

Working with REST API and React

Overview

In the project we will create a simple React application displaying varieties of news from newsapi.org

Key learnings

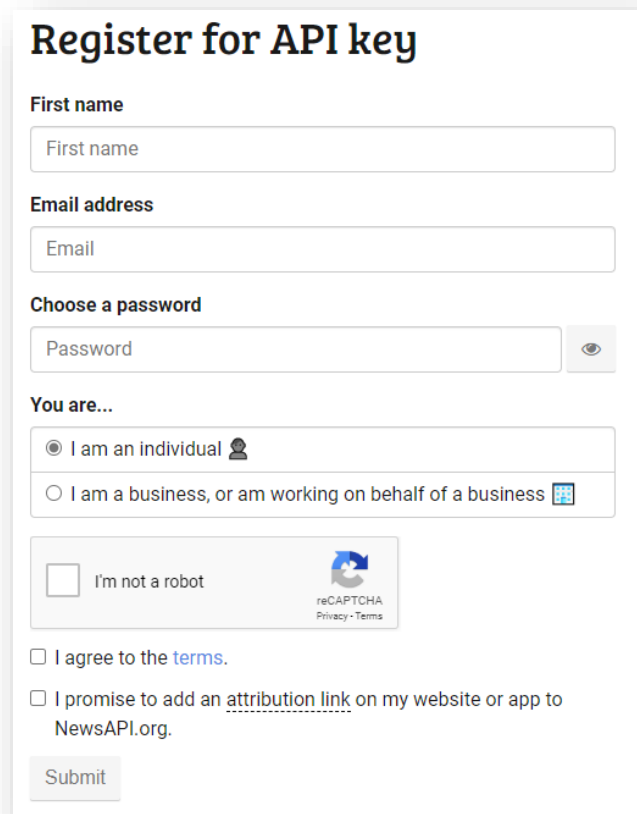
- Consuming REST API's
- Using postman to connect and test the web API
- Building simple client side React application
- Using developer tools of your browser

Introducing the REST API

News API is an API that provides different news items. It has many end points each providing news based on certain input. To use this API you have to register first in order to get a key. The key must be used in the client side in order for the web API recognize that the request is coming from a registered account.

Registering at the newsapi.org

Registering process is very simple and straight forward. In below steps we will get a free key to access the API. You need to fill in the needed information as shown below.



The screenshot shows a registration form for NewsAPI. It includes fields for 'First name', 'Email address', and 'Choose a password'. Below these is a section 'You are...' with two radio button options: 'I am an individual' (selected) and 'I am a business, or am working on behalf of a business'. There is a reCAPTCHA checkbox labeled 'I'm not a robot' and a 'Submit' button at the bottom. Two checkboxes for terms and conditions are also present.

Register for API key

First name
First name

Email address
Email

Choose a password
Password

You are...

☒ I am an individual

☐ I am a business, or am working on behalf of a business

☐ I'm not a robot

☐ I agree to the [terms](#).

☐ I promise to add an [attribution link](#) on my website or app to NewsAPI.org.

Submit

Once you register you will get an email with the API key. This key will be associated with all requests coming from the client, the React application in our case. The client application will embed the key inside the requested URL, part of the query string. This key is used for development / testing purposes. Therefore, it has a capacity of 500 requests per day, so keep this in mind.

Web API end points

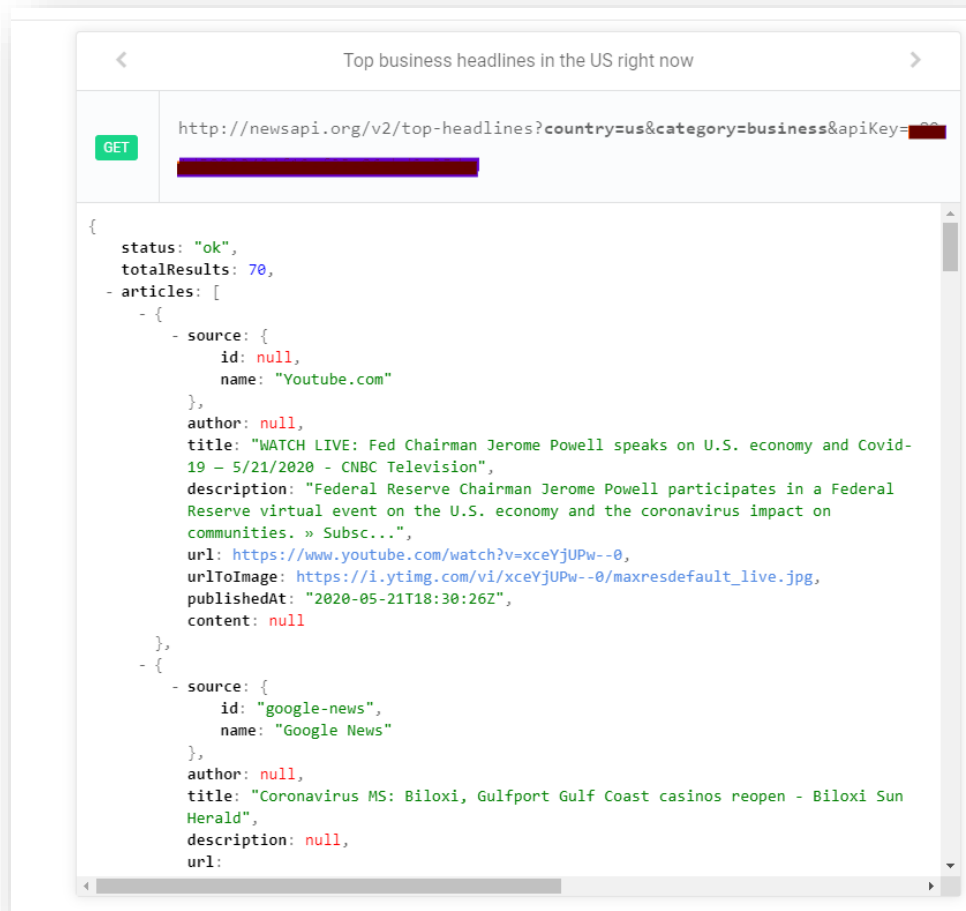
This API has three main end points

1. [Top headlines /v2/top-headlines](#) - returns breaking news headlines for a country and category, or currently running on a single or multiple sources. You can get news from this end point based on **Sources, country, Keywords or a phrase to search and category**

2. [Everything /v2/everything](#) - indexes every recent news and blog articles published by over 50,000 different sources, and you can search through them with this endpoint. You can get news from this end point based on **Keywords or phrases to search, sources, domains, language and dates**
3. [Sources /v2/sources](#) - returns information (including name, description, and category) about the most notable indexed sources. You can get news from this end point based on **language, country and category**

All the endpoints need the API key as well.

Below is an example of how the query string looks like and the results.



A Closer Look at newsapi.org results

If we inspect above results from the web API we see that it returns an object with three fields: `status`, `totalResults` and `articles`. `Articles` field is an array of news items objects. Each news item has 8 fields: `source`, `author`, `title`, `description`, `url`, `urlToImage`, `publishedAt` and `content`. `Source` itself is an object with two fields: `id` and `name`.

