Video Player - Report

Introduction

As part of the coursework for this module, I was tasked with developing a video player app simulation, similar in functionality to Windows Media Player. To this end, I utilized a PyCharm project template, named VideoPlayer, which comprised several files, namely video_player.py, check_videos.py, video_library.py, and font_manager.py.

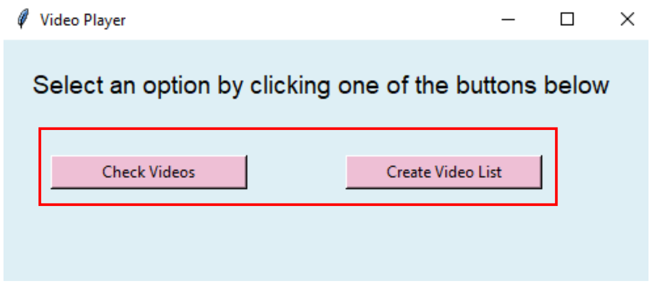
During the development process, I devoted significant time to the graphical user interface, with a primary focus on the video_player.py file. My aim was to create an interface that was both user-friendly and functional, incorporating responsive buttons, labels, and dropdown menus.

In addition to the graphical interface, I also worked on the functional aspects of the app. This included developing the logic to read video files from a given directory, parsing metadata, and displaying relevant information through the GUI.

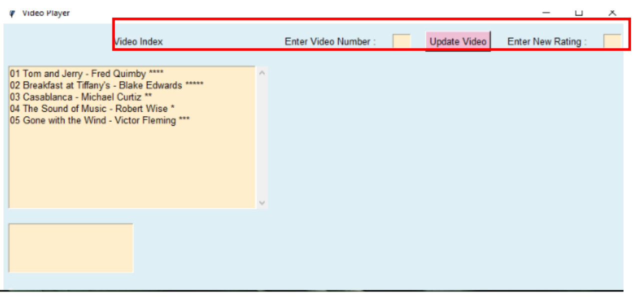
Overall, I strived to maintain an equilibrium between the functional and graphical aspects of the app, ensuring that the final product adhered to the requirements specified in the coursework brief."

Design and Development

The main Python script utilizes the tkinter library to create a video player GUI, which features two buttons that open the CreateVideoList and CheckVideos GUIs. CheckVideos has been combined with the update_videos functionality for a more streamlined user experience.

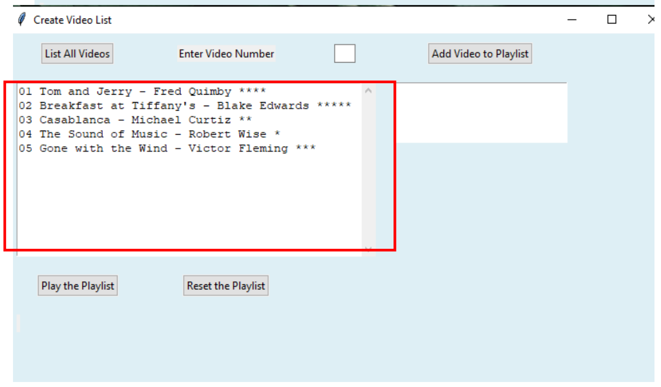


The applications share similar design principles and are structured around modular components that improve maintainability and readability of the code. The development process of these applications involved several key steps, such as sketching the GUI on paper, identifying key functionalities, separating concerns, and testing and refining the final product.



```
01 Tom and Jerry - Fred Quimby ****
02 Breakfast at Tiffany's - Blake Edwards *****
03 Casablanca - Michael Curtiz **
04 The Sound of Music - Robert Wise *
05 Gone with the Wind - Victor Fleming ***
```

