Summify - Technical Specification

Student 1 Name: Benjamin Olojo **ID Number:** 19500599

Student 2 Name: Przemyslaw Majda **ID Number:** 20505049

Staff Member Consulted for supervision: Dr. Jennifer Foster

Table of Contents

1. Introduction

1.1 Overview

Summify is a web application that allows users to generate textual summaries of YouTube videos through Natural Language Processing. The application primarily aims to automate the process of summarising lengthy educational videos, such as video lectures, through creating abstractive summaries of the video transcripts. Within the application, video transcripts are fetched through user-specified URLs using the python YouTube Transcript API[1]. These video transcripts are then divided into 5 minute segments which are used as input material for generating abstractive text summaries of the video segments. An overall text summary is then generated from the segment summaries and contains key information that is presented to the user.

The video transcripts are summarised using the GPT-3[2] language model developed and maintained by OpenAI, accessible within the application through the OpenAI Developers API. A zero shot summarisation approach is utilised with two GPT-3 models, Curie and DaVinci, being used to produce the summaries.

The application additionally features an Information Retrieval functionality that leverages the spaCy NLP[3] library and TextRank[4] Keyword Extraction algorithm to identify key terms within the video transcripts. Once keywords are identified, the user is provided with links to the most relevant Wikipedia pages through the python Google Search API[5].

[1] https://pypi.org/project/youtube-transcript-api/
[2] https://openai.com/api/
[3] https://spacy.io/
[4] https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf
[5] https://developers.google.com/custom-search/v1/overview

1.2 Glossary

NLP → Natural Language Processing refers to the application of computational techniques to the analysis and synthesis of natural language and speech.

IR → Information Retrieval refers to the techniques of storing, recovering and disseminating recorded data through the use of a computerised system.

Fact Extraction → The task of automatically extracting structured information from unstructured and/or semi-structured machine-readable documents.

GPT-3 → Generative Pre-trained Transformer 3 is an autoregressive language model that uses deep learning to produce human-like text.

Zero Shot Summarisation → A text summarisation approach where a pre-trained language model produces summaries for novel categories of samples.

Extractive summarisation → Aims to identify the salient information within text that is then extracted and grouped to form a concise summary.

Abstractive summarisation → Abstractive summarisation is a method for generating novel sentences to represent the salient information in a body of text.

Lemma → The canonical form, dictionary form, or citation form of a set of word forms.

Stop words → Commonly used words that don't provide much context within a natural language, including articles, prepositions, pronouns, conjunctions, etc

Pre-training → Pre-training refers to training a model with a series of tasks to help it produce useful data representations that can be used in other tasks.

Fine-tuning → Re-training a pre-trained model using custom data so that the weights of the original model are updated to account for the characteristics of the domain data.

spaCy → Open-source python library for advanced natural language processing.

PageRank → An algorithm used by Google Search to rank websites in their search engine results.

ROUGE → Recall Oriented Understudy for Gisting Evaluation, or ROUGE, is a set of metrics for evaluating automatic summarisation in NLP (Lin 2004) The metrics compare an automatically produced summary against a reference or goal (human-produced) summary.

