Homework: Create Youtube Browser Application

Higher level description:

we will have a search bar where the user will input some word/text and we will call the public youtube api to give us back a list of videos that matches the term/text the user inputs into the search bar.

Once the api gives us back a list of videos, we will display the list of videos to the right hand side of the screen. This is the list with the thumbnail of the video as well as a short preview description of the video. Once the list is populated on the web app, users can click one of the videos.

Once the user clicks on one of those videos, it will show that entire video the left side of the screen along with a description of the video directly under it.

Components each team is responsible for:

Team #1:
SearchBar and VideoDetail

Team #2:
VideoList and VideoItem

Instructions for everyone:
1. go to Documents and run npx create-react-app youtubeHomework and then cd into your youtubeHomework
2. Create your own components directory in your src directory of your new project
3. In your src folder, create an index.js file
4. In the index.js please refer to the pics project index.js because we will want to initalize the App component inside the index.js but pay attention to how we structure the index.js. The pics project's index.js is a good reference to follow.
5. Go to your src/components folder and create a new file called App.js . This will be a class based component.
6. Create the App class component and make sure to define your render method. Inside the render method just have a return statement that has a div that says App. We are only doing this since we need App to be defined as we initialize it inside the index.js file in src.
7. Run npm start in your project folder and you should just see the word App!
8. Dont forget to add the semantics url in the index.html in your public folder.

Search Bar Component Instructions: Team #1

Note: This SearchBar is going to be really similar to the pics project but please still try to figure out how to do it on your own without looking at the past project too much! Try to follow the instructions here instead.

1. Create a SearchBar.js file inside of your components folder. Inside just define the Class based component of Search Bar and have a render method with a div . Inside the div just say Search Bar.
2. Go back to your App component and create an instance of Search Bar in the App class. Make sure to import the Search Bar in the App class and export it in the Search Bar class. Now you should see Search Bar text in your web browser.
3. Create a div with className = "search-bar ui segment". Keep the search-bar tag for now because we are going to use it for custom styling later. Inside that div, create a form div with className = "ui form". Inside the form div, you will have another div with className= "field". Inside that div create a label and a input div. Look at the "Content" section of this Semantics UI link to understand the class names as we are naming it based on semantics ui (https://semantic-ui.com/collections/form.html)
4. The search bar is going to be a little congested so lets add some stlying. Go back to the App component class and inside the div that holds the search bar create a className of "ui container" to help with the margins of the search bar.
5. Now we want the text input to be controlled (and not uncontrolled), so lets create a state to hold the term or text of the input from the user.
6. Once you create the state, inside the input tag add a value prop and set the value equal to the term from the state.
7. Now, I want you guys to figure out which special prop needs to be added along with value in the input tag. Those special props are onClicked, onChange, and onSubmit. Pick one that makes sense for the search bar use case.
8. When you add the special prop in the input tag, you need to pass in a reference to a callback function. Once you set the reference of the callback function, you need to also define the call back function in your Search Bar class.
9. Inside your callback function, make sure to change the state when the user enters a new term into the search bar.
10. Now we need to go back to the special props again (onClicked, onChange, and onSubmit), inside our form tag we need to add one of those special props to deal with the logic of when an user presses enter and submits the term. We also need a callback function passed in and a callback function defined in the class to handle all the events when there is a submission of a term.
11. Inside the submit call back function, we need to call preventDefault() in order to prevent the browser from default resetting itself and not re-rendering the component which causes the term to be reset to an empty string. In short, it prevents the web app from being refreshed and re-rendered.

