LABORATORY REPORT

# Application Development Lab
# (CS33002)

**B.Tech Program in ECSc**

Submitted By

**Name:-** SAHIL KUMAR

**Roll No:** 2230189


**Kalinga Institute of Industrial Technology**
**(Deemed to be University)**
**Bhubaneswar, India**

Spring 2024-2025

# Table of Content

| | |
|---|---|
| **Experiment Number** | 8 |
| **Experiment Title** | Sentiment Prediction API Using FastAPI and Youtube comments |
| **Date of Experiment** | 17\|03\|2025 |
| **Date of Submission** | 24\|03\|2025 |

## 1. Objective:-

- The objective of this lab experiment is to create a sentiment prediction API using FastAPI, which analyzes Youtube comments for positive, negative, or neutral sentiment. This lab integrates natural language processing (NLP) techniques with a lightweight and high-performing API framework.

## 2. Procedure:- (Steps Followed)

- Install Required Libraries Install FastAPI Google API Client Google Generative AI Groq scikit-learn pandas and uvicorn using the command pip install fastapi google-api-python-client google-generativeai groq pandas scikit-learn uvicorn python-dotenv
- Create a Google Cloud Project and Enable YouTube Data API Go to Google Cloud Console create a new project enable the YouTube Data API v3 and generate an API key
- Set Up API Keys Store your YouTube API key Gemini API key or Groq API key in a env.local file
- Authenticate and Fetch YouTube Comments Use the googleapiclient.discovery module to interact with the YouTube API extract video ID from a given YouTube URL and fetch a specified number of comments
- Perform Sentiment Analysis on Comments Use Gemini AI or Groq AI to analyze comment sentiment and if API limits are reached use a basic keyword-based sentiment analysis
- Define Sentiment Categories Classify comments as Positive Neutral or Negative based on AI-generated results
- Create a Function to Process Comments Fetch comments analyze sentiment and return structured data including author text sentiment likes and timestamp
- Initialize a FastAPI Application Set up a FastAPI app to handle requests

- Define API Endpoints Create a GET endpoint to return a welcome message and a POST api comments endpoint to accept a YouTube URL and the number of comments to fetch
- Integrate YouTube Comment Fetching and Sentiment Analysis Ensure the api comments endpoint fetches comments processes them and returns sentiment results
- Run the API Using Uvicorn Use the command uvicorn app app host 0.0.0.0 port 8000 reload to start the server
- Test the API Using Postman cURL or a Browser The GET endpoint should return a welcome message and the POST api comments endpoint should return analyzed comments with sentiment
- Verify the Output Ensure fetched comments include proper sentiment classification

## 3. Code:-

Index.html file:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
            <meta   name="viewport"   content="width=device-width,
initial-scale=1.0" />
    <title>YouTube Comment Sentiment Analyzer</title>
    <link
      rel="stylesheet"

href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500
;600;700&display=swap"
    />
    <link rel="stylesheet" href="./styles/styles.css" />
  </head>
  <body>
    <div class="container">
      <header>
        <h1>YouTube Comment Sentiment Analyzer</h1>
           <p>Analyze the sentiment of comments from any YouTube
video</p>
      </header>
```

```
      <div class="form-container">
        <form id="analysis-form">
          <div class="form-group">
            <label for="video-url">YouTube Video URL:</label>
            <input
              type="text"
              id="video-url"
              name="video_url"
              placeholder="https://www.youtube.com/watch?v=..."
              required
            />
          </div>

          <div class="form-group">
                <label for="comment-count">Number of Comments to
Analyze:</label>
            <input
              type="number"
              id="comment-count"
              name="comment_count"
              min="1"
              max="500"
              value="50"
              required
            />
          </div>

          <div class="form-group">
            <label for="model">Sentiment Analysis Model:</label>
            <select id="model" name="model" required>
              <option value="gemini">Google Gemini</option>
              <option value="groq">Groq</option>
            </select>
          </div>

                <button type="submit" id="analyze-btn">Analyze
Comments</button>
        </form>
      </div>

      <div class="loading-container" id="loading">
        <div class="spinner"></div>
```

```
        <p>Analyzing comments... This may take a moment.</p>
      </div>

      <div class="results-container" id="results">
        <div class="summary-section">
          <h2>Sentiment Summary</h2>
          <div class="sentiment-chart">
            <div class="chart-container" id="chart-container">
              <!-- Chart will be rendered here -->
            </div>
            <div class="sentiment-stats" id="sentiment-stats">
              <!-- Stats will be filled here -->
            </div>
          </div>
          <div class="video-preview" id="video-preview">
            <!-- Video embed will appear here -->
          </div>
        </div>