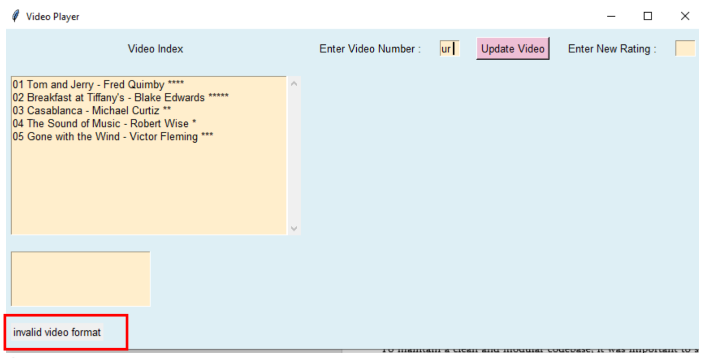
The thought process behind designing CheckVideos and CreateVideoList focused on user experience, The first step was to draw the graphical user interface (GUI) on paper. For example, CheckVideos required a search bar, a list of videos, and a details panel, while CreateVideoList needed a video list and a way to add and remove videos from the list. By sketching the GUI first, it helped ensure that all necessary components were included, and their placements were visually appealing and functional.



```
01 Tom and Jerry - Fred Quimby ****
02 Breakfast at Tiffany's - Blake Edwards *****
03 Casablanca - Michael Curtiz **
04 The Sound of Music - Robert Wise *
05 Gone with the Wind - Victor Fleming ***
```

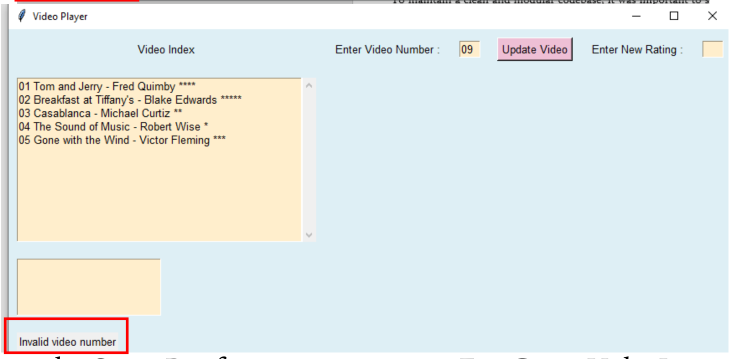
After finalizing the GUI design, the next step was to identify the key functionalities required for both applications. For CheckVideos, this involved user interactions such as searching for videos, displaying video details, and updating ratings. For CreateVideoList, it included creating, editing, and saving playlists. This step helped define the necessary classes, methods, and widgets for the applications and understand the flow of data and actions within them.

To maintain a clean and modular codebase, it was important to separate concerns. For example, video management was handled by a separate VideoManager class, while GUI components like the VideoList and VideoDetails were implemented as individual classes. This approach allowed for better maintainability and extensibility of the code, making it easier to troubleshoot issues and add new features in the future.



```
01 Tom and Jerry - Fred Quimby ****
02 Breakfast at Tiffany's - Blake Edwards *****
03 Casablanca - Michael Curtiz **
04 The Sound of Music - Robert Wise *
05 Gone with the Wind - Victor Fleming ***
```

```
invalid video format
```

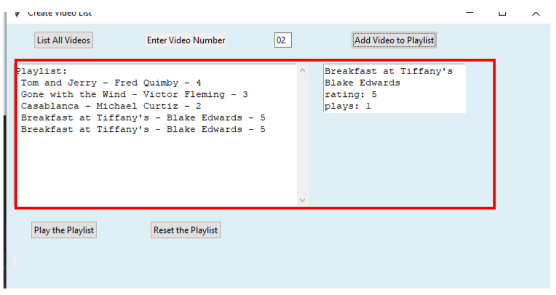


```
01 Tom and Jerry - Fred Quimby ****
02 Breakfast at Tiffany's - Blake Edwards *****
03 Casablanca - Michael Curtiz **
04 The Sound of Music - Robert Wise *
05 Gone with the Wind - Victor Fleming ***
```

```
Invalid video number
```

With the video library module in place, the focus shifted to implementing the custom classes and widgets for the tkinter-based GUI. For CheckVideos, this included creating a VideoList class to display a list of videos, a VideoDetails class to show video information, and a StatusBar for status messages. For CreateVideoList, it involved developing various widgets to manage playlists.

