JavaScript: creare una SPA (Single Page App) di base con le History API

In questo articolo creeremo una SPA (Single Page App) di base utilizzando le History API di JavaScript.

Questa API ci consente di aggiungere nuove voci nella cronologia del browser e gestire le transizioni di stato con eventi JavaScript. Immagina questo scenario: vuoi navigare dalla tua home page (rappresentata dal percorso `/`) a un'altra pagina (ad es. `/posts/9`) senza eseguire una nuova richiesta GET. Qui l'API History entra in gioco inserendo una nuova voce nell'oggetto `history` del browser in modo che vedrai il nuovo URL nella barra degli indirizzi del browser ma resterai sulla pagina principale della tua applicazione (ad esempio `index.html`).

Prima di continuare con la nostra implementazione, devi sapere che per farlo funzionare sul server, devi reindirizzare tutte le richieste GET al file HTML principale, escludendo le asset statiche dalla regola di reindirizzamento. Altrimenti, quando ricarichi un percorso creato con l'API History, riceverai un errore HTTP 404 perché in questo caso specifico stai effettivamente eseguendo una richiesta GET al server.

Iniziamo ora con la nostra implementazione. Vogliamo visualizzare un elenco di post del blog e anche i dettagli del singolo post su una pagina separata. La prima cosa da fare è creare una funzione che effettui una richiesta AJAX a un endpoint remoto, recuperi un array JSON contenente i nostri post e quindi salvi questo array nel web storage in modo da non dover effettuare nuove richieste ad ogni reload della pagina.

```
const getLatestPosts = () => {
         const posts = localStorage.getItem(getSessionKey());
         if (posts === null) {
             fetch('https://api.tld/v1/posts.json')
                 .then(response => response.json())
                 .then(posts => {
                     savePosts(posts);
                     insertPosts();
                 });
         } else {
             insertPosts();
         }
     };
```

Il `getSessionKey()` crea una chiave univoca nel `localStorage` ed è implementato come segue:

```
const getSessionKey = (prefix = 'posts') => {
         const hostHash = window.btoa(window.location.host);
         return `${prefix}-${hostHash}`;
};
```

Questa funzione trasforma il nome host in una stringa con codifica Base64 e restituisce una stringa con il prefisso del parametro che gli passiamo.

`savePosts()`, come suggerisce il nome, salva il nostro array di post nel web storage.

```
const savePosts = posts => {
     return localStorage.setItem(getSessionKey(), JSON.stringify(posts));
};
```

`insertPosts()` scorre l'array JSON e crea una stringa HTML contenente l'elenco dei nostri post.

```
const insertPosts = () => {
         const postsData = localStorage.getItem(getSessionKey());
         if (postsData === null) {
             return;
         }
         const posts = JSON.parse(postsData);
         const container = document.querySelector('.app-content');

         let html = '<div class="posts">';

         for(const post of posts) {
            // ... Aggiunge contenuto alla stringa
         }
         html += '</div>';
         container.innerHTML = html;
}
```

Ora diamo un'occhiata alla struttura HTML del nostro file principale:

```
<main class="app">
     <section class="container">
         <header class="app-header">
             <h1 class="app-title">Latest posts</h1>
         </header>
         <section class="app-content"></section>
     </section>
</main>
```

`.app-content` è l'elemento dove vogliamo inserire i nostri contenuti principali quando navighiamo dalla home page alla pagina del singolo post e viceversa. Dato che abbiamo l'elemento `.app-header` nella struttura iniziale del DOM, dobbiamo nasconderlo quando ci troviamo nella pagina di dettaglio.

A questo punto, dobbiamo aggiungere un gestore di eventi a tutti i collegamenti nel nostro elenco che puntano alla pagina dei dettagli. Questo gestore si occuperà di inserire una nuova voce nella cronologia del browser, recuperando il singolo post dal nostro array e visualizzandolo nella pagina.

```
const getPost = url => {
         const postsData = localStorage.getItem(getSessionKey());
         if (postsData === null) {
             return;
         }
         const posts = JSON.parse(postsData);
         const postid = url.split('/')[1];
         const post = posts.find(post => post.id === postId);
         return post ? post : null;
};

const displayPost = post => {
         document.title = post.title;
        document.querySelector('.app-header').classList.add('hide');
        const container = document.querySelector('.app-content');
        container.innerHTML = `Contenuto HTML qui`;
};

const navigateTo = url => {
         window.history.pushState(null, null, url);
         const post = getPost(url);
         if (post !== null) {
             displayPost(post);
         }
```

```
};

const handleNavigation = event => {
        event.preventDefault();
        const url = event.target.getAttribute('href');
        navigateTo(url);
    };

    const handlePostLinks = () => {
        document.addEventListener('click', event => {
            if (event.target.matches('.read-more')) {
                handleNavigation(event);
            }
        }, false);
    };
```

Quando un utente fa clic su un collegamento con la classe `read-more`, impediamo ai browser di eseguire una nuova richiesta GET al server utilizzando il metodo `preventDefault()` di l'oggetto `event`. Invece, inseriamo una nuova voce nella cronologia del browser chiamando il metodo `pushState()` con il percorso ottenuto dall'attributo `href` di ciascun collegamento.

Ogni percorso è nel formato `/p/:id`, quindi per visualizzare un singolo post dobbiamo estrarre l'ID del post dal percorso e usarlo con `find()` per ottenere il relativo post. Il nostro gestore di navigazione cambia anche dinamicamente l'elemento `title` della pagina e nasconde l'intestazione quando stiamo visualizzando un singolo post.

Ora è il momento di gestire il pulsante Indietro del browser:

```
const getBasePostURL = url => {
        return url.replace(`${window.location.protocol}//${window.location.host}`, '');
    };

    const togglePageDisplay = () => {
        const url = getBasePostURL(window.location.href);
        const post = getPost(url);
        if (post !== null) {
            displayPost(post);
        } else {
            document.querySelector('.app-header').classList.remove('hide');
            insertPosts();
        }
    };

    const handleHistoryNavigation = () => {
        window.addEventListener('popstate', () => {
            togglePageDisplay();
        }, false);
    };
```

Stiamo utilizzando l'evento `popstate` dell'API History per gestire il pulsante Indietro. Ottenendo l'URL corrente dall'oggetto `location`, possiamo controllare se c'è un singolo post da visualizzare o il nostro elenco di post.

L'ultima cosa da fare è gestire il ricaricamento della pagina sul singolo post aggiungendo la nostra ultima funzione all'elenco delle azioni da eseguire al caricamento del DOM.

```
document.addEventListener('DOMContentLoaded', () => {
        handleHistoryNavigation();
        handlePostLinks();
      getLatestPosts();
```

```
    togglePageDisplay();
  }, false);
```

Demo

JavaScript: History App

Applicazioni Correlate

- ### JavaScript Password Mask

  Un esempio in JavaScript di mascheramento di una password con l'aggiunta della funzionalità di copia negli appunti.
  DockerDocker ComposeJavaScript

- **Python Placeholder Image**

  Applicazione sviluppata in Python con Flask per la creazione di immagini segnaposto.
  DockerDocker ComposePythonFlaskJavaScript

- **Go Placeholder Image**

  Applicazione in Go per la creazione di immagini segnaposto.
  DockerDocker ComposeGoJavaScript