**Setting the Search Bar's State**

Welcome to the third part of the Ravenous project! Take a minute to review what you accomplished in the second part of this project. You:

* Moved business information to the container component (App)
* Passed information from parent components (App) to child components (BusinessList, Business)

In this project, you’ll complete the third part of Ravenous: **Setting the State of Ravenous Components**.

Certain components will need to handle changes in their state. For example, the sorting options in the search bar will change (and we’ll need to know their state when communicating with the Yelp API). The same goes for the two input elements. These are the kinds of changes you’ll handle in this project.

A few notes before getting started:

* You should expect to spend some time on this project. Don’t worry if you don’t complete it in one sitting.
* If you don’t understand how to implement a certain part of the project, we’ll try to provide guidance as needed via hints. However, you may need to look back at previous lessons that will provide you with the relevant information.

Let’s get started!

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Help**“ to see a **project walkthrough video**.

**Tasks**

**0/26Complete**

Mark the tasks as complete by checking them off

**Add a SearchBar Constructor**

**1.**

Throughout this project, you’ll work mainly on setting state and handling state changes in **SearchBar.js**.

By the end of this project, the search bar options will reflect changes in their state when they are clicked, and the “Let’s Go” button will respond to click events. Let’s get started.

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.

**2.**

Inside of the constructor, set the initial state of the search bar.

Use this.state and set it equal to an empty object.

**3.**

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**

**4.**

Click on the sort options above the two input elements on the Ravenous page. You’ll notice that nothing happens. They remain the same color and there is no way to determine which option has been selected. This a problem for two reasons:

* A user doesn’t receive feedback after clicking on a sort option
* The Yelp API won’t know which results to return if it isn’t clear which sort option has been selected

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() 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 after the constructor() that accepts one parameter called sortByOption.

Hint

A member variable of an object is created using this

this.sortByOptions = {

'Best Match': 'best\_match',

'Highest Rated': 'rating',

'Most Reviewed': 'review\_count'

};

**5.**

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

Hint

sortBy is in the this.state object.

**Handle a Change in Sort Option**

**6.**

getSortByClass() returns the current CSS class for a sorting option. This method will prove useful in providing visual feedback to users of Ravenous.

We’ll need another method, however, that *sets* the state of a sorting option. This 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.

**7.**

Inside of .handleSortByChange(), update the state by calling .setState(). Pass in an object to setState(). The object should set sortBy to the value of the method’s argument.

Familiarize yourself with this pattern. You’ll make use of it in upcoming steps.

**Set the Class Name of a Sort Option**

**8.**

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.

This will conditionally style each sort by option, displaying to the user which sorting option is currently selected.

**Handle a Sorting Option Change (On Click)**

**9.**

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**

**10.**

Ravenous 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.

**11.**

Inside of each method, update the state using setState(). Pass in an empty object into each call of setState().

Hint

Use step 8 to help you (you’ve done this before).

**12.**

Inside of both methods, the state of each input element should be updated to reflect the text typed *into* the respective input element.

In .handleTermChange(), the object passed to .setState() should have a key called term. Set the key to a value of event.target.value.

Hint

{term: event.target.value}

**13.**

Do the same thing in .handleLocationChange(), but name the key location instead.

**Bind Methods in the Constructor**

**14.**

Since both of these methods use this, you’ll need to bind them. Above the sortByOptions object in the constructor(), bind both methods (handleTermChange() and handleLocationChange()) to the current value of this.

**Handle a Term or Location Change (onChange)**

**15.**

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.

Hint

Use step 10 to help you (you’ve done this kind of thing before).

**searchYelp() in the App Component**

**16.**

The functionality you just built will provide feedback to the user when they select a different sorting option.

The “Let’s Go” button also needs to provide some sort of feedback to the user. Currently, clicking on it doesn’t do anything. Let’s build functionality that simulates what a search might look like.

In **App.js**, add a method called searchYelp() in the class declaration of the App component. (Place it above the render() method.)

**17.**

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

**18.**

Remember, we’re building partial functionality to *simulate* a search. You’ll complete this functionality in the next project.

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

Hint

Use interpolation to help you write the message.

**Set the searchYelp Prop in SearchBar**

**19.**

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.

Hint

Use step 10 to help you (you’ve done this kind of thing before).

**Handle a Search**

**20.**

searchYelp() will print a message to the console, simulating a search. This will only happen when the “Let’s Go” button is clicked, which is the missing functionality you’ll build now.

In **SearchBar.js**, add a method called handleSearch(). Place it under the handleLocationChange() method.

**21.**

.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.

**22.**

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

**Bind handleSearch()**

**23.**

Next, bind the handleSearch() method. Place it under three methods you previously bound.

**Handle a Search (onClick)**

**24.**

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.

**Completion and Reflection**

**25.**

Take a look at Ravenous in your browser. Click through the sort options at the top of the search bar. What do you notice? What kind of visual feedback do you receive upon clicking a sort option?

Next, open the developer tools in your browser (i.e. the console in Google Chrome). Enter a business name (or food type, like “pizza”) and a city into the search bar. Then click on the “Let’s Go” button. What does the console output?

**26.**

Congratulations! You’ve completed the third part of the Ravenous project. Let’s review what you built:

* Added visual feedback for the sorting options at the top of the search bar
* Set the state of sorting options and input elements
* Simulated a search query with the “Let’s Go” button

Reflect on the knowledge you used in this project. As you do so, consider the following question:

* How can we interact with the Yelp API using the “Let’s Go” button?

We’ll expand the functionality of the “Let’s Go” button in the next project to address the question above.