Web Components

Creating Custom Elements

Custom Element (A beginning)

./js/components/my-avatar/index.js

```
customElements.define('my-avatar',
 class extends HTMLElement {

  constructor () {
   super()

   this.innerHTML = `
    <div>
     <img id="avatar"
        src="js/components/my-avatar/images/default.png" />
    </div>
   `
  }
 }
)
```

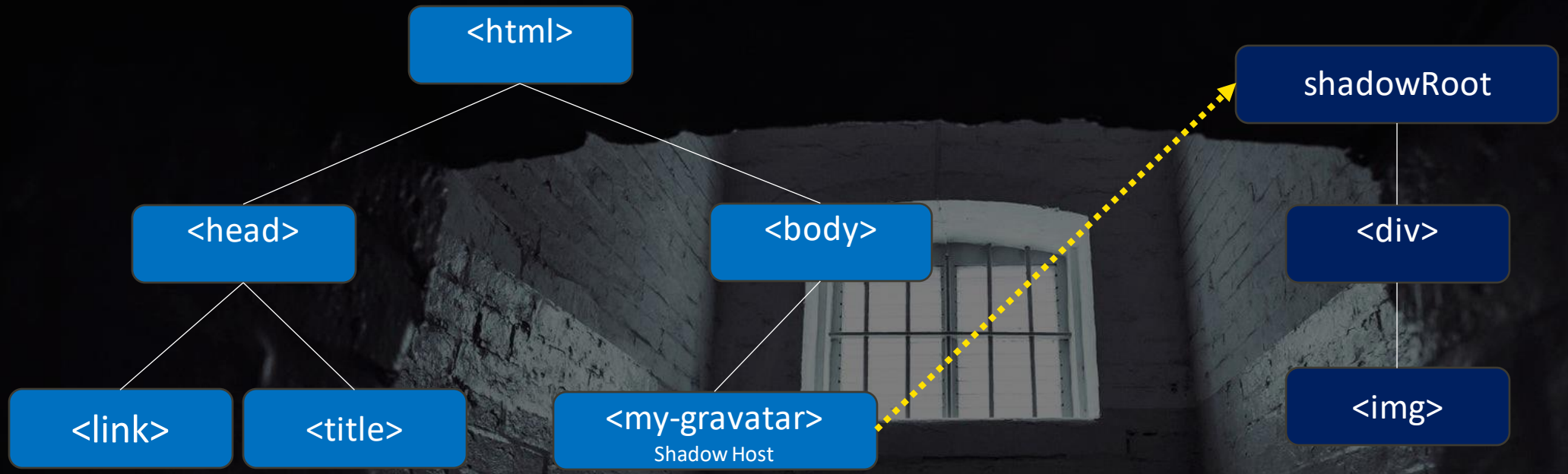
```
▶ Uncaught DOMException: Failed to construct 'CustomElement': The result must not   VM1106:1
have children
    at HTMLDocument.createElement (<anonymous>:1:1536)
    at http://localhost:8080/js/index.js:122:36
```

Shadow DOM

```
<video></video>
```

```
▼<video tabindex="-1" class="video-stream html5-main-video" controlslist="nodownload" style="width: 426px;
height: 240px; left: 0px; top: 0px;" src="blob:https://www.youtube.com/fe6f3bd1-534e-47b1-aadf-
90edf9fbbb36"> == $0
  ▼#shadow-root (user-agent)
    ▼<div pseudo="-webkit-media-controls">
      ▶<div pseudo="-webkit-media-controls-overlay-enclosure">…</div>
      ▶<div pseudo="-webkit-media-controls-enclosure">…</div>
       <div pseudo="-internal-media-controls-text-track-list" style="display: none;"></div>
      ▼<div pseudo="-internal-media-controls-overflow-menu-list" style="display: none;">
        ▶<label pseudo="-internal-media-controls-overflow-menu-list-item" style="display: none;">…</label>
        ▶<label pseudo="-internal-media-controls-overflow-menu-list-item" style="display: none;">…</label>
        ▶<label pseudo="-internal-media-controls-overflow-menu-list-item" style="display: none;">…</label>
        ▶<label pseudo="-internal-media-controls-overflow-menu-list-item" style="display: none;">…</label>
        ▶<label pseudo="-internal-media-controls-overflow-menu-list-item" style="display: none;">…</label>
        ▶<label pseudo="-internal-media-controls-overflow-menu-list-item" style="display: none;">…</label>
       </div>
     </div>
</video>
```

