MINIAPP UI COMPONENTS

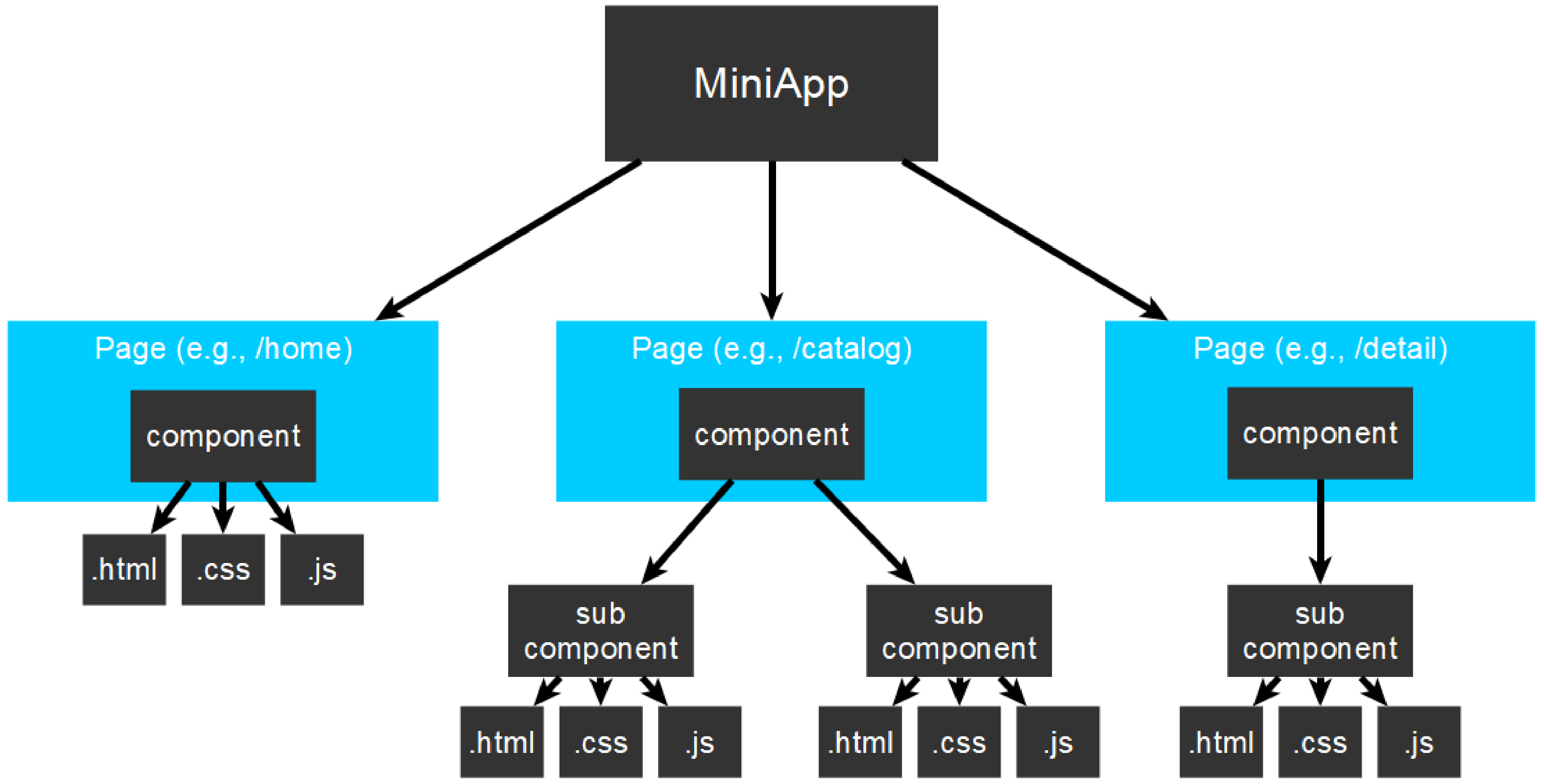
TPAC – 28 Oct 2021

W3C MiniApp CG

MINIAPP COMPONENTS

Content and structure of MiniApps

MiniApp pages and components

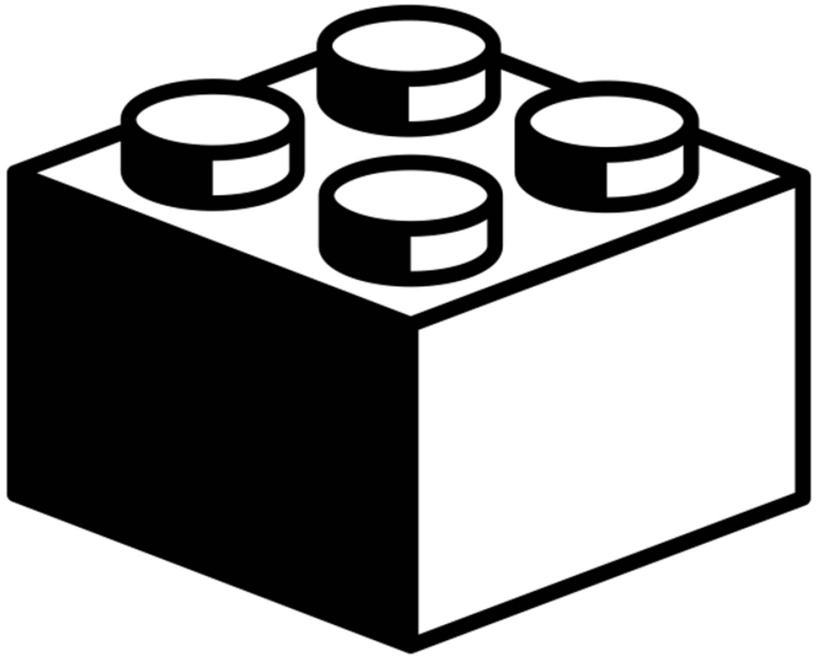


MiniApp components

- Component is an extensible and reusable high-level building block to create MiniApps
  - > Based on **HTML-like elements** (e.g., <div>, <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 essential elements: common attributes

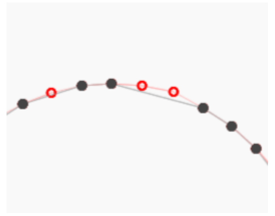
Attribute	Type	Standard equivalency	Compliant?
id	string	id attribute may be specified on any HTML element	**YES**
style	string	style attribute may be specified on any HTML element	**YES**
class	string	class attribute may be specified on any HTML element	**YES**

All these basic attributes are similar (semantics and function) to the HTML element’s attributes

MiniApp essential 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

- Most of the common events are supported by ***PointerEvent***s (W3C DOM standard)
- The ***PointerEvent*** interface includes the *getCoalescedEvents()* with a list of ***PointerEvents*** →
- Events ***longpress*** and ***swipe*** are not standard
- Controversy in the definition of these events when proposed in WebApp WG



Related:

- Pointer events https://www.w3.org/TR/pointerevents3/
  interfaces for handling hardware agnostic pointer input from devices including a mouse, pen, touchscreen, etc.
- Gestures proposal (2018):
  - https://github.com/JuntaoPeng/GestureEvents/blob/master/GestureEvents.md
  - Including Swipe, LongPress event…

Essential elements

Component	Additional Attributes	Events	HTML	Similar Standard Approach / Comments
`div`			`<div>`	Constraint of standard attributes and different events
`list`		scrollend	`<ul>`	Element event scroll as the standard. element.scrollHeight - Math.abs(element.scrollTop) === element.clientHeight
`list-item`			`<li>`	
`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, focusable, percent		`<progress>`	Element with (max, value, position, labels) attributes
`text`			`<span>`	(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>).

ALIGNMENT WITH STANDARDS

Open discussion on the challenges

Definition of MiniApp Elements (Proposal)

