

# 1. INTRODUCTION

## 1.1 Project Introduction:

The objective of this project is to design a user-friendly and reliable YouTube [1] video and playlist downloader application. At its core, the application will feature a Graphical User Interface (GUI) that facilitates easy input of YouTube video or playlist URLs and selection of download preferences. PySimpleGUI,[2] a versatile library, will be employed to construct the GUI, ensuring an intuitive experience for users.

Upon entering the URL and selecting the desired options, the application will harness the power of the yt-dlp [3] library for seamless downloads. Noteworthy features encompass:

- The capacity to download individual videos or entire playlists
- A range of download quality choices
- Options for audio-only downloads or optimal video quality
- Downloading subtitles and thumbnail images
- Specifying the target folder for downloads
- Real-time download progress tracking within the GUI
- YouTube URL validation and error management

By offering the ability to effortlessly download content from the world's premier video platform, this application caters to various needs, such as archiving, offline viewing, and more. Developed using Python [4], the aim is to deliver a polished and effective tool while also welcoming feedback and implementing continuous improvements.

During the development process, the focus remains on usability, functionality, and stability. Customization and expandability will also be considered as the project evolves. As refinements and enhancements are made, the YouTube video downloader is poised to become an invaluable resource for users.

## 1.2 Motivation

The impetus behind creating this YouTube video downloader application lies in the desire for a streamlined and adaptable tool to download content from the world's largest video platform. Several factors contribute to the inspiration for this project:

1. YouTube's extensive collection of educational, informative, and entertaining content prompts users to revisit or save videos offline, necessitating a video downloader.
2. Current video downloaders often fall short in offering features that enable users to select download quality, resolution, audio format, and other preferences tailored to their requirements. This project aspires to fill that gap by providing customization options and an easy-to-use GUI.
3. Many command-line tools make downloading YouTube videos a cumbersome, multi-step process. Developing a graphical application can simplify and expedite the download experience for users.
4. A strong demand exists for reliable and feature-rich video downloaders that are easy to deploy and operate. This project endeavors to meet that need by crafting a straightforward yet comprehensive YouTube video downloader.
5. By incorporating user feedback, the application has the potential to evolve into a valuable utility for those looking to archive or preserve videos from YouTube.

In essence, the primary motivations include:

- The necessity for an efficient method to save content from YouTube
- A shortage of video downloaders offering customizable options
- Streamlining the often complicated download process
- Producing a stable and user-friendly video downloading application
- Iteratively enhancing the software based on user needs

The central objective is to create a YouTube video downloader that strikes the perfect balance between simplicity, functionality, and durability, ultimately providing value to its users.

### 1.3 Objectives:

The primary aim of this YouTube video downloader project is to develop a straightforward application enabling users to efficiently download content from YouTube according to their needs. The principal objectives include:

1. Crafting an intuitive GUI that allows users to input the YouTube video or playlist URL they want to download, as well as choose download quality, subtitles, destination folder, and monitor progress, among other options.
2. Leveraging a suitable library, such as yt-dlp, to download video content from YouTube in the highest possible quality based on user-specified preferences.
3. Validating entered YouTube URLs to ensure they correspond to legitimate YouTube videos or playlists, thus preventing errors during the download process.
4. Creating a modular application using functions and classes, making the code reusable, extendable, and less susceptible to errors.
5. Continuously incorporating new features based on user feedback and requirements. Potential enhancements could include pausing and resuming downloads, selecting multiple videos from playlists, customizing filename formats, and more.
6. Ensuring the application's robustness, stability, and minimal bug presence through rigorous testing and addressing unforeseen errors at various stages.
7. Delivering a user-friendly, functional, and easy-to-deploy YouTube video downloader application that demands minimal technical expertise for setup and use.

In conclusion, the key objectives revolve around the usability, functionality, stability, and extensibility of the final application for end-users. The ultimate goal is to offer users a convenient method for downloading YouTube content that aligns with their unique needs.

## 1.4 Scope:

The scope of this YouTube video downloader project primarily centers on the following aspects:

- Creating a Graphical User Interface (GUI) that enables users to effortlessly input the URL of the YouTube video or playlist they wish to download.
- Offering options within the GUI for users to select:
  - Download quality and resolution
  - Audio-only or optimal video downloads
  - Subtitle downloads
  - Download folder location
- Employing the yt-dlp library to efficiently download the specified YouTube content based on user selections.
- Verifying the entered YouTube URLs to ensure they correspond to actual videos or playlists on the platform.
- Displaying useful information about the video or playlist in the GUI, such as title, uploader name, duration, view count, upload date, and more.
- Presenting the download progress within the GUI by utilizing progress hooks from the yt-dlp library.
- Gracefully handling errors and providing meaningful error messages to users.
- Offering a popup window upon download completion, prompting users to download another video if desired.
- Structuring the application in a modular and organized manner using functions, classes, and constants.

The initial scope excludes features such as:

- Pausing and resuming downloads
- Customizing file naming options
- Limiting download speeds

Though not included in the initial version, these features may be considered for future enhancements based on user feedback.

The project's primary focus, within its current scope, is to deliver a straightforward yet effective solution for users to download YouTube videos and playlists tailored to their specific needs.

## 1.5 Hardware Requirements:

As this project primarily focuses on developing a software application, hardware requirements are minimal. The application is designed to run on any machine meeting the following specifications:

- **Processor:** Any modern Intel or AMD processor with a clock speed of 2 GHz or higher. The application does not consume significant CPU resources, so even lower-end processors should suffice.
- **RAM:** At least 4 GB of RAM is recommended. The application mainly uses RAM during the downloading process when information about the video is cached. Systems with lower RAM may encounter issues when downloading larger videos or playlists.
- **Storage:** Sufficient storage space to save the downloaded videos. The exact requirements will depend on the types and sizes of videos being downloaded. An SSD is recommended for faster I/O.
- **Display:** A screen with a minimum resolution of 720p is recommended. The GUI utilizes text and images, so any modern display should work sufficiently.
- **Operating System:** The application has been developed and tested on Windows 10 and 11. It should also function on recent Ubuntu and macOS versions with Python 3 installed.
- **Internet Connection:** A stable internet connection is necessary to download content from YouTube. Higher speeds will enable faster downloads.

The application has minimal hardware dependencies and is designed to be lightweight. As long as a system meets the above general requirements, it should be able to run the YouTube video downloader software without issues.

## 1.6 Software Requirements:

The software requirements for this project mainly include:

- Python 3 - The application is developed in Python 3 so it needs to be installed on the system.
- PySimpleGUI - The Graphical User Interface (GUI) of the application is built using the PySimpleGUI library for Python.
- yt-dlp - The application utilizes the yt-dlp library to download content from YouTube.
- ffmpeg [5 ] - Required to merge the downloaded video and audio files into a single file.

The software used for development and testing is:

- Python 3.10.0
- PySimpleGUI 4.60.5
- yt-dlp 2023.03.04
- ffmpeg 5.1
- Thorium AVX2 110.0.5481.178 [6]
- Visual Studio Code [7]



## 2. System Design and Requirement Specifications

### 2.1 System Overview:

The YouTube video downloader program comprises the following primary components:

- Graphical User Interface (GUI) - The GUI, built using PySimpleGUI, enables users to:
  - Enter the YouTube video or playlist URL
  - Select download quality options
  - Choose additional options such as subtitles, audio-only, folder path, and more
  - Monitor download progress
  - Gracefully handle error messages
- YouTube URL Validation - User-entered URLs are validated to ensure they correspond to a valid YouTube video or playlist, preventing potential errors during the download process.
- Information Extraction - Relevant information about the video, including title, duration, views, and more, is extracted and displayed in the GUI to inform users.
- Download Functions - Functions that manage the actual video or playlist download using the yt-dlp library are implemented and called when users click the "Download" button.
- Progress Hooks - Progress hooks from yt-dlp are leveraged to monitor and present download progress directly in the GUI.
- Modular Design - The program is organized into logical functions and classes to enhance reusability, extendibility, and maintainability.

- Error Handling - Potential errors are gracefully addressed, and meaningful messages are provided to users.
- Testing - Comprehensive testing is conducted to ensure the program's stability and robustness.

In summary, the essential components of this YouTube video downloader program include the GUI, URL validation, information extraction, tailored download functions, progress monitoring, and a modular design with proper error handling and testing. Collectively, these components strive to achieve the overarching objective of delivering an effective and customizable solution for downloading videos and playlists from YouTube.

## 2.2 Functional Requirements

The primary functional requirements of the YouTube video downloader program are:

1. The GUI must enable users to enter a YouTube video or playlist URL and click a "Proceed" button to validate the URL and advance to the next screen.
2. The program must validate the entered URL to ensure it corresponds to a valid YouTube video or playlist. Invalid URLs should prompt an appropriate error message.
3. If the URL is valid, relevant information about the video or playlist, such as title, duration, views, upload date, and more, must be extracted and displayed in the GUI.
4. The downloader GUI must present options for users to:
  - Choose from multiple download quality options
  - Opt for audio-only or optimal video downloads
  - Download subtitles
  - Select the download folder path
  - Download video thumbnail (for video-only option)
  - Download comments (for video-only option)
5. The program must download the specified YouTube video or playlist using the options selected by the user. Downloads should commence upon clicking the "Download" button.
6. The download process must be monitored, and progress should be displayed to the user in the GUI in near real-time.
7. Upon download completion, a popup window must inquire whether the user wants to download another video using the same or different options.
8. Any potential errors during the process must be gracefully addressed, and meaningful error messages should be provided to the user.
9. The program's source code must be modular, well-structured, and easy to comprehend.

10. Comprehensive testing must be conducted to ensure the program's stability, robustness, and reliability.

These requirements aim to encompass the critical functional aspects of the YouTube video downloader system, aligning with its overall objective.

## 2.3 Non-functional Requirements

1. The graphical user interface (GUI) should be intuitive, allowing users with basic computer skills to easily navigate and use the application, minimizing the need for training or technical expertise.
2. Upon clicking the "Download" button, the application should launch and initiate the download process within a 10-20 second timeframe.
3. Download speeds must be optimized, ensuring the program takes full advantage of the user's internet connection, providing efficient downloading.
4. Compatibility is essential, as the application should function on various operating systems, including Windows, Linux, and macOS.
5. To facilitate future extensions and maintenance, the program's source code must be thoroughly documented.
6. Responsiveness is crucial, as the GUI should remain functional and not freeze or become unresponsive during the download process.
7. The application must demonstrate high reliability, successfully downloading the specified YouTube content in the user-selected quality based on their preferences.
8. Memory utilization should be minimal, avoiding significant CPU or RAM usage during normal operation.
9. Unexpected errors should be handled gracefully, with the program recovering when possible and avoiding unexpected crashes.

These non-functional requirements outline the essential qualities that the YouTube Video Downloader should possess.

## 2.4 User Personas and Use Cases

Here is a draft of the "User Personas and Use Cases" subsection under the "System Analysis and Requirement Specification" section:

### User Personas and Use Cases

The primary users of this YouTube video downloader program can be categorized into two personas:

#### Persona 1: Casual User

- Possesses basic computer skills and limited technical expertise
- Occasionally wants to download a YouTube video or two for offline viewing
- Prefers a user-friendly interface with minimal configuration options

#### Common Use Cases:

- Enters a YouTube URL and clicks "Proceed"
- Selects the optimal video quality option from the dropdown menu
- Clicks "Download" to initiate video download
- Waits for the download to complete and views the saved file

#### Persona 2: Power User

- Boasts strong computer skills and some technical expertise
- Regularly wants to download multiple YouTube videos and entire playlists
- Prefers an interface with advanced download options

#### Common Use Cases:

- Enters a YouTube playlist URL and selects playlist range

- Chooses audio-only or specific video resolution from options
- Selects to download subtitles, thumbnail, and comments
- Specifies a custom download folder path
- Clicks "Download Playlist" to start downloading selected videos
- Monitors progress of individual video downloads
- Views downloaded files after completion

The above personas and use cases aim to capture the needs and workflow of potential users for the YouTube video downloader program. The requirements are specified to accommodate these personas and cover their common use case scenarios.

## 2.5 Data Requirements:

1. The application needs to store the YouTube video or playlist URL provided by the user. This storage is crucial for validating the URL, extracting content information, and downloading the content.
2. Temporarily store information extracted from YouTube about the video or playlist, such as the title, duration, views, upload date, and number of videos in the playlist while the program is running.
3. Preserve user-selected options like download quality, subtitles, audio-only, and folder path. This information helps configure the yt-dlp library to download content per user specifications.
4. Monitor and store download progress data for various purposes, including:
  - Displaying the download percentage completed in the GUI
  - Indicating the download speed and estimating the time of arrival (ETA) in the GUI
  - Identifying when the download is complete
5. Retain any error information encountered during program execution to present meaningful error messages to the user.

