Creating A Live News App Using React

react 02 Oct, 2019

In this tutorial we'll be creating a news application with updated and real-life news. We'll learn how to fetch data from a third party API and present the result using a UI component framework.

Goals

The application that we'll be building, will let a user search for news articles related to a specific topic. We'll be using a third party API called News API, which is free to use. We'll also be using Semantic UI React which is a UI component framework which comes with prebuilt and styled React components.

The tutorial will teach you how to connect to a third party API and process the JSON metadata returned by the API. You will also learn how to use prebuilt components from Semantic UI with basic responsiveness features.

You can find the source code for the finished project below.

- View Source

Prerequisites

In order to follow along with this tutorial, you'll need some basic knowledge of HTML, CSS, and JavaScript/ES6. You should also know the fundamentals of React, which you can learn by reading Getting Started with React.

Since we'll be connecting to a third party API, the News API, it's recommended to have a basic

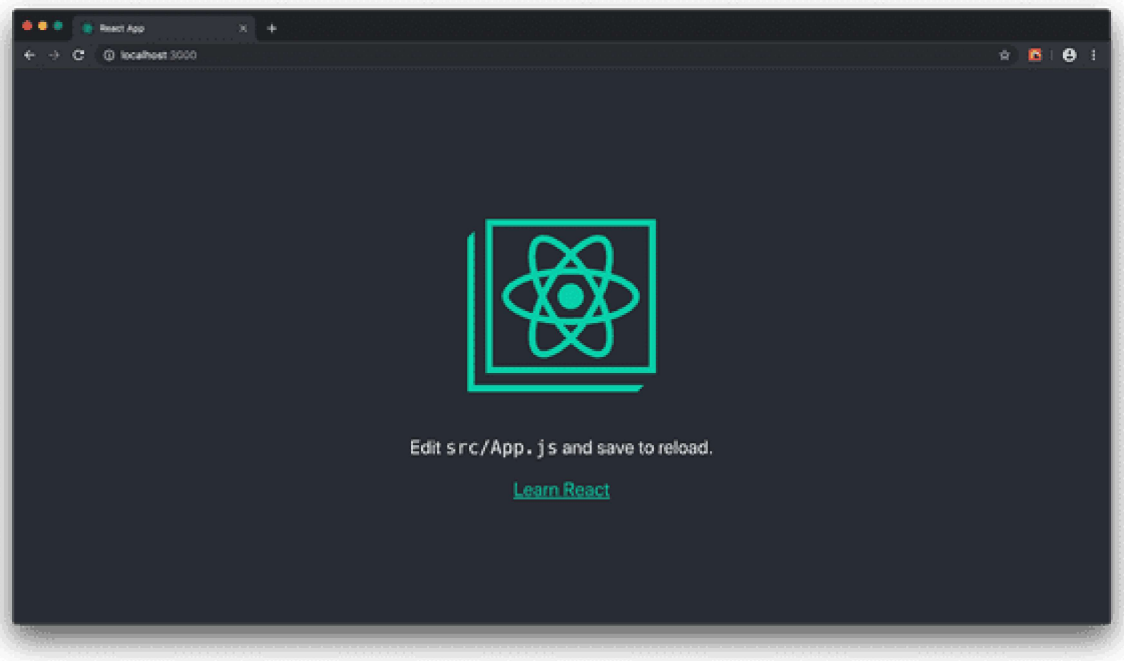
Create React App

Let's begin by initializing a new React project using the command `create-react-app`. In you terminal, run the following command one directory above where you want the project to live.

```
npx create-react-app news-app
```

Once the loading is complete, you can start the application and go to localhost:3000 to see your application.

```
cd news-app
npm start
```



Initial Setup

We'll start by cleaning up the project a bit and delete some files that are created by the boilerplate. Delete all the files in the `/src` folder except `App.js`, `index.js`, `index.css`.

