ReactJS

Goals

- What is React? History & Goals
- Components
- Play with a Real Component
- Write Your Own Component!
- JSX and Babel (briefly)

Front End Frameworks

- Larger JS libraries
- Provide “blueprint” for apps
- “Opinionated”
  - “This is how you should design a JS app”
- Often: provide for code re-use
- Often: provide templating of HTML

Popular Front End Frameworks

- Angular
- Ember
- Vue
- React

There are many others, but these are among the most popular.

There are differences between them but they largely share lots of common ideas and after learning one framework, you’ll be in a better position to learn about others.

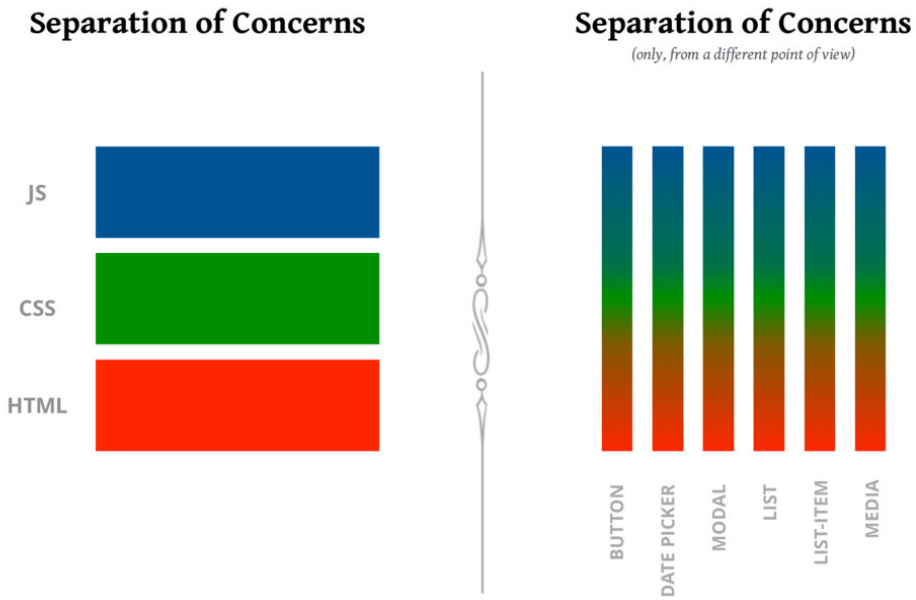
React

Popular, powerful front-end framework.

Developed by and sponsored by Facebook.

- Make it easy to make reusable “view components”
  - These “encapsulate” logic and HTML into a class
- Often make it easier to build modular applications

Goal



<_images/components.jpg>

Components

- The building blocks of React
- Pieces of UI & view logic
- Classes that know how to render themselves into HTML

(a bit like this)

```
class Dog {
  constructor (name, color) {
    this.name = name;
    this.color = color;
  }

  render() {
    return `<p>${this.name}</p>`
  }
}
```

Demo: Hello

demo/hello/index.html

```
<!DOCTYPE html>
<html>
<body>
```

```
<h1>Demo: Hello</h1>

<div id="root"> <!-- component will go in this div --> </div>

<script src=
  "https://unpkg.com/react/umd/react.development.js"></script>
<script src=
  "https://unpkg.com/react-dom/umd/react-dom.development.js">
</script>

<script src="https://unpkg.com/babel-standalone"></script>

<script src="index.js" type="text/jsx"></script>

</body>
</html>
```

A component is a React class with a ***render*** method:

demo/hello/index.js

```
class Hello extends React.Component {
  render() {
    return <p>Hi Everyone!</p>;
  }
}
```

We add our component to HTML with ***ReactDOM.render***:

demo/hello/index.js

```
ReactDOM.render(<Hello />,
  document.getElementById("root"));
```

JSX

demo/hello/index.js

```
class Hello extends React.Component {
  render() {
    return <p>Hi Everyone!</p>;
  }
}

ReactDOM.render(<Hello />,
  document.getElementById("root"));
```

What's this HTML in our JavaScript?