We can access the author's file of news' items by using this line :

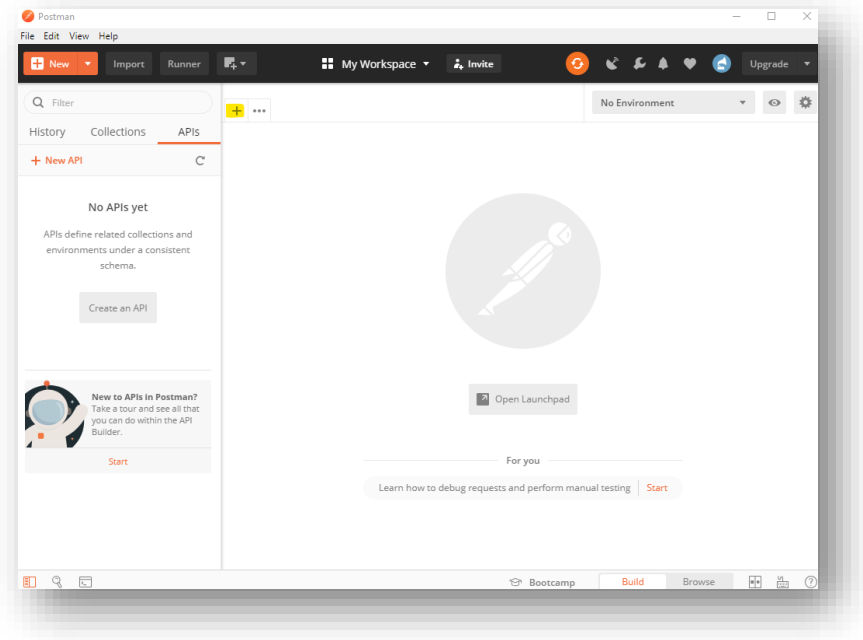
```
articles[index].title
```

and to access the source name we use below code

```
articles[index].source.name
```

Postman

Postman is a tool that is used to call web API's to test their functionality and their responses. [Download](#) this tool and let's check the web API responses!



Open a new tab

If not opened yet, open a new tab and paste any url provided from news.org webpage along with your KPI key.

The screenshot shows a REST client interface with a GET request to `http://newsapi.org/v2/top-headlines?country=us&category=business&apiKey=c801d58828424f19af23a86ebd0a35de`. The response is a JSON object with the following structure:

```
{
  "status": "ok",
  "totalResults": 70,
  "articles": [
    {
      "source": {
        "id": null,
        "name": "Youtube.com"
      },
      "author": null,
      "title": "WATCH LIVE: Fed Chairman Jerome Powell speaks on U.S. economy and Covid-19 - 5/21/2020 - CNBC Television",
      "description": "Federal Reserve Chairman Jerome Powell participates in a Federal Reserve virtual event on the U.S. economy and the coronavirus impact on communities. » Subsc...",
      "url": "https://www.youtube.com/watch?v=xceYjUPw--0",
      "urlToImage": "https://i.ytimg.com/vi/xceYjUPw--0/maxresdefault_live.jpg",
      "publishedAt": "2020-05-21T18:30:26Z",
      "content": null
    },
    {
      "source": {
        "id": "google-news",
        "name": "Google News"
      },
      "author": null,
      "title": "Coronavirus MS: Biloxi, Gulfport Gulf Coast casinos reopen - Biloxi Sun Herald",
      "description": null,
      "url": "https://news.google.com/_i/rss/rd/articles/..."
    }
  ]
}
```

The upper part of the purple line represents the request and the lower part represents the results. The results are in json format. To test an end point we put it in the upper part next to the GET action.

Inspect the structure of the response for each news items. These are the fields that you will use in the React client application.

Creating the client application (React Application)

All React applications needs Node.js to be installed in your system. [Download](#) Node.js and install it. Node.js comes with NPM by default.

Next you need to create the React application using Create-React-App command. Navigate first to a parent folder where you want to create your application. For example, if you want to create your application in a folder in the desktop navigate to your desktop and then follow steps below.

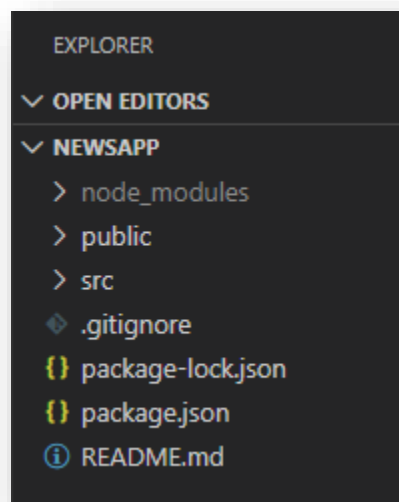
Open the command prompt and execute below commands. Pay attention that

- my-app is the folder name where you want your application to run
- npx comes with npm 5.2. Double check the npm version by executing `npm -version` in the command prompt

```
npx create-react-app my-app  
cd my-app  
npm start
```

you will see your application running on your default browser.

To start creating your application open the folder <my-app> in Visual Studio Code. In our example, the folder name is NEWSAPP.



Building the client application

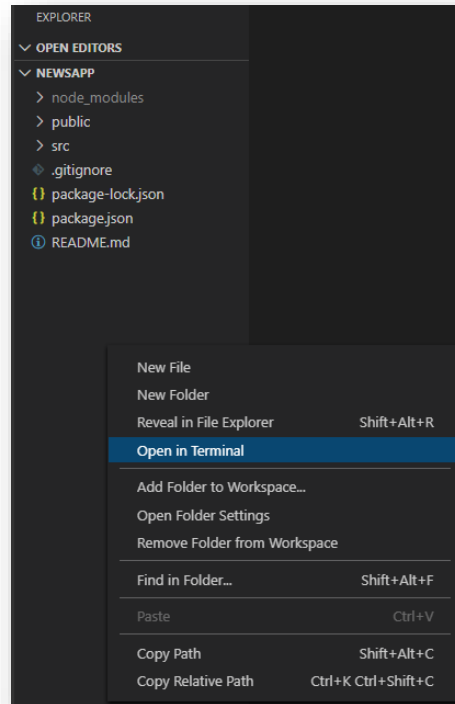
We will create the client application gradually in 3 main phases.

- Phase one will define the structure of React component we are going to use and makes sure the application is working in its simplest format.
- Phase 2 we will use the fetch JavaScript function to link the client application with the web API
- And in phase three we will add some beautifying design features to the UI.

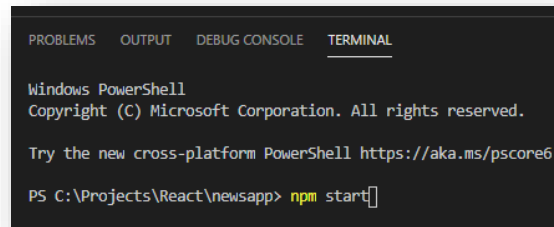
The application components

We will start with a very basic application. When we created the react application using `create-react-app` the application was built with the App.js component.