Instructions for Everyone: Setting up the Youtube API

1. Go to console.developers.google.com
2. Look at the very top of the Google APIs logo on the top left of the website and click the drop down menu that says "select a project". Then, click NEW PROJECT on the right hand corner. Give a project name and click create button.
3. Give it some time. There is going to be a spinner runner that means the project is still being created. Once its done spinning, your project is finished and ready to use. Once it is finished, clic on "select a project" again and click your project.
4. Click the Enable APIs and Services then in the search bar and look for Youtube. Click the Youtube Data API v3 and click that and click Enable.
5. Now we need to make our access keys so click Create Credentials button in the right hand side of the screen.
   - In the form in the drop down menu select Youtube Data API v3
   - Secondly, pick web browser (javascript)
   - Lastly, click public data.
   - Then click "what Credentials Do I need" then it will take you to your API key. Copy it and save it somewhere.

6. Create an api folder in your src folder and then create a file named youtube.js. For now, make a variable with all caps KEY and set that equal to your access key from the youtube api.

7. Install npm install axios@0.18.1 in your project. Run this specific command because for some reason the youtube api is throwing a 400 error with some missing parameter error. To avoid this error, install this version of axios inside your project.
8. Youtube documentation for making api call to get list of videos:
https://developers.google.com/youtube/v3/docs/search/list
9. Take a look at the GET request: GET https://www.googleapis.com/youtube/v3/search
10. For the GET API parameters, all we need are these parameters:
   - for the part parameter set it to snippet so it should look like this: part: snippet
   - set maxResults to 5 to show 5 videos at a time so it should look like this: maxResults: 5
   - set q to surfing. it should look like this : q: surfing.

11. Parameters clarification: part parameter is what information we want from the youtube api and snippet value means we want a snippet summary of the video such as description of the video, url of the video, etc
12. the q parameter is just a query and when we set it to surfing which just means we give the api a search term to give us something back.

12. In the youtube.js we want to configure a client of the youtube api in this file.
   - import axios in this file
   - do an export default axios.create in this file.
   - define a baseURL: https://www.googleapis.com/youtube/v3

- then define all your parameters here like part and maxResults. I know q is missing in params but that is in purpose because we only want to pass that parameter when we make a query (aka when a user makes an input to the search bar). Instead of q , in the params add a key param and set it equal to your KEY variable. Please note that snippet is a string type and 5 is a integer type.
- One more key to add in your params object in your axios create is a type key and set that to the string 'video' so we only get videos and not playlists of videos.

13. Add a callback method in the App class that will be triggered everytime someone makes an input in the search bar. Call the callback function whatever makes sense.
14. import your axios client config by doing import youtube from '../apis/youtube'
15. Make a prop in the SearchBar component in the App component and set that prop equal to the reference of the call back function.

16. Inside the callback function in App class component, do a get method call from your youtube axios object but here is where things get a little tricky. You need to pass in term for the parameter for the callback function but inside the get function we need two parameters. The first parameter is the endpoint. Since the youtube axios client has the root url all we need to specfiy now is '/search' but we need a second parameter called which is a javascript object so you need to have a {} and inside that object we need to define the q which is the query parameter in the youtube api. I will let you guys figure out what to set q as.
17. In your callback function in the app class, make sure to define the function as async since the api call is async. For the async style syntax refer back to the pics project.
18. For the api response, take a look at the data property that will contain items which will contain the array of videos that we want to be displayed in the screen. It will look like response.data.items.
19. Create a state in the app component and set the list of videos to this state.

Quick note to Team #2:
You technically dont need to wait for team #1 to finish the search bar component. You can create a regular function that can be invoked in the App class component. This can just be a function call that will contain the logic of calling the youtube api. You can pass in hardcoded values for testing your VideoDetail and VideoList component . This will be the most challenging part of the homework for you two.

Team #1 Search Bar Part 2:

1. Figure out where to make the callback function call with props in the Search Bar class component as we need to pass the term to the callback from the App class component.

Team #2 Instructions for Video List and Video Item:

1. Please think about how you can call the api without the search bar component. All you need is the response of list of videos in order for your work to be done successfully.