        <div class="comments-section">
          <h2>Detailed Comment Analysis</h2>
          <div class="filter-controls">
            <label>Filter by sentiment:</label>
                              <button   class="filter-btn   active"
data-sentiment="all">All</button>
            <button class="filter-btn" data-sentiment="positive">
              Positive
            </button>
                                      <button    class="filter-btn"
data-sentiment="neutral">Neutral</button>
            <button class="filter-btn" data-sentiment="negative">
              Negative
            </button>
          </div>
          <div class="comments-list" id="comments-list">
                <!-- Comments will be filled here dynamically by
JavaScript -->
          </div>
        </div>
      </div>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
```

```
    <script src="./styles/script.js"></script>
  </body>
</html>
```

Styles.css file:

```
:root {
    --primary-color: #4361ee;
    --secondary-color: #3a0ca3;
    --positive-color: #4cc9f0;
    --neutral-color: #f72585;
    --negative-color: #7209b7;
    --bg-color: #f8f9fa;
    --card-bg: #ffffff;
    --text-color: #2b2d42;
    --border-color: #e9ecef;
     --shadow: 0 10px 15px -3px rgba(0,0,0,0.1), 0 4px 6px -2px
rgba(0,0,0,0.05);
    --transition: all 0.3s ease;
}

* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body {
     font-family: 'Inter', 'Segoe UI', system-ui, -apple-system,
sans-serif;
    line-height: 1.6;
    color: var(--text-color);
    background-color: var(--bg-color);
    padding: 20px;
}

.container {
    max-width: 1200px;
```

```
    margin: 0 auto;
}

header {
    text-align: center;
    margin-bottom: 40px;
    padding: 30px;
       background:  linear-gradient(135deg,  var(--primary-color),
var(--secondary-color));
    color: white;
    border-radius: 12px;
    box-shadow: var(--shadow);
}

header h1 {
    margin-bottom: 10px;
    font-size: 2.4rem;
    font-weight: 700;
    letter-spacing: -0.5px;
}

header p {
    font-size: 1.1rem;
    opacity: 0.9;
}

.form-container {
    background-color: var(--card-bg);
    padding: 30px;
    border-radius: 12px;
    box-shadow: var(--shadow);
    margin-bottom: 40px;
    transition: var(--transition);
}

.form-container:hover {
    transform: translateY(-3px);
     box-shadow: 0 15px 20px -3px rgba(0,0,0,0.1), 0 6px 8px -2px
rgba(0,0,0,0.05);
}

.form-group {
    margin-bottom: 24px;
```

```
}

label {
    display: block;
    margin-bottom: 8px;
    font-weight: 600;
    font-size: 0.95rem;
    color: var(--text-color);
}

input, select {
    width: 100%;
    padding: 14px;
    border: 1px solid var(--border-color);
    border-radius: 8px;
    font-size: 1rem;
    transition: var(--transition);
    background-color: #fafafa;
}

input:focus, select:focus {
    outline: none;
    border-color: var(--primary-color);
    box-shadow: 0 0 0 3px rgba(67, 97, 238, 0.15);
}

button {
    background-color: var(--primary-color);
    color: white;
    padding: 14px 28px;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    font-size: 1rem;
    font-weight: 600;
    transition: var(--transition);
    display: inline-flex;
    align-items: center;
    justify-content: center;
    box-shadow: 0 4px 6px rgba(67, 97, 238, 0.2);
}

button:hover {
```

```
    background-color: #3651d1;
    transform: translateY(-2px);
    box-shadow: 0 6px 8px rgba(67, 97, 238, 0.25);
}

button:active {
    transform: translateY(0);
}

.loading-container {
    display: none;
    flex-direction: column;
    align-items: center;
    margin: 60px 0;
}

.spinner {
    border: 4px solid rgba(67, 97, 238, 0.1);
    border-radius: 50%;
    border-top: 4px solid var(--primary-color);
    width: 60px;
    height: 60px;
    animation: spin 1s linear infinite;
    margin-bottom: 20px;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

.loading-container p {
    font-size: 1.1rem;
    color: #666;
}

.results-container {
    display: none;
}

.summary-section, .comments-section {
    background-color: var(--card-bg);
    padding: 35px;
```

```
    border-radius: 12px;
    box-shadow: var(--shadow);
    margin-bottom: 40px;
    transition: var(--transition);
}

.summary-section:hover, .comments-section:hover {
    transform: translateY(-3px);
     box-shadow: 0 15px 20px -3px rgba(0,0,0,0.1), 0 6px 8px -2px
rgba(0,0,0,0.05);
}

h2 {
    margin-bottom: 25px;
    color: var(--secondary-color);
    border-bottom: 2px solid var(--border-color);
    padding-bottom: 12px;
    font-weight: 700;
    font-size: 1.8rem;
    letter-spacing: -0.3px;
}

