**APOLLO PROJECT PART-2**

In this project, you’ll complete the second part of Apollo: **Passing Information to Components**.

So far, you’ve hard coded business information and rendered it manually. Now we’ll focus on how information (like business information) can pass from component to component with less hard coding. This is crucial as we build towards the goal of constructing a Yelp-like clone.

**Get Started  
1.**In terminal window, go into the Project directory and npm start to start a development server and open up a preview of your app in a browser tab.

**Create businesses Array  
2.**When the Yelp API is implemented, a list of businesses will be returned to **App.js**. Therefore, open **Business.js**. Cut the entire business object and paste it into **App.js.** Make sure to paste it right after the imports and above the App class declaration.

Go back to **App.js**. Under the business object, create a businesses array. The array  
should hold six references to the business object.

**Setting a BusinessList Prop  
3.**In **app.js**, add a businesses property to the <BusinessList /> component inside of the render() method. Set the property equal to the businesses array. (Don’t forget your curly braces!)

**Add Functionality to the BusinessList Prop  
4.**Open **BusinessList.js**, inside of the BusinessList div, access the businesses prop and call the .map() method on it.  
**5.**Inside of the map() method, pass a callback function with one parameter called business.

**6.**The callback function should return a <Business />component. The returned <Business /> should have a property called business. Set the property equal to the parameter of the callback function. (Don’t forget your curly braces and semicolon!)

**Modifying Business.js using Props  
7.**Open **Business.js**. What do you notice about how information is accessed inside of the return statement?  
Business information now has to be accessed via the business prop you send.

**Add a SearchBar Constructor  
8.**Throughout this project, you’ll work mainly on setting state and handling state changes in **SearchBar.js**.  
Open **SearchBar.js**. Add a constructor in the SearchBar component. Make sure to pass the constructor props and call super(props) on the first line.

**9.**Inside of the constructor, set the state of the search bar. Use this.state and set it equal to an empty object.

**10.**Add three keys to the state object you just created. The keys should be term, location, and sortBy.  
term will refer to the search term located in the search input, location will refer to the location to search near from the location input, and sortBy will represent the selected sorting option to use.  
The first two keys should be set to empty strings (“”). The last key should be set to 'best\_match'. This completes the constructor.

**Get a Sort Option's Class  
11.**On the Apollo page there is no way to determine which option has been selected. This a problem; we’ll fix this by creating a method that returns the current CSS class of the sort options, returning whether or not each one should be styled as if it has been selected.

Move the sortByOptions object to the last line of the constuctor() (inside component) and change it from a local variable to a member variable using this. Be sure to also update the two references to sortByOptions in the renderSortByOptions() method to now use this.

Then, add a new method called getSortByClass that accepts one parameter called sortByOption.

**12.**Inside, use an if statement to check if the state value of sortBy is equal to the provided parameter sortByOption. If it is, return 'active', otherwise, return an empty string (“”).

**Handle a Change in Sort Option  
13.**getSortByClass() returns the current CSS class for a sorting option. This method will prove useful in providing visual feedback to users.

We’ll need another method, will be useful when communicating with the Yelp API in the future.  
Add a new method called handleSortByChange. It should accept a parameter called sortByOption.

**14.**Inside of .handleSortByChange(), update the state of sortBy by calling .setState(). Set to the value of the method’s argument.

**Set the Class Name of a Sort Option  
15.**Let’s put these two new methods to work. Take a look at the .renderSortByOptions() method. You’ll modify the return statement that returns an <li> element with these two new methods.:  
Add a className attribute to the <li> element. Set it equal to the return value of the getSortByClass() method. Pass in sortByOptionValue as the argument.

**Handle a Sorting Option Change (On Click)  
16.**Next, we’ll update the state of a sorting option when it is clicked. Add an onClick attribute to the <li> element. Set it equal to handleSortByChange.bind(). Pass in two arguments to .bind(): this and sortByOptionValue.

This will allow us to both bind to the current value of this (as we usually do in the constructor()) but also bind the current sortByOptionValue as the first argument to the method call, ensuring the method is called with the appropriate value when clicked.