Let's also get rid of some code in the `App.js` file so we can start from scratch.

```
function App() {
  return <div>We'll add our code here!</div>;
}

export default App;
```

And inside the `index.js` file, we clean up so we only have the following code left.

```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";

ReactDOM.render(<App />, document.getElementById("root"));
```

Semantic UI React

The next step is to install and setup Semantic UI React. The easiest way to get started with Semantic UI React, is to run the following command.

```
npm install semantic-ui-react
```

Semantic UI React also needs the general Semantic UI stylesheet to be be styled. So let's install that as well.

```
npm install semantic-ui-css
```

Once it has finished installing, we need to import it into the entry point of our React app: `index.js`.

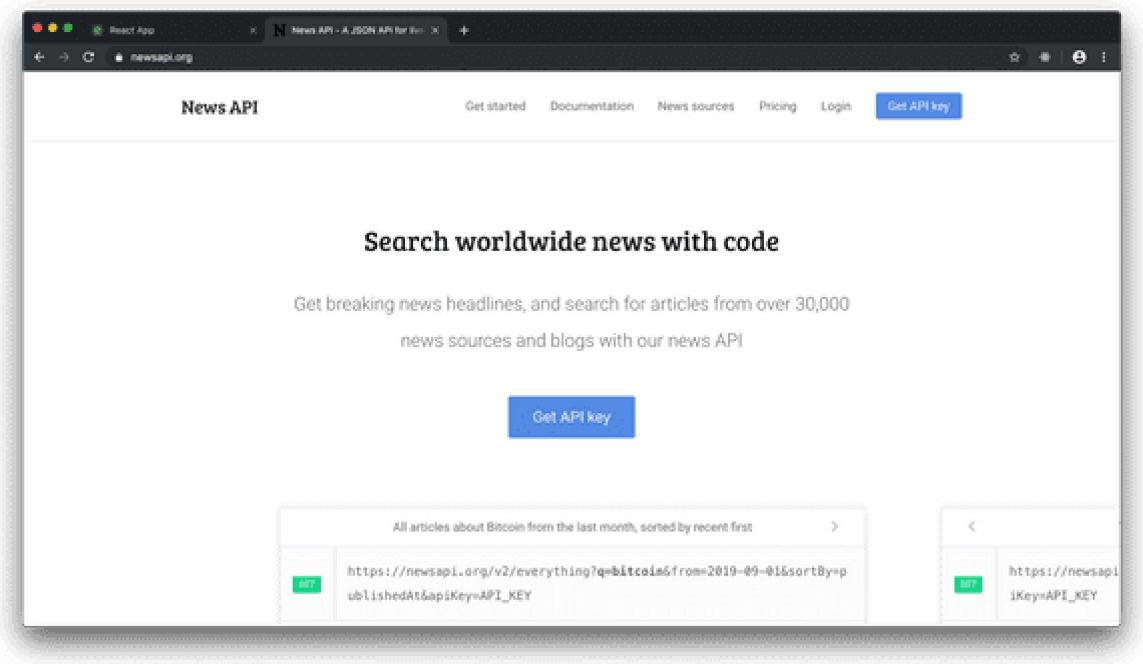
```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";
import "semantic-ui-css/semantic.min.css";

ReactDOM.render(<App />, document.getElementById("root"));
```

Now, we are ready to use Semantic UI in our React application.

News API

To be able to connect and receive articles from the News API, you'll need a API key. You can get one if you head over to News API and click on the button "Get API key". You'll need to create an account and before getting access to the key.



Now we need to store the key within our application so we can use it to access the News API. Create a new file `config.js` inside the `/src` folder. Inside the file, create a constant `NEWS_API_KEY` that holds the value of your News API key. Don't forget to export it so we can access it from other files in our project.

```
export const NEWS_API_KEY = "YOUR API KEY";
```

Note: The key is private and should be kept private and secure. If someone gets access to your key, they can access the API on your behalf. If you're connecting to a paid third party API, they can make request which you pay for. So make sure you know how to secure them properly before deploying your website.