Shadow DOM


<html>
<head>
<body>
<link>
<title>
<my-gravatar>
Shadow Host
shadowRoot

Shadow DOM

./js/components/my-avatar/my-avatar.js

```
const template = document.createElement('template')
template.innerHTML = `
 <style>
  img { clip-path: circle(50%); }
 </style>
 <div>
  <img id="avatar" src="" />
 </div>`
customElements.define('my-avatar',
class extends HTMLElement {

 #image

 constructor () {
  super()
  // Attach a shadow DOM tree to this element and
  // append the template to the shadow root.
  this.attachShadow({ mode: 'open' })
  .appendChild(template.content.cloneNode(true))

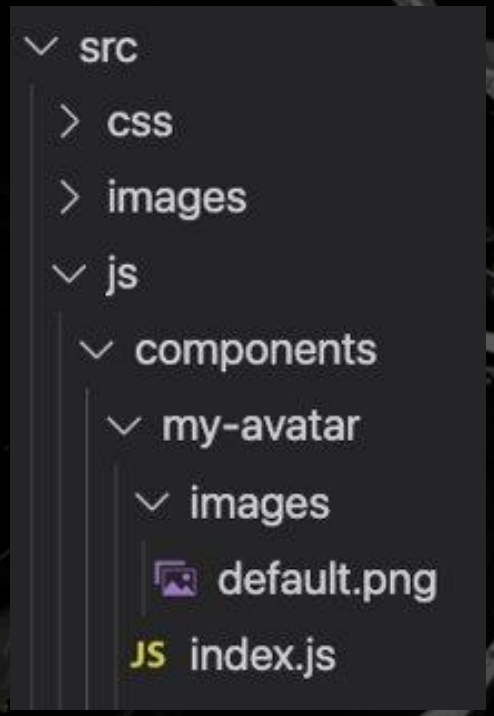
  // Get a reference to the image-element.
  this.#image = this.shadowRoot.querySelector('#avatar')
 }
}
```

import.meta.url

```
// http://localhost:3000/js/components/my-avatar/index.js
let pathToModule = import.meta.url

// http://localhost:3000/js/components/my-avatar/images/default.png
let path = new URL('./images/default.png', pathToModule)

... <img id="avatar" src="${path}" /> ...                    ./js/components/my-avatar/index.js
```



Web Components

Custom Elements
HTML Templates
Shadow DOM
ES Modules

Lifecycle callbacks (custom element reactions)

- **constructor**
  Called by the browser when an element is created or upgraded
- **attributeChangedCallback**
  Called by the browser when the value of an observed attribute change (add/remove/change) and when an element is created or upgraded
- **connectedCallback**
  Called by the browser when the element is added to the DOM (or a new DOM)
- **disconnectedCallback**
  Called by the browser when the element is removed from the DOM. Need to clean up?
- adaptedCallback
  Called by the browser when the element is moved to another document.

Lifecycle callbacks (custom element reactions)

./js/components/my-component/my-component.js

```
const template = document.createElement('template')
template.innerHTML = ``

customElements.define('my-component’,
 class extends HTMLElement {

   constructor () {
    super()
   }

   static get observedAttributes () {
    return ['attributeName’]
   }

   attributeChangedCallback (name, oldValue, newValue) { }

   connectedCallback () {}

   disconnectedCallback () {}
 }
)
```

constructor

Invoked when an element is created or upgraded

- Initialize state
- Creating a shadow DOM
- Setting up some event listeners

./js/components/my-avatar/my-avatar.js

```
constructor () {
  super()

  this.attachShadow({ mode: 'open' })
  .appendChild(template.content.cloneNode(true))

  this.#image = this.shadowRoot.querySelector('#avatar')
}
```

attributeChangeCallback

Invoked when the value of an **observed** attribute change (add/remove/change) and when an element is created or upgraded

./js/components/my-avatar/my-avatar.js

```
static get observedAttributes () {
  return ['src']
}

attributeChangedCallback (name, oldValue, newValue) {
  if (name === 'src' && newValue !== oldValue) {
    this.#image.setAttribute('src', newValue)
  }
}
```

Slot

./index.html

```
<my-gravatar email="johan.leitet@lnu.se">
  <p>
    <strong>Adding</strong> some content to our element.
  </p>
</my-gravatar>
```

./js/components/my-avatar/index.js

```
template.innerHTML = `
<div>
  <img class="avatar" id="gravatar" src="" />
  <slot class="info"></slot>
</div>`
```

```
▼<my-gravatar email="johan.leitet@lnu.se">
  ▼#shadow-root (open)
    ▶<style>…</style>
    ▼<div>
       <img class="avatar" id="gravatar" src=
      ▼<slot class="info">
         ↪ <p> reveal
       </slot>
     </div>
  ▶<p>…</p>
```