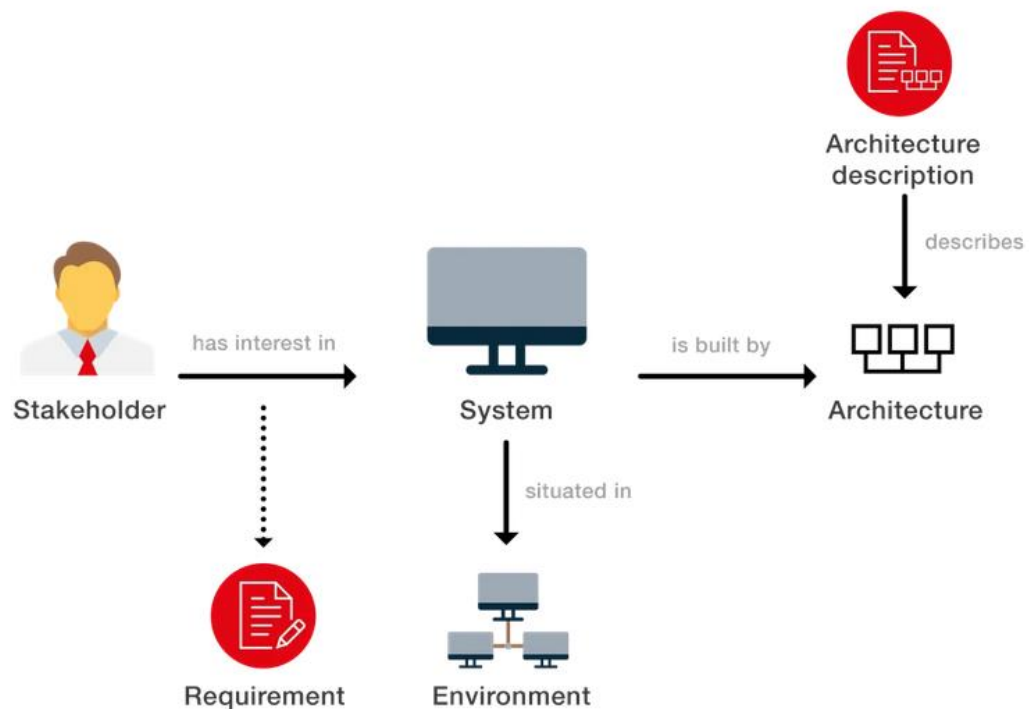
In summary, the critical data requirements are:

- YouTube URL
- Video/playlist metadata from YouTube
- User-selected download options
- Download progress details
- Error information

This data is temporary and stored in memory during the program's execution to perform required tasks, with no need for permanent storage.



## 2.6 System Architecture:



**Fig. 2.6 General System Architecture [8]**

### 1. User Interface Layer:

- The Graphical User Interface (GUI) is created with PySimpleGUI, featuring:
- Input fields for YouTube URL entry
- Download quality, subtitle, and audio-only selection options
- A "Download" button initiating the download process
- A progress bar for monitoring download progress
- Error message and notification displays

### 2. Business Logic Layer:

- This layer contains essential functions, such as:
- YouTube URL validation
- Information extraction
- Video and playlist downloading
- Progress hook implementation
- Error handling

### 3. External Library Layer:

- The yt-dlp library efficiently downloads YouTube content based on user-selected options.

The architecture follows a logical layering approach:

- Users interact with the GUI at the top layer.
- The middle layer consists of the business logic functions that process user input and configure yt-dlp.
- The lowest layer contains the yt-dlp library, responsible for executing download tasks.

This layered structure aims to separate concerns, promoting modularity, reusability, and maintainability of the codebase.

The system architecture's key components include:

- Graphical User Interface
- Business logic functions
- The yt-dlp external library

## 2.7 Interfaces and Integration

### 1. User Interface:

- The Graphical User Interface (GUI), designed with PySimpleGUI, allows users to:
  - Input the YouTube URL
  - Choose download options
  - Track download progress
  - View error messages

### 2. Integration with yt-dlp Library:

- The program leverages the yt-dlp library for efficient YouTube content downloading. Integration with yt-dlp includes:
  - Configuring options, such as download quality, subtitle inclusion, filename format, and progress hooks
  - Initiating the actual download process via the yt-dlp API's download() method

### 3. Integration with Operating System:

- The application integrates with the underlying operating system to perform tasks like:
  - Opening folders for download location specification
  - Verifying the existence of the download folder
  - Saving downloaded video and audio files
  - Displaying notifications and popups to the user

In summary, the essential interfaces for the YouTube Video Downloader program are:

- Graphical User Interface
- Integration with the yt-dlp library
- Integration with operating system features

## 2.8 Modules:

### 2.8.1 yt-dlp

yt-dlp is a command-line program to download videos from more than 1000 websites. It is a continuation of the legacy youtube-dl project and adds many improvements and new features. Some key points about yt-dlp:

- It can download videos and playlists from YouTube and thousands of other sites like Facebook, Vimeo, Twitch and more.
- It supports downloading in high video qualities up to 8K and also allows downloading only the audio.
- It can download video thumbnails, subtitles and descriptions. It can also download the video comments.
- It has a very flexible settings system to customize the downloads based on a user's needs.
- It has a Python API that allows it to be integrated into other applications.

In the YouTube video downloader code, yt-dlp is used through its Python API as follows:

- A `yt-dlp.YoutubeDL()` object is instantiated to represent a downloader.
- Options are configured based on the user selections like download quality, subtitles option, filename, progress hooks, etc.
- The `yt-dlp.YoutubeDL.download()` method is called to actually download the specified YouTube video or playlist. It takes the YouTube URL as an argument.
- A progress hook function is passed to yt-dlp to monitor and extract download progress information like percentage, speed and ETA.

- This progress information is then used to update the GUI in near real-time to show the user the download status.
- Once the download is complete, the progress hook function is informed and the GUI is accordingly updated.

So in summary, the yt-dlp module is used through its Python API to configure the download options based on user input, start the actual downloads and monitor the progress - all from within the YouTube video downloader Python code.

### **2.8.2 json**

The JSON (JavaScript Object Notation) format is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language.

The json module in Python provides an API for parsing, generating and manipulating JSON data. It allows us to convert between JSON and Python objects. This is useful for serializing data to store it or transmit it to other applications.

Some key facts about the json module:

- It allows conversion between JSON and Python data types like lists, dicts, strings, numbers, booleans, etc.
- It has two main functions - `json.loads()` and `json.dumps()`.

`json.loads()` - Converts a JSON string to a Python object. It takes a JSON string and parses it to create the corresponding Python object.

`json.dumps()` - Converts a Python object to a JSON string. It takes a Python object and converts it to a JSON formatted string.

- It can parse JSON strings into nested structures like lists within lists and dicts within dicts.

- It raises a `JSONDecodeError` when the given JSON string is invalid or malformed.
- It can also load JSON data from files using `json.load()` and write JSON data to files using `json.dump()`.

In the code, I used the `json` module to:

1) Extract information from the YouTube video URL that was provided. I called `validate_youtube_url()` which used `yt_dlp` to extract info from the YouTube URL. If no error occurred, I saved the video data in `json_data`:

```
```python
error, json_data = validate_youtube_url(url)
```
```

2) In the `video_screen()` and `playlist_screen()` functions, I used the `json_data` to populate the GUI with information like the video title, uploader, duration, etc. This data was extracted from the JSON response.

3) In the `progress_hook()` function, I parsed the JSON progress data sent by `yt_dlp` to update the download progress in the GUI.

So in summary, I used the `json` module to:

- Extract relevant video information from the YouTube JSON response
- Parse the JSON data to populate the GUI fields
- Parse the JSON progress data to show download progress

### 2.8.3 PySimpleGUI

I apologize, here is an expanded version:

PySimpleGUI is an excellent Python module for creating simple GUI applications with very little code. It offers a high-level wrapper around TKinter, making GUI creation fast and easy.

In my YouTube video downloader code, PySimpleGUI simplified and accelerated the GUI development process in several major ways:

Window Layouts - I was able to dynamically layout my GUI windows programmatically using columns, frames and other container elements. This allowed me to quickly design forms to suit my specific needs. For the initial window, video screen and playlist screen, I used different layouts tailored to each use case.

Events and Values - PySimpleGUI handles all the event management and input parsing behind the scenes. I simply read events and values from a single window.Read() call. This returned everything I needed to trigger the relevant logic for that input.

Output Area - The cprint() function provided an easy way to print formatted text and progress updates to the GUI's output area. I used this to keep the user informed as videos downloaded.

Popups - Simple functions like Popup allowed me to show error messages or confirmation dialogs when needed. This created a more polished user experience.

Window Management - The Window class made it effortless to open new windows, close existing ones and refresh windows to update elements in real time. This dynamic window switching suited the needs of my application.

Themes - Being able to apply themes to my GUIs with a single line of code gave me the flexibility to quickly experiment with different visual styles.

In conclusion, PySimpleGUI allowed me to focus my mental energy on the core logic of downloading and processing YouTube videos. The GUI portion of the code, which tends to be complex and tedious in many GUI libraries, became remarkably simple and compact thanks to PySimpleGUI. This let me develop the application much faster than I could have with a lower-level GUI toolkit.

In summary, PySimpleGUI:

- Greatly simplified and accelerated GUI development
- Let me focus on core logic rather than GUI details
- Provided tools that matched the exact needs of my application
- Resulted in a compact, Pythonic GUI codebase

## 2.8.4 os

The `os` module in Python is a built-in library that provides a convenient way to interact with the underlying operating system. It offers a collection of functions to perform common tasks, such as file and directory manipulation, environment variable management, and process control, among others. By using the `os` module, developers can write cross-platform code that works on various operating systems, including Windows, macOS, and Linux.

### File and Directory Manipulation

One of the primary uses of the `os` module is to manage files and directories. It offers a range of functions to create, delete, and modify files and directories, as well as to navigate and inspect the file system.

Some commonly used file and directory manipulation functions include:

- `os.mkdir(path)`: Creates a new directory at the specified path.
- `os.rmdir(path)`: Removes the specified directory. Note that the directory must be empty.
- `os.makedirs(path)`: Creates a directory hierarchy, including all intermediate-level directories, at the specified path.
- `os.removedirs(path)`: Removes a directory hierarchy, as long as the directories are empty.
- `os.rename(src, dst)`: Renames a file or directory from `src` (source) to `dst` (destination).
- `os.listdir(path)`: Lists the files and directories in the specified directory.
- `os.getcwd()`: Returns the current working directory.
- `os.chdir(path)`: Changes the current working directory to the specified path.
- `os.path.exists(path)`: Checks if the specified path exists.



- `os.path.isfile(path)`: Determines if the specified path is a file.
- `os.path.isdir(path)`: Determines if the specified path is a directory.

## Environment Variables

The `os` module also allows access to environment variables, which are key-value pairs containing information about the system and its configuration. Environment variables can be used to store configuration settings, paths to executables, and other data that varies between different environments.

Some functions for working with environment variables are:

- `os.environ`: A dictionary-like object representing the environment variables.
- `os.environ.get(key)`: Retrieves the value of the specified environment variable, or `None` if it doesn't exist.
- `os.environ[key]`: Accesses the value of the specified environment variable, raising a `KeyError` if it doesn't exist.
- `os.environ.setdefault(key, value)`: Sets the value of the specified environment variable if it doesn't already exist.

## Process Control

The `os` module provides functions to interact with, create, and manage processes. A process is an instance of a running program, and Python scripts can use the `os` module to control or communicate with other processes on the system.

Some process control functions include:

- `os.system(command)`: Executes the specified command in a subshell and returns the exit code.
- `os.spawn*()`: A family of functions to create new processes, with different options for managing input and output, and controlling the parent and child processes.
- `os.fork()`: Creates a new child process by duplicating the current process (Unix-based systems only).

- `os.exec*()`: A family of functions to replace the current process with a new one, running a specified command or program.

## Path Manipulation

The `os.path` submodule offers a variety of functions to work with file system paths. These functions make it easier to manipulate path strings, find information about files, and maintain platform independence.

Some commonly used `os.path` functions are:

- `os.path.join(*paths)`: Joins multiple path components into a single path.
- `os.path.split(path)`: Splits a path into a tuple containing the head (directory) and tail (file or last directory).
- `os.path.splitext(path)`: Splits a path into a tuple containing the root (path without extension) and extension.
- `os.path.abspath(path)`: Returns the absolute version of a path, resolving any relative references.
- `os.path.dirname(path)`: Returns the directory component of a path.
- `os.path.basename(path)`: Returns the last part of a path, either a file or directory name.

In conclusion, the `os` module in Python is a versatile and powerful library that simplifies interaction with the operating system, enabling developers to write more portable and efficient code. By understanding and utilizing the functions provided by the `os` module, programmers can perform file and directory operations, manage environment variables, control processes, and manipulate paths with ease.

## 2.8.5 datetime

The `datetime` module in Python is a built-in library that provides a comprehensive set of tools for working with dates, times, and intervals. It offers various classes for manipulating and formatting date and time values, as well as for performing arithmetic operations and comparisons. The `datetime` module is essential for handling time-based data, scheduling tasks, and tracking durations in a wide range of applications.

