MOBA2

MOBILE WEB: COMPONENT DRIVEN UIs

OVERVIEW

- Component Driven UIs
- Web Components
- Other Tools and Libraries
- Introduction to React.js

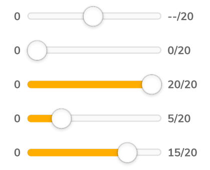
OVERVIEW

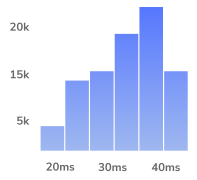
- Component Driven UIs
- Web Components
- Other Tools and Libraries
- Introduction to React.js

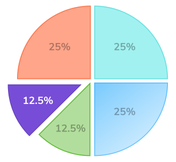
MODERN USER INTERFACES

- Modern user interfaces are complicated
- People expect compelling, personalized experiences
- Should work across devices
- More logic embedded into UIs
- Large UIs are brittle, painful to debug

COMPONENTS







WHY COMPONENTS?

- Necessary to break UIs down in a modular way
- Components enable interchangeability
- Isolate state from application business logic
- Decompose complex screens into simple components
- Each component has a well-defined API and states
- Components can be recomposed to build different UIs

WHAT ARE COMPONENTS?

- Standardized, interchangeable building blocks of UIs
- Encapsulate the appearance and function of UI pieces

COMPONENT DRIVEN DEVELOPMENT

- Build one component at a time
  Avatar, Button, Input, Tooltip
- Combine components
  Form, Header, List, Table
- Assemble pages
  Home page, Settings page, Profile page
- Integrate pages into your project
  Web app, Marketing site, Docs site

BENEFITS

- Focus development
- Increase UI coverage
- Target feedback
- Build a component library
- Parallelize development
- Test visually

TOOLS: COMPONENT EXPLORERS

- Showcase the components in various test “states”
- A state is essentially a visual test case
- Test a given component in all important states
- Workflow where you build one component at a time

COMPONENT STORY FORMAT (CSF)

- Open standard for component examples
- Based on JavaScript ES6 modules
- Simple to write component “stories”
- Doesn’t require vendor-specific libraries
- Declarative syntax

https://github.com/ComponentDriven/csf

STORYBOOK

- Frontend for building UI components and pages in isolation
- Suitable for UI development, testing, and documentation
- Mock hard-to-reach edge cases as stories
- Drop the finished UI components into your app
- Open source and free

https://storybook.js.org

STORYBOOK

Component Driven Development

COMPONENTS AND FRAMEWORKS

- Web Components
  - Stencil, Polymer, ...
- Client side UI logic and components
  - React, Vue, ...
- Presentation layer frameworks
  - Ionic, jQuery Mobile, ...
- Native Components
  - React Native, NativeScript, ...

OVERVIEW

- Component Driven UIs
- Web Components
- Other Tools and Libraries
- Introduction to React.js

WEB DEVELOPMENT

In many instances you're either copying huge chunks of HTML out of some doc and then pasting that into your app ...

A Guide to Web Components

HTML should be ...

- ... expressive enough to create complex UI widgets
- ... extensible to fill in any gaps with our own tags

This is eventually possible with Web Components

WEB COMPONENTS

- Bundle markup and styles into custom HTML elements
- Fully encapsulate all of their HTML and CSS
- Introduced by Alex Russell at Fronteers Conference 2011

EXAMPLE: IMAGE SLIDER

```
<div id="slider">
  <input type="radio" name="slider" id="slide1" selected="false" checked>
  <input type="radio" name="slider" id="slide2" selected="false"> ...
  <div id="slides">
    <div id="overflow">
      <div class="inner">
        <img src="images//rock.jpg">
        <img src="images/grooves.jpg">...
      </div>
    </div>
  </div>
  <label for="slide1"></label>
  <label for="slide2"></label>...
</div>
```

codepen.io/robdodson/pen/rCGvJ

EXAMPLE: BETTER IMAGE SLIDER

```
<img-slider>
  <img src="images/sunset.jpg" alt="a dramatic sunset">
  <img src="images/arch.jpg" alt="a rock arch">
  <img src="images/grooves.jpg" alt="some neat grooves">
  <img src="images/rock.jpg" alt="an interesting rock">
</img-slider>
```

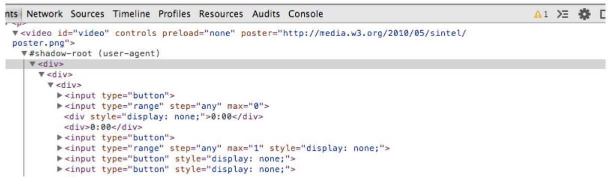
THE VIDEO ELEMENT

```
<video src="./foo.webm" controls></video>
```

- There’s a play button, a scrubber, timecodes, a volume slider
- A way to build the *video* element from these parts was needed
- Browser makers created a secret place: the *Shadow DOM*

You can activate *Show user agent shadow DOM* in the browser’s DevTools

THE VIDEO ELEMENT



```
▼<video id="video" controls preload="none" poster="http://media.w3.org/2010/05/sintel/
poster.png">
  ▼#shadow-root (user-agent)
    ▼<div>
      ▼<div>
        ▼<div>
          ▶<input type="button">
          ▶<input type="range" step="any" max="0">
           <div style="display: none;">0:00</div>
           <div>0:00</div>
          ▶<input type="button">
          ▶<input type="range" step="any" max="1" style="display: none;">
          ▶<input type="button" style="display: none;">
          ▶<input type="button" style="display: none;">
```

TEMPLATES

- The `template` element
- Not rendered on the page until it is activated using JavaScript

```
<template>
  <h1>Hello there!</h1>
  <p>This content is top secret :)</p>
</template>
```

SHADOW DOM

Select an element and call its *attachShadow* method

```
<!-- HTML -->
<div class="container"></div>
```

```
// JavaScript
var host = document.querySelector('.container')
var root = host.attachShadow({mode: 'open'})
root.innerHTML = '<p>How <em>you</em> doin?</p>'
```

SHADOW HOST AND SHADOW ROOT

- Shadow Host
  - Element that *attachShadow* is called on
  - The only piece visible in the element hierarchy
  - The place where the element is supplied with content
  - Example: the *video* element is the shadow host
- Shadow Root
  - Document fragment returned by *attachShadow*
  - It and its descendants are hidden
  - But they’re what the browser will actually render

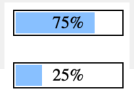
CUSTOM ELEMENT

```
class ImageSlider extends HTMLElement {
  constructor() {
    super()
    const shadowRoot = this.attachShadow({mode: 'closed'})
    shadowRoot.innerHTML = `
      <style></style>
      <div class="slider">
        ...
      </div>
    `
  }
}

customElements.define('image-slider', ImageSlider)
```

ANOTHER EXAMPLE – DEMO

```
<custom-progress-bar class="size">
<custom-progress-bar value="25">
<script>
  document.querySelector('.size').progress = 75;
</script>
```



WEB COMPONENTS SUMMARY

Based on these pieces:

- Shadow DOM
- Custom Elements
- HTML Templates
- CSS additions

https://github.com/WICG/webcomponents
https://developer.mozilla.org/en-US/docs/Web/Web_Components

BROWSER SUPPORT

- Web Comonents were introduced in 2011
- By now, Web Components should be everywhere
- Browser support: good
  caniuse.com/#search=Web%20Components
- Reason for slow progress: vendors couldn’t agree
- Web Components were a Google effort

WEB COMPONENT LIBRARIES

- Stencil: Web Component compiler
  https://stenciljs.com
- Lit (Successor of Polymer)
  https://lit.dev
- X-Tag: Mozilla's alternative
  www.x-tags.org

OVERVIEW

- Component Driven UIs
- Web Components
- Other Tools and Libraries
- Introduction to React.js

