

Vision Statement: YouTube Summary and Feedback System

Vision

We aim to create a tool that leverages large language models to provide actionable insights about YouTube videos and audience engagement. This system will provide video summaries, sentiment breakdowns, and targeted feedback extraction.

Target Users

- ❖ **Primary Users:** YouTube creators seeking quick, structured insights to improve video engagement and content strategy.
- ❖ **Secondary Users:** Educators, and businesses using YouTube for content outreach.

System Components

1. Data Retrieval & Preprocessing

Purpose: Gather essential data—video transcripts and viewer comments—for meaningful analysis while ensuring data quality.

- ❖ **YouTube API Integration:** Extracts video metadata, transcripts, and comments while managing API request limits.
- ❖ **Preprocessing Pipeline:**
 - Cleans transcript data by removing timestamps, filler words, and extraneous content.
 - Filters out irrelevant comments before any further processing, including:
 - Spam & Bot Comments (e.g., “Check out my channel!”)
 - Off-Topic Discussions (e.g., “This reminds me of my childhood.”)
 - Self-Promotion (e.g., links to unrelated content or advertisements)
 - Unrelated Conversations (e.g., personal arguments or jokes not related to the video)
- ❖ **Data Storage:** Saves clean transcripts and relevant comments.
- ❖ **Operational Constraints:**
 - Videos longer than 1 hour are excluded due to the generation of larger transcripts as time increases.
 - Minimum 100 comments required to ensure a balanced sentiment analysis and constructive feedback.

2. Video Summarization

Purpose: Condense lengthy video transcripts into summaries.

- ❖ **Summarization Engine:**

- Uses ChatGPT API to generate key takeaways.
 - Segments transcript intelligently adhere to API processing limits.
- ❖ **Format Handling:** Output summaries in paragraphs for clarity.

3. Sentiment Analysis

Purpose: Classify audience sentiment to understand viewer engagement and reactions.

- ❖ **Sentiment Classification:**

- Uses VADER (Valence Aware Dictionary and sEntiment Reasoner) for fast, reliable scoring.
- Labels comments as positive, negative, or neutral based on textual sentiment patterns.

4. Constructive Feedback Extraction

Purpose: Identify patterns in comments to highlight content strengths and areas needing improvement.

- ❖ **ChatGPT API for Pros & Cons Analysis:**

- Highlights specific content strengths (e.g., “Great storytelling” or “Excellent editing”).
- Identifies areas for improvement (e.g., “Audio quality needs enhancement” or “More concise explanations needed”).

- ❖ **Structured Report Generation:**

- Provides a “What’s Working” vs. “Needs Improvement” section to streamline creator insights.

5. User Interface & Report Generation

Purpose: Home page as webpage 1 and present results including video summary, sentiment analysis and constructive feedback(strengths and weaknesses) in webpage 2.

- ❖ **Home Page:** Allows users to paste a YouTube URL and initiate analysis with minimal effort.
- ❖ **Results Presentation:** Provides structured summaries, sentiment visuals, and feedback highlights in a clean layout in a new webpage.

Competitive Advantage

Most existing tools—like YouTube Studio, VidIQ, and TubeBuddy—focus on SEO metrics and engagement analytics but lack deep audience insight generation. This system goes further by offering:

- ❖ AI-powered video summarization.
- ❖ Structured audience sentiment analysis beyond basic engagement metrics.

- ❖ ChatGPT-based feedback extraction for direct, actionable insights.
- ❖ Automated irrelevant comment filtering to ensure meaningful data analysis.
- ❖ Designed to be achievable within a semester, with a phased feature rollout.

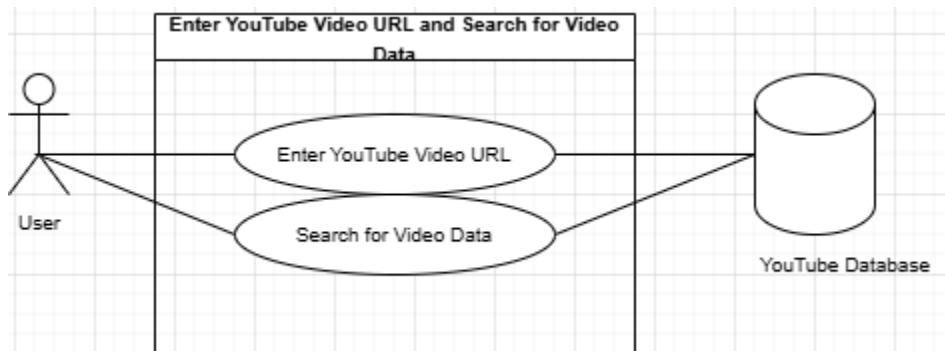
ID	Product Backlog	Estimation	Priority
1	As a User, I can enter a YouTube video URL on the home page and click “Search” to initiate video data retrieval (metadata including title, description and duration), transcript, and comments).	3	1
2	The Web Server can validate the YouTube URL format and extract the video ID.	2	2
3	The Web Server can retrieve video metadata (e.g., title, description, duration) from the YouTube API using the video ID.	3	3
4	The Web Server can retrieve the video transcript from the YouTube API using the video ID.	3	4
5	The Web Server can retrieve at least 100 video comments from the YouTube API using the video ID.	2	5
6	The Web Server will enforce operational constraints by excluding videos longer than 1 hour and ensuring at least 100 comments are available; if not, it notifies the User.	2	6
7	The Web Server can remove timestamps from the retrieved transcript.	1	7
8	The Web Server can remove filler words from the retrieved transcript.	1	8
9	The Web Server can remove spam content from the retrieved transcript.	2	9
10	The Web Server can remove off-topic remarks from the retrieved transcript.	2	10
11	The Web Server can remove unrelated discussions from the retrieved transcript.	2	11
12	The Web Server can process the cleaned transcript for further analysis.	1	12
13	The Web Server can store the processed transcript data.	1	13
14	The Web Server can send the clean transcript to the ChatGPT API to generate a concise video summary in paragraph form and save it.	5	14
15	VADER sentiment analysis library from Python can define and implement the interface.	2	15
16	The Web Server will use VADER from Python library to classify filtered comments as positive, negative, or neutral, and store the sentiment analysis results.	3	16
17	The ChatGPT API can define and implement the interface for processing filtered comments.	1	17
18	The ChatGPT API can extract actionable “What’s Working” feedback from filtered comments.	1	18
19	The ChatGPT API can extract actionable “Needs Improvement” feedback from filtered comments.	1	19
20	The Web Server can generate a structured feedback report categorizing “What’s Working” and “Needs Improvement” and store the report.	2	20
21	As a User, I can view a results page that displays the generated video summary.	3	21

22	As a User, I can view a results page that displays detailed sentiment analysis results of comments.	3	22
23	As a User, I can view a results page that displays the structured feedback report categorizing “What’s Working” and “Needs Improvement”.	4	23

Estimated: 50 days

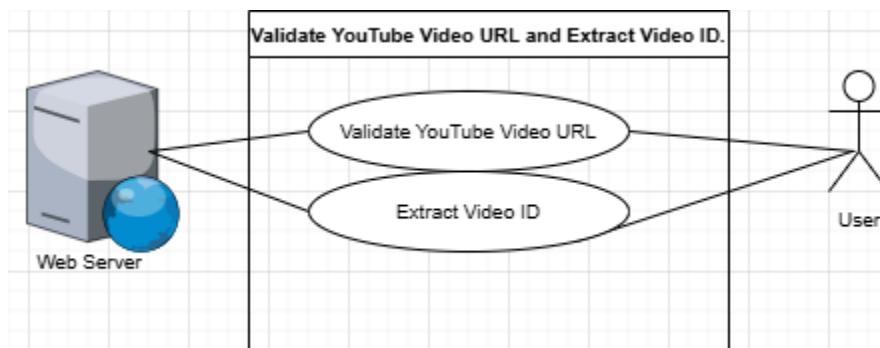
Use Case: Enter YouTube Video URL and Search

Use Case Name	Enter YouTube Video URL and Search
Reference to Product Backlog	PB-1
Scope	YouTube Video Analysis System
Level	User-goal
Primary Actor	User
Stakeholders and Their Interests	User: Needs an uncomplicated method to begin video analysis by entering a URL System: Relies on receiving a properly formatted URL to proceed
Preconditions	User is on the home page with a clearly visible YouTube URL input field network connection is stable
Success Criteria	The system accepts the URL and submits it for further processing
Main Success Scenario	1. User navigates to the home page 2. User enters a YouTube video URL 3. User clicks "Search" 4. The URL is submitted to the system
Extensions	If no URL is entered, display "URL required" if the URL appears malformed, prompt the user to correct it
Special Requirements	Input must strictly follow standard YouTube URL format
Technology and Data Variations	Web-based interface accessible via common browsers URL data is in text format
Frequency	Each time a user presses a search button
Open Issues	None



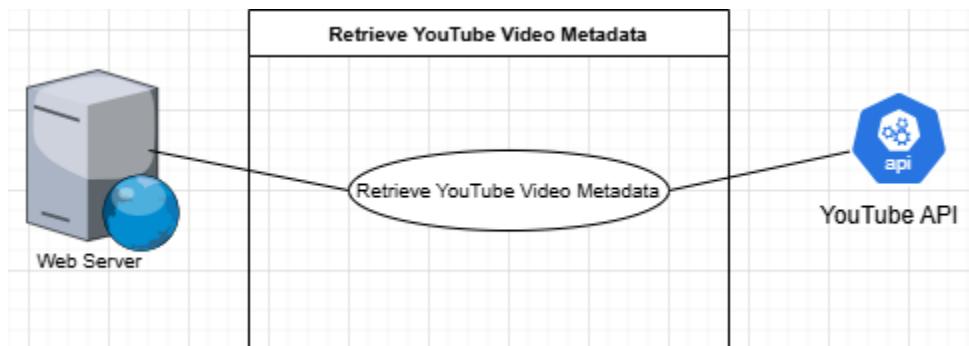
Use Case: Validate YouTube Video URL and Extract Video ID

Use Case Name	Validate YouTube Video URL and Extract Video ID
Reference to Product Backlog	PB-2
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Expects analysis to proceed only with a valid URL System: Requires a correct video ID to request video data
Preconditions	A YouTube video URL is received from the user (via UC-1) network connection is stable
Success Criteria	The system confirms the URL is valid and successfully extracts the video ID
Main Success Scenario	<ol style="list-style-type: none"> 1. Web Server receives the URL from the user 2. The system verifies the URL format against YouTube standards 3. If valid, the system extracts the video ID 4. The extracted video ID is used to request data from the YouTube API
Extensions	If validation fails, display “Invalid URL format. Please enter a valid YouTube video URL” and prompt re-entry if extraction fails, display an error
Special Requirements	URL must exactly match YouTube’s recognized format
Technology and Data Variations	Data is received and processed in JSON format from the API
Frequency	Each URL submission
Open Issues	None



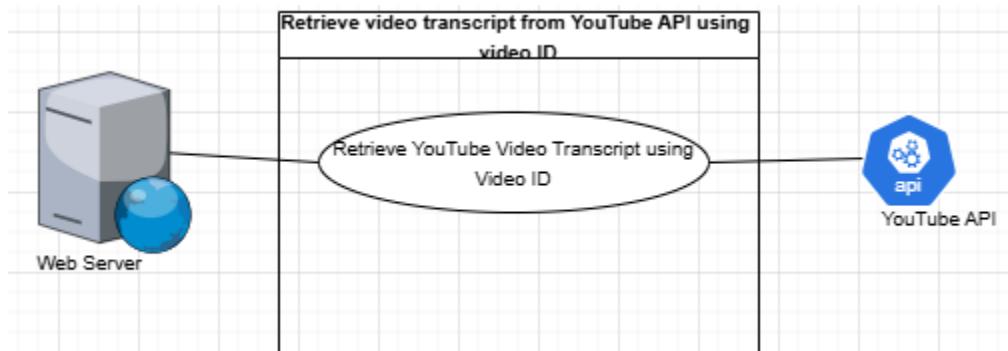
Use Case: Retrieve YouTube Video Metadata

Use Case Name	Retrieve YouTube Video Metadata
Reference to Product Backlog	PB-3
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Needs accurate video details (title, description, duration) System: Must retrieve complete metadata to support subsequent analysis
Preconditions	A valid video ID is available from UC-2 network connection is stable
Success Criteria	The system retrieves complete video metadata including title, description, and duration
Main Success Scenario	<ol style="list-style-type: none"> 1. Web Server sends a request to the YouTube API using the video ID 2. API returns the video metadata 3. The system parses the metadata for use in analysis
Extensions	If the API call fails or metadata is incomplete, display an error
Special Requirements	Metadata must include title, description, and duration
Technology and Data Variations	JSON format from the YouTube API
Frequency	For every valid video URL processed
Open Issues	None



Use Case: Retrieve Video Transcript from YouTube API using Video ID

Use Case Name	Retrieve Video Transcript from YouTube API
Reference to Product Backlog	PB-4
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Requires the transcript text to understand the video content System: Needs the transcript for real-time analysis
Preconditions	A valid video ID is available network connection is stable
Success Criteria	The system retrieves the full transcript in plain text
Main Success Scenario	<ol style="list-style-type: none"> 1. Web Server requests the transcript from the YouTube API using the video ID 2. API returns the transcript 3. The system obtains the transcript for analysis
Extensions	If the transcript is unavailable or the API fails, display an error
Special Requirements	Transcript must be provided in plain text
Technology and Data Variations	Transcript data may be delivered as plain text or in JSON
Frequency	For each video that includes a transcript
Open Issues	None



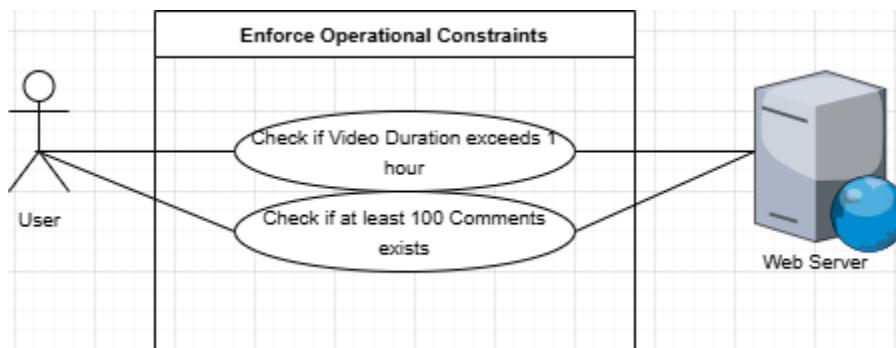
Use Case: Retrieve YouTube Video Comments

Use Case Name	Retrieve YouTube Video Comments from YouTube API
Reference to Product Backlog	PB-5
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Requires a sufficient number of comments to ensure meaningful analysis System: Must retrieve at least 100 comments
Preconditions	A valid video ID is available network connection is stable
Success Criteria	The system retrieves a minimum of 100 comments
Main Success Scenario	<ol style="list-style-type: none"> 1. Web Server sends a comment request to the YouTube API using the video ID 2. API returns a list of comments 3. The system confirms that at least 100 comments have been retrieved
Extensions	If fewer than 100 comments are returned, display “Insufficient comments. Please select a video with at least 100 comments”
Special Requirements	Must retrieve a minimum of 100 comments
Technology and Data Variations	Comments are returned in JSON (possibly paginated)
Frequency	For each valid video URL processed
Open Issues	None



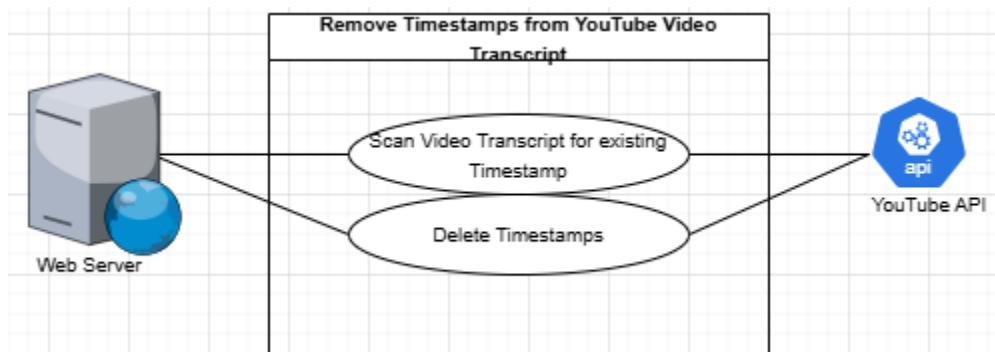
Use Case: Enforce Operational Constraints

Use Case Name	Enforce Operational Constraints
Reference to Product Backlog	PB-6
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Expects that only videos meeting specific criteria are analyzed System: Must enforce video duration and comment count limits to ensure processing quality
Preconditions	Video metadata and comment data are available from UC-3 and UC-5 network connection is stable
Success Criteria	The video's duration is 60 minutes or less and the comment count is 100 or more
Main Success Scenario	<ol style="list-style-type: none"> 1. Web Server reads the video duration from metadata 2. Checks that the duration is \leq 60 minutes 3. Counts the comments 4. If comments are 100 or more, the video is approved for analysis
Extensions	If the video duration exceeds 60 minutes, display "Video duration exceeds the limit. Please select a shorter video" if comments are fewer than 100, display "Not enough comments. Please select a different video"
Special Requirements	Duration must be \leq 60 minutes comment count must be \geq 100
Technology and Data Variations	Numerical data obtained from API responses
Frequency	Each time a video is submitted for analysis
Open Issues	None



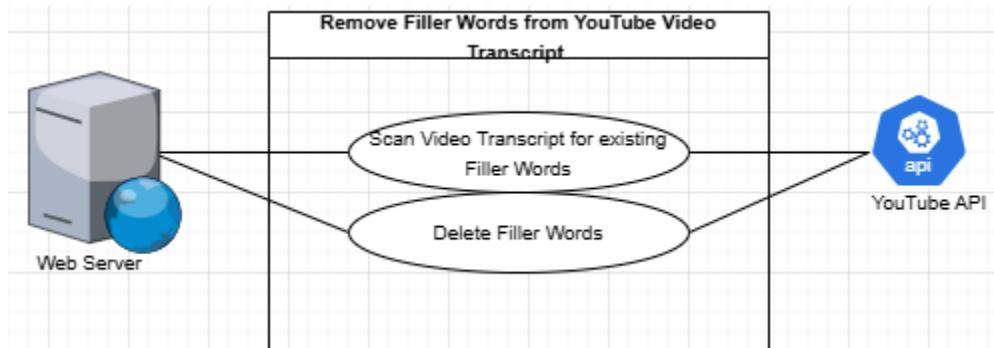
Use Case: Remove Timestamps from Transcript

Use Case Name	Remove Timestamps from Transcript
Reference to Product Backlog	PB-7
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Expects a transcript without distracting time markers System: Must eliminate all timestamp patterns to clarify content
Preconditions	A raw transcript is received network connection is stable
Success Criteria	The transcript is provided with all timestamp markers removed
Main Success Scenario	1. Web Server receives the raw transcript 2. Identifies standard timestamp patterns (e.g., "00:01:23") 3. Removes these patterns from the transcript
Extensions	If any timestamps remain due to non-standard formatting, display an error
Special Requirements	Removal must target only standard timestamp formats without affecting the dialogue
Technology and Data Variations	Utilizes regular expressions to detect timestamps
Frequency	Each time a transcript is processed
Open Issues	None



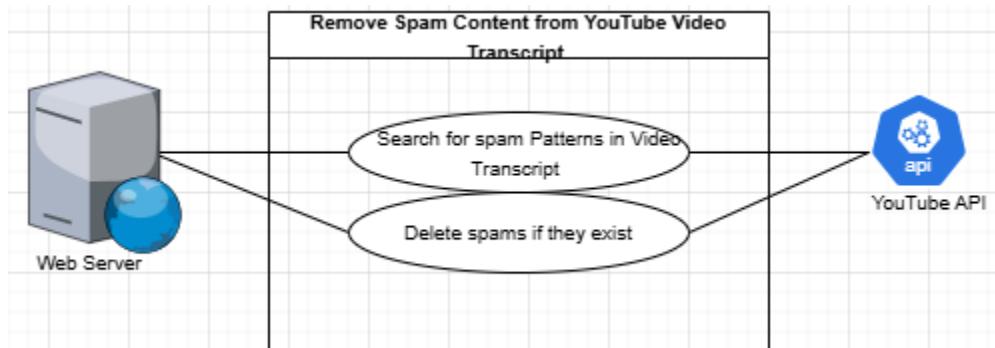
Use Case: Remove Filler Words from Transcript

Use Case Name	Remove Filler Words from Transcript
Reference to Product Backlog	PB-8
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Requires a transcript that is free of non-essential verbal fillers System: Must remove a specified set of filler words to enhance clarity
Preconditions	A raw transcript is received network connection is stable
Success Criteria	The transcript is presented without any filler words (e.g., "um", "uh")
Main Success Scenario	1. Web Server receives the raw transcript 2. Identifies filler words based on a predetermined list 3. Removes all occurrences of these words
Extensions	If any filler words remain, display an error
Special Requirements	Only the words in the specified list are to be removed without altering the meaning
Technology and Data Variations	Uses a text search algorithm based on a fixed list
Frequency	Each time a transcript is processed
Open Issues	None



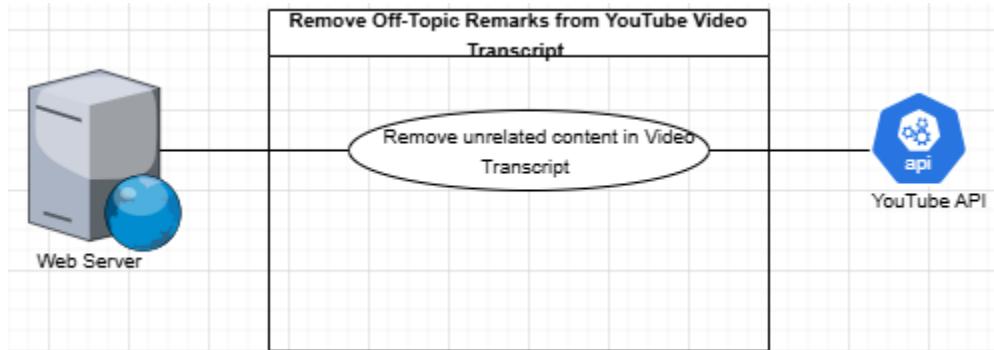
Use Case: Remove Spam Content from Transcript

Use Case Name	Remove Spam Content from Transcript
Reference to Product Backlog	PB-9
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Requires a transcript devoid of promotional or repetitive content System: Must remove text that qualifies as spam according to predefined criteria
Preconditions	A raw transcript is received network connection is stable
Success Criteria	The transcript is cleansed of all content identified as spam
Main Success Scenario	1. Web Server receives the raw transcript 2. Scans the text for content matching the spam criteria 3. Removes the identified spam segments
Extensions	If spam is not entirely removed, display an error
Special Requirements	Removal must target only text meeting the established spam rules
Technology and Data Variations	Uses text matching algorithms
Frequency	Each time a transcript is processed
Open Issues	None



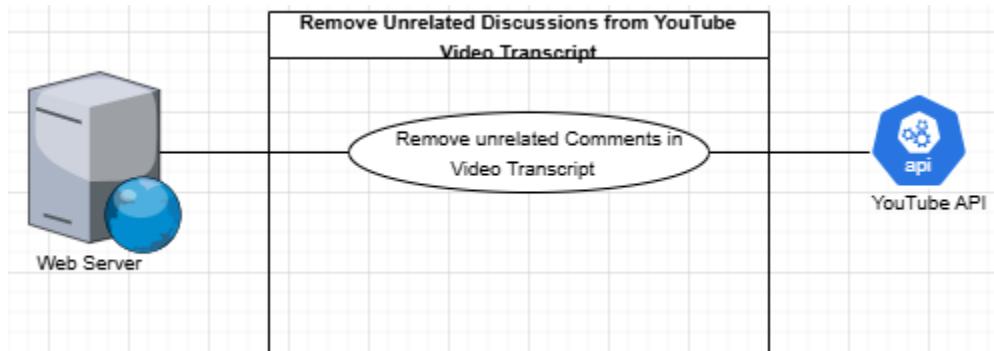
Use Case: Remove Off-Topic Remarks from Transcript

Use Case Name	Remove Off-Topic Remarks from Transcript
Reference to Product Backlog	PB-10
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Expects a transcript that focuses solely on the video's subject matter System: Must remove comments that deviates from the core topic
Preconditions	A raw transcript is received network connection is stable
Success Criteria	The transcript is delivered without any off-topic remarks
Main Success Scenario	1. Web Server receives the raw transcript 2. Identifies segments that do not relate to the main video topic 3. Removes these segments
Extensions	If off-topic content remains, display an error
Special Requirements	Removal must target only non-relevant dialogue
Technology and Data Variations	Uses keyword matching methods
Frequency	Each time a transcript is processed
Open Issues	None



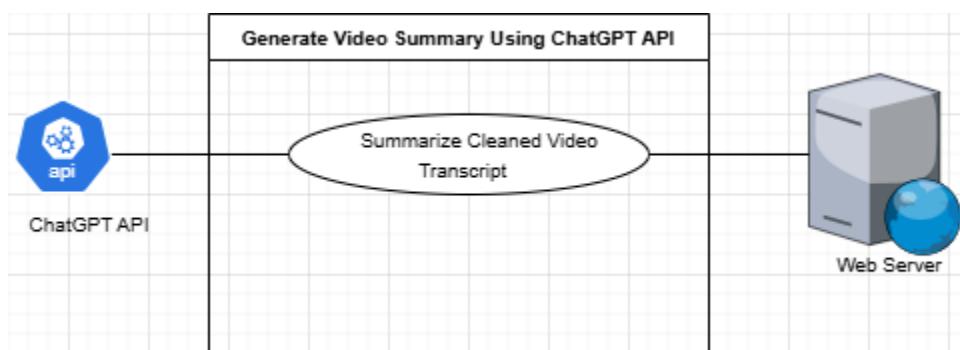
Use Case: Remove Unrelated Discussions from Transcript

Use Case Name	Remove Unrelated Discussions from Transcript
Reference to Product Backlog	PB-11
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Expects a transcript that reflects only the video content System: Must eliminate discussions that are not directly related to the video subject
Preconditions	A raw transcript is received network connection is stable
Success Criteria	The transcript is free from discussions unrelated to the video's subject
Main Success Scenario	1. Web Server receives the raw transcript 2. Identifies segments that are not directly related to the video content 3. Removes these unrelated segments
Extensions	If unrelated discussions persist, display an error
Special Requirements	Removal must target only off-topic discussions
Technology and Data Variations	Uses semantic filtering with predefined criteria
Frequency	Each time a transcript is processed
Open Issues	None



