Day-4

Installation of Postman

Postman is a popular tool for testing APIs. Here's how to install it:

1. **Download Postman**:
   - Go to Postman Download Page.
   - Choose the version for your OS (Windows, macOS, or Linux).
2. **Installation**:
   - On Windows: Run the `.exe` file and follow the installation wizard.
   - On macOS: Drag the Postman app into your Applications folder.
   - On Linux: Follow the terminal commands as specified on their website.
3. **Launch Postman**:
   - Open Postman and sign in or create a new account.

Event Listeners

Event listeners are functions that wait for specific events to occur (e.g., clicks, mouse movements). Here's an example with HTML and JavaScript:

HTML Example (eventlistener.html):

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Event Listener Example</title>
</head>
<body>
    <button id="clickBtn">Click Me!</button>
    <script>
        const button = document.getElementById('clickBtn');
        button.addEventListener('click', () => {
            alert('Button Clicked!');
        });
    </script>
</body>
</html>
```

- The `addEventListener()` method attaches an event handler to the button, triggering the alert when clicked.

setInterval and setTimeout

These are JavaScript functions to execute code after a certain delay.

1. **setTimeout**: Executes a function after a specified time.

   ```
   javascript
   Copy code
   setTimeout(() => {
       console.log('This will run after 3 seconds');
   }, 3000); // 3000 milliseconds = 3 seconds
   ```

2. **setInterval**: Executes a function repeatedly after specified intervals.

   ```
   javascript
   Copy code
   setInterval(() => {
       console.log('This message will log every 2 seconds');
   }, 2000);
   ```

Async, Await, and Promises

Promises allow you to handle asynchronous operations more effectively. Here's how:

1. **Promises**:

   ```
   javascript
   Copy code
   const myPromise = new Promise((resolve, reject) => {
       setTimeout(() => {
           resolve('Promise resolved!');
       }, 2000);
   });

   myPromise.then((message) => {
       console.log(message);  // Logs "Promise resolved!" after 2
   seconds
   });
   ```

2. **Async/Await**: Syntax to work with Promises in a cleaner way.

   ```
   javascript
   Copy code
   async function fetchData() {
       try {
   ```

```
        const data = await myPromise;
        console.log(data);  // Waits for promise to resolve
    } catch (error) {
        console.log(error);
    }
}
fetchData();
```

Node.js Modules and File System (fs) Concept

Modules in Node.js are like libraries that you can import and use. The `fs` module is used to interact with the file system.

1. **Importing Modules**:

   ```
   javascript
   Copy code
   const fs = require('fs');  // Import fs module
   ```

2. **Reading a File**:

   ```
   javascript
   Copy code
   fs.readFile('example.txt', 'utf8', (err, data) => {
       if (err) {
           console.log(err);
       } else {
           console.log(data);
       }
   });
   ```

3. **Writing to a File**:

   ```
   javascript
   Copy code
   fs.writeFile('output.txt', 'This is Node.js', (err) => {
       if (err) throw err;
       console.log('File has been written!');
   });
   ```

Creating a Server with HTTP in Node.js

The HTTP module allows you to create web servers.

1. **Basic HTTP Server**:

   ```
   javascript
   Copy code
   const http = require('http');

   const server = http.createServer((req, res) => {
       res.statusCode = 200;
       res.setHeader('Content-Type', 'text/plain');
       res.end('Hello, Node.js Server!\n');
   });
   ```

```
server.listen(3000, () => {
    console.log('Server running on port 3000');
});
```

2. **Serving an HTML File**:

```
javascript
Copy code
const http = require('http');
const fs = require('fs');
const path = require('path');

const server = http.createServer((req, res) => {
    const filePath = path.join(__dirname, 'index.html');
    fs.readFile(filePath, (err, data) => {
        if (err) {
            res.statusCode = 500;
            res.end('Error loading the file');
        } else {
            res.statusCode = 200;
            res.setHeader('Content-Type', 'text/html');
            res.end(data);
        }
    });
});

server.listen(3000, () => {
    console.log('Server running on port 3000');
});
```

Creating a Login and Signup Page with HTTP API in Node.js

We will create a simple login and signup API using the HTTP module. This API will allow users to register and login by sending HTTP requests.

Steps:

1. **Basic Node.js HTTP Server**.
2. **POST Request Handling for Signup**.
3. **POST Request Handling for Login**.

Code:

1. **Setting Up HTTP Server**:

```
javascript
Copy code
const http = require('http');
const fs = require('fs');
const url = require('url');

const users = {};  // A simple object to store users

const server = http.createServer((req, res) => {
    const parsedUrl = url.parse(req.url, true);
    if (req.method === 'POST' && parsedUrl.pathname === '/signup') {
        let body = '';
        req.on('data', chunk => {
            body += chunk.toString();
        });
        req.on('end', () => {
            const { username, password } = JSON.parse(body);
            if (!users[username]) {
                users[username] = password;
                res.statusCode = 200;
                res.end('Signup successful');
            } else {
                res.statusCode = 400;
                res.end('User already exists');
            }
        });
    } else if (req.method === 'POST' && parsedUrl.pathname ===
'/login') {
        let body = '';
        req.on('data', chunk => {
            body += chunk.toString();
        });
        req.on('end', () => {
            const { username, password } = JSON.parse(body);
            if (users[username] && users[username] === password) {
                res.statusCode = 200;
                res.end('Login successful');
            } else {
                res.statusCode = 400;
                res.end('Invalid credentials');
            }
        });
    } else {
        res.statusCode = 404;
        res.end('Not Found');
    }
});

server.listen(3000, () => {
    console.log('Server running on port 3000');
});
```

Explanation:

- **Signup API**: Collects `username` and `password` and stores them in the `users` object.
- **Login API**: Checks the stored `username` and `password` and verifies credentials.

To test the API, you can use **Postman** to send POST requests to `http://localhost:3000/signup` and `http://localhost:3000/login`.