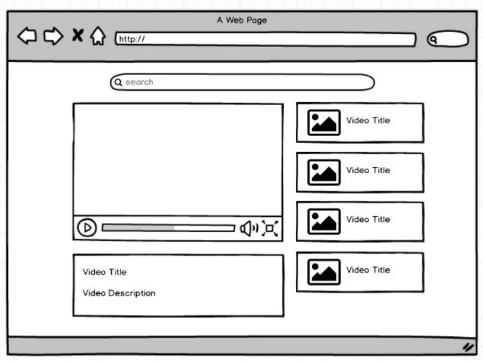
Youtube Player in ReactJS

This document provides a step-by-step explanation of how to create a Youtube player in ReactJS. To follow through the instruction easily, code snips and snips of localhost UI are provided. You will also find some theory, for example:

- Difference between functional and class based component
- Use of an arrow function
- Definition and use of State
- Definition and use of Controlled Field
- Use of Downwards data flow
- Use of Props
- Definition and use of Key
- Handling Null Process
- Use of callback

1. Make **Component Structure**



Make **Component Structure** for the App

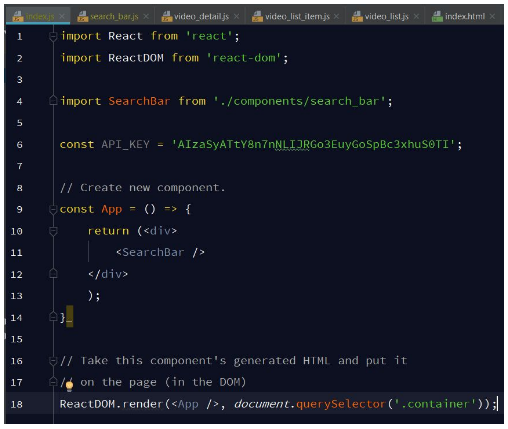
2. Get **Youtube API**

from https://console.developers.google.com/
Insert API in your code in index.js
Download and install the package that helps to do the search request

```
npm install --save youtube-api-search
```

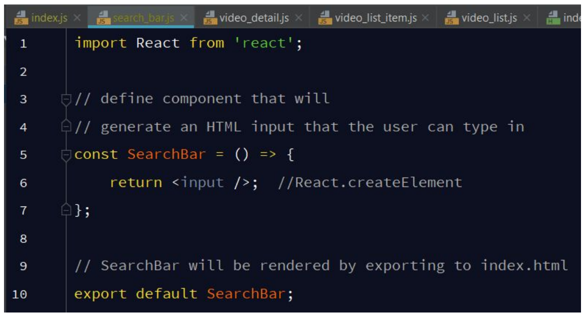
3. SearchBar component

3.1. Create, export and render the **SearchBar component**. It's **functional** component because it's literally is a function.



```
import React from 'react';
import ReactDOM from 'react-dom';

import SearchBar from './components/search_bar';

const API_KEY = 'AIzaSyATtY8n7nNLIJRGo3EuyGoSpBc3xhuS0TI';

// Create new component.
const App = () => {
    return (<div>
        <SearchBar />
    </div>
    );
}

// Take this component's generated HTML and put it
// on the page (in the DOM)
ReactDOM.render(<App />, document.querySelector('.container'));
```

Index.js

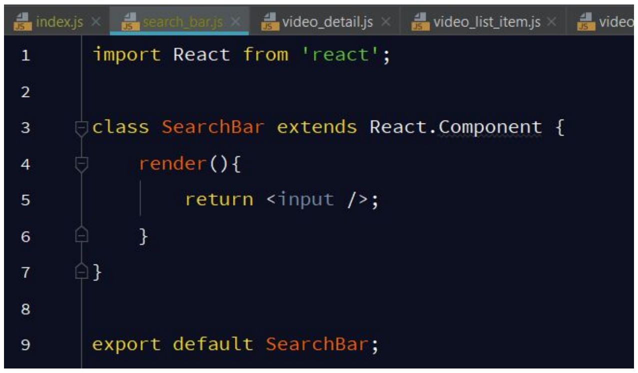


```
import React from 'react';

// define component that will
// generate an HTML input that the user can type in
const SearchBar = () => {
    return <input />;  //React.createElement
};

// SearchBar will be rendered by exporting to index.html
export default SearchBar;
```

Search_bar.js **functional** component

3.2. Refactor the SearchBar component from a **functional** component to a **class based** component using ES6 class.

a) Declare a new JS class
b) Give this class all of the functionality that React.Component has
c) Give the class the ability to be rendered
d) Return some JSX from the render method



```
import React from 'react';

class SearchBar extends React.Component {
    render(){
        return <input />;
    }
}

export default SearchBar;
```

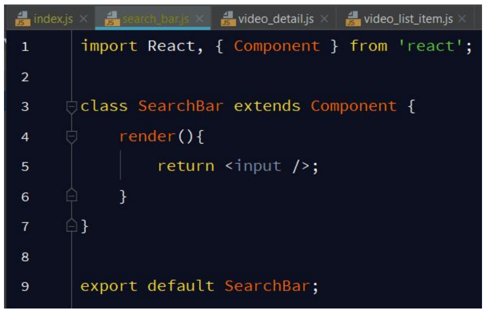
Search_bar.js **class based** component

A **functional** component can contain a **class** component.

3.3. Clean up some of the code by using more ES6 syntax

<u>Why did we refactor from **functional** to a **class based** component?</u>
Because we want this component to communicate with all the other components that we're going to be creting, to say "Hey, the user just typed in the input and here is what they have just typed"

<u>**Functional** vs **class based** component</u>
Start with functional, and only if you decide that you need add functionality - you start to refactor it to a class.



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

class SearchBar extends Component {
    render(){
        return <input />;
    }
}

export default SearchBar;
```

Clean Search_bar.js **class based** component

4. Handling User Events

When a user types something we want to know:
→ that they type something
→ what they type

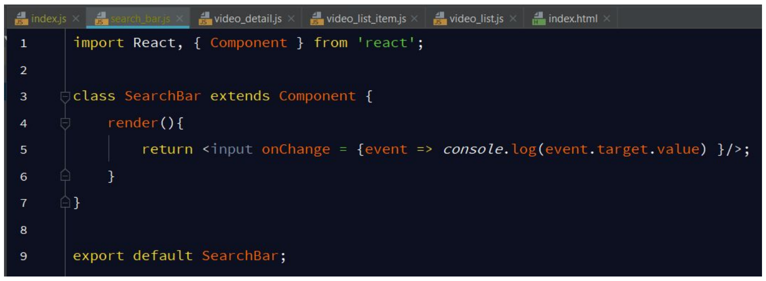
Handling Events in React has 2 steps :
1 step. Declare an event handler
2 step. Pass the event handler to the element that we want to monitor for events

Process:

In our case, we want to **know** whenever the input element inside of our search bar has its **text changed**.
We have an input element and we want to add an event handler to it.
The event handler will be triggered whenever the event occurs, specifically we want to watch for the change event on input . So, to get access to the Change input we write a react special property - onChange.
Then we're writing the event handler = {this.onInputChange}.
Then inside of the class we define another method for the event handler and we choose to add the argument here - event , because most event handlers with this event object. And we can use this event object to get more information about the event that just occured.