JSX is like HTML embedded in JavaScript:

```
if (score > 100) {
  return <b>You win!</b>
}
```

You can also "re-embed" JavaScript in JSX:

```
if (score > 100) {
  return <b>You win, { playerName }</b>
}
```

(looks for JavaScript variable ***playerName***)

Using JSX

- JSX isn't legal JavaScript
  - It has to be "transpiled" to JavaScript
- You can do this with Babel <https://babeljs.io>

Transpiling JSX in Browser

- Easy for getting started — nothing to install!
- Load ***Babel*** standalone library:

```
<script
  src="https://unpkg.com/babel-standalone"></script>
```

- Mark JSX files with `type="text/jsx"`:

```
<script src="index.js" type="text/jsx"></script>
```

- Read handouts to learn how to do on command line

> **Note: Use Babel on Command Line**
>
> While it's convenient to transpile JSX into JavaScript directly in the browser like this, it's not suitable for real-world deployment: it takes a second to do the conversion, leading to a poor experience for users.
>
> Better for deployment is to convert JSX to JavaScript once, via the command line, and then save and use the converted JS directly.
>
> To do this:
>
> 1. You need to install npm <http://npmjs.com>
> 2. Then use ***npm*** to install Babel and settings for React:
>
> ```
> $ npm install @babel/core @babel/cli @babel/preset-react
> ```
>
> 3. To convert a file:
>
> ```
> $ node_modules/@babel/cli/bin/babel.js --presets @babel/react
>     file.jsx > file.js
> ```

Serving Demo

For security reasons, Babel won't work with ***file://*** scripts

Run files under a simple static server:

```
$ python3 -m http.server
```

Then can visit at *http://localhost:8000/yourfile.html*

JSX Rules

JSX is more strict than HTML — elements must either:

- Have an explicit closing tag: `<b> ... </b>`
- Be explicitly self-closed: `<input name="msg" />`
  - Cannot leave off that `/` or will get syntax error

What JSX looks like when transpiled

Let's take a look! <https://babeljs.io/repl>

Layout

Our demo had ***Hello*** component in same ***index.js*** as placement code:

demo/hello/index.js

```
class Hello extends React.Component {
  render() {
    return <p>Hi Everyone!</p>;
  }
}

ReactDOM.render(<Hello />,
  document.getElementById("root"));
```

You'll often have >1 component, it's good to keep in separate files.

index.js

```
ReactDOM.render(<Hello />,
  document.getElementById("root"));
```

Convention: 1 component per file, with component name as filename:

Hello.js

```
class Hello extends React.Component {
  render() {
    return <p>Hi Everyone!</p>;
  }
}
```

App

It's conventional for the top-level component to be named ***App***.

This renders the other components:

App.js

```
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Greetings!</h1>
        <Hello/>
        <Goodbye/>
      </div>
    );
  }
}
```

- This way, readers of code know where to start
- This is usually the only thing rendered in ***index.js***

Order of Script Tags

demo/hello-2/index.html

```
<script src=
  "http://unpkg.com/react/umd/react.development.js"></script>
<script src=
  "http://unpkg.com/react-dom/umd/react-dom.development.js">
</script>

<script src="http://unpkg.com/babel-standalone"></script>

<script src="Hello.js" type="text/jsx"></script>
<script src="index.js" type="text/jsx"></script>
```

Make sure any components you need in a file are loaded by a previous ***script*** tag. (We'll learn about a better way to manage imports soon.)

Properties

aka. Props

A useful component is a reusable one.

This often means making it configurable or customizable.

Hello.js

```
class Hello extends React.Component {
  render() {
    return <p>Hi Everyone!</p>;
  }
}
```

It would be better if we could *configure* our greeting.

Our greeting will be *Hi _____ from _____*.

Let's make two "properties":

to

Who we are greeting

from

Who our greeting is from

Demo: Hello-2

demo/hello-2/index.js

```
ReactDOM.render(
  <Hello to="me" from="you" />,
  document.getElementById("root")
);
```

Set properties on element; get using ***this.props.propName***.

demo/hello-2/Hello.js

```
class Hello extends React.Component {
  render() {
    return (
      <div>
        <p>Secret Message: </p>
        <p>Hi {this.props.to} from {this.props.from}</p>
      </div>
    );
  }
}
```

Reusing Component

You can use a component many times:

Adding to different places:

index.js

```
ReactDOM.render(
  <Hello to="Kay" from="Kim" />,
  document.getElementById("root")
);

ReactDOM.render(
  <Hello to="me" from="you" />,
  document.getElementById("also")
);
```

Adding several at once:

index.js

```
ReactDOM.render(
  <div>
    <Hello to="Kay" from="Kim" />
    <Hello to="me" from="you" />
  </div>,
  document.getElementById("root")
);
```

Note ***div*** wrapper — JSX often renders a *single top-level element.*

> **Note: Rendering Multiple Top-Level Elements**
>
> Prior to React 16, every component had to render a single top-level element. In newer versions of React, it's possible to render siblings at the top level, but the syntax isn't quite as clean. You're welcome to look into this if you're curious, but all of our Component files will render a single element at the top of their hierarchy.