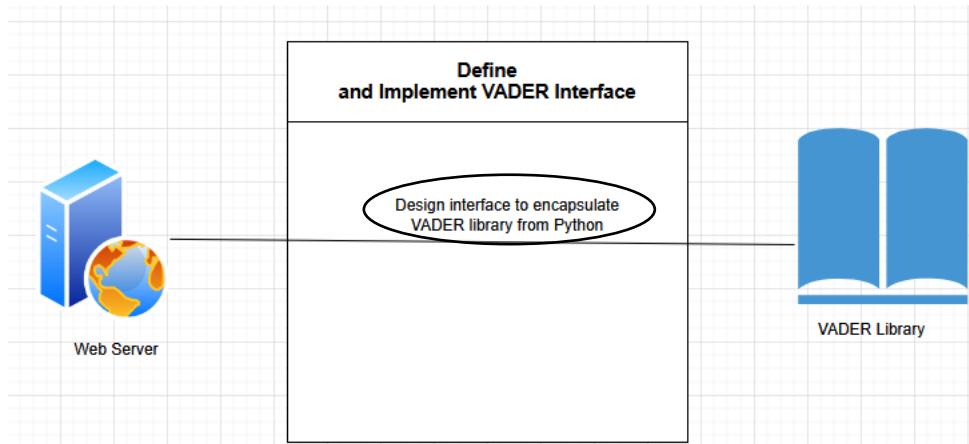
Use Case: Generate Video Summary Using ChatGPT API

Use Case Name	Generate Video Summary Using ChatGPT API
Reference to Product Backlog	PB-14
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Expects a brief and clear summary of the video content System: Must generate the summary on the fly using the clean transcript
Preconditions	A clean transcript is available from previous processing network connection is stable
Success Criteria	A concise, one-paragraph summary is produced and presented
Main Success Scenario	<ol style="list-style-type: none"> 1. Web Server sends the clean transcript to the ChatGPT API 2. ChatGPT API processes the transcript and returns a summary focusing on key topics 3. Web Server displays the summary on the results page
Extensions	If the API call fails, display “Failed to generate summary. Please try again later”
Special Requirements	The summary must be confined to one paragraph and contain only essential information
Technology and Data Variations	None
Frequency	Every time a clean transcript is processed for summarization
Open Issues	None



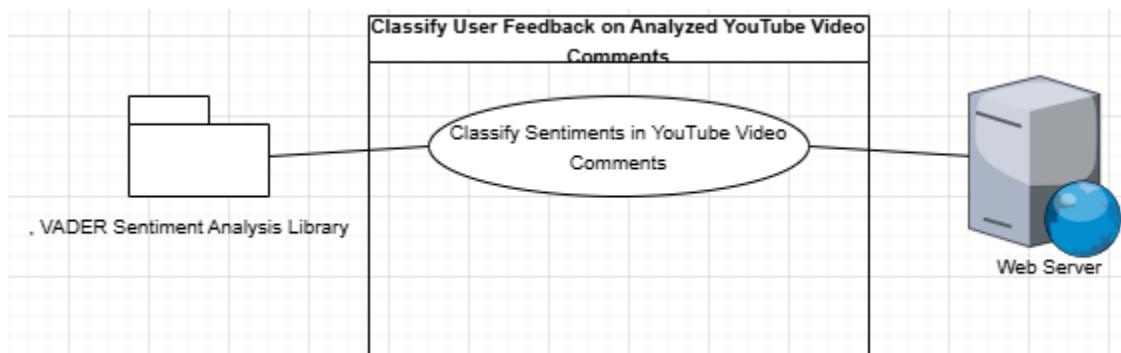
Use Case: Define and Implement VADER Interface

Use Case Name	Define and Implement VADER Interface
Reference to Product Backlog	PB-15
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	System: Requires a dedicated interface for integrating the VADER sentiment analysis library to evaluate comment sentiment
Preconditions	Sentiment analysis requirements are documented network connection is stable
Success Criteria	A VADER interface is implemented and successfully returns sentiment scores for sample text
Main Success Scenario	<ol style="list-style-type: none"> 1. Review VADER library documentation 2. Design an interface encapsulating VADER's functions 3. Implement the interface in the Web Server 4. Validate the interface using sample text inputs
Extensions	If implementation fails, log errors and refine the design until tests pass
Special Requirements	Interface must conform exactly to VADER library specifications
Technology and Data Variations	Developed in Python accepts text input and outputs numerical sentiment scores in JSON
Frequency	Implemented once during system integration maintained thereafter
Open Issues	None



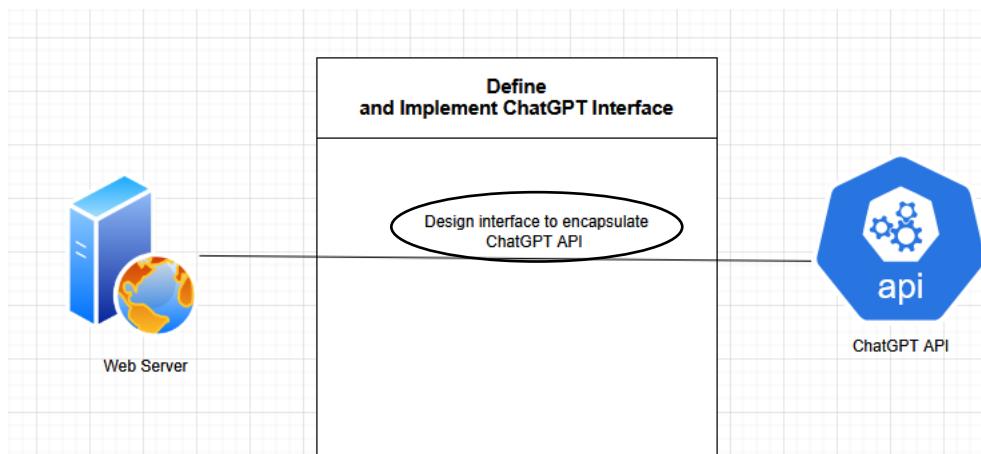
Use Case: Classify Comments with VADER

Use Case Name	Classify Comments with VADER
Reference to Product Backlog	PB-16
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Expects each comment to receive a sentiment label (positive, negative, neutral) System: Must reliably classify comment sentiment
Preconditions	Filtered video comments are available VADER interface is operational network connection is stable
Success Criteria	Every comment is assigned a sentiment classification in real time
Main Success Scenario	<ol style="list-style-type: none"> 1. Web Server iterates over each filtered comment 2. Each comment is submitted to the VADER interface 3. The system receives and records the sentiment classification for each comment
Extensions	If classification fails for any comment, log the error and continue processing remaining comments
Special Requirements	Predefined sentiment thresholds must be applied consistently
Technology and Data Variations	Utilizes the VADER Python library output is in JSON format
Frequency	Each time a new batch of comments is processed
Open Issues	None



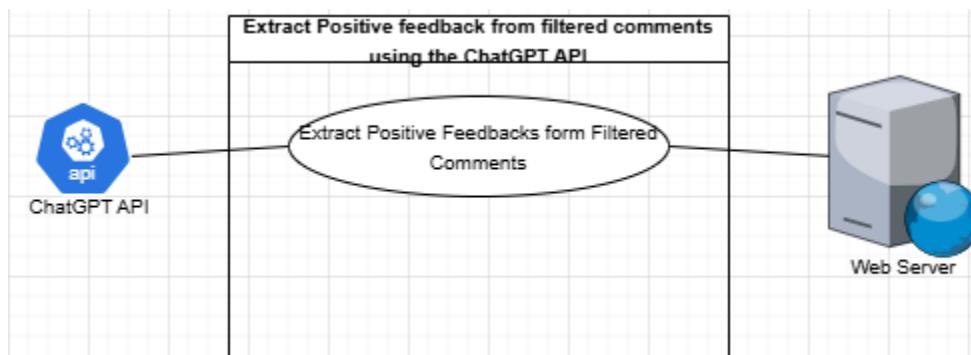
Use Case: Define ChatGPT Interface for Comment Processing

Use Case Name	Define ChatGPT Interface for Comment Processing
Reference to Product Backlog	PB-17
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	System: Requires a robust interface for processing comments via the ChatGPT API for deriving feedback
Preconditions	Comment processing requirements are clearly defined network connection is stable
Success Criteria	The ChatGPT interface is implemented and returns expected outputs for sample comment inputs
Main Success Scenario	<ol style="list-style-type: none"> 1. Review ChatGPT API documentation 2. Design an interface for processing filtered comments 3. Implement the interface within the Web Server 4. Test with sample inputs to verify correct output
Extensions	If testing fails, log errors and adjust the design until successful
Special Requirements	Interface must comply with ChatGPT API specifications
Technology and Data Variations	Uses HTTP API calls with JSON responses
Frequency	Implemented once during integration
Open Issues	None



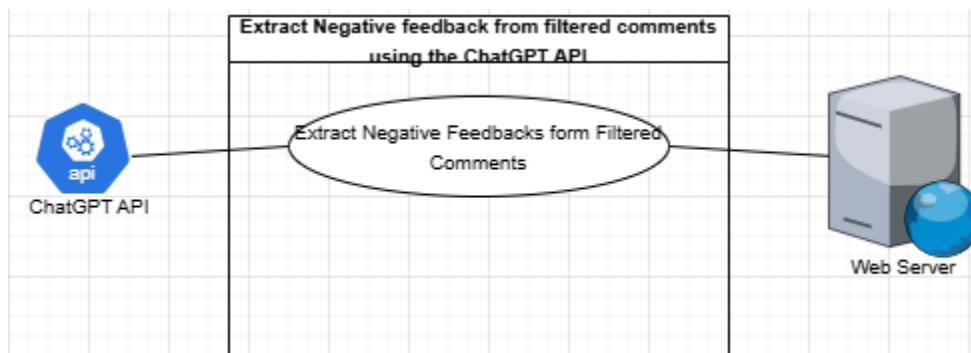
Use Case: Extract “What’s Working” Feedback via ChatGPT

Use Case Name	Extract “What’s Working” Feedback via ChatGPT
Reference to Product Backlog	PB-18
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	ChatGPT API (accessed via Web Server)
Stakeholders and Their Interests	User: Expects to see specific positive aspects from video comments System: Must extract feedback that highlights the strengths reflected in the comments
Preconditions	Filtered comments are available ChatGPT interface is operational network connection is stable
Success Criteria	Positive feedback is extracted and made available for immediate display
Main Success Scenario	1. Web Server sends the filtered comments to the ChatGPT API 2. ChatGPT API processes the comments and extracts feedback on positive aspects 3. The extracted positive feedback is presented on the results page
Extensions	If extraction fails, display an error message
Special Requirements	Feedback must include only positive observations without mixing in neutral or negative remarks
Technology and Data Variations	Uses HTTP API with JSON responses
Frequency	Each time new comment data is processed
Open Issues	None



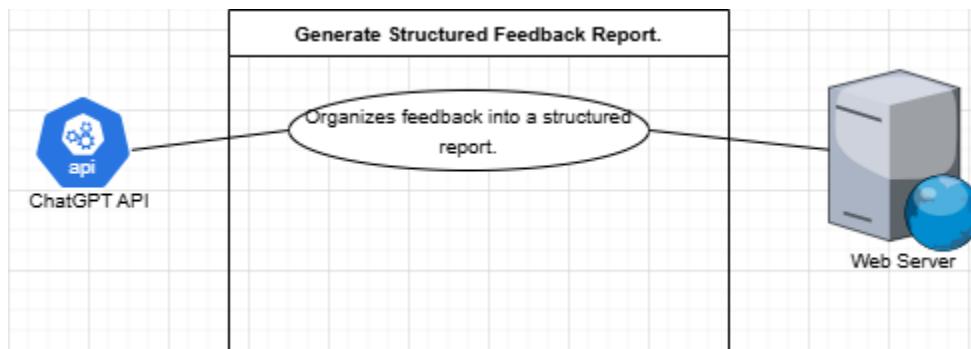
Use Case: Extract “Needs Improvement” Feedback via ChatGPT

Use Case Name	Extract “Needs Improvement” Feedback via ChatGPT
Reference to Product Backlog	PB-19
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	ChatGPT API (accessed via Web Server)
Stakeholders and Their Interests	User: Expects clear suggestions for improvement based on comment analysis System: Must extract feedback that identifies specific areas where the video content can be improved
Preconditions	Filtered comments are available ChatGPT interface is operational network connection is stable
Success Criteria	Feedback focused on improvement areas is extracted and immediately available
Main Success Scenario	<ol style="list-style-type: none"> 1. Web Server sends the filtered comments to the ChatGPT API 2. ChatGPT API processes the comments and extracts feedback on improvement needs 3. The extracted improvement feedback is presented on the results page
Extensions	If extraction fails, display an error message
Special Requirements	Feedback must include only improvement suggestions, clearly separated from positive feedback
Technology and Data Variations	Uses HTTP API with JSON responses
Frequency	Each time new comment data is processed
Open Issues	None



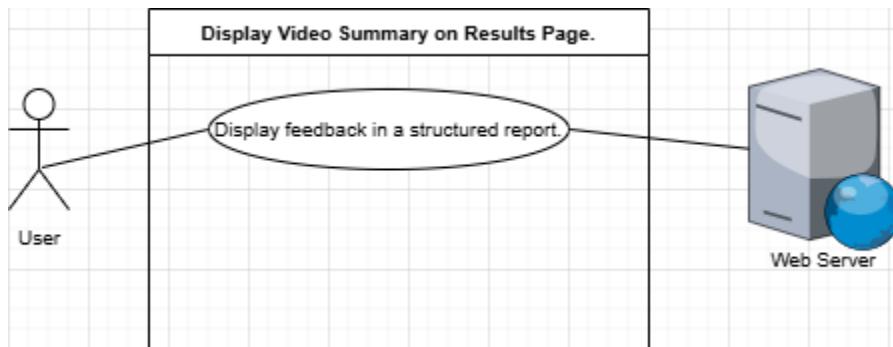
Use Case: Generate Structured Feedback Report

Use Case Name	Generate Structured Feedback Report
Reference to Product Backlog	PB-20
Scope	YouTube Video Analysis System
Level	System-level
Primary Actor	Web Server
Stakeholders and Their Interests	User: Expects a report that clearly distinguishes positive feedback from improvement suggestions System: Must generate the report dynamically and in an organized format
Preconditions	"What's Working" and "Needs Improvement" feedback are available network connection is stable
Success Criteria	A structured report with two clearly defined sections is generated and displayed
Main Success Scenario	1. Web Server retrieves both positive and improvement feedback 2. Organizes the feedback into two separate sections labeled "What's Working" and "Needs Improvement" 3. Displays the report on the results page
Extensions	If report generation fails, display an error message
Special Requirements	The report must clearly separate the two types of feedback without overlap
Technology and Data Variations	Output is rendered in structured text via the web interface
Frequency	Each time comment feedback is processed
Open Issues	None



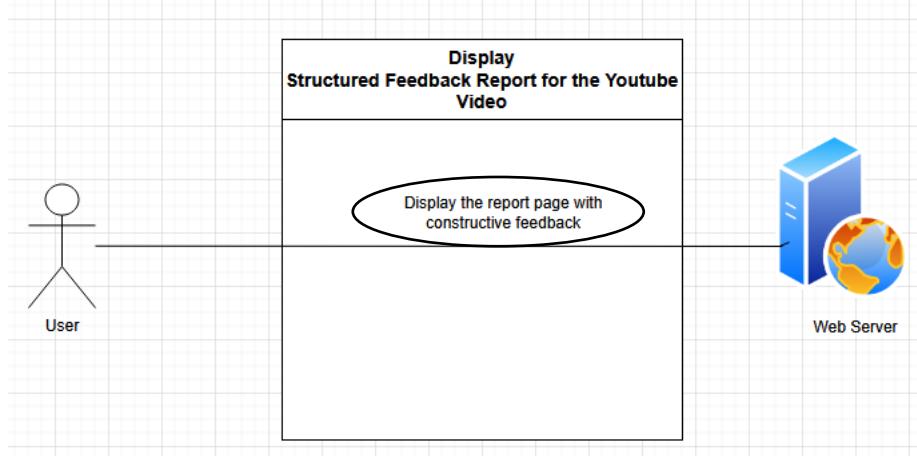
Use Case: Display Video Summary

Use Case Name	Display Video Summary
Reference to Product Backlog	PB-21
Scope	YouTube Video Analysis System
Level	User-goal
Primary Actor	User
Stakeholders and Their Interests	User: Expects a succinct summary of the video content System: Must present the summary clearly and succinctly
Preconditions	Video summary is generated from UC-14 network connection is stable
Success Criteria	Video summary is displayed as a single, clear paragraph on the results page
Main Success Scenario	1. Web Server retrieves the video summary 2. Formats the summary into one clear paragraph 3. Displays the summary on the results page
Extensions	If the summary is not available, display an error message
Special Requirements	Summary must be confined to one paragraph and cover only the key points
Technology and Data Variations	Standard web display format
Frequency	Every time a video is analyzed
Open Issues	None



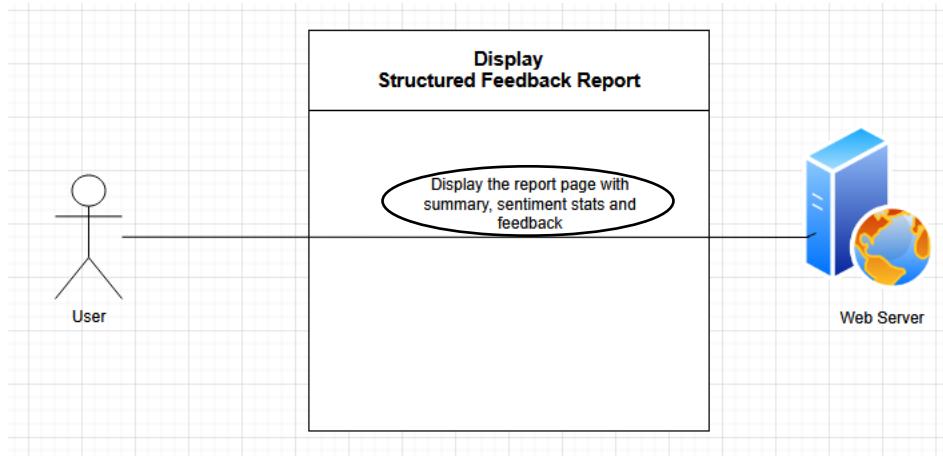
Use Case: Display Sentiment Analysis Results

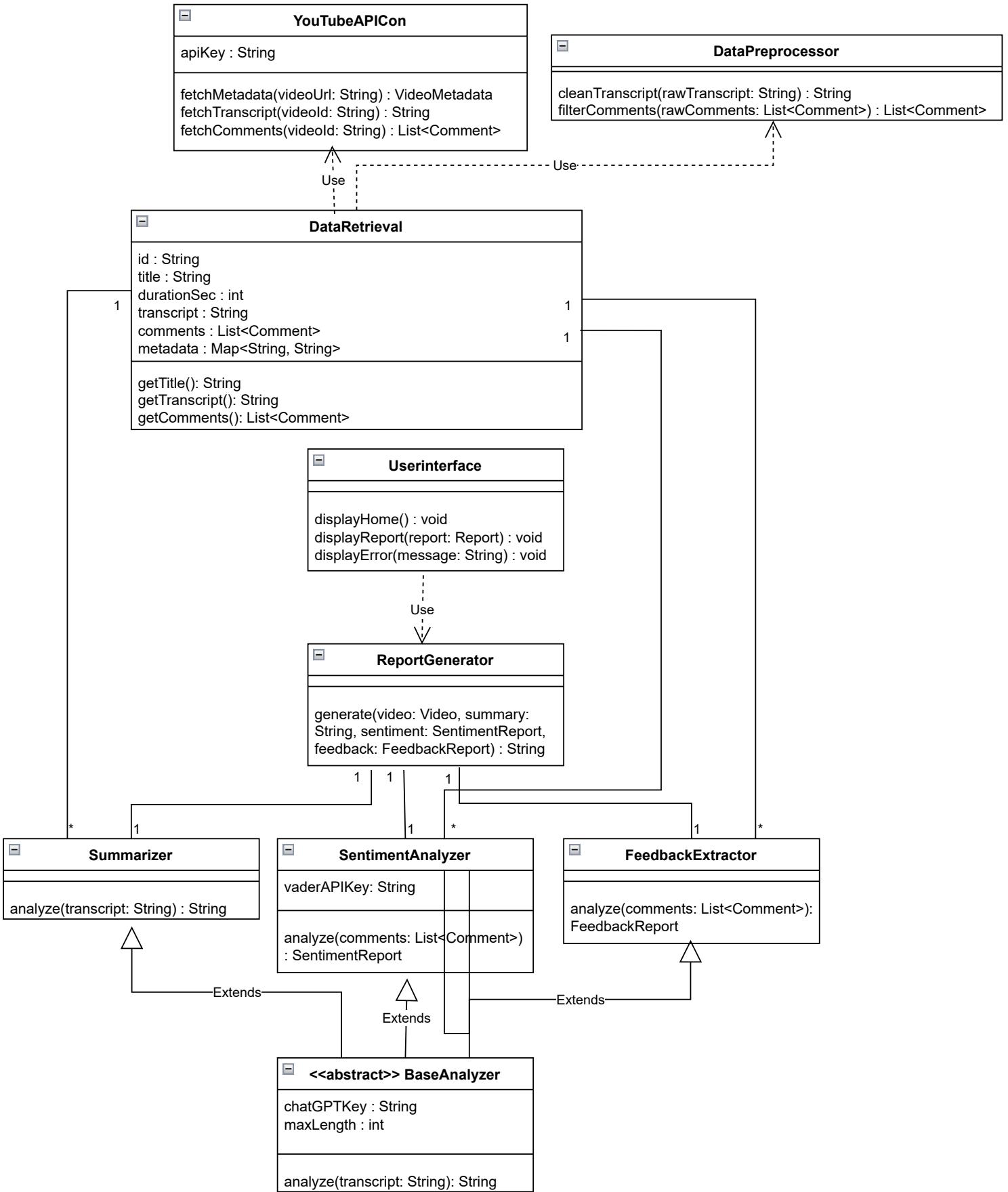
Use Case Name	Display Sentiment Analysis Results
Reference to Product Backlog	PB-22
Scope	YouTube Video Analysis System
Level	User-goal
Primary Actor	User
Stakeholders and Their Interests	<p>User: Expects to see clear sentiment classifications (positive, negative, neutral) for comments</p> <p>System: Must present sentiment data in an understandable manner</p>
Preconditions	Sentiment analysis results from UC-16 are available network connection is stable
Success Criteria	Sentiment analysis results are accurately displayed on the results page
Main Success Scenario	<ol style="list-style-type: none"> 1. Web Server retrieves sentiment analysis results 2. Formats the results with clear labels for positive, negative, and neutral 3. Displays the results on the results page
Extensions	If sentiment results are missing or unclear, display an error message
Special Requirements	Results must clearly indicate each comment's sentiment category
Technology and Data Variations	Standard web display format
Frequency	Every time comment sentiment is processed
Open Issues	None



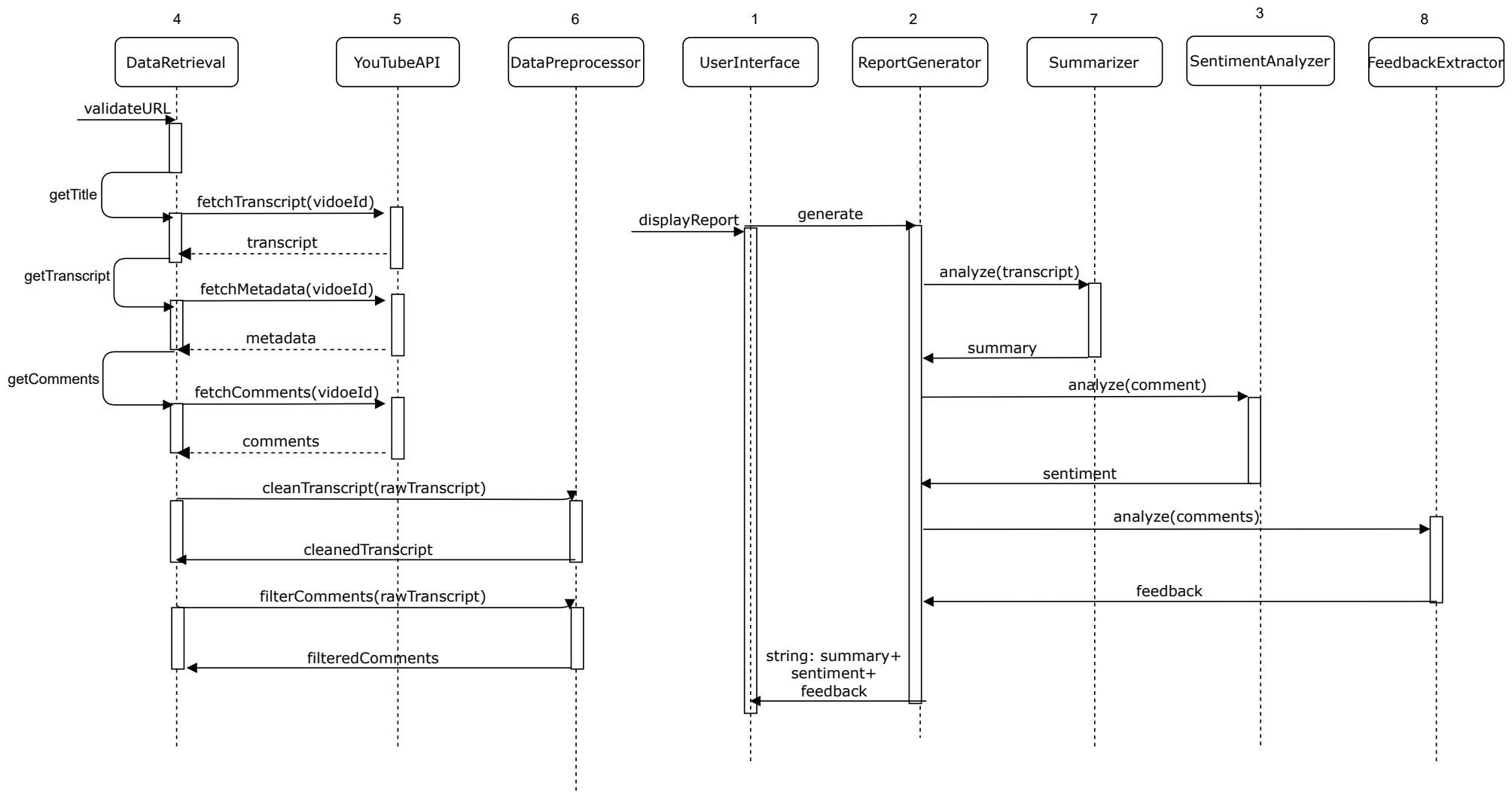
Use Case: Display Structured Feedback Report