2. Create a new functional component called VideoList and create a new file inside your components folder.
3. Then go back to your App component and import Video list and create an instance of Video List in your App component and figure out what component yall need to pass down from the App component to the VideoList component.
4. Now, we want to create a VideoItem component because we have many individual videos from the video list. We want to create a component for each video from the video list. This video item will be a functional component.
5. Which file should we import the VideoItem to? where is it going to be used?
6. Next, inside the Video List, you want to call a map function and pass in a parameter video so we can perform mapping to each video in the video list from the api response similar to the pics project. Please remember the map function returns back a new array of the mapped videos. Therefore, set the map function to a variable called renderedList because that will contain all the videos that have the mapped logic to each video.
7. For the map function, just return an instance of VideoItem each and every time. If we do this , the new array from the map function will contain an instance of VideoItem component and we can also add props to each VideoItem component which will be applied in the VideoItem component layer. For props, pass in a video from the vide list as a prop down to the Video Item component.
8. Now your VideoItem should have a video object from the video list from the api response. Each video object has a item field and inside item there is a list of objects. Click on one of those objects you will see a snippet property and inside the snippet property there is a title property inside that we want to use.
9. To get an image create an img tag inside of the div and if you look into snippet again, there is a thumnail property. Inside the thumbnail property, use the medium property and then use the url property inside the medium property (looks like snippet.thumbnails.medium.url)

Team #2 Styling Instructions for Video List and Video Item:

1. Go to semantics ui docs (https://semantic-ui.com/elements/list.html) and go to the elements section and go to List documentation. You should see a github looking example and look at the html source code. Look at the "ui relaxed divided list" div className and inside of that div there are child divs with its own className.
2. In the VideoList you can apply "ui relaxed divided list" to every video . Figure out where you can add this div for styling. If we want all videos in the video list to have that className for styling for semantics ui where is a good place to add it in your VideoList component?
3. Now go to your VideoItem component. Every video item needs to have a className of item since that is the structure of the semantics ui styling component we are using. Also figure out where to add this "item" className in which div?
4. Scroll down to the image section in semantics ui because we want to place the image to the left side of the text. There should be an avatar with some text. Expand that html code and look at the content and header classNames which we will use for our Video Item component. Since we are styling the image create a className for the img tag and in this tag we just need to

define the parent class of semantics ui for this styling component which is ui image. Do not do ui image avatar because that just puts your image into a circle.
5. Then we want to wrap the snippet.title around other divs that will include the content and header className from semantics ui. Figure out if those divs need to be nested together.
6. Create a VideoItem.css to apply more styling. Just copy and paste my css file contents to your css file.
7. Once I give you the css file and you copy it, identify the root div in where you can declare the css into your Video Item. Also do not forget to import the css file into your VideoItem component.

Team #1 and Team #2 work together - Communicating from Child to Parent Components for a Selected Video: This process is needed to Set Up Video Detail component . Work on this logic together then Team #1 can work on VideoDetail together.

1. Go to your App class component and create a new field in the state called selectedVideo (default the value to be null if no state is set yet) and we want to pass that selected video to your VideoDetail component and that video will be rendered on the screen on the far left side.
2. When the user clicks on the right hand side of videos, it needs to update the state of selectedVideo in the App component.
3. To handle this logic, we need to create another call back function called onVideoSelect and we are going to pass this callback function to our VideoList component. However, the Video List component will also pass this callback function to its child which is Video Item.

Diagram:
App onVideoSelect prop > Video List > VideoItem

Basically the onVideoSelect callback will be passed down from App to Video List . Because we have a map function of each video in the videoList , each VideoItem component will also get passed down the onVideoSelect callback function as well. Every VideoItem component will be passed down a prop of onVideoSelect callback function. So when a user clicks the video item that was selected it will trigger that onVideoSelect callback function in the VideoItem that was selected by the user.

4. Figure out where to pass the callback in the VideoList component in the App class component.

5. Now inside the VideoList component figure out how to pass the callback function onVideoSelect to each VideoItem component.

6. Once each VideoItem component is passed down the callback function as a prop we need to use the special props of (onClick, onSubmit, and onChanged) in one of the divs in the VideoItem component class. I will let you decide which speical prop to use and where to apply it

in the VideoItem component. You need to invoke the callback function in one of these special props.