**Handle a Term or Location Change  
17.**Apollo will also need to handle changes in the two input elements. Specifically, it will need to handle changes in “Terms” (businesses) and “Location” (location to search in).

Under the handleSortByChange() method, add two new methods:  
• handleTermChange()  
• handleLocationChange()  
Since both will be related to events being triggered, both should accept event as an argument.  
**18.**Inside of both methods, the state of each input element should be updated with the value of event.target.value.

**Handle a Term or Location Change (onChange)  
19.**Let’s use the methods!  
Inside of the return statement of the component’s render() method,  
add onChange attributes to each <input> element:  
• Set the first attribute to handle term changes.  
• Set the second attribute to handle location changes.

**searchYelp() in the App Component  
20.**The “Let’s Go” button also needs to provide some sort of feedback to the user. In **App.js**, add a method called searchYelp() in the class declaration of the App component. (Place it above the render() method.)

**21.**searchYelp() should accept three parameters: term, location, and sortBy. These parameters represent the three pieces of information we’ll send to the Yelp API.

**22.**You’ll complete this functionality later. For now, inside of searchYelp(), log a message to the console that uses the three parameters. Here’s an example message:  
“Searching Yelp with Pizza, Brooklyn, best\_match”

**Set the searchYelp Prop in SearchBar  
23.**Let’s use this method. Add a searchYelp property to the SearchBar component in the return statement of .render(). Set it equal to this.searchYelp.

**Handle a Search  
24.**searchYelp() will print a message to the console, simulating a search. This  
will only happen when the “Let’s Go” button is clicked.  
In **SearchBar.js**, add a method called handleSearch(). Place it under the handleLocationChange() method.  
**25.**.handleSearch() should accept an event parameter. Inside of .handleSearch(), call the passed down .searchYelp() method (located on props). Pass in the current state values of term, location, and sortBy as arguments.

**26.**On the next line, call event.preventDefault() to prevent the default action of clicking a link from triggering at the end of the method.

**Handle a Search (onClick)  
27.**The last step is to put handleSearch() to work. Add an onClick attribute to the “Let’s Go” button. Set it equal to this.handleSearch.

**APOLLO PROJECT PART-3**

Welcome to the final part of the Apollo project! Interacting with the Yelp API.

**Create a util Directory  
28.**We’ll store the Yelp functionality in its own directory. Create a folder called **util** in the **src/** directory.

**29.**Inside **util/** directory create a file called **Yelp.js**.

**Retrieve Yelp API v3 Credentials  
30.**APIs require that you retrieve and use valid credentials (provided by them) in order to use the API.  
Start by navigating to the “Manage App” section of the [documentation page](https://www.yelp.com/developers/v3/manage_app). Note that you’ll have to log in (or create an account if you don’t already have one).

**31.**Once you login, create a new app using the menu to the left. Fill out the required fields for the app (you can use “Apollo” for “App Name”). Agree to the terms and conditions and create the app.

**32.**You should see an “API Key” presented to you. Take note of this, you’ll use it in **Yelp.js**.

**Add Credentials to Yelp.js  
33.**Open **Yelp.js**. At the top of the file, create a constant called apiKey. Inside it, store a string of your API key from the Yelp API.

**Create a Yelp Module  
34.**Create an empty object called Yelp. This object will store the functionality needed to interact with the Yelp API.

**Add search()  
35.**Inside object, create a method called search. This is the method we’ll use to retrieve search results from the Yelp API. Pass in term, location, and sortBy parameters to the method.

**36.**Inside of the method, begin the method with a return which will return a promise that will ultimately resolve to our list of businesses. After return begin your chain of calls by calling fetch().

**37.**To retrieve businesses, you’ll have to hit the /businesses endpoint of the Yelp API. Pass in the following path as the first argument to fetch():  
*https://api.yelp.com/v3/businesses/search?term=TERM&location=LOCATION&sort\_by=SORT\_BY*

Use interpolation to replace TERM, LOCATION, and SORT\_BY with the correct variables in the path above.