To run the react application in visual studio code, open the application folder in a terminal.



Then run `npm start` command



All what the App.js component does is that it displays a page with a spinning icon as of now. Try and see



Phase 1 – Adding the NewsList.js component

Now let's add our first component to the application. This component will only displays "Hello from NewsList component" message.

Below are the main steps we will do in this initial phase.

1. Create the NewsList.js component
2. Use the newly created component in the App.js component

Create the NewsList.js component

1. Create a new file in the `src` folder, by selecting the `src` folder and then adding a new file
2. Name the file as `NewsList.js`
3. Paste below code into the file and save the file

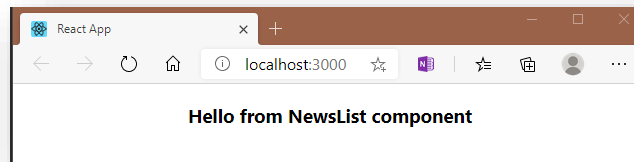
```
import React, {Component} from 'react';
class NewsList extends Component //inherits from Component
{
  render ()
  {
    return (
      <h3>Hello from NewsList component</h3>
    )
  }
}
export default NewsList; // export is necessary to make it visible to other components
```

Updating the App.js component

Paste the following code in the `App.js` and then save the file.

```
import React , {Component} from 'react';
import './App.css';
import NewsList from './NewsList'
class App extends Component {
  render () {
    return (
      <div className="App">
        <NewsList />
      </div>
    );
  }
}
export default App;
```

once the files are saved the webpage will be updated automatically! And it will display the message set in the `NewsList` component



Homework

Try to add another `NewsList` component in the render method of the `App.js` component. Check what will be rendered in the browser.

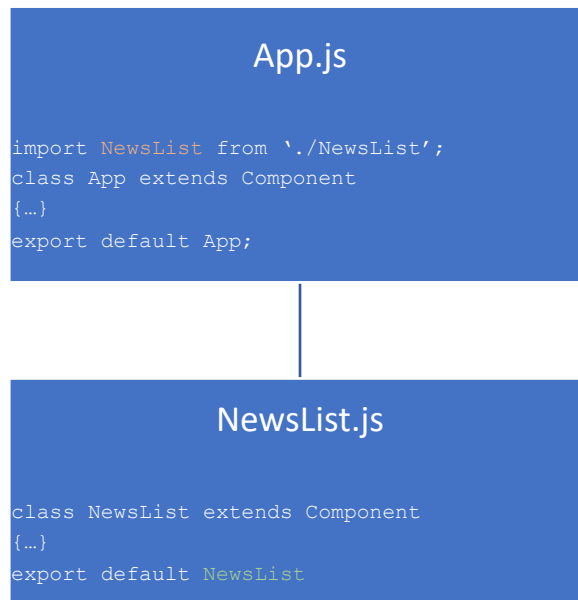
What we did

1. Created a `NewsList` component
 - a. Inherited from the `Component` class in the react library
 - b. Exported the class at the end of the module
2. Imported the component in the main `App.js` to display the message in the browser

Before we proceed to the next phase, remember the following rules.

- a. Remember to **export** the component at the end of the file. Check (`NewsList.js`) below
- b. To use a component in other components you have to **import** it at the beginning of the file. Check (`App.js`) below
- c. The name of the component in the **import** statement should match the name in the **export** statement
- d. To use the component use JSX tags `<NewsList />`

Below figure shows the outcome of this phase on a high level.



Phase 2 – Connecting the react application to the web API

We will use the `fetch` JavaScript function to call the news API. But first we need to cover two prerequisites in order to complete this phase:

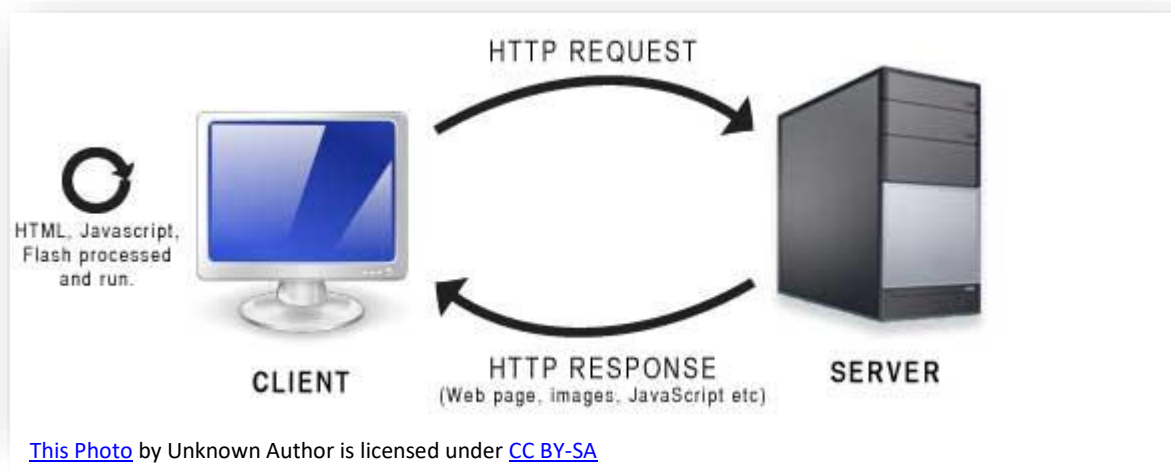
1. Understand the JavaScript `fetch()` function
2. Understand components life-cycle

A closer look at `fetch`

- It is used to send network requests to the server and gets back with the response
- It is compatible with most modern browsers
- It is asynchronous and can be used with `async/await` keywords or with another syntax using `.then/.catch`. The focus in this document will be on the latter.

To use `fetch` we need to know the URL and some options to use it successfully. Since we are calling a web API we need to know the URL for that API and its end points.

A web API is a class which is accessible by the outside world using http protocol. The client sends https requests and the server respond back with http responses.



Each of the class methods represent an end point. Each end point represents a certain functionality on the web. Some end points return lists of items, other end points accepts some input and save whatever is received to a database. When we use a web API we are calling individual end points. But the calls to these end points must match to the functionality of that end point. We use web action when calling a certain web API end points. There are 4 main actions which are relatively close to [CRUD](#) operations:

1. GET action (the R in CRUD). This action is used to read or get a resource from the server. A resource could be a database record/row, a file, ... etc.
2. POST action (the C in CRUD). This action is used to add a resource to the server
3. PUT action (the U in CRUD). This action is used to update an existing resource in the server
4. DELETE action (the D in CRUD). This action is used to remove a resource from the server

There are other http actions but above text is enough as an introductory to web API's and their end points.

Fetch JavaScript function can be used to call any of the above actions. However, we will only use the GET action because we are only reading from the news API.