2. System Architecture

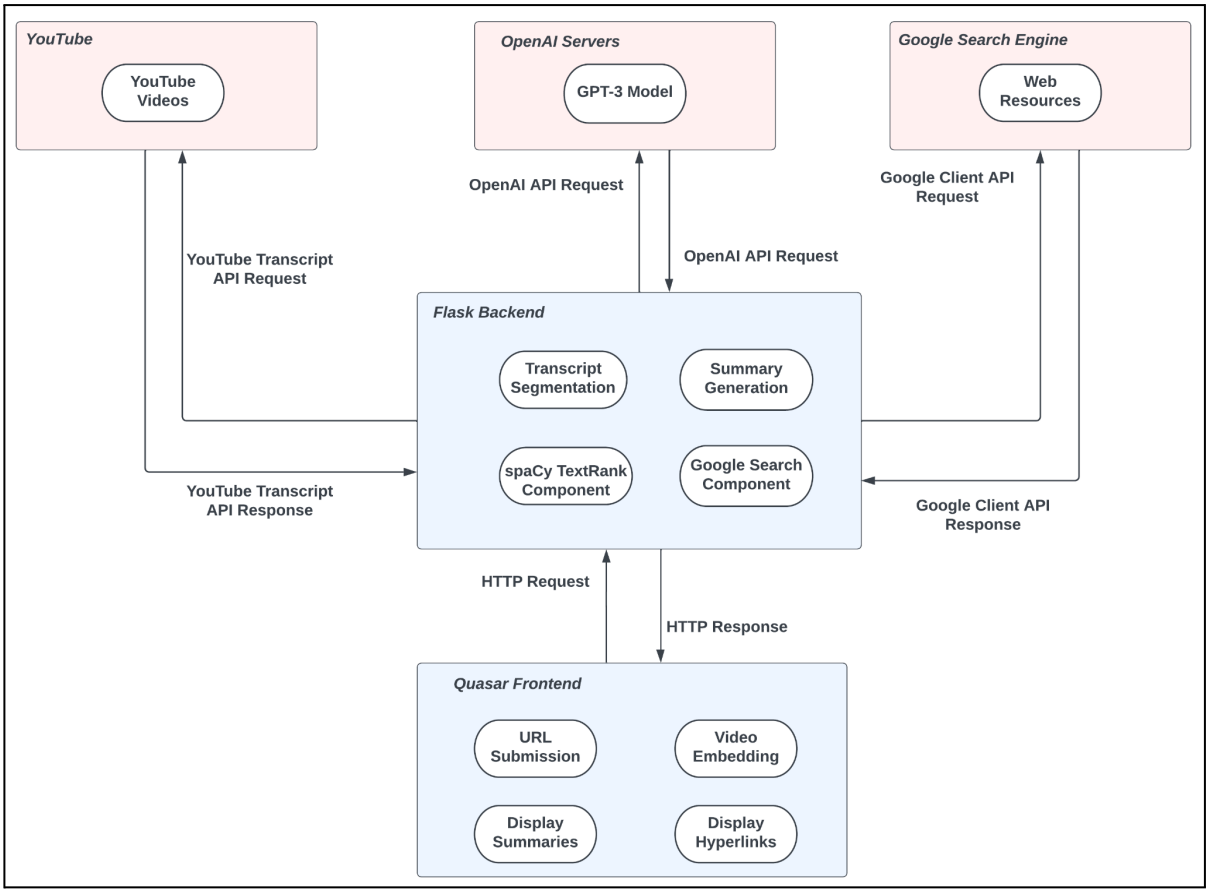
2.1 Overall System Architecture

Designed to adhere to the client-server model, the Summify web application features a Python Flask[6] application running on a backend server and a JavaScript Quasar[7] application running on a frontend server. The application also operates in conjunction with third-party systems that include GPT-3 hosted on the OpenAI servers, the Google Search Engine service and the YouTube application.

The Flask backend contains the python modules for making API calls to these third-party systems and utilises the results for video transcription and summary generation, fact extraction of key terms and the Google search results. The Vue.js based Quasar frontend serves as the main mechanism for obtaining user input and displaying results within a webpage available to the user through a web browser.

Communication between the backend and frontend applications is facilitated through HTTP requests between the Flask and Quasar applications API endpoints.

Fig. 1 - Summify System Architecture



[6] https://flask.palletsprojects.com/en/2.2.x/
[7] https://quasar.dev/

2.2 Backend Architecture

In addition the Flask application, the backend contains a number of modules for completing tasks that include fetching the transcripts of the YouTube video, segmenting the transcript into shorter segments, making API requests to GPT-3, performing concurrent API requests, utilising the TextRank component for fact extraction within a spaCy NLP pipeline and making google searches for the most relevant Wikipedia pages.

The Flask server is instantiated with Cross-origin resource sharing (CORS) enabled to allow it to interface with the frontend Quasar application. The application is configured to have an endpoint for summarising video transcripts and an endpoint for fetching the hyperlinks of Wikipedia pages matching key terms extracted from the transcripts. It is also configured to cache the recent responses to the GET requests it receives.

2.3 Frontend Architecture

The frontend consists of a single-page Quasar application that contains the endpoint for making a GET request to the Flask application and modules for displaying the results within the web page served to the user.

Within the Quasar application, the main Vue layout is extended by other web pages and contains the header and scroll function. The header holds the title of the page, and the scroll function minimises the header once the user has scrolled down in order to save space on the page.

The index page, which extends the main layout, contains an input field for users to enter a YouTube link. When the submit button is pressed, a function processes the link to extract the video ID, and sends that information to the backend in the form of a HTTP GET request. The responses containing the summaries and links to relevant Wikipedia pages are then displayed on the index page, along with the embedded youtube video. The index page uses dynamic rendering (Vue) to render the summarised transcript of the video when it is returned from the backend, without needing to refresh the page.

The blog page also extends the main layout. This page simply contains the development blog which documents our progress from the start of the project to the end.

3. High-level Design

Transcript Summarisation

Within the frontend Quasar UI, the user is presented with a text input field where they are prompted to enter a YouTube video URL. Once a valid URL is submitted, a GET request is made to the backend `/summarise` route from the Quasar application for the video transcript obtained through the YouTube transcript API.

The video transcripts are returned with timestamps and are divided into 5 minute segments so that the segment summaries can be produced. The video and transcript lengths are then checked to ensure adherence to the maximum token limits for the GPT-3 models provided by OpenAI.

