Advanced JS - Guided Project

Introduction

In this phase of our E-Commerce application development, we'll focus on enabling seamless interaction between the frontend and backend. You'll learn how to fetch and post data using a local JSON Server, which will simulate real-world backend operations. Additionally, we'll implement user authentication using **Local Storage** and **Session Storage**.

Problem Statement

Follow the instructions in the Readme file from the source code to launch a JSON server and complete the tasks below.

1. Table Data

a. Use these URLs to **fetch data** and display it in the corresponding **tables on your web pages**.
b. Create a new JavaScript file inside the `js` folder and link it to the appropriate HTML file.
c. Use the **Fetch API** to retrieve data from the server and dynamically add it to the table's `<tbody>` section.
d. Repeat this process for **all tables** in the application to ensure each one is populated with server data.

```
Index:
http://localhost:3000/

Static files:
Serving ./public directory if it exists

Endpoints:
http://localhost:3000/products
http://localhost:3000/categories
http://localhost:3000/orders
```

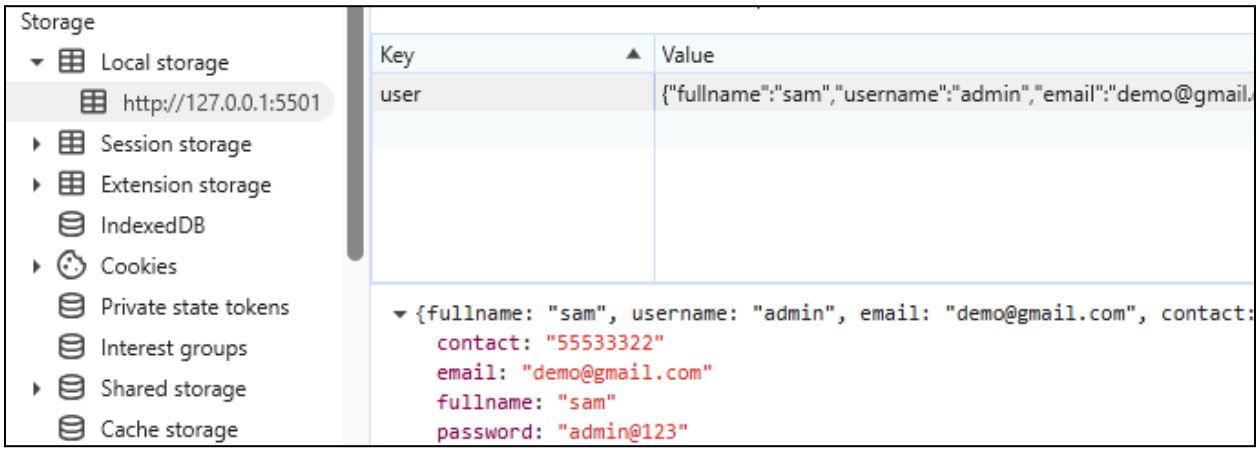
2. Form Data

a. When the form is submitted, **collect and validate** the input data to ensure it's complete and correct.
b. Use the **Fetch API** to **upload the data to the JSON Server** via a POST request.
c. **After successful submission, reset the form** and **redirect the user** to the corresponding table view page
   *—for example, submitting the **Add Product** form should navigate to the **View Products** page.*
d. Ensure the **newly added data appears** in the table immediately.
e. Repeat this process for **all form web pages** in the application to ensure the server is populated with form data.