EXAMPLE (WBE RECAP)

- In order to compare various approaches
- Create a list from an array

```
/* input: */
let data = ["Maria", "Hans", "Eva", "Peter"]
```

```
<!-- DOM structure to be created: -->
<ul>
  <li>Maria</li>
  <li>Hans</li>
  <li>Eva</li>
  <li>Peter</li>
</ul>
```

DOM SCRIPTING

```
function List (data) {
  let node = document.createElement("ul")
  for (item of data) {
    let elem = document.createElement("li")
    let elemText = document.createTextNode(item)
    elem.appendChild(elemText)
    node.appendChild(elem)
  }
  return node
}
```

- Simple abstraction: a `List` component
- Based on DOM functions

DOM SCRIPTING

```
function init () {
  let app = document.querySelector(".app")
  let data = ["Maria", "Hans", "Eva", "Peter"]
  render(List(data), app)
}

function render (tree, elem) {
  while (elem.firstChild) { elem.removeChild(elem.firstChild) }
  elem.appendChild(tree)
}
```

DOM SCRIPTING ENHANCED

```
function domElt (type, attrs, ...children) {
  let node = document.createElement(type)
  if (attrs) Object.keys(attrs).forEach(key => {
    node.setAttribute(key, attrs[key])
  })
  for (let child of children) {
    if (typeof(child) instanceof HTMLElement) node.appendChild(child)
    else node.appendChild(document.createTextNode(child))
  }
  return node
}
```

DOM SCRIPTING ENHANCED

- Abstraction enables a simpler `List` component
- DOM functions hidden in function `domElt`

```
function List (data) {
  return domElt("ul", {}, ...data.map(item => domElt("li", {}, item)))
}
```

JQUERY

```
function List (data) {
  return $("<ul>").append(...data.map(item => $("<li>").text(item)))
}

function render (tree, elem) {
  while (elem.firstChild) { elem.removeChild(elem.firstChild) }
  $(elem).append(tree)
}
```

- `List` returns a jQuery object
- Minor modification to the `render` function needed

REACT.JS

```
const List = ({data}) => (
  <ul>
    { data.map(item => (<li key={item}>{item}</li>)) }
  </ul>
)
ReactDOM.render(
  ( <List data={["Maria", "Hans", "Eva", "Peter"]} /> ),
  document.getElementById('app')
)
```

- XML syntax in JavaScript: JSX
- Needs to be translated to JavaScript
- More in a moment...

VUE.JS

```
<div id="app">
  <ol>
    <li v-for="item in items">
      {{ item.text }}
    </li>
  </ol>
</div>
```

https://vuejs.org

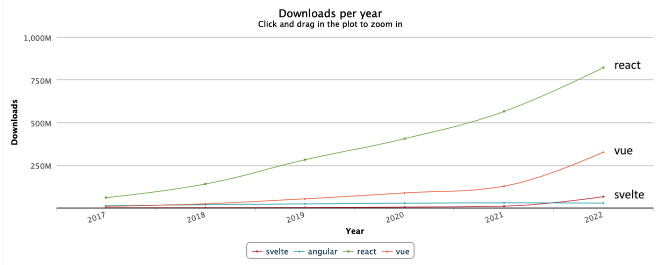
```
var app4 = new Vue({
  el: '#app',
  data: {
    items: [
      { text: 'Learn JavaScript' },
      { text: 'Learn Vue' },
      { text: 'Build something awesome' }
    ]
  }
})
```

SVELTEJS

- Framework for building UIs, like Vue or React
- Svelte is a compiler, unlike React or Vue
- No virtual DOM, code compiled to vanilla JS
- Truly reactive framework, no complex state management libraries

https://svelte.dev

NPM STATS



Total number of downloads between *2017-01-01* and *2022-12-31*:

package	**downloads**
react	2,269,072,195
vue	623,085,259
angular	133,375,554
svelte	79,180,699

COMPONENT DRIVEN DEVELOPMENT

React | Vue.js | ANGULAR | ember | SVELTE

RIOT | Storybook | chromatic | docz

bit

Backlight | stencil | anima

DESIGN AND PROTOTYPING

Sketch | Figma | Xd | invision | Framer

UXPin | ZEPLIN | Abstract | zeroheight

draftbit | Interplay. | relate | penpot

OVERVIEW

- Component Driven UIs
- Web Components
- Other Tools and Libraries
- Introduction to React.js

WHAT IS REACT?