To ensure that transcripts that are auto-generated by YouTube have correct sentence structure and punctuation, each auto-generated transcript is rewritten concurrently through text generation using the DaVinci GPT-3 model. If it is a manually written transcript, correct sentence structure and punctuation is assumed.

The structured transcript segments are then summarised concurrently using the Curie GPT-3 model with an input prompt that provides context of the summarisation task. Once the transcript segments are summarised, they are joined together and a request for an overall summary is made to the DaVinci GPT-3 model. The responses are cached within the Flask application, with a key denoting the video ID.

Once all of the summaries have been generated, they are displayed for the user within the frontend Quasar application.

Information Retrieval

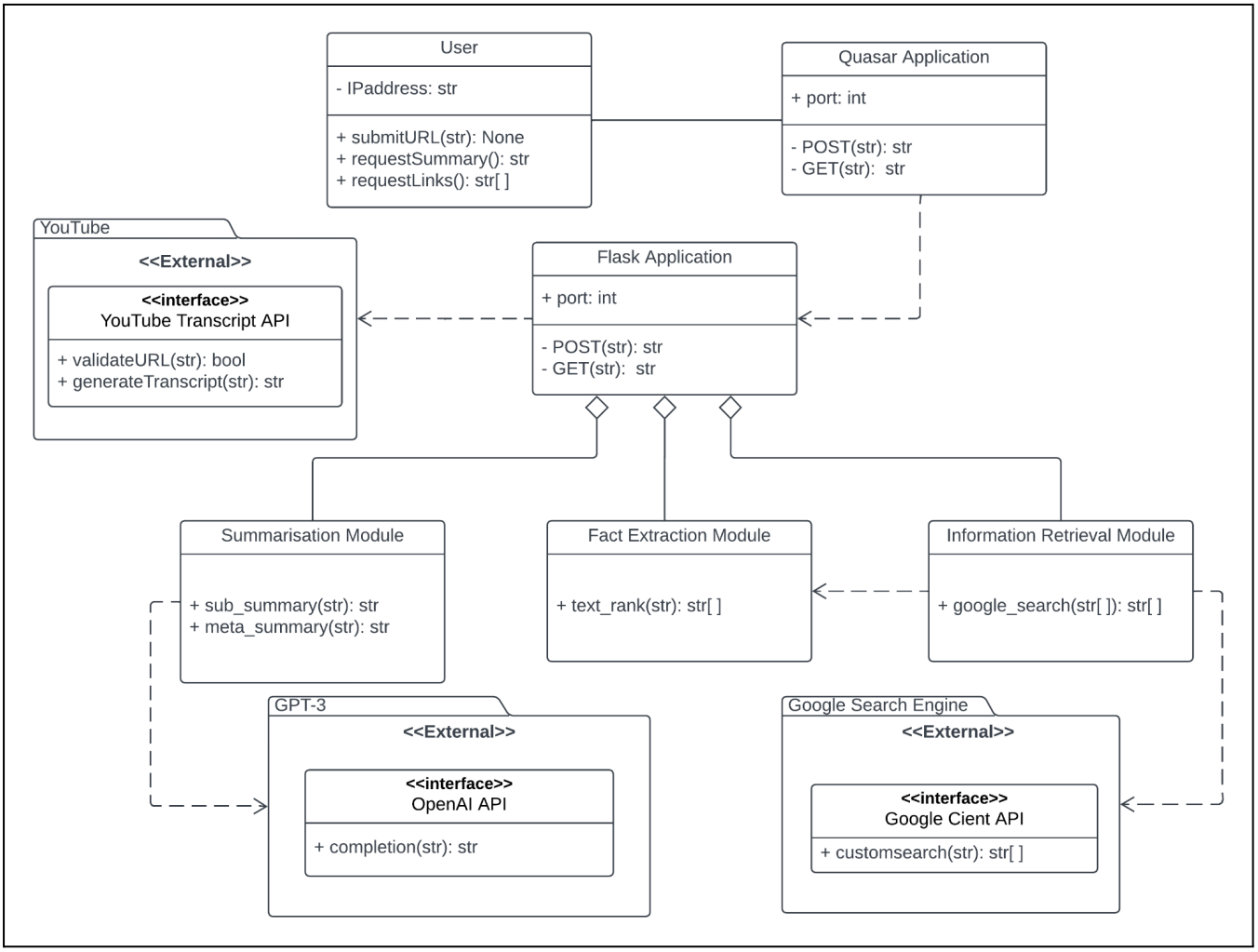
Once the summaries have been completed, the Quasar application automatically makes a GET request to the `/links` route for the most relevant Wikipedia links of the key terms extracted using the TextRank algorithm.

