React Report

ReactJS is a JavaScript library used to build and render user interface components on the client side by using a combination of HTML, JSX, and JavaScript. For our project, we use React to manipulate the HTML DOM and write HTML to build out the front-end of our web application.

React first creates the root document that acts as the HTML source by containing the DOM elements and it can be rendered in the browser: https://github.com/facebook/react/blob/master/packages/react-dom/src/client/ReactDOMRoot.js#L160. Every object located within a document is a node of some kind. So after creating the root element, React can also create components for us to use as HTML elements here in this code: https://github.com/facebook/react/blob/master/packages/react-dom/src/client/ReactDOMComponent.js#L355. In the very same file, React can also create text nodes: https://github.com/facebook/react/blob/master/packages/react-dom/src/client/ReactDOMComponent.js#L462. The ReactDOMComponent can be added to the ReactDOMComponentTree to be rendered as a DOM node of our root document: https://github.com/facebook/react/blob/master/packages/react-dom/src/client/ReactDOMComponentTree.js#L181. In addition to this, the ReactDOMComponentTree can return the ReactDOMComponent or ReactDOMTextComponent instance, or null if the node was not rendered by React: https://github.com/facebook/react/blob/master/packages/react-dom/src/client/ReactDOMComponentTree.js#L161. After a document and its nodes have been created, the nodes are sent to ReactDOM and the document is checked to see if it can be rendered: https://github.com/facebook/react/blob/master/packages/react-dom/src/client/ReactDOM.js#L131. We can also set and change the elements of our document through setInnerHTML and setTextContent: https://github.com/facebook/react/blob/master/packages/react-dom/src/client/setInnerHTML.js#L24 & https://github.com/facebook/react/blob/master/packages/react-dom/src/client/setTextContent.js. We are also ensuring that the DOM is properly nested prior to rendering here: https://github.com/facebook/react/blob/master/packages/react-dom/src/client/validateDOMNesting.js.

In order to ensure that our DOM and information is consistently being updated when an action is being input, we used hooks provided by React to ensure that our front-end is always aligned with our back-end. These hooks are found inhttps://github.com/facebook/react/blob/master/packages/react/src/ReactHooks.js, and the ones most helpful to this project are useEffect(): https://github.com/facebook/react/blob/57768ef90bdb0110c7b9266185a8f6446934b331/packages/react/src/ReactHooks.js#L95, and useState() https://github.com/facebook/react/blob/57768ef90bdb0110c7b9266185a8f6446934b331/packages/react/src/ReactHooks.js#L74 . useEffect() allows our page to act upon when the page is loaded or updated, and useState() allows us to store game/ user variables in the front-end so

that as long as insensitive information stays constant, we do not have to keep sending socket events. Everytime a hook is called upon, it is called to in the resolveDispatcher(): https://github.com/facebook/react/blob/57768ef90bdb0110c7b9266185a8f6446934b331/packages/react/src/ReactHooks.js#L24. This function validates if the hooks are suitable to be acted upon, and if it is, it is returned to the rest of the code in the folder of React-Fiber.

React Fiber is core to our project with its feature of incremental rendering. What this means is that unlike normal updating where the whole page is updated and rendered, react pages are updated in frames. Chunks of the page are loaded individually, giving the ability to update information without touching outside components. useEffect() is a function that capitalizes on this advantage, and so the rest of the useEffect() code is here: https://github.com/facebook/react/blob/57768ef90bdb0110c7b9266185a8f6446934b331/packages/react-reconciler/src/ReactFiberHooks.new.js#L1104. Since useEffect() is always recurring, it will always check whether there is a change between the current snapshot of the page it just put in the queue and the previous placed in the queue, and if there is, the component[s] that is different is going to be re-rendered. useState() is a function that uses useEffect() to update the snapshot in the queue, and that is how states are read from the most recent iteration of that state in the queue.

React makes it easy for components to present any necessary data and updated data without reloading. We also used React Router to help render different components. React Router is a routing library that enables dynamic routing in React so users can navigate between components on the website.

In React, components are like JavaScript functions that take in inputs, or more commonly known as props, and they return React elements that describe what should appear on screen. In addition to props we have state which is data that can change within the components' lifetime. This is useful for updating the components in real time. The component is seen here: https://github.com/facebook/react/blob/master/packages/react/src/ReactBaseClasses.js#L20. After the component output has been rendered, the component also checks the state through ReactNoopUpdateQueue shown here: https://github.com/facebook/react/blob/master/packages/react/src/ReactNoopUpdateQueue.js#L35. If it is necessary to update the state, ReactNoopUpdateQueue will call enqueueForceUpdate() which helps change values inside the immutable state here: https://github.com/facebook/react/blob/master/packages/react/src/ReactNoopUpdateQueue.js#L62.

In React-Router, the router listens for location changes in the constructor in case there are redirects at the initial render: https://github.com/ReactTraining/react-router/blob/fc91700e08df8147bd2bb1be19a299cbb14dbcaa/packages/react-router/modules/Router.js#L60.

Whenever a <Link> is referenced inside a <Router> element, react-router constructs a forward reference that pushes the props to another location using href.location. The code for this can be found at https://github.com/ReactTraining/react-router/blob/master/packages/react-router-dom/modules/Link.js#L74 for <Link> and https://github.com/ReactTraining/react-router/blob/fc91700e08df8147bd2bb1be19a299cbb14dbcaa/packages/react-router/modules/__tests__/Route-test.js#L513 for ForwardRef.

Whenever <Switch> is referenced, it is used as the central hub in order to switch between different url_paths referenced inside its element. It is necessary to render the different routes by matching the path a user inputs to a specific route. This code is referenced at https://github.com/ReactTraining/react-router/blob/fc91700e08df8147bd2bb1be19a299cbb14dbcaa/packages/react-router/modules/Switch.js#L12.

Whenever <Redirect> is referenced, it is used in order to redirect to a specific page when a user inputs the original url. The code does this by generating a new location (the redirected location) for the user whenever the original url is inputted. This code is referenced at https://github.com/ReactTraining/react-router/blob/fc91700e08df8147bd2bb1be19a299cbb14dbcaa/packages/react-router/modules/Redirect.js#L13.

The license attached to the usages of the ReactJS library and the React Router library is the MIT License which grants access to these libraries to any person and allows the distribution of any software incorporating copies of the software.