Intro to React

INFO 253A: Frontend Web Architecture

Kay Ashaolu

Now it is time to learn React

- You now have a better handle of JavaScript
- It's time now to learn a new way of developing UI in the browser (and beyond)
- Note: there are other extensions to React (e.g. React 360)

Review: React is a library

- React is a JavaScript library for building user interfaces
- Build declarative components based on the current state of your application
- Differs from the Event Driven approach we have been doing so far

Example, Event Driven App

- Lets say you have an array of strings, each representing the subject of a todo
- And the HTML code was rendered as follows

```
<ul id="todos">
        <li>Task 1</li>
        <li>Task 2</li>
</ul>
```

Example, Event Driven App

- If you wanted to add a new task on a click of a button, what would you do?
- If you wanted to delete a task on a click of the task, what would you do?

Example, Event Driven App

- To add a new task, you would have to select the *#todos ul*, and then manipulate the html so that there is a new li at the end of the list
- To delete a new task you would need to find the right *li* and delete that from the tree

Example, Event Driven App

- This is doable, but there are some things to consider:
  - What happens if there are multiple todo lists in a single page?
  - What happens if the user tries to add and delete a task at the same time?

Events add up

- You will need to take several precautions to ensure that each todo list is completely independent and that events do not collide with each other
- This is not trivial to do for large systems

React: a different approach

- React uses a declarative programing paradigm
- Instead of worrying about every action that could happen with your list, you first define what your todo list would look like, given an array of strings.
- You create a component using your above defintion containing state that contains the titles of all of the tasks
- On click events, you modify this internal state and the component will update itself

Lets then learn React

- At this point your browser will not understand your code
- Reason 1: some browsers do not understand ES6 JavaScript
- Reason 2: some React syntax is not valid JavaScript

Let's set up your environment

- Install NodeJS on your computer
- Follow these steps from Create React App

About your dev server

- The NodeJS server you installed uses NodeJS, Babel, and Webpack, as well as the React codebase to bundle all of your source Javascript in a single file
- The NodeJS server also builds your single HTML page, as well as keeps a development server running to reload any changes

So without further ado, let's get into React!

Hello World

src/index.js

```
1 import React from "react";
2 import ReactDOM from "react-dom";
3
4 const jsx_element = <h1>Hello, world!</h1>;
5 const dom_element = document.getElementById('root');
6 ReactDOM.render(jsx_element, dom_element);
```

Hello World

src/index.html

```
<!DOCTYPE html>
<html lang="en">
<head></head>
<body>
    <div id="root"></div>
</body>
</html>
```

Hello World Explained: index.html

- Your index.html file is an empty file that contains one empty div in the body section
- Note that the empty div's id is "root"
- This div is the entry point for our react app: we will tell React in our script to replace this div with our react application
- This html file will largely **remain unchanged**

Hello World Explained: index.js

- We are first importing the React and ReactDOM packages into our JavaScript
- Next we are telling ReactDOM to render given:
  1. The content of your website
  2. What element in your index.html file that will house your React App

Hello World Explained: JSX

- JSX stands for JavaScript XML
- JSX is an extension of JavaScript that enables you to write HTML like syntax directly in your Javascript
- This enables the ability to write HTML templates directly into your JavaScript code
- You can also embed expressions, variables, and properties directly into JSX

Example

src/index.js

```
let formatName = (user) => {
  return user.firstName + ' ' + user.lastName;
}
const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};
const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Example explained

- Note: the index.html has not changed. To use React for your entire app, you can define a single div in the body that is React's entry point
- We defined a function called formatName that takes a object that has two properties: a firstName and a lastName
- The formatName function returns a single string with both of those elements
- We use this function 'formatName' inside of our JSX code (the const element)
- This shows how you can use these properties within your JSX code

Why JSX?

- Remember separating content from presentation?
- Separating HTML (content) from CSS (presentation) is core to the web
- However once we start using JavaScript, we have the ability to change the HTML rendered on the page
- That means HTML code can possibly be throughout our JavaScript codebase
- JSX gives us the ability to write out templated HTML code in a very intuitive fashion

Another Example

src/index.js

```
1  import React from "react";
2  import ReactDOM from "react-dom";
3
4  function tick() {
5    const element = (
6      <div>
7        <h1>Hello, world!</h1>
8        <h2>It is {new Date().toLocaleTimeString()}.</h2>
9      </div>
10   );
11   ReactDOM.render(element, document.getElementById('root'));
12 }
13 setInterval(tick, 1000);
```

Wait, what?

- Every call to ReactDOM.render tells React to re-render elments given the data that it currently has
- The code setInterval(tick, 1000) is a special function that tells JavaScript to execute the tick function every 1000 millliseconds.
- The tick function then defines the element and passes in its properties (namely {new Date().toLocaleTimeString()}) before that components is rendered
- This is why you see the clock ticking every second

Components

- This is one of the things I like the most about React
- The focus on components as independent, resusable pieces that can be placed anywhere
- This uses the composability relationship: each element can be composed by other elements

Components

- Using a combination of JSX and JavaScript, you can bundle look and feel and functionality in a single JavaScript class
- You can consider these React Elements and HTML Elements that you can place wherever you like
- Let's get into the anatomy of a Component

Component Example

```
1  import React from "react";
2  import ReactDOM from "react-dom";
3
4  function FormatName(props) {
5          return (
6        <h1>
7          Hello, {props.firstName} {props.lastName}!
8        </h1>
9      );
10 }
11
12 ReactDOM.render(
13   <FormatName firstName="Kay" lastName="Ashaolu" />,
14   document.getElementById('root')
15 );
```

Component Example Explained

- We are creating a component by creating a function
- The name of the function (i.e. FormatName) is the name of the element
- The React library takes that function and creates a component out of it that can be used

Props

- A parameter called *props* is passed into this function
- props is an object that contains all of the attributes and values that are passed into the element

Component Instance Properties

- When you add the attribute firstName="Kay", this props object will have a key named "firstName"
- And a value named "Kay"
- These properties are immutable

```
1 <FormatName firstName="Kay" lastName="Ashaolu" />,
```

Return

- What the function returns is the "html" that is generated by the component
- This function is executed and the "html" is generated in a number of areas in react (e.g. on a call on ReactDOM.render())
- This function returns JSX code that the component would render into

Questions?