Fetch expects two parameters when is used in client-side JavaScript applications, URL and Options. If Options parameter is omitted, fetch will assume that the call is a GET request. This will make our application straight forward.

```
fetch(url, [options])
```

since fetch is asynchronous function, it can be called using below syntax:

```
fetch(url)
  .then(response => response.json())
  .then(data => console.log(data))
  .catch (err=> console.log(err));
```

Keep in mind:

- When using Fetch the response from the server has two parts:
 - The server returns a promise, an object with headers only
 - To get the actual data we have to call another method of the promise object which is called when data is ready.
- The response in the first .then() does not contain the data (response body). The data can be acquired by calling any of the following methods:
 - `response.text()` – returns the response body as text
 - `response.json()` – returns the response body as [JSON](#)
 - `response.formData()` – return the response as [FormData](#) object
 - `response.blob()` – returns the response body as Blob (binary data)
 - `response.arrayBuffer()` – return the response as [ArrayBuffer](#)

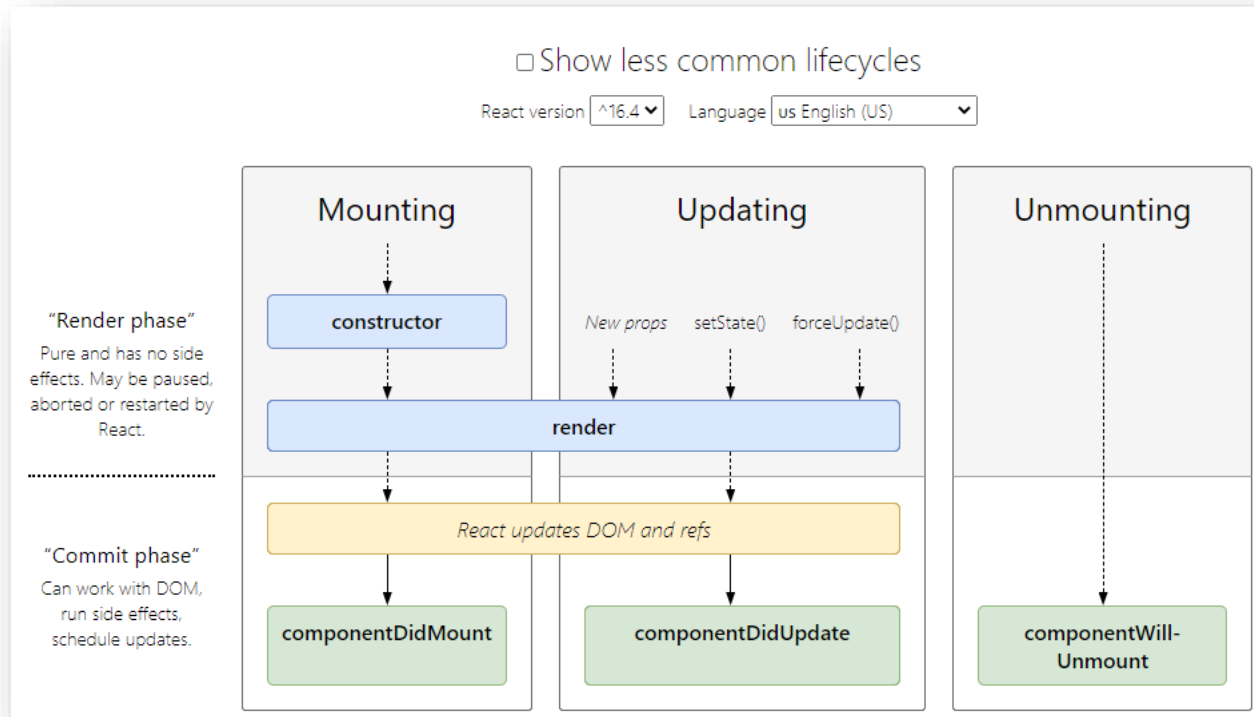
Our focus here is `response.json()` which is called automatically when the data is available and ready. Thanks to `.then()` which calls a function that has a response as a parameter and returns `response.json()`

- The 2nd `.then()` is calling a function which includes the actual data from the server
- `.catch()` calls a function with an error object as an argument. It is called only if something wrong takes place

To know more about this function, <https://javascript.info/fetch> is a great place to check.

We will use the `fetch()` function in the `NewsList` component. The best way to connect to database or call web API's in React application is when the component is mounted, inside the `componentDidMount()` method. What is this method? The React library has defined certain methods of a component that are called on certain events. [The React official site](#) has great explanation in case you need more information. According to their website below are the component lifecycle events.

Component lifecycle



Please note:

1. **constructor** and **componentDidMount** methods are called only once in component life cycle when the component is first loaded and **componentWillUnmount** when the component is about to close
2. **Render** and **componentDidUpdate** are called many times based on the change to either the state of the same component or to properties passed to it

We will start to update the code soon but let's first install React Developer Tools to our browser.

1. Chrome: search for React Developer Tools from <https://chrome.google.com/webstore/> and add the extension
2. Edge: search for React Developer Tools from <https://microsoftedge.microsoft.com/addons/Microsoft-Edge-Extensions-Home> and add the extension

Now let's add the `fetch()` function into `NewsList` component but first you need to prepare your API key from newsapi.org. you can copy any of the urls provided from their website.

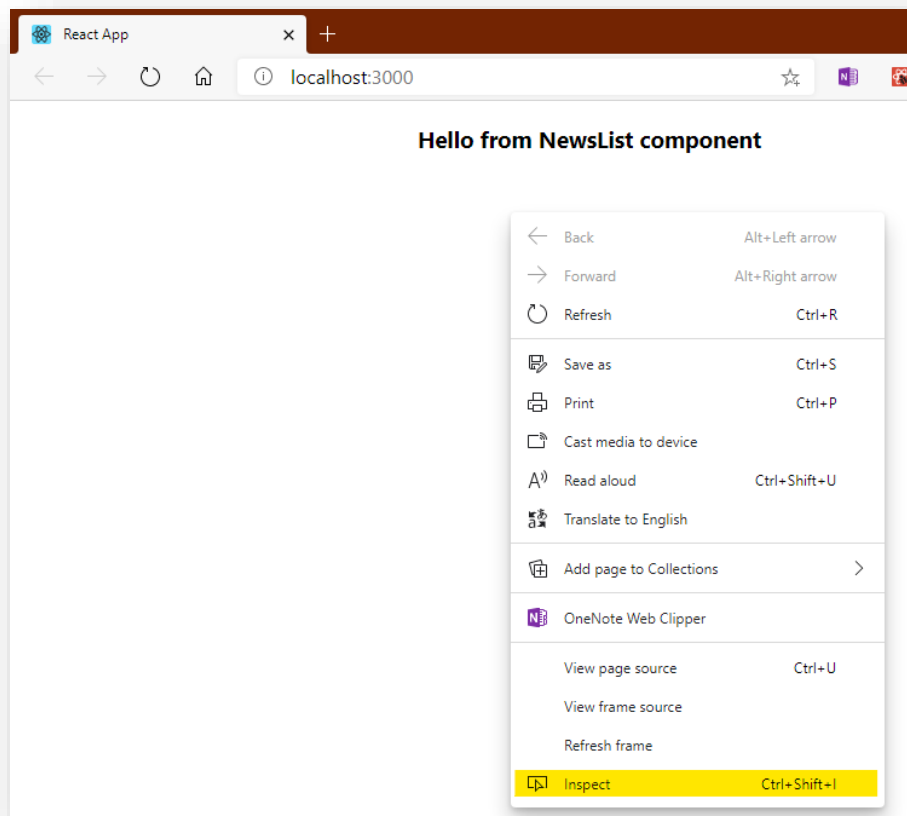
Important: below API key is not correct and you have to replace it with your key. So copy/paste to your project will not work!