## \*Key Classes in the `datetime` Module\*

The `datetime` module contains several classes, each designed to represent and manipulate different aspects of date and time information:

1. `datetime.date`: Represents a date (year, month, and day) without time information.
2. `datetime.time`: Represents time (hour, minute, second, microsecond) without date information.
3. `datetime.datetime`: Represents both date and time information.
4. `datetime.timedelta`: Represents a duration or the difference between two dates or times.
5. `datetime.tzinfo`: An abstract base class for defining time zone information.
6. `datetime.timezone`: A concrete implementation of `tzinfo` for representing time zones with a fixed offset from UTC.

## \*Working with Dates\*

The `date` class provides various methods and attributes for working with dates:

- `date.today()`: Returns the current local date.
- `date(year, month, day)`: Creates a new date object with the specified year, month, and day.
- `date.fromtimestamp(timestamp)`: Creates a date object from a POSIX timestamp.
- `date.fromordinal(ordinal)`: Creates a date object from a Gregorian ordinal, where January 1 of year 1 has the ordinal 1.
- `date.year`, `date.month`, and `date.day`: Attributes that hold the year, month, and day values of a date object, respectively.
- `date.weekday()`: Returns the day of the week as an integer, where Monday is 0 and Sunday is 6.
- `date.isoweekday()`: Returns the day of the week as an integer, where Monday is 1 and Sunday is 7.
- `date.isoformat()`: Returns a string representing the date in the ISO 8601 format (YYYY-MM-DD).

## \*Working with Times\*

The ``time`` class offers methods and attributes for handling time values:

- ``time(hour, minute, second, microsecond, tzinfo)``: Creates a new time object with the specified values.
- ``time.hour``, ``time.minute``, ``time.second``, ``time.microsecond``, and ``time.tzinfo``: Attributes that hold the corresponding time components of a time object.
- ``time.isoformat()``: Returns a string representing the time in the ISO 8601 format (HH:MM:SS.mmmmmmm).

### **\*Working with Date and Time\***

The ``datetime`` class combines both date and time information and provides several useful methods and attributes:

- ``datetime.now(tz=None)``: Returns the current local date and time. If ``tz`` is provided, returns the current date and time in the specified time zone.
- ``datetime.utcnow()``: Returns the current UTC date and time.
- ``datetime.fromtimestamp(timestamp, tz=None)``: Creates a datetime object from a POSIX timestamp. If ``tz`` is provided, creates a timezone-aware datetime object.
- ``datetime.combine(date, time)``: Combines a date object and a time object to create a datetime object.
- ``datetime.date()``, ``datetime.time()``, and ``datetime.timez()``: Returns the date, time, and timezone-aware time components of a datetime object, respectively.

### **\*Working with Time Deltas\***

The ``timedelta`` class is used to represent durations or differences between two dates or times:

- ``timedelta(days, seconds, microseconds)``: Creates a new timedelta object with the specified values.
- Arithmetic operations, such as addition, subtraction, and multiplication, can be performed with timedelta objects.
- ``timedelta.total_seconds()``: Returns the total number of seconds in a timedelta object.

In summary, the ``datetime`` module in Python is a powerful and versatile library for handling date, time, and interval values in various formats and time zones. By understanding and utilizing the classes and methods provided by the ``datetime`` module, developers can effectively manage time-based data, schedule tasks, and track durations in their applications. The ``datetime`` module is essential for working with temporal information and is a fundamental part of the Python standard library.

## 3. Project Management

### 3.1 Project Planning:



Fig 3.1 Project Planning [9]

Embarking on the creation of the YouTube video downloader program, I adopted an iterative and incremental strategy. This methodical progression involved a sequence of clearly defined milestones, each culminating in a functional prototype. By adhering to a well-organized and structured development process, I was able to refine the design and architecture systematically, ultimately yielding a practical and efficient application.

Initially, I concentrated on pinpointing the essential features of the program, which encompassed the following:

- The capacity to download individual videos or entire playlists from any YouTube URL.
- A user-friendly graphical interface that streamlines input and selection of options.
- Displaying download progress and reporting errors in a manner that is straightforward and comprehensible.

With these requirements in mind, I established a timeline consisting of eight key development milestones:

1. Design a rudimentary GUI layout for inputting a single video URL.
2. Validate the submitted YouTube URL and extract the relevant video information.
3. Develop a GUI layout for choosing single video download options.
4. Incorporate the functionality to download a single YouTube video.
5. Create a GUI layout for selecting playlist download options.
6. Integrate the capability to download entire YouTube playlists.
7. Introduce additional features, such as subtitle downloading and an audio-only option.
8. Embed progress indication and error reporting mechanisms.

Each milestone culminated in a working prototype that underwent rigorous testing before moving on to the next phase. I identified libraries like PySimpleGUI and yt\_dlp early in the project to facilitate the development of the GUI and YouTube-related functionality, respectively.

I designed the application's architecture with an emphasis on extensibility, reusability, modularity, and maintainability. I devised functions, classes, and data structures to accommodate expected expansions and variations in the future. Moreover, I incorporated suitable exception handling measures at every stage of development.

To ensure the reliability of each prototype, I created exhaustive test cases that scrutinized edge conditions, invalid inputs, and unexpected errors. By conducting these tests, I aimed to identify weaknesses, detect failures, and guarantee robustness before proceeding to the next milestone.

In summary, my incremental development approach allowed me to build the YouTube video downloader feature by feature. I only incorporated new elements once the preceding components were functioning correctly. This organized process enabled me to manage complexity, minimize risks, and ultimately produce an application architecture that aligned with my initial requirements.

### 3.2 Risk Management:

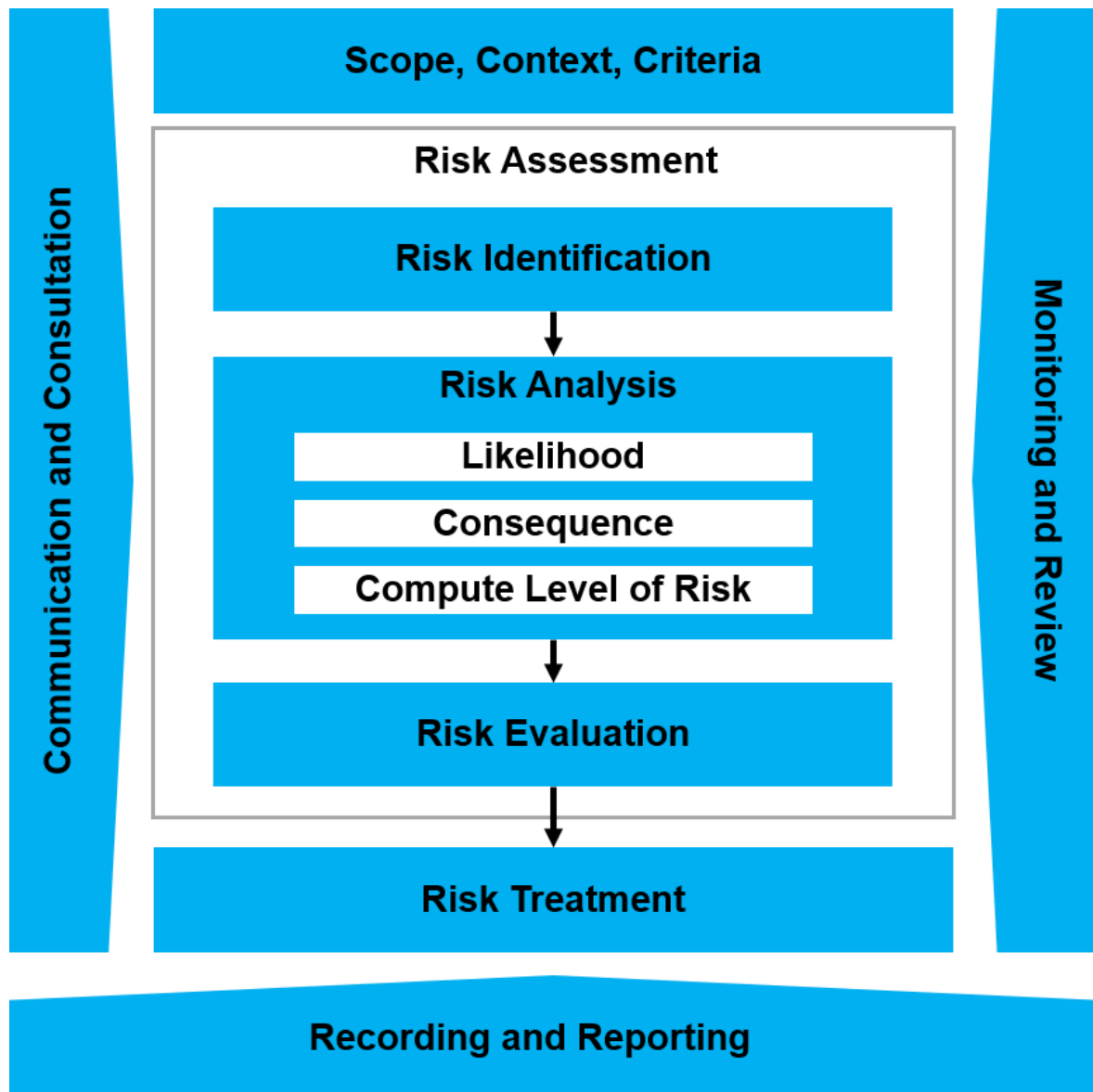


Fig. 3.2 Risk Management [10]

#### 3.2.1 Identifying Risks:

In any software project, it is important to identify potential risks early to develop appropriate mitigation strategies. For the YouTube video downloader application, the following steps can be taken to identify the key risks:



1. Review the functional and non-functional requirements: Go through each requirement in detail and think of possible uncertainties, issues or failures that could arise when trying to implement that requirement. This helps uncover technical, operational and performance risks.
2. Analyze dependencies: Evaluate all external dependencies like APIs, libraries, frameworks, hardware etc. Determine what could go wrong with each dependency and how serious the impact would be.
3. Conduct stakeholder interviews: Speak to key stakeholders like users, managers, developers to understand their concerns. This can surface political, organizational and communication risks not previously considered.
4. Review the project plan: Evaluate activities, milestones, schedules and resources. Determine what uncertainties exist and how sensitive the plan is to those uncertainties. This reveals schedule, cost and resource risks.
5. Conduct a brainstorming session: Involve the project team and other relevant parties to brainstorm potential risks together in an open and transparent environment. New ideas are more likely to surface this way.
6. Review similar past projects: Examine lessons learned from previous, related projects. Identify commonly encountered risks and determine their applicability to the current project.
7. Use a risk identification checklist: Consult a standard risk identification checklist tailored to your type of project to ensure no major categories of risks are overlooked.

For the YouTube downloader, potential risks identified may include:

- Incomplete requirements
- Dependencies on external APIs
- Complexity of the GUI
- Integration of multiple components
- Delays due to bugs or testing
- Resource constraints

In conclusion, using multiple methods and gathering inputs from different perspectives can help ensure a comprehensive list of risks is identified to set the project up for proper risk management.

### **3.2.2 Analyzing Risks:**

Any software project involves various risks that need to be identified, analyzed and mitigated where possible. The following risks have been identified for the YouTube video downloader program:

Technical risks:

The program relies on external APIs and libraries like YouTube Data API and yt\_dlp to function. Any changes to these APIs or breakages could potentially break the program. Regular testing and version updates of the libraries will be required to mitigate this risk.

API rate limit issues could cause interruptions in the program's functioning if too many requests are made in a short time. Storing an API key and implementing request throttling can help reduce this risk.

User interface risks:

The GUI design could be improved to make it more intuitive and user-friendly. Adding more options, validation and feedback messages could reduce the chances of user error. Regular user testing and interface updates will be required.

Legal and compliance risks:

Downloading and storing copyrighted content could pose legal risks. The program will come with disclaimers stating that it is only intended for legal uses as per terms of service of the video platforms.

System and security risks:

Any software is susceptible to system errors, crashes and security vulnerabilities. Regular testing, monitoring and updates will help reduce these risks. Security best practices like least privilege access and input sanitization will be implemented.

In summary, the primary risks identified relate to the program's dependencies on external APIs, interfaces and systems. Mitigation strategies focusing on testing, monitoring, version updates and user education will be adopted and improved throughout the project lifecycle to manage these risks effectively. Regular risk reviews will also be performed to identify any new risks and updates to mitigation strategies.

### **3.2.3 Risk Mitigation Planning**

The following mitigation strategies will be implemented to manage the risks identified for the YouTube video downloader program:

Technical risks:

- Regular testing of the program will be performed after library/API updates to ensure functionality is maintained.
- An API key for the YouTube Data API will be stored and request throttling will be implemented to avoid rate limit issues.
- The program code will be reviewed regularly for potential bugs, crashes and security vulnerabilities. Updates will be released as needed.