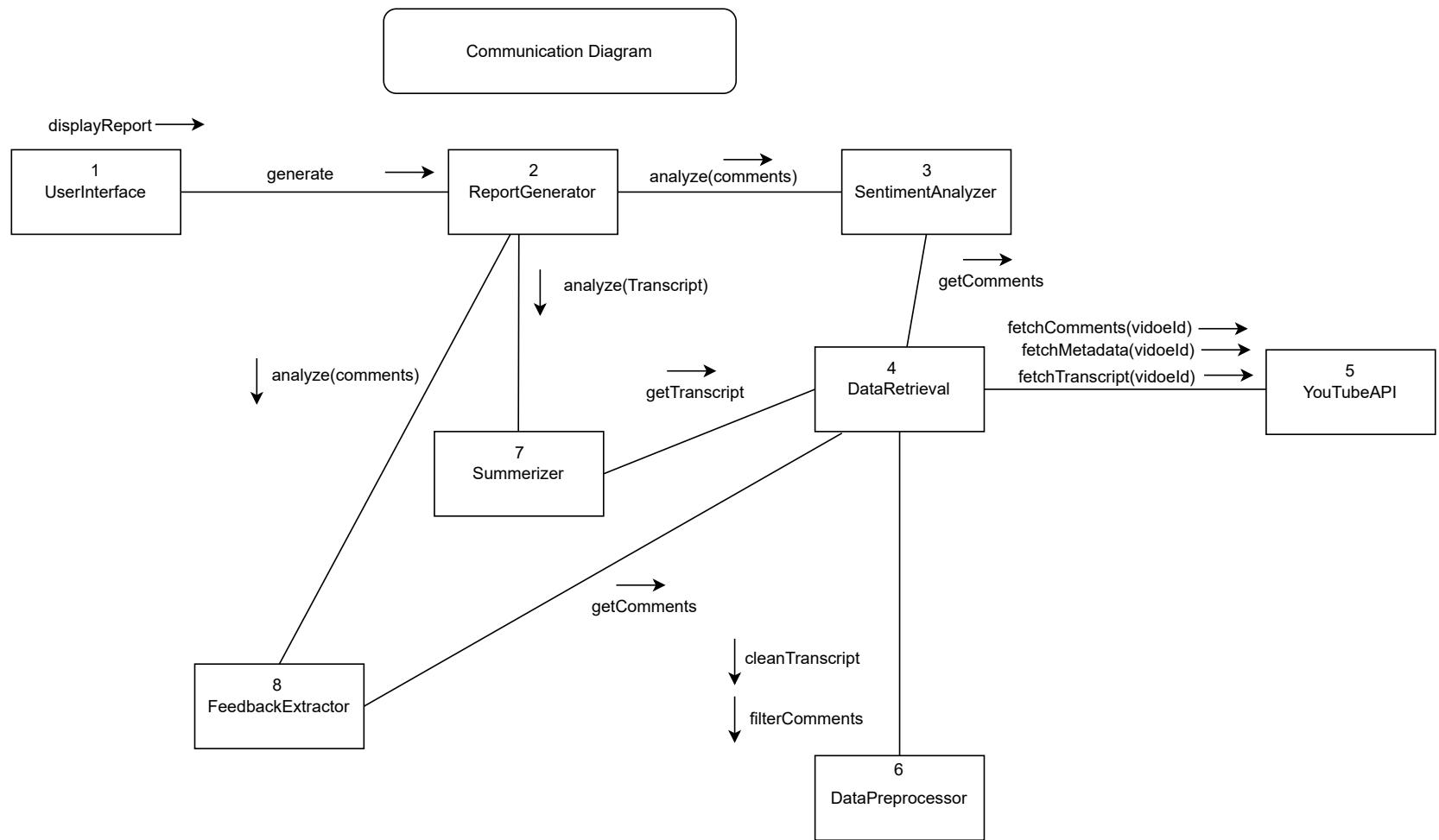
Use Case Name	Display Structured Feedback Report
Reference to Product Backlog	PB-23
Scope	YouTube Video Analysis System
Level	User-goal
Primary Actor	User
Stakeholders and Their Interests	User: Expects a detailed report that clearly separates positive feedback from improvement suggestions System: Must generate and display the report in an organized manner
Preconditions	Feedback from UC-18 and UC-19 is available network connection is stable
Success Criteria	The feedback report is displayed with two distinct sections labeled "What's Working" and "Needs Improvement"
Main Success Scenario	<ol style="list-style-type: none"> Web Server retrieves the positive and improvement feedback Organizes the feedback into two distinct sections Displays the structured report on the results page
Extensions	If the report cannot be generated, display an error message
Special Requirements	Report must clearly delineate the two types of feedback
Technology and Data Variations	Standard web display format
Frequency	Every time a video analysis is completed
Open Issues	None





Sequence Diagram

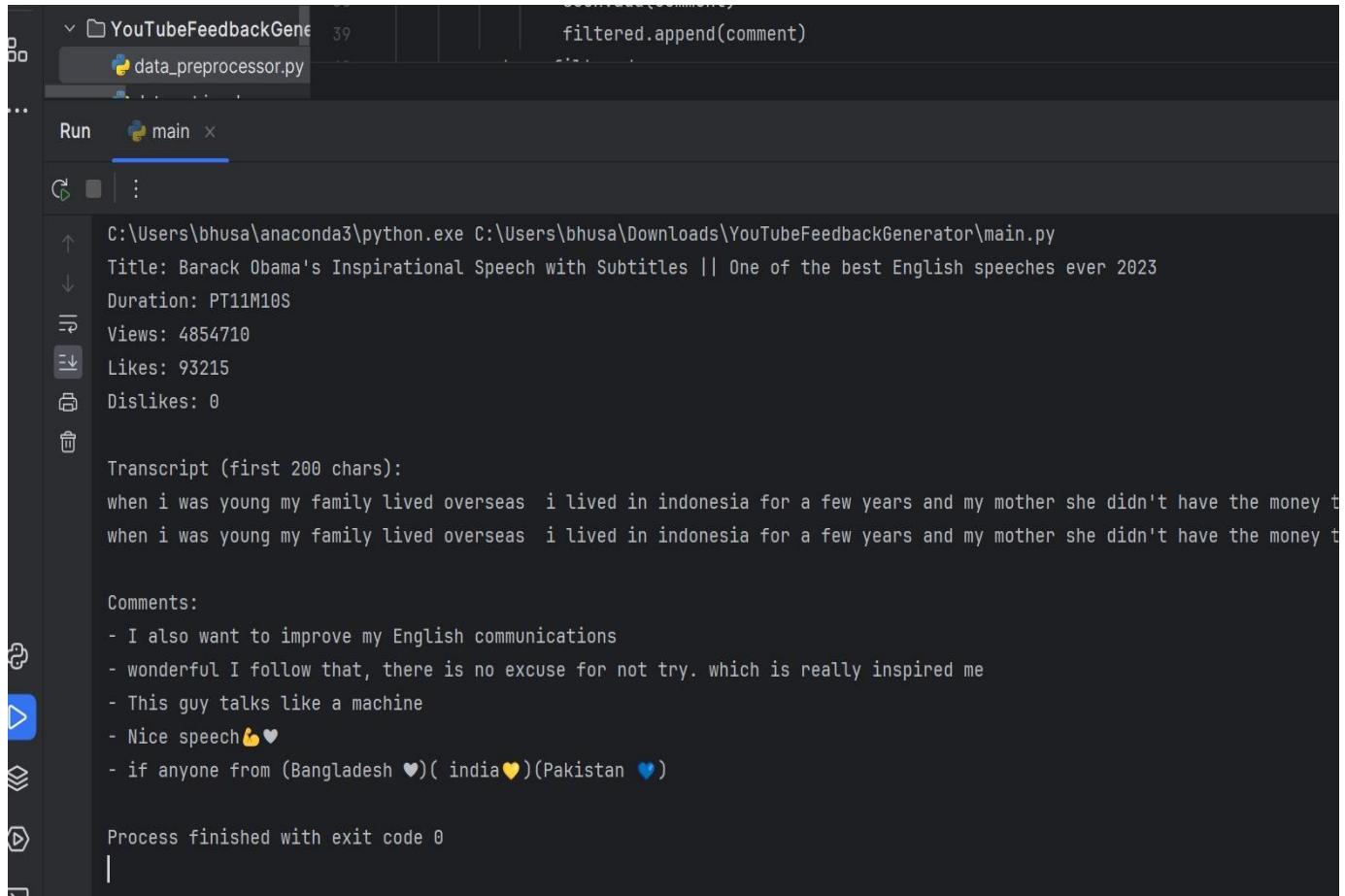




Implementation of Sprint 1:

- **Extract user feedback from comments using API**
- **URL validation and meta data retrieval:**
- **Transcript and comment retrieval:** fetch video transcript, retrieve at least 100 comments from APA and exceeds 1 hour.

Output:



```
filtered.append(comment)
...
Run  main x
C:\Users\bhusa\anaconda3\python.exe C:\Users\bhusa\Downloads\YouTubeFeedbackGenerator\main.py
Title: Barack Obama's Inspirational Speech with Subtitles || One of the best English speeches ever 2023
Duration: PT11M10S
Views: 4854710
Likes: 93215
Dislikes: 0
Transcript (first 200 chars):
when i was young my family lived overseas i lived in indonesia for a few years and my mother she didn't have the money t
when i was young my family lived overseas i lived in indonesia for a few years and my mother she didn't have the money t

Comments:
- I also want to improve my English communications
- wonderful I follow that, there is no excuse for not try. which is really inspired me
- This guy talks like a machine
- Nice speech❤️
- if anyone from (Bangladesh ❤️)( india❤️)(Pakistan ❤️)

Process finished with exit code 0
```

The output displays YouTube data retrieved from the YouTube API. The data includes:

- i. Metadata: includes data from YouTube videos such as title, duration, views, likes.
- ii. Transcript: transcript from YouTube video.
- iii. Comments: all comments from YouTube video.

Program Code:

main.py



The image shows a screenshot of a code editor with a dark theme. The file is named 'main.py'. The code itself is as follows:

```
 1  from data_retrieval import DataRetrieval
 2
 3
 4  usage
 5  def main():
 6      api_key = "AIzaSyBD8EDSacRnoKbCfq9riyK2k2th_LYnZks"
 7      video_url = "https://www.youtube.com/watch?v=PGUdWfB8nLg"
 8
 9      dr = DataRetrieval(api_key, video_url)
10
11     try:
12         dr.validate_url()
13         metadata = dr.get_metadata()
14         transcript = dr.get_transcript()
15         comments = dr.get_comments()
16
17         print(f"Title: {metadata['title']}")
18         print(f"Duration: {metadata['duration']}")
19         print(f"Views: {metadata['views']}")
20         print(f"Likes: {metadata['likes']}")
21         print(f"Dislikes: {metadata['dislikes']}")  

22
23         print("\nTranscript (first 200 chars):")
24         print(transcript[:200])
25         print(transcript)
26
27         print("\nComments:")
28         for comment in comments[:5]:
29             print(f"- {comment}")
30
31     except ValueError as e:
32         print(f"Error: {e}")
33     except ConnectionError as e:
34         print(f"Connection error: {e}")
35     except Exception as e:
36         print(f"Unexpected error: {e}")
37
38  if __name__ == "__main__":
39      main()
```

data_preprocessor.py

```
main.py      data_preprocessor.py x

1 import re
2 from typing import List
3
4 usages
5 class DataPreprocessor:
6     1 usage
7         @staticmethod
8             def remove_filler_words(transcript: str) -> str:
9                 """Removes common filler words from the transcript"""
10                fillers = ["um", "uh", "like", "you know", "so", "actually", "basically", "literally", "right"]
11                pattern = r'\b(' + '|'.join(fillers) + r')\b'
12                cleaned_transcript = re.sub(pattern, repl: '', transcript, flags=re.IGNORECASE)
13                return cleaned_transcript
14
15         1 usage
16             @staticmethod
17                 def clean_transcript(raw_transcript: str) -> str:
18                     """Cleans transcript by removing timestamps, filler words, and spam content"""
19                     cleaned = re.sub(pattern: r'\[(\d+:\d+(?:\d+)?)]', repl: '', raw_transcript) # Remove timestamps
20                     cleaned = DataPreprocessor.remove_filler_words(cleaned) # Remove filler words
21                     return cleaned.lower()
22
23         1 usage
24             @staticmethod
25                 def filter_comments(raw_comments: List[str]) -> List[str]:
26                     # Remove duplicate comments
27                     seen = set()
28                     filtered = []
29                     # Remove spam content using regex patterns
30                     spam_patterns = [
31                         r'\b(free|subscribe|click here|visit our website|spam)\b',
32                         r'(http|https)://\S+', # URLs
33                         r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b', # Emails
34                         r'\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b' # IP addresses
35                     ]
36                     for comment in raw_comments:
37                         if comment and comment not in seen:
38                             for pattern in spam_patterns:
39                                 comment = re.sub(pattern, repl: '', comment, flags=re.IGNORECASE)
40                             seen.add(comment)
41                             filtered.append(comment)
42                     return filtered
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
```

data_retrieval.py

```
 1  from youtubeAPICon import YouTubeAPICon
 2  from data_preprocessor import DataPreprocessor
 3  import re
 4
 5
 6  2 usages
 7  class DataRetrieval:
 8      def __init__(self, api_key: str, video_url: str):
 9          self.api = YouTubeAPICon(api_key)
10          self.video_url = video_url
11          self.video_id = None
12
13          1 usage
14      def validate_url(self):
15          """Validate YouTube URL and extract video ID"""
16          pattern = r"^(https://(www\.)?(youtube\.com|youtu\.be)/(watch\?v=)?([a-zA-Z0-9_-]{11})$"
17          match = re.match(pattern, self.video_url)
18          if not match:
19              raise ValueError("Invalid YouTube URL")
20          self.video_id = match.group(4)
21
22          1 usage
23      def get_metadata(self) -> dict:
24          """Fetch and return video metadata"""
25          if not self.video_id:
26              raise RuntimeError("URL validation required before data retrieval")
27          metadata = self.api.fetch_metadata(self.video_id)
28          # Enforce operational constraints
29          if metadata["duration"].startswith("PT") and "H" in metadata["duration"]:
# Checks for 1+ hour videos
30              raise ValueError("Video exceeds the allowed duration of 1 hour")
31
32          return metadata
33
34  2 usages (1 dynamic)
35  def get_transcript(self) -> str:
36      """Fetch and return cleaned transcript"""
37      if not self.video_id:
38          raise RuntimeError("URL validation required before data retrieval")
39      raw_transcript = self.api.fetch_transcript(self.video_id)
40      return DataPreprocessor.clean_transcript(raw_transcript)
41
42      1 usage
43  def get_comments(self) -> list[str]:
44      """Fetch and return filtered comments"""
45      if not self.video_id:
46          raise RuntimeError("URL validation required before data retrieval")
47      raw_comments = self.api.fetch_comments(self.video_id)
48
49      if len(raw_comments) < 100:
50          raise ValueError("Video does not have the required minimum of 100 comments")
51
52      return DataPreprocessor.filter_comments(raw_comments)
```

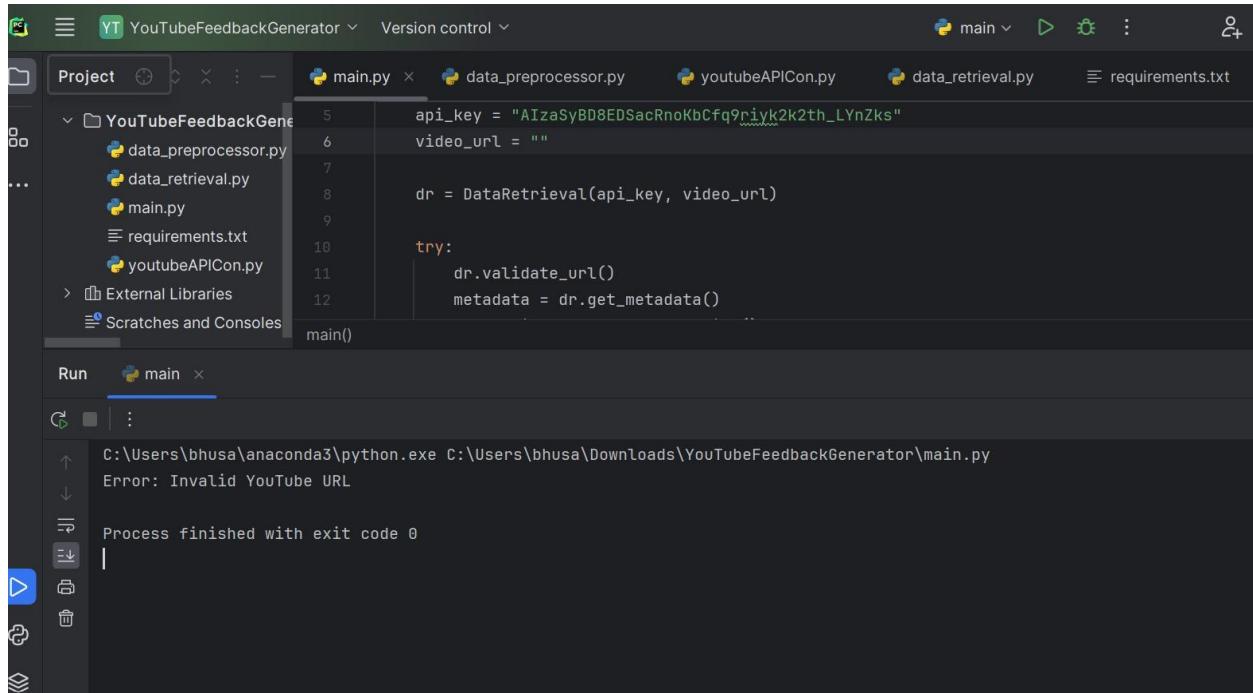
youtubeAPICon.py

```
 1 from googleapiclient.discovery import build
 2 from googleapiclient.errors import HttpError
 3 from youtube_transcript_api import YouTubeTranscriptApi
 4 from typing import Dict, List
 5
 6
 7 2 usages
 8 class YouTubeAPICon:
 9     def __init__(self, api_key: str):
10         self.api_key = api_key
11         self.youtube = build('youtube', 'v3', developerKey=self.api_key)
12
13     1 usage
14     def fetch_metadata(self, video_id: str) -> Dict[str, str]:
15         try:
16             response = self.youtube.videos().list(
17                 part="snippet,contentDetails,statistics",
18                 id=video_id
19             ).execute()
20             items = response.get('items', [])
21             if not items:
22                 raise ValueError("Video not found")
23
24             metadata = items[0]
25             return {
26                 "title": metadata['snippet']['title'],
27                 "duration": metadata['contentDetails']['duration'],
28                 "views": metadata['statistics']['viewCount'],
29                 "likes": metadata['statistics'].get('likeCount', '0'),
30                 "dislikes": metadata['statistics'].get('dislikeCount', '0')
31             }
32         except HttpError as e:
33             raise ConnectionError(f"Metadata API error: {e.resp.status}") from e
34
35 1 usage
```

```
main.py      data_preprocessor.py      data_retrieval.py      youtubeAPICon.py ×
33     def fetch_comments(self, video_id: str, max_comments: int = 3000) -> List[str]:
34         """Fetch up to max_comments (default 3000) top-level comments from YouTube API"""
35         comments = []
36         try:
37             request = self.youtube.commentThreads().list(
38                 part="snippet",
39                 videoId=video_id,
40                 textFormat="plainText",
41                 maxResults=100 # Fetch 100 comments per page (max allowed by YouTube API)
42             )
43             while request and len(comments) < max_comments:
44                 response = request.execute()
45                 for item in response['items']:
46                     comments.append(
47                         item['snippet']['topLevelComment']['snippet']['textDisplay']
48                     )
49                     if len(comments) >= max_comments:
50                         break # Stop fetching more comments once limit is reached
51
52             # Check if there are more pages of comments
53             if 'nextPageToken' in response and len(comments) < max_comments:
54                 request = self.youtube.commentThreads().list(
55                     part="snippet",
56                     videoId=video_id,
57                     textFormat="plainText",
58                     maxResults=100,
59                     pageToken=response['nextPageToken']
60                 )
61             else:
62                 break # No more pages, exit loop
63
64         return comments
65     except HttpError as e:
66         raise ConnectionError(f"Comments API error: {e.resp.status}") from e
67
68     1 usage
69     def fetch_transcript(self, video_id: str) -> str:
70         try:
71             transcript_list = YouTubeTranscriptApi.get_transcript(video_id)
72             transcript = ' '.join([entry['text'] for entry in transcript_list])
73             return transcript
74         except Exception as e:
75             return f"Transcript unavailable: {str(e)}"
YouTubeAPICon > fetch_comments()
```

Test Cases:

Test Case 1: When the user searches without any YouTube URL or the URL entered is invalid.



The screenshot shows the PyCharm IDE interface with a dark theme. The project is named "YouTubeFeedbackGenerator". The main.py file is open, showing code related to YouTube API retrieval. The Run tab is selected, showing the command "C:\Users\bhusa\anaconda3\python.exe C:\Users\bhusa\Downloads\YouTubeFeedbackGenerator\main.py". The output window displays the error message "Error: Invalid YouTube URL".

```
api_key = "AIzaSyBD8EDSacRnoKbCf9riyk2k2th_LYnZks"
video_url = ""

dr = DataRetrieval(api_key, video_url)

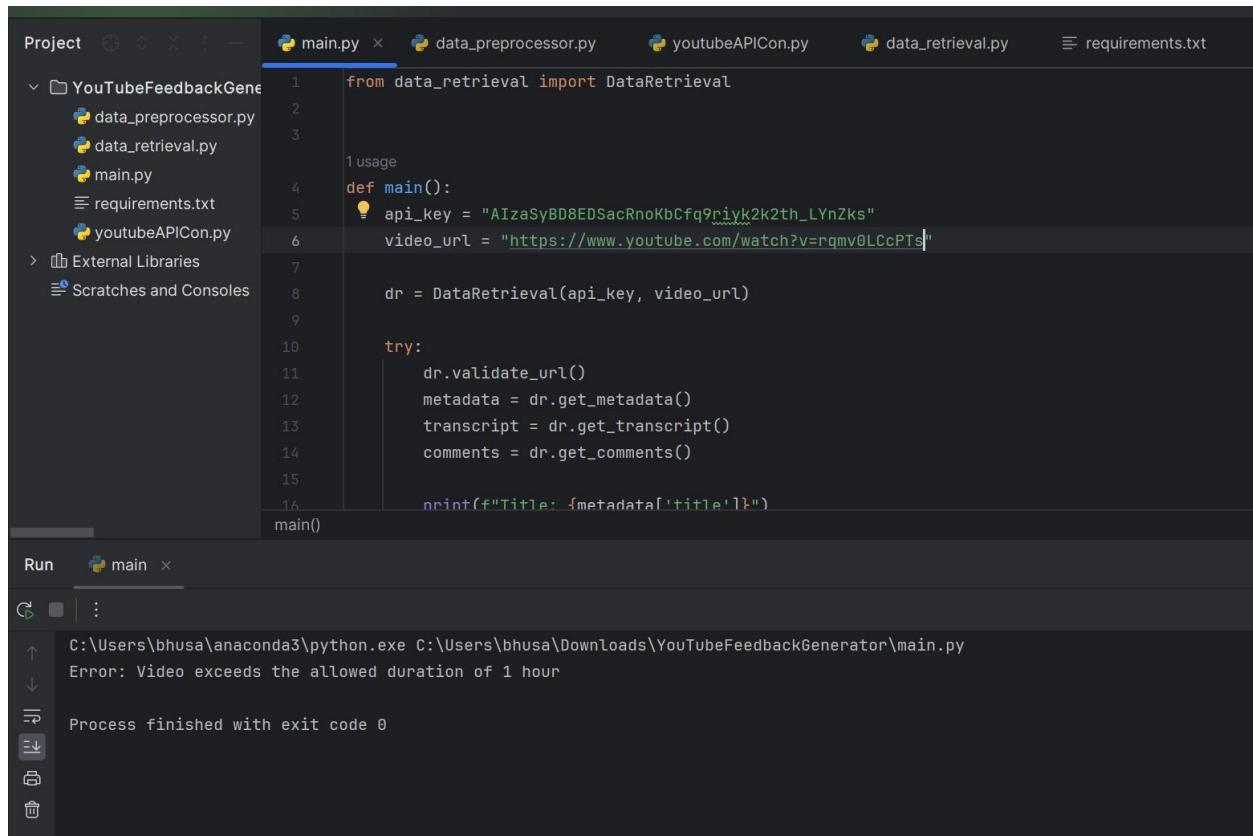
try:
    dr.validate_url()
    metadata = dr.get_metadata()

main()
```

```
C:\Users\bhusa\anaconda3\python.exe C:\Users\bhusa\Downloads\YouTubeFeedbackGenerator\main.py
Error: Invalid YouTube URL

Process finished with exit code 0
```

Test Case 2: When YouTube video duration is more than 1 hour.



The screenshot shows the PyCharm IDE interface with a dark theme. The project is named "YouTubeFeedbackGenerator". The main.py file is open, showing code related to YouTube API retrieval. The Run tab is selected, showing the command "C:\Users\bhusa\anaconda3\python.exe C:\Users\bhusa\Downloads\YouTubeFeedbackGenerator\main.py". The output window displays the error message "Error: Video exceeds the allowed duration of 1 hour".

```
from data_retrieval import DataRetrieval

1 usage
def main():
    api_key = "AIzaSyBD8EDSacRnoKbCf9riyk2k2th_LYnZks"
    video_url = "https://www.youtube.com/watch?v=rqmV0LCCPTs"

    dr = DataRetrieval(api_key, video_url)

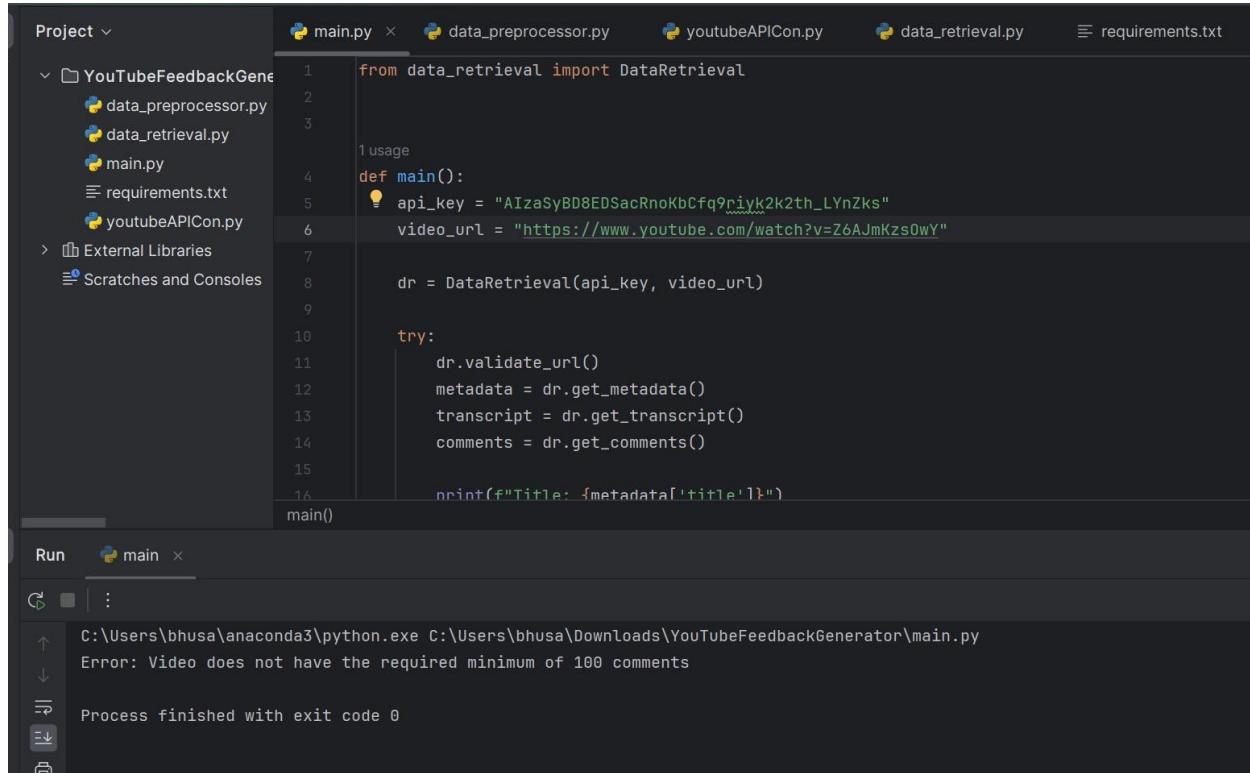
    try:
        dr.validate_url()
        metadata = dr.get_metadata()
        transcript = dr.get_transcript()
        comments = dr.get_comments()

        print(f"Title: {metadata['title']}")
```

```
C:\Users\bhusa\anaconda3\python.exe C:\Users\bhusa\Downloads\YouTubeFeedbackGenerator\main.py
Error: Video exceeds the allowed duration of 1 hour

Process finished with exit code 0
```

Test Case 3: When YouTube video has less than 100 comments. For this instance, the video has 0 comments.



