# YouTube Clone Frontend Framework

## Introduction

In the first phase of the YouTube Clone project, we built out the backend for our application, creating an API that exposes access to Comments and their Replies that are stored in a database tied to a user using the unique videoId for each video those comments are being left on.

Now, we are moving into the second phase of the project. We will be using React to build a front-end client application that will consume both your backend API for user registration, logging in, comments/replies as well as the YouTube Data API for video information and related videos.   
  
This front-end will increase the complexity of asynchronous programming, as multiple components will be dependent on data coming from external resources.

## Technologies

React.js, React Hooks, HTML/CSS, Bootstrap, Axios, YouTube Data API

## Learning Objective

The objective of this project is to leverage modern React features and multiple external data sources to create a wrapper around the YouTube Data API – creating a minimal alternative client that exposes only the critical features needed to enjoy videos! You will also sharpen design skills by crafting a wireframe for the front-end and making your application look as close to the design as possible!

💡 Do not progress through this document until your backend API has been checked off by an instructor!

## Resources

**PowerPoints**

* React Hooks
* Design/UX
* Wireframing + Prototyping
* React JWT
* React Router

**Documents**

* User Stories document – contains information and links relating to making requests to the YouTube Data API

**Relevant Projects**

Music Library (both front and back end)

YouTube Clone Backend

**Other Resources**

* Third Party API Integration and React Hooks video
* Postman (FOR TESTING BOTH APIs ONLY)
* Conditionally Rendering JSX - <https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications>

## Tasks

1. **Utilize the links in the user stories to obtain an API key for the YouTube Data API and test the two requests in Postman.** Pay close attention to the documentation and be sure to note the possible params you could utilize in your requests (including snippet)
2. **Watch the provided React Django Starter Code Walkthrough videos for the Frontend.**
3. **Design the React application using a wireframing/prototyping tool like Adobe XD or Figma.** Refer to the Wireframing and Prototyping PowerPoint lecture for a low fidelity version of what we expect at a minimum for the project.
4. **Build and test the React Frontend** using React best practices (components organized into their own folders with external CSS stylesheets, lifting state up as high as possible in component hierarchy, passing data from parents to children as props, etc)

## Setup Steps:

*For Step 3 from Tasks*

1. Utilize the starter code to begin building your React application. You will need these additional components at a minimum:

* SearchPage.jsx
* VideoPage.jsx
* VideoPlayer.jsx
* RelatedVideos.jsx
* SearchBar.jsx
* CommentList.jsx
* CommentForm.jsx
* Comment.jsx

For Bonus:

* ReplyList.jsx
* ReplyForm.jsx
* Reply.jsx

1. Work on getting your main pages and routes set up using the pattern established in the starter code (create a page component in the pages folder, add a route to the App.js file, protect it with the PrivateRoute component if only a logged in user can access)
2. Start with the SearchPage component and set up your Axios requests to fetch videos from the YouTube DATA API based on a provided search string.
3. When your search results are verified to be coming in, work on getting the VideoPage up and running, using an Embedded Player to play the video selected from the SearchPage. For now, just hard code the videoId string (see documentation on Embedded Player from user stories – be sure to just use the iframe).
4. Once the video iframe element is displaying and playing correctly, work on getting a single video result from your search request into your embedded video component.
5. Once you get a video populating the VideoPlayer.jsx component, begin working on the RelatedVideos component. ***HINT: You need to make sure API CALL #2 is working with the populated video before rendering the thumbnails for each.***
6. After related videos are properly displaying, begin working on comments and their replies. This is trickier than it seems at first so be patient and problem solve! Remember that we want to lift state as high as possible in our component hierarchy. If you are doing this, then the same videoId that you are using to display your video and related videos can also be used to fetch comments for that video!

## End Result

Please see recorded video on the day this project is assigned. Keep in mind that the result you see there may look a bit different from what yours looks like – be creative with your application design and have fun! **Also, while the demo mentions that the user will be automatically logged in when registered, you will actually be re-directed to the Login page after registration in the starter code.**