On the website News API there are some example queries with the request and response. We'll try fetching the first example - "All articles about Bitcoin from the last month, sorted by recent first".

We'll begin by creating a new file `api.js` inside `/src`. Inside the file, we'll be creating functions to make requests to the News API. We'll be using the Fetch API to fetch information from the News API.

First we need to import the API key from `config.js` which we'll need inside the url when we make a request to the API. Then we create a function `getBitcoinArticles` which makes a `GET` request to the News API using the `fetch` function from the Fetch API.

```
import { NEWS_API_KEY } from "./config";

export const getBitcoinArticles = async () => {
  try {
    const response = await fetch(
      `https://newsapi.org/v2/everything?q=bitcoin&sortBy=publishedAt&apiKey=${NEWS_API_KEY}`
    );
    const json = await response.json();
    console.log(json);
  } catch (error) {
    console.log(error);
  }
};
```

Let's go through the above code step-by-step.

1. We start by importing the `NEWS_API_KEY` from the `config.js` file.
2. We create and export an arrow function `export const getBitcoinArticles = async () => {}`. The keyword `async` declares that the function we create is an asynchronous function. This keyword is important as we are working with asynchronous code, when we're connecting to the News API. `async` is required in order to use `await`.
3. We create a `GET` request using the `fetch()` function to the path provided from the News API docs and assign the value to a constant `response`. Note that we add our API key in the last part of the url. The `await` keyword before the `fetch()` function is related to `async` and means that the code won't execute any more lines until we receive any response from the API. If the API returns any errors, the `await` keyword throws the rejected value.
4. The `response` that we receive from the API is just a HTTP response. To extract the `JSON` body

5. We wrap the code within a `try-catch` block, in order to catch any errors thrown when connecting to the API.

We now have a function to get some articles. Let's use this in our `app.js` to see the result. Inside the `app.js` we need to import and call the function.

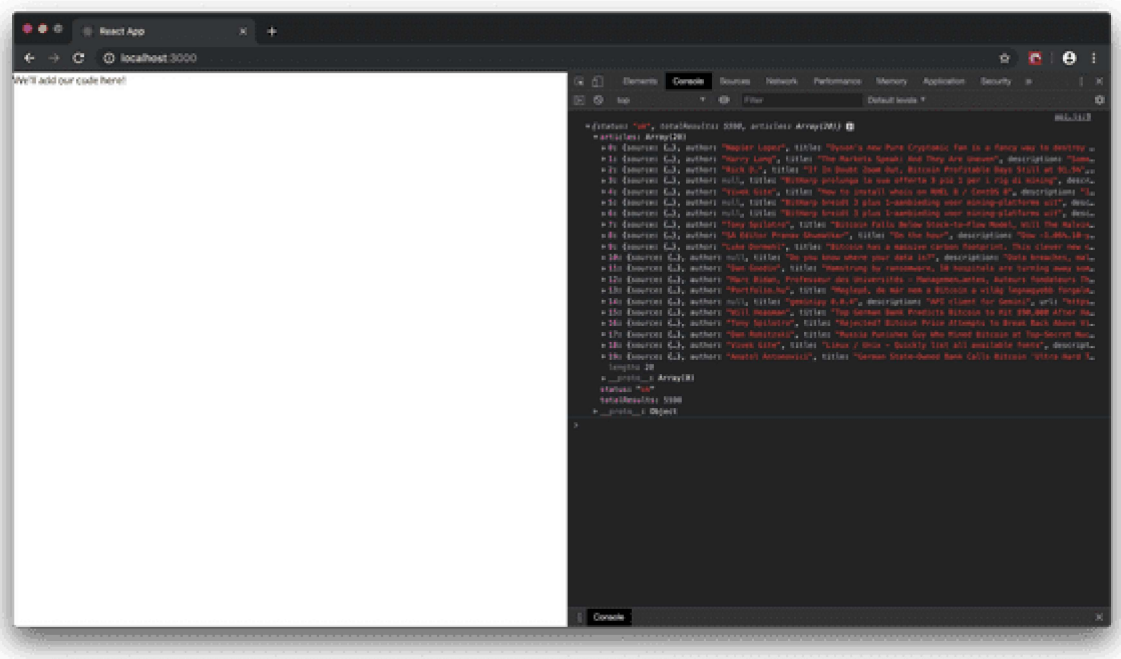
```
import React from "react";
import { getBitcoinArticles } from "./api";

function App() {
  getBitcoinArticles();

  return <div>We'll add our code here!</div>;
}

export default App;
```

