**Confronto**

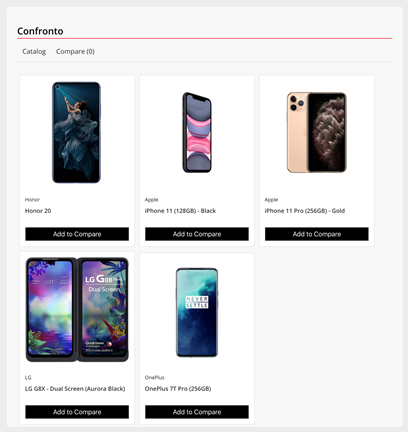
Confronto is Italian for comparison which is precisely what this project is all about. This app displays a set of products (mobile phones) and allows you to add them to a list so that you can compare their features in a tabular comparison sheet.

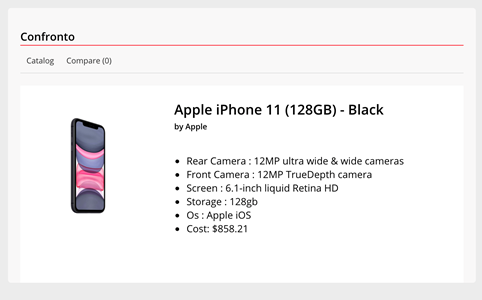
**Key Skills**

The intent of this exercise is to give you a thorough workout on the following React features:

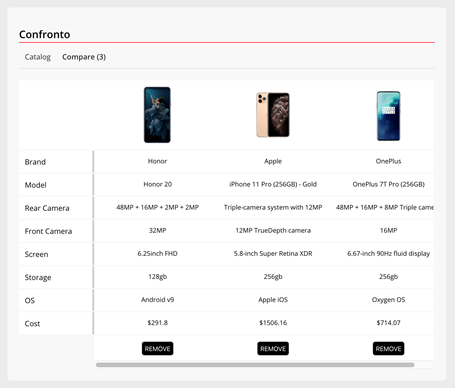
* Single page routing using React Router
* Dynamic route parameters
* Using the Context Provider component from React Router
* Using React router hooks
* Redux - Creating reducers
* Redux - Creating async action creators
* Redux - Using middleware (Redux Thunk)
* Redux - Using hooks

**User Stories**

1. The app opens with the home screen as shown below:  
  
2. Users can click on the images shown and this should navigate them to the **/product/<id>**route where <id> is the unique identifier key of the product being viewed. This should then display a UI as shown below:



3. Back on the Catalog (home route) page, clicking on the Add to Compare buttons adds phones to the comparison stack, which can be viewed by going to **/compare** route. Notice the ‘Compare’ hyperlink being displayed in bold with the number 3 indicating 3 products in the comparison stack, which is displayed as a horizontally scrolling tabular grid. You can click on Remove to remove an item from this list.



**API Endpoints**

We’re using an in-memory database and an API that you can query right away using the fetch API. The following endpoints can be used readily:

**GET/api/products:**This endpoint returns an array of products to enable you to render product cards on the home page (Catalog). The shape of the objects in the array are as follows:  
  
[{

  id: String,

  brand: String,

  model: String,

  image: String

}]

**GET/api/product/*id***: This endpoint returns an object with details of a specific mobile phone. This fetch is based on the product ID passed to the endpoint URL. The returned object has the following shape:

{

  products: {

            id: String,

            brand: String,

            model: String,

            rear\_camera: String,

            front\_camera: String,

            screen: String,

            storage: String,

            os: String,

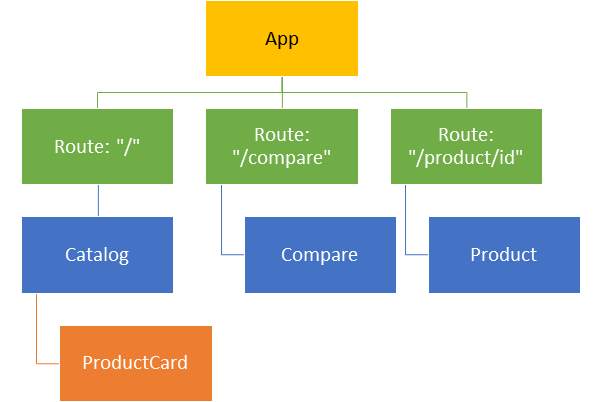
            cost: Number,

            image: String

          }

}

**Component Tree**



**Important Points**

**General Notes**

* Only function components are permitted for this assignment.
* Create all components in the */src/components.*
* Stylesheet for the assignment is provided to you and comes imported into the index.js file.
* Use the design specification documents for all components to understand how the UIs are to be built and the exact CSS selectors to use so that components render correctly.
* Please ensure all naming conventions exactly as and where mentioned for successful completion and grading of the assignment.
* Implement memoisation using React.memo().

