

<WA1/>

2020

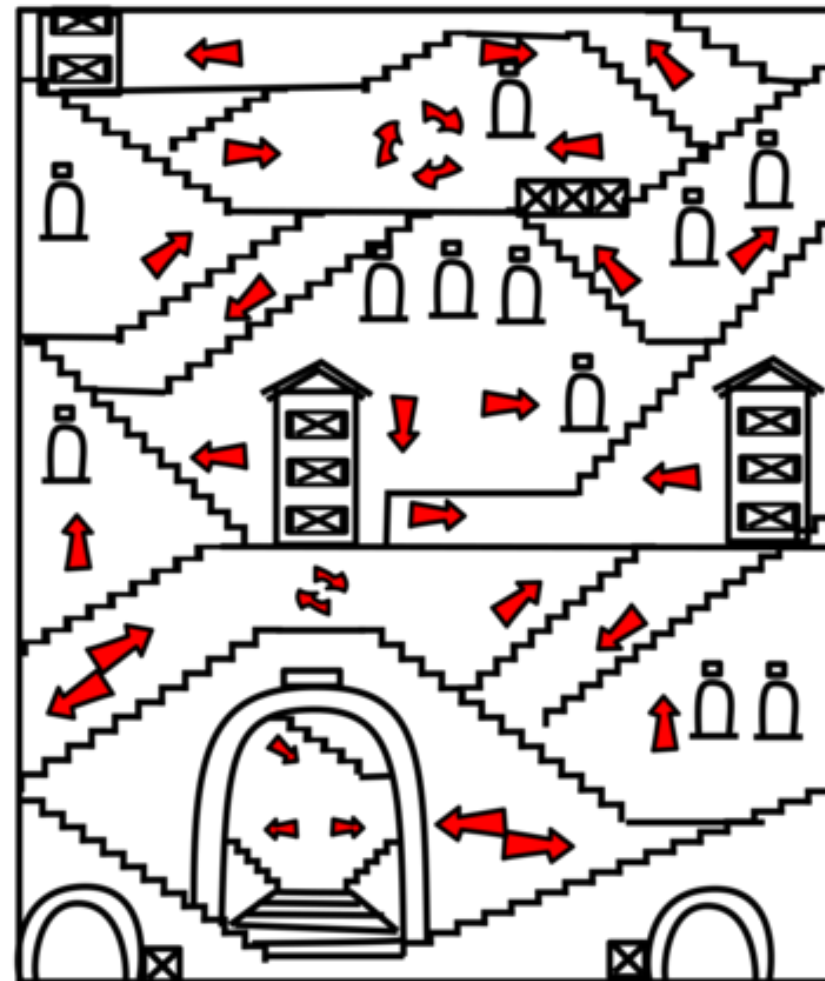
Forms

Handling user input

Enrico Masala

Fulvio Corno

Luigi De Russis



Reference: laissez-passer A38 (Asterix)

Goal

- Understanding form handling in web applications
- Knowing the most common (HTML5) form controls
- Client-side form validation
- Handling form events



Mozilla Developer Network:
Web forms — Collecting data from users
<https://developer.mozilla.org/en-US/docs/Learn/Forms>

