Index.html

This is a simple HTML page with embedded JavaScript that interacts with the **RESTful API** you created using **Express.js**. The page fetches the list of users from the server and displays them in a list (<ul> element).

1. HTML Structure

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>RESTful API with Fetch</title>
</head>
<body>
<h1>Users</h1>
<ul id="user-list"></ul>
```

- **<!DOCTYPE html>**: Declares the document as an HTML5 document.
- **<html lang="en">**: Specifies that the language of the document is English.
- **<meta charset="UTF-8">**: Sets the character encoding to UTF-8, allowing for the representation of most characters.
- **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Ensures proper scaling on mobile devices.
- **<title>**: Sets the title of the page as "**RESTful API** with Fetch".
- **<h1>**: A header that displays "Users".
- **<ul id="user-list">**: An unordered list where user data will be appended dynamically.

2. JavaScript Code for Fetching Users

```
<script>
async function fetchUsers() {
  const response = await fetch('http://localhost:3000/users');
  const users = await response.json();

  const userList = document.getElementById('user-list');
  users.forEach(user => {
    const listItem = document.createElement('li');
    listItem.textContent = `${user.id}: ${user.name}`;
    userList.appendChild(listItem);
  });
}

// Call the function to fetch users on page load
fetchUsers();
</script>
```

Breakdown of the Script:

- **fetchUsers Function**:
  - This is an **asynchronous function** defined to fetch the list of users from the API.
- **fetch('http://localhost:3000/users')**:
  - The **Fetch API** is used to make a GET request to the **RESTful API endpoint /users**.
  - It sends an HTTP **GET** request to **http://localhost:3000/users** (your Express server running locally).

- **await response.json()**:
    - **await** pauses the function execution until the promise returned by fetch is resolved.
    - **response.json()** parses the **JSON** response body into a JavaScript object (users array).
- **document.getElementById('user-list')**:
    - Gets the reference to the <ul> element where users will be displayed.
- **users.forEach(user => { ... })**:
    - Iterates over the users array.
    - For each user object, it creates a new <li> element (listItem).
- **listItem.textContent = ${user.id}: ${user.name}``**:
    - Sets the text content of the list item to display the user's ID and name.
- **userList.appendChild(listItem)**:
    - Appends the newly created list item to the <ul id="user-list"> element.
- **fetchUsers()**:
    - The function is called immediately when the page loads to populate the user list.

How It Works Together

1. **Client Request**:
    - When you open the HTML file in a browser, the JavaScript function **fetchUsers** is invoked.
    - It sends a GET request to the **Express API** server at **http://localhost:3000/users**.

2. **Server Response**:

    - The Express server handles the request and returns a JSON response with the list of users:

```
[
 { "id": 1, "name": "Vijay Kumar" },
 { "id": 2, "name": "Priya Jhosi" }
]
```

3. **Rendering Data**:

    - The JavaScript code parses the response, iterates over the users, and adds each user as a <li> element to the <ul> on the webpage.
    - The resulting output in HTML would look like:

```
<ul id="user-list">
  <li>1: Vijay Kumar</li>
  <li>2: Priya Jhosi</li>
</ul>
```

How to Run This Code

1. **Start the Express Server**:

    - Run the Express server by executing: node **<your_server_file>.js.**

    - Ensure it listens on **http://localhost:3000**.

2. **Open HTML File in Browser:**

    - Open the HTML file in a browser **(e.g., file:///path/to/your/file.html).**
    - The browser makes a request to **the Express server** to fetch users and displays them on the page.

Key Concepts and Technologies Used

- **Fetch API**: A modern way to make HTTP requests from the browser.
- **Async/Await**: Syntax for handling asynchronous operations in a more readable manner.
- **DOM Manipulation**: Dynamically creating and inserting elements into the HTML document.
- **Express.js**: A backend framework used to serve the API data.

This example illustrates a simple **client-server interaction** using **RESTful APIs** and shows how to fetch and display data dynamically in a web page using JavaScript.