7. Once all the passing of the callback function is finished, lets talk about how we can show the selected video to our web browser. In the app component, we just need to update the state in the onVideoSelect callback function to whatever the callback returns which is video. Remember what a callback is. Once the callback finishes in the VideoItem layer it will give back the user selected video back to the App component.

Team #1 - Creating Video Detail:

1. Make a new VideoDetail file in your components folder and make it a functional component.
2. Make an instance of Video Detail in your App class component and figure out what prop needs to be passed in the VideoDetail instance.
3. To prevent any null errors (this is possible as your selectedVideo in your state in App could always be null), add an if statement in your Video Detail to capture the null case.
4. Like in the seasons project, if the video is null in VideoDetail just return a div that says loading....
5. If it is not null, return a div that has the snippet.title so it shows in the video detail! Read VideoItem and VideoList section for more context of the response given back from the youtube api.
6. Styling the Video Detail:
   - For the snippet.title div , give it a class name of "ui segment" so we can put it in a nice box from semantics ui.
   - Add a h4 html tag between the snippet.title to increase the text in the screen. Give the h4 tag a className of "ui header" (another styling component from semantics ui)
   - below the h4 tag, create a paragraph tag ( ) to put the description of the video here.
   - in the snippet property from the video response, there is a description field we can reference.
7. Now lets handle displaying the video inside of VideoDetail.
   - We are going to use an iframe tag. It is an html tag that makes a request on its own to an outside website (youtube.com in our case) and the outside website gives back all the necessary html that will get displayed by the iframe. In this case, a video should be displayed.
   - Above the ui segment div className, put another div with className = "ui embed" (this is from semantics as well that puts styling to the video that will be embedded to the web browser). Then create an iframe tag which looks like this <iframe src = {} /> . We need to pass in a source to the iframe in the src. Go to any youtube video and click the share button and click the embedd button and you will see an iframe tag. This is an example of how iframes are used.
   - For the iframe src, we can get the src url from the video response
      - This part is tricky. We need to create a const variable videoSrc = `https://www.youtube.com/embed/${video.id.videoId}` . We need to use the back slash again as we want to pass in a javascript object to the root url of youtube. We need the video id because that is needed for the iframe to render that video from youtube.com.

8. We are going to get some warnings in the web app side. So we need to fix some of these warnings.

- Iframes need titles so add one. just give it a property title and set it to a string. Call it whatever makes sense.
- Second warning, each VideoItem needs an alt property in the img tag. This is seen in the pics project. Alt is just needed and shown when an image does not successfully load up so that is why we got this warning. Go to VideoItem and add that alt tag and pass in the snippet.title to the alt tag.
- Third warning is that each video item needs a key property needed in the browser side so in the map function add a key = {video.id.videoId}. This change is needed in the VideoList component.
- More styling is needed to put the video detail in the left side and the video list is in the right side of the browser. We will use the grid system in semantics ui. Go to collection and grid. (https://semantic-ui.com/collections/grid.html)
- In the App component, figure out where to add the div className = "ui grid" and another div className = "ui row". Hint: ui grid should be nested and a child of ui grid. Also, we want the videos to be in the same row.
- Now for VideoDetail wrap that in another div with className = "eleven wide column" which just means it will take 11 columns on the screen.
- For where VideoList is created in the app component class wrap that in a div as well and assign it

  className = "five wide columns"

Entire Group works on the loading issue:

1. Fix the loading message. To fix this , we can just a default search term for the user.

2. Go to your App component, in your onTermSubmit callback function , we can both update the list of videos and pick one default video from that list and set that to the selectedVideo state field so we have both the video list and video detail in sync. Just set it as the first video in the video list for the selectedVideo state. To get the first video in the list, remember the concepts of arrays.

3. To handle the loading text, just create a componentDidMount in the app component and make a default search when the app component is first rendered. In the componentDidMount, call the onTermSubmit and pass a dummy search text.