```
onInputChange(event){
  console.log(event);
}
```

Use an **arrow function** to make the code look a little bit more compact.
By using the **arrow function** we can clean up our code a lot!



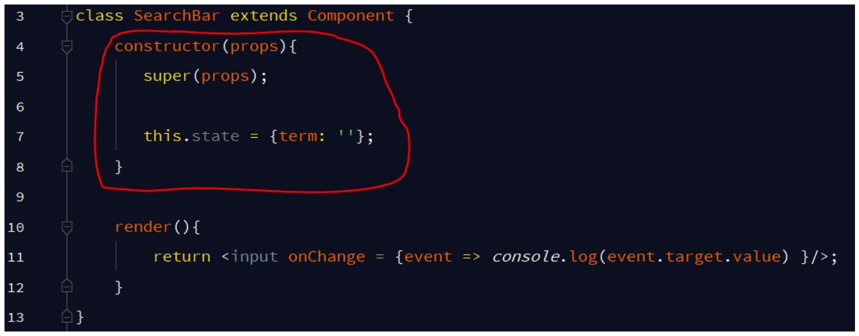
```
import React, { Component } from 'react';

class SearchBar extends Component {
    render(){
        return <input onChange = {event => console.log(event.target.value) }/>;
    }
}

export default SearchBar;
```

Handling User Event in SearchBar component

5. State

State is a plain Javascript Object that is used to **record and react to user events**. Each **class based component** has its **own state** (*functional components do not have state). Whenever a component's state is changed the component immediately **re-renders** and also forces all of its children re-render as well.

Before we ever use state inside a component, we need to **initialize the state object**.
To initialize state we state the property state to a plain Javascript object inside of the class's constructor method.



```
class SearchBar extends Component {
    constructor(props){
        super(props);

        this.state = {term: ''};
    }

    render(){
        return <input onChange = {event => console.log(event.target.value) }/>;
    }
}
```

FIGURE 8 Here we define state or initialize state

Summary of the figure 8:

All Javascript classes have a special function called **Constructor**. The Constructor function is the **first and only function called automatically** whenever the new instance (э к з е м п л я р) of the class is created.

this.state = { term: '' };
Whenever we use state we initialize it by creating a new object- { term: '' } and assigning it to this.state. The **object** we pass will also contain properties (term - short for search term in our case) that we want to record to the state.

So, as a user starts typing inside the input, we want to update this.state.term to not be empty string but to be the value of the input (console.log).

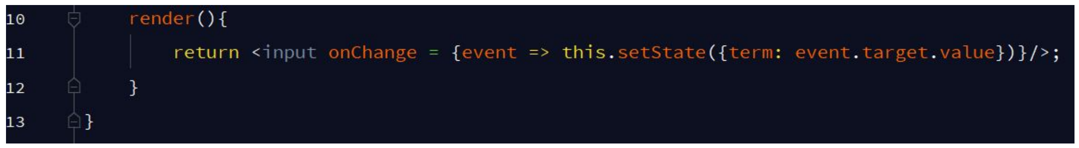
5.1. Recording the value of our input to the state

To do so, we have to update the state inside of the function handler (console.log(event.target.value)). So instead of doing console.log we are going to update our state.

How to **MANIPULATE** STATE ?

Updating the state is a little bit different than creating it.

So only inside of the Construction function do we change our state like this : this.state = { term: '' }; Everywhere else inside of all out Components we instead of using this.state = { term: '' }; will use the method called this.setState().
We pass there an object - {term: event.target.value} ,that contains a new state that we want to give our Component.



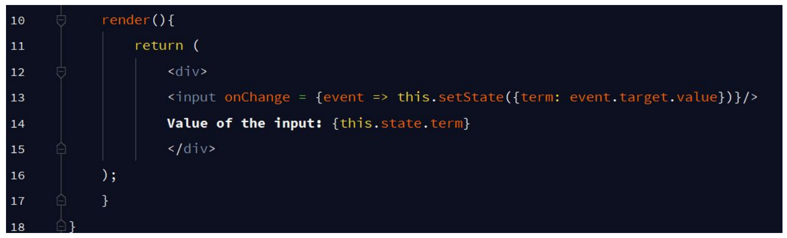
```
    render(){
        return <input onChange = {event => this.setState({term: event.target.value})}/>;
    }
}
```

FIGURE 9 Manipulating state

this.state.term = *event*.target.value //BAD - Never do that! We always manipulate our state with this.setState(). Using this.setState() allows us to maintain continuity.
So if we just change the value, React does not really know that the value changed. So instead we use this.setState() method to inform React that the state is changing and here is what the new state is.

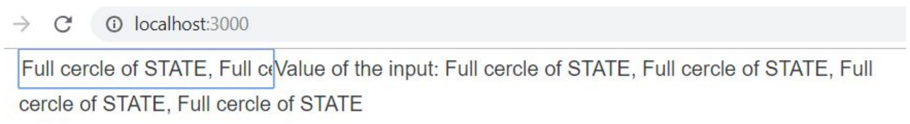
Full cercle of STATE (see figure 10):

1- Whenever we update the input element, whenever we change the value of it.
2 - event => this.setState({term: event.target.value}) function runs because its our event handler.
3- We set the state (this.setState({term: ... })) with a new new value of the input - event.target.value
4- Whenever we update our state, whenever we call this.setState it causes our component automatically re-render and push all this updated information from the render method into into the DOM.
5- Because our render method makes reference to this.state.term every time that the Component re-renders - we get the updated this.state.term in the DOM.



```
    render(){
        return (
            <div>
            <input onChange = {event => this.setState({term: event.target.value})}/>
            Value of the input: {this.state.term}
            </div>
    );
    }
}
```

FIGURE 10 Render Value of input

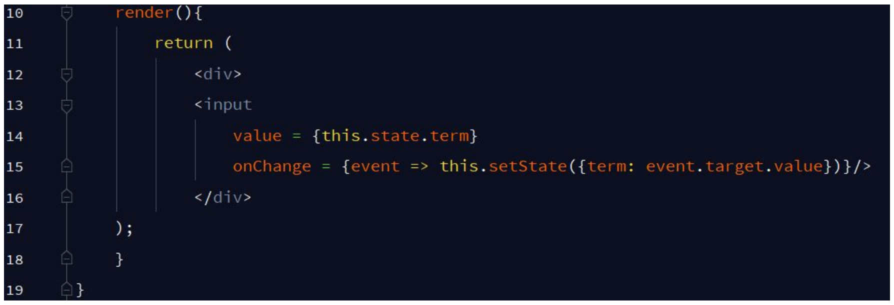

localhost:3000

Full cercle of STATE, Full ce Value of the input: Full cercle of STATE, Full cercle of STATE, Full cercle of STATE, Full cercle of STATE

FIGURE 11 Render Value of input (localhost UI)

5.2. Controlled Component

Controlled Field - is a form element like an input whose value is set by the state rather than the other way around. So its value only ever changes when its state changes.



