MINIAPP COMPONENTS

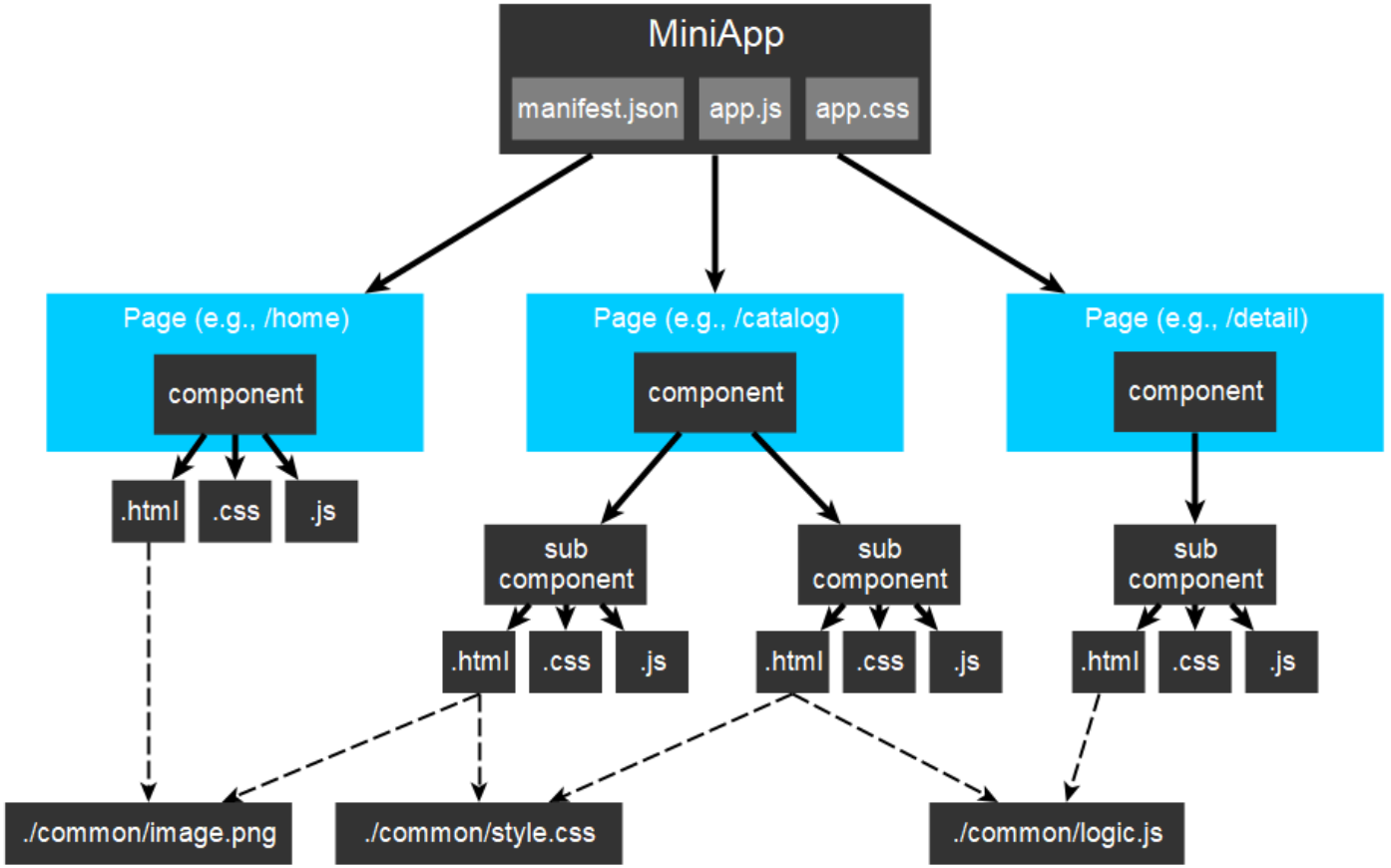
22 Jun 2022

W3C MiniApp CG Monthly Meeting

MINIAPP COMPONENTS

Background

MiniApp pages and components

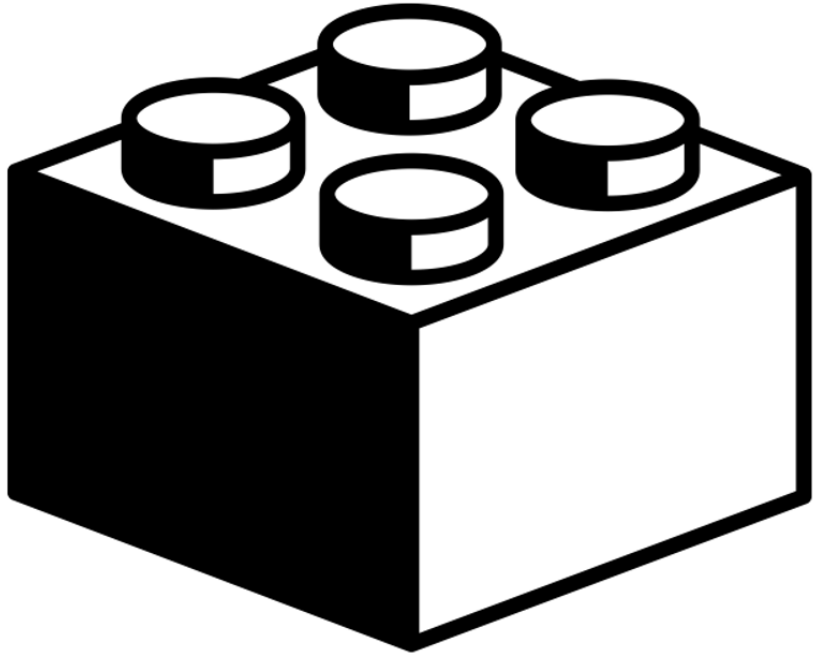


MiniApp components

- Component is an extensible and reusable high-level building block to create MiniApps
  - > Based on **HTML-like elements** (e.g., , <image>, <text>,…)
  - > Supporting **events specific** (e.g, click, swipe…)
  - > Supporting a **subset of CSS**
  - > Components support **data binding**, like text interpolation.

- **Concept similar to Web Components**
  - > Custom HTML Elements
  - > HTML modules
  - > HTML Templates
  - > Shadow DOM
  - > CSS

- Note: MiniApp uses **Virtual DOM**



lego by jon trillana from the Noun Project

MiniApp elements: common attributes

Attribute	Type	Standard equivalency	Compliant?
id	string	id attribute may be specified on any HTML element	**YES**
style	string	style attribute may be specified on any HTML element	**YES**
class	string	class attribute may be specified on any HTML element	**YES**

Difference with standard approach:

- None. All these basic attributes are similar (semantics and function) to the HTML element's attributes.

Proposed solution based on standards:

- Adoption of HTML standard element → MiniApp elements to implement the Element interface
- In this case, MiniApp elements would be compatible with the standard HTML elements