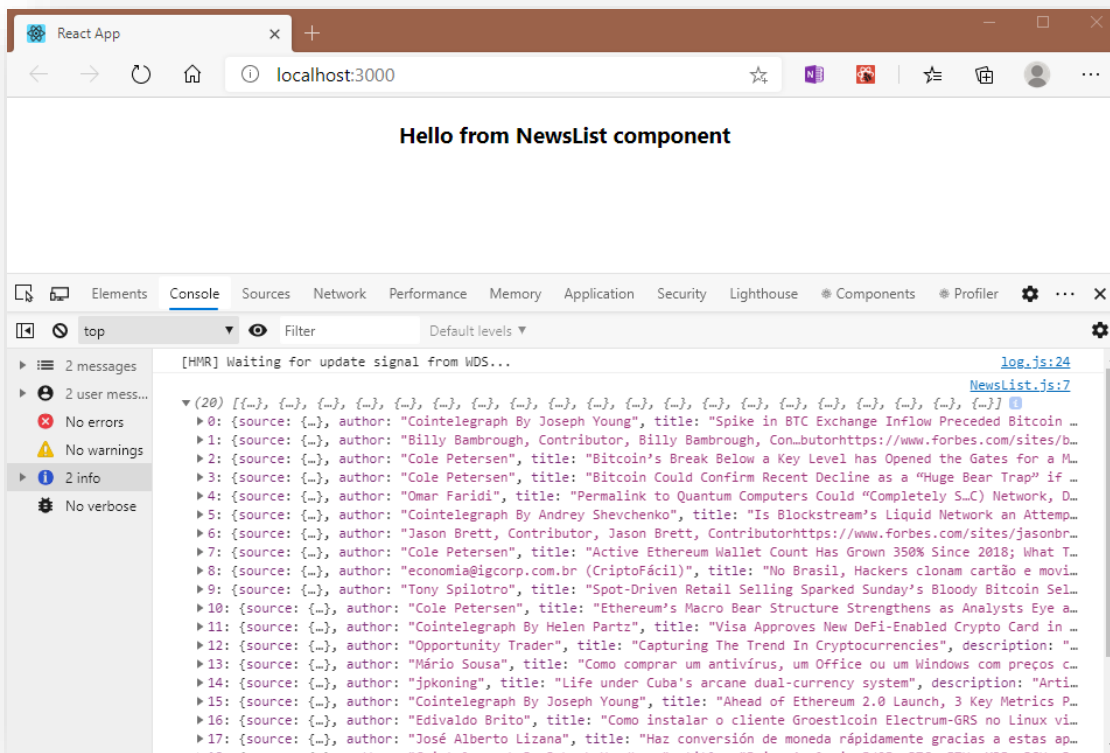
```
import React, {Component} from 'react';

class NewsList extends Component //inherits from Component
```

```
{
  componentDidMount ()
  {
    fetch ('http://newsapi.org/v2/everything?domains=wsj.com&apiKey=c338828f193a8335de')
      .then(response=>response.json())
      .then(data=>console.log(data.articles))
      .catch(error=>console.log(error));
  }
  render ()
  {
    return (
      <h3>Hello from NewsList component</h3>
    )
  }
}
export default NewsList; // export is necessary to make it visible to other components
```

Above code will write to the console of the web browser what we get from this web API. Let's try this code by saving NewsList.js file and open the console of the web browser by right clicking on the page and select **Inspect** and then go to Console tab.





There we go.. our client has got the data from the web API. Now let's save this result inside the state of this component.

To use state object you have to remember:

1. State can only be initialized inside the component class constructor
2. You have to call `super(props)` ;
3. You can immediately access the state object inside the constructor only using `this.state={}` statement
4. To change the state object outside the constructor use `this.setState({})` ; statement
5. Every time the state changes it triggers the render method of that component

Update the code of `NewsList` component as below.

```
import React, {Component} from 'react';

class NewsList extends Component //inherits from Component
{
  constructor(props)
  {
    super(props);
    this.state={
      newsList:[] // empty array
    }
  }

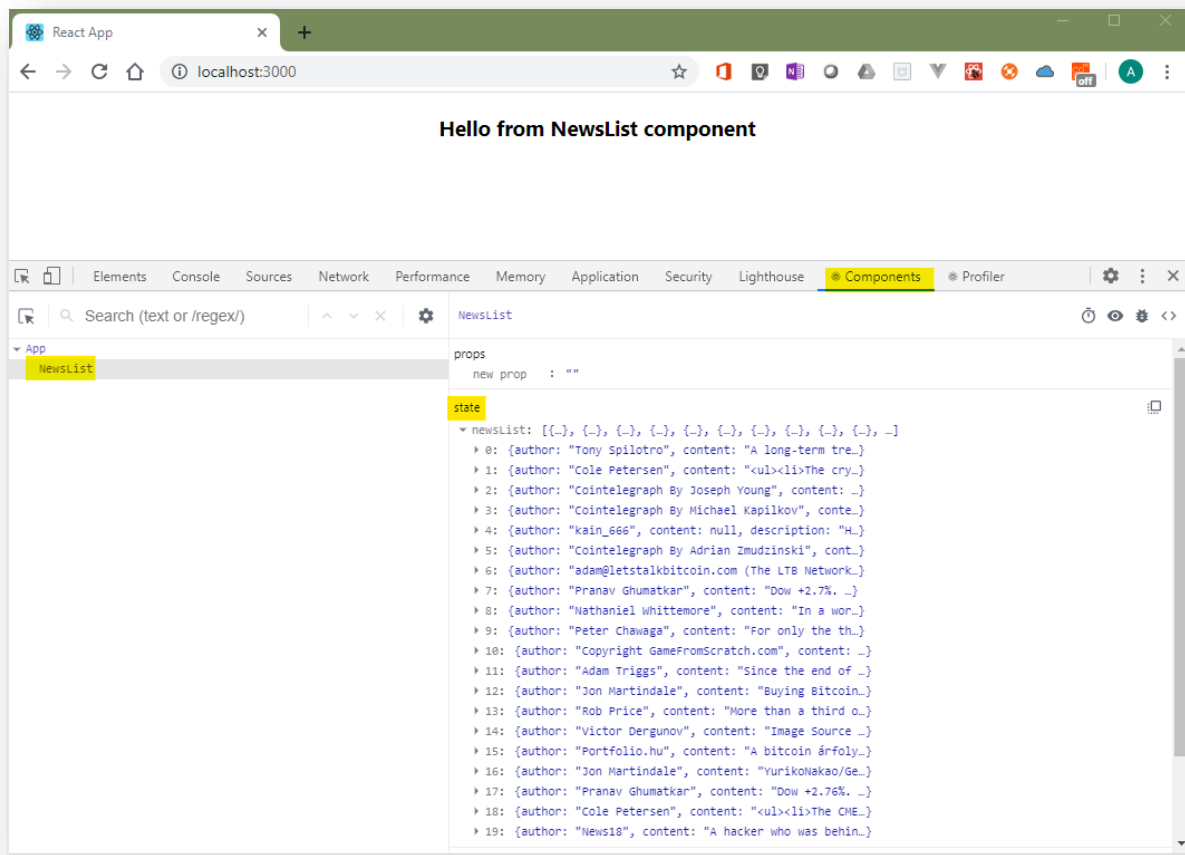
  componentDidMount ()
```

```
{
  fetch ('http://newsapi.org/v2/everything?domains=wsj.com&apiKey= c338828f193a8335de')
  .then(response=>response.json())
  .then(data=>this.setState({newsList:data.articles}))
  .catch(error=>console.log(error));
}

render ()
{
  return (
    <h3>Hello from NewsList component</h3>
  )
}
}

export default NewsList; // export is necessary to make it visible to other components
```

