**Application Component Structure**

The application and all its components are all loaded on the first page load, regardless if some of these components are initially shown or not. This means a slight increase in initial page load but no page refreshes or loading when switching views thanks to Reacts’ Virtual DOM. As far as the browser is concerned there’s only one HTML file it has to fetch which is ‘index.html’ which is also where all the components live, this gives the application more of a desktop program feel rather than a website, much like how Netflix or Spotify work.

React components allow for very modular designs where components can easily be swapped, duplicated or removed whenever necessary without changing any of the core architecture. There are, however, components which are more crucial to the functionality than others and which have different and important areas of responsibility. Here, the components are structured in such a way so that there is a separation of concerns:

**App.jsx:** Is the ancestor of all the other components meaning that any components that are in the application are either called directly from App.jsx or indirectly through one of its child components. This components handles routing by invoking components which are dependent on the URL and communicates directly with both the Login.jsx and Navbar.jsx components.

**Navbar.jsx:** Is child to App.jsx and sibling of Dashboard.jsx. This component handles the individual routes which are displayed on the navigation bar of the application. Note the difference between routing and routes; the former checks the URL for a specific path and displays the appropriate content while the latter supplies some sort of means (button or link) to populate the URL with a path. This component also has the necessary checks in place to provide the path to Login.jsx or Dashboard.jsx depending or not if the user has successfully authenticated.

**Login.jsx:** The login component is supplied by the Okta authentication service and is not modified by the Alphaquants devs. It captures the input information and sends it to the Okta servers, the component then receives a response in the form of an Access Token and an Id Token, in the case that the input matches Okta’s database information the Dashboard.jsx component is invoked along with all its descendants. The Access and Id tokens are stored in LocalStorage meaning that, once authenticated, even if the user closes the browser entirely, when accessing the application again from a new window he/she will not have to re-authenticate. Each Token does have an expiry timestamp however.

**Dashboard.jsx:** Since this is the first component that is invoked when a user has successfully authenticated it will be referred to as the main component or, more accurately, the parent component of the dashboard. Up to here the components have been mostly linear top-down where each component has one parent and one child, but from here the components start to branch out with components having multiple children, siblings and grandchildren. The main purpose of the Dashboard.jsx component is to receive the ‘claims’ which is the information Okta provides with its Id Token about the user profile and doesn’t render any visible content by itself. Dasboard.jsx is child of App.jsx and parent of DataAdapter.jsx.

**DataAdapter.jsx:** All of the data coming into and out of the application, either from databases or JSON, files go through this component. This allows for data abstraction and promotes low coupling between components. It separates of *what* can be done with the collection of data from *how* it is done. Any descendant components which require access to data need only to know the specific variables rather than access data from storage themselves. Again, this component only handles data manipulation and doesn’t render any visible content by itself, it does however invoke the following child components: LeftMenu.jsx, Graph.jsx, Dividend,jsx, Profile.jsx, Footer.jsx and CurrencyPortfolio.jsx.

**LeftMenu.jsx, Dividend.jsx, Profile.jsx, Footer.jsx:** These components render the visible content on the page and their names reflect the different sections of the application in order to increase readability and maintainability, they provide the necessary HTML for the page structure and use the DataAdapter.jsx properties to display user information where appropriate. These components are all children to DataAdapter.jsx and grandchildren to Dashboard.jsx.

**Graph.jsx, CurrencyPortfolio.jsx:** Exactly the same as the previously mentioned components with the only difference being that these two components also have children of their own. Graph.jsx and CurrencyPortfolio.jsx both receive properties from DataAdapter.jsx but instead of displaying the data it is passed on to their children.

**PerformanceInception.jsx, PerformanceMonth.jsx, PerformanceDay.jsx:** These subcomponents’ only purpose is to display the relevant data gathered from DataAdapter.jsx properties at the correct location in the Graph.jsx component. They are siblings, children to Graph.jsx and grandchildren to DataHandler.jsx.

**Increase.jsx, Equity.jsx, Balance.jsx:** These subcomponents’ only purpose is to display the relevant data gathered from DataAdapter.jsx properties at the correct location in the CurrencyPortfolio.jsx component. They are siblings, children to CurrencyPortfolio.jsx and grandchildren to DataHandler.jsx.

The follow diagram illustrates the React component hierarchy and the parent-child-sibling relationship between these components:

