Tip 46: Maintain State Over Time with LocalStorage

In this tip, you'll learn how to save user data with localStorage.

We'll cover the following

Preserving user data

Users love to personalize applications. Conversely, they hate entering the same data every time they visit an app or a page. When you're working with front-end JavaScript, you're in a bit of a bind. How do you preserve user data with minimal interference?

An obvious solution is to create a *login*. The problem is that many users will abandon a site if they're forced to log in. A better, albeit imperfect, solution is to *store data locally*. When you save data locally, you can preserve information on a particular browser on a particular device.

Of course, everyone uses multiple devices. So saving data to a browser won't help a user who works across multiple devices. Still, it's far less intrusive than demanding a user make yet another account.

Using `localStorge` to store user data

You can easily save user information with `localStorage`. `localStorage` is like a tiny *database* that exists only in your browser. You can *add* and *retrieve* information from it, but it isn't accessible by JavaScript in the browser.

Think back to your *pet adoption* site from Tip 13, Update Key-Value Data Clearly with Maps. You set up a series of filters to show only relevant pets. Because pet owners tend to prefer certain types of animals—lovers of labradors probably won't look for tiny dogs—you could do them a favor if you *save* their searches between sessions.

Start by saving a `breed` preference. To save a `breed`, you just need to set the value on the `localStorage` object using the `setItem()` method. You pass the *key* as the *first* argument and the *value* as the *second*. The syntax should look familiar. It’s nearly identical to the method for setting data on a `map`.

JavaScript

```
function saveBreed(breed) {
    localStorage.setItem('breed', breed);
}
saveBreed('labrador');
console.log(localStorage.getItem('breed'));
```

Console

Clear

```
labrador
```

When the user leaves and then returns to the page later, you can pull out the data with a similar command.

JavaScript

```
function getSavedBreed() {
  return localStorage.getItem('breed');
}
console.log(getSavedBreed());
```

Console

Clear

```
labrador
```

And if you want to remove an item, you can do that, too.

JavaScript

```
function saveBreed(breed) {
    localStorage.setItem('breed', breed);
}

function removeBreed() {
  return localStorage.removeItem('breed');
}

saveBreed('labrador');
console.log(localStorage.getItem('breed'));
removeBreed();
console.log(localStorage.getItem('breed'));
```

Console

Clear

```
labrador
Cannot read property 'target' of null
```

Now think about why this is so powerful. You can save user data without requiring any extra effort from the user. That means that when they return to the page or even refresh the page, you can set the application exactly as they left it.

For example, when you initialize your filters, you can add in the `breed` information from `localStorage` if it exists.

JavaScript

```
function saveBreed(breed) {
    localStorage.setItem('breed', breed);
}

function getSavedBreed() {
  return localStorage.getItem('breed');
}

function applyBreedPreference(filters){
  const breed = getSavedBreed();
  if (breed) {
    filters.set('breed', breed);
  }
  return filters;
}
```

```
const defaults = new Map();
saveBreed('labrador');

const filters = applyBreedPreference(defaults);
console.log(filters.get('breed'));
```

Console

Clear

```
labrador
```

Like any object, you can have as many keys as you want. If you wanted to save all your user's filters, you could save each item individually, but it's much easier to save the whole group. It's already structured data, so why spend time taking it apart?

Using `localStorge` to save an array/object

The only downside to `localStorage` is that your value *must* be a string. You can't save an *array or an object* in `localStorage`. Fortunately, the fix is simple. Just use `JSON.stringify()` to convert your data to a string and `JSON.parse` to convert it back to a JavaScript object.

If you wanted to save all of your user's search, you can convert all the filters to a *string*. Remember that because you were using a `map`, you'll need to spread it into an *array* first.

JavaScript

```
function savePreferences(filters) {
  const filterString = JSON.stringify([...filters]);
  localStorage.setItem('preferences', filterString);
}
const filters = new Map()
  .set('color', 'black');
savePreferences(filters);
console.log(localStorage.getItem('preferences'));
```

Console

Clear

```
[["color","black"]]
```

When you want to use it, you'll just need to pull the data from `localStorage` and convert it back into a `map`. Of course, if you're saving objects or arrays, all you need to do is parse the string.

JavaScript

```
function retrievePreferences() {
  const preferences = JSON.parse(localStorage.getItem('preferences'));
  return new Map(preferences);
}
localStorage.setItem('preferences', '[["color","black"]]');
const prefences = retrievePreferences();
console.log(prefences.get('color'));
```

Console

Clear

```
black
```

And, on occasion, you may just want to get back to a clean slate. In that case, you can remove all key-values with `clear()`.

JavaScript

```
function retrievePreferences() {
  const preferences = JSON.parse(localStorage.getItem('preferences'));
  return new Map(preferences);
}

function clearPreferences() {
  localStorage.clear();
}

localStorage.setItem('preferences', '[["color","black"]]');
clearPreferences();
const prefences = retrievePreferences();
console.log(prefences.get('color'));
```

Console

Clear

```
undefined
```

`localStorage` is one of those tools that’s simple but incredibly powerful. It will make you users happy and it’s simple to use. The data won’t persist across devices, but the benefit of avoiding a login far outweighs this downside.

In addition, you can also temporarily save data with sessionStorage. The usage is identical except `sessionStorage` doesn’t persist after a tab is closed. This is a great tool when you have a project that mixes server-side rendering and clientside functionality. You can save preferences between page refreshes while also ensuring that the user will have fresh state when they return.

You now have the tools to make fully integrated single page applications. Between locally saved information and API access, you only need servers to render the page once. `fetch()` and `localStorage` are incredibly simple, but they open the door to limitless opportunities to create powerful software in the browser.

In the next chapter, you’ll take a step back and look at how to architect and organize your code when building applications longer than a few lines of code.