CO324 Lab 8: Browser Scripting

E/20/280

Part 1: Basics of Browser Scripting

1. What is Browser Scripting?

Browser scripting is use of programming language, primarily JavaScript, to create dynamic and interactive web pages. It enables client-side scriptings such as real time content changes, user input handling, and communication with servers without requiring a full page reload.

JavaScript and Browser Scripting: JavaScript, a lightweight language, is the primary language used for browser scripting. It interacts with the Document Object Model (DOM), enabling developers to manipulate HTML elements, handle user events, perform AJAX requests, and store data locally.

2. Common Features of Browser Scripting

DOM Manipulation: This enables interaction with the webpage structure by accessing and modifying HTML elements.

Event Handling: Allows real-time reactions to user actions like clicks or key presses.

AJAX: Allows real-time reactions to user actions like clicks or key presses.

Local Storage: Stores data for persistent use across sessions.

3. Example Use Cases

1. **Form Validation:** Checking if a user has filled required fields in correct format before submitting a form.
2. **Dynamic Content Updates:** Displaying a notification count that updates in real-time.
3. **Interactive UI Components:** Creating sliders, modals, or dropdown menus.

Example Interaction: When using an online shopping website, the cart count updates instantly after adding an item.

Part 3: Discussion Questions

1. **Why is browser scripting important for enhancing user experience on the web?**

- It allows for dynamic updates, reducing page reloads and improving responsiveness
- Enables real-time feedback for user actions, such as form validation
- Supports interactive elements like modals, dropdowns, and auto-suggestions
- Allows personalization to the user

2. **What are some common challenges when implementing browser scripting?**

- Cross-browser compatibility issues
- Debugging asynchronous behavior
- Managing state between different components
- Security concerns like cross-site scripting (XSS)
- DOM manipulations, might cause Erros

3. **How does browser scripting differ from server-side scripting in terms of use cases?**

- **Browser Scripting (Client-side):** Runs on the users browser, handles UI interactions, and performs lightweight tasks on users browser

- **Server-side Scripting:** Runs on a server, processes data, interacts with databases, and handles authentication

4. **Among the extension challenges in the exercises, which one was the most difficult for you to implement? Explain the specific challenges you faced and why they were difficult. Additionally, describe how you attempted to solve the problem and whether you were successful.**

- **Challenges Faced**: Initially, I found it hard to handle errors properly. The Fetch API doesn't automatically throw errors for HTTP errors (like 404 or 500), but, it resolves successfully but returns a response indicating failure. I had to carefully check if the response status was in the successful range (200–299) and handle cases where the API call itself failed (e.g., network issues).
- **Solution** : To solve this, I added a `.catch()` method to the `fetch()` call to catch network errors and a condition to check the `response.ok` status to ensure the response was successful. I also displayed a fallback error message to inform the user.