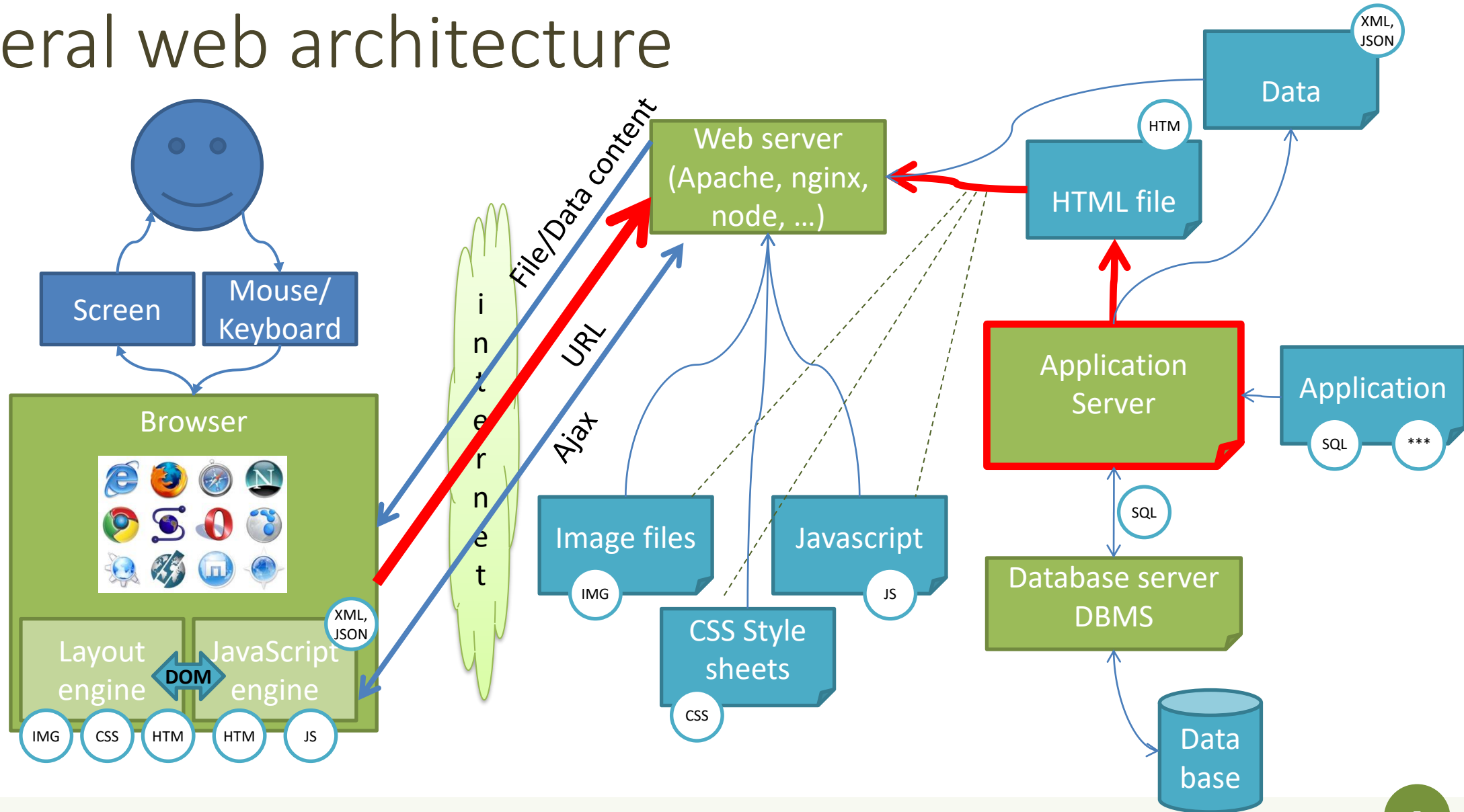
Handling user input

HANDLING FORMS

Traditional web application cycle

- Server provides the first HTML page, with forms
 - HTML forms present since HTML 2 spec (1995)
- User **inputs data** via form controls (input boxes, checkboxes, etc.)
- User submits the form data via a special **Submit** button
 - Data is encoded in a standard way and sent, via HTTP, using the GET method or the POST method
- The server application processes the data, and responds with a **new HTML page** that is parsed by the browser and presented to the user

General web architecture



Modern web application interaction

- Server provides the first HTML page
- The page contains HTML forms which allows user input and interaction
- JavaScript in the browser listens for change events in form controls, and reacts appropriately
 - Modify page content (via DOM) depending on user input, validate content, asynchronously request additional data from server (AJAX), etc.
 - Eventually and optionally, send data to the server
 - Asynchronously, and remain **on the same page**
 - Synchronously, and **reload** the page (as in traditional applications)



Mozilla Developer Network:
Web forms — Collecting data from users
<https://developer.mozilla.org/en-US/docs/Learn/Forms>