.sentiment-chart {
    display: flex;
    flex-wrap: wrap;
    gap: 40px;
    margin-bottom: 40px;
}

.chart-container {
    flex: 1;
    min-width: 300px;
    height: 300px;
    position: relative;
}

.sentiment-stats {
    flex: 1;
    min-width: 300px;
    display: flex;
    flex-direction: column;
    justify-content: center;
    background-color: #f8f9fa;
```

```
    padding: 25px;
    border-radius: 10px;
}

.stat-item {
    display: flex;
    align-items: center;
    margin-bottom: 18px;
    padding: 10px 15px;
    border-radius: 8px;
    background-color: white;
    box-shadow: 0 2px 5px rgba(0,0,0,0.05);
    transition: var(--transition);
}

.stat-item:hover {
    transform: translateX(5px);
    box-shadow: 0 3px 7px rgba(0,0,0,0.08);
}

.stat-color {
    width: 24px;
    height: 24px;
    border-radius: 50%;
    margin-right: 15px;
}

.positive-bg {
    background-color: var(--positive-color);
    box-shadow: 0 0 0 3px rgba(76, 201, 240, 0.3);
}

.neutral-bg {
    background-color: var(--neutral-color);
    box-shadow: 0 0 0 3px rgba(247, 37, 133, 0.3);
}

.negative-bg {
    background-color: var(--negative-color);
    box-shadow: 0 0 0 3px rgba(114, 9, 183, 0.3);
}

.video-preview {
```

```
    width: 100%;
    max-width: 700px;
    margin: 30px auto 10px;
}

.video-preview iframe {
    width: 100%;
    aspect-ratio: 16/9;
    border: none;
    border-radius: 12px;
    box-shadow: var(--shadow);
    transition: var(--transition);
}

.video-preview iframe:hover {
    transform: scale(1.01);
}

.filter-controls {
    margin-bottom: 25px;
    display: flex;
    align-items: center;
    flex-wrap: wrap;
    gap: 12px;
    padding: 15px;
    background-color: #f8f9fa;
    border-radius: 10px;
}

.filter-controls label {
    margin-bottom: 0;
    margin-right: 8px;
}

.filter-btn {
    background-color: white;
    color: var(--text-color);
    padding: 10px 18px;
    font-weight: 500;
    border-radius: 30px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.05);
    transition: var(--transition);
}
```

```css
.filter-btn:hover {
    background-color: #f1f3f5;
    transform: translateY(-2px);
}

.filter-btn.active {
    background-color: var(--primary-color);
    color: white;
    transform: translateY(-2px);
    box-shadow: 0 4px 6px rgba(67, 97, 238, 0.2);
}

.comments-list {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(350px, 1fr));
    gap: 25px;
    grid-auto-rows: 1fr;
}

.comment-card {
    background-color: white;
    border-radius: 12px;
    padding: 25px;
    box-shadow: 0 4px 10px rgba(0,0,0,0.08);
    transition: var(--transition);
    border-top: 5px solid;
    display: flex;
    flex-direction: column;
    height: 100%;
}

.comment-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 10px 15px rgba(0,0,0,0.1);
}

.comment-card.positive {
    border-top-color: var(--positive-color);
}

.comment-card.neutral {
    border-top-color: var(--neutral-color);
```

```
}

.comment-card.negative {
    border-top-color: var(--negative-color);
}

.comment-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 15px;
    padding-bottom: 12px;
    border-bottom: 1px solid var(--border-color);
    flex-wrap: wrap;
    gap: 8px;
}

.comment-author {
    font-weight: 600;
    font-size: 1.05rem;
    display: flex;
    align-items: center;
    gap: 8px;
    overflow: hidden;
    text-overflow: ellipsis;
    max-width: 70%;
    white-space: nowrap;
}

.comment-author::before {
    content: '';
    display: inline-block;
    width: 10px;
    height: 10px;
    border-radius: 50%;
    background-color: #ccc;
}

.comment-card.positive .comment-author::before {
    background-color: var(--positive-color);
}

.comment-card.neutral .comment-author::before {
```

```
    background-color: var(--neutral-color);
}

.comment-card.negative .comment-author::before {
    background-color: var(--negative-color);
}

.comment-content {
    flex-grow: 1;
    line-height: 1.5;
    color: #4a4a4a;
    margin-bottom: 15px;
    font-size: 0.95rem;
    overflow-y: auto;
    max-height: 150px;
}

.comment-sentiment {
    padding: 5px 12px;
    border-radius: 20px;
    font-size: 0.8rem;
    font-weight: 600;
    text-transform: capitalize;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
    white-space: nowrap;
}