Element interface adoption Implications

- Reusing the *HTMLElement* interface, means that any MiniApp element inherits the standard attributes and methods (e.g., *addEventListener*, *parentNode*, *nextSibling*…)
- These elements implement DOM manipulation methods.
- These elements are exposed to the Window interface

```
[Exposed=Window]
interface Element : Node {
  readonly attribute DOMString? namespaceURI;
  readonly attribute DOMString? prefix;
  readonly attribute DOMString localName;
  readonly attribute DOMString tagName;

  [CEReactions] attribute DOMString id;
  [CEReactions] attribute DOMString className;
  [SameObject, PutForwards=value] readonly attribute DOMTokenList classList;
  [CEReactions, Unscopable] attribute DOMString slot;

  boolean hasAttributes();
  [SameObject] readonly attribute NamedNodeMap attributes;
  sequence<DOMString> getAttributeNames();
  DOMString? getAttribute(DOMString qualifiedName);
  DOMString? getAttributeNS(DOMString? namespace, DOMString localName);
  [CEReactions] undefined setAttribute(DOMString qualifiedName, DOMString value);
  [CEReactions] undefined setAttributeNS(DOMString? namespace, DOMString qualifiedName, DOMString value);
  [CEReactions] undefined removeAttribute(DOMString qualifiedName);
  [CEReactions] undefined removeAttributeNS(DOMString? namespace, DOMString localName);
  [CEReactions] boolean toggleAttribute(DOMString qualifiedName, optional boolean force);
  boolean hasAttribute(DOMString qualifiedName);
  boolean hasAttributeNS(DOMString? namespace, DOMString localName);

  Attr? getAttributeNode(DOMString qualifiedName);
  Attr? getAttributeNodeNS(DOMString? namespace, DOMString localName);
  [CEReactions] Attr? setAttributeNode(Attr attr);
  [CEReactions] Attr? setAttributeNodeNS(Attr attr);
  [CEReactions] Attr removeAttributeNode(Attr attr);

  ShadowRoot attachShadow(ShadowRootInit init);
  readonly attribute ShadowRoot? shadowRoot;

  Element? closest(DOMString selectors);
  boolean matches(DOMString selectors);
  boolean webkitMatchesSelector(DOMString selectors); // legacy alias of .matches

  HTMLCollection getElementsByTagName(DOMString qualifiedName);
  HTMLCollection getElementsByTagNameNS(DOMString? namespace, DOMString localName);
  HTMLCollection getElementsByClassName(DOMString classNames);

  [CEReactions] Element? insertAdjacentElement(DOMString where, Element element); // legacy
  undefined insertAdjacentText(DOMString where, DOMString data); // legacy
};
```

MiniApp elements: common events

Attribute	Interface	Similar DOM Standard Events	Compliant?
click	BasicEvent	Click (PointerEvent)	**YES**
longpress	BasicEvent	No equivalent standard (it can be implemented as a CustomEvent) Can be implemented based on agnostic pointer events + using the Event.timestamp attribute. (Similar example)	NO
swipe	BasicEvent	No equivalent standard (it can be created as a CustomEvent) – [Similar example]	NO
touchstart	TouchEvent	pointerdown (PointerEvent) on all HTML elements	NO
touchmove	TouchEvent	pointermove (PointerEvent) on all HTML elements	NO
touchcancel	TouchEvent	pointercancel (PointerEvent) on all HTML elements	NO
touchend	TouchEvent	pointerup (PointerEvent) on all HTML elements	NO

Difference with standard approach:

- Similar standard approach using W3C DOM standard (*PointerEvents*)
- Non-standard events (*swipe, longpress*) can be implemented with existing standards.

Proposed solution based on standards:

- If MiniApp elements are based on HTML standard elements → MiniApp elements support the standard events and can use the existing equivalent.
- Non-standard events could be proposed for standardization within W3C.

MiniApp basic elements

Similar semantics

Component	Additional Attributes	Events	HTML	Similar Standard Approach / Comments
div			** **	Constraint of standard attributes and different events
list		scrollend	*** ***	Element event scroll as the standard. element.scrollHeight - Math.abs(element.scrollTop) === element.clientHeight
list-item			** **	
swiper	index, loop, vertical	change	-	loop attribute, similar to Media loop attribute. Similar to change.event.
tabs	index, vertical, disabled		-	OpenUI's issue on tabs for HTML
tab-bar	mode		-	
tab-content	scrollable		-	Standard CSS *`overflow: scroll`*
refresh	offset, type, refreshing, lasttime, friction, disabled,		-	No "pull-to-refresh" standard but it could be implemented using CSS overscroll-behavior-y
image	src, alt, disabled	complete, error	**<img>**	<img> element could be used with some changes
progress	Type (circular, lineal), percent		**<progress>**	Element with (max, value, position, labels) attributes
text			**<label>**	(generic element, without semantics, <p>?, <label>?). Similar to SVG's text.
input	type, placeholder…, headericon, disable, focusable	change	**<input>**	input element (with all types supported), using attribute disabled, no headericon.
button	type, value, icon, waiting		**<button>**	button element, attribute disabled, no *waiting*, no *icon*. (in OpenUI) *type* attribute is for styles instead of functions.
label	target, disable		**<label>**	Labelable elements: button, input, meter, output, progress, select, textarea *for* instead of *target* in the standard element.
select	disable	change	**<select>**	select element. Attribute disabled (in OpenUI)
slider	min, max, value, disable	change	**<input>**	input@type="range" (in OpenUI)
switch	checked, showtext, texton, textoff, disable	Change	-	No equivalent. Toggle switch as a checkbox? (in OpenUI)
picker	type (text, date, time, datetime, multi-text), disable		-	input element (date, time, datetime-local) and datalist element (with attribute options).
video	muted, src, autoplay, poster, controls	prepared, seeked,…	**<video>**	video element includes all the MiniApp attributes, some differences in events
canvas			**<canvas>**	Equivalent