The screenshot shows a Python development environment with the following details:

- Project View:** Shows a project named "YouTubeFeedbackGene" containing files: data_preprocessor.py, data_retrieval.py, main.py, requirements.txt, and youtubeAPICon.py.
- Main Editor:** Displays the content of main.py. The code imports DataRetrieval, defines a main() function, and attempts to retrieve metadata and comments for a specific video URL using a provided API key.
- Run Tab:** Shows the command run: C:\Users\bhusa\anaconda3\python.exe C:\Users\bhusa\Downloads\YouTubeFeedbackGenerator\main.py
- Output Tab:** Displays the error message: "Error: Video does not have the required minimum of 100 comments".
- Status Bar:** Shows "Process finished with exit code 0".

```
from data_retrieval import DataRetrieval

# usage
def main():
    api_key = "AIzaSyBD8EDSacRnoKbCfq9riyk2k2th_LYnZks"
    video_url = "https://www.youtube.com/watch?v=Z6AJmKzs0wY"

    dr = DataRetrieval(api_key, video_url)

    try:
        dr.validate_url()
        metadata = dr.get_metadata()
        transcript = dr.get_transcript()
        comments = dr.get_comments()

        print(f"Title: {metadata['title']}")
    except Exception as e:
        print(f"An error occurred: {e}")

if __name__ == "__main__":
    main()
```

Reflection of Sprint 1:

What worked well:

- ❖ YouTube URL validation and video ID extraction are correctly implemented in validate_url() in data_retrieval.py.
- ❖ Metadata retrieval (title, description, duration) works well, enforcing constraints to exclude videos longer than 1 hour.
- ❖ Transcript retrieval and cleaning successfully remove timestamps and filler words using clean_transcript() in data_preprocessor.py.
- ❖ Comment filtering partially works; duplicates and some spam patterns are removed.

What didn't work well:

- ❖ **Filtering Comments Is Complex:**
Spam detection is challenging since spam formats vary.
Simple regex filters might not be enough—NLP-based techniques should be explored.
Irrelevant comments require topic modeling to determine whether they are related to the video.
- ❖ **Comment processing speed:**
A 3000-comment limit was set to balance performance, but if scaling up, processing time could increase significantly.
Batch processing or asynchronous API calls may be needed for efficiency.

Lessons learnt:

- ❖ Underestimated the complexity of cleaning the data including comments and transcripts.

Impact on project:

- ❖ Could have worked on less User Stories to complete all user stories deliverables promised in Sprint 1.

Invoice (in hours):

Documentation	Ayush – 32, Anthony – 32, Nabin – 32
Research	Ayush – 32, Anthony – 32, Nabin – 34
Design	Ayush – 16, Anthony – 17, Nabin – 16
Code	Ayush – 14, Anthony – 12, Nabin – 12
Testing	Ayush – 8, Anthony – 6, Nabin – 6

Estimated for Remaining Work: As per Product Backlog $50-20 = 30$ days. 5 extra days to complete what didn't work well in Sprint 1.

Total: 35 days

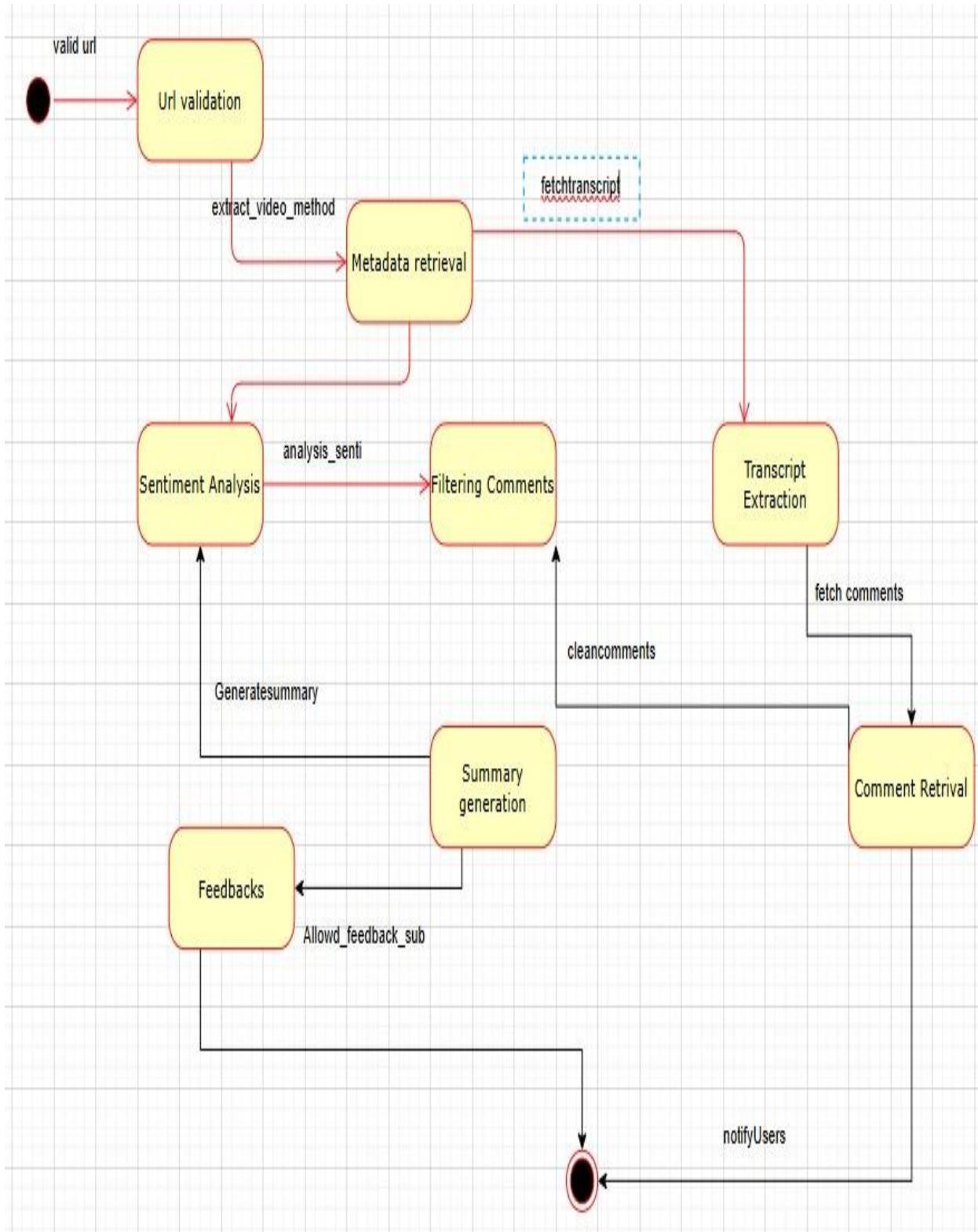
Product Backlogs for Sprint 2 & 3:

ID	Product Backlog	Estimation	Priority
14	The Web Server can send the clean transcript to the ChatGPT API to generate a concise video summary in paragraph form and save it.	5	14
15	Define and implement the interface for the VADER sentiment analysis tool.		15
16	The Web Server will use VADER to classify filtered comments as positive, negative, or neutral, and store the sentiment analysis results.		16
17	Define and implement the interface with the ChatGPT API for processing filtered comments.		17
18	Extract actionable “What’s Working” feedback from filtered comments using the ChatGPT API.	1	18
19	Extract actionable “Needs Improvement” feedback from filtered comments using the ChatGPT API.	1	19
20	Generate a structured feedback report categorizing “What’s Working” and “Needs Improvement” and store the report.	2	20

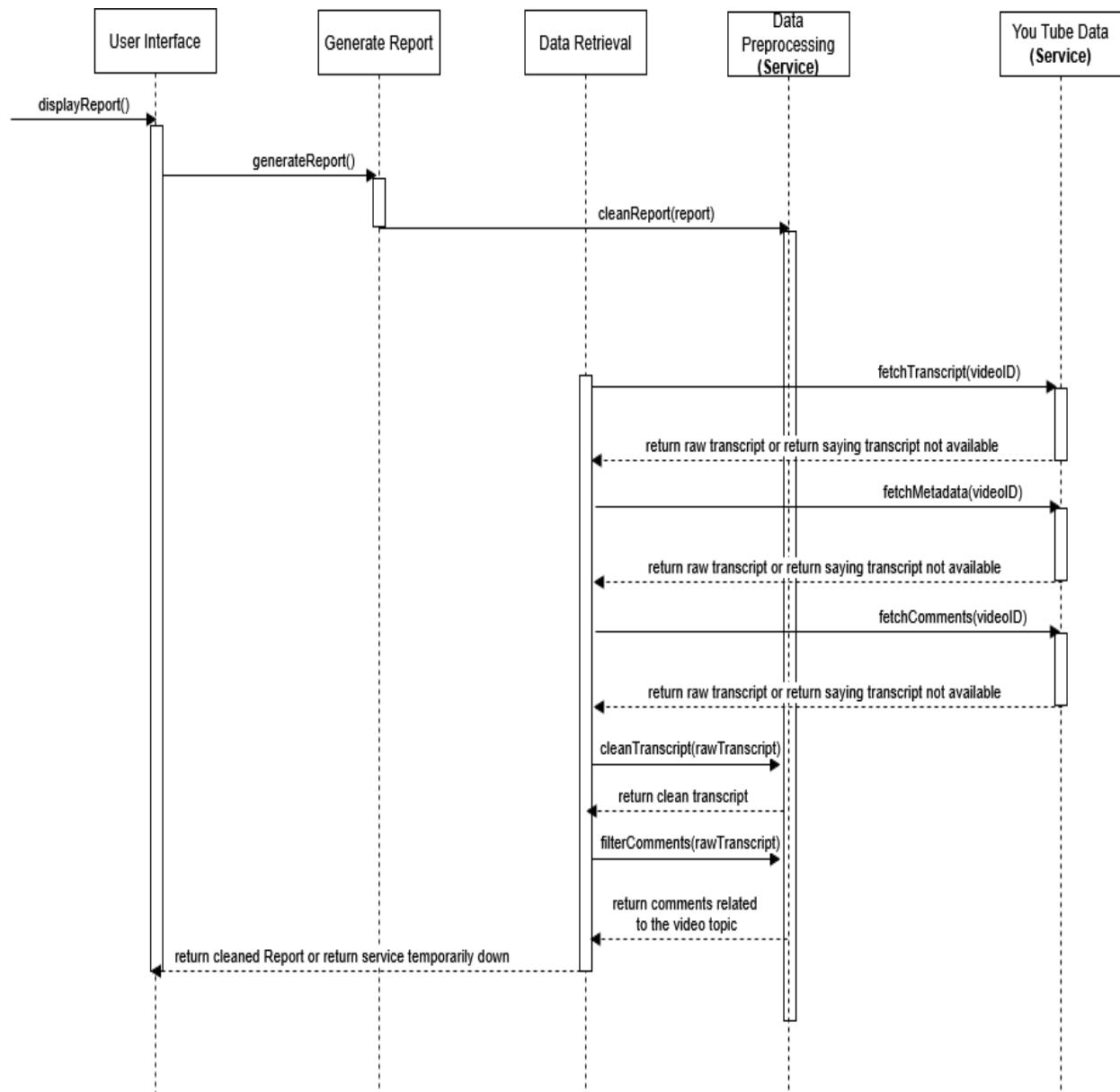
Tools Used:

- ❖ **Software used for this system** – PyCharm
- ❖ **Software used to produce your documentation** – Microsoft Word and PowerPoint
- ❖ **Other Tools** – <https://www.draw.io/>

State diagram



SOA SERVICES



Implementation of Sprint 2:

- Transcript summarization completed
- Sentiment analysis integrated
- Feedback extraction implemented
- Basic UI added for displaying results

Output:

Video Summary:
In this speech addressed to students, the speaker (implied to be Barack Obama) stresses the **critical importance of education and personal responsibility**.
He shares personal stories of his own upbringing and Michelle Obama's, highlighting how they overcame challenges through hard work and education despite difficult circumstances. While acknowledging that students may face hardships (lack of support, financial issues, unsafe environments), he firmly states these are *not excuses* for neglecting schoolwork or giving up.
He emphasizes that education is **essential for future careers** and crucial for **solving the nation's biggest challenges** (health, environment, economy, social justice). He urges students to **set goals, work diligently, embrace failure as a learning opportunity, seek help when needed, and persist** even when things are tough. The core message is that students must take ownership of their education, as their success determines not only their own future but the future of the country.

Sentiment Analysis:
- Positive: 60
- Negative: 9
- Neutral: 27

Feedback Report:
Okay, here are the insights gleaned from the provided comments, categorized for clarity:

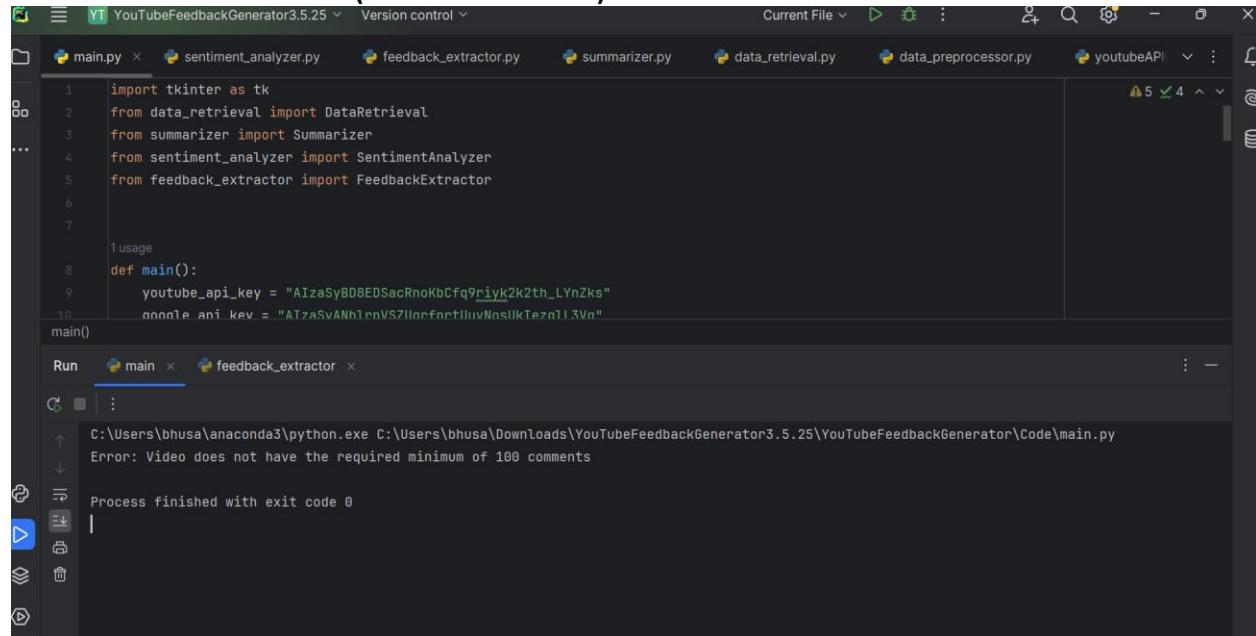
1. Overwhelmingly Positive Reception & Motivational Impact:
* **Strong Theme of Motivation:** The most frequent comment type praises the speech as "motivational," "inspiring," "powerful," and "life-changing." Phrases like "signifies a sense of hope," "amazing speech," and "great impression" reinforce this theme.
* **Action-Oriented Inspiration:** Viewers felt spurred to action or adopted key messages like "Work hard and smart," "Don't run after the problems Work on your solution," and especially "There is no excuse for not trying."
* **Emotional Connection:** Comments like "I find it very motivated 😊 and also i feel very compassionate about it" and the use of heart emojis (❤️) indicate a strong emotional resonance.

2. Identification and Reaction to the Speaker (Presumed Obama):
* **Clear Identification:** Many comments explicitly mention "Obama" or "Mister president."

Fig: Normal Demo of project

Test Cases

Test case 1: if no comments (or comments < 100)



```
import tkinter as tk
from data_retrieval import DataRetrieval
from summarizer import Summarizer
from sentiment_analyzer import SentimentAnalyzer
from feedback_extractor import FeedbackExtractor

usage
def main():
    youtube_api_key = "AIzaSyBD8EDSacRnoKbCfq9riyk2k2th_LYnZks"
    annote_ani_key = "ATzaSvANhIrnVS7llnrfnrtluvNasIkTeza!13Vn"

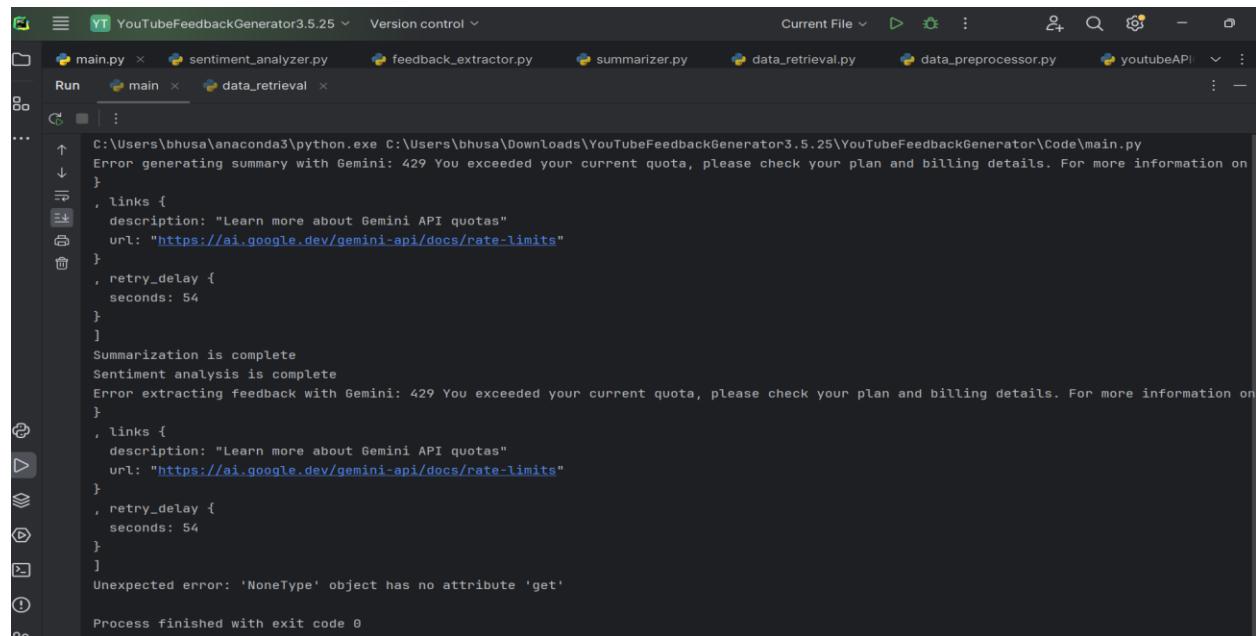
main()
```

Run main

```
C:\Users\bhusa\anaconda3\python.exe C:\Users\bhusa\Downloads\YouTubeFeedbackGenerator3.5.25\YouTubeFeedbackGenerator\Code\main.py
Error: Video does not have the required minimum of 100 comments

Process finished with exit code 0
```

Test case 2: If the API limit exceeded



```
... C:\Users\bhusa\anaconda3\python.exe C:\Users\bhusa\Downloads\YouTubeFeedbackGenerator3.5.25\YouTubeFeedbackGenerator\Code\main.py
Error generating summary with Gemini: 429 You exceeded your current quota, please check your plan and billing details. For more information on this error, see https://ai.google.dev/gemini-api/docs/rate-limits
}
, links {
    description: "Learn more about Gemini API quotas"
    url: "https://ai.google.dev/gemini-api/docs/rate-limits"
}
, retry_delay {
    seconds: 54
}
]
Summarization is complete
Sentiment analysis is complete
Error extracting feedback with Gemini: 429 You exceeded your current quota, please check your plan and billing details. For more information on this error, see https://ai.google.dev/gemini-api/docs/rate-limits
}
, links {
    description: "Learn more about Gemini API quotas"
    url: "https://ai.google.dev/gemini-api/docs/rate-limits"
}
, retry_delay {
    seconds: 54
}
]
Unexpected error: 'NoneType' object has no attribute 'get'

Process finished with exit code 0
```

Test case 3: If API not connected

```
import tkinter as tk
from data_retrieval import DataRetrieval
from summarizer import Summarizer
from sentiment_analyzer import SentimentAnalyzer
from feedback_extractor import FeedbackExtractor

usage
def main():
    youtube_api_key = "IzaSyBD8EDSacRnoKbCfq9riyk2k2th_LYnZks"
    google_api_key = "AIzaSyDd5SC1kluPd7MqyC9ZASDBNjadal5e_C8"
    video_url = "https://www.youtube.com/watch?v=PGUdWfB8nLg"
main()

Run main × data_retrieval ×
C:\Users\bhusa\anaconda3\python.exe C:\Users\bhusa\Downloads\YouTubeFeedbackGenerator3.5.25\YouTubeFeedbackGenerator3.5.25\main.py
Connection error: Metadata API error: 400
Process finished with exit code 0
```

Test case 4: IF no transcript in video

```
Video Summary:
Based on the information provided, I cannot summarize the video content because:
**The transcript for the video ('https://www.youtube.com/watch?v=qkxufkqjxwy') is unavailable in English.**

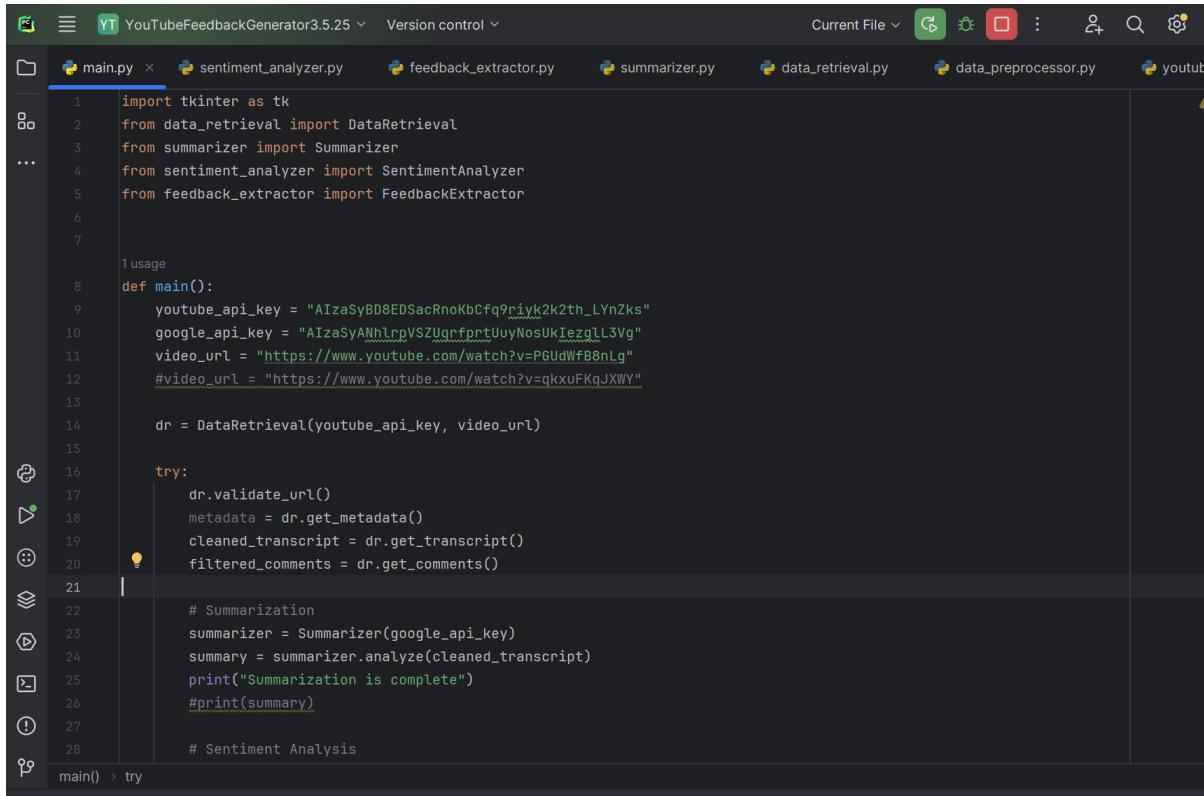
The system failed to retrieve an English transcript, although an auto-generated one might exist in Hindi. Without the transcript text, a summary cannot be generated.

Sentiment Analysis:
- Positive: 97
- Negative: 3
- Neutral: 100

Feedback Report:
These comments reveal overwhelming appreciation for the video, consistently finding it hilarious, legendary, and highly rewatchesble, with many viewers noting they've watched it numerous times and are still viewing it years later (often mentioning 2025/2026). Viewers frequently highlight specific memorable moments and lines (like "Buddy ake father aye h") using timestamps, praising Bassi's unique storytelling, comedic delivery, and ability to connect jokes. While a small minority expresses criticism or doesn't find it funny, the dominant sentiment is strong amusement and admiration for the performance and its enduring appeal.

Process finished with exit code 0
```

Program Code:



The screenshot shows a code editor interface with a dark theme. The file being edited is `main.py`, which is part of a project named `YouTubeFeedbackGenerator3.5.25`. The code itself is as follows:

```
1 import tkinter as tk
2 from data_retrieval import DataRetrieval
3 from summarizer import Summarizer
4 from sentiment_analyzer import SentimentAnalyzer
5 from feedback_extractor import FeedbackExtractor
6
7
8 usage
9 def main():
10     youtube_api_key = "AIzaSyBD8EDSacRnoKbCfq9riyk2k2th_LYnZks"
11     google_api_key = "AIzaSyANh1rpVSZUorfprtUuyN0sUkIezglL3Vg"
12     video_url = "https://www.youtube.com/watch?v=PGUDwfB8nLg"
13     #video_url = "https://www.youtube.com/watch?v=qkxuFKqJXWY"
14
15     dr = DataRetrieval(youtube_api_key, video_url)
16
17     try:
18         dr.validate_url()
19         metadata = dr.get_metadata()
20         cleaned_transcript = dr.get_transcript()
21         filtered_comments = dr.get_comments()
22
23         # Summarization
24         summarizer = Summarizer(google_api_key)
25         summary = summarizer.analyze(cleaned_transcript)
26         print("Summarization is complete")
27         #print(summary)
28
29     # Sentiment Analysis
30
31 main() > try
```