- To implement the new MiniApps Components as Custom Elements and *polyfill* them
- User Agents to implement compatibility with standard HTML and DOM specs.
- OpenUI CG to document and incubate new elements (e.g., *Picker*, *Tab*)

WHY?

- Aligned with Web standards and the Web Architecture
- More flexibility to developers
- Browser oriented version

CHALLENGES:

- Adaptation to the standard elements
- Render engine must support DOM manipulation, events and Web APIs
- Render engine to support HTML elements

About WebComponents: https://github.com/WICG/webcomponents

FAST project (implementation example): https://www.fast.design/

Discussion: https://github.com/w3c/miniapp-packaging/issues/2

Challenge: DOM Manipulation

Context:

- Client-side JS libraries uses DOM (and shadow DOM).
- MiniApps don't expose the DOM to the user agent (the logic layer cannot access "document.whatever")
- Assistive technologies usually rely on the DOM to “understand” the document

To discuss: **Can MiniApp expose DOM APIs to the logic layer?**

> so Web developers can use existing JS libraries to modify the DOM;
> assistive tools may also access the accessibility features of the DOM;
> MVVM framework still applicable with this solution.

Discusion: https://github.com/w3c/miniapp/issues/162

Shadow DOM: https://www.w3.org/TR/shadow-dom/

WAI-ARIA considerations

The specification should include additional information on how MiniApp user agents handle accessibility.

Some considerations:

- The DOM generated MUST include the content attribute for *role* and its WAI-ARIA *role values*, as well as the *WAI-ARIA States and Properties* in the DOM.
- To include an **accessibility tree**
  > The accessibility tree and the DOM tree are parallel structures. The accessibility tree includes the user interface objects of the user agent and the objects of the document. Accessible objects are created in the accessibility tree for every DOM element that should be exposed to an assistive technology.

Namespace recommended for attributes: http://www.w3.org/ns/wai-aria/

Accessibility tree: https://w3c.github.io/aria/#accessibility_tree

More information: https://w3c.github.io/aria/

Thank you.

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