Now we can see the changes in our browser if you open the `Chrome Developer Tools`. Inside the console we see that the response returned from the API is an object with the `statusCode`, the total search results `totalResults` and an `array` of the `articles`.



Adding State

an array of articles. `articles` into the state which we can assign the response from the API to. We also add a string `apiError` into the state to handle any errors we receive from the API.

```
import React from "react";
import { getBitcoinArticles } from "./api";

class App extends React.Component {
  state = {
    articles: [],
    apiError: "",
  };

  render() {
    return <div>We'll add our code here!</div>;
  }
}

export default App;
```

The next step is to use the method that we created earlier, `getBitcoinArticles()`, to make a request to the News API. We'll use the `componentDidMount()` method which is a React method that is called immediately after the component has been mounted. The method suits well for making API requests. But first we'll make some changes in the method `getBitcoinArticles()` inside the `api.js` file.

```
export const getBitcoinArticles = async () => {
  const response = await fetch(
    `https://newsapi.org/v2/everything?q=bitcoin&sortBy=publishedAt&apiKey=${NEWS_API_KEY}`
  );
  const json = await response.json();
  return json;
};
```

Here we have remove the `try-catch` block and instead add it inside the `componentDidMount()` method. By doing so, we can update the `apiError` inside the state if an error is thrown. When the API executes successfully, we can update `articles` in the state and assign the `array` of articles from the response.

```
    this.setState({ articles: response.articles });
  } catch (error) {
    this.setState({ apiError: "Could not find any articles" });
  }
}
```

Note: The API returns an object with the keys: status, totalResults and articles. We'll be using the totalResults later.

With the current setup, we should have an `array` of articles in the state. Now it's time to create a component to present the article.

Creating the UI

Here we'll utilize some components from the Semantic UI library. We'll also be adding some inline styling, but this is something we won't discuss as it's not the focus of this tutorial. Inside the `render()` method of the `App` component we'll create the core design. Let's start by import the components from Semantic UI.

```
import { Container, Header } from "semantic-ui-react";
```

The `Container` component limits the content to a reasonable maximum width based on users screen size and thereby adds responsiveness to the app. We also import `Header` which is just as the name suggests, which we use to add a title to our list of articles. You can go to the documentation at Semantic UI React to read more about the available components and features.

Now, we'll use this components inside the `render()` method.

```
render() {
  const { articles, apiError } = this.state;
  return (
    <Container>
      <Header as="h2" style={{ textAlign: "center", margin: 20 }}>
        Bitcoin articles
```

```
    </Container>
  );
}
```

By wrapping all the content inside the `Container` component we'll get some adding space and responsive design. Inside the `Container`, we first create a title by using the `Header` component as a `h2` title (indicating the hierarchy level). Then we take advantage of the JSX feature and check if articles exists in the state. If articles exists, it will render the component `ArticleList`. We'll be creating this custom component soon, but for now we'll just be passing down `articles` from the state as a prop. Next, we do the similar for the `apiError` and render a message if an error exists.