We've added the constructor method to initialize the state and updated the state in the `componentDidMount` method. Keep in mind that this will cause the render method to fire twice but this is not a problem. To check the outcome of this step we will use the React Development Tools we have installed previously. After saving the file we can open the components tab in the developer tools and see the articles are now stored in the state of the `NewsList` component.



The next step in this phase is to render this array in the state to our screen using a simple un ordered list. So, let's change the render method as below.

```
render ()
{
  return (
    <div>
      <h3>My News List</h3>
      {this.state.newsList.map((item, index, array)=>(
        <div key={index}>
          <a href = {item.url} target="_blank">{item.title}</a>
        </div>))}
    </div>
  )
}
```

We have rendered a simple list with news article title as a hyper link to the news url. Keep in mind that the render method must return only one root html element that is why we have to wrap everything in a **div tag**.

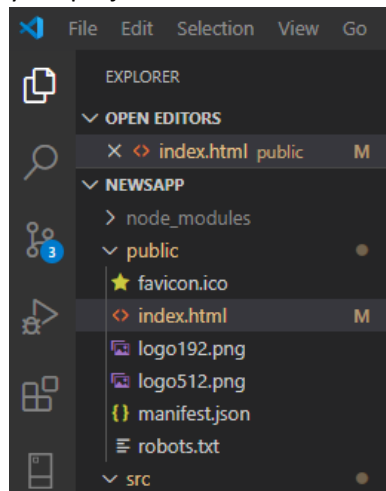
Two notes

1. We have used the map function which is a standard JavaScript array function that iterates an array. You can check [here](#) for more information on array iteration
2. Key is needed by React to know which item has changed in the DOM. Check [Lists and Keys](#) for better explanation

The final step in this phase is to make the `NewsList` component renders a better-looking news items with pictures, titles and links. We will use bootstrap 4 to render the articles array:

1. Bootstrap [grid](#) will determine how many news article will be displayed in one row for different screens
2. [Card component](#) is the best container for the news article

To use bootstrap, we should include bootstrap CDNs to index.html file, inside the head element. This file is located inside the public folder in your project.



Below is the complete code of the index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

    <!--bootstrap CDN-->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" integrity="sha384-9aIt2nRpC12Uk9gS9baD1411NQApFmC26EwAOH8WgZl5MYYYxFfc+NcPb1dKGj7Sk" crossorigin="anonymous">
```

```
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></scrip
t>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js" integrit
y="sha384-
OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI" crossorigin="anonymous"></scrip
t>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity=
"sha384-
Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></scrip
t>

<title>React App</title>
</head>
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root">
    <!--the App.js component and all other subcomponents are rendered here -->
  </div>
</body>
</html>
```

That's it... We can use bootstrap inside the render method in all React components!

Before proceeding in implementing the bootstrap cards and grid let's clean up our App.js component to clear any referenced styles. Those references were setup by `create-react-app` command.

```
import React , {Component} from 'react';
import NewsList from './NewsList'

class App extends Component {

  render () {
    return (
      <div>
        <NewsList />
      </div>
    );
  }
}

export default App;
```

Now update the render method of the NewsList.js component as below:

```
render ()
```

```
{
  return (
    <div className="container">
      <p className="h1 text-center text-primary">My News List</p>
      <div className="row">
        {this.state.newsList.map((item, index, array)=>(
          <div className="col-sm-6 col-md-4 col-lg-3 mb-3" key={index}>
            <div className="card" >
              <img src={item.urlToImage}
                className="card-img-top"
                alt={item.title}
                style={{height: 175 + 'px'}}/>
              <div className="card-body">
                <h5 className="card-title">{item.title}</h5>
                <p className="card-text">{item.description}</p>
                <a href={item.url}
                  className="btn btn-primary"
                  target="_blank">Read More...</a>
              </div>
            </div>
          </div>
        ))}
      </div>
    </div>
  )
}
```

Few notes about the code above:

1. We have used bootstrap grid with the div tags of row and col className attribute
2. The col is responsive based on the width of the screen. That is the beauty of bootstrap
3. Notice how the style attribute of the card image is written. The style attribute in react is different from the normal html. React uses JSX syntax and in order to write a style attribute we have to surround the value by double curly braces `{{}}`. You can check this [page](#) for more information

Now our page looks like below.. much better, right?

My News List



New Crypto Bull Run 'Inevitable' Says Data Analytics Company CEO

A new crypto bull run could be on its way as a result of government economic measures and mainstream interest.

[Read More...](#)



マイナンバーカードの未来？米国「SSN」の課題解決を狙うStilt、シード資金獲得

ピックアップ：Stilt, which provides financial services for immigrants, raises \$7.5 million seed round
ニュースサマリー：FinTechスタートアップ「Stilt」は9日、シードラウンドにて750万ドルの資金調達を実施したと発表した。投資家にはSteamlined Ventures、Bragiel Brother...

[Read More...](#)



数千人から選抜「45名のクリプト起業家」が学んでいるもの

ピックアップ：The Crypto Price-Innovation Cycle
ニュースサマリー：Andreessen Horowitz (a16z) は同社が運営するクリプトスタートアップ向けブートキャンプ「a16z Crypto Startup School」の一部コースをテーマごとに公開している。同社によれば、数千の応募者の内参加資格を得たのは45名だったという。2月終りから約2カ月にわたり...

[Read More...](#)



This Crypto Could Blow Bitcoin Out Of The Water This Year After 1,000% Gain—Here's Why

As a number of smaller cryptocurrencies and bitcoin rivals make big gains, one tiny token has soared over 1,000% since its March lows...

[Read More...](#)



Here's Why Ethereum's Bull Case Remains Strong Despite Decline to \$200

Ethereum has been underperforming Bitcoin and many of its other peers over the past couple of months, with its sharp decline from yearly highs of \$290 doing some damage to mid-term market structure. It now



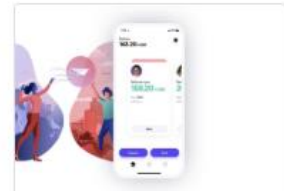
Bitcoin at Risk of Collapsing to \$7,000 as Selling Pressure Ramps Up

Bitcoin has been finding some slight support at \$8,800 throughout the past day, but the buying pressure here appears to be quickly degrading. The crypto is also on the cusp of breaking below a technical formation



\$35/OUNCE GOLD: Don't Blink – Just Hit the MOTHER LODE!

On August 15th, 1971, President Nixon changed the MONETARY LANDSCAPE forever. For the first time in human history, following his televised announcement, the entire global economy shifted from currencies that are BACKED BY GOLD to ones



Calibra, el monedero digital de Facebook, pasa a ser Novi y surgen detalles de cómo funcionará con WhatsApp para transferir dinero

Calibra se presentó en junio de 2019 junto a Libra, la criptomoneda respaldada por Facebook y otros socios. Mientras que la segunda es la

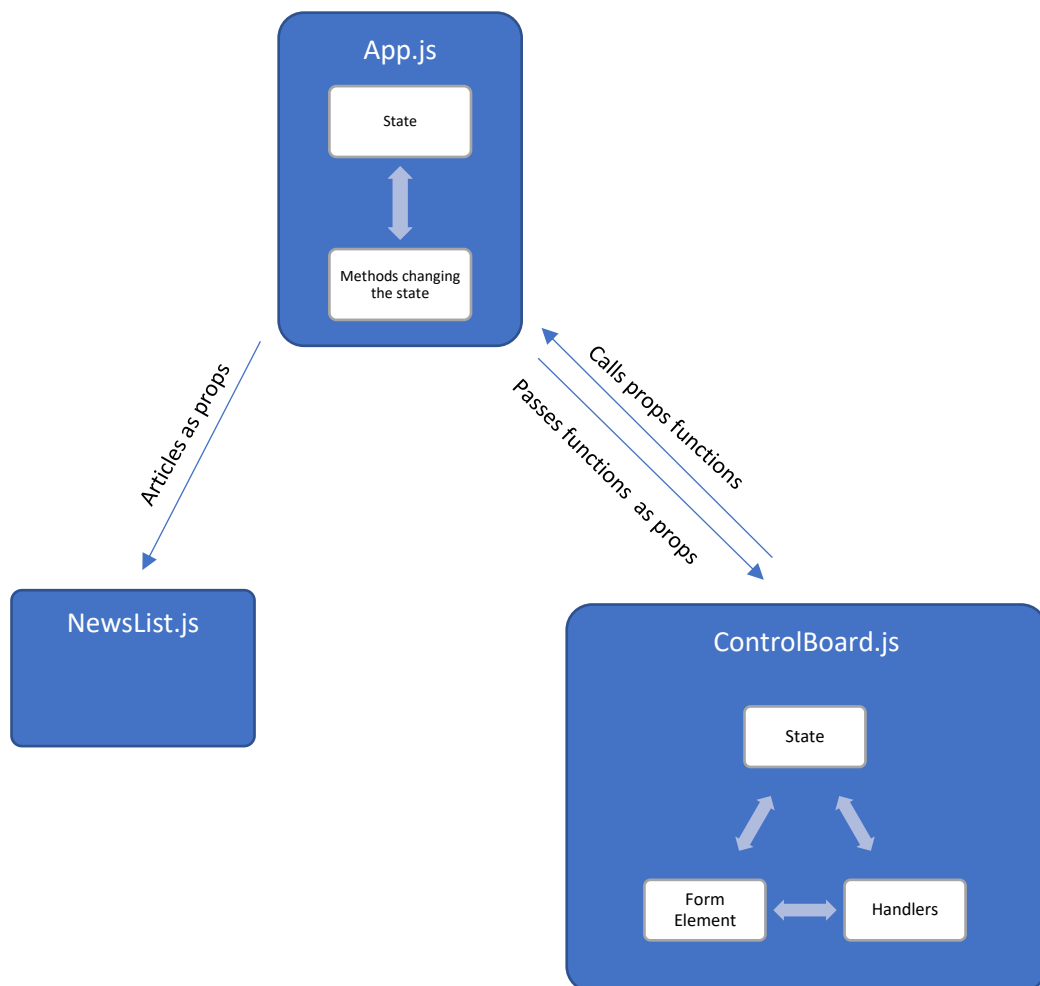
Phase 3 – Controlling what to display

We have built our simple react app with a single url which is hard coded inside our `NewsList` component. Now it is the time to check other end points of the web API and select what we want to see. Below are the steps of this final phase of our application.

What we will do

1. Use Top headlines end point of the web API
2. Create another component which allows us to select what country and language to display and one search box. This name of this component will be `ControlBoard` and it will be on top of our page acting like a navigation bar. It will update the `App` component about the user's input via props functions
3. The `App` component will determine the url, fetches the API, store the articles in the state and pass the articles to `NewsList` component
4. `NewsList` will only display what is gets from the `App` component and therefore there will be no need for a state in this component

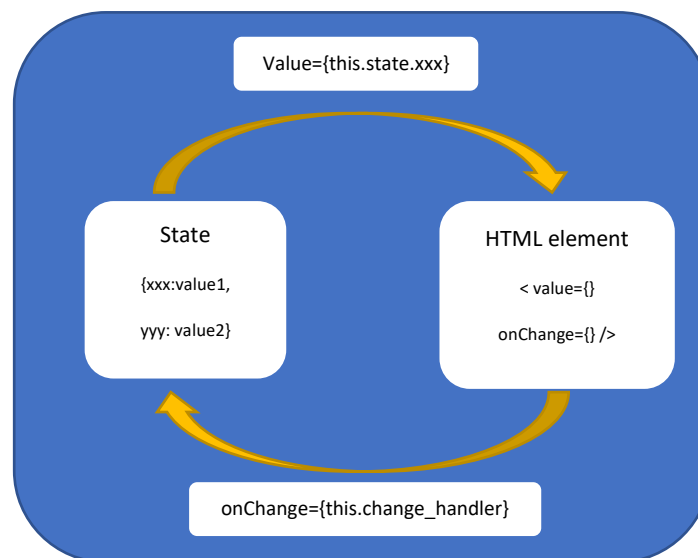
Below is the outcome of phase 3



Let's get started!

Creating the ControlBoard component

1. Create a new file in *src* folder and name it *ControlBoard.js*. This component will include a form. When dealing with forms we have to remember one important thing, we have to make the state of that component as the single source of truth. To achieve this we have to bind the forms elements values to the state object by
 - a. Setting the value property of the element to a filed of the state object
 - b. Creating change handlers of each form element. The purpose of those handler is to update the state object



2. Insert below code to the file and save it

```
import React, {Component} from 'react';

class ControlBoard extends Component
{
  constructor(props)
  {
    super (props);
    this.state={
      country: 'us',
      category: 'general',
      query: ''
    }
  }
}
```

```
handleChange = (e) => this.setState({[e.target.name]: e.target.value})
handleClick = (e) => {
  e.preventDefault();
  this.props.search(this.state.country, this.state.category, this.state.query);
}

render ()
{
  return (
    <nav className="navbar navbar-light bg-light">
      <form className="form-inline">
        <select name = "country"
          className="form-control mr-sm-2"
          value={this.state.country}
          onChange={this.handleChange}>
          <option value="us">USA</option>
          <option value="ca">Canada</option>
          <option value="gb">UK</option>