Form elements have the 'data' and 'focusable' attribute. The latter is equivalent of tabindex (i.e., <focusable=false> === <tabindex=null>).

MiniApp basic elements: direct equivalences

Component	Equivalent HTML Element (w/ minor changes)	OpenUI Component Research	Comments
div			
image	<img>		
video	<video>		
canvas	<canvas>		
progress	<progress>		
input	<input>		
picker	<input type="date|time">		
slider	<input type="range">		
label	<label>		
button	<button>	<button> (research)	
select	<select>	<select> (proposal)	
switch		<switch> (research)	
text		Text (research)	Like <label>? <p>? but only text
tabs		Tabs (research)	
list			
list-item			
swiper (carousel)			
refresh			

MINIAPP COMPONENTS

Challenges and proposal based on standards

Challenges and proposals based on standards

Challenges	Potential Solutions
New elements, not included in the HTML specification (but somehow present in some web frameworks, like <swiper>, <tabs>, <switch>,…)	1. These elements could be defined using stand-alone Web Components, and **Custom Elements**. Following the **OpenUI CG process.**
Similar elements (elements that are semantically similar to HTML standard elements like <image> <progress> <text> <input> <video> <canvas> <slider> <button>…)	1. Extension of the existing HTML elements (using inheritance). We could define the new attributes, events or redefine the existing ones **defining new interfaces in the specification**. 2. Reuse and refine the existing HTML elements **using the OpenUI CG process**. Ideal to leverage the existing Web capabilities.
Non-standard attributes in MiniApp components (e.g., disable, focusable..)	1. Adapt and use directly the standard attributes. If new attributes are needed, to propose changes in the standards. 2. Creation of a basic, essential MiniAppElement that extends the **HTMLElement** with the specific requirements (new attributes, redefinition of existing attributes). Problem: this prevent MiniApps to reuse Web Components.
Similar events in MiniApp components that are similar to existing standard ones (e.g., touchstart, touchend, etc.)	1. Adoption of the standard version as it is (**PointerEvents**).
New events for MiniApp components (e.g., longpress, swipe…)	1. Propose new events to the existing standards if really needed. 2. Definition of new **CustomEvents** based on the **standard DOM** interfaces.

Creation of profiles for **STANDARD HTML, CSS** and **DOM** → MiniApps supported by browsers

How to implement a MiniApp component

Extension of HTMLElement

Mapping of attributes

Definition of new events

```
class MiniappElement extends HTMLElement {
  constructor() {
    super();

    // Attributes Mapping (e.g., "disable" === "disabled")
    if (this.hasAttribute('disable') && this.getAttribute('disable')==='true') {
      this.setAttribute('disabled', true)
    }
    if (this.hasAttribute('focusable') && this.getAttribute('focusable')==='false') {
      this.setAttribute('inert', true)
    }

    // Common Events
    // longpress event
    this.addEventListener('pointerdown', (e) => {
      this.dataset.timestamp= e.timeStamp
    })
    this.addEventListener('pointerup', (e) => {
      // Checks if a longpress happened (long = over 1s)
      if ((e.timeStamp - this.dataset.timestamp) >= 1000) {
        console.log('Triggering longpress event (over 1s)')
        e.target.dispatchEvent(new CustomEvent('longpress'))
      }
      delete this.dataset.timestamp
    })
    // Binds the new event
    this.addEventListener('longpress', function(e) {
      console.log('long pressed')
    })
  }
  connectedCallback() {
    console.log('MiniappElement connected')
  }
  disconnectedCallback() {
    console.log('MiniappElement disconnected')
  }
}
```