Create a new folder `/components` inside the `/src` folder. Inside it we'll create a file `ArticleList.js` where we'll create our custom `ArticleList` component.

```
import React from "react";
import { List } from "semantic-ui-react";

const ArticleList = (props) => {
  return (
    <List divided style={{ maxWidth: 900, margin: "0 auto" }}>
      {props.articles.map((article) => (
        <List.Item key={article.title}>{article.title}</List.Item>
      ))}
    </List>
  );
};

export default ArticleList;
```

Since we passed down the `array` of articles as a `prop`, we can access it through `props.articles` inside this component. The component returns a `List` component from Semantic UI and inside we want to have a list item for each article in the `array`. We achieve this by using the `map()` method on `props.articles` and return a `List.Item` with the title for each article. We give each child in the `array` a `key` prop set to the title.

The `key` prop should be assigned a unique value, preferably a unique id number. React uses this keyword to identify an element inside the DOM. In our case, we choose the title of the article for simplicity. However, this implies that two articles could have the same key if they have the same

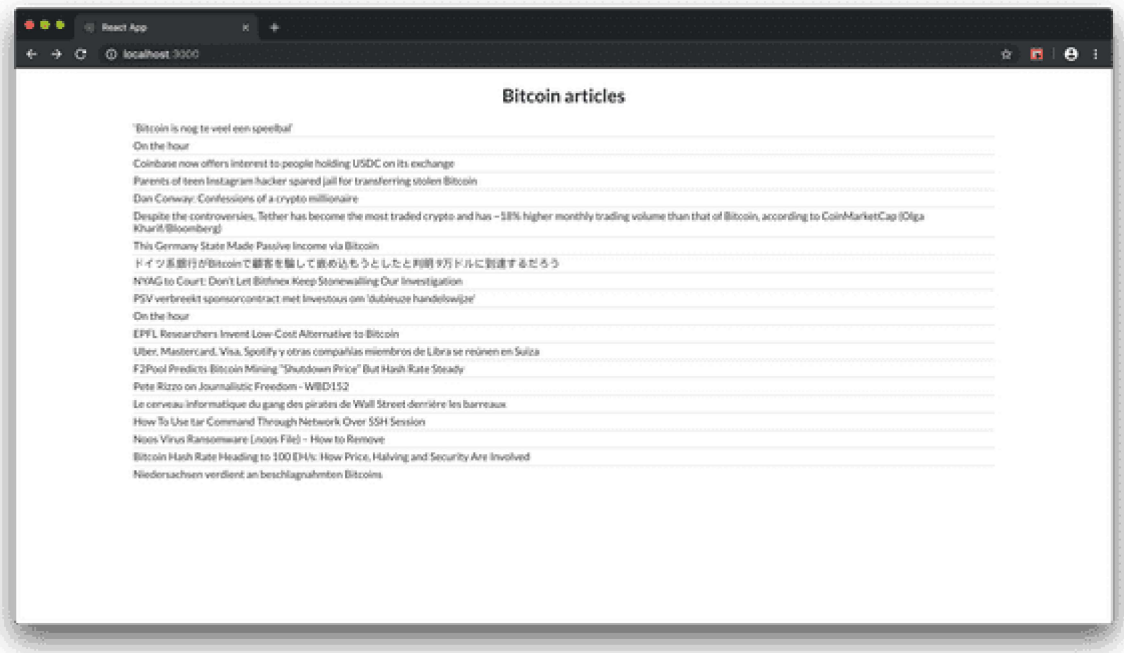
browser, you might receive an error stating that two children have the same key (if two articles have the same title).

In order to fix this, we'll modify the key with the value of the title of the article followed by the index of the mapped item.

```
props.articles.map((article) => (
  <List.Item key={article.title + index}>{article.title}</List.Item>
));
```

> **Note:** In our case, the list and it's items are static and won't change once fetched. Also, the list can't be reordered or filtered. If your working with lists that dynamically change, it's recommended to use an id (if it exists) or use a unique id generator such as shortid.

You should now be able to see a list of article titles if you go to your browser.