Fig: Main code 1

```
27
28     # Sentiment Analysis
29     sentiment_analyzer = SentimentAnalyzer()
30     sentiment_results = sentiment_analyzer.analyze(filtered_comments)
31     print("Sentiment analysis is complete")
32     #print(sentiment_results)
33
34     # Feedback Extraction
35     feedback_extractor = FeedbackExtractor(google_api_key)
36     feedback_report = feedback_extractor.analyze(filtered_comments)
37     feedback_analysis = feedback_report.get("feedback", "No feedback generated.")
38     print("Feedback analysis is complete")
39     #print(feedback_analysis)
40
41     # Basic Tkinter window
42     root = tk.Tk()
43     root.title("YouTube Analysis")
44
45     # Text Widget for Display
46     text_output = tk.Text(root, wrap="word", height=80, width=80)
47     text_output.pack(padx=10, pady=10)
48
49     # Insert Analysis Results
50     text_output.insert(tk.END, f"Video Summary:\n{summary}\n\n")
51     text_output.insert(tk.END, f"Sentiment Analysis:\n")
52     text_output.insert(tk.END, f"- Positive: {len(sentiment_results['positive'])}\n")
53     text_output.insert(tk.END, f"- Negative: {len(sentiment_results['negative'])}\n")
54     text_output.insert(tk.END, f"- Neutral: {len(sentiment_results['neutral'])}\n\n")
55     text_output.insert(tk.END, f"Feedback Report:\n{feedback_analysis}")
56
57     text_output.config(state="disabled") # Make text read-only
58
59     root.mainloop()
60
61 except ValueError as e:
62     print(f"Error: {e}")
63 except ConnectionError as e:
64     print(f"Connection error: {e}")
65 except Exception as e:
66     print(f"Unexpected error: {e}")
67
68 if __name__ == "__main__":
69     main()
70
71
72 main() : try
```

Fig: Main code 2

```
45     # Text Widget for Display
46     text_output = tk.Text(root, wrap="word", height=80, width=80)
47     text_output.pack(padx=10, pady=10)
48
49     # Insert Analysis Results
50     text_output.insert(tk.END, f"Video Summary:\n{summary}\n\n")
51     text_output.insert(tk.END, f"Sentiment Analysis:\n")
52     text_output.insert(tk.END, f"- Positive: {len(sentiment_results['positive'])}\n")
53     text_output.insert(tk.END, f"- Negative: {len(sentiment_results['negative'])}\n")
54     text_output.insert(tk.END, f"- Neutral: {len(sentiment_results['neutral'])}\n\n")
55     text_output.insert(tk.END, f"Feedback Report:\n{feedback_analysis}")
56
57     text_output.config(state="disabled") # Make text read-only
58
59     root.mainloop()
60
61 except ValueError as e:
62     print(f"Error: {e}")
63 except ConnectionError as e:
64     print(f"Connection error: {e}")
65 except Exception as e:
66     print(f"Unexpected error: {e}")
67
68 if __name__ == "__main__":
69     main()
70
71
72 main() : try
```

Fig: Main code 3

```
1 import google.generativeai as genai
2 from typing import List, Dict, Optional
3 from base_analyzer import BaseAnalyzer
4
5 2 usages
6 class FeedbackExtractor(BaseAnalyzer):
7     """Processes comments using Gemini AI to extract 'What's Working' and 'Needs Improvement' feedback."""
8
9     def __init__(self, api_key: str):
10         super().__init__(api_key)
11         genai.configure(api_key=self.api_key)
12
13     1 usage
14     def analyze(self, comments: List[str]) -> Optional[Dict[str, str]]:
15         """Uses Gemini AI to extract positive and constructive feedback from comments."""
16         try:
17             model = genai.GenerativeModel("gemini-2.5-pro-exp-03-25")
18             # Role-play Prompt for Feedback Extraction
19             prompt = (
20                 f"Get concises insights from these following comments in one paragraph:\n\n{comments}"
21             )
22             response = model.generate_content(prompt)
23
24             return {"feedback": response.text.strip()}
25
26         except Exception as e:
27             print(f"Error extracting feedback with Gemini: {e}")
28             return None
29
FeedbackExtractor > __init__()
```

Fig: Feedback code

```
1 from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
2 from typing import List, Dict
3 from base_analyzer import BaseAnalyzer
4
5 2 usages
6 class SentimentAnalyzer(BaseAnalyzer):
7     """Classifies comments as positive, negative, or neutral using VADER."""
8
9     def __init__(self):
10         super().__init__()
11         self.analyzer = SentimentIntensityAnalyzer()
12
13     1 usage
14     def analyze(self, comments: List[str]) -> Dict[str, List[str]]:
15         """Classifies comments into sentiment categories using VADER."""
16         sentiment_results = {"positive": [], "negative": [], "neutral": []}
17
18         for comment in comments:
19             score = self.analyzer.polarity_scores(comment)['compound']
20             if score >= 0.05:
21                 sentiment_results["positive"].append(comment)
22             elif score <= -0.05:
23                 sentiment_results["negative"].append(comment)
24             else:
25                 sentiment_results["neutral"].append(comment)
26
27         return sentiment_results
```

sentiment analysis code

Fig:

```
1 import google.generativeai as genai
2 from typing import Optional
3 from base_analyzer import BaseAnalyzer
...
5
6 class Summarizer(BaseAnalyzer):
7     """Handles summarization of a video transcript using Google Gemini AI."""
8
9     def __init__(self, api_key: str):
10         super().__init__(api_key)
11         genai.configure(api_key=api_key)
12
13     def analyze(self, cleaned_transcript: str) -> Optional[str]:
14         """Generates a concise video summary using Gemini AI."""
15         try:
16             model = genai.GenerativeModel("gemini-2.5-pro-exp-03-25")
17             response = model.generate_content(
18                 f"Generate a concise summary of the video.\n\n{cleaned_transcript}")
19             return response.text.strip()
20
21         except Exception as e:
22             print(f"Error generating summary with Gemini: {e}")
23             return None
24
```

Fig: summarizer code

Reflection of Sprint 2:

What worked well:

1. Feature Implementation

- ❖ **Transcript Summarization:** Successfully implemented using Gemini AI to generate concise summaries.
- ❖ **Sentiment Analysis:** Integrated VADER for classifying comments into positive, negative, and neutral categories.
- ❖ **Feedback Extraction:** AI-driven extraction of constructive feedback worked as expected.
- ❖ **Basic UI:** A Tkinter-based interface was added to display the analysis results.

2. Code Structure & Modularity

- ❖ The system follows a clear modular design (Summarizer, SentimentAnalyzer, FeedbackExtractor).
- ❖ Separation of concerns makes it easy to maintain and extend.

3. Error Handling & Stability

- ❖ Handled potential issues like invalid URLs, API failures, and connection errors gracefully.

Some of the challenges we faced

I. API Key Security

- ❖ Hardcoded API keys in main.py are a security risk. Moving them to a config file is required for the next sprint.

II. Processing Efficiency

- ❖ Handling large comment datasets might slow down execution. Optimization techniques like multithreading need to be researched.

III. UI Improvements

- ❖ The Tkinter interface is functional but minimal. Adding scrollbars, formatting, or interactive elements could enhance usability.

IV. Time Management

- ❖ The team needs to manage the time slot. This sprint was a slight rush.

Lessons learnt:

The lesson learned is to secure API configurations, optimize processing and UI, and improve time management to enhance project execution.

Impact on Project

These challenges have led to a strategic reallocation of resources and revised timelines to improve security, efficiency, and overall usability.

Invoice (in hours):

Documentation	Ayush – 34, Anthony – 30, Nabin – 32
Research	Ayush – 18, Anthony – 17, Nabin – 16
Design	Ayush – 14, Anthony – 12, Nabin – 15
Code	Ayush – 20, Anthony – 18, Nabin – 16
Testing	Ayush – 8, Anthony – 8, Nabin – 9

Estimated for Remaining Work:

As per Product Backlog: 50 - 20 = 30 days.

5 extra days to complete what didn't work well in Sprint 2.

Total: 35 days

Product Backlog for 3:

ID	Product Backlog	Estimation	Priority
1	As a User, I can enter a YouTube video URL on the home page and click “Search” to initiate video data retrieval (metadata, transcript, and comments).	3	1
18	Extract actionable “What’s Working” feedback from filtered comments using the ChatGPT API.	1	18
19	Extract actionable “Needs Improvement” feedback from filtered comments using the ChatGPT API.	1	19
20	Generate a structured feedback report categorizing “What’s Working” and “Needs Improvement” and store the report.	2	20
21	As a User, I can view a results page that displays the compiled video metadata (e.g., title, description, duration).	3	21
22	As a User, I can view a results page that displays the generated video summary.	2	22
23	As a User, I can view a results page that displays detailed sentiment analysis results of comments.	2	23
24	As a User, I can view a results page that displays the structured feedback report categorizing “What’s Working” and “Needs Improvement”.	3	24

Tools Used:

- ❖ Software used for this system – PyCharm
- ❖ Software used to produce your documentation – Microsoft Word and PowerPoint
- ❖ Other Tools – <https://www.draw.io/>

1. Quality Attributes – Runtime Qualities

Attribute	Bus Rank	Design align	Run time Quality Attribute Documents
Performance			<p>Definition: System performance describes attributes of the system including navigational abilities and connectivity with interfacing systems. May include system response time, number of transactions per minute, and other critical performance characteristics which define this project.</p> <p>Reason for Business Rank <> 3: Performance is essential to deliver timely video summaries and real-time sentiment analysis for a responsive user experience.</p> <p>Explanation of Design Alignment <> Meets: The system employs optimized API calls, efficient data retrieval, and parallel processing to ensure performance targets are met.</p>
Reliability			<p>Definition: Reliability defines the ability of the system to operate correctly over time. Includes consistent stability of the system during available hours.</p> <p>Reason for Business Rank <> 3: High reliability is critical to maintain user trust and ensure continuous availability of video data and analysis.</p> <p>Explanation of Design Alignment <> Meets: Redundant mechanisms, robust error handling, and comprehensive logging are implemented to achieve the required reliability.</p>
Scalability			<p>Definition: Scalability defines the parameters of growth within a given period. The solution must scale so that additional capacity can be added expediently in a technically and economically feasible way.</p> <p>Reason for Business Rank <> 3: Scalability is needed to handle increasing volumes of video and comment data as the user base grows.</p> <p>Explanation of Design Alignment <> Meets: A modular and</p>

			distributed architecture enables the system to scale horizontally as needed.
Security			<p>Definition: Security is the ability of the system to resist unauthorized attempts to access the system and/or its components.</p> <p>Reason for Business Rank <> 3: Security protects sensitive video metadata, transcripts, and user data from unauthorized access.</p> <p>Explanation of Design Alignment <> Meets: The system employs encryption, access controls, and secure API integrations to meet stringent security requirements.</p>
Availability			<p>Definition: Availability defines the days and hours the system is anticipated and/or required to be operational.</p> <p>Reason for Business Rank <> 3: Continuous system availability is crucial to support global users and uninterrupted service.</p> <p>Explanation of Design Alignment <> Meets: The deployment includes redundant systems and failover mechanisms to ensure 24/7 availability.</p>
Supportability			<p>Definition: Supportability defines the requirement that problems can be diagnosed and addressed quickly, with appropriately trained resources.</p> <p>Reason for Business Rank <> 3: Quick diagnosis and resolution of issues minimize downtime and maintain service quality.</p> <p>Explanation of Design Alignment <> Meets: Integrated logging, monitoring tools, and automated alerts are in place to support rapid troubleshooting.</p>
Traceability			<p>Definition: Traceability defines the ability to trace the history, application, or location of an item or activity by means of recorded identification.</p> <p>Reason for Business Rank <> 3: It is important for audit trails and debugging to have full traceability of data and changes.</p> <p>Explanation of Design Alignment <> Meets: Detailed logging of data transformations (e.g., transcript cleaning, comment</p>

			filtering) ensures complete traceability throughout the processing pipeline.
User Usability			<p>Definition: User Usability defines how easy the interface is to use to successfully complete tasks, including clear navigation, terminology, and error messaging.</p> <p>Reason for Business Rank <> 3: A user-friendly interface directly impacts user satisfaction and adoption.</p> <p>Explanation of Design Alignment <> Meets: The design features an intuitive dashboard with clear visualizations for video summaries, sentiment analysis, and structured feedback.</p>

2. Quality Attributes- Development Qualities

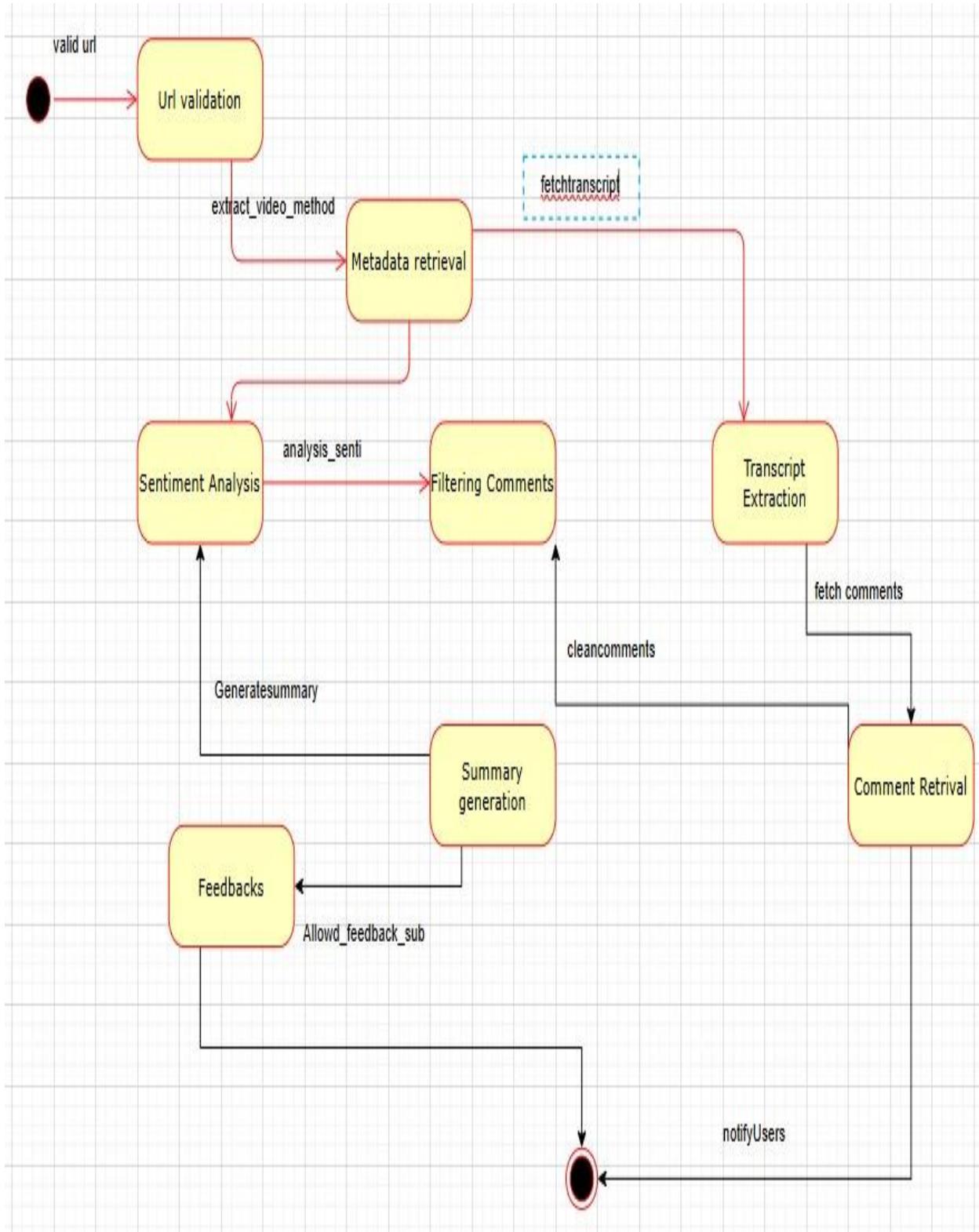
Attributes	Bus Rank	Design Align	Run Time Quality Attributes Documents
Reusability			<p>Definition: The reusability of the system is the ability to reuse portions of the system in other applications.</p> <p>Reason for Business Rank <> 3: High reusability reduces development time and cost by allowing components to be leveraged in future projects.</p> <p>Explanation of Design Alignment <> Meets: The system is designed with modular components (data retrieval, transcript cleaning, sentiment analysis) that can be reused across different applications.</p>
Extensibility			<p>Definition: Extensibility defines the degree to which the system can be extended to incorporate new functionality without changing the fundamental architecture, design, or core source code.</p> <p>Reason for Business Rank <> 3: Extensibility is essential to accommodate evolving business needs and integration with additional data sources.</p> <p>Explanation of Design Alignment <> Meets: Published interfaces such as APIs and metadata definitions allow for seamless extension of system capabilities.</p>

Modifiability		<p>Definition: Modifiability (maintainability) defines the efficiency of making enhancements or maintenance changes.</p> <p>Reason for Business Rank <> 3: High modifiability minimizes disruption and cost when updating the system.</p> <p>Explanation of Design Alignment <> Meets: A well-structured, documented, and modular codebase allows for quick modifications without affecting core functionality.</p>
Ease of Integration		<p>Definition: Ease of Integration defines how easily the application or its components can be integrated or packaged with new applications.</p> <p>Reason for Business Rank <> 3: Seamless integration is key to interoperability with external systems and tools.</p> <p>Explanation of Design Alignment <> Meets: The system uses standardized, RESTful APIs and common data formats to facilitate integration.</p>
Phase-In “Ability”		<p>Definition: Phase In “ability” (subset-ability) defines how the application accommodates implementation/rollout schedules without impacting overall design.</p> <p>Reason for Business Rank <> 3: Phased deployment minimizes risk and allows for early user feedback.</p> <p>Explanation of Design Alignment <> Meets: The architecture supports incremental deployment, enabling features to be released in stages.</p>
Conceptual Integrity		<p>Definition: Conceptual Integrity refers to the ability of the architecture to communicate a clear, concise vision in line with the enterprise architecture.</p> <p>Reason for Business Rank <> 3: Ensures a unified approach and consistency across the system, which is vital for long-term maintainability.</p> <p>Explanation of Design Alignment <> Meets: Consistent design principles and a unified processing pipeline ensure all components work cohesively.</p>

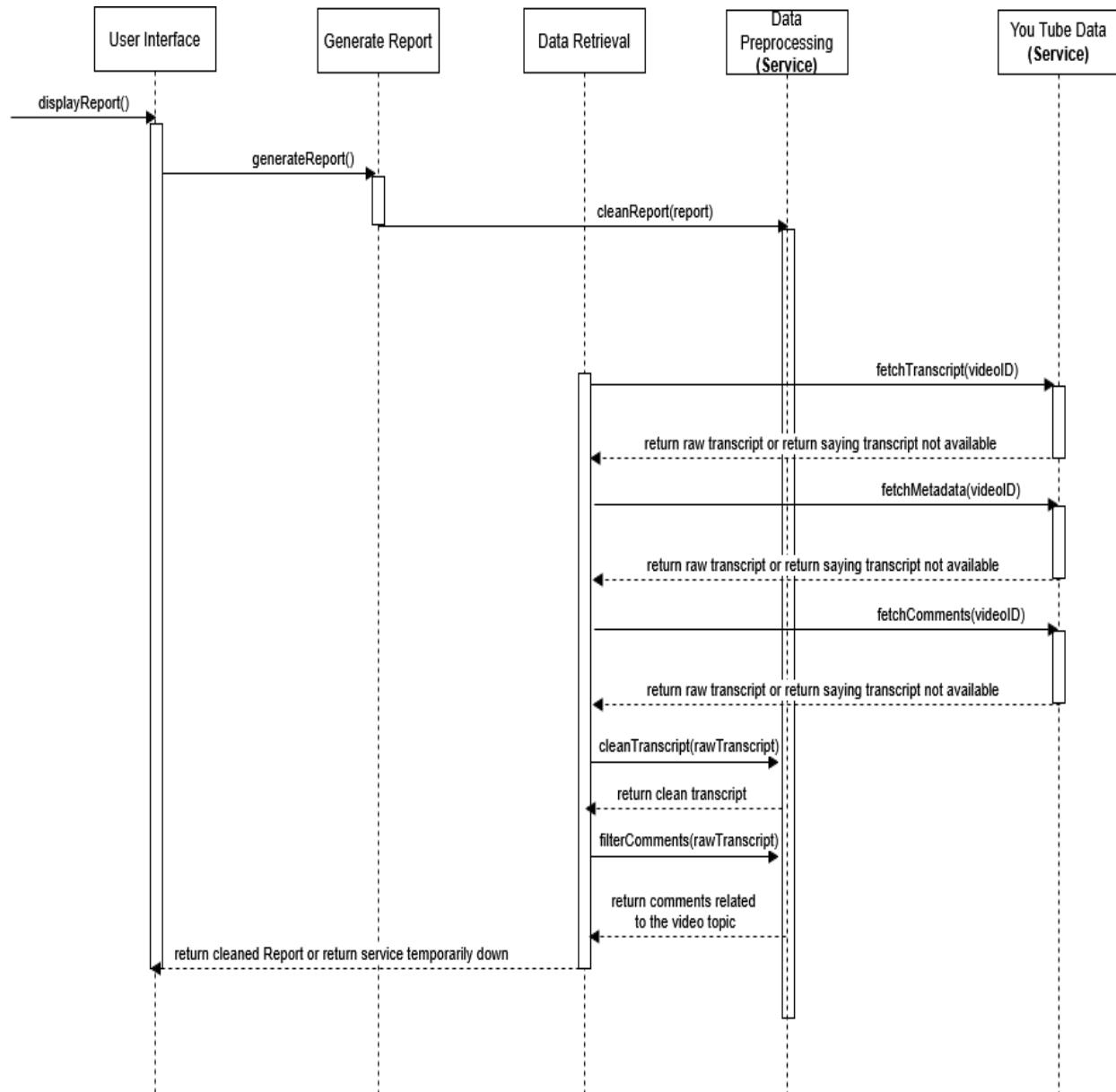
Testability			<p>Definition: Testability describes the ability to test each component independently (unit tests) as well as in an integrated environment.</p> <p>Reason for Business Rank <> 3: High testability reduces risk of defects and ensures reliable system performance with each update.</p> <p>Explanation of Design Alignment <> Meets: A comprehensive suite of automated tests is integrated into the CI/CD pipeline to verify all critical components.</p>
Portability			<p>Definition: Portability measures the ease with which the system can be moved to different platforms (hardware, operating systems, virtualization, etc.).</p> <p>Reason for Business Rank <> 3: Portability provides flexibility in deployment and minimizes vendor lock-in.</p> <p>Explanation of Design Alignment <> Meets: The system is developed using cross-platform frameworks and containerization, ensuring compatibility across environments.</p>
Developer Complexity			<p>Definition: Developer Complexity is how easy it is for developers to understand and create the system.</p> <p>Reason for Business Rank <> 3: Lower complexity accelerates onboarding and reduces development errors.</p> <p>Explanation of Design Alignment <> Meets: The architecture is simplified with clear module boundaries and detailed documentation, reducing the overall complexity.</p>
Buildability			<p>Definition: Buildability defines the degree to which the system can be compiled, packaged, archived, and deployed using automated tools and standard processes.</p> <p>Reason for Business Rank <> 3: Efficient build processes accelerate time-to-market and ensure consistency.</p> <p>Explanation of Design Alignment <> Meets: The integration of an automated CI/CD pipeline and build automation tools streamlines deployment and minimizes errors.</p>

SPRINT 3

State diagram



SOA SERVICES



ID	Product Backlog	Estimation	Priority
1	As a User, I can enter a YouTube video URL on the home page and click "Search" to initiate video data retrieval (metadata, transcript, and comments).	3	1
18	Extract actionable "What's Working" feedback from filtered comments using the ChatGPT API.	1	18
19	Extract actionable "Needs Improvement" feedback from filtered comments using the ChatGPT API.	1	19
20	Generate a structured feedback report categorizing "What's Working" and "Needs Improvement" and store the report.	2	20
21	As a User, I can view a results page that displays the compiled video metadata (e.g., title, description, duration).	3	21
22	As a User, I can view a results page that displays the generated video summary.	2	22
23	As a User, I can view a results page that displays detailed sentiment analysis results of comments.	2	23
24	As a User, I can view a results page that displays the structured feedback report categorizing "What's Working" and "Needs Improvement".	3	24

Implementation of Sprint 3:

- Integrated Tkinter UI with HomePage (ID 1) for input and ResultsPage to display analysis (ID 21, 22, 23, 24)
- Applied prompt engineering to improve structure and clarity of AI-generated outputs (ID 18, 19)
- Auto-displayed video summary, sentiment %, and bullet-point feedback in scrollable UI
- Implemented background threading with loading popup for smooth UX
- Saved feedback as timestamped .txt report in /reports folder (ID 20)

Retrospective: What went well

- Prompt engineering improved accuracy and structure of AI outputs
- Gemini-based bullet-point feedback displayed clearly with bolded subtopics
- UI handled empty inputs, bad URLs, and API errors without crashing
- Background thread + animated loading popup kept UI responsive
- Feedback report saved correctly in local folder

Retrospective: Challenges

- Regex spam filtering still misses nuanced promotional comments (data_preprocessor.py)
- API keys remain hardcoded in main.py (Sprint 2 issue not resolved)
- Sentiment classification slows down at ~3000 comments (batching not implemented)
- Bold text workaround required due to tk.Label formatting limitations

Output:

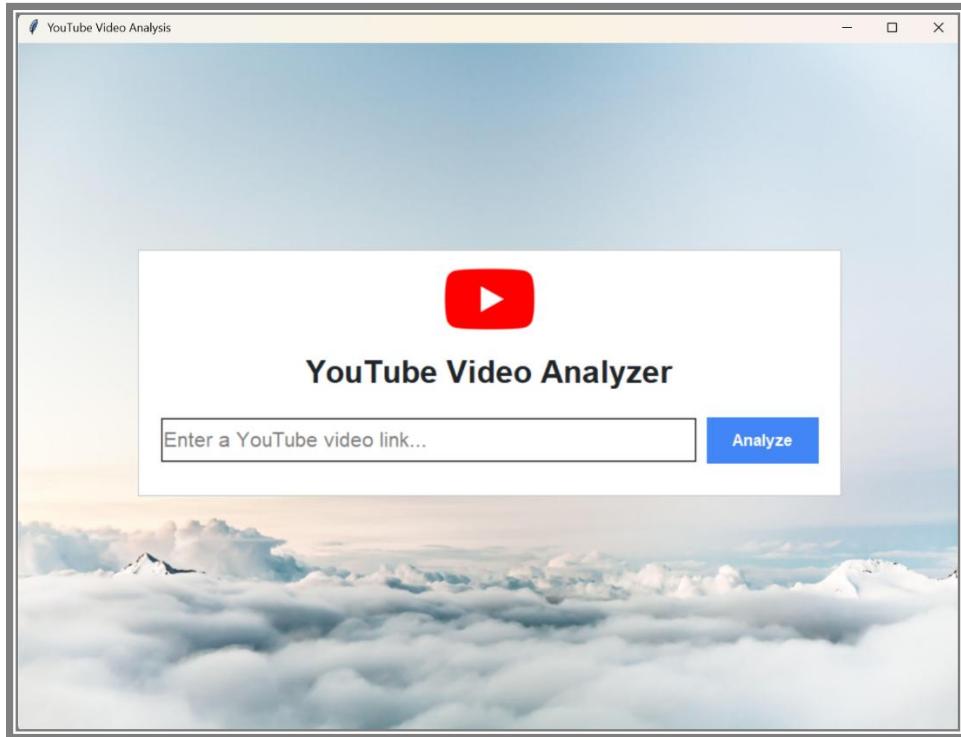


Fig: Normal Demo of project

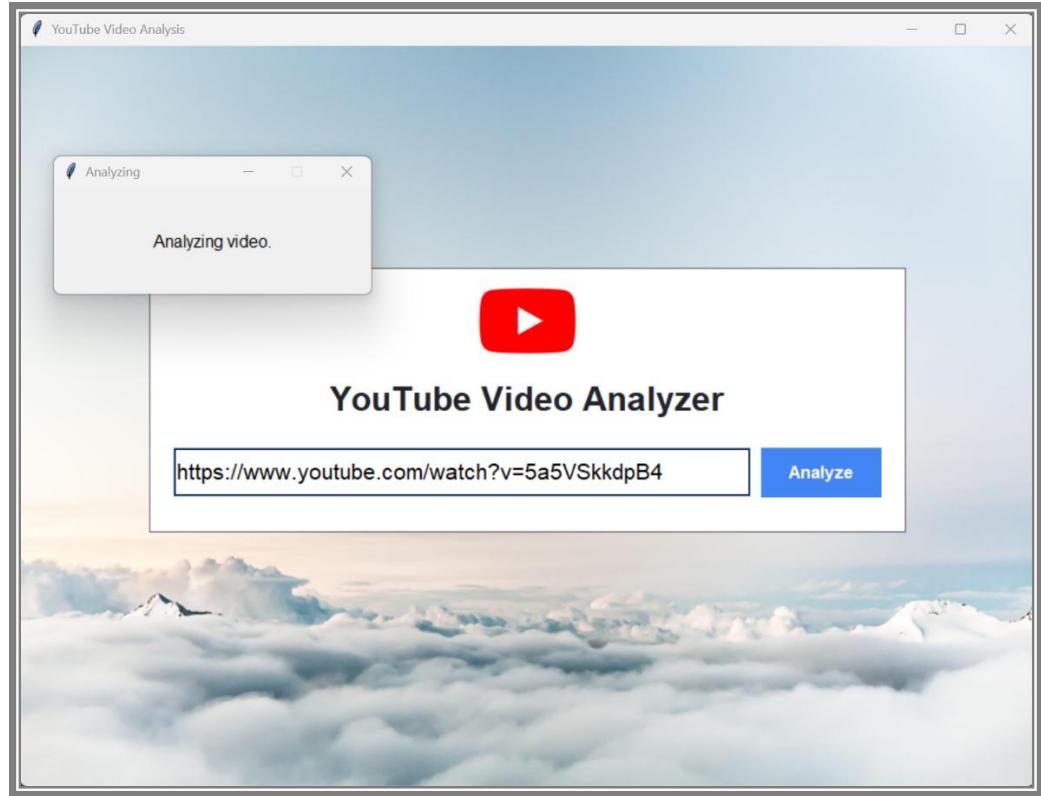


Fig: Home page when analyzing

A screenshot of the "Video Analysis Results" page. It features a dark header bar with a "Return to Home" button and a "Video Analysis Results" title. The main content area is divided into three sections: "Video Summary", "Sentiment Analysis", and "What's Working".