The TextRank algorithm is applied through a pipeline component of the spaCy NLP model with custom components for lemmatisation and stop word removal. The top ranked terms extracted from the transcript summaries are then passed as parameters to a custom Google Search Engine via the Google Client API. This programmable search engine returns links to the Wikipedia pages of the most relevant topics from the video transcript summary.

Once these links are returned, they are displayed for the user within the frontend Quasar application.

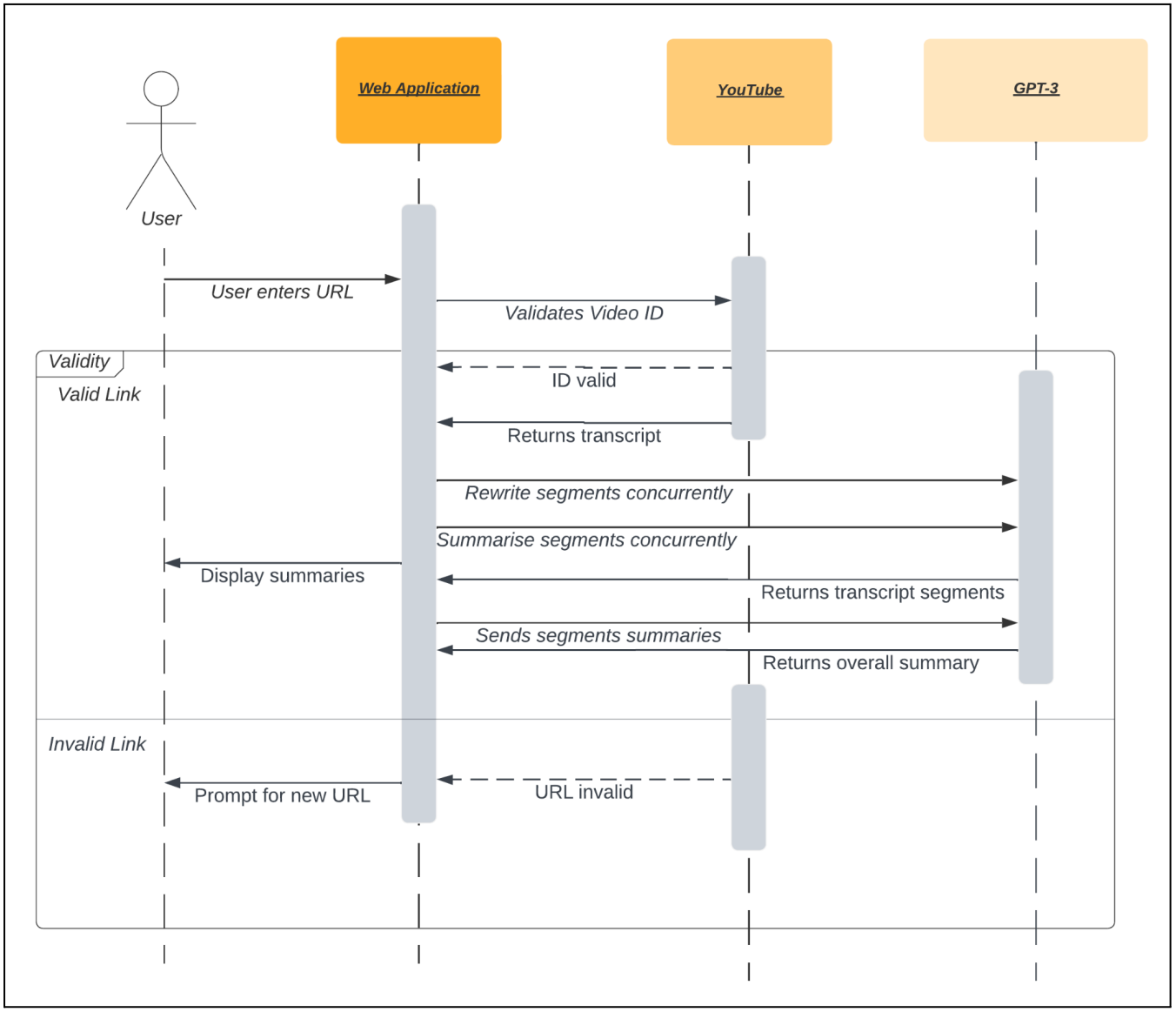