.sentiment-positive {
    background-color: var(--positive-color);
    color: white;
}

.sentiment-neutral {
    background-color: var(--neutral-color);
    color: white;
}

.sentiment-negative {
    background-color: var(--negative-color);
    color: white;
}

.comment-metadata {
```

```
    display: flex;
    justify-content: space-between;
    align-items: center;
    font-size: 0.8rem;
    color: #6c757d;
    margin-top: auto;
    padding-top: 12px;
    border-top: 1px solid var(--border-color);
}

.comment-likes, .comment-date {
    display: flex;
    align-items: center;
    gap: 5px;
}

.comment-likes::before {
    content: '👍';
    font-size: 0.9rem;
}

.comment-date::before {
    content: '🕒';
    font-size: 0.9rem;
}

@media (max-width: 768px) {
    .comments-list {
        grid-template-columns: 1fr;
    }

    .sentiment-chart {
        flex-direction: column;
    }

    header {
        padding: 20px;
    }

    header h1 {
        font-size: 1.8rem;
    }
```

```
    .form-container, .summary-section, .comments-section {
        padding: 20px;
    }
}
```

script.js file

```
document.addEventListener("DOMContentLoaded", function () {
  const form = document.getElementById("analysis-form");
  const loadingElement = document.getElementById("loading");
  const resultsElement = document.getElementById("results");
                    const          commentsListElement          =
document.getElementById("comments-list");
                    const          videoPreviewElement          =
document.getElementById("video-preview");
                  const          sentimentStatsElement          =
document.getElementById("sentiment-stats");
                      const           chartContainer           =
document.getElementById("chart-container");
  const analyzeBtn = document.getElementById("analyze-btn");

  let sentimentChart;

  form.addEventListener("submit", async function (event) {
    event.preventDefault();

    loadingElement.style.display = "flex";
    resultsElement.style.display = "none";

    analyzeBtn.disabled = true;
    analyzeBtn.textContent = "Processing...";

    try {
      const formData = new FormData(form);

      const response = await fetch("/analyze", {
        method: "POST",
        body: formData,
      });
```

```
    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.detail || "An error occurred");
    }

    const data = await response.json();
    displayResults(data);

    document
      .querySelector(".loading-container")
      .scrollIntoView({ behavior: "smooth", block: "start" });
  } catch (error) {
    alert("Error: " + error.message);
  } finally {
    loadingElement.style.display = "none";

    analyzeBtn.disabled = false;
    analyzeBtn.textContent = "Analyze Comments";
  }
});

document.querySelectorAll(".filter-btn").forEach((button) => {
  button.addEventListener("click", function () {
    document
      .querySelectorAll(".filter-btn")
      .forEach((btn) => btn.classList.remove("active"));
    this.classList.add("active");

    const sentiment = this.dataset.sentiment;
    filterComments(sentiment);
  });
});

function displayResults(data) {
  resultsElement.style.display = "block";

  embedVideo(data.video_id);

  displaySentimentStats(data.stats);

  renderSentimentChart(data.stats);

  displayComments(data.comments);
```

```
    document
      .querySelector(".loading-container")
      .scrollIntoView({ behavior: "smooth", block: "start" });
  }

  function embedVideo(videoId) {
    videoPreviewElement.innerHTML = `
      <iframe
        src="https:
        allowfullscreen
        title="YouTube video player"
      ></iframe>
    `;
  }

  function displaySentimentStats(stats) {
          const total = stats.positive + stats.neutral +
stats.negative;
        const positivePercent = ((stats.positive / total) *
100).toFixed(1);
         const neutralPercent = ((stats.neutral / total) *
100).toFixed(1);
        const negativePercent = ((stats.negative / total) *
100).toFixed(1);

    sentimentStatsElement.innerHTML = `
      <div class="stat-item">
        <div class="stat-color positive-bg"></div>
        <div>
          <strong>Positive:</strong> ${stats.positive} comments
          <span>(${positivePercent}%)</span>
        </div>
      </div>
      <div class="stat-item">
        <div class="stat-color neutral-bg"></div>
        <div>
          <strong>Neutral:</strong> ${stats.neutral} comments
          <span>(${neutralPercent}%)</span>
        </div>
      </div>
      <div class="stat-item">
        <div class="stat-color negative-bg"></div>
```