- Video Summary:** A note stating "Okay, I'm ready to summarize!" followed by a placeholder message: "However, the transcript you intended to provide seems to be missing. The input says "transcript unavailable: no element found: line 1, column 0". Please provide the actual text of the YouTube video transcript. Once you do, I will be happy to read it and produce a concise, insightful summary focusing on key points, main topics, and important takeaways, as requested."
- Sentiment Analysis:** Displays the distribution of sentiment: Positive: 57.9%, Negative: 9.5%, Neutral: 32.6%.
- What's Working:** A list of positive findings:
 - Motivational Impact: The speech was highly praised for its ability to inspire and encourage viewers, fostering a positive and proactive mindset towards achieving personal goals.
 - Emphasis on Education: Audiences appreciated the strong focus on the significance of education as a fundamental tool for personal development and future success.
 - Advocacy for Personal Responsibility: The core message promoting self-accountability, diligent effort, and perseverance in overcoming obstacles deeply resonated with viewers for its empowering nature.
 - Speaker's Eloquence: The articulate and engaging delivery of the speech was frequently commended, contributing significantly to the message's effectiveness and memorability.
 - Profound Life Lessons: Viewers found the speech rich in meaningful wisdom and practical advice, offering valuable guidance that extended beyond academics into personal character.
 - Authentic Connection: The speaker's perceived sincerity and the relatable nature of his message, sometimes reflecting his own journey, fostered a sense of genuine connection with the audience.
 - Youth Empowerment: The specific encouragement directed towards students and young individuals was recognized as particularly impactful for instilling confidence and a sense of purpose in them.

Fig: Results page part one

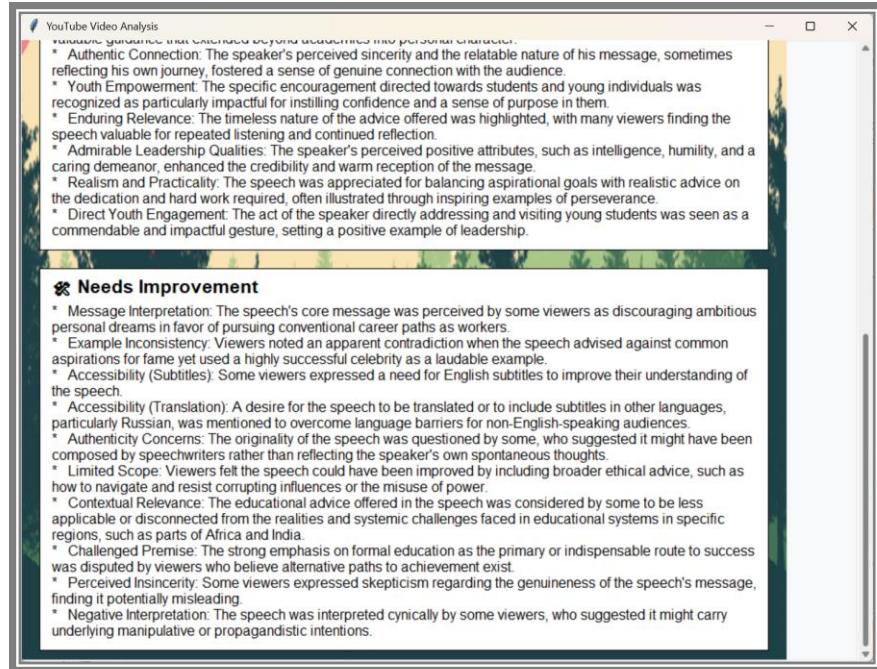
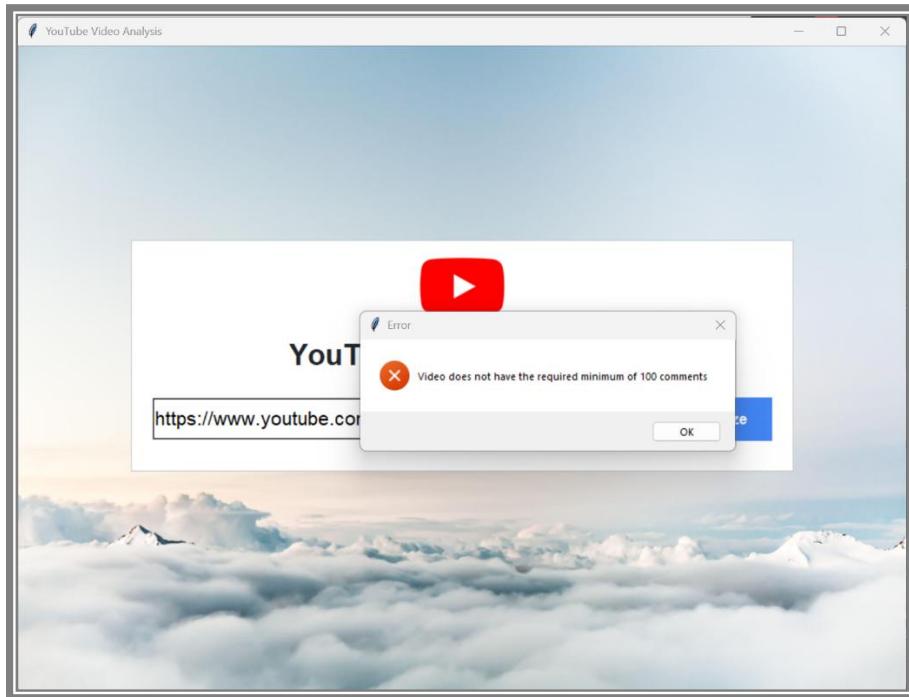


Fig: Results page part two

Test Cases

Test case 1: If comments < 100



Test case 2: When there is no transcript

The screenshot shows the "YouTube Video Analysis" application window titled "Video Analysis Results". The interface is divided into several sections:

- Video Summary:** A message states: "I understand you'd like me to summarize the YouTube video from the link you provided. Unfortunately, as the message indicates, I'm unable to retrieve the transcript for this specific video because 'subtitles are disabled for this video.' Without the transcript (the text content of the video), I cannot process the information to create a summary." It also suggests providing the transcript text directly.
- Sentiment Analysis:** Shows Positive: 58.7%, Negative: 4.9%, Neutral: 36.3%.
- What's Working:** No feedback available.
- Needs Improvement:** No improvement suggestions.

Test case 3: When no transcript but comments part one

The screenshot shows the 'YouTube Video Analysis' software interface. At the top, there's a navigation bar with a logo and the text 'YouTube Video Analysis'. Below it is a 'Return to Home' button. The main area is titled 'Video Analysis Results' with a video thumbnail on the right.

❖ Video Summary

Okay, I understand the task. You want me to summarize the YouTube video from the provided link.
However, the system message indicates: "transcript unavailable: could not retrieve a transcript for the video https://www.youtube.com/watch?v=vcbwtejuuy! this is most likely caused by: subtitles are disabled for this video."
Since the transcript is unavailable (likely because subtitles are disabled), I cannot access the video's content to create a summary.
If you can provide the transcript text directly, or if subtitles become available for the video, I would be happy to help you summarize it.

❖ Sentiment Analysis

Positive: 58.7% Negative: 4.9% Neutral: 36.3%

☑ What's Working

- * Timeless Appeal: The song is consistently praised for its enduring quality, often described as a masterpiece that evokes strong nostalgia and remains enjoyable across different generations.
- * Shah Rukh Khan's Dancing: Viewers frequently lauded Shah Rukh Khan's dance skills, noting the perceived perfection, remarkable flexibility, and an effortless grace in his performance.
- * Juhi Chawla's Screen Presence: Juhi Chawla was positively recognized for her stunning appearance and engaging performance, significantly enhancing the song's visual appeal.
- * Vocal Artistry: The singing, especially by Sonu Nigam and Alka Yagnik, received high acclaim for its mesmerizing quality and impactful contribution to the overall song.
- * Comedic Performances: The contributions of actors like Johnny Lever and Satish Shah were specifically enjoyed for adding a distinct layer of humor and light-hearted entertainment.
- * Choreography Quality: The dance sequences were appreciated for their excellent quality and engaging nature.

Test case 4: When no transcript but comments part two

The screenshot shows the same 'YouTube Video Analysis' software interface as the previous one, but with a scroll-down view of the 'What's Working' section.

☑ What's Working

- * Comedic Chemistry: The interactions of actors like Johnny Lever and Satish Shah were specifically enjoyed for adding a distinct layer of humor and light-hearted entertainment.
- * Choreography Quality: The dance sequences were appreciated for their excellent quality and engaging nature, with some viewers highlighting the complexity and skill involved in the movements.
- * Song's Musicality: The overall music, including its distinctive beats and composition, was found to be excellent and was often cited as a significant stress reliever.
- * Lead Pair Chemistry: The on-screen dynamic between Shah Rukh Khan and Juhi Chawla was widely admired, with many viewers considering them an ideal and highly enjoyable pairing.
- * Lyrical Depth: The song's lyrics were commended for their profound meaning and, for some, the valued use of pure Hindi, adding to its artistic merit.
- * Overall Energetic Vibe: The song was celebrated for creating an amazing and infectious energy that viewers found uplifting and capable of making them feel alive.
- * Dancer Costuming: Specific positive attention was given to the costumes, particularly those of the male dancers, which were noted for being awesome and well-designed.
- * Lasting Impression: The song's ability to stay in viewers' minds and spontaneously trigger fond memories was a significant aspect of its positive reception.
- * Ensemble Contribution: Audiences valued the harmonious collaboration of talents, including singers, actors, and the lyricist, viewing it as a key element of the song's success.

☒ Needs Improvement

- * Crediting - Choreographer: Viewers pointed out the absence of the choreographer's name, indicating a desire for proper recognition of their work in the video's details.
- * Song Lyrics - Repetition: Some viewers felt that the song's lyrical content, excluding the rap segment, was overly repetitive.
- * Crediting - Singer in Title: It was mentioned that the singer's name was often omitted from video titles for their songs, implying this video might also lack this attribution.
- * Costuming - Female Lead: Concerns were raised regarding the female lead's attire, with some viewers finding it unconvincing or ill-considered.
- * Choreography - Female Lead: Some viewers expressed dissatisfaction with the dance steps performed by the female lead, suggesting they lacked impact or quality.
- * Recognition - Music Composers: An observation was made that the music composers were not receiving due acknowledgement for their contribution to the song.
- * Technical Issue - Offline Playback: A viewer reported a functional problem where the video, when downloaded for offline use, only played audio and not the visual component.
- * Vocal Style - Enhancement Suggestion: A suggestion was made that the song could have been improved if the male vocalist had incorporated more Western-style vocal flourishes.

Test case 5: when it has foreign transcript

The screenshot shows a software window titled "YouTube Video Analysis". The main title bar says "Video Analysis Results". A "Return to Home" button is visible in the top left corner.

❖ Video Summary

Unfortunately, I was unable to retrieve an English transcript for the YouTube video you provided (<https://www.youtube.com/watch?v=-rzp19fpumc>).

The system indicates that no direct English transcript (neither manually created nor auto-generated) is available for this video. It did find an auto-generated transcript in Hindi, and English is listed as a language into which an existing transcript could be translated.

Since I need an English transcript to work from, I cannot provide a summary as requested.

You might want to check if English captions are available directly on YouTube for this video, or consider if a summary based on a translation from Hindi would be an option (though this would require an additional step not directly supported by this current process).

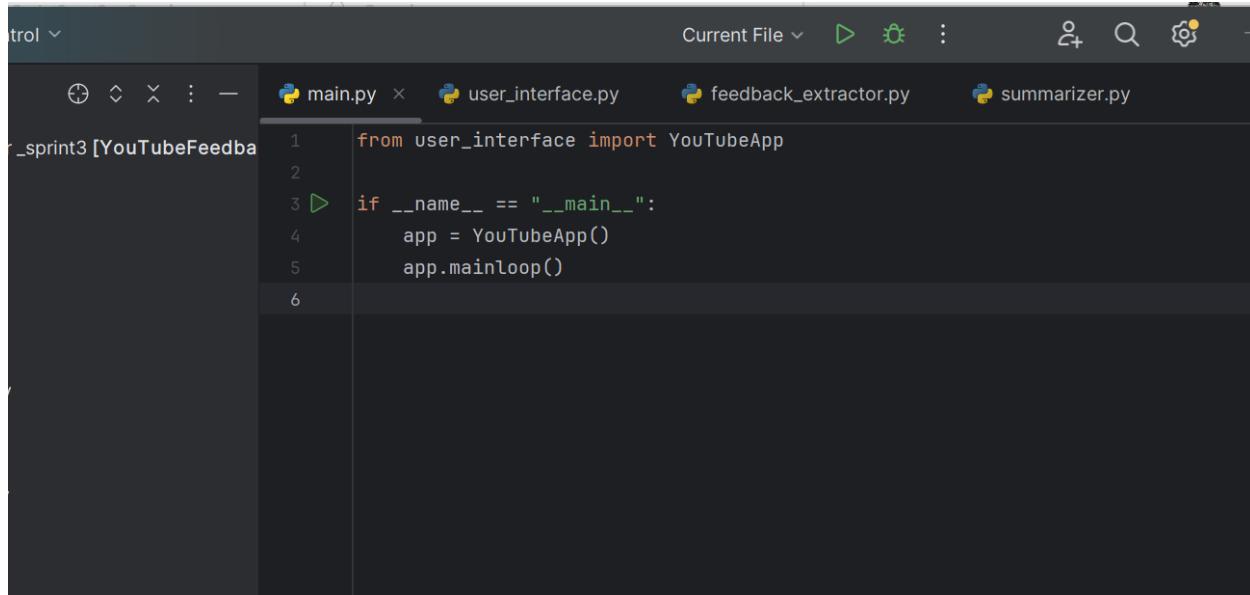
☒ Sentiment Analysis

Positive: 63.7% Negative: 9.8% Neutral: 26.5%

☑ What's Working

- * Nostalgic Resonance: The song effectively evoked strong feelings of nostalgia, prompting viewers to reflect on the passage of time, their youth, and past friendships.
- * Emotional Depth: Audiences found the song deeply moving, with many expressing that it touched their hearts profoundly and even brought them to tears due to its heartfelt nature.
- * Lyrical Craftsmanship: The lyrics were frequently praised for their relatability and poignant exploration of themes like aging and cherished memories, resonating deeply with listeners.
- * Musical Arrangement: The composition, often described as melodious and organic, was appreciated for its pleasing sound and, for some, its successful blend of contemporary and traditional influences.
- * Vocal Performance: The singer's delivery was highlighted for its emotional suitability to the song's themes, with

Program Code:



The screenshot shows a code editor interface with a dark theme. At the top, there is a toolbar with various icons. Below the toolbar, the title bar displays the current file as "main.py" and other files like "user_interface.py", "feedback_extractor.py", and "summarizer.py". The main area of the editor shows the code for "main.py". The code is as follows:

```
from user_interface import YouTubeApp
if __name__ == "__main__":
    app = YouTubeApp()
    app.mainloop()
```

Fig: main.py

The screenshot shows a code editor interface with a dark theme. The top bar includes tabs for 'n.py' (selected), 'Version control', 'Current File', and other icons. Below the tabs, there are four tabs: 'main.py', 'user_interface.py' (selected), 'feedback_extractor.py', and 'summarizer.py'. The left sidebar lists project files: 'base_analyzer.py', 'data_preprocessor.py', 'data_retrieval.py', 'feedback_extractor.py', 'main.py', 'requirements.txt', 'sentiment_analyzer.py', 'summarizer.py', 'user_interface.py', and 'youtubeAPICon.py'. The right sidebar shows 'Local Libraries' and 'Lines and Consoles'. The main code area displays the following Python code:

```
2 from tkinter import messagebox
3 from data_retrieval import DataRetrieval
4 from summarizer import Summarizer
5 from sentiment_analyzer import SentimentAnalyzer
6 from feedback_extractor import FeedbackExtractor
7 from PIL import Image, ImageTk
8 import threading
9 import re
10 import os
11 from datetime import datetime
12
13
14 def save_feedback_as_txt(what_works, needs_improvement, output_dir="reports"):
15     os.makedirs(output_dir, exist_ok=True)
16     filename = f"feedback_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt"
17     filepath = os.path.join(output_dir, filename)
18
19     with open(filepath, "w", encoding="utf-8") as f:
20         f.write("YouTube Feedback Report\n")
21         f.write("=====\\n\\n")
22         f.write("✓ What's Working:\\n")
23         f.write(what_works.strip() + "\\n\\n")
24         f.write("✗ Needs Improvement:\\n")
25         f.write(needs_improvement.strip() + "\\n")
26
27     print(f"✓ Text report saved at: {filepath}")
28
29
```

Fig: user_interface.py part 1

```
 2 usages
30     class YouTubeApp(tk.Tk):
31         def __init__(self):
32             super().__init__()
33             self.title("YouTube Video Analysis")
34             self.geometry("960x700")
35             self.configure(bg="#f8f9fa")
36
37             self.frames = []
38             container = tk.Frame(self)
39             container.pack(fill="both", expand=True)
40             container.grid_rowconfigure(index: 0, weight=1)
41             container.grid_columnconfigure(index: 0, weight=1)
42
43             for F in (HomePage, ResultsPage):
44                 frame = F(container, self)
45                 self.frames[F] = frame
46                 frame.grid(row=0, column=0, sticky="nsew")
47
48             self.show_frame(HomePage)
49
50             self.video_summary = ""
51             self.sentiment_data = {}
52             self.feedback_data = {}
53
54             3 usages (1 dynamic)
55             def show_frame(self, page):
56                 self.frames[page].tkraise()
```

Fig: user_interface.py part 2

A screenshot of a code editor showing the file `user_interface.py`. The code defines a class `HomePage` that inherits from `tk.Frame`. It has an `__init__` method that initializes the frame with a light gray background and sets up a grid configuration. It then tries to open a background image from the assets folder, creates a photo image, and places it in a label that spans the entire frame.

```
def __init__(self, parent, controller):
    super().__init__(parent, bg="#f0f0f0")
    self.controller = controller
    self.grid_rowconfigure( index: 0, weight=1)
    self.grid_columnconfigure( index: 0, weight=1)

    try:
        bg_image = Image.open("assets/background.jpg").resize((960, 700))
        self.bg_photo = ImageTk.PhotoImage(bg_image)
        bg_label = tk.Label(self, image=self.bg_photo)
        bg_label.place(relwidth=1, relheight=1)
```

Fig: user_interface.py part 3

```
1 usage (1 dynamic)
def load_video_data(self, video_url):
    youtube_api_key = "AIzaSyBD8EDSacRnoKbCfq9riyk2k2th_LYnZks"
    gemini_api_key = "AIzaSyANhlrpVSZUgrfpptUuyNosUkIezglL3Vg"
    dr = DataRetrieval(youtube_api_key, video_url)
    dr.validate_url()
    dr.get_metadata()
    transcript = dr.get_transcript()
    comments = dr.get_comments()

    self.video_summary = Summarizer(gemini_api_key).analyze(transcript)
    self.sentiment_data = SentimentAnalyzer().analyze(comments)

    feedback = FeedbackExtractor(gemini_api_key).analyze(comments)
    if feedback is None:
        feedback = {
            "what_works": "No feedback available.",
            "needs_improvement": "No improvement suggestions."
        }
    self.feedback_data = feedback

    save_feedback_as_txt(
        self.feedback_data["what_works"],
        self.feedback_data["needs_improvement"]
    )
```

Fig: user_interface.py load_video_data function

```
TubeFeedback 99
Code 100
assets 101
reports 102
base_analy 103
data_prep 104
data_retriev 105
feedback_e 106
main.py 107
requiremen 108
sentiment_ 109
summarize 110
user_interf 111
youtubeAP 112
ernal Libraries 113
atches and Co 114
115
116
117
118
119
120
121
122
123
124
125
126
127
YouTubeApp > load_video_data()

try:
    bg_image = Image.open("assets/background.jpg").resize((960, 700))
    self.bg_photo = ImageTk.PhotoImage(bg_image)
    bg_label = tk.Label(self, image=self.bg_photo)
    bg_label.place(relwidth=1, relheight=1)
except:
    pass

content_frame = tk.Frame(self, bg="#ffffff", bd=2, highlightbackground="#cccccc", highlightthickness=1)
content_frame.place(relx=0.5, rely=0.3, anchor="n")

try:
    logo_img = Image.open("assets/youtube_logo.png").resize((100, 70))
    self.logo = ImageTk.PhotoImage(logo_img)
    tk.Label(content_frame, image=self.logo, bg="#ffffff").pack(pady=(10, 10))
except:
    tk.Label(content_frame, text="YouTube Video Analyzer", font=("Helvetica", 36), bg="#ffffff").pack(pady=(10, 10))

tk.Label(content_frame, text="Enter a YouTube video link...", font=("Arial", 16), width=45, fg='gray', relief="solid", bd=1,
        highlightthickness=1, highlightcolor="#4285F4", highlightbackground="#ddddd")
self.url_entry.insert(index=0, string="Enter a YouTube video link...")
self.url_entry.bind("<FocusIn>", self.clear_placeholder)
self.url_entry.pack(side="left", ipady=8, padx=(0, 10), expand=True, fill="x")
```

Fig: user_interface.py part 4

```
—   main.py      user_interface.py  x  feedback_extractor.py    summarizer.py
dbac 129         search_button = tk.Button(entry_frame, text="Analyze", font=("Arial", 13, "bold"),
130             command=self.on_search, bg="#4285F4", fg="white", padx=20, pady=8, relief="flat",
131             cursor="hand2")
s 132         search_button.pack(side="left")
analy 133
repr 134
etrie 134     1 usage
ck_e 135         def clear_placeholder(self, event):
y 136             if self.url_entry.get() == "Enter a YouTube video link...":
137                 self.url_entry.delete( first: 0, tk.END)
emen 138                 self.url_entry.config(fg="black")
ent_ 139
arize 139     1 usage
interfa 140         def on_search(self):
peAP 141             video_url = self.url_entry.get().strip()
aries 142             if not video_url or "youtube" not in video_url:
143                 messagebox.showerror( title: "Error", message: "Please enter a valid YouTube video URL.")
d Cc 143             return
144
145             self.loading_popup = tk.Toplevel(self)
146             self.loading_popup.title("Analyzing")
147             self.loading_popup.geometry("300x100")
148             self.loading_popup.resizable( width: False, height: False)
149             self.loading_popup.grab_set()
150
151             self.status_label = tk.Label(self.loading_popup, text="Initializing...", font=("Arial", 12))
152             self.status_label.pack(expand=True, pady=10)
153
154             self.dots = 0
155             self.animate_loading()
YouTubeApp > load_video_data()
```

Fig: user_interface.py part 5

The screenshot shows a code editor window with several tabs at the top: main.py, user_interface.py (which is the active tab), feedback_extractor.py, and summarizer.py. The code in user_interface.py is displayed, showing parts of the application's logic for fetching video metadata and displaying results.

```
157     2 usages
158     def animate_loading(self):
159         if hasattr(self, 'loading_popup') and self.loading_popup.winfo_exists():
160             self.dots = (self.dots + 1) % 4
161             self.status_label.config(text=f"Analyzing video{'.' * self.dots}")
162             self.after(ms=500, self.animate_loading)
163
164     1 usage
165     def run_analysis(self, video_url):
166         try:
167             self.status_label.config(text="Fetching metadata...")
168             self.controller.load_video_data(video_url)
169             if hasattr(self, 'loading_popup'):
170                 self.loading_popup.destroy()
171             self.controller.show_results_page()
172         except Exception as e:
173             if hasattr(self, 'loading_popup'):
174                 self.loading_popup.destroy()
175             messagebox.showerror(title="Error", str(e))
176
177     3 usages
178     class ResultsPage(tk.Frame):
179         def __init__(self, parent, controller):
180             super().__init__(parent)
181             self.controller = controller
182             canvas = tk.Canvas(self, borderwidth=0, background="#f8f9fa")
```