3. Register

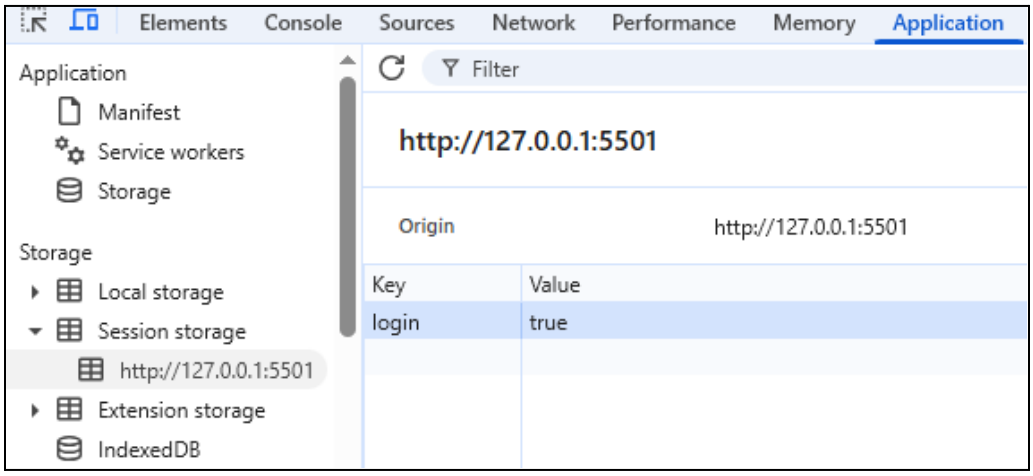
A user **must be registered** before accessing any page on the website.

a. Use the **Registration form** to collect user details and handle the **form submission** using JavaScript.
b. **Fetch the input data** and **validate** it upon submission.
c. Store the **data** as an **object in Local Storage** for reference and authentication purposes as shown below.



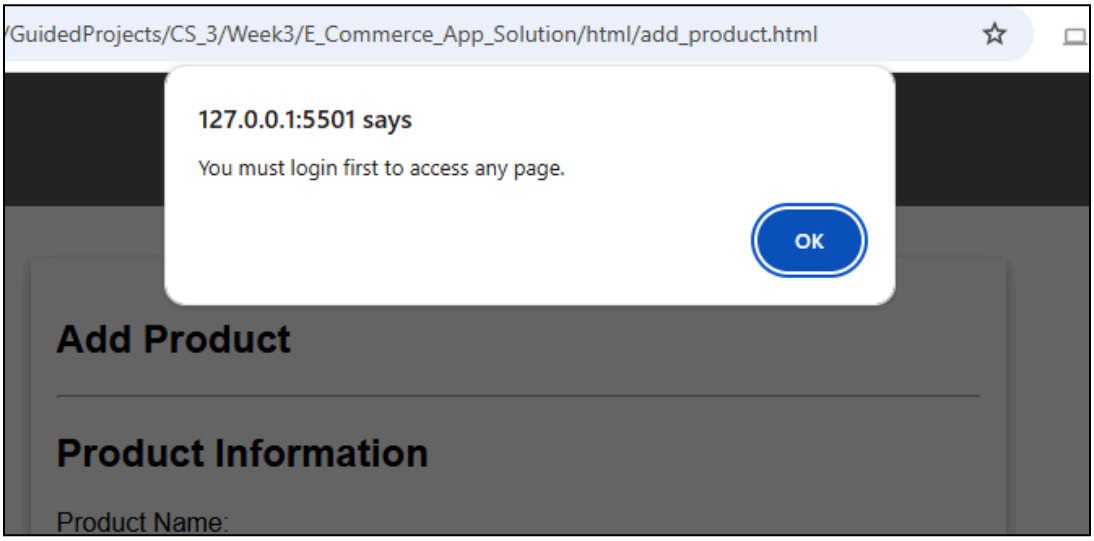
4. Sign In

a. Use the **Sign In form** to **capture the username and password** upon submission.
b. **Retrieve the stored user data** from **Local Storage** and **verify the credentials** against it.
c. If the credentials are valid, **set the login status to true** and **store it in Session Storage**. Also, **redirect the user to the home page** of the application.
d. If the credentials are not valid, **show a message on the page to register before Sign in.**
e. Ensure that access to the home page is **restricted** unless the user has successfully **signed in** and the login status is verified.



5. Validate Sign In

a. On loading **any web page**, first **check the log-in status** from **Session Storage**.
b. If the login status is **false or missing**, display an **alert** prompting the user to sign in, and **redirect** them to the **Sign In** page.
c. If the login status is true in session storage, allow access to the page and **display the username in the header**.
d. Apply this validation logic to **all HTML pages** in the application **except** `sign-in.html` and `register.html`.
e. Add an event listener to the **Sign Out** button on every page header, when clicked, **removes the login status from Session Storage** and **redirects the user to the Sign In page**.



!!! Happy Coding !!!