Handling user input

FORM CONTROLS

Form declaration

- `<form>` tag
- Specifies URL to be used for submission (attribute `action`)
- Specifies HTTP method (attribute `method`, default GET)

```
...  
<form action="/new-task" method="POST" id="userdata">  
    ...normal HTML content...  
        and  
    ...FORM Controls...  
</form>  
...
```


Form controls

- A set of HTML elements allowing different types of user input/interaction. Each element should be uniquely identified by the value of the name attribute
- Several control categories
 - Input
 - Selection
 - Button
- Support elements
 - Label
 - Datalist

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element#Forms>

Input control

- `<input>` tag
- Text input example
- The `value` attribute will hold user-provided text

```
...  
<input type="text" name="firstname" placeholder="Your username"></input>  
...
```

Locating a FORM in the DOM

- `document.forms` is a collection of all forms in the page
`const myForm = document.forms['form ID']`
- The form node has an **elements** property, that collects all data-containing inner elements
`const myElement = myForm.elements['element ID']`

<https://developer.mozilla.org/en-US/docs/Web/API/HTMLFormElement>

Input control (1)



- type attribute
 - Button
 - Checkbox
 - Color
 - Date
 - Email
 - File
 - Hidden
 - Month
 - Number
 - Password

Type	Description	Basic Examples	Spec
button	A push button with no default behavior displaying the value of the <code>value</code> attribute, empty by default.	<input type="button"/>	
checkbox	A check box allowing single values to be selected/deselected.	<input type="checkbox"/>	
color	A control for specifying a color; opening a color picker when active in supporting browsers.	<input type="color"/>	HTML5
date	A control for entering a date (year, month, and day, with no time). Opens a date picker or numeric wheels for year, month, day when active in supporting browsers.	<input type="date"/>	HTML5
datetime-local	A control for entering a date and time, with no time zone. Opens a date picker or numeric wheels for date- and time-components when active in supporting browsers.	<input type="datetime-local"/>	HTML5
email	A field for editing an email address. Looks like a <code>text</code> input, but has validation parameters and relevant keyboard in supporting browsers and devices with dynamic keyboards.	<input type="email"/>	HTML5
file	A control that lets the user select a file. Use the <code>accept</code> attribute to define the types of files that the control can select.	<input type="file"/>	
hidden	A control that is not displayed but whose value is submitted to the server. There is an example in the next column, but it's hidden!	<input type="hidden"/>	
image	A graphical <code>submit</code> button. Displays an image defined by the <code>src</code> attribute. The <code>alt</code> attribute displays if the image <code>src</code> is missing.	<input type="image"/>	
month	A control for entering a month and year, with no time zone.	<input type="month"/>	HTML5
number	A control for entering a number. Displays a spinner and adds default validation when supported. Displays a numeric keypad in some devices with dynamic keypads.	<input type="number"/>	HTML5
password	A single-line text field whose value is obscured. Will alert user if site is not secure.	<input type="password"/>	

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input>

Input control (2)

- type attribute
 - Radio button
 - Range
 - Submit/Reset button
 - Search
 - Tel
 - Text
 - Url
 - Week

radio	A radio button, allowing a single value to be selected out of multiple choices with the same <code>name</code> value.	<input type="radio"/>	
range	A control for entering a number whose exact value is not important. Displays as a range widget defaulting to the middle value. Used in conjunction <code>htmlattrdefmin</code> and <code>htmlattrdefmax</code> to define the range of acceptable values.	<input type="range"/>	HTML5
reset	A button that resets the contents of the form to default values. Not recommended.	<input type="reset" value="Reset"/>	
search	A single-line text field for entering search strings. Line-breaks are automatically removed from the input value. May include a delete icon in supporting browsers that can be used to clear the field. Displays a search icon instead of enter key on some devices with dynamic keypads.	<input type="search"/>	HTML5
submit	A button that submits the form.	<input type="submit" value="Submit"/>	
tel	A control for entering a telephone number. Displays a telephone keypad in some devices with dynamic keypads.	<input type="tel"/>	HTML5
text	The default value. A single-line text field. Line-breaks are automatically removed from the input value.	<input type="text"/>	
time	A control for entering a time value with no time zone.	<input type="time" value="--:--"/>	HTML5
url	A field for entering a URL. Looks like a text input, but has validation parameters and relevant keyboard in supporting browsers and devices with dynamic keyboards.	<input type="url"/>	HTML5
week	A control for entering a date consisting of a week-year number and a week number with no time zone.	<input type="week" value="Week --, ----"/>	HTML5
Obsolete values			
datetime	  A control for entering a date and time (hour, minute, second, and fraction of a second) based on UTC time zone.	<input type="datetime"/>	HTML5