Fig: user_interface.py part 6

```
pac          3 usages
177      class ResultsPage(tk.Frame):
178          def __init__(self, parent, controller):
179              super().__init__(parent)
180              self.controller = controller
181
182              canvas = tk.Canvas(self, borderwidth=0, background="#f8f9fa")
183              self.scroll_frame = tk.Frame(canvas, bg="#f8f9fa")
184              scrollbar = tk.Scrollbar(self, orient="vertical", command=canvas.yview)
185              canvas.configure(yscrollcommand=scrollbar.set)
186
187              scrollbar.pack(side="right", fill="y")
188              canvas.pack(side="left", fill="both", expand=True)
189              canvas.create_window((0, 0), window=self.scroll_frame, anchor="nw")
190              self.scroll_frame.bind("<Configure>", lambda e: canvas.configure(scrollregion=canvas.bbox("all")))
191
192          # === Background Image ===
193          try:
194              bg_image = Image.open("assets/background2.jpg").resize((960, 1600))
195              self.bg_photo = ImageTk.PhotoImage(bg_image)
196              bg_label = tk.Label(self.scroll_frame, image=self.bg_photo)
197              bg_label.place(x=0, y=0, relwidth=1, relheight=1)
198          except:
199              self.scroll_frame.configure(bg="#e0e0e0")
200
201          # === Header ===
202          header = tk.Frame(self.scroll_frame, bg="#f8f9fa")
203          header.pack(fill="x", padx=10, pady=(10, 5))
204
```

YouTubeApp > load_video_data()

Fig: user_interface.py part 7

```
 201     # === Header ===
 202     header = tk.Frame(self.scroll_frame, bg="#f8f9fa")
 203     header.pack(fill="x", padx=10, pady=(10, 5))
 204
 205     tk.Button(header, text="← Return to Home", font=("Arial", 11, "bold"),
 206               command=lambda: controller.show_frame(controller.frames.keys().__iter__().__next__()),
 207               bg="#343a40", fg="white", padx=10, pady=5, relief="flat").pack(side="left")
 208
 209     tk.Label(self.scroll_frame, text="💡 Video Analysis Results", font=("Helvetica", 18, "bold"),
 210             bg="#f8f9fa").pack(pady=(5, 10))
 211
 212     # === Summary Section ===
 213     self.summary_frame = self._create_section("📌 Video Summary")
 214     self.summary_text = self._create_text_label(self.summary_frame)
 215
 216     # === Sentiment Section ===
 217     self.sentiment_frame = self._create_section("📈 Sentiment Analysis")
 218     self.sentiment_text = self._create_text_label(self.sentiment_frame)
 219
 220     # === What's Working Section ===
 221     self.works_frame = self._create_section("✅ What's Working")
 222     self.works_text = self._create_text_label(self.works_frame)
 223
 224     # === Needs Improvement Section ===
 225     self.needs_frame = self._create_section("✖️ Needs Improvement")
 226     self.needs_text = self._create_text_label(self.needs_frame)
 227
 228     4 usages
def _create_section(self, title):
```

Fig: user_interface.py part 8

The screenshot shows a code editor with several tabs at the top: main.py, user_interface.py (which is the active tab), feedback_extractor.py, and summarizer.py. The user_interface.py tab displays the following code:

```
def _create_section(self, title):
    frame = tk.Frame(self.scroll_frame, bg="#ffffff", bd=1, relief="solid")
    tk.Label(frame, text=title, font=("Helvetica", 16, "bold"), bg="#ffffff").pack(anchor="w", padx=10, pady=(5, 0))
    frame.pack(fill="x", padx=20, pady=10)
    return frame

4 usages
def _create_text_label(self, parent):
    label = tk.Label(parent, text="", wraplength=800, justify="left",
                     font=("Helvetica", 12), bg="#ffffff", anchor="w")
    label.pack(fill="x", padx=10, pady=(0, 10))
    return label

3 usages
def _remove_formatting(self, text):
    return re.sub(pattern=r'\*\*(.*?)\*\*', repl=r'\1', text)

1 usage
def update_content(self, summary, sentiment, feedback):
    self.summary_text.config(text=self._remove_formatting(summary.strip()))

    total = sum(len(lst) for lst in sentiment.values()) or 1
    pos = (len(sentiment["positive"]) / total) * 100
    neg = (len(sentiment["negative"]) / total) * 100
    neu = (len(sentiment["neutral"]) / total) * 100
    sentiment_summary = f"Positive: {pos:.1f}%  Negative: {neg:.1f}%  Neutral: {neu:.1f}%" 
    self.sentiment_text.config(text=sentiment_summary)

    self.works_text.config(text=self._remove_formatting(feedback.get("what_works", "").strip()))
    self.needs_text.config(text=self._remove_formatting(feedback.get("needs_improvement", "").strip()))
```

The code implements a section creation function, a text labeling function, a regular expression-based formatting removal function, and an update content function. The update content function calculates sentiment percentages and updates corresponding text labels.

Fig: user_interface.py part 9

The screenshot shows a code editor with four tabs at the top: main.py, user_interface.py, feedback_extractor.py (which is the active tab), and summarizer.py. The code in feedback_extractor.py defines a class FeedbackExtractor that inherits from BaseAnalyzer. It includes two prompts: WORKS_PROMPT and IMPROVEMENT_PROMPT, which describe the task of analyzing YouTube comments about a video.

```
1 import google.generativeai as genai
2 from typing import List, Dict, Optional
3 from base_analyzer import BaseAnalyzer
4
5
6 4 usages
7
8 class FeedbackExtractor(BaseAnalyzer):
9     """Processes comments using Gemini AI to extract 'What's Working' and 'Needs Improvement' feedback."""
10
11     def __init__(self, api_key: str):
12         super().__init__(api_key)
13         genai.configure(api_key=self.api_key)
14
15     WORKS_PROMPT = (
16         "You are a professional video content analyst. Given the following YouTube comments about a video, "
17         "analyze and summarize"
18         "the most praised elements with insight. Your task is to extract deeper feedback trends that reflect what the "
19         "audience appreciated."
20         "List these as bullet points. Each bullet point must start with a bolded subtopic (e.g., **Clarity**)"
21         "followed by a concise 1-2 sentence"
22         "explanation of why this was positively received. Do not quote comments directly. Do not provide introductory "
23         "or concluding sentences."
24         "Do not group points or use themes. Be objective and informative. Note only use insights from comments, "
25         "not your simulation.\n\nComments:\n{comments}"
26     )
27
28     IMPROVEMENT_PROMPT = (
29         "You are a professional video content analyst. Given the following YouTube comments about a video, "
30         "analyze and summarize"
```

Fig: Feedback code part 1

The screenshot shows a code editor with four tabs at the top: main.py, user_interface.py, feedback_extractor.py (which is the active tab), and summarizer.py. The code in feedback_extractor.py is as follows:

```
27     "You are a professional video content analyst. Given the following YouTube comments about a video, "
28     "analyze and summarize"
29     "the most commonly mentioned criticisms or suggestions for improvement. Your task is to extract deeper "
30     "insight into what viewers found lacking."
31     "List these as bullet points. Each bullet point must start with a bolded subtopic (e.g., **Pacing**) followed "
32     "by a concise 1-2 sentence"
33     "explanation of the issue. Do not quote comments directly. Do not provide introductory or concluding sentences."
34     "Do not group points or use themes. Be direct, clear, and objective. Note only use insights from comments, "
35     "not your simulation.\n\nComments:{comments}"
36 )
37
38 usage
39 def analyze(self, comments: List[str]) -> Optional[Dict[str, str]]:
40     try:
41         model = genai.GenerativeModel("gemini-2.5-pro-exp-03-25")
42         comments_text = "\n".join(comments)
43         # Generate positive feedback
44         prompt_working = FeedbackExtractor.WORKS_PROMPT.format(comments=comments_text)
45         res_working = model.generate_content(prompt_working)
46         what_works = res_working.text.strip()
47         # Generate improvement feedback
48         prompt_needs = FeedbackExtractor.IMPROVEMENT_PROMPT.format(comments=comments_text)
49         res_needs = model.generate_content(prompt_needs)
50         needs_improvement = res_needs.text.strip()
51         print(what_works, needs_improvement)
52         return {"what_works": what_works, "needs_improvement": needs_improvement}
53     except Exception as e:
54         print(f"[ERROR] Feedback extraction failed: {e}")
55         return None
FeedbackExtractor > analyze() > try
```

Fig: Feedback code part 2

The screenshot shows a code editor window with the title "YouTubeFeedbackGenerator3.5.25". The current file is "sentiment_analyzer.py". The code implements a VADER-based sentiment analyzer. It imports SentimentIntensityAnalyzer from vaderSentiment, List and Dict from typing, and BaseAnalyzer from base_analyzer. The class SentimentAnalyzer inherits from BaseAnalyzer and uses the VADER analyzer to classify comments into positive, negative, or neutral categories based on their compound score.

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from typing import List, Dict
from base_analyzer import BaseAnalyzer

...
2 usages
class SentimentAnalyzer(BaseAnalyzer):
    """Classifies comments as positive, negative, or neutral using VADER."""

    def __init__(self):
        super().__init__()
        self.analyzer = SentimentIntensityAnalyzer()

    1 usage
    def analyze(self, comments: List[str]) -> Dict[str, List[str]]:
        """Classifies comments into sentiment categories using VADER."""
        sentiment_results = {"positive": [], "negative": [], "neutral": []}

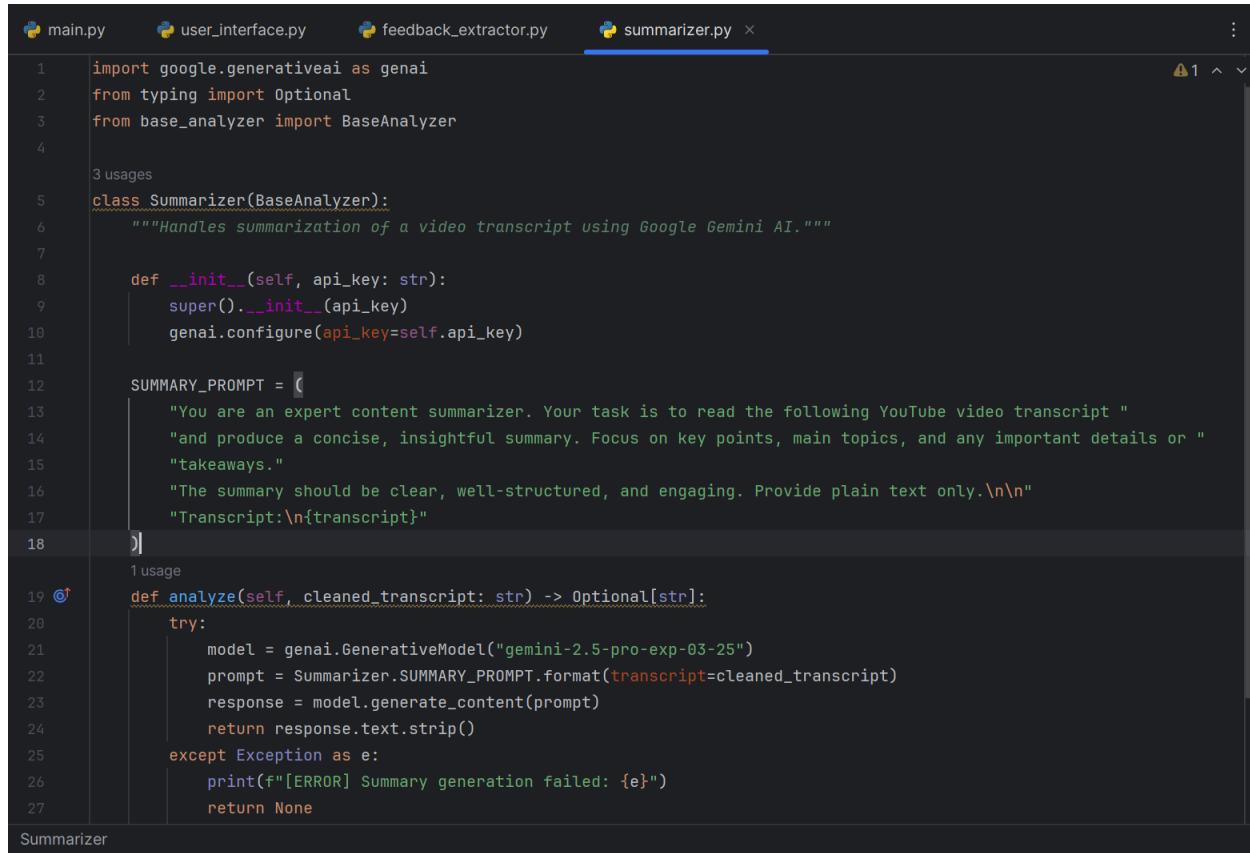
        for comment in comments:
            score = self.analyzer.polarity_scores(comment)['compound']
            if score >= 0.05:
                sentiment_results["positive"].append(comment)
            elif score <= -0.05:
                sentiment_results["negative"].append(comment)
            else:
                sentiment_results["neutral"].append(comment)

        return sentiment_results

```

Fig:

Fig: sentiment analysis code



The screenshot shows a code editor interface with several tabs at the top: main.py, user_interface.py, feedback_extractor.py, and summarizer.py (which is the active tab). The code in summarizer.py is as follows:

```
1 import google.generativeai as genai
2 from typing import Optional
3 from base_analyzer import BaseAnalyzer
4
5 3 usages
6 class Summarizer(BaseAnalyzer):
7     """Handles summarization of a video transcript using Google Gemini AI."""
8
9     def __init__(self, api_key: str):
10         super().__init__(api_key)
11         genai.configure(api_key=self.api_key)
12
13     SUMMARY_PROMPT = [
14         "You are an expert content summarizer. Your task is to read the following YouTube video transcript",
15         "and produce a concise, insightful summary. Focus on key points, main topics, and any important details or",
16         "takeaways.",
17         "The summary should be clear, well-structured, and engaging. Provide plain text only.\n\n",
18         "Transcript:\n{transcript}"
19     ]
20
21     1 usage
22     def analyze(self, cleaned_transcript: str) -> Optional[str]:
23         try:
24             model = genai.GenerativeModel("gemini-2.5-pro-exp-03-25")
25             prompt = Summarizer.SUMMARY_PROMPT.format(transcript=cleaned_transcript)
26             response = model.generate_content(prompt)
27             return response.text.strip()
28         except Exception as e:
29             print(f"[ERROR] Summary generation failed: {e}")
30
31     return None
```

Fig: summarizer code

The screenshot shows a code editor interface with several tabs at the top: main.py, user_interface.py, feedback_extractor.py, summarizer.py, and youtubeAPICon.py (the active tab). The code in youtubeAPICon.py defines a class YouTubeAPICon. It imports build, HttpError, YouTubeTranscriptApi, and Dict, List from their respective modules. The class has an __init__ method that initializes self.api_key and sets self.youtube to a build object with service_name 'youtube', version 'v3', and developer key self.api_key. The fetch_metadata method takes a video_id and returns a dictionary with keys 'title', 'duration', 'views', and 'likes'. The code uses Google's API client library to fetch video metadata.

```
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
from youtube_transcript_api import YouTubeTranscriptApi
from typing import Dict, List

2 usages
class YouTubeAPICon:
    def __init__(self, api_key: str):
        self.api_key = api_key
        self.youtube = build(serviceName='youtube', version='v3', developerKey=self.api_key)

1 usage
def fetch_metadata(self, video_id: str) -> Dict[str, str]:
    try:
        response = self.youtube.videos().list(
            part="snippet,contentDetails,statistics",
            id=video_id
        ).execute()
        items = response.get('items', [])
        if not items:
            raise ValueError("Video not found")

        metadata = items[0]
        return {
            "title": metadata['snippet']['title'],
            "duration": metadata['contentDetails']['duration'],
            "views": metadata['statistics']['viewCount'],
            "likes": metadata['statistics'].get('likeCount', '0'),
```

Fig: YouTubeAPICon class

The screenshot shows a code editor interface with several tabs at the top: main.py, user_interface.py, feedback_extractor.py, summarizer.py, and youtubeAPICon.py (which is the active tab). The code in youtubeAPICon.py is a continuation of the previous part, handling comments for a video. It includes error handling for HTTP errors, a usage note, and a detailed explanation of the fetch_comments method. The method uses the YouTube API's commentThreads endpoint to fetch up to 3000 top-level comments per page, stopping once the maximum allowed by the API (100 comments per page) is reached or the specified limit is met. It also handles pagination if there are more pages of comments.

```
29     }
30     except HttpError as e:
31         raise ConnectionError(f"Metadata API error: {e.resp.status}") from e
32
33     1 usage
34     def fetch_comments(self, video_id: str, max_comments: int = 3000) -> List[str]:
35         """Fetch up to max_comments (default 3000) top-level comments from YouTube API"""
36         comments = []
37         try:
38             request = self.youtube.commentThreads().list(
39                 part="snippet",
40                 videoId=video_id,
41                 textFormat="plainText",
42                 maxResults=100 # Fetch 100 comments per page (max allowed by YouTube API)
43             )
44             while request and len(comments) < max_comments:
45                 response = request.execute()
46                 for item in response['items']:
47                     comments.append(
48                         item['snippet']['topLevelComment']['snippet']['textDisplay']
49                     )
50                     if len(comments) >= max_comments:
51                         break # Stop fetching more comments once limit is reached
52
53             # Check if there are more pages of comments
54             if 'nextPageToken' in response and len(comments) < max_comments:
55                 request = self.youtube.commentThreads().list(
56                     part="snippet",
57                     videoId=video_id,
```

Fig: YouTubeAPICon class part 2

The screenshot shows a code editor with several tabs at the top: main.py, user_interface.py, feedback_extractor.py, summarizer.py, and youtubeAPICon.py (which is the active tab). The code in youtubeAPICon.py is as follows:

```
52     # Check if there are more pages of comments
53     if 'nextPageToken' in response and len(comments) < max_comments:
54         request = self.youtube.commentThreads().list(
55             part="snippet",
56             videoId=video_id,
57             textFormat="plainText",
58             maxResults=100,
59             pageToken=response['nextPageToken']
60         )
61     else:
62         break # No more pages, exit loop
63
64     return comments
65 except HttpError as e:
66     raise ConnectionError(f"Comments API error: {e.resp.status}") from e
67
68     1 usage
69 def fetch_transcript(self, video_id: str) -> str:
70     try:
71         transcript_list = YouTubeTranscriptApi.get_transcript(video_id)
72         transcript = ' '.join([entry['text'] for entry in transcript_list])
73         return transcript
74     except Exception as e:
75         return f"Transcript unavailable: {str(e)}"
```

Fig: YouTubeAPICon class part 3

```
1 from youtubeAPICon import YouTubeAPICon
2 from data_preprocessor import DataPreprocessor
3 import re
4
5
6     2 usages
7     <class DataRetrieval:
8         def __init__(self, api_key: str, video_url: str):
9             self.api = YouTubeAPICon(api_key)
10            self.video_url = video_url
11            self.video_id = None
12
13            1 usage
14            def validate_url(self):
15                """Validate YouTube URL and extract video ID"""
16                pattern = r"^(https://(www\.)?(youtube\.com|youtu\.be)/(watch\?v=)?([a-zA-Z0-9_-]{11})$"
17                match = re.match(pattern, self.video_url)
18                if not match:
19                    raise ValueError("Invalid YouTube URL")
20                self.video_id = match.group(4)
21
22            1 usage
23            def get_metadata(self) -> dict:
24                """Fetch and return video metadata"""
25                if not self.video_id:
26                    raise RuntimeError("URL validation required before data retrieval")
27                metadata = self.api.fetch_metadata(self.video_id)
28                # Enforce operational constraints
29                if metadata["duration"].startswith("PT") and "H" in metadata["duration"]:
30                    # Checks for 1+ hour videos
```

Fig: Data Retrieval .py part 1

```
 21     """Fetch and return video metadata"""
 22     if not self.video_id:
 23         raise RuntimeError("URL validation required before data retrieval")
 24     metadata = self.api.fetch_metadata(self.video_id)
 25     # Enforce operational constraints
 26     if metadata["duration"].startswith("PT") and "H" in metadata["duration"]:
 27         # Checks for 1+ hour videos
 28         raise ValueError("Video exceeds the allowed duration of 1 hour")
 29
 30     return metadata
 31
 32     1 usage
 33     def get_transcript(self) -> str:
 34         """Fetch and return cleaned transcript"""
 35         if not self.video_id:
 36             raise RuntimeError("URL validation required before data retrieval")
 37         raw_transcript = self.api.fetch_transcript(self.video_id)
 38         return DataPreprocessor.clean_transcript(raw_transcript)
 39
 40     1 usage
 41     def get_comments(self) -> list[str]:
 42         """Fetch and return filtered comments"""
 43         if not self.video_id:
 44             raise RuntimeError("URL validation required before data retrieval")
 45         raw_comments = self.api.fetch_comments(self.video_id, max_comments=500)
 46
 47         if len(raw_comments) < 100:
 48             raise ValueError("Video does not have the required minimum of 100 comments")
 49
 50     return DataPreprocessor.filter_comments(raw_comments)
```

Fig: data retrieval.py part 2

```
1 import re
2 from typing import List
3
4
5 4 usages
6
7 class DataPreprocessor:
8     1 usage
9         @staticmethod
10        def remove_filler_words(transcript: str) -> str:
11            """Removes common filler words from the transcript"""
12            fillers = ["um", "uh", "like", "you know", "so", "actually", "basically", "literally", "right"]
13            pattern = r'\b(' + '|'.join(fillers) + r')\b'
14            cleaned_transcript = re.sub(pattern, repl: '', transcript, flags=re.IGNORECASE)
15            return cleaned_transcript
16
17     1 usage
18         @staticmethod
19        def clean_transcript(raw_transcript: str) -> str:
20            """Cleans transcript by removing timestamps, filler words, and spam content"""
21            cleaned = re.sub(pattern: r'\[\d+:\d+(?:\.\d+)?]', repl: '', raw_transcript) # Remove timestamps
22            cleaned = DataPreprocessor.remove_filler_words(cleaned) # Remove filler words
23            return cleaned.lower()
24
25     1 usage
26         @staticmethod
27        def filter_comments(raw_comments: List[str]) -> List[str]:
28            # Remove duplicate comments
29            seen = set()
30            filtered = []
31
```

Fig: data_preprocessor.py part 1

The screenshot shows a code editor interface with multiple tabs at the top: main.py, user_interface.py, feedback_extractor.py, summarizer.py, data_retrieval.py, and data_preprocessor.py (which is the active tab). The code in data_preprocessor.py is as follows:

```
16     """Cleans transcript by removing timestamps, filler words, and spam content"""
17     cleaned = re.sub(pattern=r'\[\d+:\d+(?::\d+)?]', repl='', raw_transcript) # Remove timestamps
18     cleaned = DataPreprocessor.remove_filler_words(cleaned) # Remove filler words
19     return cleaned.lower()
20
21     1 usage
22     @staticmethod
23     def filter_comments(raw_comments: List[str]) -> List[str]:
24         # Remove duplicate comments
25         seen = set()
26         filtered = []
27         # Remove spam content using regex patterns
28         spam_patterns = [
29             r'\b(free|subscribe|click here|visit our website|spam)\b',
30             r'(http|https://)\S+', # URLs
31             r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b', # Emails
32             r'\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b' # IP addresses
33         ]
34
35         for comment in raw_comments:
36             if comment and comment not in seen:
37                 for pattern in spam_patterns:
38                     comment = re.sub(pattern, repl='', comment, flags=re.IGNORECASE)
39                 seen.add(comment)
40                 filtered.append(comment)
41
42     return filtered
```

Fig: data_preprocessor.py part 2