A JavaScript library for building user interfaces

- It’s not a mega framework
- It’s not a full-stack solution

WHAT IS REACT?

A JavaScript library for building user interfaces

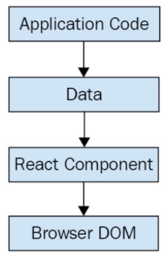
- Facebook, Instagram
- First introduced in 2013

https://reactjs.org

React wraps an imperative API with a declarative one

```
(data) => view
```

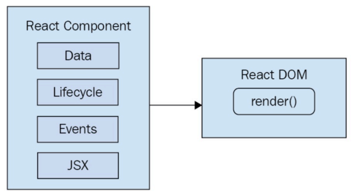
REACT IS JUST THE VIEW



- Application logic generates some data
- React component uses the data to generate the HTML and CSS code
- Avoids two way data binding

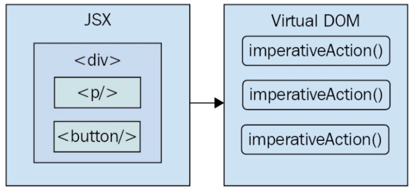
TWO PARTS

- React DOM
  - Performs the actual rendering on a web page
- React Component API
  - Data to be rendered
  - Lifecycle support
  - Events: respond to user interactions
  - JSX: syntax used to describe UI structures



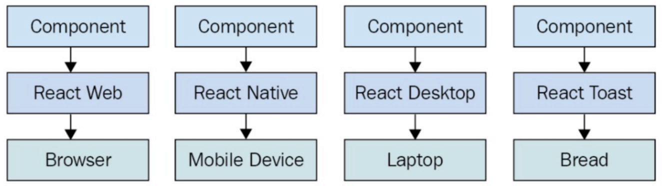
PERFORMANCE MATTERS

- Challenge of the declarative approach: performance
- React uses a Virtual DOM: representation of the real DOM elements in memory
- Calculates differences on rendering and executes only the necessary DOM operations



THE RIGHT LEVEL OF ABSTRACTION

- We don't necessarily care what the render target is
- React has the potential to be used for any UI



JSX

```
const Hello = () => (
  <p>Hello World</p>
)
```

- Syntax used by React components
- Component renders content by returning some JSX
- HTML markup, mixed with custom tags
- JSX = JavaScript XML (or: JavaScript Syntax Extension?)

https://facebook.github.io/jsx/

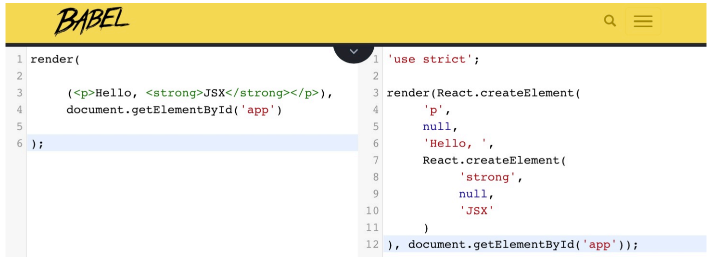
HELLO JSX

```
// The "render()" function will render JSX markup and
// place the resulting content into a DOM node. The "React"
// object isn't explicitly used here, but it's used
// by the transpiled JSX source.
import React from 'react'
import { render } from 'react-dom'

// Renders the JSX markup. Notice the XML syntax
// mixed with JavaScript? This is replaced by the
// transpiler before it reaches the browser.
render(
     (<p>Hello, <strong>JSX</strong></p>),
     document.getElementById('app')
)
```

