Project Report: VueTube – an Alternative YouTube Frontend

Table of Contents

1. Introduction

Project Overview

The Vue YouTube Alternative Frontend is a web application that provides users with an alternative interface for accessing and interacting with YouTube content. It utilizes the YouTube Data API to fetch and display videos, playlists, and user data. This project aims to create a user-friendly and feature-rich Vue.js application that seamlessly integrates with YouTube's extensive library of content.

Purpose and Scope

The primary purpose of this project is to demonstrate the capabilities of Vue.js in building a modern web application. The scope includes creating a frontend that allows users to search for videos, view video details, and interact with the content. While this project focuses on the frontend, it assumes that the backend authentication and API key management are handled separately.

2. Technologies Used

Vue.js

Vue.js is a progressive JavaScript framework for building user interfaces. It provides a flexible and efficient way to develop interactive web applications.

YouTube Data API

The YouTube Data API v3 allows developers to access YouTube's vast repository of videos, playlists, channels, and user data programmatically.

PouchDB

It enables applications to store data locally while offline, then synchronize it with CouchDB and compatible servers when the application is back online, keeping the user's data in sync no matter where they next login.

3. Project Architecture

Frontend Structure

The Vue YouTube Alternative Frontend follows a component-based architecture. Key components include:

- **SearchBar**: Allows users to search for videos.
- **VideoList**: Displays search results or a list of recommended videos.
- **VideoPlayer**: Plays selected videos.
- **User Interaction**: Provides features for liking, commenting, and sharing videos.

YouTube Data API Integration

The integration with the YouTube Data API is essential for fetching data. API endpoints include search, video details, user playlists, and video comments. API key management and authentication are assumed to be handled separately.

4. Features

Search and Browse Videos

Users can search for videos using keywords. Search results are displayed as a list of video thumbnails and titles.

Video Playback

Users can click on a video thumbnail to play the video in the VideoPlayer component. Playback controls are provided.

User Interaction

Users can like, unlike, and view his liked videos. These interactions are linked to the user's web browser using PouchDB.

5. Challenges

Managing API Requests

Efficiently managing API requests, handling rate limits, and dealing with pagination are challenges when working with the YouTube Data API.

UI/UX Design

Creating an intuitive and visually appealing user interface that enhances the user's experience is a continuous challenge.

6. Future Improvements

Future improvements for this project may include:

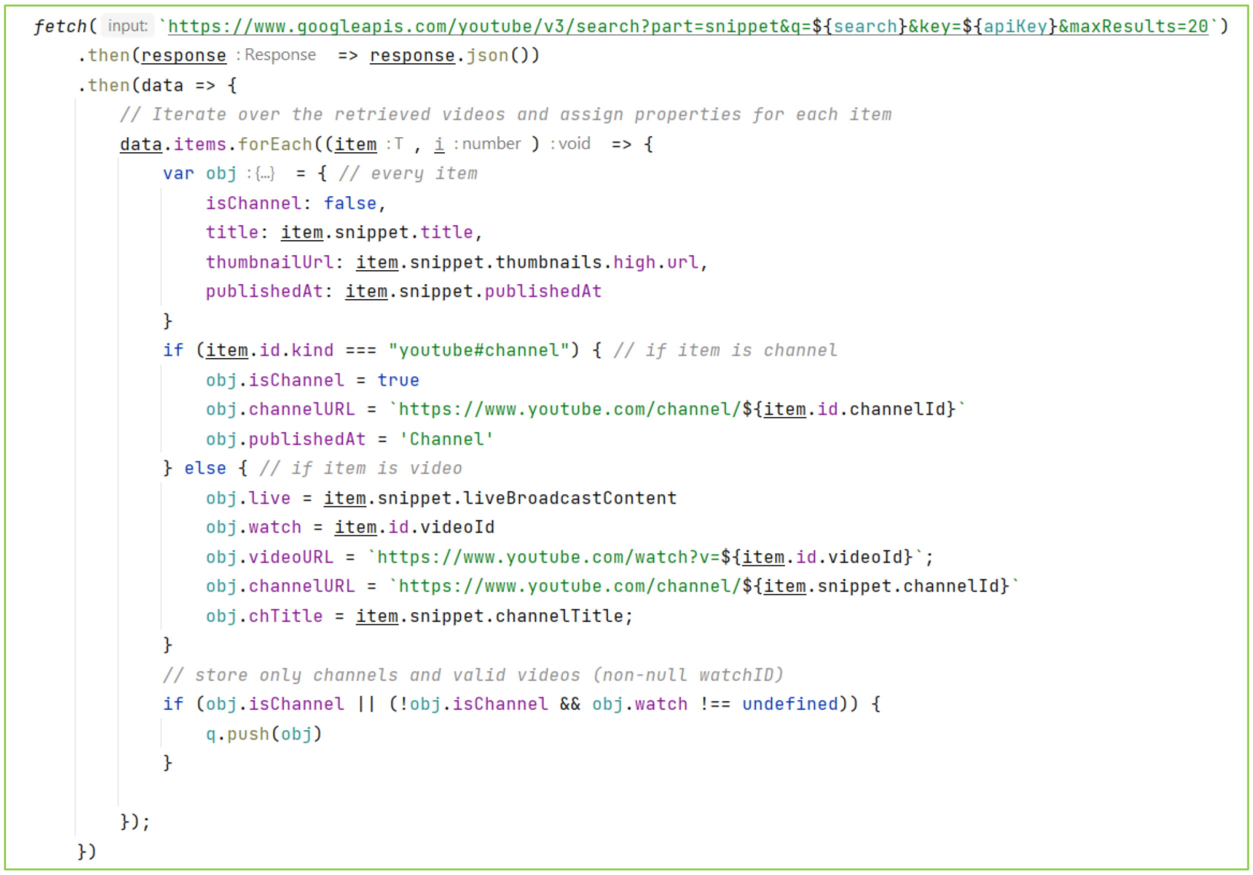
- Implementing more advanced search filters.
- Supporting user playlists and subscriptions.
- Enhancing the video recommendation algorithm.
- Implementing a responsive design for mobile devices.
- Adding support for video uploads (if applicable).

7. Conclusion

The Vue YouTube Alternative Frontend demonstrates the power of Vue.js in building a user-friendly web application that interacts with the YouTube Data API. It provides a foundation for creating a full-fledged YouTube alternative, emphasizing frontend functionality. Further development and integration with a backend system would be required for a complete solution.