        </select>

        <select name = "category"
          className="form-control mr-sm-2"
          value={this.state.category}
          onChange={this.handleChange}>
          <option value="general">All</option>
          <option value="business">Business</option>
          <option value="entertainment">Entertainment</option>
          <option value="health">Health</option>

        </select>

        <input className="form-control mr-sm-2"
          type="search" placeholder="Search"
          aria-label="Search" name="query"
          value={this.state.query}
          onChange={this.handleChange} />

        <button className="btn btn-outline-success my-2 my-sm-0"
          type="button" onClick={this.handleClick}>Search</button>
      </form>
    </nav>
  )
}
```



```
export default ControlBoard;
```

Updating the App.js component

3. Now let's add above component to our root component (App.js). Don't forget to save the file.

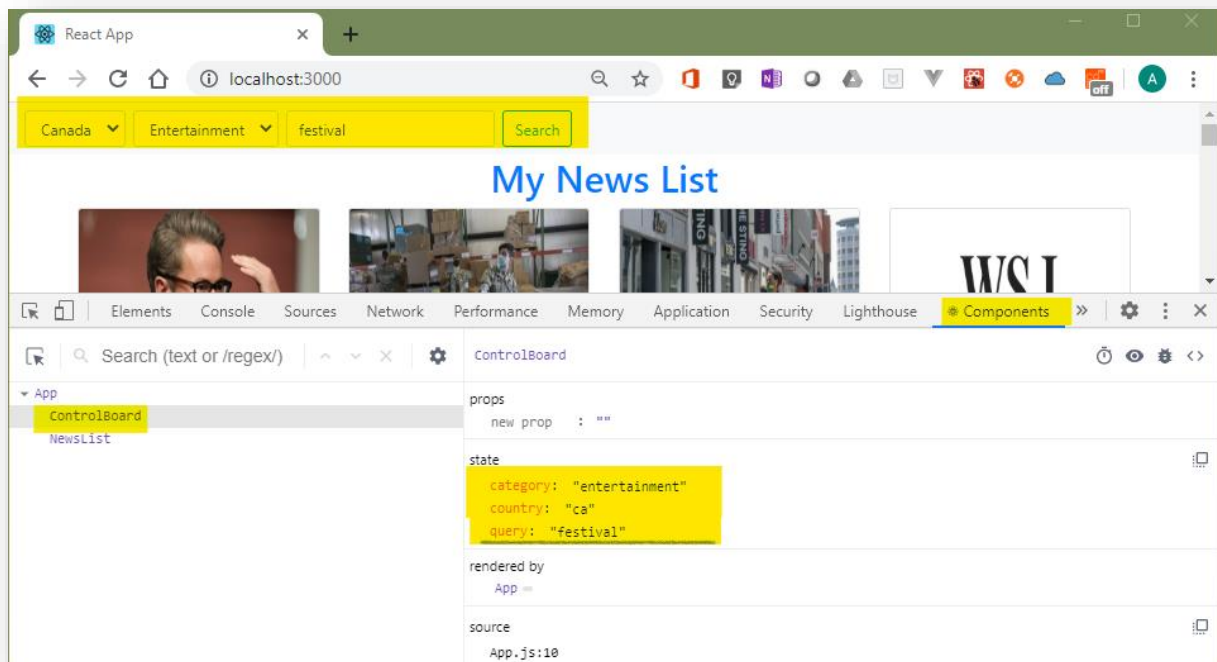
```
import React , {Component} from 'react';
import NewsList from './NewsList'
import ControBoard from './ControlBoard';

class App extends Component {

  render () {
    return (
      <div>
        <ControBoard />
        <NewsList />
      </div>
    );
  }
}

export default App;
```

4. Now let's check how this form is bound to the state
 - a. Right click on the page and click on inspect if not opened already
 - b. Try to change the selections in the form and make sure how they are updating the state object



5. Now let's add a state in the App.js, define a method that updates the state and send it as a prop to `ControlBoard` component and pass the articles array to `NewsList` component

```
import React, {Component} from 'react';
import NewsList from './NewsList'
import ControlBoard from './ControlBoard';

class App extends Component {
  constructor(props)
  {
    super(props);
    this.state={url:'https://newsapi.org/v2/top-headlines?country=us&category=general&q=&apiKey=c338828f193a8335de',
      newsList:[]}
  }

  updateUrl = (country, category, q) => {
    let newUrl = 'https://newsapi.org/v2/top-headlines?country='+country+'&category='+category+'&q='+q+'&apiKey=c801d58828424f19af23a86ebd0a35de';
    fetch (newUrl)
      .then(response=>response.json())
      .then(data=> this.setState({newsList: data.articles, url: newUrl}))
  }
}
```

```
        .catch(error=>console.log(error));
    }

    componentDidMount()
    {
        fetch (this.state.url)
            .then(response=>response.json())
            .then(data=> this.setState({newsList: data.articles}))
            .catch(error=>console.log(error));
    }

    render () {
        return (
            <div>
                <ControBoard search={this.updateUrl} />
                <NewsList articles={this.state.newsList} />
            </div>
        );
    }
}

export default App;
```

the state has been initialized in two different places: the url has been initialized in the constructor while the articles array is initialized later, in the `componentDidMount` life cycle method. Notice how props are passed to `ControlBoard` and `NewsList` components

Updating the `NewsList.js` component

5. And finally let's update the `NewsList` component to consider the newly passed prop

```
import React, {Component} from 'react';

class NewsList extends Component //inherits from Component
{

    render ()
    {

        return (
            <div className="container">
                <p className="h1 text-center text-primary">My News List</p>
                <div className="row">
                    {
                        this.props.articles.map((item, index, array)=>(
                            <div className="col-sm-6 col-md-4 col-lg-3 mb-3" key={index}>
```

```
        <div className="card" >
          <img src={item.urlToImage}
            className="card-img-top"
            alt={item.title}
            style={{height: 175 + 'px'}}/>
          <div className="card-body">
            <h5 className="card-title">{item.title}</h5>
            <p className="card-text">{item.description}</p>
            <a href={item.url}
              className="btn btn-primary"
              target="_blank" rel="noopener noreferrer">Read More...</a>
          </div>
        </div>
      </div>))
    }
  </div>
</div>
)
}
}

export default NewsList; // export is necessary to make it visible to other components
```

Since there is no state in this component, we can skip the constructor. `ComponentDidMount` method is not needed anymore and the `render` method is reading from the `articles` array sent by the `App` component.

Homework

1. Try to update the `ControlBoard` component to call other end points. Check what possible query strings for each of those end points and modify the html form elements to fit the requirements
2. Since `NewsList` does not need a state rewrite it using [function components](#)
3. Try to make the page look nicer. Make the cards have equal sizes by limiting the number of characters of title and description:
 - a. When exceeding certain limit add '...' to the end
 - b. Suffix spaces when not exceeding the limit

Thank you!