User interface risks:

- User testing of the interface will be conducted on an ongoing basis to identify areas for improvement.
- Additional input validation, feedback messages and error handling will be added where needed.

- The interface design and options will be reviewed and updated to make navigation more intuitive.

#### Legal and compliance risks:

- Clear disclaimers will be provided stating that the program is intended only for legal uses as authorized by the terms of service of YouTube and other video platforms.
- The program will not be marketed or distributed for illegal purposes.

#### System and security risks:

- Least privilege access principles will be followed to limit potential damage from system errors or security breaches.
- Input sanitization and output encoding techniques will be implemented to mitigate risks of injection attacks.
- Regular system monitoring and program logging will be used to detect anomalies and issues promptly.
- Timely software updates and patching will be performed to address newly discovered vulnerabilities.

These mitigation strategies will be constantly evaluated and improved upon throughout the project life cycle. Regular risk reviews and testing will also be conducted to ensure the strategies remain appropriate and effective in managing the identified risks for the YouTube video downloader program.

### **3.2.4 Risk Mitigation Implementation**

Any software project involves potential risks that need to be identified and appropriately mitigated to ensure successful completion. The YouTube video downloader program is no exception, with several risks that require careful mitigation strategies.

The key risks identified for this project along with mitigation implementations are:

Dependency issues - The program relies on external libraries like yt\_dlp and PySimpleGUI. There is a risk of compatibility issues, bugs or failures in these dependencies impacting the application.

To mitigate this risk, the following measures have been taken:

- Checking for correct installation and latest versions of dependencies before running the program.
- Implementing 'try/except' blocks around sections that use dependencies to catch any errors at runtime.
- Writing unit tests for functions that utilize dependencies to ensure robustness.
- Monitoring dependency repositories for bug fixes and updating the program accordingly.

Network issues - The program requires an active Internet connection to download videos from YouTube. There is a risk of network failures or instability impacting downloads.

The following measures help reduce this risk:

- Checking for an active Internet connection before attempting downloads.
- Implementing retries and timeout mechanisms in the download functions to handle transient network issues.
- Displaying informative error messages to users in case of network failures, along with suggestions to try again once the network is restored.

- Saving partial downloads and resuming from the previous point in case of interruptions.

Bugs and errors - As with any software, there is an inherent risk of bugs, errors and crashes impacting users.

The following practices are employed to mitigate this risk:

- Writing unit and integration tests to cover key functionality and edge cases.
- Performing thorough testing of the program before release.
- Implementing a structured logging system to record error information for debugging.
- Releasing updates in a timely manner to fix reported bugs.

### **3.2.5 Risk Monitoring and Control**

To ensure the effectiveness of risk mitigation strategies over time, the following monitoring and control measures will be implemented for the YouTube video downloader program:

Regular Testing:

Automated unit and integration tests will run after each change to the codebase. This will help detect any regressions or breaks early. Manual testing will also be performed periodically to test new features and functionality.

Library/API Monitoring:

The libraries and APIs used by the program will be regularly monitored for updates, bug fixes and security vulnerabilities. Updates will be applied in a timely manner and tested thoroughly before release.

#### Usage and Error Monitoring:

Program logs capturing user inputs, exceptions, crashes and other events will be reviewed on a weekly basis. This will help identify issues, errors and abnormal usage patterns. The logs can then be used to further improve the software.

#### User Feedback:

A formal process will be established to collect user feedback on an ongoing basis through surveys, feedback forms, social media and direct communication. This feedback will be reviewed regularly and used to continuously improve the program.

#### Regular Risk Reviews:

The effectiveness of risk mitigation strategies will be formally reviewed on a quarterly basis. This will involve revisiting the identified risks, evaluating current controls and identifying any new or emerging risks. The strategies will then be updated as needed.

#### Reporting:

Regular status reports summarizing test results, issues identified, actions taken and upcoming work will be produced and shared with stakeholders. This reporting will provide transparency and accountability for risk management.

By regularly monitoring key aspects of the program and continually evaluating the risks and controls, the risk management process will become embedded in the development lifecycle. This will help to proactively manage risks and keep the YouTube video downloader program secure and reliable over time.

### **3.2.6 Risk reporting:**

Risk reporting is an essential part of the risk management process. It ensures that all relevant stakeholders are aware of potential risks and the actions being taken to mitigate those risks. The following outlines our approach to risk reporting for the YouTube Video Downloader project.

Formal risk reports will be generated on a monthly basis during the development and deployment of the project. These reports will be submitted to the university project sponsor and management team. The risk reports will include:

- Identification of all known risks - The risks will be listed along with a brief description. Any changes to previously identified risks will be noted.
- Assessment of likelihood and impact - Each risk will be assigned a rating for its likelihood of occurrence and expected impact on the project. This will allow for prioritizing risks.
- Proposed and existing risk responses - For each risk, the report will outline any actions already taken to reduce, accept, or transfer the risk. Any new proposed responses will also be documented.
- Status of existing risks - The report will provide an updated status on any previously identified risks, noting if the risk has been resolved, mitigated to an acceptable level, or is still ongoing.
- Emerging risks - Any new risks identified since the previous report will be documented along with initial response recommendations.

The goal of the risk reports is to provide transparency around the project's risks and the team's approach to managing those risks.

### **3.2.7 Lessons Learned:**



Upon completion of the YouTube Video Downloader project, it is important to document the lessons learned and best practices identified. This will allow for continuous improvement in future projects. The following outlines the main lessons learned regarding risk management for this project:

- The importance of early risk identification - The development team realized that several risks emerged late in the project that could have been identified earlier through more thorough upfront risk analysis. For future projects, a more comprehensive initial risk assessment will be performed.
- The value of frequent risk reporting - Producing monthly risk reports proved very valuable in keeping stakeholders informed and the development team on track. For future projects, risk reporting will be instituted from the start and performed at least biweekly.
- Benefits of assigning risk owners - Assigning specific individuals as "risk owners" to be responsible for monitoring and updating risks increased accountability and follow-through on risk responses. Risk owners will be designated from the beginning of all future projects.
- Worthwhile risk responses take time - Some risks required multiple iterations of responses before an effective solution was found. The development team learned to be patient and persistent with high-impact risks.
- Managing risks proactively reduces crisis management - The majority of risks that were managed proactively through planned responses did not become major issues. For high-likelihood risks, proactive response will be the default approach going forward.
- Continuous risk identification is critical - New risks emerged throughout the project lifecycle, reinforcing the need for an ongoing process of risk analysis and identification. Future projects will institutionalize periodic risk re-assessments.

These lessons highlight areas where the effectiveness of risk management on the project can be improved. Taking these lessons into account and putting the identified best practices into action should lead to more successful risk management for projects going forward.

### 3.3 Cost Estimation:

The cost estimation for the YouTube video downloader program has been done based on the functional and technical requirements as well as the current scope of the project. Several factors have been considered to arrive at the total estimated cost.

#### Software and Hardware Costs:

The software requirements include Python, PySimpleGUI, ffmpeg and yt\_dlp - which are open source. A computer costing ₹30,000 will be needed for development.

#### Development and Testing Costs:

The major cost driver will be the development time required to implement the functionality as per requirements. Costs will also include time spent on testing and bug fixing during development. An average development effort of 150 man hours has been estimated. Testing efforts are estimated at 30 man hours.

#### Internet Costs:

A broadband internet connection costing ₹500 per month will be required during the development and testing phase.

#### Total Estimated Cost:

- Hardware Costs - ₹30,000
- Development Cost (@₹400/hour) - ₹60,000
- Testing Cost (@₹400/hour) - ₹12,000
- Internet Costs (@₹500/month for 3 months) - ₹1,500

Total Cost - ₹1,03,500 (~₹1 lakhs)

The cost estimation has been done based on certain assumptions and is intended to provide an initial guide. The estimates are subject to change based on actual development time/effort involved, hardware requirements, number of bugs encountered, etc.

## 4. Input Design

### 4.1 Input Specification:

The main input methods for the YouTube Video Downloader tool include the following:

1. YouTube video/playlist URL - This is the primary input required from the user. The user enters the URL of either a YouTube video or playlist in the provided input field. The tool then processes the URL to extract the relevant video data.

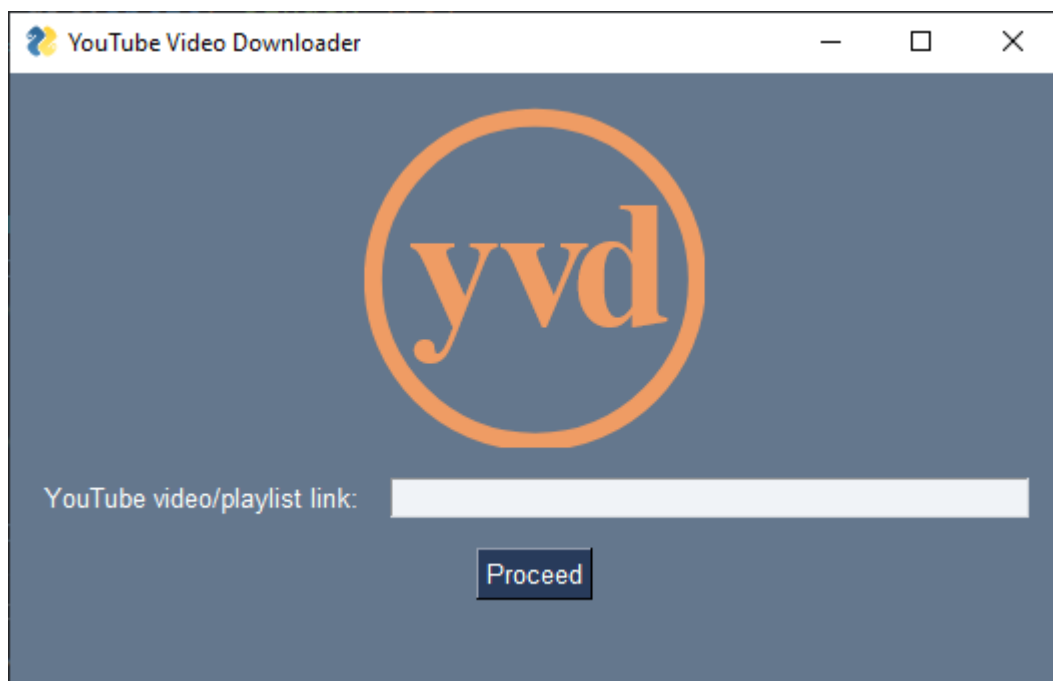


Fig. 4.1.1 Initial Input Screen

2. Video quality selection - The user is presented with a dropdown list of available video qualities for the video or all videos in the playlist. The user selects the desired video quality from the list.

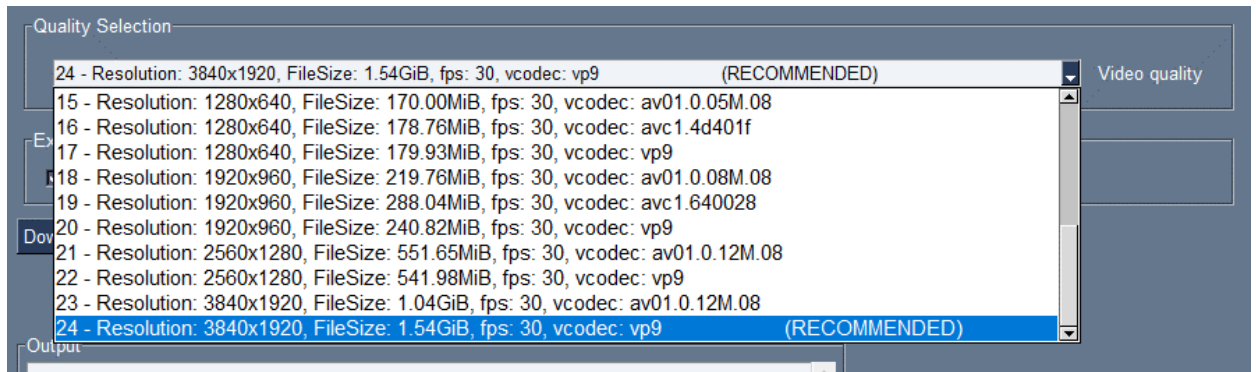


Fig. 4.1.2 Quality Selection

3. Extra options - The user can select various extra options like downloading subtitles, thumbnails, audio only, etc. via checkboxes.