https://jsbin.com/qetaguy/edit?html,output

Definition of new elements

Definition of <miniapp-text> as a new element

Extension of MiniAppElement

```
class MiniappText extends MiniappElement {
  constructor() {
    super();
  }
  connectedCallback() {
    const template = document.querySelector('template#text-component');
    const cloned = document.importNode(template.content, true);
    // Create a shadow root
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.appendChild(cloned);
    console.log('MiniApp-Text connected');
  }
  disconnectedCallback() {
    console.log('MiniApp-Text disconnected');
  }
}
customElements.define('miniapp-text', MiniappText);
```

Template of the <miniapp-text>

Definition of styles

```
<template id='text-component'>
  <style>
    * {
      background-color: orange;
    }
  </style>
  <span><slot/></span>
</template>

<miniapp-text>Hello MiniApp</miniapp-text>
```

Use of the <miniapp-text> (developers only need to use this)

Output

Hello MiniApp

https://jsbin.com/qetaguy/edit?html,output

Extension of the existing elements

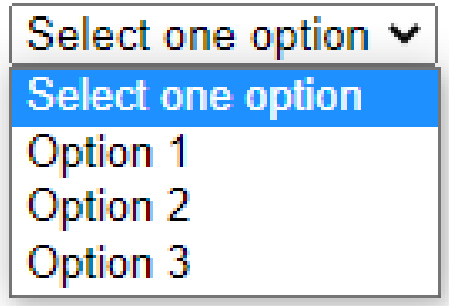
Definition of the specific behavior for the MiniApp component, with specific attributes, events, etc. Extending the standard element

```
class MiniappSelect extends HTMLSelectElement {
    constructor() {
      super();

      // Mapping to existing attributes (e.g., "disable" === "disabled")

      if (this.hasAttribute('disable') && this.getAttribute('disable')==='true') {
        this.setAttribute('disabled', true)
      }
      if (this.hasAttribute('focusable') && this.getAttribute('focusable')==='false') {
        this.setAttribute('inert', true)
      }


    }
    connectedCallback() {
      const firstOption = this.firstChild;

      // Adding an option on top of the list with the text ('select one option')
      const optionDefault = document.createElement('option');
      optionDefault.appendChild(document.createTextNode('Select one option'));
      this.insertBefore(optionDefault, firstOption);
      console.log('MiniApp-Select connected');
      // Select the first one (the new option)
      this.selectedIndex = 0;
    }
    disconnectedCallback() {
      console.log('MiniApp-Select disconnected');
    }
  }
```

```
customElements.define('miniapp-select', MiniappSelect, { extends: 'select' });
```

```
<select is='miniapp-select' focusable='false'>
  <option>Option 1</option>
  <option>Option 2</option>
  <option>Option 3</option>
</select>
```

Using the standard version of the component but indicating the profile (this select is a ‘miniapp-select’)



https://jsbin.com/qetaguy/edit?html,output

MINIAPP COMPONENTS

How to write the UI Components specification

How to write the UI Components specification
Case 1: non-standard DOM/HTML

If the MiniApp model doesn't follow the standard DOM/HTML (incompatible with existing Web standards)