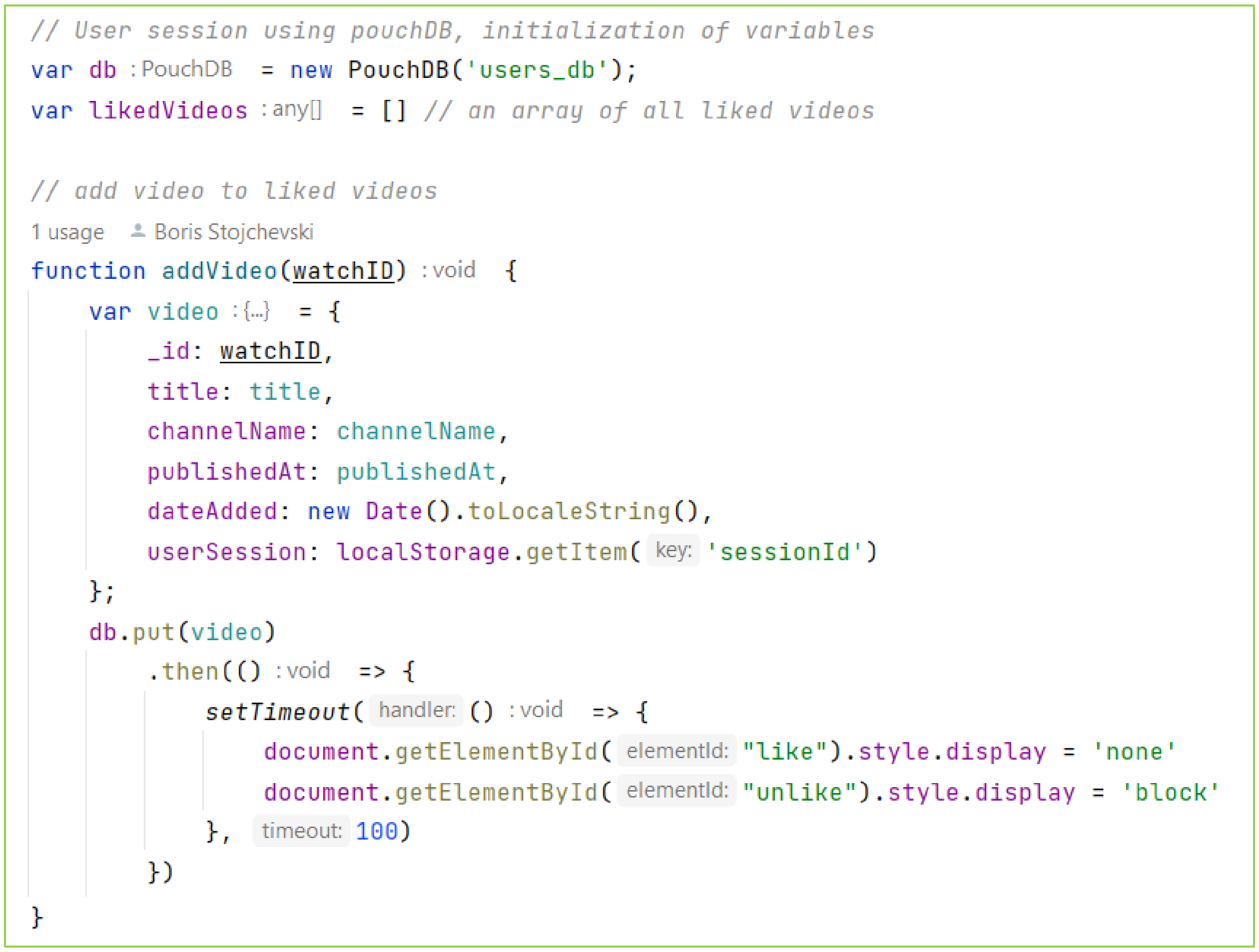
8. Appendix

Code Snippets



```
fetch( input: `https://www.googleapis.com/youtube/v3/search?part=snippet&q=${search}&key=${apiKey}&maxResults=20`)
    .then(response : Response  => response.json())
    .then(data => {
        // Iterate over the retrieved videos and assign properties for each item
        data.items.forEach((item :T , i :number ) :void  => {
            var obj :{…}  = { // every item
                isChannel: false,
                title: item.snippet.title,
                thumbnailUrl: item.snippet.thumbnails.high.url,
                publishedAt: item.snippet.publishedAt
            }
            if (item.id.kind === "youtube#channel") { // if item is channel
                obj.isChannel = true
                obj.channelURL = `https://www.youtube.com/channel/${item.id.channelId}`
                obj.publishedAt = 'Channel'
            } else { // if item is video
                obj.live = item.snippet.liveBroadcastContent
                obj.watch = item.id.videoId
                obj.videoURL = `https://www.youtube.com/watch?v=${item.id.videoId}`;
                obj.channelURL = `https://www.youtube.com/channel/${item.snippet.channelId}`
                obj.chTitle = item.snippet.channelTitle;
            }
            // store only channels and valid videos (non-null watchID)
            if (obj.isChannel || (!obj.isChannel && obj.watch !== undefined)) {
                q.push(obj)
            }

        });
    })
```