```

1  from abc import ABC, abstractmethod
2  from typing import Any
3
4  class BaseAnalyzer(ABC):
5      """
6          Abstract base class for all analyzers.
7          Ensures consistency in the analyze() method implementation.
8      """
9
10     def __init__(self, api_key: str = None):
11         self.api_key = api_key
12
13     @abstractmethod
14     def analyze(self, data: Any) -> Any:
15         """Abstract method to be implemented by all subclasses."""
16         pass

```

Fig: base analyzer(abstract class connecting summarizer, feedback_extractor and sentiment analysis)

Future deliverables:

- **Move:** Move API keys to a secure configuration file to eliminate hardcoding risks.
- **Optimize:** Optimize comment processing through batching or asynchronous execution.
- **Enhance:** Enhance results UI with sentiment graphs, word clouds, and thumbnails.
- **Transition:** Transition of the application from desktop (Tkinter) to a web-based interface.
- **Improve:** Improve spam filtering using smarter rule sets or lightweight ML classifiers.

Invoice (in hours):

Documentation	Ayush – 33, Anthony – 31, Nabin – 32
Research	Ayush – 19, Anthony – 16, Nabin – 17
Design	Ayush – 16, Anthony – 10, Nabin – 14
Code	Ayush – 18, Anthony – 18, Nabin - 19
Testing	Ayush – 9, Anthony – 10, Nabin – 7

Estimated for Remaining Work:

As per Product Backlog: $50 - 20 = 30$ days.

5 extra days to complete what didn't work well in Sprint 2.

Predicted Total: 35 days

Actual Total: 30 days

Tools Used:

- ❖ Software used for this system – PyCharm
- ❖ Software used to produce your documentation – Microsoft Word and PowerPoint
- ❖ Other Tools – <https://www.draw.io/>

1. Quality Attributes – Runtime Qualities

Attribute	Bus Rank	Design align	Run time Quality Attribute Documents
Performance			<p>Definition: System performance describes attributes of the system including navigational abilities and connectivity with interfacing systems. May include system response time, number of transactions per minute, and other critical performance characteristics which define this project.</p> <p>Reason for Business Rank <> 3: Performance is essential to deliver timely video summaries and real-time sentiment analysis for a responsive user experience.</p> <p>Explanation of Design Alignment <> Meets: The system employs optimized API calls, efficient data retrieval, and parallel processing to ensure performance targets are met.</p>
Reliability			<p>Definition: Reliability defines the ability of the system to operate correctly over time. Includes consistent stability of the system during available hours.</p>

			<p>Reason for Business Rank <> 3: High reliability is critical to maintain user trust and ensure continuous availability of video data and analysis.</p> <p>Explanation of Design Alignment <> Meets: Redundant mechanisms, robust error handling, and comprehensive logging are implemented to achieve the required reliability.</p>
Scalability			<p>Definition: Scalability defines the parameters of growth within a given period. The solution must scale so that additional capacity can be added expediently in a technically and economically feasible way.</p> <p>Reason for Business Rank <> 3: Scalability is needed to handle increasing volumes of video and comment data as the user base grows.</p> <p>Explanation of Design Alignment <> Meets: A modular and distributed architecture enables the system to scale horizontally as needed.</p>
Security			<p>Definition: Security is the ability of the system to resist unauthorized attempts to access the system and/or its components.</p> <p>Reason for Business Rank <> 3: Security protects sensitive video metadata, transcripts, and user data from unauthorized access.</p> <p>Explanation of Design Alignment <> Meets: The system employs encryption, access controls, and secure API integrations to meet stringent security requirements.</p>
Availability			<p>Definition: Availability defines the days and hours the system is anticipated and/or required to be operational.</p> <p>Reason for Business Rank <> 3: Continuous system availability is crucial to support global users and uninterrupted service.</p> <p>Explanation of Design Alignment <> Meets: The</p>

			deployment includes redundant systems and failover mechanisms to ensure 24/7 availability.
Supportability			<p>Definition: Supportability defines the requirement that problems can be diagnosed and addressed quickly, with appropriately trained resources.</p> <p>Reason for Business Rank <> 3: Quick diagnosis and resolution of issues minimize downtime and maintain service quality.</p> <p>Explanation of Design Alignment <> Meets: Integrated logging, monitoring tools, and automated alerts are in place to support rapid troubleshooting.</p>
Traceability			<p>Definition: Traceability defines the ability to trace the history, application, or location of an item or activity by means of recorded identification.</p> <p>Reason for Business Rank <> 3: It is important for audit trails and debugging to have full traceability of data and changes.</p> <p>Explanation of Design Alignment <> Meets: Detailed logging of data transformations (e.g., transcript cleaning, comment filtering) ensures complete traceability throughout the processing pipeline.</p>
User Usability			<p>Definition: User Usability defines how easy the interface is to use to successfully complete tasks, including clear navigation, terminology, and error messaging.</p> <p>Reason for Business Rank <> 3: A user-friendly interface directly impacts user satisfaction and adoption.</p> <p>Explanation of Design Alignment <> Meets: The design features an intuitive dashboard with clear visualizations for video summaries, sentiment analysis, and structured feedback.</p>

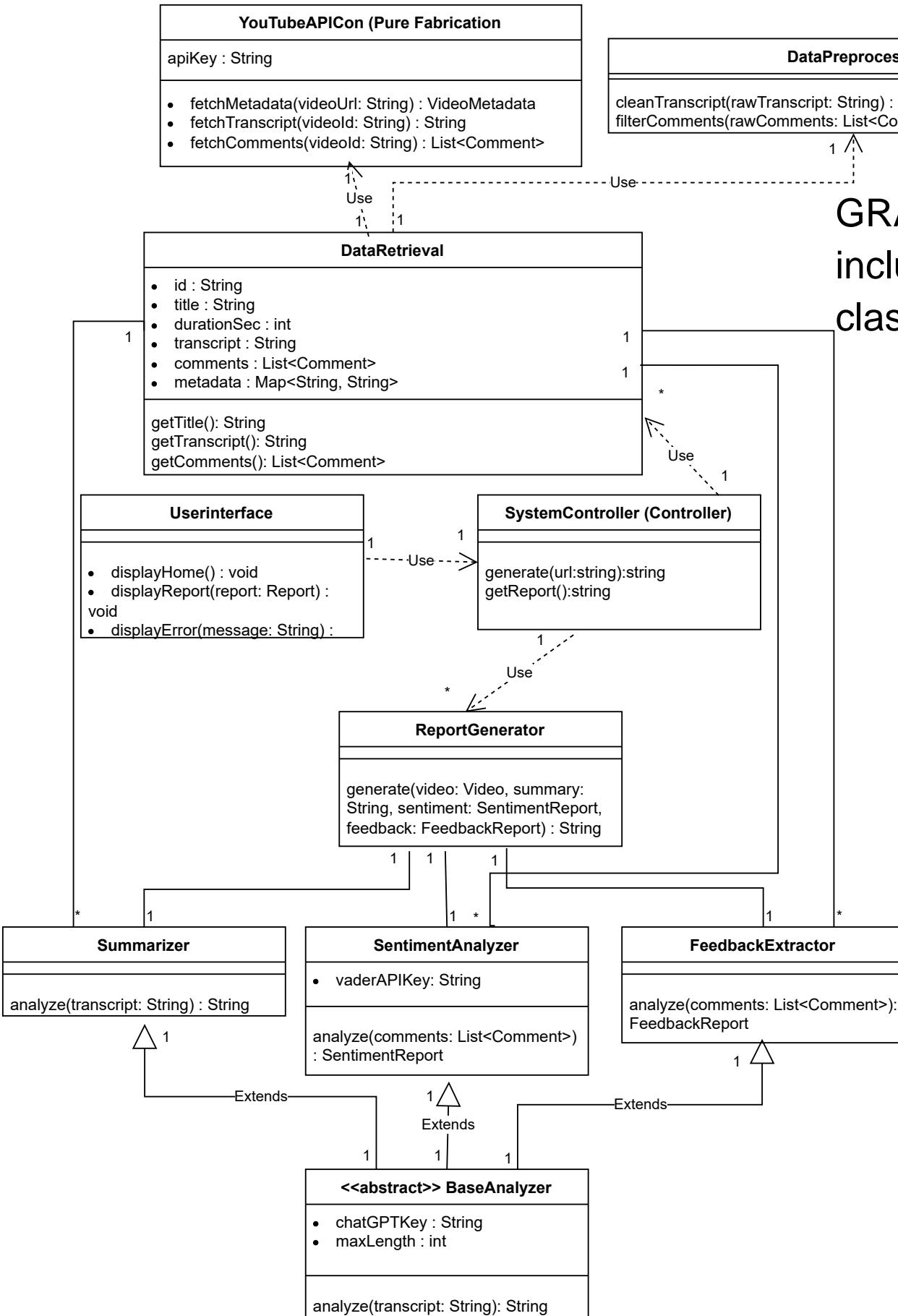
2. Quality Attributes- Development Qualities

Attributes	Bus Rank	Design Align	Run Time Quality Attributes Documents
Reusability			<p>Definition: The reusability of the system is the ability to reuse portions of the system in other applications.</p> <p>Reason for Business Rank <> 3: High reusability reduces development time and cost by allowing components to be leveraged in future projects.</p> <p>Explanation of Design Alignment <> Meets: The system is designed with modular components (data retrieval, transcript cleaning, sentiment analysis) that can be reused across different applications.</p>
Extensibility			<p>Definition: Extensibility defines the degree to which the system can be extended to incorporate new functionality without changing the fundamental architecture, design, or core source code.</p> <p>Reason for Business Rank <> 3: Extensibility is essential to accommodate evolving business needs and integration with additional data sources.</p> <p>Explanation of Design Alignment <> Meets: Published interfaces such as APIs and metadata definitions allow for seamless extension of system capabilities.</p>
Modifiability			<p>Definition: Modifiability (maintainability) defines the efficiency of making enhancements or maintenance changes.</p> <p>Reason for Business Rank <> 3: High modifiability minimizes disruption and cost when updating the system.</p> <p>Explanation of Design Alignment <> Meets: A well-structured, documented, and modular codebase allows for quick modifications without affecting core functionality.</p>
Ease of Integration			<p>Definition: Ease of Integration defines how easily the application or its components can be integrated or packaged</p>

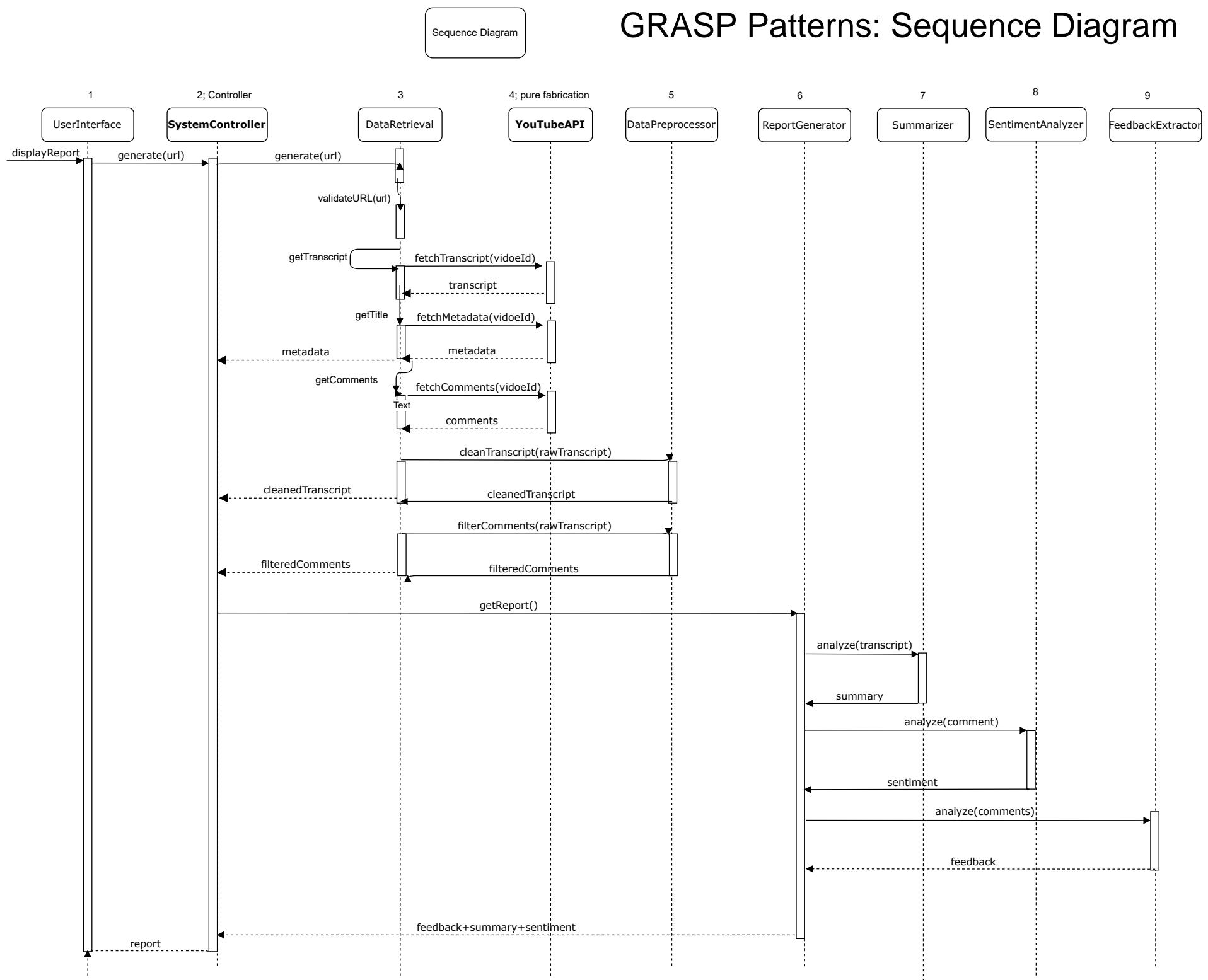
		<p>with new applications.</p> <p>Reason for Business Rank <> 3: Seamless integration is key to interoperability with external systems and tools.</p> <p>Explanation of Design Alignment <> Meets: The system uses standardized, RESTful APIs and common data formats to facilitate integration.</p>
Phase-In “Ability”		<p>Definition: Phase In “ability” (subset-ability) defines how the application accommodates implementation/rollout schedules without impacting overall design.</p> <p>Reason for Business Rank <> 3: Phased deployment minimizes risk and allows for early user feedback.</p> <p>Explanation of Design Alignment <> Meets: The architecture supports incremental deployment, enabling features to be released in stages.</p>
Conceptual Integrity		<p>Definition: Conceptual Integrity refers to the ability of the architecture to communicate a clear, concise vision in line with the enterprise architecture.</p> <p>Reason for Business Rank <> 3: Ensures a unified approach and consistency across the system, which is vital for long-term maintainability.</p> <p>Explanation of Design Alignment <> Meets: Consistent design principles and a unified processing pipeline ensure all components work cohesively.</p>
Testability		<p>Definition: Testability describes the ability to test each component independently (unit tests) as well as in an integrated environment.</p> <p>Reason for Business Rank <> 3: High testability reduces risk of defects and ensures reliable system performance with each update.</p> <p>Explanation of Design Alignment <> Meets: A comprehensive suite of automated tests is integrated into the</p>

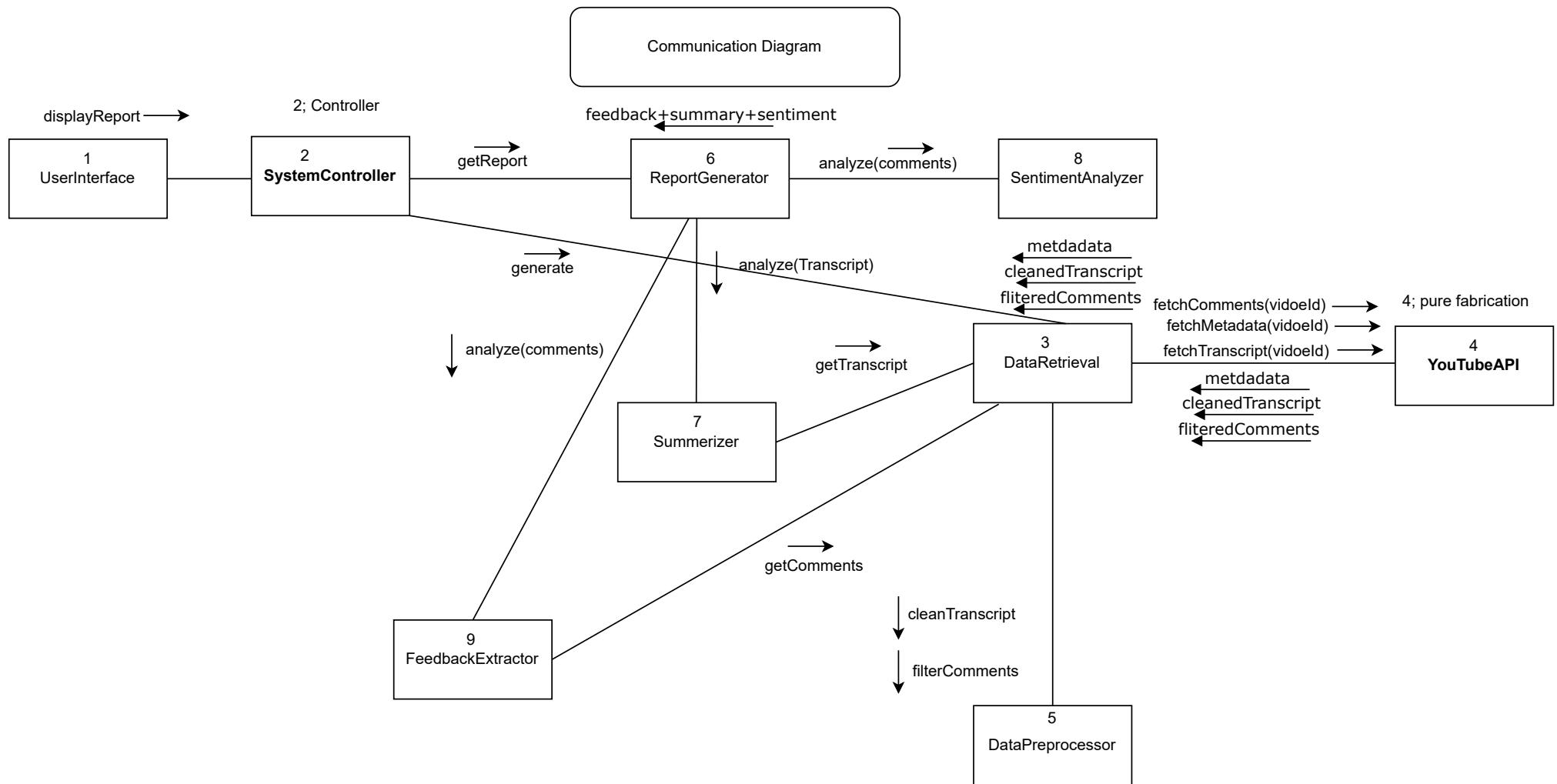
		CI/CD pipeline to verify all critical components.
Portability		<p>Definition: Portability measures the ease with which the system can be moved to different platforms (hardware, operating systems, virtualization, etc.).</p> <p>Reason for Business Rank <> 3: Portability provides flexibility in deployment and minimizes vendor lock-in.</p> <p>Explanation of Design Alignment <> Meets: The system is developed using cross-platform frameworks and containerization, ensuring compatibility across environments.</p>
Developer Complexity		<p>Definition: Developer Complexity is how easy it is for developers to understand and create the system.</p> <p>Reason for Business Rank <> 3: Lower complexity accelerates onboarding and reduces development errors.</p> <p>Explanation of Design Alignment <> Meets: The architecture is simplified with clear module boundaries and detailed documentation, reducing the overall complexity.</p>
Buildability		<p>Definition: Buildability defines the degree to which the system can be compiled, packaged, archived, and deployed using automated tools and standard processes.</p> <p>Reason for Business Rank <> 3: Efficient build processes accelerate time-to-market and ensure consistency.</p> <p>Explanation of Design Alignment <> Meets: The integration of an automated CI/CD pipeline and build automation tools streamlines deployment and minimizes errors.</p>

GRASP Patterns included in the class diagram



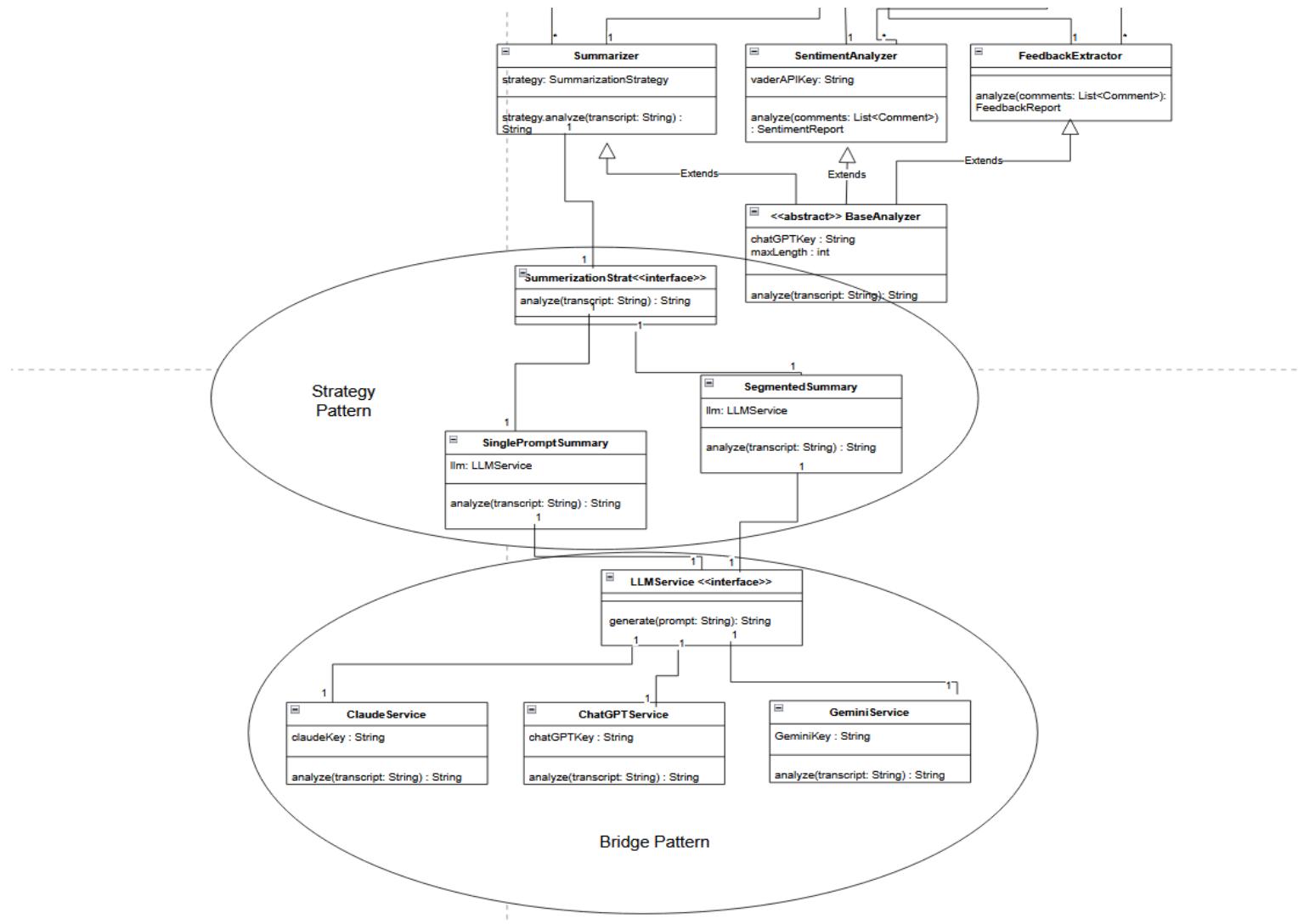
GRASP Patterns: Sequence Diagram



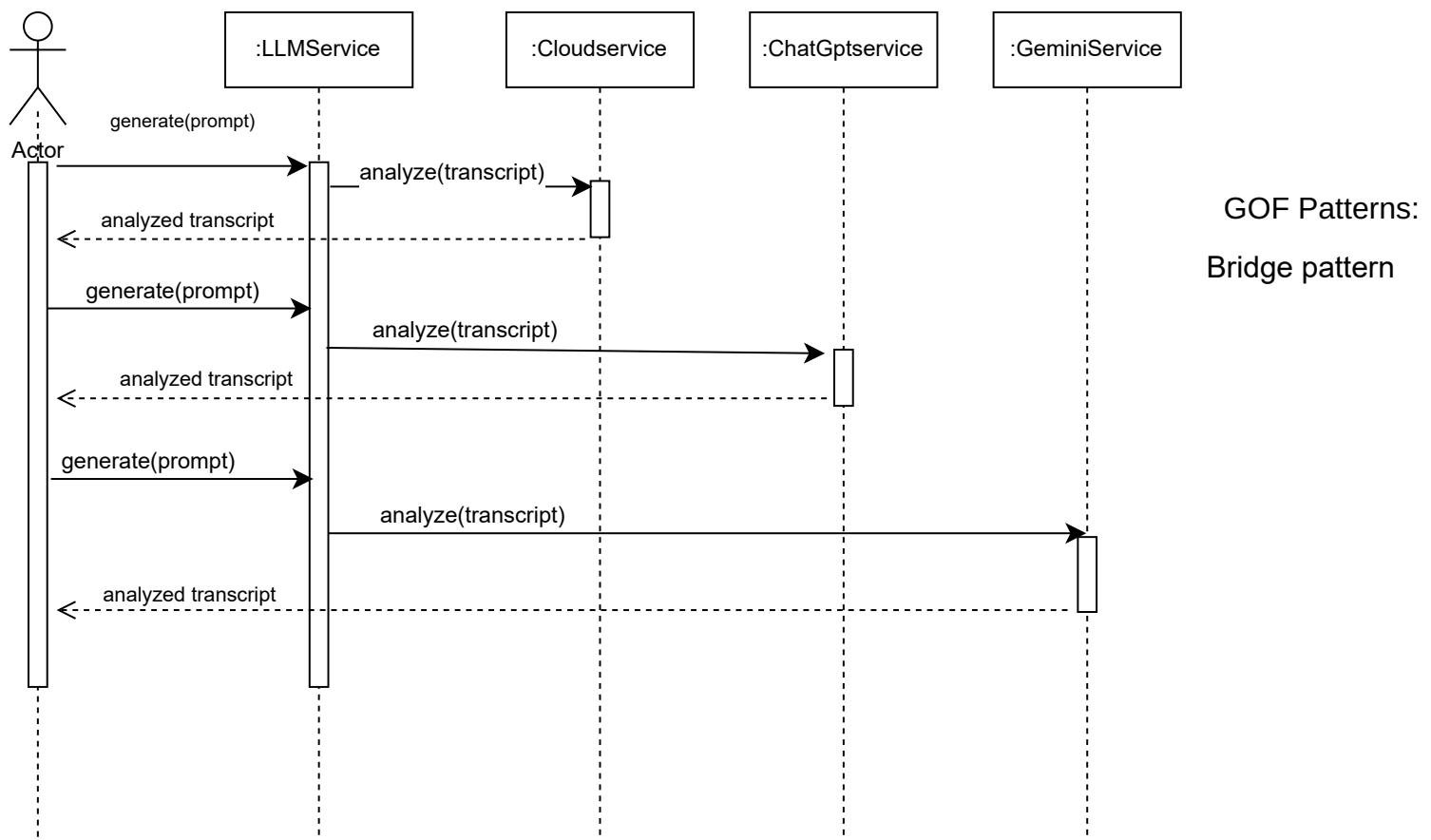


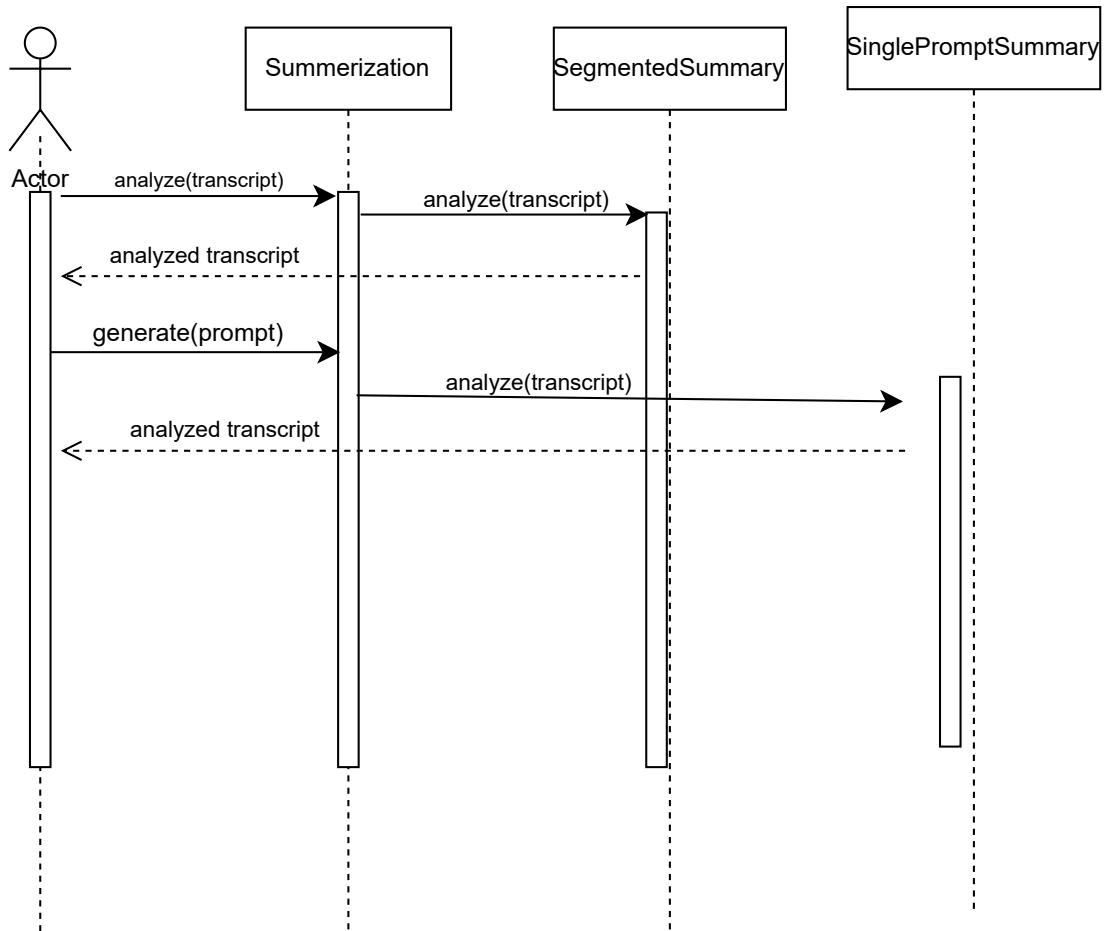
GRASP Patterns: Communication Diagram

DIAGRAM



GOF PATTERNS: Strategy and Bridge Patterns





GOF Patterns:
Strategy pattern