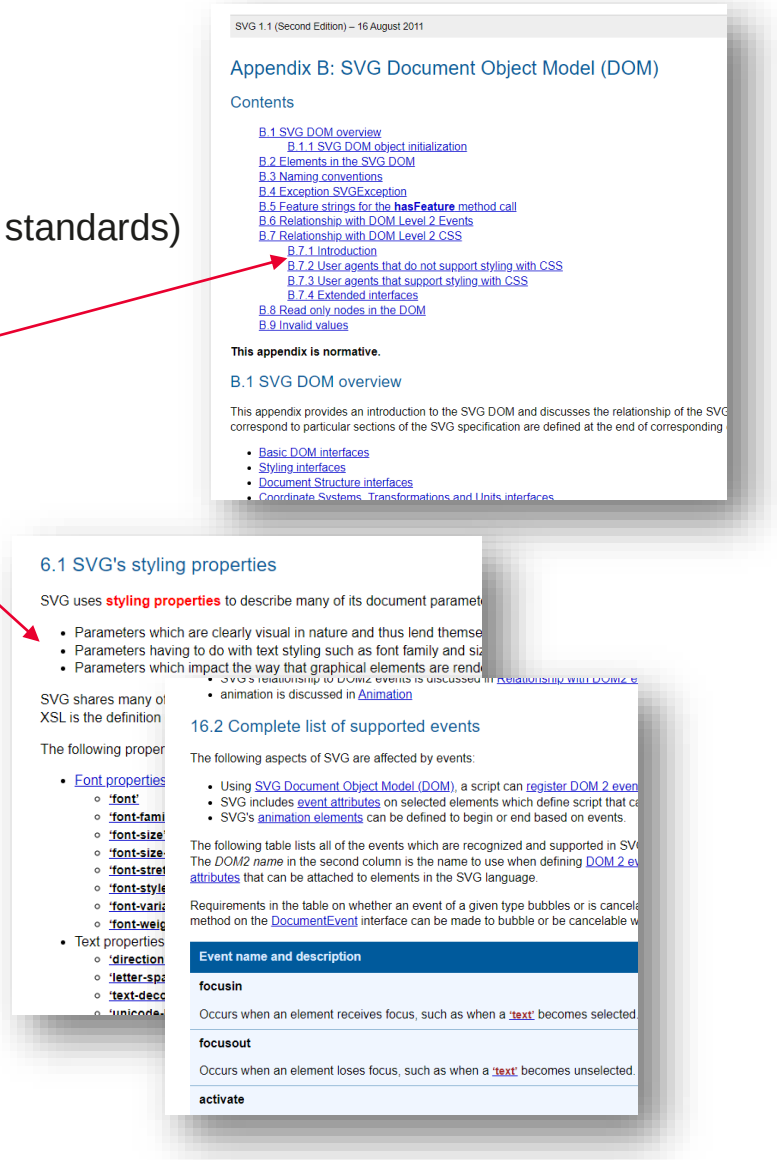
> We need to define a new model, or at least a new markup language (for instance, based on XML).

> We should indicate how to interact/manipulate the DOM for our solution (example in the SVG spec.).

> We can use some examples as references: SMIL, TTML, SVG.

> This solution would require at least the definition of XML schemas for the markup language.

> We need to indicate how to bind CSS stylesheets and events (in the SVG spec, we have examples)



How to write the UI Components specification Case 2: standard DOM/HTML

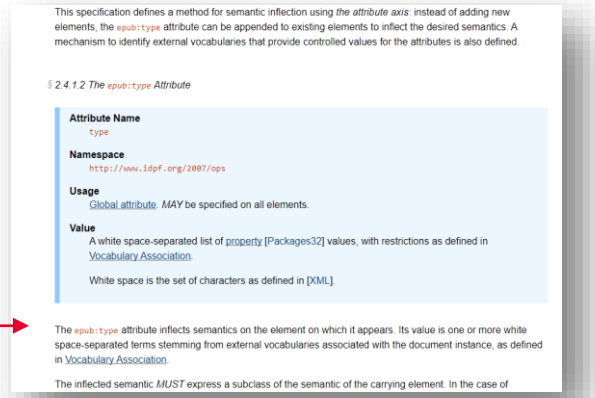
If the MiniApp model follows the standard DOM/HTML (compatible with existing Web standards)