```
    render(){
        return (
            <div>
            <input
                value = {this.state.term}
                onChange = {event => this.setState({term: event.target.value})}/>
            </div>
        );
    }
}
```

FIGURE 11

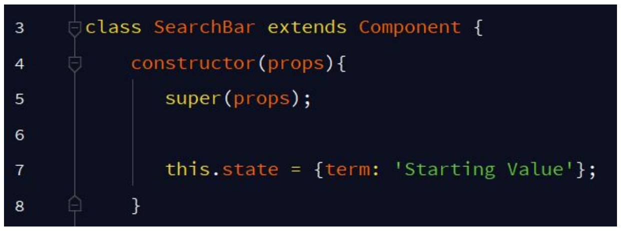
When we add value = {this.state.term} we turn input to a Controlled Component.

When the user types something - it does not update the state but trigger an event. Because we updated the state with this event - it causes the value the input to change.

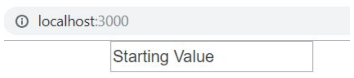
the value of the input = state

Why we use Controller Component ?

1) It allows us to have default input inside our input. Not a placeholder but an actual value pre-populate there.



```
class SearchBar extends Component {
    constructor(props){
        super(props);

        this.state = {term: 'Starting Value'};
    }
```

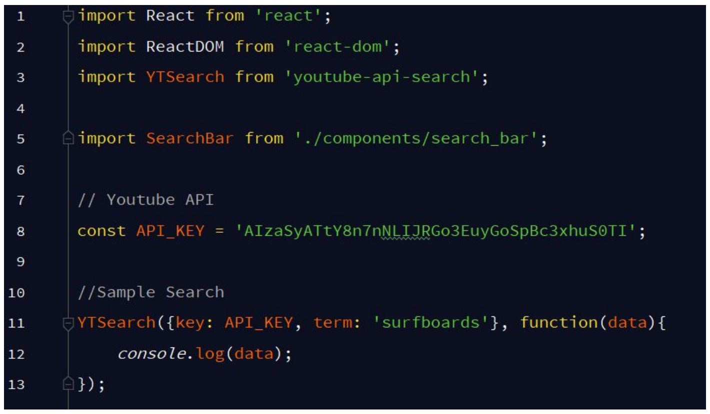

localhost:3000

Starting Value

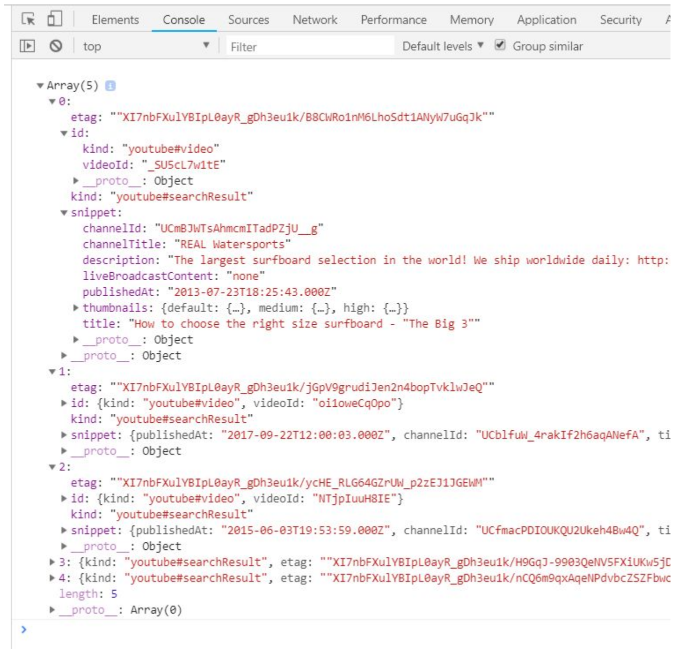
2) Allows to read the value of the input very easily then if we had used jquery

6. Youtube Search Response
    1) make call to Youtube API to get some info
    2) spread the info throughout the entire App

Downwards data flow - only the most parent Component in the application should be responsible for fetching data. Be it from API or flux framework or redux itself.
In our case index is the most parent component that we have.



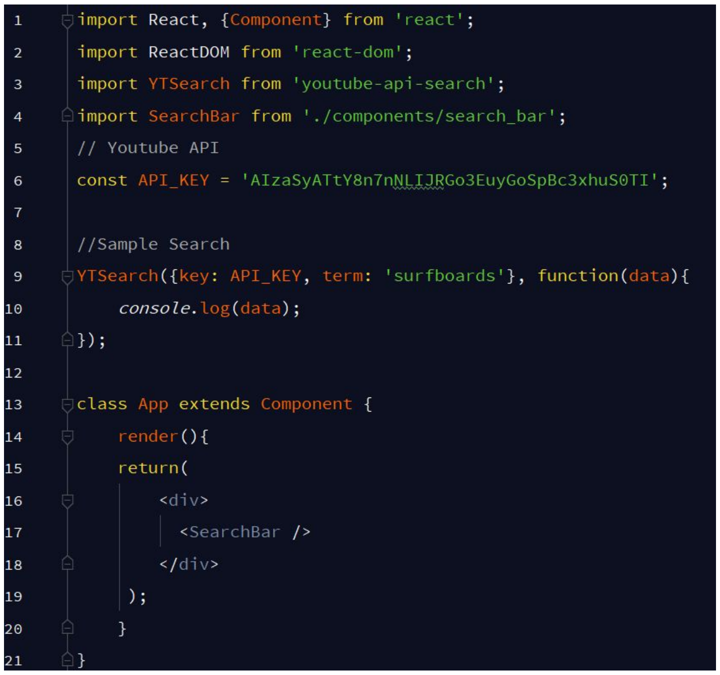
```
import React from 'react';
import ReactDOM from 'react-dom';
import YTSearch from 'youtube-api-search';

import SearchBar from './components/search_bar';

// Youtube API
const API_KEY = 'AIzaSyATtY8n7nNLIJRGo3EuyGoSpBc3xhuS0TI';

//Sample Search
YTSearch({key: API_KEY, term: 'surfboards'}, function(data){
    console.log(data);
});
```