```
        <div>
          <strong>Negative:</strong> ${stats.negative} comments
          <span>(${negativePercent}%)</span>
        </div>
      </div>
      <div class="stat-item" style="background-color: #f0f4f8;">
        <div style="margin-left: 34px;">
          <strong>Total analyzed:</strong> ${total} comments
        </div>
      </div>
    `;
  }

  function renderSentimentChart(stats) {
    if (sentimentChart) {
      sentimentChart.destroy();
    }

                      chartContainer.innerHTML      =      '<canvas
id="sentiment-chart"></canvas>';
                                        const          ctx          =
document.getElementById("sentiment-chart").getContext("2d");

    sentimentChart = new Chart(ctx, {
      type: "doughnut",
      data: {
        labels: ["Positive", "Neutral", "Negative"],
        datasets: [
          {
                           data:  [stats.positive,  stats.neutral,
stats.negative],
            backgroundColor: ["#4cc9f0", "#f72585", "#7209b7"],
            borderWidth: 0,
          },
        ],
      },
      options: {
        responsive: true,
        maintainAspectRatio: false,
        cutout: "70%",
        plugins: {
          legend: {
            position: "bottom",
```

```
            labels: {
              padding: 20,
              usePointStyle: true,
              font: {
                size: 12,
              },
            },
          },
          tooltip: {
            backgroundColor: "rgba(0, 0, 0, 0.8)",
            padding: 12,
            cornerRadius: 8,
            titleFont: {
              size: 14,
              weight: "bold",
            },
            bodyFont: {
              size: 13,
            },
            displayColors: true,
            callbacks: {
              label: function (context) {
                  const total = context.dataset.data.reduce((a, b)
=> a + b, 0);
                    const percentage = Math.round((context.raw /
total) * 100);
                                return  `${context.raw}  comments
(${percentage}%)`;
              },
            },
          },
        },
        animation: {
          animateScale: true,
          animateRotate: true,
          duration: 800,
        },
      },
    });
  }

  function displayComments(comments) {
    commentsListElement.innerHTML = "";
```

```
    comments.forEach((comment) => {
      const commentCard = createCommentCard(comment);
      commentsListElement.appendChild(commentCard);
    });
  }

  function createCommentCard(comment) {
    const date = new Date(comment.published_at);
    const formattedDate = date.toLocaleDateString(undefined, {
      year: "numeric",
      month: "short",
      day: "numeric",
    });

    const card = document.createElement("div");
                          card.className      =      `comment-card
${comment.sentiment.toLowerCase()}`;
    card.dataset.sentiment = comment.sentiment.toLowerCase();

    card.innerHTML = `
      <div class="comment-header">
                                                                <div
class="comment-author">${escapeHTML(comment.author)}</div>
                                  <span    class="comment-sentiment
sentiment-${comment.sentiment.toLowerCase()}">${
      comment.sentiment
    }</span>
      </div>
                                                                <div
class="comment-content">${escapeHTML(comment.text)}</div>
      <div class="comment-metadata">
        <div class="comment-date">${formattedDate}</div>
        <div class="comment-likes">${comment.likes}</div>
      </div>
    `;

    return card;
  }

  function escapeHTML(str) {
    return str
      .replace(/&/g, "&amp;")
```

```
      .replace(/</g, "&lt;")
      .replace(/>/g, "&gt;")
      .replace(/"/g, "&quot;")
      .replace(/'/g, "&#039;");
  }

  function filterComments(sentiment) {
    const comments = document.querySelectorAll(".comment-card");
    const filterAnimation = [
      { opacity: 0.5, transform: "scale(0.98)" },
      { opacity: 1, transform: "scale(1)" },
    ];

    const filterTiming = {
      duration: 300,
      easing: "ease-out",
    };

    comments.forEach((comment) => {
        if (sentiment === "all" || comment.dataset.sentiment ===
sentiment) {
        comment.style.display = "flex";
        comment.animate(filterAnimation, filterTiming);
      } else {
        comment.style.display = "none";
      }
    });
  }
});
```

app.py:-

```
from fastapi import FastAPI, Request, Form, HTTPException
from fastapi.responses import HTMLResponse
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles
import os
import re
import time
import asyncio
```

```
from dotenv import load_dotenv
from googleapiclient.discovery import build
from typing import List, Dict, Any
import google.generativeai as genai
from groq import AsyncGroq
import logging
import json


logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)


load_dotenv(".env.local")

app = FastAPI()


app.mount("/styles",              StaticFiles(directory="styles"),
name="styles")
templates = Jinja2Templates(directory=".")