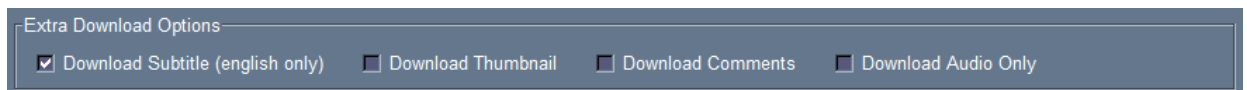


Fig. 4.1.3.1 Extra Options Video

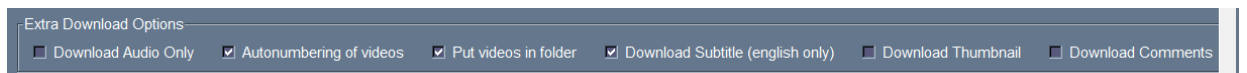


Fig. 4.1.3.2 Extra Option Playlist

4. Folder path - The user inputs or browses to select the download folder location.

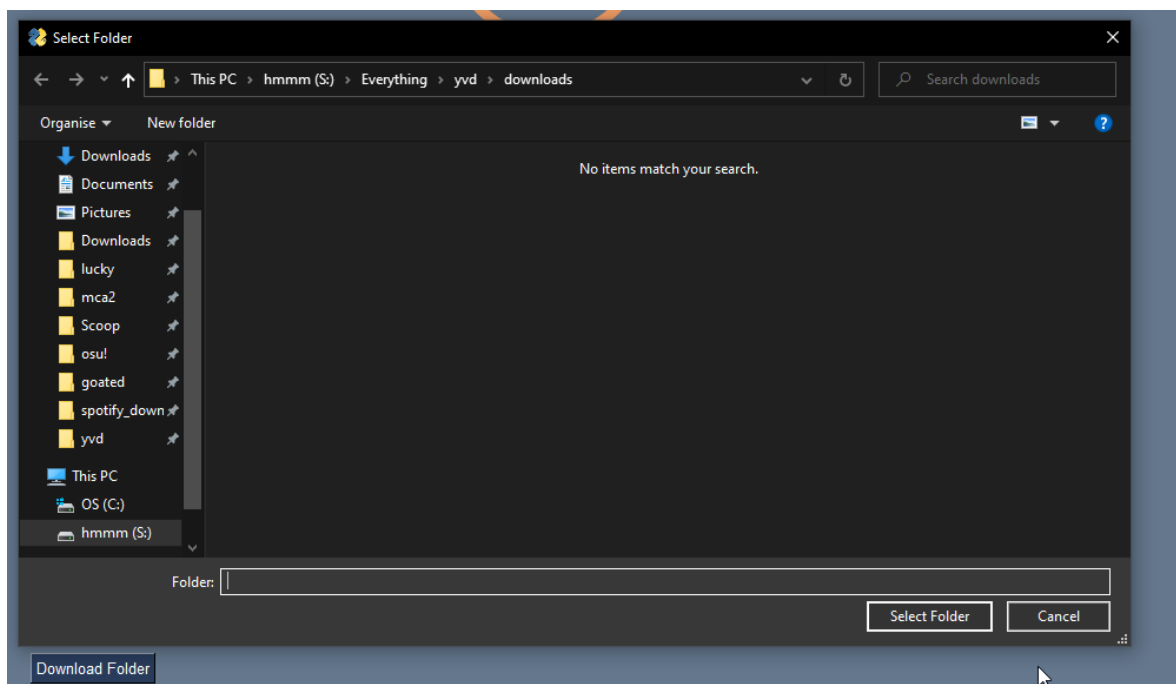


Fig. 4.1.4 Folder Selection

5. Playlist range - For playlists, the user can specify a start and end index to download a range of videos from the playlist.


A screenshot of a software interface titled "Playlist Range Options". It contains two horizontal input fields. The first field on the left contains the number "1", and the second field on the right contains the number "11". The fields are light gray with dark borders.

Fig. 4.1.5 Playlist Range

The tool's input design aims to minimize the inputs required from the user while still providing enough options to customize the download based on individual needs and preferences. A help or instructions section could be provided to explain the various input options to users who are unfamiliar with the tool.

Overall, the key input parameters - the YouTube URL, video quality selection, and download folder location - provide the minimum necessary inputs to accomplish the tool's main function of downloading YouTube videos.

## 4.2 Input Validation:

Proper validation of user inputs is critical to ensure the correct functioning of the YouTube Video Downloader tool. The following input validation techniques are employed:

### YouTube URL Validation

When the user inputs a YouTube URL, the tool uses the yt\_dlp library to extract information about the URL. If the URL is invalid or does not point to a YouTube video or playlist, a YouTube API error will be returned. The tool catches this error and shows an error message to the user, prompting them to input a valid YouTube URL.

```
url = values['url'].strip()

url = url.split('&index')[0]
temp = url.split('=')
url = temp[0] if len(temp) == 1 else temp[1]
url = url.split('&')[0]

if len(url) > 11 and 'channel' not in url:
    url = "https://www.youtube.com/playlist?list=" + url
print(url)

error, json_data = validate_youtube_url(url)
print(json_data)
```

Fig. 4.2.1 URL validation

### Video Quality Validation

The tool generates a dropdown list of available video qualities based on the formats available for the given video. If the user selects a value that is not in the provided list, an error message is shown. This ensures that only valid video qualities that can actually be downloaded are selected.

### Playlist Range Validation

For playlist inputs, the user provides a start and end index to specify a range of videos to download. The tool validates that the provided start and end indexes are:

- Positive integers
- Within the bounds of the number of videos in the playlist
- With the end index greater than or equal to the start index

If any of these conditions fail, an appropriate error message is shown.

#### Folder Path Validation

```
if folder_path:
    return folder_path + '/' + tpl
else:
    f = os.getcwd()
    if not os.path.exists(f + '/downloads'):
        os.mkdir(f + '/downloads')
    return f + '/downloads/' + tpl
```

Fig. 4.2.2 Folder validation/creation

When the user inputs or browses to select a download folder, the tool checks that the provided folder path actually exists on the system. If the folder path is invalid, an error message is shown.

Overall, input validation is implemented for all key inputs - the URL, video quality, playlist range, and folder path. If any validation checks fail, user-friendly error messages are shown to assist the user in providing correct inputs. This approach aims to catch invalid inputs early, provide helpful feedback, and ensure the reliable functioning of the tool.



### 4.3 Input Interface:

The user interface for accepting inputs from the user has been designed with the following considerations in mind:

- **Simplicity** - Only the minimum necessary inputs have been requested from the user. This includes the essential inputs of the YouTube URL, video quality selection, and download folder location.
- **Clarity** - The purpose and function of each input field and option has been clearly labelled to ensure users understand what to fill in each box.
- **Flexibility** - Additional optional inputs have been provided (e.g. subtitles, playlist range) to give users flexibility in customizing their download while keeping the main workflow simple.
- **Familiar elements** - Where possible, standard interface elements like text boxes, drop-down lists, checkboxes and file browse buttons have been used for inputs.
- **Context-sensitive help** - An instruction or help section can be provided to explain the various input options, especially for first-time or infrequent users.

The design of the input interface aims to adhere to these principles:

#### YouTube URL Input

A standard text input box has been provided, clearly labelled "YouTube video/playlist link". Users simply paste or type their desired YouTube URL into this field.

#### Video Quality Selection

A drop-down list populated with the available quality options for the given video/playlist has been used. This ensures users select only valid options, while providing a simple selection interface.

## Optional Inputs

Checkboxes have been used to enable users to optionally select inputs like subtitles, comments, thumbnails, etc. Checkboxes provide a simple "on/off" input interface.

## Folder Browse Button

A standard file browse button labelled "Download Folder" allows users to easily select and navigate to the desired download location.

## 4.4 Input Types:

The YouTube Video Downloader tool utilizes several different types of inputs to accept the necessary information from users:

### Text Input Box

A standard text input box is used to accept the YouTube video/playlist URL from the user. This allows the user to simply paste or type in the full URL.

### Drop-down List

For selecting the desired video quality, a drop-down list is populated dynamically based on the available formats for the given video. This ensures the user can only select a valid format that can actually be downloaded.

### Checkboxes

Checkboxes are used to provide yes/no options for features like downloading subtitles, comments, thumbnails, etc. This allows users to easily enable or disable these optional features.

### File Browser

A file browser input is provided, in the form of a standard "Browse" button, to allow the user to navigate and select the desired download folder location.

### Numeric Input Boxes

For playlist inputs, numeric input boxes are used to accept the start and end indexes that define the range of videos to download from the playlist.

### Template Input

A customizable filename template is provided, based on standard and optional metadata fields, to allow the user to define the desired format for downloaded video file names.

The various input types are chosen based on what provides the most intuitive and easy-to-use interface for accepting the specific data required from the user. Text boxes, drop-down lists and checkboxes are used for options where simplicity is key, while file browsers and numeric inputs follow standard interface conventions for those types of inputs.

## 4.5 Default Values:

Where appropriate, default values have been assigned to various input options within the YouTube Video Downloader tool in order to provide a recommended starting point for users. The following outlines the assigned default values:

### Video Quality Selection

The highest available video quality (e.g. 1080p or 2160p) is selected by default in the video quality drop-down list. This ensures the best possible video quality is downloaded unless the user specifies otherwise.

### Optional Features

For optional features like subtitles, comments, thumbnails, etc., the following default values are assigned:

- Downloading English subtitles: Checked by default
- Downloading thumbnail: Unchecked by default
- Downloading comments: Unchecked by default
- Downloading audio only: Unchecked by default

### Playlist Range

For playlist inputs, reasonable default values are assigned:

- Start index: 1 (the first video in the playlist)
- End index: The total number of videos in the playlist

This default range will download the entire playlist, while allowing the user to specify a subset of videos if desired.

### Filename Template

A default filename template is assigned of the format:

```
"[% (title)s]-[% (id)s].% (ext)s"
```

This template will name videos based on their title and YouTube video ID, while assigning the appropriate file extension (.mp4, .webm, etc.)

The assigning of sensible default values aims to provide a good "out-of-the-box" experience for users. The defaults can then be easily changed or specified differently by the user as needed. The default values also demonstrate the recommended options, which may help educate users, especially for their first use of the tool.

## 4.6 Error Handling:

Proper error handling is implemented to deal with invalid or malformed user inputs, as well as technical errors that may occur during the download process. The following outlines the approach to error handling within the YouTube Video Downloader tool:

### Input Validation Errors

As described in the "Input Validation" subsection, the tool performs validation checks on key inputs like the URL, video quality selection, and download folder path.

If any validation fails, a user-friendly error message is shown to prompt the user to correct the invalid input. The message specifies the exact nature of the validation error to assist the user.

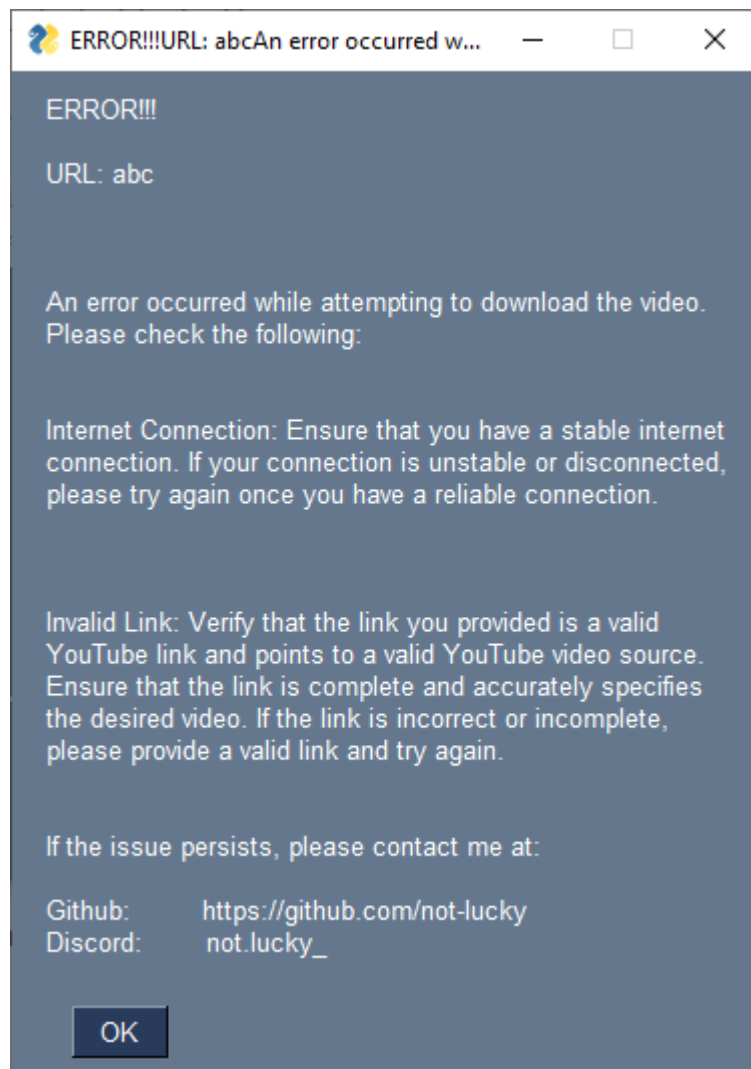


Fig. 4.6.1 Error

## Technical Errors

The tool uses the yt\_dlp library to download YouTube videos. If any technical errors occur during this process, they are caught and a generic error message is shown.

The error message notifies the user that a download error occurred, without specifying complex technical details. It then provides recommendations for common causes of errors, like an unstable internet connection or invalid URL.