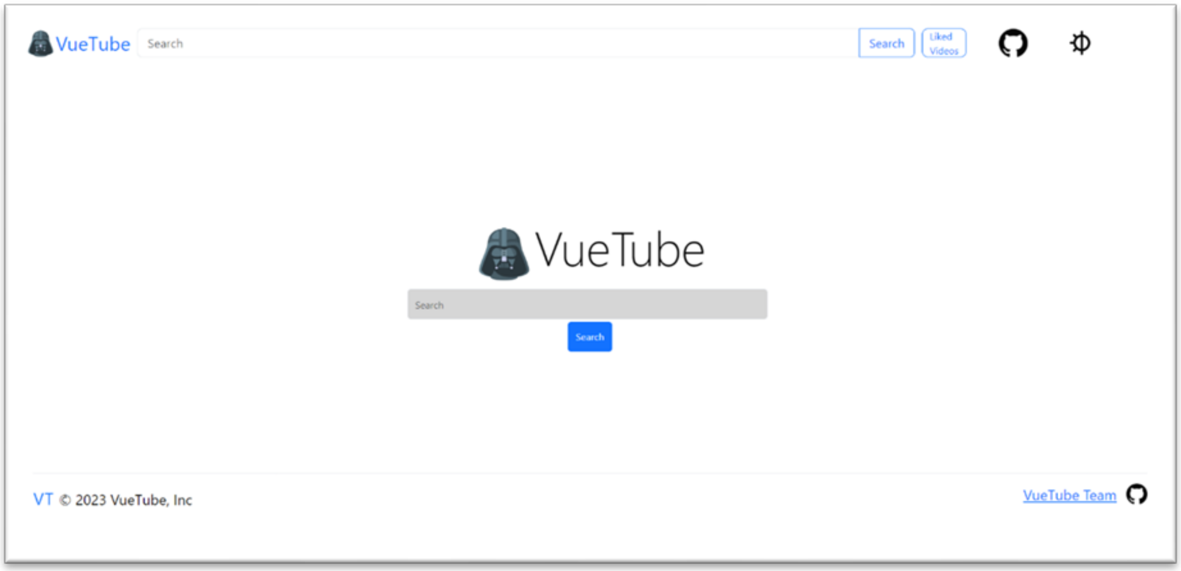
Code Snippet 1: Retrieving data from YouTube Data API using preregistered API key, with the queryParameter maxResults set to 20 items (which can be set to 50). The API call returns an array of objects. For each item it creates a new object and it assigns each key-value pairs that are of need. At the end, if the object is valid it is being pushed to a result array which will be to display the retrieved data.



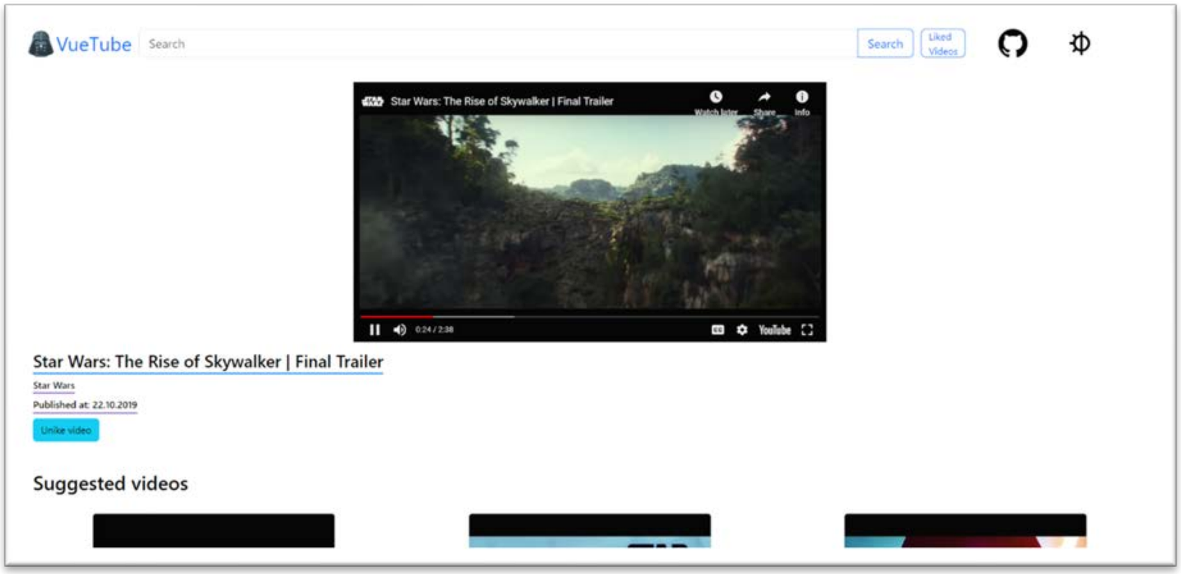
```
// User session using pouchDB, initialization of variables
var db : PouchDB  = new PouchDB('users_db');
var likedVideos : any[]  = [] // an array of all liked videos

// add video to liked videos
1 usage   Boris Stojchevski
function addVideo(watchID) : void  {
    var video : {...}  = {
        _id: watchID,
        title: title,
        channelName: channelName,
        publishedAt: publishedAt,
        dateAdded: new Date().toLocaleString(),
        userSession: localStorage.getItem( key: 'sessionId')
    };
    db.put(video)
        .then(() : void  => {
            setTimeout( handler: () : void  => {
                document.getElementById( elementId: "like").style.display = 'none'
                document.getElementById( elementId: "unlike").style.display = 'block'
            },  timeout: 100)
        })
}
```

