JavaScript Local Storage and CRUD Review

Persistent Storage

- **Definition**: Persistent storage refers to a way of saving data in a way that it stays available even after the power is turned off or the device is restarted.

Create, Read, Update, Delete (CRUD)

- **Create**: This refers to the process of creating new data. For example, in a web app, this could be when a user adds a new post to a blog.
- **Read**: This is the operation where data is retrieved from a database. For instance, when you visit a blog post or view your profile on a website, you're performing a read operation to fetch and display data stored in the database.
- **Update**: This involves modifying existing data in the database. An example would be editing a blog post or updating your profile information.
- **Delete**: This is the operation that removes data from a database. For instance, when you delete a blog post or account, you're performing a delete operation.

HTTP Methods

- **Definition**: HTTP stands for Hypertext Transfer Protocol and it is the foundation for data communication on the web. There are HTTP methods which define the actions that can be performed on resources over the web. The common methods are GET, POST, PUT, PATCH, DELETE.
- **GET Method**: This is used to fetch data from a server.
- **POST Method**: This is used to submit data to a server which creates a new resource.
- **PUT Method**: This is used to update a resource by replacing it entirely.
- **PATCH Method**: This is used to partially update a resource.
- **DELETE Method**: This is used to remove records from a database.

localStorage and sessionStorage Properties

- **Web Storage API**: This API provides a mechanism for browsers to store key-value pairs right within the browser, allowing developers to store information that can be used across different page reloads and sessions. The two main components for the Web Storage API are the localStorage and sessionStorage properties.

- **localStorage Property**: localStorage is the part of the Web Storage API that allows data to persist even after the browser window is closed or the page is refreshed. This data remains available until it is explicitly removed by the application or the user.
- **localStorage.setItem() Method**: This method is used to store a key-value pair in localStorage.

Example Code

```
localStorage.setItem('username', 'Jessica');
```

- **localStorage.getItem() Method**: This method is used to retrieve the value of a given key from localStorage.

Example Code

```
localStorage.setItem('username', 'codingRules');


let username = localStorage.getItem('username');

console.log(username); // codingRules
```

- **localStorage.removeItem() Method**: This method is used to remove a specific item from localStorage using its key.

Example Code

```
localStorage.removeItem('username');
```

- **localStorage.clear() Method**: This method is used to clear all of the stored data in localStorage.

Example Code

```
localStorage.clear();
```

- **sessionStorage Property**: Stores data that lasts only for the current session and is cleared when the browser tab or window is closed.
- **sessionStorage.setItem() Method**: This method is used to store a key-value pair in sessionStorage.

Example Code

```
sessionStorage.setItem('cart', '3 items');
```

- **sessionStorage.getItem() Method**: This method is used to retrieve the value of a given key from sessionStorage.

Example Code

```
sessionStorage.setItem('cart', '3 items');


let cart = sessionStorage.getItem('cart');

console.log(cart); // '3 items'
```

- **sessionStorage.removeItem() Method**: This method is used to remove a specific item from sessionStorage using its key.

Example Code

```
sessionStorage.removeItem('cart');
```

- **sessionStorage.clear() Method**: This method is used to clear all data stored in sessionStorage.

Example Code

```
sessionStorage.clear();
```

Working with Cookies

- **Definition**: Cookies, also known as web cookies or browser cookies, are small pieces of data that a server sends to a user's web browser. These cookies are stored on the user's device and sent back to the server with subsequent requests. Cookies are essential in helping web applications maintain state and remember user information, which is especially important since HTTP is a stateless protocol.
- **Session Cookies**: These cookies only last for the duration of the user's session on the website. Once the user closes the browser or tab, the session cookie is deleted. These cookies are typically used for tasks like keeping a user logged in during their visit.
- **Secure Cookies**: These cookies are only sent over HTTPS, ensuring that they cannot be intercepted by an attacker in transit.
- **HttpOnly Cookies**: These cookies cannot be accessed or modified by JavaScript running in the browser, making them more secure against cross-site scripting (XSS) attacks.

- **Set-Cookie Header**: When you visit a website, the server can send a Set-Cookie header in the HTTP response. This header tells your browser to save a cookie with specific information. For example, it might store a unique ID that helps the site recognize you the next time you visit.

You can manually set a cookie in JavaScript using document.cookie:

Example Code

```
document.cookie = "organization=freeCodeCamp; expires=Fri, 31 Dec 2021 23:59:59 GMT; path=/";
```

To delete a cookie, you can set its expiration date to a time in the past.

Example Code

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 GMT; path=/";
```

Cache API

- **Definition**: Caching is the process of storing copies of files in a temporary storage location, so that they can be accessed more quickly. The Cache API is used to store network requests and responses, making web applications work more efficiently and even function offline. It is part of the broader Service Worker API and is crucial for creating Progressive Web Apps (PWAs) that can work under unreliable or slow network conditions.

The Cache API is a storage mechanism that stores Request and Response objects. When a request is made to a server, the application can store the response and later retrieve it from the cache instead of making a new network request. This reduces load times, saves bandwidth, and improves the overall user experience.

- **Cache Storage**: This is used to store key-value pairs of HTTP requests and their corresponding responses. This enables efficient retrieval of previously requested resources, reducing the need to fetch them from the network on subsequent visits and improving performance.
- **Cache-Control**: Developers can define how long a cached resource should be kept, and if it should be revalidated or served directly from cache.
- **Offline Support**: By using the Cache API, you can create offline-first web applications. For example, a PWA can serve cached assets when the user is disconnected from the network.

Negative Patterns and Client Side Storage

- **Excessive Tracking**: This refers to the practice of collecting and storing an overabundance of user data in client-side storage (such as cookies, local storage, or session storage) without clear, informed consent or a legitimate need. This often involves tracking user behavior, preferences, and interactions across multiple sites or sessions, which can infringe on user privacy.
- **Browser Fingerprinting**: A technique used to track and identify individual users based on unique characteristics of their device and browser, rather than relying on cookies or other traditional tracking methods. Unlike cookies, which are stored locally on a user's device, fingerprinting involves collecting a range of information that can be used to create a distinctive "fingerprint" of a user's browser session.
- **Setting Passwords in LocalStorage**: This might seem like a more obvious negative pattern, but setting any sensitive data like passwords in local storage posses a security risk. Local Storage is not encrypted and can be accessed easily. So you should never store any type of sensitive data in there.

IndexedDB

- **Definition**: IndexedDB is used for storing structured data in the browser. This is built into modern web browsers, allowing web apps to store and fetch JavaScript objects efficiently.

Cache/Service Workers

- **Definition**: A Service Worker is a script that runs in the background which is separate from your web page. It can intercept network requests, access the cache, and make the web app work offline. This is a key component of Progressive Web Apps.