The user is also given contact information for the tool's developer in case the issue persists. This allows the developer to diagnose and resolve any recurring technical problems.

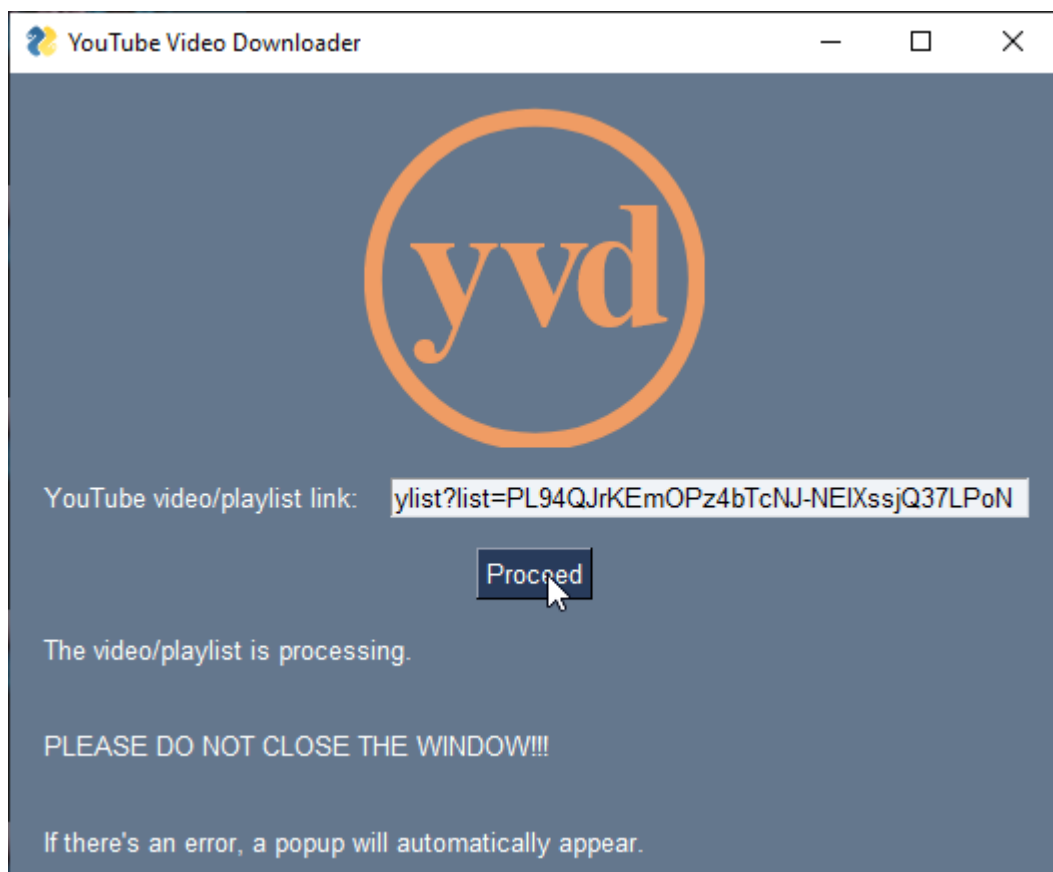


Fig. 4.6.2 Message

The goal of the error handling approach is to:

- Catch invalid inputs early and notify users in a respectful, helpful manner



- Handle technical errors gracefully by informing users that an issue occurred while avoiding overwhelming them with complex details
- Provide recommendations and next steps to help users resolve common error causes
- Allow users to contact the developer for assistance with persistent problems

Overall, the error handling balances the needs of users for simplicity and assistance with the ability of the developer to diagnose and resolve complex technical problems behind the scenes.

## 5. Output Design

### 5.1 Description of User Interface

The user interface of the YouTube video downloader application consists of two main graphical user interface (GUI) screens: one for downloading single videos and one for downloading playlists. The single video GUI screen contains the following elements:

- A header containing the application title and branding image.
- An input field for the user to enter a YouTube video URL.
- A "Proceed" button to validate the URL and proceed to the video information screen.
- A processing message area to inform the user that the URL is being validated.

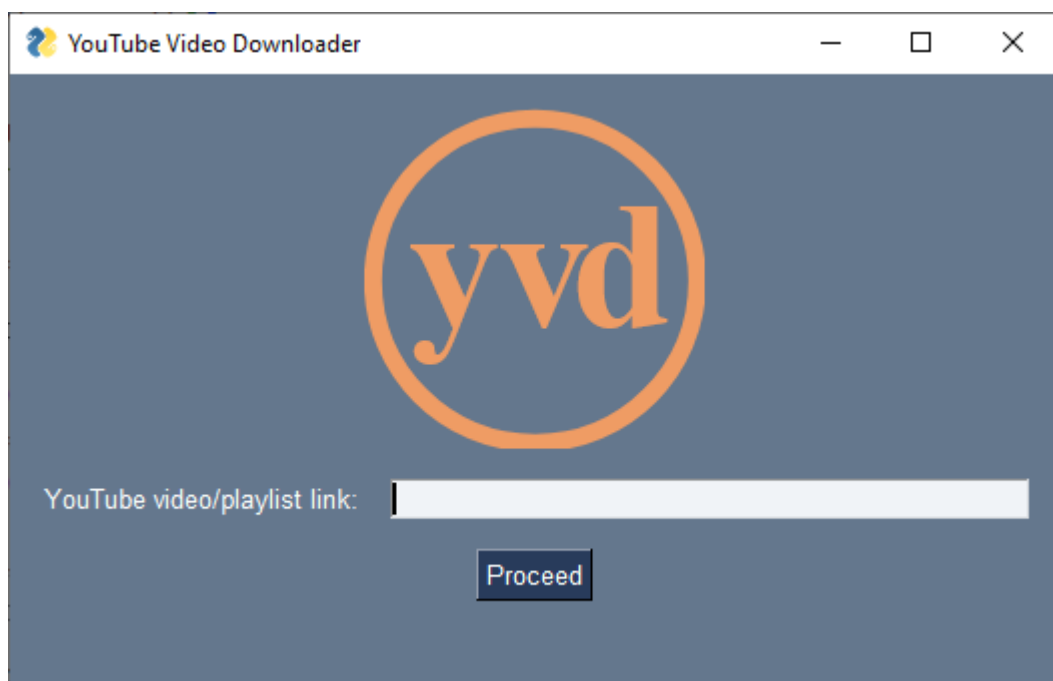


Fig. 5.1.1 Initial Screen

The video information screen contains:

- A thumbnail image of the video.
- Details about the video such as the title, uploader, duration, view count and upload date.
- A combo box for the user to select the desired video quality and format.
- Checkboxes to select additional download options like subtitles, audio only, thumbnail and comments.
- An input field for the user to specify the download folder location.
- A "Download Video" button to start the download.
- A multiline text area to display download status messages and errors.

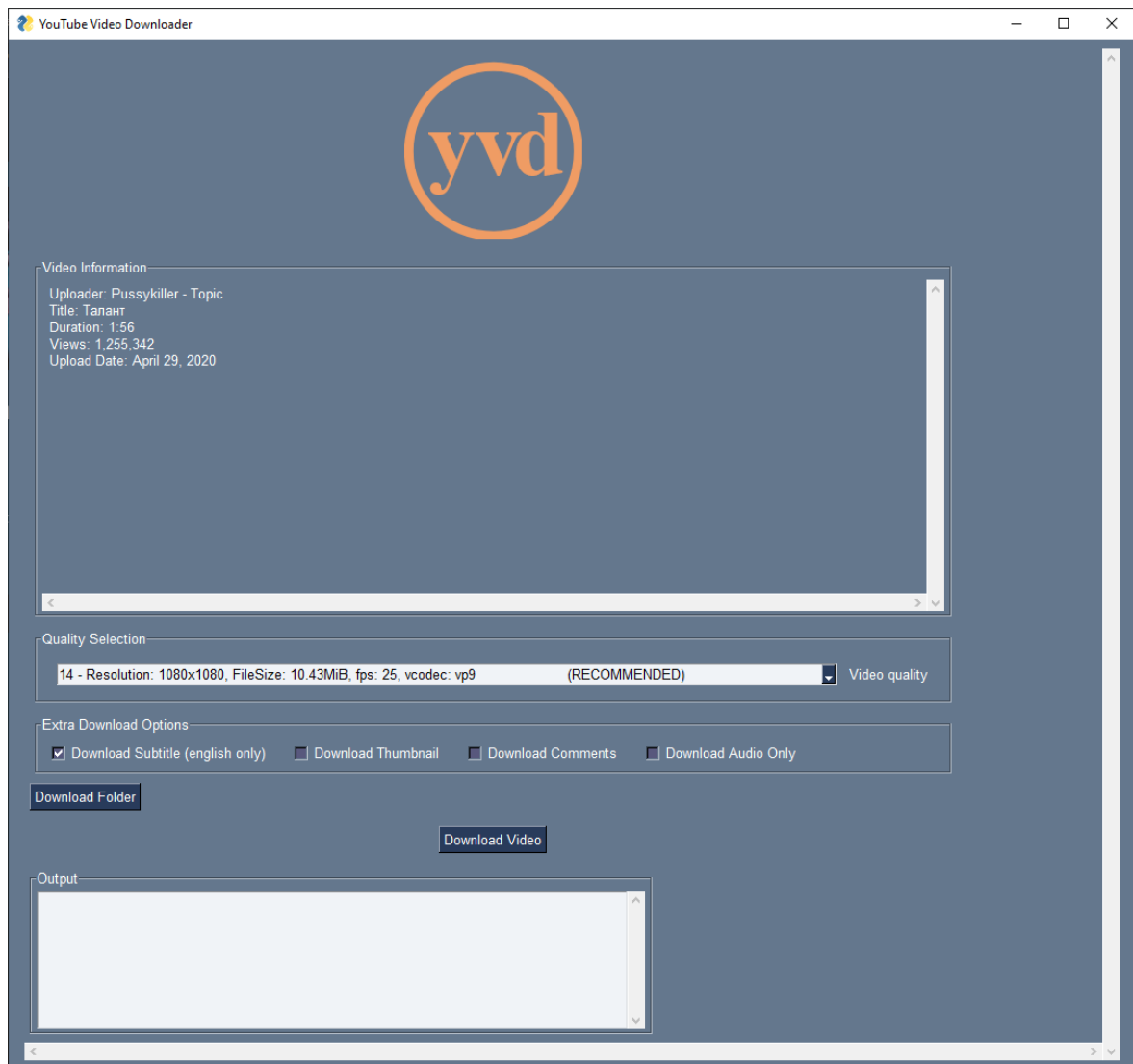


Fig. 5.1.2 Video Screen

The playlist GUI screen contains similar elements but is tailored specifically for playlist downloads. It includes the following:

- Details about the playlist title and videos.
- Inputs for the user to specify a range of playlist videos to download.
- A combo box to select the maximum video quality.

- Checkboxes for additional options like putting videos in a subfolder, numbering videos and downloading audio only.
- Input fields for the download folder location.
- A "Download Playlist" button to start the playlist download.
- A multiline text area for status messages and errors.