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input>

Input control: commonly used attributes

Attribute	Meaning
checked	radio/checkbox is selected
disabled	control is disabled
readonly	value cannot be edited
required	need a valid input to allow form submission
size	the size of the control (pixels or characters)
value	the value inserted by the user
autocomplete	hint for form autofill feature of the browser

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Attributes>

Input control: other attributes

- Depends on the control

```
<input type="number" name="age" placeholder="Your age" min="18" max="110" />
```

```
<input type="text" name="username" pattern="[a-zA-Z]{8}" />
```

```
<input type="file" name="docs" accept=".jpg, .jpeg, .png" />
```

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Attributes>


Label tag

- The HTML `<label>` element represents a caption for an item in a user interface. Associated with `for` attribute and `id` on input
- Important for accessibility purposes (e.g. screenreader etc.), clicking the label activates the control (larger activation area e.g. in touch screens)

```
<div class="preference">
  <label for="cheese">Do you like cheese?</label>
  <input type="checkbox" name="cheese" id="cheese">
</div>
<div class="preference">
  <label for="peas">Do you like peas?</label>
  <input type="checkbox" name="peas" id="peas">
</div>
```

Do you like cheese?

Do you like peas?



<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/label>

JS Properties for input elements

- All HTML attributes are available through the DOM, in particular:
 - `value` (on text inputs): Returns / Sets the current value of the control
 - `checked` (on checkbox or radio): Returns / Sets the current state of the element
 - `validity`: Returns the element's current validity state

<https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement>

Other form controls

- `<textarea>`:
A multi-line text field

Default

Focus

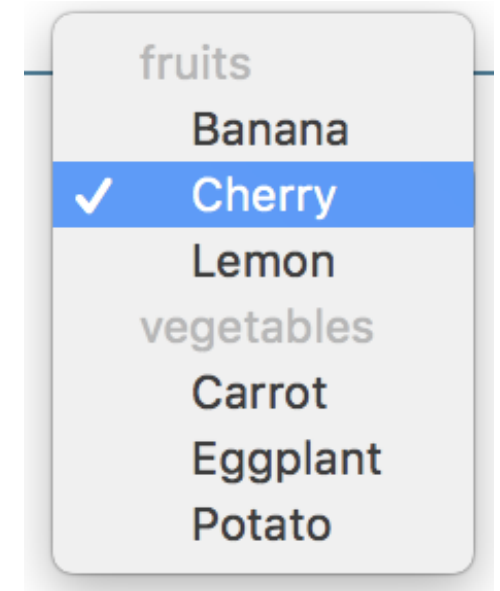
Disabled

https://developer.mozilla.org/en-US/docs/Learn/Forms/Other_form_controls

Other form controls

- Drop-down controls

```
<select id="groups" name="groups">
  <optgroup label="fruits">
    <option>Banana</option>
    <option selected>Cherry</option>
    <option>Lemon</option>
  </optgroup>
  <optgroup label="vegetables">
    <option>Carrot</option>
    <option>Eggplant</option>
    <option>Potato</option>
  </optgroup>
</select>
```



[https://developer.mozilla.org/en-US/docs/Learn/Forms/Other form controls](https://developer.mozilla.org/en-US/docs/Learn/Forms/Other_form_controls)

Button control

- `<button>` tag
- Three types of buttons
 - Submit: submits the form to the server
 - Reset: reset the content of the form to the initial value
 - Button: just a button, a behavior needs to be specified by JavaScript

```
...  
<button type="submit" value="Send data" />  
...
```

Button vs input type=button

- More flexible, can have content (markup, images, etc.)

