINGSOLUTIONNOD...": `challenge4_fsjs`, `challenge5_http.js`, and `challenge6_events.js`. The `challenge6_events.js` file is currently selected and open in the main editor area. The code implements a `MyNotifier` class that extends `EventEmitter`. It defines three event listeners: `'userLoggedIn'`, `'userLoggedOut'`, and `'sessionExpired'`. The `'userLoggedIn'` and `'userLoggedOut'` events log the user's name. The `'sessionExpired'` event logs a message about the session expiring. A function `simulateUserActivity` is also defined to demonstrate dynamic emits. The terminal at the bottom shows the command `node challenge6_events.js` being run, followed by the console output: "User John logged in.", "User John logged out.", and "Session for John expired."

```
const EventEmitter = require('events');
class MyNotifier extends EventEmitter { }
const notifier = new MyNotifier();
notifier.on('userLoggedIn', (username) => {
  console.log(`User ${username} logged in.`);
});
notifier.on('userLoggedOut', (username) => {
  console.log(`User ${username} logged out.`);
});
// Bonus: sessionExpired
notifier.on('sessionExpired', (username) => {
  console.log(`Session for ${username} expired.`);
});
// Simulate dynamic emits
function simulateUserActivity(username) {
  // user logs in
  notifier.emit("userLoggedIn", username);
  // after 3 seconds, user logs out
  setTimeout(() => {
    notifier.emit("userLoggedOut", username);
  }, 3000);
}
simulateUserActivity("John");
```

PS D:\Wipro Training\OlympusAssignments\Day18CodingSolutionNodeJS> node challenge6\_events.js  
User John logged in.  
User John logged out.  
Session for John expired.  
PS D:\Wipro Training\OlympusAssignments\Day18CodingSolutionNodeJS>