Code Snippet 2: An explanation of how client-side database is working. After being embedded to the html, it is innitialized and stored in the db variable. For each video there is a like button, associated with the addVideo(watchID) function. The function stores the specified video as liked to the associated user.

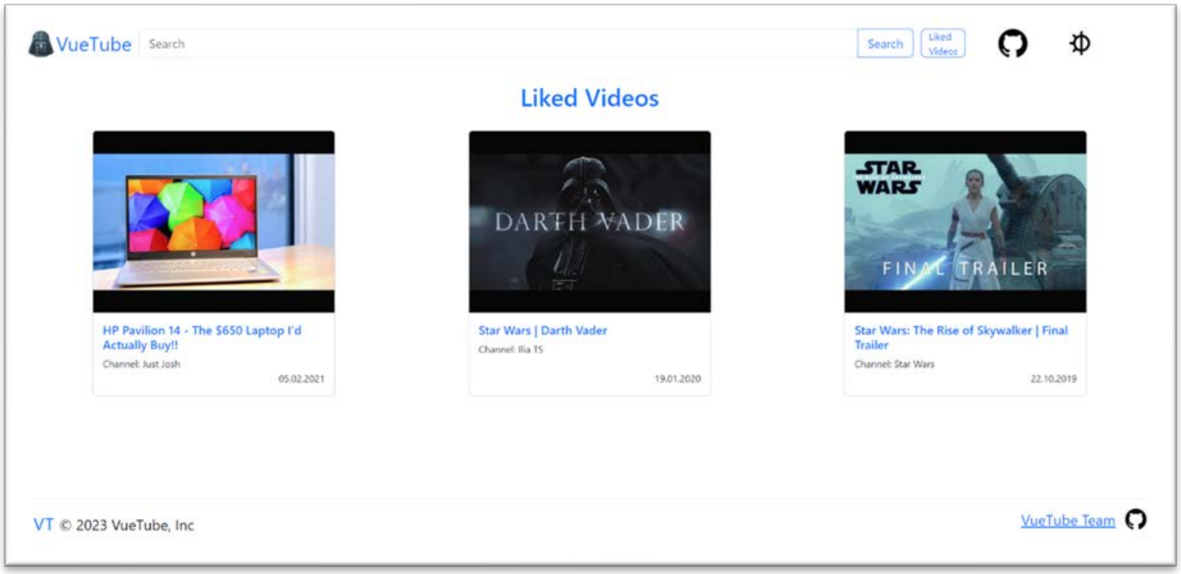
Screenshots



Screenshot 1



Screenshot 2



Screenshot 3

References

- Vue
- Bootstrap
- jQuery
- YouTube Data API
- PouchDB
- HTML
- CSS
- JavaScript