```
▼Array(5)
  ▼0:
     etag: ""XI7nbFXulYBIpL0ayR_gDh3eu1k/B8CWRo1nM6LhoSdt1ANyW7uGqJk""
   ▼id:
       kind: "youtube#video"
       videoId: "_SU5cL7w1tE"
     ▶__proto__: Object
     kind: "youtube#searchResult"
   ▼snippet:
       channelId: "UCmBJWTsAhmcmITadPZjU__g"
       channelTitle: "REAL Watersports"
       description: "The largest surfboard selection in the world! We ship worldwide daily: http:
       liveBroadcastContent: "none"
       publishedAt: "2013-07-23T18:25:43.000Z"
     ▶thumbnails: {default: {…}, medium: {…}, high: {…}}
       title: "How to choose the right size surfboard - "The Big 3""
     ▶__proto__: Object
   ▶__proto__: Object
  ▼1:
     etag: ""XI7nbFXulYBIpL0ayR_gDh3eu1k/jGpV9grudiJen2n4bopTvklwJeQ""
   ▶id: {kind: "youtube#video", videoId: "oi1oweCqOpo"}
     kind: "youtube#searchResult"
   ▶snippet: {publishedAt: "2017-09-22T12:00:03.000Z", channelId: "UCblfuW_4rakIf2h6aqANefA", ti
   ▶__proto__: Object
  ▼2:
     etag: ""XI7nbFXulYBIpL0ayR_gDh3eu1k/ycHE_RLG64GZrUW_p2zEJ1JGEWM""
   ▶id: {kind: "youtube#video", videoId: "NTjpIuuH8IE"}
     kind: "youtube#searchResult"
   ▶snippet: {publishedAt: "2015-06-03T19:53:59.000Z", channelId: "UCfmacPDIOUKQU2Ukeh4Bw4Q", ti
   ▶__proto__: Object
  ▶3: {kind: "youtube#searchResult", etag: ""XI7nbFXulYBIpL0ayR_gDh3eu1k/H9GqJ-9903QeNV5FXiUKw5jD
  ▶4: {kind: "youtube#searchResult", etag: ""XI7nbFXulYBIpL0ayR_gDh3eu1k/nCQ6m9qxAqeNPdvbcZSZFbwc
   length: 5
  ▶__proto__: Array(0)
>
```

7. Refactoring Functional Component to Class Component

We do refactoring to make sure that the App can keep track of the list of videos by recording them on it's state.

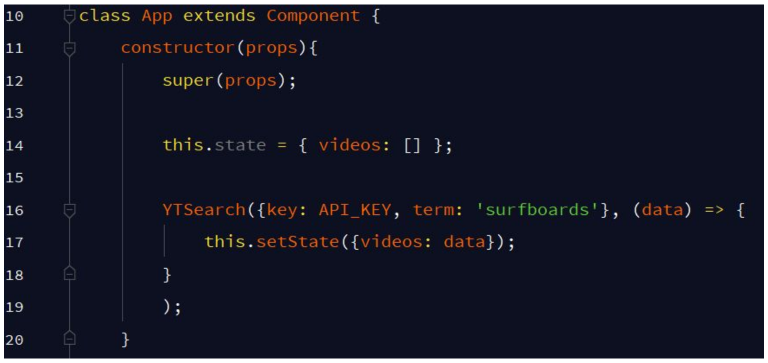


```
import React, {Component} from 'react';
import ReactDOM from 'react-dom';
import YTSearch from 'youtube-api-search';
import SearchBar from './components/search_bar';
// Youtube API
const API_KEY = 'AIzaSyATtY8n7nNLIJRGo3EuyGoSpBc3xhuS0TI';

//Sample Search
YTSearch({key: API_KEY, term: 'surfboards'}, function(data){
    console.log(data);
});

class App extends Component {
    render(){
    return(
        <div>
          <SearchBar />
        </div>
     );
    }
}
```

So our list of videos is going to be changed over time. Right now it's a list of 5 videos but as ppl search for data or as the user types stuff in the search we want to update it with the new list of videos.

Data changing over time.

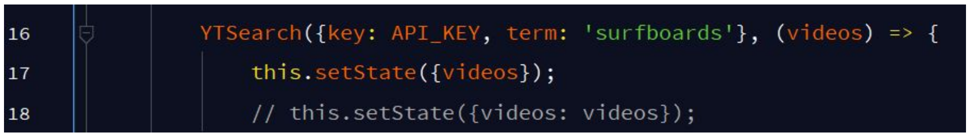
-it's a great use for **state**. So whenever user enters some new search input, we need to conduct a new search and set the result of that search to on state.
This means we have to set up **constructor** function:



```
class App extends Component {
    constructor(props){
        super(props);

        this.state = { videos: [] };

        YTSearch({key: API_KEY, term: 'surfboards'}, (data) => {
            this.setState({videos: data});
        }
        );
    }
```

So now,

1-whenever the App first boots up and we get an instance of App and we get an instance of App on the screen
2-the constructor will run right away because we make a new instance of App and that will immediately kick of the search with term: 'surfboards' (16)
3-the fat arrow function will be called with the list of videos (18)
4-this.setState updates with the list of videos (17)

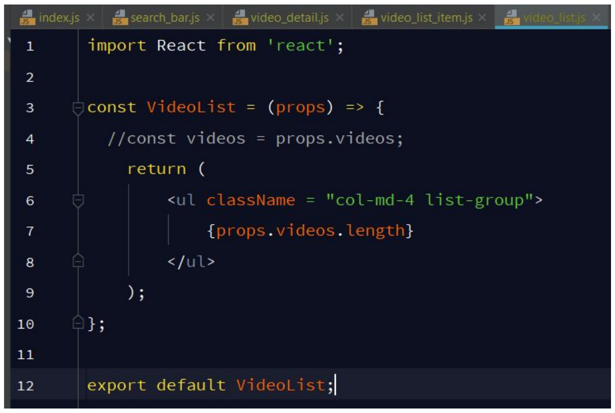


```
16        YTSearch({key: API_KEY, term: 'surfboards'}, (videos) => {
17            this.setState({videos});
18            // this.setState({videos: videos});
```

If the key and the property are the same variable name we can use advansed JSX syntax (like here videos: videos).

8. Props

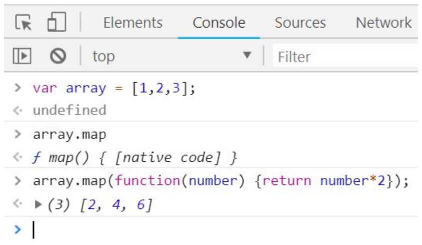
In a **functional component** the **props object** is an **argument**.
In a **class based component** - **props** are available in anywhere, in any method we define as **this.props**.



```
import React from 'react';

const VideoList = (props) => {
  //const videos = props.videos;
    return (
        <ul className = "col-md-4 list-group">
            {props.videos.length}
        </ul>
    );
};

export default VideoList;
```

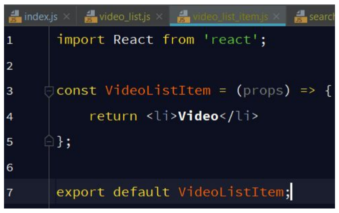
So, we now have a functional video list component and it have an array of videos to render. Next we will have to add one video list item per video. For this we will loop over this array:props.videos. And for each video we will generate one instance of video list item.

9. Building List with Map

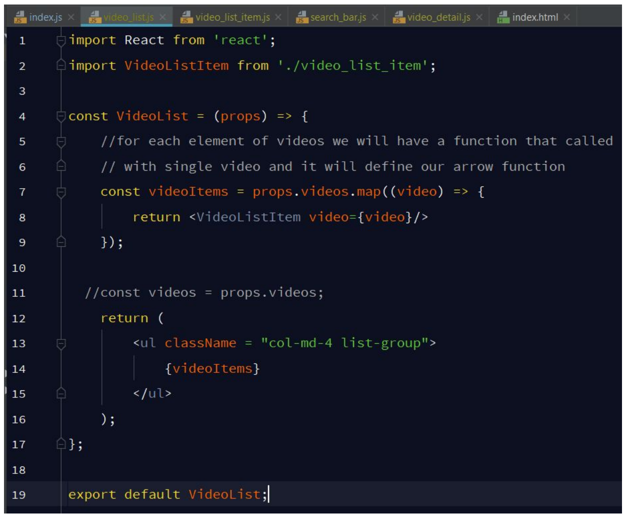