YOUTUBE_API_KEY = os.getenv("YOUTUBE_API_KEY")
youtube = build('youtube', 'v3', developerKey=YOUTUBE_API_KEY)


GEMINI_API_KEY = os.getenv("GEMINI_API_KEY")
GROQ_API_KEY = os.getenv("GROQ_API_KEY")


if GEMINI_API_KEY:
    genai.configure(api_key=GEMINI_API_KEY)
    gemini_model = genai.GenerativeModel('gemini-2.0-flash')


if GROQ_API_KEY:
    groq_client = AsyncGroq(api_key=GROQ_API_KEY)


MIN_REQUEST_INTERVAL = 1
```

```
last_request_time = {
    "gemini": 0,
    "groq": 0
}


def extract_video_id(url):
    """Extract YouTube video ID from URL"""

    patterns = [
        r'(?:v=|\/)([0-9A-Za-z_-]{11}).*',
        r'(?:embed\/)([0-9A-Za-z_-]{11})',
        r'(?:youtu\.be\/)([0-9A-Za-z_-]{11})'
    ]

    for pattern in patterns:
        match = re.search(pattern, url)
        if match:
            return match.group(1)

    raise ValueError("Invalid YouTube URL")


async def fetch_comments(video_id: str, max_comments: int) ->
List[dict]:
    """Fetch comments from a YouTube video"""
    comments = []
    next_page_token = None

    try:
        while len(comments) < max_comments:

            request = youtube.commentThreads().list(
                part="snippet",
                videoId=video_id,
                maxResults=min(100, max_comments - len(comments)
                               ),
                pageToken=next_page_token,
                textFormat="plainText"
            )

            response = request.execute()
```

```
            for item in response['items']:
                                                    comment    =
item['snippet']['topLevelComment']['snippet']
                comments.append({
                    'author': comment['authorDisplayName'],
                    'text': comment['textDisplay'],
                    'likes': comment['likeCount'],
                    'published_at': comment['publishedAt']
                })

            next_page_token = response.get('nextPageToken')
                    if not next_page_token or len(comments) >=
max_comments:
                break

        return comments[:max_comments]

    except Exception as e:
        logger.error(f"Error fetching comments: {str(e)}")
        raise HTTPException(
               status_code=500, detail=f"Error fetching comments:
{str(e)}")


async def rate_limit_request(api_name):
    """Implement rate limiting to avoid 429 errors"""
    current_time = time.time()
    elapsed = current_time - last_request_time[api_name]

    if elapsed < MIN_REQUEST_INTERVAL:
        wait_time = MIN_REQUEST_INTERVAL - elapsed
        await asyncio.sleep(wait_time)

    last_request_time[api_name] = time.time()


async def analyze_batch_with_gemini(comments_batch: List[Dict])
-> List[str]:
    """Analyze a batch of comments with Gemini API"""
    await rate_limit_request("gemini")

    try:
```

```
        comments_text = "\n".join(
              [f"Comment {i+1}: {comment['text']}" for i, comment
in enumerate(comments_batch)])

        prompt = f"""
              Analyze the sentiment of each of the following
{len(comments_batch)} YouTube comments.
          For each comment, classify it as 'positive', 'neutral',
or 'negative'.

        {comments_text}

          Respond in JSON format with comment numbers as keys and
sentiment as values:
        {{
          "1": "positive",
          "2": "negative",
          ...
        }}

        Only return the JSON object, nothing else.
        """

        response = gemini_model.generate_content(prompt)
        response_text = response.text.strip()

        if response_text.startswith("```json"):
            response_text = response_text.split("```json")[1]
        if response_text.endswith("```"):
            response_text = response_text.split("```")[0]

        response_text = response_text.strip()
        if not response_text.startswith("{"):

            start_idx = response_text.find("{")
            end_idx = response_text.rfind("}") + 1
            if start_idx >= 0 and end_idx > start_idx:
                response_text = response_text[start_idx:end_idx]

        results = json.loads(response_text)

        sentiments = []
        for i in range(len(comments_batch)):
```

```
            sentiment = results.get(str(i+1), "neutral").lower()
                    if sentiment not in ["positive", "neutral",
"negative"]:
                sentiment = "neutral"
            sentiments.append(sentiment)

        return sentiments

    except Exception as e:
        logger.error(f"Error with Gemini API batch: {e}")

        return ["neutral"] * len(comments_batch)


async def analyze_batch_with_groq(comments_batch: List[Dict]) ->
List[str]:
    """Analyze a batch of comments with Groq API"""
    await rate_limit_request("groq")

    try:

        comments_text = "\n".join(
              [f"Comment {i+1}: {comment['text']}" for i, comment
in enumerate(comments_batch)])

        response = await groq_client.chat.completions.create(
            model="llama3-8b-8192",
            messages=[
                {
                    "role": "system",
                         "content": "You are a sentiment analysis
assistant. You will receive multiple YouTube comments. Analyze
each comment and classify it as 'positive', 'neutral', or
'negative'. Return results as a JSON object."
                },
                {
                    "role": "user",
                    "content": f"""
                           Analyze the sentiment of each of the
following {len(comments_batch)} YouTube comments.