3.1 Class Diagram

Fig. 2 - Summify Class Diagram



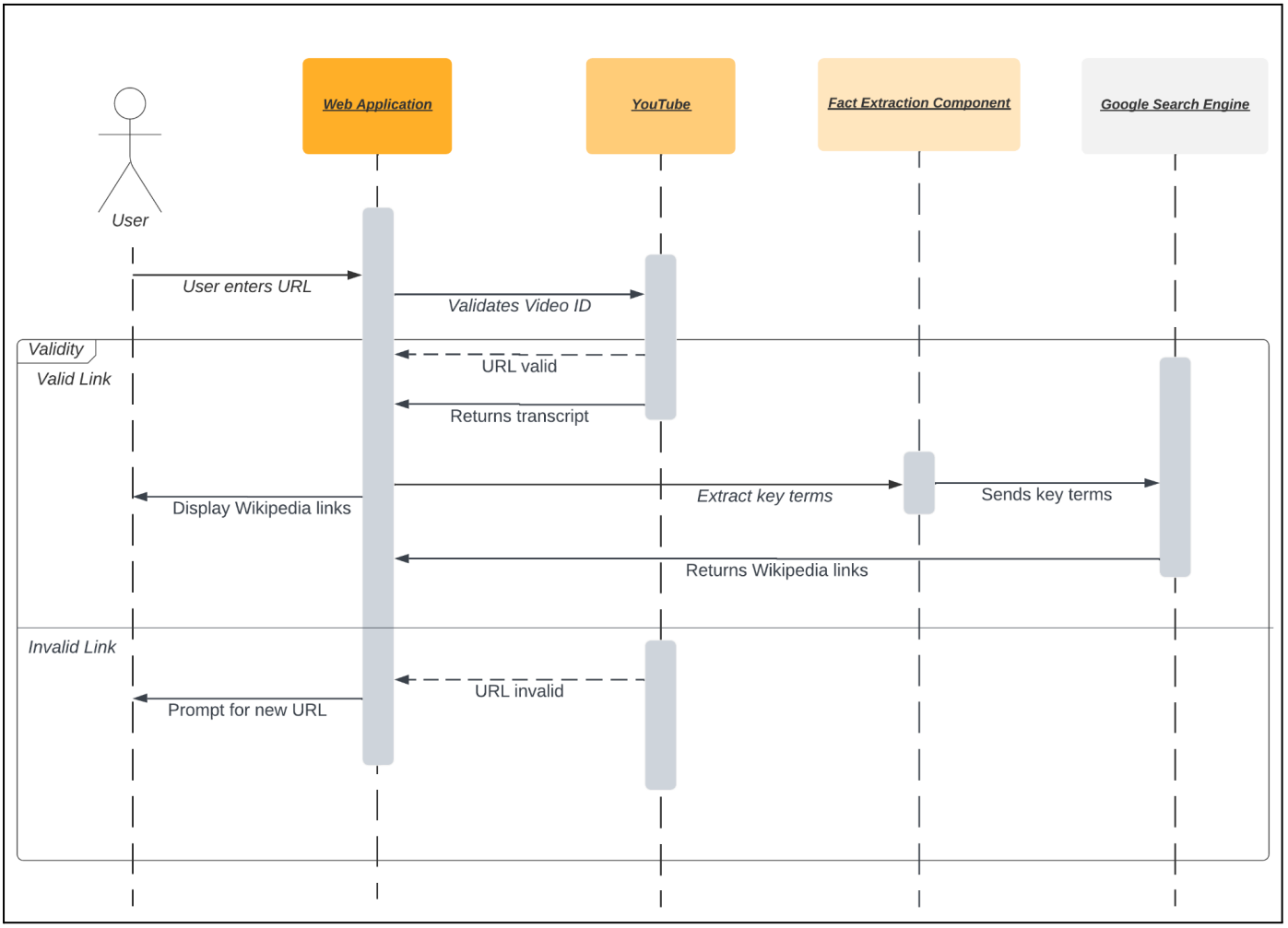
3.2 Summarisation Sequence Diagram

Fig. 3 - Video Summarisation Sequence Diagram



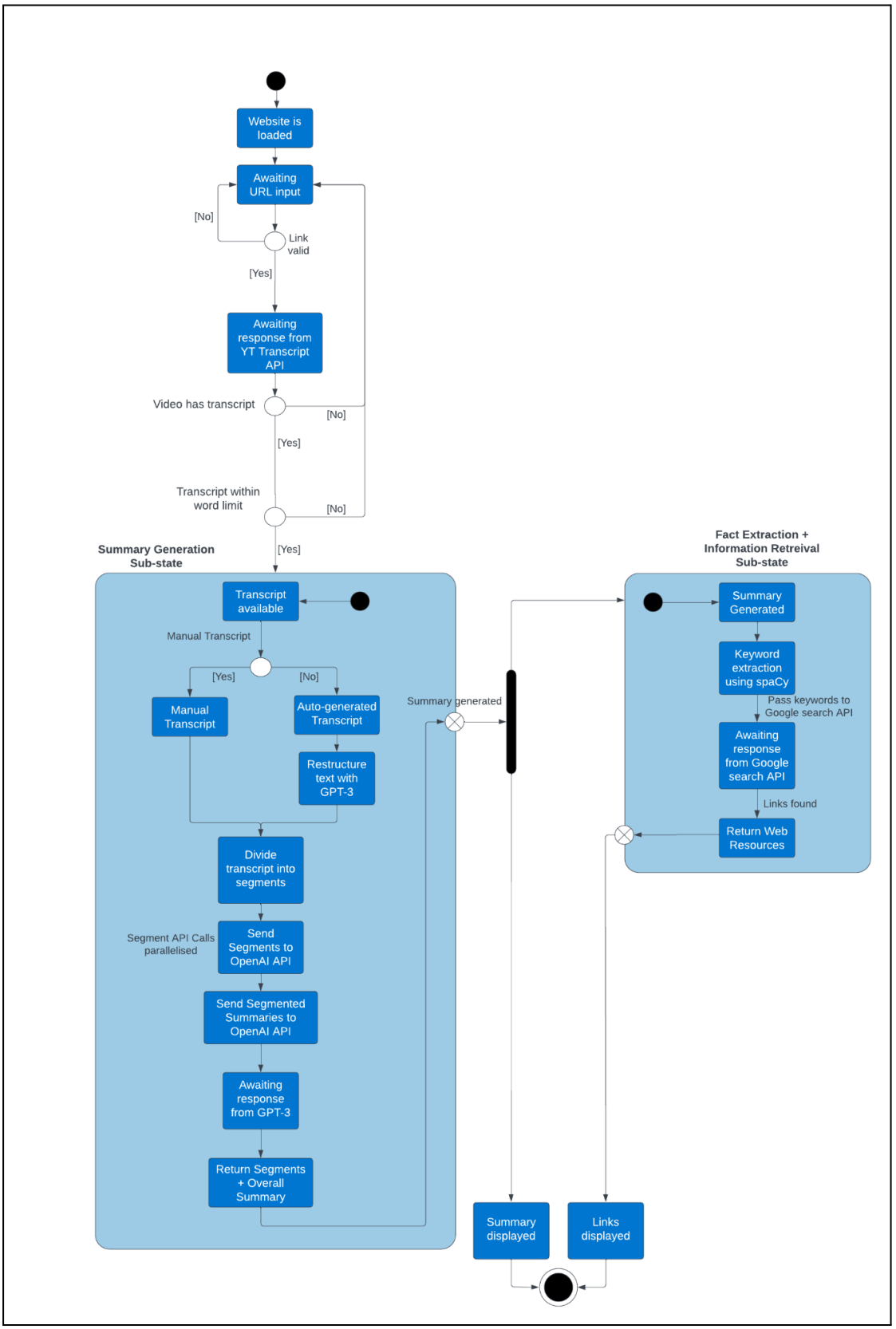
3.3 Information Retrieval Sequence Diagram