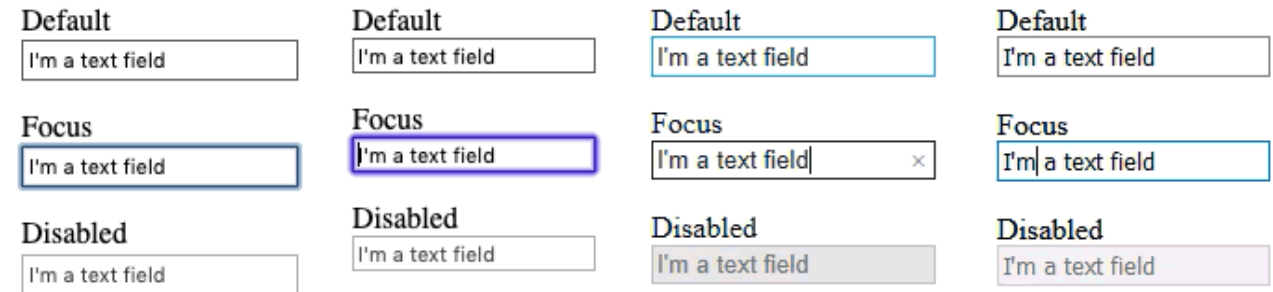
```
...  
<button class="favorite styled"  
        type="button">  
    Add to favorites  
</button>  
...  
<button name="favorite">  
    <svg aria-hidden="true" viewBox="0 0 10 10"><path  
d="M7 9L5 8 3 9V6L1 4h3l1-3 1 3h3L7 6z"/></svg>  
    Add to favorites  
</button>  
...
```



<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button>

Default appearance may vary

- Solve with CSS, but
- Some problems still remain
 - See: "Styling web forms" in MDN
 - Examples of controls difficult to manage:
 - Bad: Checkboxes, ...
 - Ugly: Color, Range, File: cannot be styled via CSS



https://developer.mozilla.org/en-US/docs/Learn/Forms/Styling_web_forms

The road to nicer forms

- Useful libraries (frameworks) and polyfills
 - Especially for controls difficult to handle via css
 - Rely on JavaScript
- Suggestions
 - Bootstrap
 - JQuery UI: customizable widgets (e.g., date picker)
 - Using libraries may improve accessibility

https://developer.mozilla.org/en-US/docs/Learn/Forms/Advanced_form_styling



Mozilla Developer Network:
Web forms — Form Validation

https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

Handling user input

FORM VALIDATION

What is form validation

- When entering data into a form, the browser will check to see if the data is in the correct format and with the constraints set by the application
 - Client side validation: via HTML5 and JavaScript
 - Server side validation: the application server will take care of it
- After client-side validation, data can be submitted to the server
- Why client-side validation?
 - We want to get the right data in the right format before processing the data
 - We want to protect users' data (e.g., enforcing secure passwords)
 - We want to protect the application (however, **NEVER TRUST** client-side validation on server side)

Types of client-side validation

- Built-in form validation by HTML5 input elements. Examples:
 - Email: check if the inserted value is a valid email (syntax only)
 - URL: check if it is a valid URL
 - Number: check if the text is a number
 - Attribute required: if a value is not present, form cannot be submitted
 - ...
- JavaScript validation: custom code is used to check correctness of values

Built-in form validation

- Mainly relies on element attributes such as:
 - **required**: if a value is not present, form cannot be submitted
 - **minlength** **maxlength** for text
 - **min** **max** for numerical values
 - **type**: type of data (email, url, etc.)
 - **pattern**: regular expression to be matched
- When element is valid, the `:valid` CSS pseudo-class applies, which can be used to style valid elements, otherwise `:invalid` applies

Built-in form validation styling

```
...  
<form>  
  <label for="e_addr">Email Address:<label>  
  <input type="email" id="e_addr" id="email" required  
placeholder="Enter a valid email address">  
</form>  
...
```

```
input:invalid {  
  border: 2px dashed red;  
}  
  
input:valid {  
  border: 2px solid black;  
}
```