                    {comments_text}
```

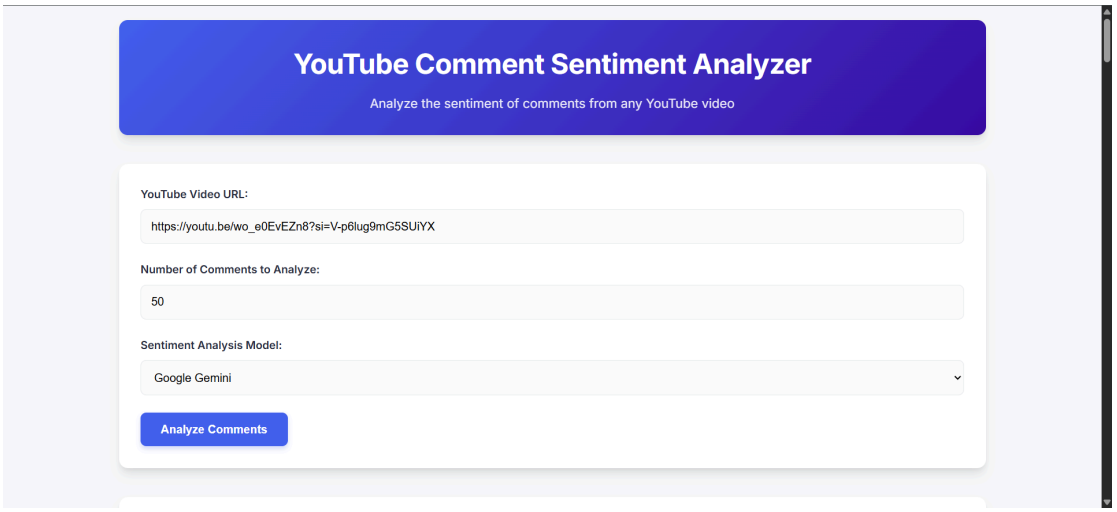
```
                    Return a JSON object with comment numbers as
keys and sentiment values ('positive', 'neutral', or 'negative'):
                    {{
                      "1": "positive",
                      "2": "negative",
                      ...
                    }}
                    Only return the JSON object, nothing else.
                    """
                }
            ],
            max_tokens=1000
        )

                                                response_text      =
response.choices[0].message.content.strip()

        if response_text.startswith("```json"):
            response_text = response_text.split("```json")[1]
        if response_text.endswith("```"):
            response_text = response_text.split("```")[0]

        response_text = response_text.strip()
        if not response_text.startswith("{"):

            start_idx = response_text.find("{")
            end_idx = response_text.rfind("}") + 1
            if start_idx >= 0 and end_idx > start_idx:
                response_text = response_text[start_idx:end_idx]

        results = json.loads(response_text)

        sentiments = []
        for i in range(len(comments_batch)):
            sentiment = results.get(str(i+1), "neutral").lower()
                    if  sentiment  not  in  ["positive",  "neutral",
"negative"]:
                sentiment = "neutral"
            sentiments.append(sentiment)

        return sentiments

    except Exception as e:
```

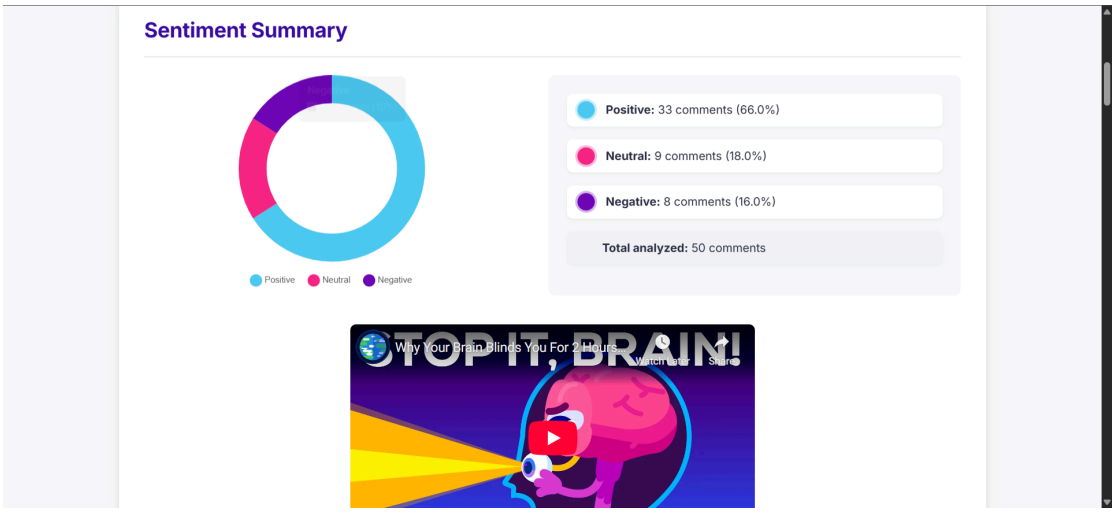
```
        logger.error(f"Error with Groq API batch: {e}")

