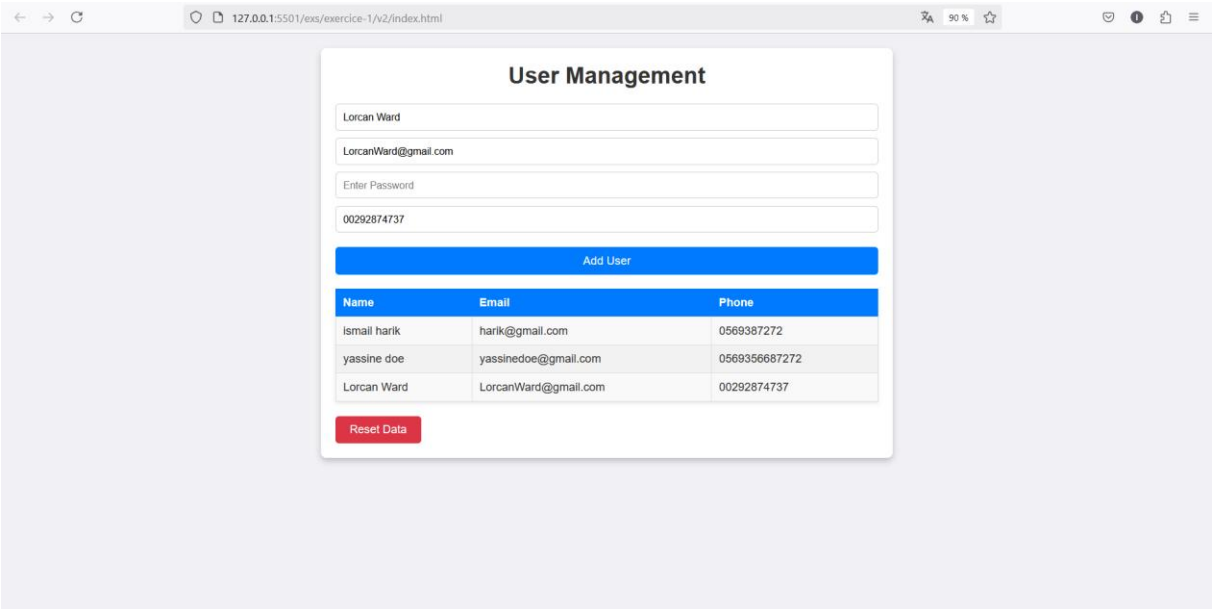
ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Dom TP2 User management SYSTEM :



Project Setup:

a. *Frontend: Use the following HTML and CSS code as your starting point*

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>User Management</title>
  <link rel="stylesheet" href="../style.css">
</head>
<body>
  <div class="container">
    <h1>User Management</h1>

    <form id="userForm">
      <input type="text" id="nameInput" placeholder="Enter Name" required />
      <input type="email" id="emailInput" placeholder="Enter Email" required />
      <input type="password" id="passwordInput" placeholder="Enter Password" required
/>
      <input type="tel" id="phoneInput" placeholder="Enter Phone" required />
      <button type="submit">Add User</button>
    </form>
```

```
  <table id="userTable">
    <thead>
      <tr>
        <th>Name</th>
        <th>Email</th>
        <th>Phone</th>
      </tr>
    </thead>
    <tbody id="userList">
      <!-- Users -->
    </tbody>
  </table>

  <button id="resetButton">Reset Data</button>
 </div>

 <script src="script.js"></script>
</body>
</html>
```

b. ***Backend Setup:***

1. Install nodeJS https://nodejs.org/fr
2. Create a **server.js** file with the following code below (const express…).
3. Create a users.json file in the same directory with the following content: **[ ].**
4. Install dependencies **npm install express body-parser cors.**
5. Run the server**: node server.js**.

```
const express = require("express");
const bodyParser = require("body-parser");
const fs = require("fs");
const cors = require("cors");
const app = express();
const PORT = 3000;
app.use(cors());
app.use(bodyParser.json());
const USERS_FILE = "users.json";
// Serve the JSON file
app.get("/userss", (req, res) => {
  fs.readFile(USERS_FILE, "utf8", (err, data) => {
    if (err) {
      return res.status(500).send({ error: "Error reading file" });
    }
    res.send(JSON.parse(data));
  });
});
// Save users to JSON file
app.post("/userss", (req, res) => {
  const users = req.body;
  fs.writeFile(USERS_FILE, JSON.stringify(users, null, 2), (err) => {
    if (err) {
      return res.status(500).send({ error: "Error saving file" });
    }
    res.send({ message: "Users saved successfully!" });
  });
});
app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

Questions:

1: Fetch Users from the API

1. Write a function named fetchUsers that:
    - Makes a GET request to the apiUrl.
    - Parses the JSON response.
    - Handles errors by logging them to the console and displaying an alert message.
2. After fetching the users, call another function to display them in a table. What function should you call here?

2: Display Users in the Table

1. Write a function named displayUsers that:
    - Accepts an array of users as an argument.
    - Uses the userList element (a <tbody>) to display the users.
    - Dynamically creates <tr> elements for each user with their name, email, and phone.
2. What happens if the users array is empty? Add a row with the message No users found. in that case.

3: Save Users to the API

1. **Write a function named saveUsers that:**
    - Accepts an array of users as an argument.
    - Sends a **POST** request to the **apiUrl** with the updated users as the request body.
    - Handles errors by logging them to the console.
    - How can you confirm the users were saved successfully? Add a success message
2. **Write a function named addUser that:**
    - Prevents the default form submission behavior.
    - Retrieves input values (name, email, password, and phone) from the form fields.
    - Fetches the existing users from the API.
    - Adds the new user to the array of users.
    - Saves the updated users array back to the API using the saveUsers function.
    - Refreshes the table by calling fetchUsers.

3. Reset Data.

Write a function named **resetData** that:

a. Saves an empty array to the API using the saveUsers function.
b. Refreshes the user list by calling fetchUsers.
c. Why is it necessary to call fetchUsers after resetting the data?

4. Attach Event Listeners

- Attach an event listener to the form (userForm) that calls the addUser function when the form is submitted.
- Attach an event listener to the reset button (resetButton) that calls the resetData function when the button is clicked.