Web Storage

HyperionDev

Muhammad Zahir Junejo

Lecture – Housekeeping

- ❑ The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
  - ❑ Please review Code of Conduct (in Student Undertaking Agreement) if unsure
- ❑ No question is daft or silly - **ask them!**
- ❑ Q&A session at the end of the lesson, should you wish to ask any follow-up questions.
- ❑ Should you have any questions after the lecture, please schedule a mentor session.
- ❑ For all non-academic questions, please submit a query: www.hyperiondev.com/support

Lecture Objectives

1. Understanding the Need for Web Storage
2. Introduction to Web Storage APIs
3. Tradeoffs and Choosing the Right Storage
4. Securely Storing User Application Data

Understanding the Need for Web Storage

- ❏ Web Storage is essential for web applications to store data on the client-side, enabling:
  - ❏ Persistence: Data can be retained even after the user leaves the website or refreshes the page.
  - ❏ Efficiency: Quick access to data without the need for repeated server requests.
  - ❏ Improved User Experience: Storing user preferences, cart items, or application state.
- ❏ Challenges with Server-side Storage: Traditional server-side storage (e.g., databases) can be slow, introduce latency, and lead to bottlenecks, especially for frequently changing data.
- ❏ Benefits of Client-side Data Storage: Web storage provides faster access to data, reduces server load, and enhances application performance and responsiveness.

Hyperiondev

Introduction to Web Storage APIs

- Web Storage APIs offer client-side storage solutions for web applications. They consist of three primary components:
  - localStorage: A simple key-value store that retains data even after the browser is closed or the user navigates away from the page. Ideal for user preferences, cached data, and persistent settings.
  - sessionStorage: Similar to localStorage but limited to the duration of a page session. Useful for maintaining data temporarily across page refreshes or multiple tabs/windows.
  - IndexedDB: A more complex database system capable of storing structured data and handling larger datasets. Ideal for offline applications, data synchronization, and advanced queries.

Local Storage

- localStorage is a straightforward key-value store, making it suitable for many purposes:
  - Storing User Preferences/Settings: Themes, language preferences, layout settings, etc.
  - Caching Data: Storing frequently used data to reduce server requests.
  - User Data Persistence: Remembering user login states or form input data.

```
// Storing data in localStorage
localStorage.setItem('username', 'JohnDoe');

// Retrieving data from localStorage
const username = localStorage.getItem('username');
```

Session Storage

- ❑ sessionStorage is designed for session-based data management:
  - ❑ During User Visit: Data persists as long as the user stays on the same page or across page reloads.
  - ❑ Single Session: Data is discarded when the session ends (e.g., when the user closes the tab/window).

```
// Storing data in sessionStorage
sessionStorage.setItem('cartTotal', '100');

// Retrieving data from sessionStorage
const cartTotal = sessionStorage.getItem('cartTotal');
```

IndexedDB

- ❑ IndexedDB offers advanced data storage capabilities:
  - ❑ Structured Data: It supports the storage of structured data, such as records with keys and values.
  - ❑ Large Datasets: Ideal for handling substantial amounts of data, such as caching web content for offline use or managing extensive databases.
  - ❑ Transactions: Supports transactional operations, ensuring data consistency.

```
// Opening an IndexedDB database
const request = indexedDB.open('myDatabase');

// Creating an object store
request.onupgradeneeded = function(event) {
    const db = event.target.result;
    const objectStore = db.createObjectStore('customers', { keyPath: 'id' });
};
```

Hyperiondev

Choosing the Right Storage

- ❑ Choosing the Right Storage Mechanism: The choice between localStorage, sessionStorage, and IndexedDB depends on various factors, including:
  - ❑ Data Size: For small amounts of data, localStorage or sessionStorage may suffice. For large datasets, consider IndexedDB.
  - ❑ Persistence: Decide whether data should persist beyond a session or tab.
  - ❑ Complexity: Simplicity vs. advanced data manipulation requirements.
  - ❑ Synchronization: Consider the need for synchronization with server data.
- ❑ Trade-Offs: Evaluate the tradeoffs in terms of storage capacity, lifespan, and complexity to make an informed decision.

Securely Storing User Data

- ❑ Security is paramount when storing user data:
  - ❑ Encryption: Use encryption mechanisms for sensitive data to prevent unauthorized access.
  - ❑ Data Sanitization: Validate and sanitize user inputs to mitigate security vulnerabilities like SQL injection or XSS attacks.
  - ❑ Authentication: Implement proper authentication and authorization mechanisms to restrict data access to authorized users.

References

- ❑ https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
- ❑ https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage

Questions and Answers

Thank You!