**YouTube Channel Data**

**1. Introduction**

The purpose of this document is to provide a functional overview of the application feature that fetches YouTube channel data. This feature aims to retrieve various information about a YouTube channel using the YouTube Data API.

**2. Requirements**

* The application should allow the user to input a YouTube channel ID.
* The application should retrieve the following YouTube channel data:
  + Subscriber count
  + Joining date
  + Audience base information
  + Audience activity analytics
  + Audience demographics (country, language, age, gender)
  + Sentiment analysis of the latest comments

**3. Technical Specifications**

**Programming Language:** Python

**Libraries and APIs:**

Google API Python Client Library

YouTube Data API v3

TextBlob (for sentiment analysis)

NLTK (for sentiment analysis)

**Database:** Redis

**API Framework:** Django/Flask

**4. Workflow**

* User launches the application and provides a YouTube channel ID as input with an authenticated API  
  **API Request**

POST https://<host>/getChannelDetails HTTP/1.1

Content-Type: application/json

Authorization: Bearer <token>

{

“Link” : “<https://www.youtube.com/@tseries>”

}

**Response valid case:**

200 OK

{

“status” : “Accepted”

}

**Response invalid case:**

400 Bad Request

{

“status” : “Invalid input”

}

**Response processing:**

200 OK

{

“status” : “Processing”

}

**Response processed:**

200 OK

{

“status” : “Processed”,

“timestamp”: “000000000”,

“channel\_name” : “T-Series”,

“channel\_user\_name” : “tseries”,

“subscriber\_count” : “999999”,

“views”: “9999”,

“join\_date”: “2020-12-01”,

“Parsed\_email\_ids” : [“a@a.com”, “b@b.com”]

“latest\_videos”: [

{

“title” : “...”,

“Comment\_word\_cloud” : [“aa”,”bb”],

“sentiment\_score” : “-0.01”

},{

...

}

]

}

* The application parses the URL and grads the channel ID. Regex match should be used to validate the URL pattern. In case of invalid input the API should error out immediately.
* If it is a valid channel the data is entered into redis for further processing. The value is marked as “Processing”. Timestamp is also added. In case the record already exists and is in processing state, do not do anything. In case it is processed and the timestamp + 30 is less than current timestamp respond with the details.
* An event listener job keeps running in the background which checks for entries in Redis and starts processing in case the timestamp + 20 mins is less than current timestamp else mark as error
* The application authenticates using the Google API credentials file.
* creds = Credentials.from\_authorized\_user\_file(API\_CREDENTIALS\_FILE)
* youtube = build('youtube', 'v3', credentials=creds)
* The application connects to the YouTube Data API and retrieves the YouTube channel data using the provided channel ID.
* The application extracts and processes the fetched data, including subscriber count, joining date, audience base, audience activity analytics, audience demographics, and sentiment analysis of the latest comments.
* # Retrieve channel details
* channel\_request = youtube.channels().list(
* part='snippet,statistics',
* id=CHANNEL\_ID
* )
* channel\_response = channel\_request.execute()
* # Extract relevant information
* channel = channel\_response['items'][0]
* subscriber\_count = channel['statistics']['subscriberCount']
* joining\_date = channel['snippet']['publishedAt']
* total\_views = channel['statistics']['viewCount']
* # Retrieve audience analytics
* analytics\_request = youtube.reports().query(
* ids='channel==' + CHANNEL\_ID,
* startDate=(datetime.now() - timedelta(days=365)).strftime('%Y-%m-%d'),
* endDate=datetime.now().strftime('%Y-%m-%d'),
* metrics='viewerPercentage',
* dimensions='day',
* sort='day'
* )
* analytics\_response = analytics\_request.execute()
* # Extract audience activity analytics
* audience\_activity = {}
* for row in analytics\_response.get('rows', []):
* date = row[0]
* percentage = float(row[1])
* audience\_activity[date] = percentage
* # Retrieve audience demographics
* demographics\_request = youtube.channels().list(
* part='brandingSettings',
* id=CHANNEL\_ID
* )
* demographics\_response = demographics\_request.execute()
* demographics = demographics\_response['items'][0]['brandingSettings']['channel']['targetedDemographics']
* Fetch the last 10 (max) videos and analyse top 10 comments. Get the latest 100-150 comments and get the number of likes and replies. Sort the data based on the number of likes and number of replies. (number of replies would have more priority while sorting). Take the
* comments\_request = youtube.commentThreads().list(
* part='snippet',
* allThreadsRelatedToChannelId=CHANNEL\_ID,
* order='time',
* maxResults=10
* )
* comments\_response = comments\_request.execute()
* # Perform sentiment analysis on comments
* sentiment\_analyzer = SentimentIntensityAnalyzer()
* sentiments = []
* for item in comments\_response['items']:
* comment\_text = item['snippet']['topLevelComment']['snippet']['textDisplay']
* sentiment\_scores = sentiment\_analyzer.polarity\_scores(comment\_text)
* sentiment = sentiment\_scores['compound']
* sentiments.append(sentiment)
* # Calculate overall sentiment score
* sentiment\_score = sum(sentiments) / len(sentiments) if sentiments else 0
* The application presents the fetched data to the user in a structured format
* The fetched YouTube channel data is updated into Redis and also updated into a relational database for future reference.
* All the channels usernames that are queried using the API are put into a MySQL database for continuous data collection based on the following criteria.: If the subscriber count is > 1000 the query should be made every day. If it is between 5000 to 10000 it should be done every alternate day, and if it is less than 500 it should be queried on a weekly basis.

**5. Error Handling**

* Invalid or missing YouTube channel ID provided by the user.
* Authentication failure due to incorrect or missing API credentials.
* Failed connection to the YouTube Data API.
* Invalid or incomplete data retrieved from the YouTube Data API.
* The application should handle these errors gracefully, providing meaningful error messages and guidance to the user.