We're now displaying a list of titles for the most recent articles related to "Bitcoin". It's quite boring and ugly, let's add some more information about the article and add some styling.

Let's start by creating a new component `ArticleItem` inside the `ArticleList.js` file to make the code a bit more readable. We'll add this component inside the `map()` function for each item in the `array`

```
const ArticleItem = (props) => {
  const { article } = props;
  return <List.Item>{article.title}</List.Item>;
};

const ArticleList = (props) => {
  return (
    <List divided style={{ maxWidth: 900, margin: "0 auto" }}>
      {props.articles.map((article, index) => (
        <ArticleItem article={article} key={article.title + index} />
      ))}
    </List>
  );
};
```

The plan is to have the article title, description, publish date and publisher to the left and the article image to the right. We can accomplish this by using the `Grid` component from Semantic UI and creating two `Grid.Column`.

```
const ArticleItem = (props) => {
  const { article } = props;
  return (
    <List.Item style={{ padding: 30 }}>
      <Grid>
        <Grid.Column
          width={11}
          style={{
            display: "flex",
            flexDirection: "column",
            justifyContent: "flex-start",
          }}
        >
          <Header as="h3">{article.title}</Header>
          <List.Description style={{ margin: "20px 0" }}>
            {article.description}
          </List.Description>
          <List bulleted horizontal>
            <List.Item>
              <a href={article.url}>{article.source.name}</a>
            </List.Item>
            <List.Item>{article.publishedAt.split("T")[0]}</List.Item>
          </List>
        </Grid.Column>
```

```
      </Grid>
    </List.Item>
  );
};
```

We just added some lines of code and you might feel overwhelmed. We won't go through the code line-by-line as the majority of the code is related to design. The lesson learned here, is how to connect to a third party API and combine the returned information with components from a UI component library, such as Semantic UI, to create a cool application.

If you go to your browser, you should be able to see more information about each article and the new layout.

There's one more thing to do before we finish this tutorial, adding a search bar so a user can search for topics other than bitcoin.

Adding a Search Bar

We're currently only showing news articles related to bitcoin. Let's add a search bar so a user can search for topics that they're interested of.

Create a new file `searchBar.js` inside the `/components` folder. Inside this file, we'll create a form with

forms before in React, I recommend checking out my previous post How to create a Todo List App with React where I cover the fundamentals.

Let's start by creating the actual form and then add the methods for handling the search. We'll create a stateful class component since we'll be using the local state to keep track of the user input. Semantic UI React has a `Form` component which we'll use. Again, I won't go into any detail on the design of the form.

```
import React from "react";
import { Button, Form } from "semantic-ui-react";

class SearchBar extends React.Component {
  constructor(props) {
    super(props);
    this.state = { searchTopic: "" };
  }

  render() {
    return (
      <div style={{ display: "flex", justifyContent: "center" }}>
        <Form onSubmit={this.handleSubmit}>
          <Form.Group>
            <Form.Input
              placeholder="Search topic"
              name="topic"
              value={this.state.searchTopic}
              onChange={this.handleChange}
            />
            <Button type="submit" color="green">
              Search
            </Button>
          </Form.Group>
        </Form>
      </div>
    );
  }
}

export default SearchBar;
```

There are two things to notice here, the first is that we set the value of the form input to the value of the state. The second is the two methods that we need to create to handle the form submission,

When a user starts typing for a topic, we call the `handleChange()` method and inside update `searchTopic` inside the state with the value. When the form is submitted, we call the `onSubmit()` method and first prevent the default HTML form behavior. Then we call a method `searchForTopic()` with the `searchTopic` as argument. We'll create this method inside the `App` component to handle the search and pass the method down as a `prop`.