HELLO JSX

- JSX is transpiled into JavaScript statements
- Browsers have no idea what JSX is



```
render(

    (<p>Hello, <strong>JSX</strong></p>),
    document.getElementById('app')

);
```

```
'use strict';

render(React.createElement(
    'p',
    null,
    'Hello, ',
    React.createElement(
        'strong',
        null,
        'JSX'
    )
), document.getElementById('app'));
```

https://babeljs.io/repl/

BUILT-IN HTML TAGS

- React comes with HTML components
- So we can render arbitrary HTML tags

```
import React from 'react'
import { render } from 'react-dom'

render((
  <section>
    <header>
      <h1>A Header</h1>
    </header>
  </section>
  ),
  document.getElementById('app')
)
```

COMPONENTS

```
// Function components return some JSX markup. In this case,
// "MyComponent" encapsulates an HTML structure.

const MyComponent = () => (
  <section>
    <h1>My Component</h1>
    <p>Content in my component...</p>
  </section>
)

render(
  <MyComponent />,
  document.getElementById('app')
)
```

CLASS COMPONENTS

```
class MyComponent extends Component {
  render() {
    // class components have a "render()" method
    return (
      <section>
        <h1>My Component</h1>
        <p>Content in my component...</p>
      </section>
    )
  }
}
render(
  <MyComponent />,
  document.getElementById('app')
)
```

NESTED ELEMENTS (1)

```
import React from 'react'
import { render } from 'react-dom'

// Imports our two components that render children...
import MySection from './MySection'
import MyButton from './MyButton'

// Renders the "MySection" element, which has a child
// component of "MyButton", which in turn has child text.
render((
  <MySection>
    <MyButton>My Button Text</MyButton>
  </MySection>
  ),
  document.getElementById('app')
)
```

NESTED ELEMENTS (2)

```
// MySection.js

// Renders a "<section>" element. The section has
// a heading element and this is followed by
// "props.children".

export default const MySection = (props) => (
  <section>
    <h2>My Section</h2>
    {props.children}
  </section>
)
```

NESTED ELEMENTS (3)

```
// MyButton.js

// Renders a "<button>" element, using
// "props.children" as the text.

export default const MyButton = (props) => (
  <button>{props.children}</button>
)
```

NESTED ELEMENTS

- Use `{props.children}` to access nested elements or text
- In class components: `{this.props.children}`
- Braces are used for JavaScript expressions in JSX
- In the example, the button text is passed through *MySection*
- React handles the messy details

DYNAMIC PROPERTY VALUES

```
const enabled = false
const text = 'A Button'
const placeholder = 'input value...'
const size = 50

render((
  <section>
    <button disabled={!enabled}>{text}</button>
    <input placeholder={placeholder} size={size} />
  </section>
  ),
  document.getElementById('app')
)
```

MAPPING COLLECTIONS

```
const array = [ 'First', 'Second', 'Third' ]

render((
  <section>
    <h1>Array</h1>

    <ul>
      { array.map(i => (
        <li key={i}>{i}</li>
      )) }
    </ul>
  </section>
  ),
  document.getElementById('app')
)
```

No imperative logic needed 😄

OUTLOOK

- Properties and State
- React Hooks
- Developer Tools
- Event Handling
- Reusable Components

READING MATERIAL, SOURCES

DOCS AND TUTORIALS

- React: Quick Start and Docs
  https://reactjs.org/docs/hello-world.html
- Tutorial: Intro To React
  https://reactjs.org/tutorial/tutorial.html
- Babel – a JavaScript compiler
  http://babeljs.io

SOURCES

- React – A JavaScript library for building user interfaces
  https://reactjs.org
- Adam Boduch: React and React Native
  Second Edition, Packt Publishing, 2018
  Packt Online Shop