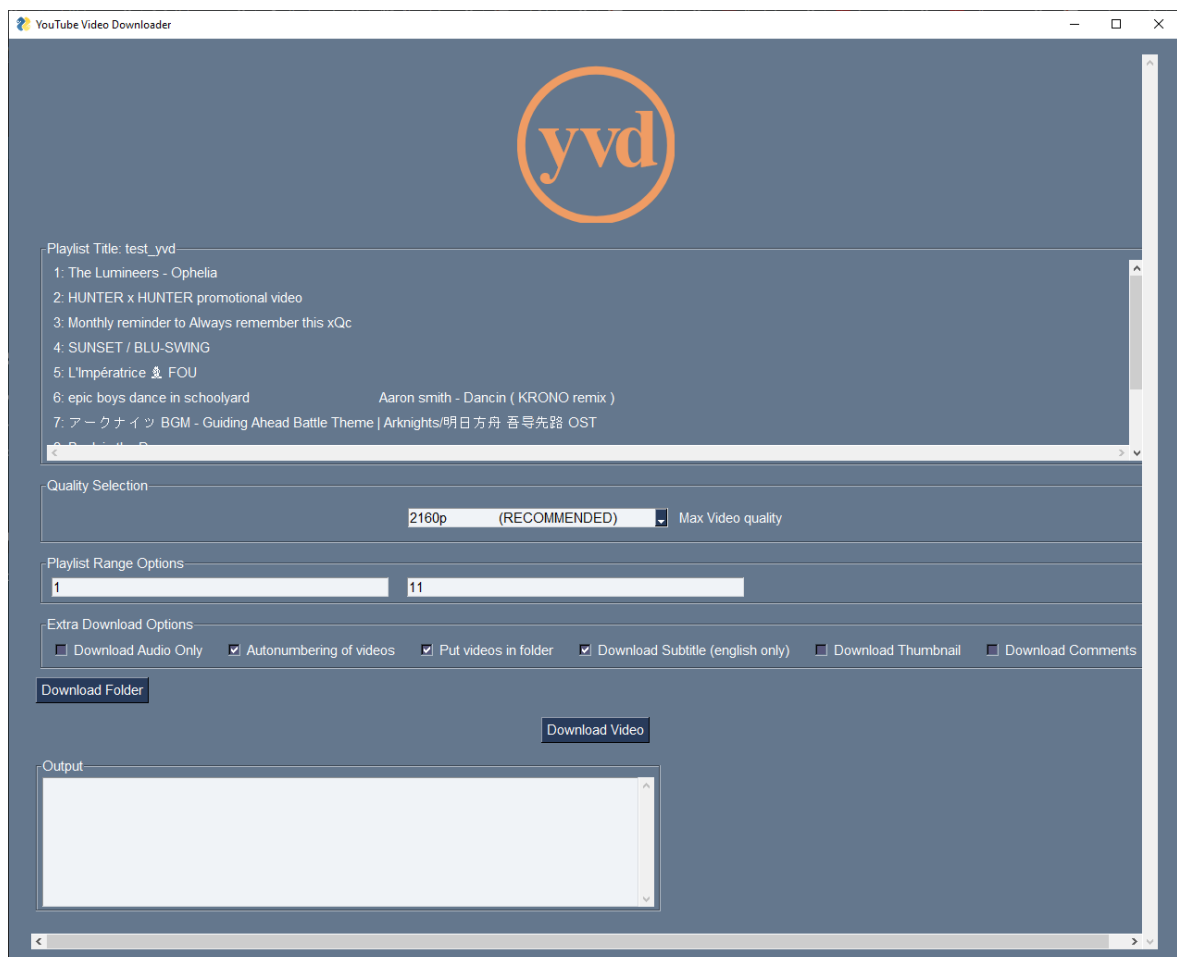


Fig. 5.1.3 Playlist Screen

After a download completes, a popup window is shown with buttons for the user to either continue downloading from the same GUI screen or start a fresh download from the initial GUI screen.

In summary, the user interface uses standard GUI elements like inputs, buttons, combo boxes and checkboxes in an organized and intuitive manner. Relevant details are shown for both single video downloads and playlist downloads, tailored to each download type.

## 5.2 Details of output

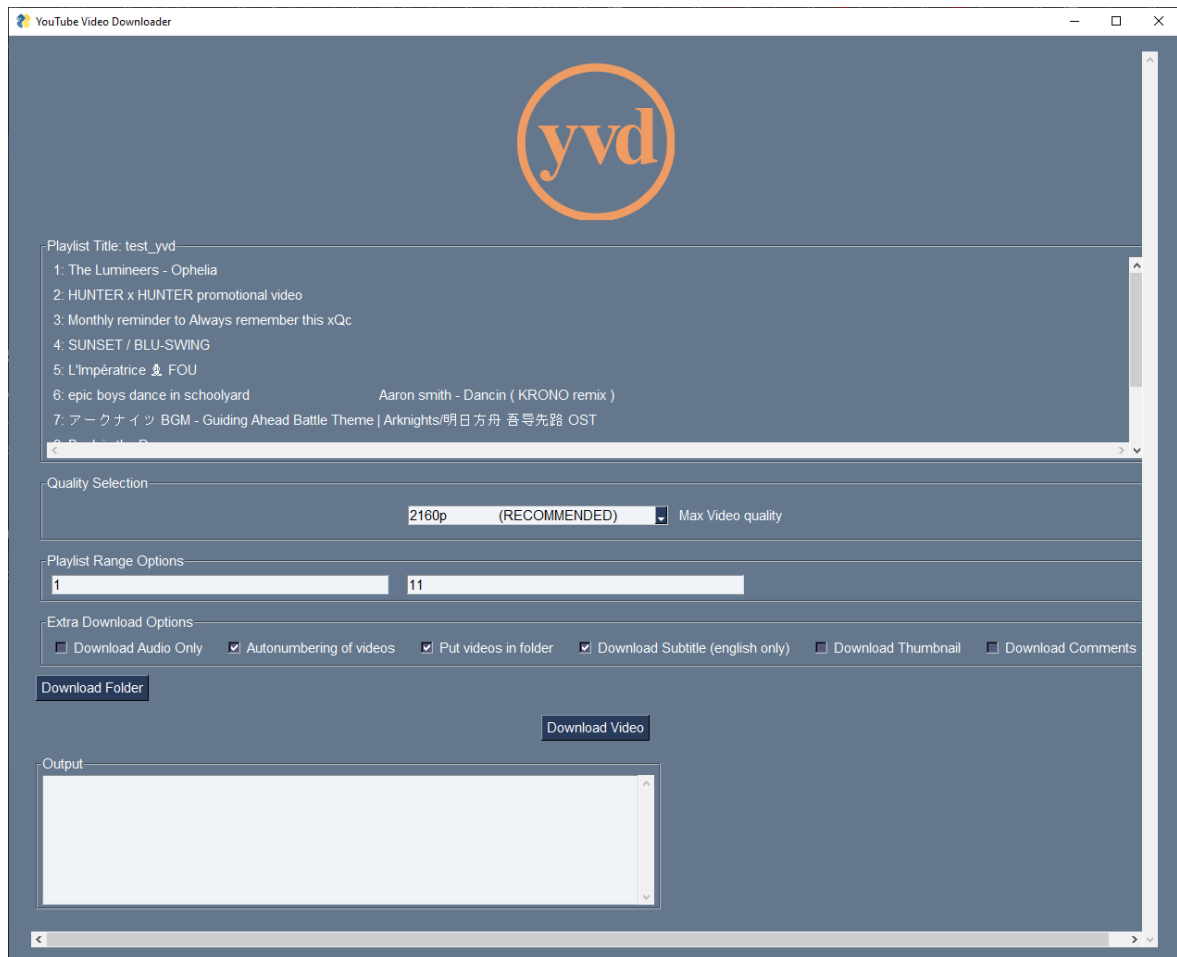


Fig. 5.2.1 Details

The application provides the following types of output to the user:

- Status updates and progress tracking: While a video is being downloaded, status updates are displayed in the "Output" text area showing the current progress, download speed, time remaining and file name. This provides real-time information on the download progress.



Fig. 5.2.2 Download Status

- Download completion notification: Once a video download is complete, a notification message is shown in the "Output" text area informing the user that the download has finished.

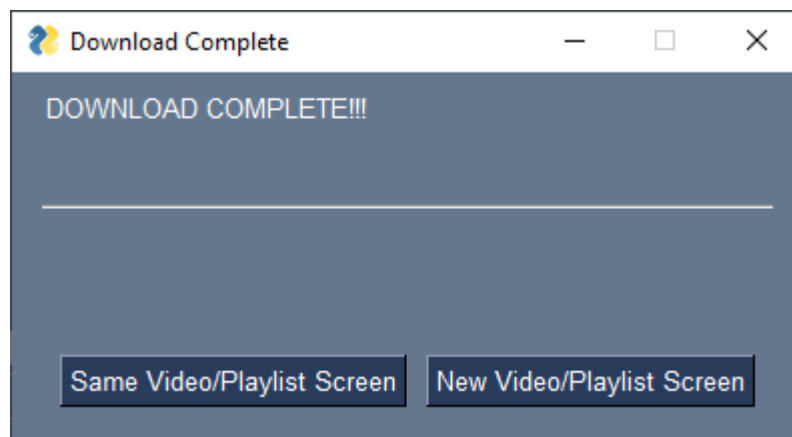


Fig. 5.2.3 Complete notification

- Error messages: In case of any errors during link validation or video download, appropriate error messages are displayed to help the user troubleshoot the issue. The user is also prompted to try again once the issue is resolved.
- Video information: Key details about the video like title, uploader name, duration, views and upload date are fetched from YouTube and displayed on the screen. This allows the user to verify that the correct video will be downloaded.
- Format and quality selection: The user is presented with a dropdown to select the video format and quality based on available options for that video. This ensures the downloaded video meets the user's requirements.



- Extra options status: The status of extra download options like subtitles, thumbnail and comments is displayed to the user via checkbox controls.

The outputs aim to provide relevant, timely and actionable information to the user at each stage of the application workflow - from link validation, format selection, download progress monitoring and completion notification. Error messages are designed to be clear and simple to understand, with suggestions to help the user resolve issues.

### 5.3 Expected Warnings/Errors

The following warnings and errors were anticipated and accordingly handled in the application's output design:

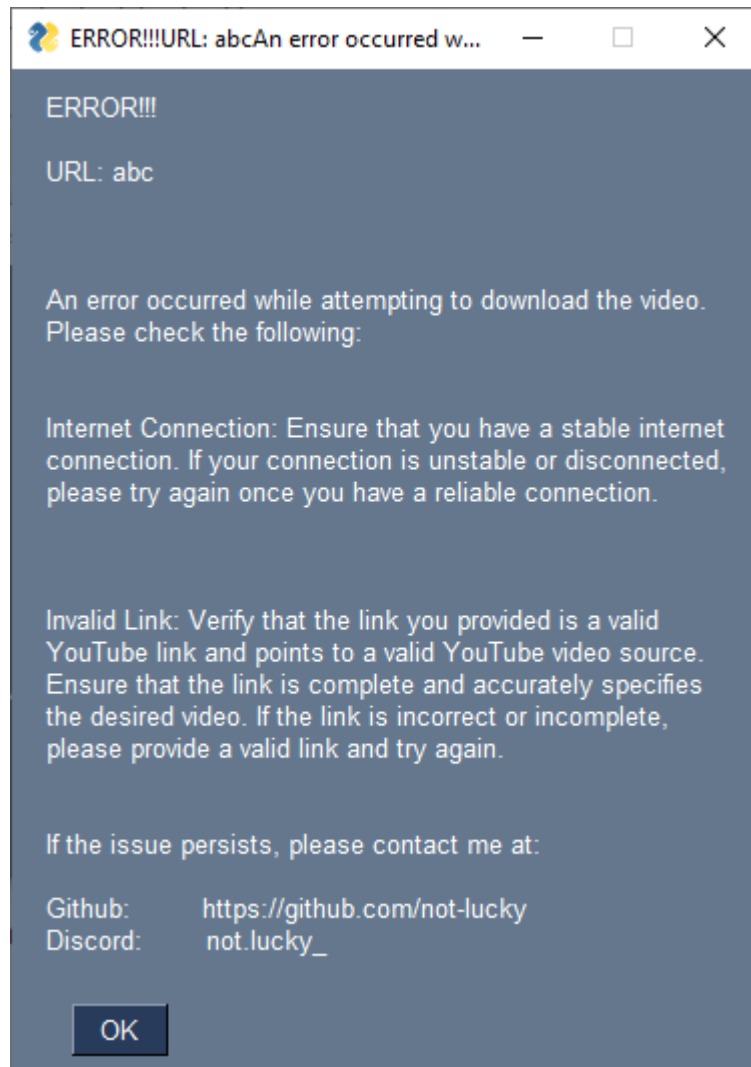


Fig. 5.3 Error

#### 1. Invalid YouTube Link

If the user provides an invalid or incomplete YouTube link, an appropriate error message is displayed:

"An error occurred while attempting to download the video. Please check that the link you provided is a valid YouTube link and points to a valid YouTube video source. Ensure that

the link is complete and accurately specifies the desired video. If the link is incorrect or incomplete, please provide a valid link and try again."

The user is then prompted to provide a valid YouTube link.

## 2. Unstable Internet Connection

If the internet connection is unstable during the video download process, an error may occur. The error message suggests checking the internet connection and then trying the download again once there is a reliable connection:

"An error occurred while attempting to download the video. Please check Internet Connection: Ensure that you have a stable internet connection. If your connection is unstable or disconnected, please try again once you have a reliable connection."

## 3. yt\_dlp Errors

Any exceptions raised by the yt\_dlp library during the download process are caught and a general error message is displayed, advising the user to check the YouTube link and internet connection:

"An error occurred while attempting to download the video. Please check the following:

Internet Connection...

Invalid Link..."

In all cases, the error messages provide clear guidance to the user on potential causes and ways to resolve the issue so they can successfully download the desired YouTube video. Clear and informative warnings and errors are an integral part of the application's output design to provide a positive user experience.

## 6. System Testing, Implementation & Maintenance

### 6.1 System Testing:

Thorough system testing was conducted to ensure that the application functions as intended. Various test scenarios were performed:

- Valid URL: Videos and playlists from YouTube were successfully downloaded in different formats and qualities. Subtitles, thumbnails and comments were also downloaded where applicable.
- Invalid URL: The application handled invalid YouTube URLs gracefully by showing an error message to the user.
- Progress tracking: The progress bars and status updates were tested to ensure they function properly during the downloading process.
- Error handling: Errors during the download process were captured and handled appropriately by showing informative error messages to the user.
- Single video download: Various options like video format, subtitle download, thumbnail download, etc. were tested for single video downloads.
- Playlist download: Ability to download entire playlists or a range of videos in a playlist was tested.

