JavaScript - Frameworks - React

# Overview

Quick js front end framework

# Setup

## Base HTML

React single page web apps render client HTML dynamically upon load of the javascript bundle. Therefore, the server can serve a very simple base HTML response, which loads the javascript bundle and contains a root element for the React HTML to be rendered into. There can be as many root elements as required for a project, which can be particuarly useful for integrating react into existing projects.

A typical react HTML page would look like:

<div id="root"></div>

Then the element would be passed into the React DOM using:

const element = <h1>Hello, world</h1>;

ReactDOM.render(element, document.getElementById('root'));

## Sass

# Concepts

## JSX

JSX is a syntax extension of javascript which allows HTML templating to be combined directly into javascript without the need for string interpolation. While normal web apps separate concerns by putting the markup and logic in separate files, react separates the concerns using components for each website feature, where inside the component the HTML template and logic are seperated.

JSX HTML is interpreted automatically when assigned to variables, variables and any valid javascript can be inserted into JSX using curly brackets:

const name = 'Josh Perez';

const element = <h1>Hello, {name}</h1>; << JSX

Attributes can be specified in JSX using string literals or curly brackets:

const element = <img src={user.avatarUrl}></img>;

Empty tags can be closed immeditaley with a />:

const element = <img src={user.avatarUrl} />;

By default, React DOM escapes any values embedded into JSX before rendering, therefore XSS attacks will be mitigated for user content.

Notes:

* Use either curly brackets or quotes for element attributes, not both
* JSX uses a camelCase naming convention

## Elements

React elements are plain javascript objects, meaning they are cheap to create. For an element to be rendered on the page, it is passed to the React DOM, which updates the browser DOM to match the elements. A simple react element would be:

const element = <h1>Hello, world</h1>;

ReactDOM.render(element, document.getElementById('root'));

React elements are immutable, meaning once they are created, they and their children cannot be changed. Therefore, the only way to update elements on the page is to re-render the element and pass it into the React DOM again.

When updating the React DOM compared the new elements to the previous, and only updates what has changed, bringing in the new state.

## Components

Components are the building blocks of a React app, allowing elements on a web app to be split into independent, reusable pieces, each with their own state.

The most basic level of a component is a function componet which accepts the argument "prop" (properties) and return a react element:

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

ES6 classes can also be used to define a class component, which allows for state management. Class components contain a render function and props is coverted to this.props:

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

Inside components, elements can be rendered in JSX using their function name, any attributed added to the element are passed as props:

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

const element = <Welcome name="Sara" />;

Typically React apps have a single App component at the top of the hierachy which is built up of other components. Using SRP components should be as small as possible while keeping their core functionality.

In respect to props, React defines that props should never be mutated or re-assinged.

## State

State can be added to a class component using the construction function to define the state, the base constructor should always be passed props:

constructor(props) {

super(props);

this.state = {

date: new Date()

};

}

State can then be accessed with this.state.<variable>, the component will then trigger a re-render each time the state changes due to a lifecycle method. State can be set using the setState method:

this.setState({comment: 'Hello'});

Due to asychronous processing, state should not be directly updated using previous state values, instead a function passing state and props should be used:

this.setState((state, props) => ({

counter: state.counter + props.increment

}));

State is kept entirely in the model it is defined in, parent or child componet should not know whether a certain component is stateful. However, state can be passed down to child components in props.

The first time a component is rendered to the DOM it is called 'mounting', when a component is removed or replaced in the DOM it is called 'unmounting'. The function componentDidMount() can be used to setup functions on component upon render, the function componentWillUnmount() can be used to trigger tear down functions to remove event listeners, intervals, and others. One use of the componentDidMount function is to async load an api request into state upon page load:

componentDidMount() {

fetchPosts().then(response => {

this.setState({

posts: response.posts

});

});

}

## Handling Events

In JSX events such as onClick are handled in camelCale and by passing a defined function, instead of a string as in standard HTML, preventDefaults must be explictly defined. Commonly in class components, events are defined as class functions, however if the event is called in JSX HTML, 'this' inside render functions may related