**38.**Your fetch() will currently not function correctly due to [CORS restrictions](https://www.codecademy.com/articles/what-is-cors). We can bypass this restriction with an API called [CORS Anywhere](https://cors-anywhere.herokuapp.com/). CORS Anywhere will take requests sent to its API endpoint, make them for the requesting app with the proper CORS permissions, and then return the response back to the requesting app. Prepend the URL path you passed to the first argument in fetch() with the following:  
*https://cors-anywhere.herokuapp.com/*

**39.**When we make requests to the Yelp API, we have to present a form of identification for the browser. This is because the Yelp API wants to know that we are authorized to access the API.  
This identification is presented using our API key as a browser header.  
Pass in a second argument to the fetch() call. It should be an object with a key of headers.

The value of headers should be another object. This object should have Authorization as a key. The value of this key should be:  
*`Bearer ${apiKey}`*

**40.**We will need to convert the returned response to JSON for us to be able to effectively utilize our list of businesses.  
Chain a call using then() to the end of the fetch() method.  
Pass in a callback function. The callback function should:  
• Use arrow function syntax  
• Take one parameter called response  
• Return a call to .json() on response

**41.**Chain another .then() call after the previous to retrieve the list of businesses from the converted JSON response. Pass in a callback function that takes one parameter called jsonResponse.

**42.**Inside of the callback function, add an if statement that checks to see if jsonResponse has a businesses key (this would represent a valid response returned by the Yelp API).  
If it doesn’t, we don’t want our site to crash trying to render a list of businesses that don’t exist.

**43.**If this key does exists in the JSON response, we should return an array that has all of the business properties we’ll need (the ones we previously hard coded, like name, address, city, and more).

In the if statement, iterate through jsonResponse.businesses using map().

**44.**Pass a callback function to the map() method. The callback function should take a parameter called business.

**Accessing the JSON Response  
45.**Inside of the callback function, return an object.  
This object will be lengthy and should include all of the attributes needed to display a business in Apollo.  
We’ll provide the list of keys that should be in the jsonReponse object:  
• id  
• imageSrc  
• name  
• address  
• city  
• state  
• zipCode  
• category  
• rating  
• reviewCount

**46.**Finally, export your Yelp module.

**Import Yelp.js  
47.**The Yelp API functionality you wrote will need to be used by App. Open **App.js** and import Yelp at the top of the file (after the first four imports).

**Remove Hard Coded Information  
48.**Remove the hard coded business and businesses information from **App.js**.

**Add an App Constructor  
49.**Above the searchYelp() method, add a constructor.  
Inside of the constructor:  
• Set this.state with a key of businesses set to an empty array.  
• Bind the searchYelp() method to the current value of this (since we are about to update it to use this)

**Modify searchYelp()  
50.**Remove the console.log() message from searchYelp().

Inside of searchYelp(), call Yelp.search(). Pass the same three arguments supplied to .searchYelp() in your Yelp.search() call.

**51.**Once we retrieve our list of businesses, we will need to update the state.  
Chain a then() call to the end of Yelp.search().  
Pass it a callback function that takes one parameter called businesses.  
Inside of the callback function, update the state using setState().

**52.**Now that you are querying Yelp for real businesses and passing down the returned list of businesses, your app should work as expected. Try out querying for a few different types of restaurants and zipcodes to make sure it works as expected.

**Add a key to <Business />  
53.**Now that we have real data, we can modify the Business component that BusinessList renders. Every list item rendered by a React component needs to have a unique key, we were unable to hard code this, but we can now use the unique ID of each business as the key.  
Open **BusinessList.js**. Add a key property to the rendered <Business/> component. Set it equal to the ID of business.

Congratulations you just made a full front-end web application that updates automatically and uses real data! This is a truly awesome achievement.

**Completion and Reflection**You’ve completed the Apollo project!  
You’ve come a long way. You started the project by:  
• Hard coding business information  
• Simulating a list of businesses  
• Simulating a query to the Yelp API  
Now, your project is at a state where all of these initial issues have been  
solved. You can now search the Yelp API for restaurants that match a search option!

We suggest that you push your React knowledge by adding features to your project. This will allow you to not only reinforce your new skills, but create something unique and exciting in the process. Enjoy!

Hint: Suggestion; add a search option to your page, may be “distance”.