JS Forms

Learning Objectives

- knowing the default behavior of form submit
  - understanding why to prevent this behavior with `.preventDefault()`
- knowing how to listen to submit events: the `event` object and its `target` property
- reading input values:
  - `event.target.elements`
  - `FormData`
  - the role of `name` attributes for form fields

Understanding the Default Behavior of Form Submit

If you click the submit button of a form, it triggers the following default behavior (without writing *any* JavaScript):

- The form sends a GET request with names and their values as prop inside an URL like `/?firstName=value1&lastName=value2&...`.
- The page is reloaded and thus the data is lost for us.

None of that is useful, if we want to do something with the submitted data in our frontend code. You can prevent this behavior with a method called `.preventDefault()`.

Listening to the `submit` event and preventing the Default Behavior

In order to prevent this behavior of the `submit` event, you need to

- receive the event object as an argument of the event listener arrow function
- call `event.preventDefault()`

```js
const form = document.querySelector('[data-js="form"]');

form.addEventListener("submit", (event) => {
  event.preventDefault();
});
```

By calling `event.preventDefault()` the browser will not perform a GET request that would cause the page to reload on submit.

The `event` Object and `event.target`

The `event` object is created whenever an event is triggered. You can accept it as the first parameter in the callback function and thus access it inside the function body (e.g. via `event.preventDefault()`).

For now, the most important method of the `event` object is `.preventDefault()`.

`event.target` is a reference to the element to which the event originated from - in this case - the form.

```
form.addEventListener("submit", (event) => {
  event.preventDefault();

  console.log(event.target);
});
// Output:
// <form data-js="form">
//      <fieldset>...</fieldset>
//      ...
//      <button type="submit">Submit</button>
//  </form>
```

Accessing Interactive Fields: `event.target.elements` and the `name` Attribute

While `event.target` represents the entire form, `event.target.elements` is a collection of all form elements (form fields, field sets and buttons).

You get access to a specific form field via its `name` attribute and dot notation:

```
form.addEventListener("submit", (event) => {
  event.preventDefault();

  const formElements = event.target.elements;

  console.log(formElements.firstName);
  console.log(formElements.firstName.value);
});
```

Note that

- `event.target.elements` is stored in the variable `formElements` for better readability,
- `firstName` is the string value of the corresponding `name` attribute, as in `<input name="firstName"/>`, and
- `firstName.value` returns the user input for the field with `name="firstName"`.

Using Input Values

You can access all input values of the form by using `FormData()`. This constructor uses `event.target` and can be transformed into a usable object afterwards:

```
form.addEventListener("submit", (event) => {
  event.preventDefault();
```

```
    const formData = new FormData(event.target);
    const data = Object.fromEntries(formData);

    console.log(data);
});
```

This is very useful to easily access the input data of an entire form.

> 💡 Despite the fact that using `FormData` is much less verbose, `event.target.elements` is very useful if you want to access single form field. (Spoiler alert: In case you want to focus a specific field after resetting the form, for example.)

Exception: Reading Values from Checkboxes

Checkboxes have two states: checked ("true") and not checked ("false"). In contrast to other input types, the `value` attribute does not reflect this change, but is only used as an identifier for the checkbox.

You can access the checkbox's state via the `.checked` property instead.

Imagine the following checkbox

```
<input type="checkbox" name="colorBlue" value="blue" />
```

and its corresponding JavaScript:

```
console.log(formElements.colorBlue.checked); // output: true or false
console.log(formElements.colorBlue.value); // output (always): blue
```

Resources

- Event interface