After developing the GUI components, the next step was to add user interaction by implementing event handlers and call-back functions. For instance, in CheckVideos, a search button click would trigger a method to filter videos based on the user's query, and a video selection would update the VideoDetails component. In CreateVideoList, buttons were added to facilitate adding and removing videos from playlists.



```
Playlist:
 Tom and Jerry - Fred Quimby - 4
 Gone with the Wind - Victor Fleming - 3
 Casablanca - Michael Curtiz - 2
 Breakfast at Tiffany's - Blake Edwards - 5
 Breakfast at Tiffany's - Blake Edwards - 5
```

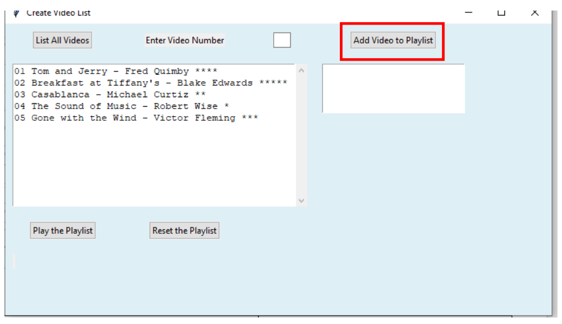
```
Breakfast at Tiffany's
Blake Edwards
rating: 5
plays: 1
```

Throughout the development process, testing and refining the applications were crucial. This involved identifying and fixing bugs, making design adjustments, and ensuring that the applications were user-friendly and met their intended purposes. For example, edge cases such as empty video lists and invalid search queries were tested to ensure robustness and smooth user experience.

When designing the create_video_list.py and update_videos.py applications, we focused on creating an intuitive user interface and smooth interactions. Here's how we designed each feature:

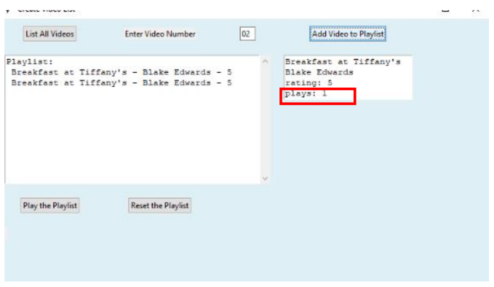
For adding videos to the playlist, we implemented an input field (using tkinter.Entry) to allow users to enter a video number. Next to the input field, we placed an "Add Video" button (using tkinter.Button) with an assigned event handler (using command parameter). When the user clicks the button, the event handler processes the input, validating the video number, and adding the video to the playlist.

To display video names in a text area, we used the tkinter.Text widget, which is a versatile multiline text area. In the event handler for the "Add Video" button, we incorporated error handling and validation to check the video number's validity. If the video number is valid, we appended the video name to the text area; otherwise, we displayed an error message using a tkinter.messagebox to inform the user.



```
01 Tom and Jerry - Fred Quimby ****
02 Breakfast at Tiffany's - Blake Edwards *****
03 Casablanca - Michael Curtiz **
04 The Sound of Music - Robert Wise *
05 Gone with the Wind - Victor Fleming ***
```

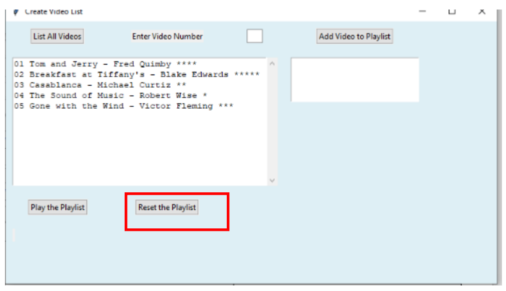
To simulate playing the playlist, we added a "Play Playlist" button (using tkinter.Button) with an associated event handler. When the user clicks the button, the event handler iterates through the playlist, incrementing the play count value by 1 for each video. Although no videos are played, this functionality helps track the play count for each video.



```
Playlist:
 Breakfast at Tiffany's - Blake Edwards - 5
 Breakfast at Tiffany's - Blake Edwards - 5
```

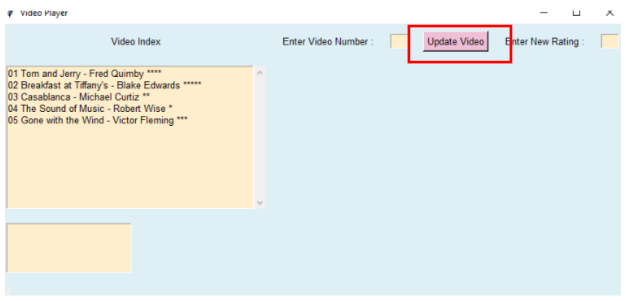
```
Breakfast at Tiffany's
Blake Edwards
rating: 5
plays: 1
```