```
> var array = [1,2,3];
< undefined
> array.map
< ƒ map() { [native code] }
> array.map(function(number) {return number*2});
< ▶(3) [2, 4, 6]
> |
```

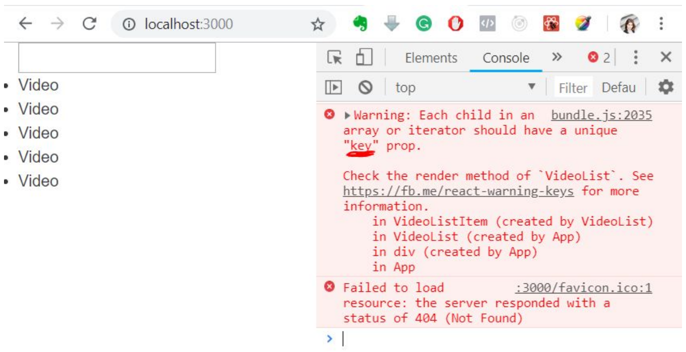
Using Map function for an array



```
import React from 'react';

const VideoListItem = (props) => {
    return <li>Video</li>
};

export default VideoListItem;
```



```
import React from 'react';
import VideoListItem from './video_list_item';

const VideoList = (props) => {
    //for each element of videos we will have a function that called
    // with single video and it will define our arrow function
    const videoItems = props.videos.map((video) => {
        return <VideoListItem video={video}/>
    });

  //const videos = props.videos;
    return (
        <ul className = "col-md-4 list-group">
            {videoItems}
        </ul>
    );
};

export default VideoList;
```



- Video
- Video
- Video
- Video
- Video

```
▶Warning: Each child in an   bundle.js:2035
array or iterator should have a unique
"key" prop.

Check the render method of `VideoList`. See
https://fb.me/react-warning-keys for more
information.
    in VideoListItem (created by VideoList)
    in VideoList (created by App)
    in div (created by App)
    in App
Failed to load          :3000/favicon.ico:1
resource: the server responded with a
status of 404 (Not Found)
> |
```

10. List Item Keys

We got an error on the step 9 because whenever we render an array of items of the same type React correctly assumes that we're building a list.

Example with cards:

Imaging you have a stack of note cards that consist of some important info printed on each of them. Now imagine that every 30 seconds someone came to you and notify that at middle of the desk a very particular card needed its info updated. But they did not know exactly which card it was (some card). The only option is to through away the entire stack and to recreate all of the index cards.
You would have to recreate the entire list.
*If you have an **ID** of the cards however the process will happen much faster!*

The same (as described above) happening with React when it's trying to render a list.
It builds a list if it has an ID for a particular element. And when that element changes it knows which element it needs to grab and update.

Add a key.
Key - a consistent and unique piece of information. Unique to the particular record.

In our case each video has **etag**.

To a key, we just define it as a property: `key= {video.etag}`

Result: No error!

11. Video List Item

Here we will make the list of videos look better then they do now.

1)Define a new variable `const video` to pull of the video from the **props** object.
Remember , inside the video_list.js when we created the videoListItem, we passed in the video as property video:
```
return <VideoListItem key= {video.etag} video={video}/>
```

So to get access to the video inside of video_list_item.js we have to say `props.video`

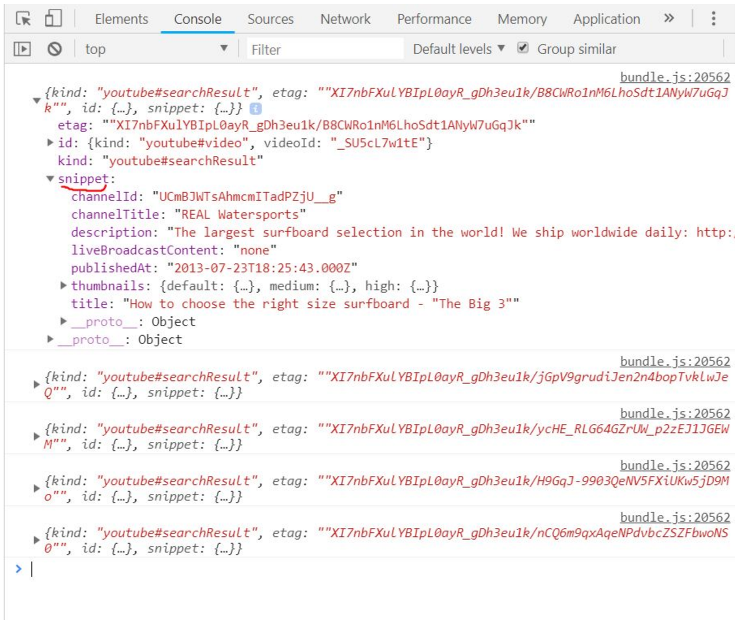
```
const VideoListItem = ({props}) => {

    const video = props.video;

  return <li>Video</li>
};
```

=

JSX Syntax

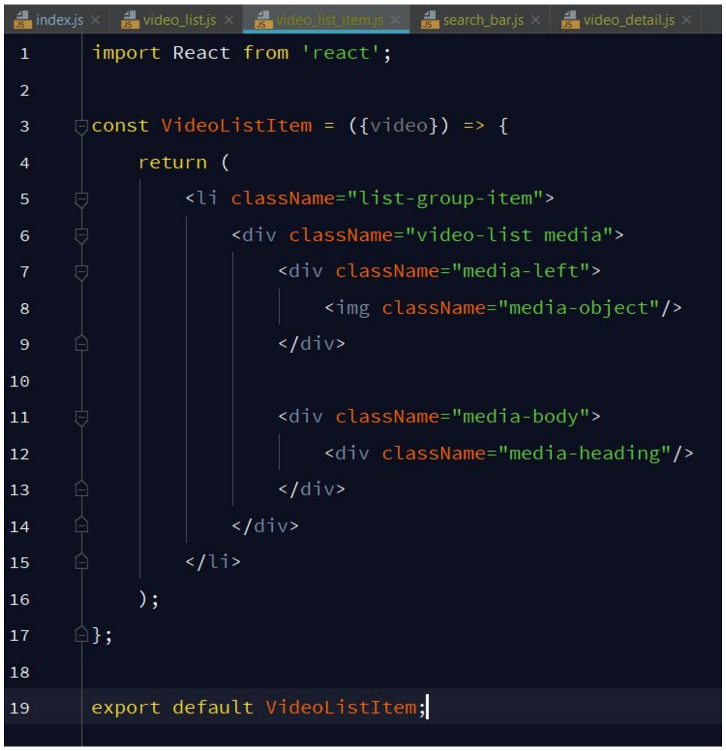
```
const VideoListItem = ({video}) => {

  return <li>Video</li>
};
```

What is going on inside the video?

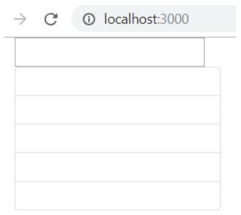