Properties Requirements

- Properties are for *configuring* your component
- Properties are immutable
- Properties can be strings:

```
<User name="Jane" title="CEO" />
```

- For other types, embed JS expression using the curly braces:

```
<User name="Jane" salary={ 100000 }
    hobbies={ ["bridge", "reading", "tea"] } />
```

Using Properties

- Get to properties inside class with ***this.props.propertyName***
- Properties are immutable — cannot change!

Conditionals in JSX

The ***render()*** method can return either:

- a **single valid** DOM object (`return <div>...</div>`)
- an array of DOM objects (but don't do this yet!)
- ***null*** (***undefined*** is not ok!)

You can put whatever logic you want in your ***render()*** method for this:

```
class Lottery extends React.Component {
  render() {
    if (this.props.winner)
      return <b>You win!</b>;
    else
      return <b>You lose!</b>;
```

```
  }
}
```

Ternary

It's very common in ***render()*** to use ternary operators:

```
class Lottery extends React.Component {
  render() {
    return (
      <b>You {this.props.winner ? "win" : "lose"}!</b>
    )
  }
}
```

Demo: Slots!

demo/slots/Machine.js

```
class Machine extends React.Component {
  render() {
    const { s1, s2, s3 } = this.props;
    const winner = (s1 === s2) && (s2 === s3);

    return (
      <div className="Machine">
        <b>{s1}</b> <b>{s2}</b> <b>{s3}</b>
        <p>You {winner ? "win!" : "lose!"}</p>
      </div>
    );
  }
}
```

demo/slots/index.js

```
ReactDOM.render(
  <Machine s1="🐚" s2="🐚" s3="🐚" />,
  document.getElementById("root")
);
```

Looping in JSX

It's common to use ***array.map(fn)*** to output loops in JSX:

```
class Messages extends React.Component {
  render() {
    const msgs = [
      {id: 1, text: "Greetings!"},
      {id: 2, text: "Goodbye!"},
    ];

    return (
      <ul>
        { msgs.map(m => <li>{m.text}</li>) }
```

```
      </ul>
    );
  }
}
```

Demo: Friends!

demo/friends/Friend.js

```
class Friend extends React.Component {
  render() {
    const { name, hobbies } = this.props;
    return (
      <div>
        <h1>{name}</h1>
        <ul>
          {hobbies.map(h => <li>{h}</li>)}
        </ul>
      </div>
    );
  }
}
```

demo/friends/index.js

```
ReactDOM.render(
  <div>
    <Friend name="Jessica" hobbies={["Tea", "Frisbee"]} />
    <Friend name="Jake" hobbies={["Chess", "Cats"]} />
  </div>,
  document.getElementById("root")
);
```

Default Props

Components can specify default values for missing props

Demo: Hello-3

demo/hello-3/Hello.js

```
class Hello extends React.Component {
  static defaultProps = {
    from: "Joel",
  };

  render() {
    return <p>Hi {this.props.to} from {this.props.from}</p>;
  }
}
```

Set properties on element; get using ***this.props.propName***.

demo/hello-3/index.js

```
ReactDOM.render(
  <div>
    <Hello to="Students" from="Elie" />
    <Hello to="World" />
  </div>,
  document.getElementById("root")
);
```

Styling React

You can add CSS classes in JSX.

However: since ***class*** is a reserved keyword in JS, spell it ***className*** in JSX:

```
class Message extends React.Component {
  render() {
    return <div className="urgent">Emergency!</div>
  }
}
```

You can inline CSS styles, but now ***style*** takes a JS object:

```
class Box extends React.Component {
  render() {
    const colors = {
      color: this.props.favoriteColor,
      backgroundColor: this.props.otherColor,
    };

    return <b style={colors}>{this.props.message}</b>;
  }
}
```

Debugging React

Install React Developer Tools <https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en>