Email Address:

Email Address:

Email Address:

JavaScript validation

- JavaScript must be used to take control over the look and feel of native error messages
- Approaches:
 - Constraint Validation API
 - **EventListeners** on some specific events

Constraint Validation API

- Properties and methods available via DOM on many form elements
- Via JavaScript they allow to check validity, customize error messages etc.
 - `HTMLButtonElement` (represents a `<button>` element)
 - `HTMLFieldSetElement` (represents a `<fieldset>` element)
 - `HTMLInputElement` (represents an `<input>` element)
 - `HTMLOutputElement` (represents an `<output>` element)
 - `HTMLSelectElement` (represents a `<select>` element)
 - `HTMLTextAreaElement` (represents a `<textarea>` element)

https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

Constraint Validation API: properties

Property/method	Function
<code>validationMessage</code>	a localized message describing the validation constraints that the control doesn't satisfy
<code>validity</code>	a <code>ValidityState</code> object, that includes sub-properties: <code>patternMismatch</code> , <code>tooLong</code> , <code>tooShort</code> , <code>rangeOverflow</code> , <code>rangeUnderflow</code> , <code>typeMismatch</code> , <code>valid</code> , <code>valueMissing</code> , ...
<code>willValidate</code>	true if the element will be validated when the form is submitted
<code>checkValidity()</code>	true if the element's value has no validity problems. If invalid, it fires an <i>invalid</i> event.
<code>setCustomValidity(<i>message</i>)</code>	Adds a custom error message to the element: the element is treated as invalid, and the specified error is displayed



Mozilla Developer Network:
Web forms — Form Validation

https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

Handling user input

FORM EVENTS

Events on input elements

Event	Meaning
input	the value of the element is changed (even a single character)
change	when something changed in the element (for text elements, it is fired only once when the element loses focus)
cut copy paste	when the user does the corresponding action
focus	when the element gains focus
blur	when the element loses focus
invalid	when the form is submitted, fires for each element which is invalid, and for the form itself

https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

Example

```
...  
<form action="/testadd" method="POST">  
  <input type="text">  
  <input type="submit">  
</form>  
...
```

```
const inputField = document.querySelector('input[type="text"]')  
  
inputField.addEventListener('input', event => {  
  console.log(`The current entered value is: ${inputField.value}`);  
})  
  
inputField.addEventListener('change', event => {  
  console.log(`The value has changed since last time: ${inputField.value}`);  
})
```

Form submission

- Can be intercepted with the `submit` event
- If required, default action can be prevented in `addEventListener` with the `preventDefault()` method
 - A new page is NOT loaded, everything is handled in the JavaScript: single page application

```
document.querySelector('form').addEventListener('submit', event => {  
    event.preventDefault();  
    console.log('submit');  
})
```

References (from MDN)

- Web forms — Collecting data from users
 - <https://developer.mozilla.org/en-US/docs/Learn/Forms>
- Basic native form controls
 - [https://developer.mozilla.org/en-US/docs/Learn/Forms/Basic native form controls](https://developer.mozilla.org/en-US/docs/Learn/Forms/Basic_native_form_controls)
- The HTML5 input types
 - [https://developer.mozilla.org/en-US/docs/Learn/Forms/HTML5 input types](https://developer.mozilla.org/en-US/docs/Learn/Forms/HTML5_input_types)
- Client-side form validation
 - [https://developer.mozilla.org/en-US/docs/Learn/Forms/Form validation](https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation)
- Constraint validation
 - [https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/Constraint validation](https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/Constraint_validation)
- Constraint validation API
 - [https://developer.mozilla.org/en-US/docs/Web/API/Constraint validation](https://developer.mozilla.org/en-US/docs/Web/API/Constraint_validation)



License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
 - **Share** — copy and redistribute the material in any medium or format
 - **Adapt** — remix, transform, and build upon the material
 - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
 - **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - **NonCommercial** — You may not use the material for [commercial purposes](#).
 - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.
 - **No additional restrictions** — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