```
                                                                        bundle.js:20562
▼ {kind: "youtube#searchResult", etag: ""XI7nbFXulYBIpL0ayR_gDh3eu1k/B8CWRo1nM6LhoSdt1ANyW7uGqJ
  k"", id: {…}, snippet: {…}}
    etag: ""XI7nbFXulYBIpL0ayR_gDh3eu1k/B8CWRo1nM6LhoSdt1ANyW7uGqJk""
  ▶ id: {kind: "youtube#video", videoId: "_SU5cL7w1tE"}
    kind: "youtube#searchResult"
  ▼ snippet:
      channelId: "UCmBJWTsAhmcmITadPZjU__g"
      channelTitle: "REAL Watersports"
      description: "The largest surfboard selection in the world! We ship worldwide daily: http:
      liveBroadcastContent: "none"
      publishedAt: "2013-07-23T18:25:43.000Z"
    ▶ thumbnails: {default: {…}, medium: {…}, high: {…}}
      title: "How to choose the right size surfboard - "The Big 3""
    ▶ __proto__: Object
  ▶ __proto__: Object
                                                                        bundle.js:20562
▶ {kind: "youtube#searchResult", etag: ""XI7nbFXulYBIpL0ayR_gDh3eu1k/jGpV9grudiJen2n4bopTvklwJe
  Q"", id: {…}, snippet: {…}}
                                                                        bundle.js:20562
▶ {kind: "youtube#searchResult", etag: ""XI7nbFXulYBIpL0ayR_gDh3eu1k/ycHE_RLG64GZrUW_p2zEJ1JGEW
  M"", id: {…}, snippet: {…}}
                                                                        bundle.js:20562
▶ {kind: "youtube#searchResult", etag: ""XI7nbFXulYBIpL0ayR_gDh3eu1k/H9GqJ-9903QeNV5FXiUKw5jD9M
  o"", id: {…}, snippet: {…}}
                                                                        bundle.js:20562
▶ {kind: "youtube#searchResult", etag: ""XI7nbFXulYBIpL0ayR_gDh3eu1k/nCQ6m9qxAqeNPdvbcZSZFbwoNS
  0"", id: {…}, snippet: {…}}
>
```

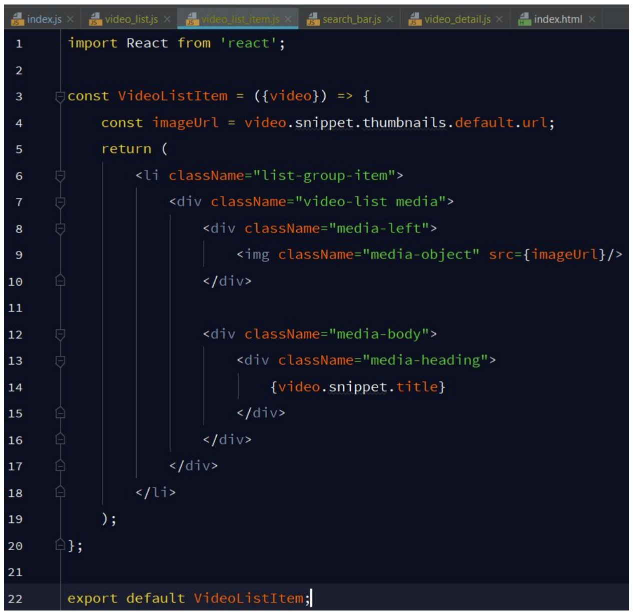
Using bootstrap we made a template of the video list:



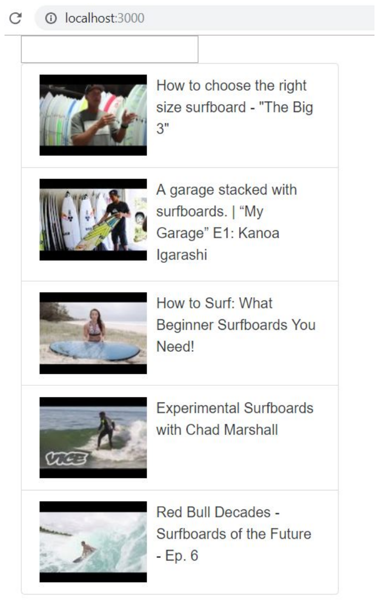
```
import React from 'react';

const VideoListItem = ({video}) => {
    return (
        <li className="list-group-item">
            <div className="video-list media">
                <div className="media-left">
                    <img className="media-object"/>
                </div>

                <div className="media-body">
                    <div className="media-heading"/>
                </div>
            </div>
        </li>
    );
};

export default VideoListItem;
```



Let's pull of the properties such as thumbnail and the title from the video object:



```
import React from 'react';

const VideoListItem = ({video}) => {
    const imageUrl = video.snippet.thumbnails.default.url;
    return (
        <li className="list-group-item">
            <div className="video-list media">
                <div className="media-left">
                    <img className="media-object" src={imageUrl}/>
                </div>

                <div className="media-body">
                    <div className="media-heading">
                        {video.snippet.title}
                    </div>
                </div>
            </div>
        </li>
    );
};

export default VideoListItem;
```



How to choose the right size surfboard - "The Big 3"

A garage stacked with surfboards. | "My Garage" E1: Kanoa Igarashi

How to Surf: What Beginner Surfboards You Need!

Experimental Surfboards with Chad Marshall

Red Bull Decades - Surfboards of the Future - Ep. 6

12. Detail Component and Template Strings

Before you write a new COMPONENT you need to ask yourself a question:
Do I expect this COMPONENT to need to maintain any type of state?

VideoDetail will be functional Component because we don't need any state.

We need to get access to the embed URL.

! Trick about Youtube

Whenever you want to embed a video or even just navigate to a video - the only thing that changes in the URL is the actual **ID** of the video.

```
const url = 'https://www.youtube.com/embed/' + videoId;
```

=

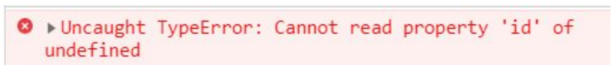
String interpolation

```
const url = `https://www.youtube.com/embed/${videoId}`;
```

13. Handling Null Process

We need a video to pass to `<VideoDetail/>`. We have just got an array of videos without any abilities to select a particular one out of the list.
Let's take the first one from the list : `<VideoDetail video = {this.state.videos[0]} />`

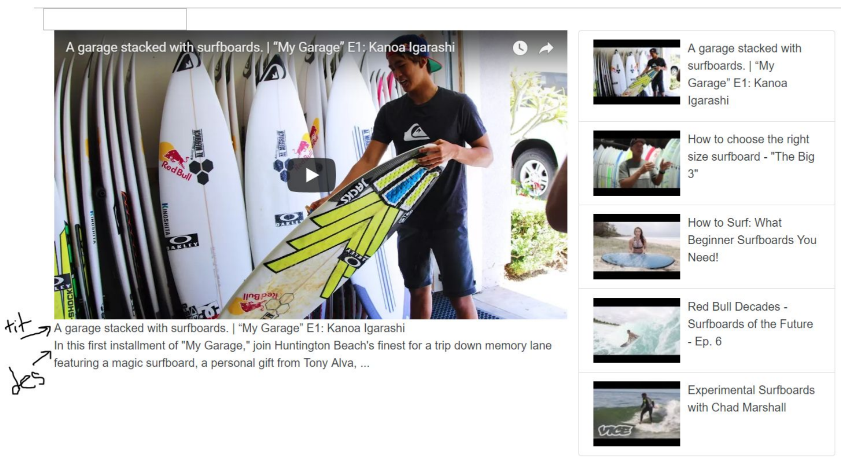
Error:



```
Uncaught TypeError: Cannot read property 'id' of
undefined
```

This happens because React always wants to render instantly, it does not want to wait.
Some parent objects just cannot fetch info fast enough to satisfy the needs of a child object.

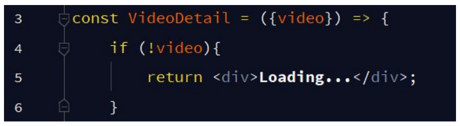
How to handle the case?
Add a check inside of our video detail Component to make sure that the video has been provided in the props before it attempts to render.



A garage stacked with surfboards. | "My Garage" E1: Kanoa Igarashi

A garage stacked with surfboards. | "My Garage" E1: Kanoa Igarashi
In this first installment of "My Garage," join Huntington Beach's finest for a trip down memory lane featuring a magic surfboard, a personal gift from Tony Alva, ...

- A garage stacked with surfboards. | "My Garage" E1: Kanoa Igarashi
- How to choose the right size surfboard - "The Big 3"
- How to Surf: What Beginner Surfboards You Need!
- Red Bull Decades - Surfboards of the Future - Ep. 6
- Experimental Surfboards with Chad Marshall



```
const VideoDetail = ({video}) => {
    if (!video){
        return <div>Loading...</div>;
    }
```

It Works!

14. Video Selection

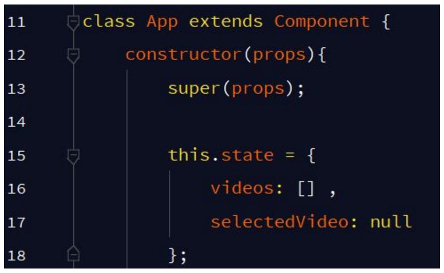
We want to give the user the ability to select a video by clicking and see it pop up on the screen.

Plan:

We will add a concept of a **Selected Video** to the **App Component** state. A **Selected Video** will be always an object and it will always passed into **Video Detail**. So, instead of passing `<VideoDetail video = {this.state.videos[0]} />` we will pass `<VideoDetail video = {this.state.selectedvideo} />`

Step 1

1.Add the concept of a **Selected Video**
Inside index.js , inside our App Component we will add an other initializer to our state.

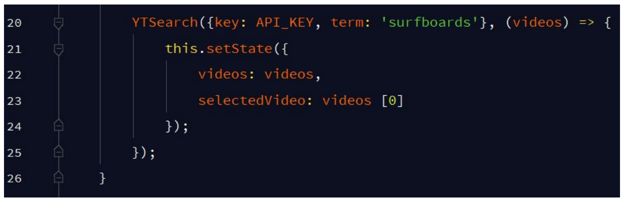


```
class App extends Component {
    constructor(props){
        super(props);

        this.state = {
            videos: [] ,
            selectedVideo: null
        };
```

2.Pass this Selected Video into the Video Detail

Refresh browser - and we stack on Loading...
This happened because we are not changing the state of the **selected video** - it's always set to **null**.

3.Inside of our Search callback, in our Constructor instead of only setting the list of videos, we will also set an inisial video.



```
        YTSearch({key: API_KEY, term: 'surfboards'}, (videos) => {
            this.setState({
                videos: videos,
                selectedVideo: videos [0]
            });
        });
    }
```

Step 2 - Implement **callback**

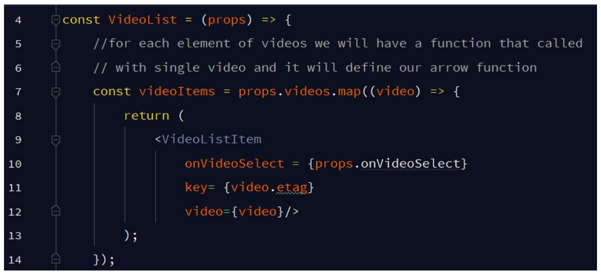
Callback is going to be a **function** that we're going to pass from **App** to **video_list.js** and finally to **video_list_item.js**.

1.Pass another function to **videoList**

```
onVideoSelect = {selectedVideo => this.setState({selectedVideo})}
```

Let's pass this function all the way through.

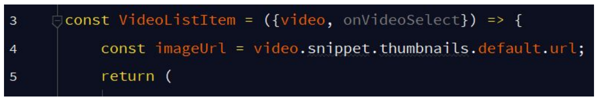
2.Remember we are passing `selectedVideo` as a property of **videoList**.
So **videoList** has a property on props called **props.onVideoSelect**.



```
const VideoList = (props) => {
    //for each element of videos we will have a function that called
    // with single video and it will define our arrow function
    const videoItems = props.videos.map((video) => {
        return (
            <VideoListItem
                onVideoSelect = {props.onVideoSelect}
                key= {video.etag}
                video={video}/>
        );
    });
```

All we doing here is: we taking a prop that is coming from **App** and we're passing it down into **VideoListItem**. So now **VideoListItem** has access to the property called **onVideoSelect**.

3.Pass the callback to video_list_item.js



```
const VideoListItem = ({video, onVideoSelect}) => {
    const imageUrl = video.snippet.thumbnails.default.url;
    return (
```

! Trick from ESX6

If you want to pull multiple properties off the props object, you can just use a curly braces and separate it with a comma:

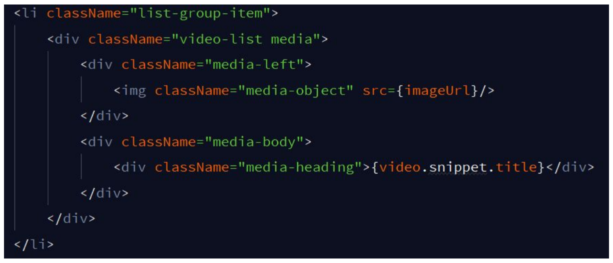
```
const VideoListItem = ({video, onVideoSelect}) => {...
```

It's identical to:

```
const video = props.video
const onVideoSelect = props.onVideoSelect
```

Now we got our **callback** all the way down to **video_list_item.js.**

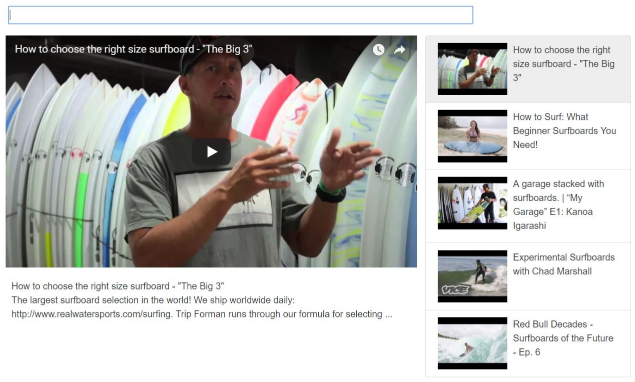
4. We need to make use of the **callback**
So whenever the user clicks on entire **<li** in **video_list_item.js** we are going to treat this as a video selection.