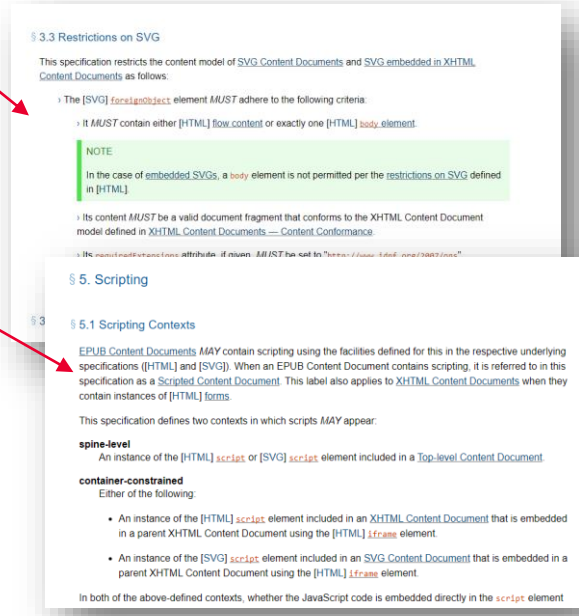
- > We could define HTML extensions (e.g., EPUB's attributes) if needed.
- > We should present what technologies (HTML, CSS) are supported.
- > We would use HTML and CSS subsets, so we need to indicate these subsets (e.g., EPUB's Deviations & Constraints or CSS).
- > We should define the scripting mechanisms supported in MiniApps (e.g., EPUB's Scripting)
- > Mechanisms to include localized strings.

First step: to organize a meeting with OpenUI CG (I am part of the CG already)

The UI Components specification could be a Group Note to define MiniApp contents, including:

- > Structure of a MiniApp page (how to bind components)
- > Overview of the basic elements (or components) → no technical definition, just high level.
- > What styles are supported (e.g., the CSS subset, how to bind external stylesheets)
- > Scripts supported (e.g., version and limitations for security)
- > Mechanisms to extend the components.





How to write the UI Components specification Case 3: Vue.js-like framework

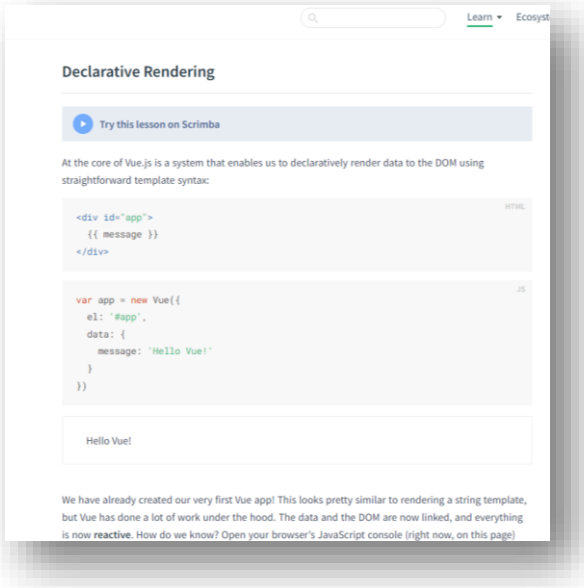
If the MiniApp model follows the existing MVVM (non-compatible with current standards)

- We could define an informative (non-standard) specification to document the architecture;
- We should include what technologies (HTML, CSS) are supported and the extensions.
- We should document the data and event binding.
- We should define the scripting mechanisms supported in MiniApps.

First step: publicly confirm that MiniApps wouldn't follow the HTML/DOM standards. No WebAPIs available

To avoid friction with other W3C groups, the UI Components specification should be an **informative Group Note** or similar to define MiniApp contents, including something like the documentation of Vue.js:

- Structure of a MiniApp page (how to bind components);
- Explain the differences with HTML.
- What styles are supported (e.g., the CSS subset, how to bind external stylesheets);
- Scripts supported (e.g., version and limitations for security);
- Data interpolation, internationalization, etc.



Thank you.

More info:
https://github.com/w3c/miniapp-components