        return ["neutral"] * len(comments_batch)


async def batch_analyze_sentiment(comments: List[dict], model:
str, batch_size: int = 10) -> List[dict]:
    """Analyze sentiment in real batches for efficiency and to
avoid rate limits"""
    results = []

    for i in range(0, len(comments), batch_size):
        batch = comments[i:i + batch_size]

        if model.lower() == "gemini" and GEMINI_API_KEY:
                                    batch_sentiments = await
analyze_batch_with_gemini(batch)
        else:
                                    batch_sentiments = await
analyze_batch_with_groq(batch)

        for j, sentiment in enumerate(batch_sentiments):
            if i + j < len(comments):
                results.append({
                    **batch[j],
                    "sentiment": sentiment
                })

    return results


@app.get("/", response_class=HTMLResponse)
async def read_root(request: Request):
    return templates.TemplateResponse("index.html", {"request":
request})


@app.post("/analyze")
async def analyze(
    request: Request,
    video_url: str = Form(...),
    comment_count: int = Form(...),
    model: str = Form(...)
```

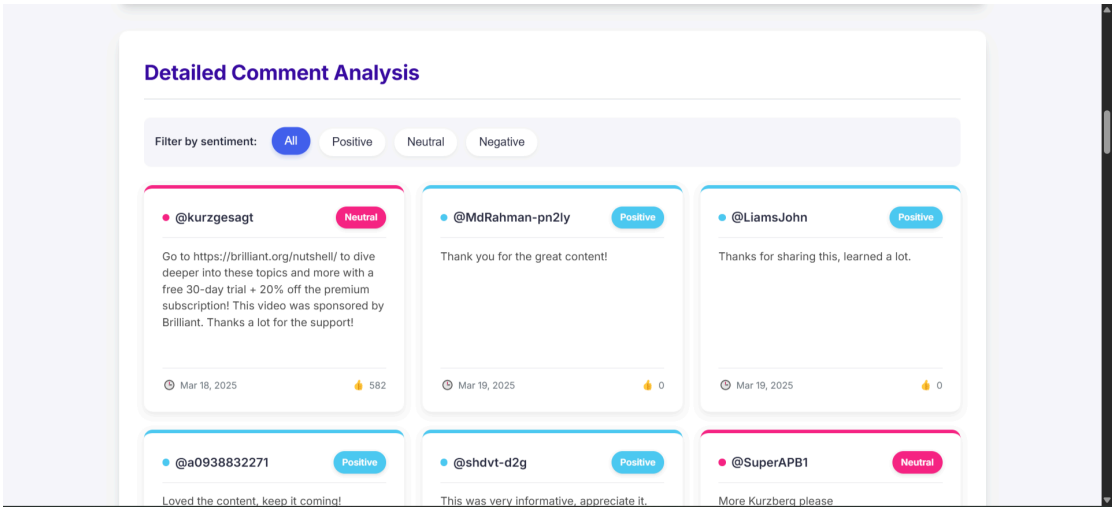
```
):
    try:

        video_id = extract_video_id(video_url)

        comments = await fetch_comments(video_id, comment_count)

        max_comments_to_process = min(len(comments), 100)
        if max_comments_to_process < len(comments):
            logger.warning(
                  f"Limiting analysis to {max_comments_to_process}
comments to avoid API rate limits")

        comments_to_process = comments[:max_comments_to_process]

        batch_size = 20
                                    analyzed_comments    =    await
batch_analyze_sentiment(comments_to_process, model, batch_size)

        sentiment_counts = {
                 "positive": sum(1 for c in analyzed_comments if
c["sentiment"] == "positive"),
                  "neutral": sum(1 for c in analyzed_comments if
c["sentiment"] == "neutral"),
                 "negative": sum(1 for c in analyzed_comments if
c["sentiment"] == "negative")
        }

        return {
            "video_id": video_id,
            "comments": analyzed_comments,
            "stats": sentiment_counts
        }

    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    except Exception as e:
        logger.error(f"Unexpected error: {str(e)}")
        raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    import uvicorn
```

```
            uvicorn.run("app:app",    host="0.0.0.0",    port=8000,
reload=True)
```

## 4. Results/Output:- Entire Screen Shot including Date & Time






## 5. Remarks:-

Git - [Github Repo]

Signature of the Student

(Name of the Student)

Signature of the Lab Coordinator

(Name of the Coordinator)