Bringing **components** to **legacy code**

@**Wesley**de**Souza**

Developer, Work & Co

Bringing **components** to **legacy code**

"You **can't use** React, our system is incompatible"

—Stakeholder

“You **can use** whatever,
we just need to adapt”

—Me

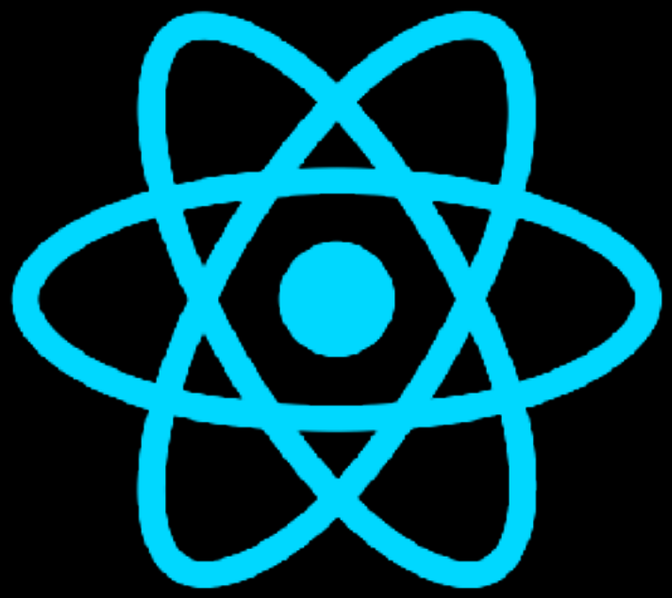
What is a
component?

A component is a software package, a web service, a web resource, or a module that encapsulates a set of related functions (or data).

—wikipedia.org

A **nice** way to
group & reuse code

—Me




ember

— **Reusability** and **composition** of code

— More **organized** code

— **Easy** to implement and expand

— Developers are **familiar** with them

What is a

legacy system?

An **existing codebase**

— **Too big** to refactor

— Necessity to **keep existing code** running

— Opportunities to **improve** and **scale**

A **restrictive system**

— **Less control** of the system

— Back-end responsible for **HTML rendering**

— Need to **compose pages** based on generated HTML

Booking.com

Drupal™

Typical framework behavior

JavaScript
sets the rules

HTML and JS loads

↓

JS defines what to do

↓

Render + Behavior

React Controls it All

```
<div id="App"></div>
```

```
render(<App />, document.getElementById('App'));
```

jQuery

```
<div class="Todo">
  <ul class="Todo--Items"></ul>
  <button class="js-todo-new">New Item</button>
</div>
```

```
$('.js-todo-new').on('click', addNewItem);
```

— Some frameworks need **total control of the DOM** for rendering

— All of them require **specific elements** to exist, configured in the JS file

— They don't expect other environments running in **parallel**

How does it play with a

legacy system

— You **can't replace** the entire system with a modern application

— The system might provide **customization through HTML**

What do we **do**?

— It's a **strategic decision** to bring a different system into place

— How does the framework **enable** the requirements of the new features?

We **flip** the responsibilities

Before:

JavaScript
sets the rules

After:

The DOM

sets the rules

HTML and JS loads

↓

DOM indicates what should load

↓

JS acts, then Render + Behavior

Using **React**

Before: React Controls it All

```
<div id="App"></div>
```

```
render(<App />, document.getElementById('App'));
```

After: Multiple React Apps

```
<div data-component="Navigation"></div>

<div data-component="MeetupHeader">
  <script type="application/json" data-props>
    {"name":"QueensJS"}
  </script>
</div>
```

After: Multiple React Apps

```
<div data-component="Navigation"></div>

<div data-component="MeetupHeader">
  <script type="application/json" data-props>
    {"name":"QueensJS"}
  </script>
</div>
```

After: Multiple React Apps

```
import Header from './components/Header';

const componentList = {
  Header: Header,
}

const collection = document.querySelectorAll('[data-component]');
Array.from(collection).forEach(element => {
  const componentName = element.getAttribute('data-component');
  const Component = componentList[componentName];

  render(<Component />, element);
});
```

After: Multiple React Apps

```
import Header from './components/Header';

const componentList = {
  Header: Header,
}

const collection = document.querySelectorAll('[data-component]');
Array.from(collection).forEach(element => {
  const componentName = element.getAttribute('data-component');
  const Component = componentList[componentName];

  render(<Component />, element);
});
```

After: Multiple React Apps

```
import Header from './components/Header';

const componentList = {
  Header: Header,
}

const collection = document.querySelectorAll('[data-component]');
Array.from(collection).forEach(element => {
  const componentName = element.getAttribute('data-component');
  const Component = componentList[componentName];

  render(<Component />, element);
});
```

After: Multiple React Apps

```
import Header from './components/Header';

const componentList = {
  Header: Header,
}

const collection = document.querySelectorAll('[data-component]');
Array.from(collection).forEach(element => {
  const componentName = element.getAttribute('data-component');
  const Component = componentList[componentName];

  render(<Component />, element);
});
```

After: Multiple React Apps

```
import Header from './components/Header';

const componentList = {
  Header: Header,
}

const collection = document.querySelectorAll('[data-component]');
Array.from(collection).forEach(element => {
  const componentName = element.getAttribute('data-component');
  const Component = componentList[componentName];

  render(<Component />, element);
});
```

⌘ + Tab

In the end

— **Data attributes** define which components are present

— Multiple systems can **coexist** in different areas of the page

— A strict back-end can **personalize** a page's components and provide data

Consider this

— Be aware of the **impact** of multiple framework applications

— **Bundling** might be a challenge

— Careful with **nesting** and **state management**

data-components

github.com/rafaelrinaldi/data-components

Thank you!

github.com/WesleydeSouza/data-component-react

wesley.so