```
<li className="list-group-item">
    <div className="video-list media">
        <div className="media-left">
            <img className="media-object" src={imageUrl}/>
        </div>
        <div className="media-body">
            <div className="media-heading">{video.snippet.title}</div>
        </div>
    </div>
</li>
```

Add `<li onClick = {() => onVideoSelect(video)}...`
It's Working!

15. Styling with CSS



How to choose the right size surfboard - "The Big 3"
The largest surfboard selection in the world! We ship worldwide daily: http://www.realwatersports.com/surfing. Trip Forman runs through our formula for selecting ...

How to choose the right size surfboard - "The Big 3"

How to Surf: What Beginner Surfboards You Need!

A garage stacked with surfboards. | "My Garage" E1: Kanoa Igarashi

Experimental Surfboards with Chad Marshall

Red Bull Decades - Surfboards of the Future - Ep. 6

16. Searching for New Videos

We will follow the similar pattern as we did with the list.
We are going to send **callback** to the **SearchBar.**
The **callback** that we pass to the **SearchBar** is going to take a **string**, a **Search Term,** and make a new **YTSearch** and when search is complete it will set the **state** of the new list of videos.

Process:

1) Define our **callback** by making a new method on our **App**.
`videoSearch(term)` This term is the search term, a string. So whenever the user types into the input.

2) Move the inisial search down into the Video Search
Change the default search term from **‘surfboards’** to **term**.
And inside the Constructor we still want to do this initial Search: `this.videoSearch('surfboards');`

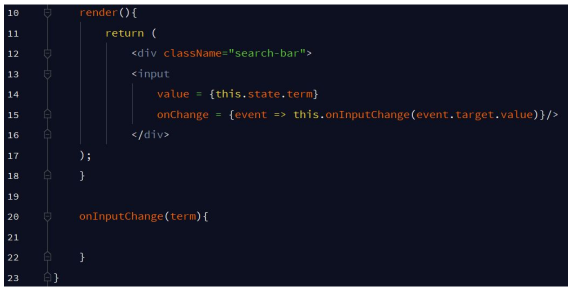
1. So now, when Component is 1st rendered to the DOM ,
2. runs a Constructor, and
3. we’re still doing our video Search ,
4. which is going to kick out the surface.

But now we have our **Searching** mehanizm inside of **videoSearch** method which means we can pass it down into **SearchBar**:
`<SearchBar onSearchTermChange={term => this.videoSearch(term)}/>`

<u>Let’s get to search_bar.js</u>

Problem: Our current **handler** for the input change is tucked away on **state or onChange little function**. If we will add callback here, it will look messy.

Let’s split this event handler out to a separate method.

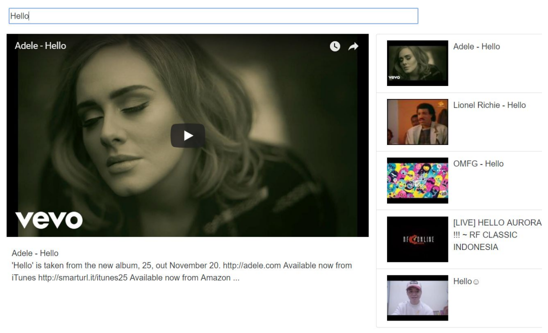


```
    render(){
        return (
            <div className="search-bar">
            <input
                value = {this.state.term}
                onChange = {event => this.onInputChange(event.target.value)}/>
            </div>
        );
    }

    onInputChange(term){

    }
}
```

So now, whenever the input is changed, **onChange**, we call **onInputChange** with the new value with the input itself.

So inside `onInputChange(term){ ...}` we want to do two things:
1. We still want to set the state with the **term**
2. We also want to call the **callback** that we got from the **index.js** from **App.**

```
onInputChange(term){
  this.setState({term});
  this.props.onSearchTermChange(term);
}
```


Hello

Adele - Hello

Adele - Hello
'Hello' is taken from the new album, 25, out November 20. http://adele.com Available now from iTunes http://smarturl.it/itunes25 Available now from Amazon ...

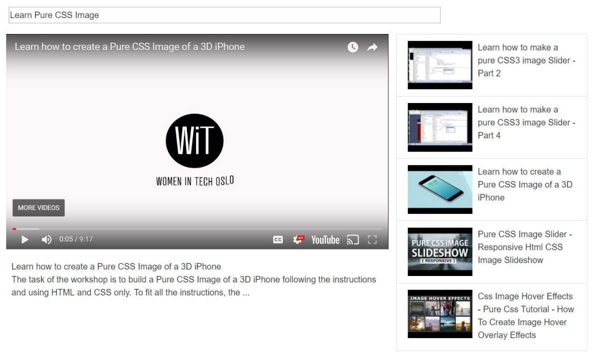
Adele - Hello

Lionel Richie - Hello

OMFG - Hello

[LIVE] HELLO AURORA !!! ~ RF CLASSIC INDONESIA

Hello☺


Learn Pure CSS Image

Learn how to create a Pure CSS Image of a 3D iPhone

Learn how to create a Pure CSS Image of a 3D iPhone
The task of the workshop is to build a Pure CSS Image of a 3D iPhone following the instructions and using HTML and CSS only. To fit all the instructions, the ...

Learn how to make a pure CSS3 image Slider - Part 2

Learn how to make a pure CSS3 image Slider - Part 4

Learn how to create a Pure CSS Image of a 3D iPhone

Pure CSS Image Slider - Responsive Html CSS Image Slideshow

Css Image Hover Effects - Pure Css Tutorial - How To Create Image Hover Overlay Effects

It works!

Little problem: As we type, the App gets a little bit slow. Because everytime we press a button (type new latter), the search does its job.

<u>Summary: Let's go through of exactly what's happening in with the **Search term**</u>

Location : index.js

1. Refactored the **Youtube Search** into its own method. The method just takes a single string - the **Search term**.
2. We took this method **VideoSearch** and we passed it down into **Search Bar** under the property onSearchTermChange.
3. So all **Search Bar** has to do is call **props.onSearchTermChange** with a new **Search term** and that will call our searching function, which will go ahead and fetch a new list of videos.

Location : search_bar.js

1. Reacactore **onChange** event. So whenever the content of input changed it now calls onInputChange with the new input value (event.target.value).
2. The following code has two goals.

onInputChange(term){	
this.setState({term});	1st goal - it sets the State of this Component
this.props.onSearchTermChange (term); }	2nd goal - it fires off the callback function onSearchTermChange

17. Throttling Search Term Input

We will throttle how often we tick off a new search.

Lodash library will help us to throttle the Search. **Lodash** contains a tons of different utility methods, one of which is called **debounce**. **Debounce** could be use to throttle how often a function should be called.

We will pass into `SearchBar` a **debounced** version of the `videoSearch` function :

1. Create the `const videoSearch = _.debounce((term) => {this.videoSearch(term)}, 300);`

Here:
1.1. `(term) => {this.videoSearch(term)}` fat arrow function passed to `debounce`
1.2. `Debounce` takes the function `(term) => {this.videoSearch(term)}` and it returns the new function that can only be called once every 300 milliseconds.
We can call this function as many times as we want , it's not going to crash. But it's just not going to call the original function `(term) => {this.videoSearch(term)}`.

2. Pass that new `videoSearch` function into property `onSearchTermChange`

By passing the new debounced version of `videoSearch` into **SearchBar**. **SearchBar** can update as often as it wants, it can call the callback `onSearchTermChange` all the time. **BUT** it's only going to run once every 300 milliseconds.