Named Slots

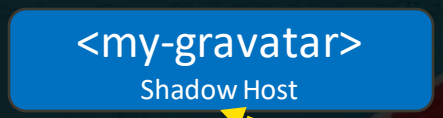
./js/components/my-gravatar/index.js

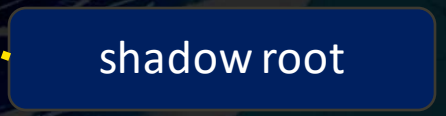
```
<div>
 <h3><slot name="name">Unknown</slot></h3>
 <p><slot name="info"></slot></p>
</div>
```

./index.html

```
<my-gravatar email="johan.leitet@lnu.se">
 <span slot="name">Johan Leitet</span>
 <span slot="info">This is some text used to explain something</span>
</my-gravatar>
```

CSS in components :host





:host

CSS-pseudo-class to access the shadow host from within a custom elements shadow DOM

CSS in components ::part()

<my-gravatar>
Shadow Host

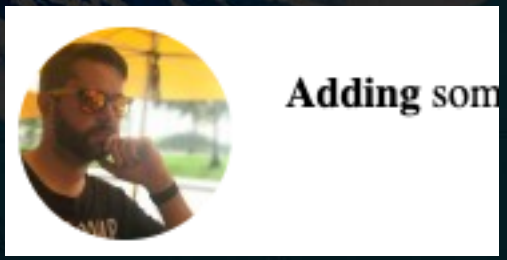
shadow root

::part()

CSS-pseudo-element used to access parts of a custom element, from outside of the custom-element.

./css/styles.css

```
my-gravatar::part(avatar) {
  clip-path: circle(50%);
}
```




```
<img part="avatar" ... />
```

./js/components/my-avatar/index.js

CSS in components ::slotted()

::slotted()

CSS-pseudo-element used to access slotted elements from within the shadow DOM.

./index.html

```
<my-gravatar-4 email="johan.leitet@lnu.se">
 <span slot="full-name">Johan Leitet</span>
 <p>This is some text used to explain something</p>
 <img src="images/big-photo.jpg" />
</my-gravatar-4>
```

./js/components/my-avatar/index.js

```
::slotted(*) {
 max-width: 100%;
 max-height: 100%;
}
```

Best practices

- ✓ Do not throw errors if possible
- ✓ Mimic Built in elements (src not source)
- ✓ Include all dependencies
- ✓ Document your Component
  - How to use
  - Attributes
  - Slots
  - Parts

Document your component

```
# &lt;bart-board&gt;

A web component is used to simulate the intro scene from Simpsons, where Bart is writing on the blackboard.

## Attributes

### `text`

A String attribute; that, if specified, contains the text that will be written out, letter by letter, on the blackboard.

Default value: `I will never ever skip the line in the task queue again.`

### `speed`

A Number indicating the speed in milliseconds, of which the letters will appear on the screen.

Default value: `50`

## Methods

### `clear()`

A method that, when called, will clear the text written on the board.

Parameters: none

Returns: Reference to self.

### `stopWriting()`

When called, it will stop the writing on the board.

Parameters: none

Returns: Reference to self.

## Events

| Event Name | Fired When |
|------------|------------|
| `filled`| The board is filled with text.

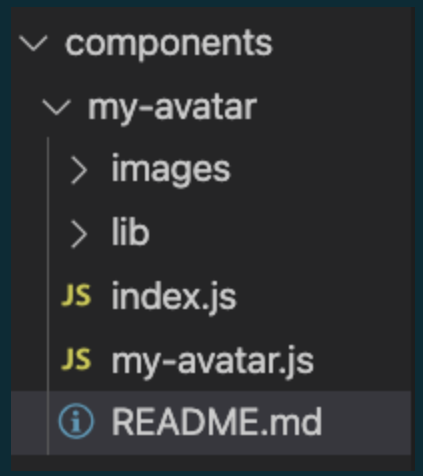
## Styling with CSS

The text (p element) is styleable using the part `text`

## Example

```html
   <bart-board text="This is the text that will be written" speed="50"></bart-board>
```

![Example of the functions of the bart-board](./.readme/example.gif)
```



README.md

<bart-board>

A web component is used to simulate the intro scene from Simpsons, where Bart is writing on the blackboard.

Attributes

`text`

A String attribute; that, if specified, contains the text that will be written out, letter by letter, on the blackboard.

Default value: `I will never ever skip the line in the task queue again.`

`speed`

A Number indicating the speed in milliseconds, of which the letters will appear on the screen.

Default value: `50`

Methods

`clear()`

A method that, when called, will clear the text written on the board.

Parameters: none

Returns: Reference to self.

`stopWriting()`

When called, it will stop the writing on the board.

Parameters: none

Returns: Reference to self.

Events

Event Name	Fired When
`filled`	The board is filled with text.

Styling with CSS

The text (p element) is styleable using the part `text`

Example

```
<bart-board text="This is the text that will be written" speed="50"></bart-board>
```