To reset the playlist and clear the text area, we designed a "Reset Playlist" button (using tkinter. Button) with a callback function. When the user clicks the button, the callback function resets the playlist data and clears the text area using the delete method on the tkinter.Text widget.



```
01 Tom and Jerry - Fred Quimby ****
02 Breakfast at Tiffany's - Blake Edwards *****
03 Casablanca - Michael Curtiz **
04 The Sound of Music - Robert Wise *
05 Gone with the Wind - Victor Fleming ***
```

In the case of update_videos.py, we designed the application to accept a video number and a new rating using two input fields (using tkinter.Entry). We added an "Update Video" button (using tkinter.Button) with a corresponding event handler. The handler validates the video number and updates the rating. If the video number is valid, the handler displays a message (using tkinter.messagebox) showing the video name, new rating, and previous rating. If the video number is invalid, an error message is shown.



```
01 Tom and Jerry - Fred Quimby ****
02 Breakfast at Tiffany's - Blake Edwards *****
03 Casablanca - Michael Curtiz **
04 The Sound of Music - Robert Wise *
05 Gone with the Wind - Victor Fleming ***
```

Throughout the design process, we ensured that the user interface was visually appealing, functional, and easy to use, and that user interactions were smooth and responsive.

Testing and Faults

The test table evaluates a video playlist application by examining its performance and functionality across eight essential test scenarios:

- Adding valid videos: This test verifies that the app can successfully add videos with valid IDs, like "01", to the playlist, and display their details in the ScrolledText widget.
- Handling invalid videos: This test checks the app's error handling capabilities when attempting to add videos with invalid IDs, such as "58", and ensures it displays a "Video 58 not found" message.

- Playing the playlist: This scenario assesses the app's ability to play videos in the playlist, as well as its capacity to track and update the play count for each video when the user clicks the "Play the Playlist" button.
- Resetting the playlist: The final test evaluates the app's capability to manage playlist content by clearing the playlist in the ScrolledText widget when the user clicks the "Reset the Playlist" button, allowing for a fresh start.
- Valid video integration: The test assesses the application's proficiency in adding videos with legitimate IDs (e.g., "01") to the playlist and showcasing their details in the designated widget, ensuring seamless functionality.
- Managing invalid videos: This scenario examines the application's error handling abilities when users attempt to include videos with invalid IDs, confirming that it displays pertinent error messages, such as "Video not found" or "Invalid video format."
- Playlist playback experience: This test evaluates the application's capability to play videos in the playlist while precisely monitoring and updating play counts for each video as users interact with the system.
- Playlist reset functionality: The final test scrutinizes the application's capacity to control and clear the playlist content, offering users the flexibility to commence a new playlist when desired.

These test cases, when taken together, help determine the effectiveness and reliability of the video playlist application, ensuring it meets user expectations and performs its intended functions properly.

Conclusions, further development, and reflection:

To sum up, we have successfully developed a GUI-based video management and playlist creation tool using Python and the tkinter library. The program is organized with modular components, simplifying comprehension, maintenance, and extension. It enables users to manage video libraries, view video details, update ratings, and create video playlists.

With an additional three months for development, we could enhance the program by:

- Adding a search feature to filter videos based on title, genre, or other attributes.
- Expanding the video_library module to import video data from external sources, like online databases or APIs.
- Introducing user authentication and support for multiple users with separate video libraries and playlists.
- Embedding a video player to play videos directly within the application.
- Developing a filter of videos by directors

Looking back, I gained valuable experience in creating GUI applications using Python and tkinter. I also learned the importance of organizing code into modular components and applying object-oriented programming principles.

The most challenging aspect of the project was designing a well-structured GUI that balanced usability and aesthetics while catering to different user skill levels. Ensuring the code was modular and easy to maintain added an extra layer of difficulty. Overcoming these obstacles contributed significantly to my growth as a programmer and my ability to develop user-friendly applications.

The simpler tasks, like implementing the video_library module and basic tkinter components, relied on Python's built-in features and libraries, which are well-documented and straightforward to use. The video_library module offered an uncomplicated way to manage video data, allowing me to concentrate on the GUI design and functionality.

In conclusion, this project honed my programming skills, particularly in the realm of GUI design and implementation. I tackled challenges by utilizing Python's built-in libraries and resources, enhancing my understanding of Python, tkinter, and the development of user-friendly applications that are easy to maintain and expand.