Fig. 4 - Information Retrieval Sequence Diagram



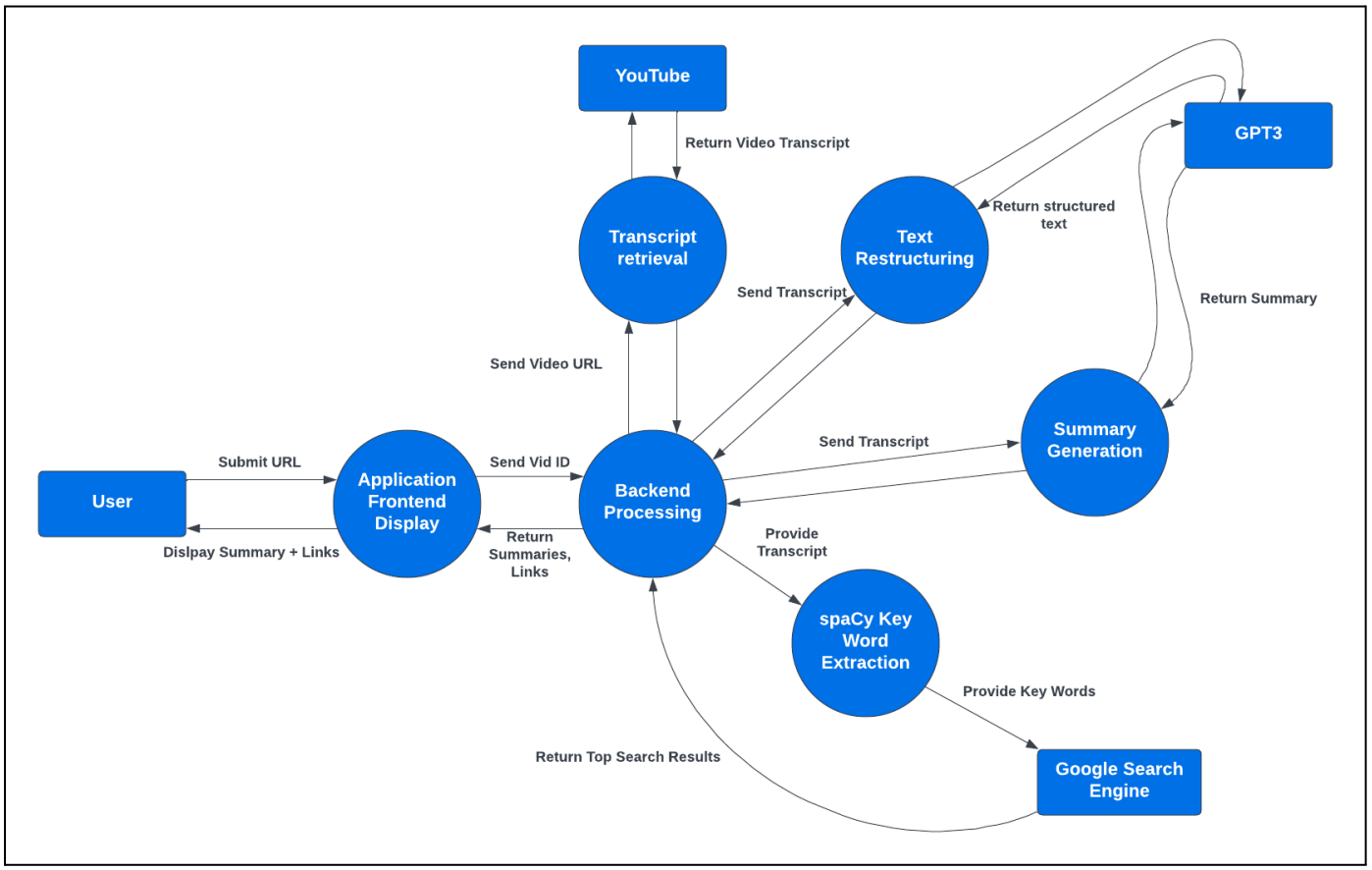
3.4 State Machine Diagram

Fig. 5 - Summify State Machine Diagram



3.5 Data Flow Diagram

Fig. 6 - Summify Data Flow Diagram



4. Problems and Resolution

GPT-3 Maximum Token Length

Each of the GPT-3 models that are available through the OpenAI API have a maximum token value which limits the length of text that can be accepted and produced by the GPT-3 model. When summarising long form videos, we were initially unable to generate summaries on the entire video transcript that exceeded the maximum token length.

To tackle this, we created python modules for segmenting the overall video transcripts into 5 minute segments, producing sub-summaries for each segment and an overall summary of these segment summaries. This involved setting parameters and curating prompts to the GPT-3 model that maximises its performance for zero shot abstractive summarisation on video transcripts.

Due to the maximum number of tokens that can be accepted for the overall summary, we also outlined a maximum video length of 100 minutes.

Unstructured Auto-generated Transcripts

Video transcripts that are auto-generated by YouTube lack the punctuation and sentence structure generally maintained by the manually produced transcripts. In order to improve the quality of summaries produced from the transcripts, auto-generated transcript segments are first rewritten using the GPT-3 text generation functionality with a suitable prompt and parameters.

GPT-3 Response Latency

Since text summarisation is a computational expensive task, producing multiple summaries or summarising long-form text can greatly increase the latency of responses for the GPT-3 models through the OpenAI API. With each transcript segment being summarised independently, we were able to make requests to GPT-3 concurrently using the concurrent.futures python module, decreasing the overall response time.

Effective Key Term Extraction

To extract key terms, the TextRank algorithm constructs a word network by identifying the words that follow one another within the video transcript summaries. Results initially yielded by the spaCy TextRank component included words that were mentioned frequently but had little significance, as well as multiple lexical variations of the same lemma.

To address this, a custom spaCy lemmatization component was developed and added to the Fact Extraction pipeline to extract the unique lemmas associated with each extracted term. Additionally, a stop word removal component with custom stop words is utilised to ignore terms within the transcripts that are generally insignificant.

Suboptimal Transcript Summaries

Video transcripts that were initially summarised through GPT-3 failed to effectively convey the salient information within the transcript while maintaining correct sentence structure and punctuation. Upon further research on prompt engineering, we discovered there are specific input prompt formats that work particularly well and align better with different tasks. After multiple iterations of reformatting and rewording our prompts, we were able to use prompts that maximised our results.

Dynamically Displaying Summaries

Within our previous full-stack projects, we used multiple HTML pages within our application. Our idea with Summify was to have a single page application (SPA) that did not require reloading or rendering a whole different HTML page. However, since the only way we knew to display different information was to switch to a different page, we originally had the site redirect the user to a summary page when it was generated.