**Notes on Redux Reducers**

You have to create two sets of reducers, one for the catalog and the other for the compare feature. Create them in the /store/reducers folder and use the /store/index.js file to combine them. This file should export the final store which is imported into the src/index.js file and configured with the Provider component.

* The catalog reducer should be initialized with the following shape:

{

  products: [],

  selected: **null**

}

* Typically, the catalog reducer should service two action types, “INITIALIZE\_CATALOG should simply populate the products array in the state with data received in action.products while setting selected to null, whereas “PRODUCT\_DETAILS” should populate the selected property with the contents of action.selected.
* The compare reducer should be initialized with the following shape:

{

    products: []

}

* The compare reducer should service two action types, “ADD\_TO\_COMPARE” which should add the contents of action.product to the products array in the state only if it doesn’t exist already. On the other hand, “REMOVE\_FROM\_COMPARE” should remove the said item from the products array.

**Notes on Redux Async Action Creators**

This project is pre-configured with the Redux-Thunk middleware, so you can build async action creators right away in the /store/actions folder.

* Async action creators for the catalog include **initCatalog()** which should fetch products from the API and dispatch the INITIALIZE\_CATALOG action. On the other hand, for fetching details of a product, the **productDetails()** action creator is used. Please ensure you name the functions as mentioned.
* Async action creators for the compare feature include **addToCompare()**which should fetch product details from the API and dispatch the ADD\_TO\_COMPARE action. On the other hand, for removing products from the stack, the **removeFromCompare()**action creator function should be used, which should dispatch the REMOVE\_FROM\_COMPARE action.

**Notes for Component: <App />**

The App Component represents the main component which composes together other components in the application and also implements state management.

* Use Function components only
* For rendering the root route (Catalog), use the Catalog component.
* For rendering the */compare* route, use the Compare component.
* For rendering the */product/<id>* route, use the Product component.
* To acquire the number of items in the compare stack on the Compare link, you’ll need to fetch the products array from the compare reducer. Implement the useSelector() hook for this purpose and use the shallowEqual function.

**Notes for Component: <Catalog />**

The catalog component renders product cards that display the image, model and brand name.

* When this component loads up as well as when the contents of the products array (catalog reducer), the list of product cards must update. Implement a useEffect() which dispatches the initCatalog() action creator and the useSelector() hook for fetching the products array.
* For rendering the product cards, use the ProductCard component.

**Notes for Component: <ProductCard />**

This component renders a card that displays the image, model and brand name of a mobile phone.

* Accept all data using props.
* This component implements a button labelled ‘Add to Compare’ which should dispatch the addToCompare action creator function, passing in the id of the product.
* The image element must also intercept mouse clicks and should navigate the user to the */product/<id>* route where <id> is the unique ID of every product received as a prop in this component. Implement the useHistory() hook here.
* For rendering the image, import the getImage() function from the /src/server module | import {getImage} from “../server” and invoke it when setting the image source, passing in the value of ‘image’ as received in the prop.

**Notes for Component: <Product />**

This component displays details of the product when the user browses to */product/<id>* where <id> is the unique id of every product as received from the API.

* Use the useParams() hook to fetch the dynamic <id> value from the route path.
* Dispatch the productDetails() action creator with the id value to fetch details of the product.
* For rendering the image, import the getImage() function from the /src/server module | import {getImage} from “../server” and invoke it when setting the image source, passing in the value of ‘image’ as received in the *selected* state object.

**Notes for Component: <Compare />**

This component renders a horizontally scrolling tabular grid of products that are selected for comparison. The intent is to let the user compare specifications easily.

* Simply fetch and render the contents of the products array from the compare reducer.
* For rendering the image, import the getImage() function from the /src/server module | import {getImage} from “../server” and invoke it when setting the image source, passing in the value of ‘image’ as received in the *products* state object.
* Dispatch the removeFromCompare() action creator when the Remove button is pressed.

Assignment Details

Marks

75

Duration

60 mins

Pass Percentage

50%

Sub-skills Covered

Routing

Redux

Reusing Code & Logic

Start Assignment