Any issues found during testing were swiftly fixed to ensure a quality user experience.

### Implementation and Maintenance

The application was implemented using Python and the PySimpleGUI library for the graphical user interface. The yt\_dlp library is used for the actual YouTube content downloading.

The code has been structured to allow for easier maintenance and extensibility. Additional features and improvements identified through testing and user feedback will be incorporated in future releases. These may include:

- Refactoring duplicate code between single video and playlist downloads.
- Enhancing the progress tracking for playlist downloads.
- Saving download history/logs to track previous downloads.
- Allowing users to pause/resume downloads.

Regular updates and releases will be made available to users, with ongoing system testing to ensure high quality and reliability. User feedback will also be incorporated where feasible to continuously improve the application.

## 6.2 Implementation

``ERROR_MESSAGE`` - Contains a generic error message that is shown to the user in case of errors while attempting to download a video. The message provides guidance on possible causes of the error and how to resolve it.

``THEME`` - Specifies the PySimpleGUI theme used for the GUI. In this case, it is set to 'Dark Blue 14'.

``THUMBNAIL_SIZE`` - Contains the width and height of video thumbnails shown in the GUI. It is set to (300, 300) pixels.

``WINDOW_TITLE`` - Specifies the title of the GUI window. It is set to "YouTube Video Downloader".

These constants help structure the code by defining values that are reused throughout the program. Some benefits of using constants are:

- Makes code more readable by giving meaningful names to values
- Ensures consistent values are used instead of "magic strings"
- Allows the values to be easily changed in one place if needed in the future

By using constants, the code avoids hard-coding literal values throughout, making it more robust and maintainable.

Here are detailed explanations of the functions:

`create_center_column():`

This function centers a PySimpleGUI Column element within the GUI window. It takes the Column as an input and returns a layout with empty rows (VPush elements) above and below the column to center it vertically within the GUI window.

It is used to center elements like buttons and images, providing a more visually appealing user interface. The column passed as an argument can contain any GUI elements like Text boxes, Buttons, Checkboxes, etc.

`datetime:`

This module provides functionality for manipulating dates and times in Python. The code imports the datetime module and defines the `upload_date()` function.

This function uses datetime to format the upload date string retrieved from the YouTube API (e.g. "20190301") into a more human readable format (e.g. "March 1, 2019"). This makes the date information more understandable for the user.

`download_video():`

This function handles downloading a single YouTube video. It takes as inputs the video format options, user selections, and the URL. It then sets download options like enabling subtitle download, thumbnail download, etc.

It calls the `yt_dlp` library to perform the actual download and while the video is downloading, it displays status updates and progress information in the GUI. Once the download is complete, it notifies the user.

`download_playlist():`

This function handles downloading an entire YouTube playlist. It takes similar inputs as `download_video()` but also handles playlist-specific options like the start and end video numbers, putting videos in a folder, and autonumbering videos.

It then calls `yt_dlp` to download all videos in the playlist, displaying individual progress information for each video download in the GUI. Once all videos have downloaded, it notifies the user.

`folderr():`

This function constructs the output file name and path for the downloaded video. It accepts parameters like the download folder path, whether it is a playlist download, and options for putting videos in a folder and autonumbering them.

It then constructs an appropriate file name template string based on those parameters, for example:

`%(title)s-[% (id)s].%(ext)s` for a single video

`%(playlist\_title)s/%(playlist\_index)s - %(title)s-[% (id)s].%(ext)s` for a playlist video with autonumbering

It then returns the full file path and name using that template. This ensures videos are saved with standardized, unambiguous names.

`initial_screen():`

This function creates the initial GUI screen that is shown when the application starts. It contains:

- An image
- A text box for the user to enter the YouTube URL
- A "Proceed" button

Once the user clicks proceed, the URL is validated. If valid, the appropriate next screen (video or playlist) is shown. If invalid, an error message is displayed.

`playlist_screen():`

This function creates the GUI screen that is shown when a YouTube playlist URL is provided. In addition to:

- Video details
- Format selection
- Options to download subtitles, thumbnails and comments



It contains options specific to playlist downloads like:

- Selecting a range of videos to download
- Putting videos in a folder
- Autonumbering videos

`popup_continue_or_not()`:

This function displays a popup window once a video or playlist has finished downloading, asking the user if they want to:

- Continue using the same video/playlist download screen

OR

- Start over with a new download screen

This allows the user to easily download multiple videos/playlists without having to reopen the application each time.

`upload_date()`:

This function formats a YouTube video upload date string (in YYYYMMDD format) retrieved from the YouTube API into a human readable format like "March 1, 2019".

It does this by:

- 1) Parsing the date string into a datetime object using `strptime()`
- 2) Reformatting the datetime object into the desired format using `strftime()`

This makes the upload date information more understandable for the user.

`validate_youtube_url()`:

This function takes a YouTube URL as input and attempts to extract metadata about the video from the YouTube API using the `yt_dlp` library.

If successful, it returns the JSON response containing metadata like the video title, upload date, description, etc.

Otherwise, if the URL is invalid or there is another error, it returns an appropriate error message.

video\_screen():

This function creates the GUI screen that is shown when a single YouTube video URL is provided.

In addition to:

- Video details retrieved from the YouTube API
- Format selection based on available options
- Options to download subtitles, thumbnails and comments

It displays relevant information needed by the user to select options for downloading that specific video.

## 6.3 Maintenance:

Regular and proactive maintenance will be performed to ensure the YouTube video downloader application continues to meet user needs and runs reliably. The following maintenance activities will be carried out:

### Bug Fixing

Any bugs or issues reported by users will be promptly investigated and reproduced. The root cause of the issue will be determined through debugging and testing. Appropriate fixes will then be implemented through code changes. Unit tests will be added or updated to prevent reoccurrence of the bug. Fixed versions will be released to users as soon as possible.

### Enhancements

Based on user feedback and requests, enhancements to the application's features and functionality will be planned and implemented. Potential enhancements have been identified such as:

- Improving progress tracking for playlist downloads
- Enabling pause/resume of downloads
- Saving download history/logs
- Refactoring duplicate code

New features may also be added according to feasibility, priorities and resource availability. All enhancements will be thoroughly tested before release.

### Code Quality

Code quality will be improved on an ongoing basis through code reviews, refactoring, and adhering to best practices. Duplicated code will be identified and consolidated. The code structure will be optimized for readability, extensibility and testability. Unit tests will be added and expanded to ensure features work as intended.

### System Documentation

All relevant system documentation will be kept up-to-date, including user guides, technical specifications, design documents, etc. This will help onboard new developers and support new users. Documentation will be reviewed and updated with each release.

### Dependency Management

Any security vulnerabilities or bug fixes in the application's dependencies (PySimpleGUI, yt\_dlp) will be promptly identified and incorporated through dependency version updates. Regular checking of dependency advisories will be performed.

Overall, a holistic, preventive and iterative approach will be taken to maintenance in order to continuously optimize the application's functionality, usability, reliability and security. User feedback will be proactively gathered and incorporated where feasible. A high-quality user experience will be the primary goal of all maintenance activities.

## 7. Summary and Future Scope

### 7.1 Summary:

The YouTube video downloader application aims to provide a simple yet effective solution for downloading YouTube videos and playlists. The key objectives of the application are:

#### Ease of use

The graphical user interface built using PySimpleGUI keeps the application easy to use for both technical and non-technical users. Users simply enter a YouTube link and select options for format, subtitles, etc.

#### Comprehensive options

Users have options to download videos in different formats and qualities. Subtitles, thumbnails and comments can also be downloaded where available. Options specific to playlist downloads are provided.

#### Progress monitoring

Status updates and progress information are continuously displayed, keeping users informed of the download progress and completion. Errors are handled gracefully with guidance on resolving issues.

#### Cross-platform compatibility

The application is developed in Python, which is cross-platform compatible. The dependencies used also have wide platform support. This ensures the application can run on Windows, Linux and macOS.

#### Future extensibility

The modular code structure, use of libraries and object-oriented approach aim to provide a foundation for

future extensions and enhancements to the application's features and functionality.

#### Regular maintenance

An ongoing maintenance strategy focusing on bug fixing, enhancements, documentation, dependency management and continuous improvement will be employed to maintain a high-quality user experience over the long term.

In summary, the YouTube video downloader application seeks to provide an intuitive, feature-rich yet simple solution for downloading YouTube content. The key focus areas are usability, flexibility, transparency and future extensibility, enabled by the choices of technologies, architecture and development approach employed.

Going forward, user feedback and testing will be integral to further optimizing the application according to needs and priorities.

## 7.2 Future Scope

The current software implementation of the YouTube video downloader proves the feasibility of the core concept. However, there exists substantial scope for augmenting and enhancing the application's feature set and functionality going forward.

One primary avenue for advancement is broadening the range of download options and user control over the process. Specifically:

- Enabling the selection of different video and audio formats would give users more choices based on their devices and purposes.
- Allowing the specification of video resolution and bitrate would facilitate customized downloads at appropriate quality levels.
- Permitting modification of subtitle information and metadata, including language options, would enhance accessibility and usability.

A second major area for improvement is refining the playlist download capabilities. Potential extensions include:

- Facilitating the download of only a subset of videos from a playlist, within a designated start and end range.
- Providing the means to automatically detect and download only newly added videos since the last run, expediting repetitive playlist downloads.
- Establishing an organized file structure with properly labeled subfolders for each downloaded playlist, improving organization of downloaded content.

Thirdly, bolstering error handling and implementing systematic logging will fortify the application's robustness and maintainability going forward. Techniques may comprise:

- Checking for and managing updates to external dependencies such as yt-dlp to preclude runtime errors.

- Establishing a structured logging system to record information on errors, warnings and successful executions for the purposes of debugging and continuous improvement.

Finally, incorporating a database backend would permit persisting user preferences and metadata between runs of the application. Stored information could then be accessed and applied automatically on each launch, enhancing user experience. Relevant options for storage comprise:

- Default download location and folder structure.
- Quality settings encompassing video resolution, audio bitrate and other options.
- Playlist progress information, indicating the last successfully downloaded video.

In summary, there exists ample scope for augmenting this initial YouTube video downloader, especially regarding download options, playlist handling, error management and integrated data persistence. With judicious development along these dimensions, the application could evolve into a robust and comprehensive tool for easily fetching content from YouTube.



## APPENDICES:

This section provides an overview of the key modules and functions used in the YouTube video downloader program.

### A.1 Modules

#### youtube\_dl.py

This module utilizes the yt\_dlp library to download YouTube videos and playlists. It handles validating the input URL, extracting video details, and performing the actual download.

#### PySimpleGUI.py

This module is used to create the GUI for the program. It builds the different windows and elements to provide an interface for the user.

#### datetime.py

This standard library module is used to format the upload date of YouTube videos.

#### os.py

This standard library module contains functions for working with files and folders. It is used to create the download folder and generate the output filename template.

#### json.py

This standard library module is used to load and parse the JSON data received from the YouTube API response.

### A.2 Functions

#### validate\_youtube\_url()

This function takes the URL input by the user and utilizes yt\_dlp to validate it. It extracts relevant video information if the URL is valid.

#### initial\_screen()

This function builds the initial GUI window where the user enters the YouTube URL. It handles validating the URL and showing appropriate output.

`video_screen()`

This function builds the GUI window for a single video download. It displays relevant video details and options for the user to choose from.

`playlist_screen()`

This function builds the GUI window for a playlist download. It displays playlist details and similar options as for a single video.

`folderr()`

This function generates the output filename template based on user options for single videos or playlist downloads.

`download_video()`

This function performs the actual download of a single YouTube video using the options selected by the user in the GUI.

`download_playlist()`

This function performs the actual download of a YouTube playlist using the options selected by the user in the GUI.

`upload_date()`

This function formats the upload date of a YouTube video into a readable string.

`create_center_column()`

This function helps center layout elements within a PySimpleGUI Column.

`progress_hook()`

This function is used as a progress hook for `yt_dlp` to show download progress in the GUI.

`popup_continue_or_not()`

This function builds the popup window asking the user if they want to continue downloading after a video/playlist.

`main()`

This is the `main()` function that runs the program and controls the flow between windows.

## **Bibliography:**

1. YouTube <https://youtube.com>
2. PySimpleGUI <https://github.com/PySimpleGUI/PySimpleGUI>
3. yt-dlp <https://github.com/yt-dlp/yt-dlp>
4. python <https://www.python.org>
5. ffmpeg <https://ffmpeg.org/>
6. thorium browser <https://github.com/Alex313031/Thorium/>
7. Visual Studio Code <https://github.com/microsoft/vscode>
8. System Architecture <https://www.imt.ch/en/insight/the-architecture-process-systematic-creation-of-a-system-and-software-architecture/>
9. Project Planning <https://blog.invgate.com/how-to-ensure-your-project-planning-is-successful>
10. Risk Management <https://practicalrisktraining.com/iso31000>