To address this, we researched Vue's dynamic rendering using `v-if` and `v-else` which render different HTML elements depending on variables in the JavaScript code. It took us a while to fully understand it but eventually we got a good grasp on dynamic rendering and it let us maintain the single page which would dynamically render the summaries once they were returned by the backend application.

Scaling Page Layout

The summaries and embedded YouTube video would look as expected on larger monitors that they were being developed on, but the page layout on smaller screens. We were initially working with flex containers but determined that they were inefficient for our purposes.

Our solution was to use `em` units rather than percentages and pixel values in CSS. We found those to work better when it came to page scaling as they did not break the page layout and kept things proportional regardless of the window size. We also applied an aspect-ratio CSS property to our iframe element to keep the video player at a consistent sizing.

5. Installation Guide

The Summify source code can be obtained by by cloning or forking the git repository at `https://gitlab.computing.dcu.ie/olojob2/2023-ca326-olojob2-majdap2`

5.1 Backend Flask Application

1. Install the necessary python libraries listed within `requirements.txt`
   Type **`pip install -r requirements.txt`** in a terminal window
2. The environment variables for the OpenAI API and Google Custom Search API secret keys must be included in a `.env` file of the `backend` directory
3. The environment variable for the custom Google Search Engine ID must also be included in the `.env` file
4. To start the Flask app, type **`flask run`** within the `backend` directory

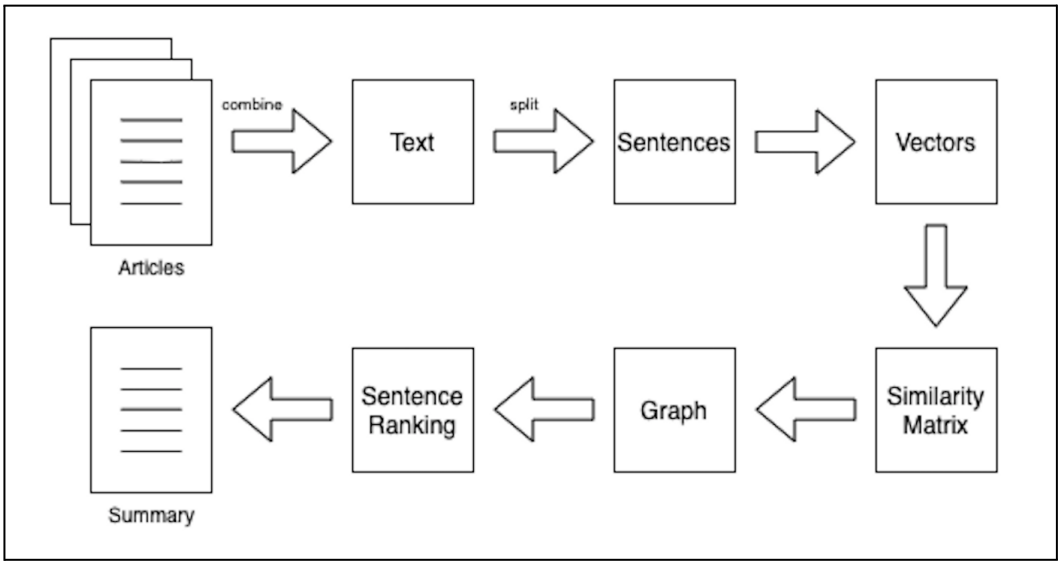
5.2 Fronted Quasar Application

1. Install Node (https://nodejs.org/en/download/)
2. Install Quasar CLI (https://quasar.dev/start/quasar-cli)
   In a terminal window, type **`npm install -g @quasar/cli`** to install quasar using Node package manager
3. In a terminal, navigate to the project folder
4. Type **`npm install`** to install the necessary packages
5. To start the Quasar app, type **`quasar dev`** within the `frontend` directory

6. Appendix

- TextRank Algorithm
    - https://ardaozmen.medium.com/extractive-text-summarization-using-textrank-algorithm-basic-logic-a6e73e9d60f0

Fig. 7 - TextRank Process Flowchart



- GPT-3 Models available through the OpenAI API
    - https://platform.openai.com/docs/models

Fig. 8 - GPT-3 Models and Maximum Tokens

LATEST MODEL	DESCRIPTION	MAX REQUEST	TRAINING DATA
`text-davinci-003`	Most capable GPT-3 model. Can do any task the other models can do, often with higher quality, longer output and better instruction-following. Also supports inserting completions within text.	4,000 tokens	Up to Jun 2021
`text-curie-001`	Very capable, but faster and lower cost than Davinci.	2,048 tokens	Up to Oct 2019
`text-babbage-001`	Capable of straightforward tasks, very fast, and lower cost.	2,048 tokens	Up to Oct 2019
`text-ada-001`	Capable of very simple tasks, usually the fastest model in the GPT-3 series, and lowest cost.	2,048 tokens	Up to Oct 2019