New API Method

Before updating the `App` component, we'll we need to create a new method inside `api.js` that makes a request to `News API` based on the topic that the user wants to search for. We create a method `getArticles()` which takes a topic as a argument. The difference from the previous method is that we add the topic inside the query using string interpolation.

```
export const getArticles = async (topic) => {
  const response = await fetch(
    `https://newsapi.org/v2/everything?q=${topic}&sortBy=publishedAt&apiKey=${NEWS_API_KEY}`
  );
  const json = await response.json();
  return json;
};
```

Updating App Component

We can now change the layout and logic of the `App` component and use the created method to fetch articles based on the topic that the users searches for. The changes that we're going to make are:

1. Import the new method `getArticles()` from `api.js` and `SearchBar` component.
2. Add `searchTopic`, `totalResults` and `loading` to the state to make the app a bit more user friendly.
3. Add the new information and `SearchBar` component inside the `render()` method of the `App` component.

The first change we need to make, is to call our API method first when the user searches for a topic. We move the code that we previously had inside the `componentDidMount()` method to a new method `searchForTopic()` and update the state with the new information. We also set `loading` to

```
searchForTopic = async (topic) => {
  try {
    this.setState({ loading: true });
    const response = await getArticles(topic);
    this.setState({
      articles: response.articles,
      searchTopic: topic,
      totalResults: response.totalResults,
    });
  } catch (error) {
    this.setState({ apiError: "Could not find any articles" });
  }
  this.setState({ loading: false });
};
```

Now we can update the UI with the new information. Let's change the title so it makes a bit more sense, add our `SearchBar` component passing down the `searchForTopic()` method as a prop, render a text when the app is loading so the user is updated and finally add the total search results found.

Before we wrap it up, we'll be adding one last line of code. We can't forget to add an attribution link to the News API, as we're able to access their service totally free.

With all these changes, the final `app.js` file looks as follows.

```
import React from "react";
import { Container, Header } from "semantic-ui-react";
import { getArticles } from "./api";
import ArticleList from "./components/articlesList";
import SearchBar from "./components/searchBar";

class App extends React.Component {
  state = {
    articles: [],
    searchTopic: "",
    totalResults: "",
    loading: false,
    apiError: "",
  };

  searchForTopic = async (topic) => {
    try {
      this.setState({ loading: true });
      const response = await getArticles(topic);
```

```
        totalResults: response.totalResults,
      });
    } catch (error) {
      this.setState({ apiError: "Could not find any articles" });
    }
    this.setState({ loading: false });
  };

  render() {
    const {
      articles,
      apiError,
      loading,
      searchTopic,
      totalResults,
    } = this.state;
    return (
      <Container>
        <Header as="h2" style={{ textAlign: "center", margin: 20 }}>
          Search for a topic
        </Header>
        <SearchBar searchForTopic={this.searchForTopic} />
        <p style={{ textAlign: "center" }}>
          Powered by <a href="https://newsapi.org/">NewsAPI.org</a>
        </p>
        {loading && (
          <p style={{ textAlign: "center" }}>Searching for articles...</p>
        )}
        {articles.length > 0 && (
          <Header as="h4" style={{ textAlign: "center", margin: 20 }}>
            Found {totalResults} articles on "{searchTopic}"
          </Header>
        )}
        {articles.length > 0 && <ArticleList articles={articles} />}
        {apiError && <p>Could not fetch any articles. Please try again.</p>}
      </Container>
    );
  }
}

export default App;
```

Let’s go try it out in the browser.

Wrapping Up

There we have it, we just created our own news app which connects to a third party API. We also learned how to set up Semantic UI React and saw some examples of how to use it. There's a lot of additional features you can add to the app, such as a news feed with trending articles and the ability for a user to filter by time or location. If you got lost during the way, you can find the source code for the final project below.

- View Source

I hope this tutorial has helped beginner React developers learn new concepts like connecting to a third party API, working with a UI component framework and hopefully also overall development of a simple React application.

Was this article helpful? Please share it!

I put in a lot of work and effort in all my posts and tutorials. I'd really appreciate if you could help and share them!