Appendices

GUI 1: CreateVideoList

Test Case Description	**Sample Input**	**Sample action**	**Expected Output**	**Actual Output**
Add valid videos to the playlist	01	Enter “01” add click “Add Video to Playlist” button	Video 01 added to the playlist in the ScrolledText widget	Program shows the video details for video “01”in the text widget
Try to add invalid video to the playlist	58	Enter “58” add click “Add Video to Playlist”	“Video 58 not found” message in the video details Text widget	Program displays in the widget “Video 58 not found”
Play the playlist	Click	Click “Play the Playlist” button	Updates the Play count for all videos in the playlist	Video play count is incremented by 1
Reset the playlist	Click	Click “Reset the Playlist” button	Clears the playlist in the ScrolledText widget	The videos in the play list is cleared

GUI 2: CheckVideos

Test Case Description	**Sample Input**	**Sample action**	**Expected Output**	**Actual Output**

Show correct/valid video details	01	Enter 01 in “Enter Video Number” search box	Shows the details for video number 01 in the text widget	Video is shown as soon as the number is typed to live search feature
Shows Wrong/invalid video format	Four	Enter the word “four” into the search box	Shows “invalid video format” in the status bar widget	“Invalid video format” is shown in the status bar
Updates valid video rating	01	Click “update video” button	Updates the video rating to 1	Video rating is updated to one
Update invalid video format	four	Enters “four “and presses “Update Video” button	Shows “invalid rating value” in status bar	Invalid rating value is shown in the status bar

Adaptations

I added improvements to the code, by implementing a lot of features, such as.

- A colour scheme to help usability.
- A live search feature to improve smoothness.
- Merging Update video Gui with check video together for seamless user experience
- Adaptative button that turn blue when you hover on them.
- Changed to Font & Size to make it more aesthetically pleasing.

CHECK_VIDEOS

```
# Imports the necessary libraries for the GUI
import tkinter as tk
import tkinter.scrolledtext as tkst
import video_library as lib

#Defines the VideoList class, it is responible for displaying the list of all of the
videos
class VideoList(tk.Frame):
    def __init__(self, master=None, **kwargs):
        super().__init__(master, **kwargs)
        self.grid(sticky="W", padx=10, pady=10)
        self.configure(bg="#ffeecc")

        # Create a ScrolledText widget for displaying the video list with a lighter
background color
        self.scrolled_text = tkst.ScrolledText(self, width=48, height=12, wrap="none",
font=("Helvetica", 10), insertbackground="#ffeecc", bg="#ffeecc")
        self.scrolled_text.grid()

    # This method updates the content of the ScrolledText widget with the provided
video list
    def update_list(self, video_list):
        self.scrolled_text.delete("1.0", tk.END)
        self.scrolled_text.insert(1.0, video_list)

# Define the VideoDetails class, responsible for displaying the details of a specific
video
class VideoDetails(tk.Frame):
    def __init__(self, master=None, **kwargs):
            super().__init__(master, **kwargs)
            self.grid(sticky="NW", padx=10, pady=10)
            self.configure(bg="#ffeecc")

            # Create a Text widget for displaying video details with a lighter
background color
            self.text_widget = tk.Text(self, width=24, height=4, wrap="none",
font=("Helvetica", 10),
                                        insertbackground="#ffeecc", bg="#ffeecc")
            self.text_widget.grid()

    def update_details(self, details):
        # Update the content of the Text widget with the provided video details
        self.text_widget.delete("1.0", tk.END)
        self.text_widget.insert(1.0, details)

# Defines the StatusBar class, responsible for displaying status messages
class StatusBar(tk.Frame):
    def __init__(self, master=None, **kwargs):
        super().__init__(master, **kwargs)
        self.grid(sticky="W", padx=10, pady=10)
        self.configure(bg="#deeff5")


        # Create a Label widget for displaying status messages
        self.message = tk.StringVar()
        self.message_label = tk.Label(self, textvariable=self.message,
font=("Helvetica", 10))
        self.message_label.grid()

    def update_status(self, status):
        # Update the content of the Label widget with the provided status message
        self.message.set(status)

# Define the CheckVideos class, which is the main application class
class CheckVideos(tk.Frame):
    def __init__(self, master=None, **kwargs):
        super().__init__(master, **kwargs)
        self.grid()
        self.configure(bg="#deeff5")


        self.create_widgets()
        self.list_videos_clicked()

    # Function to create and place all the widgets on the main application window
    def create_widgets(self):

        # Create a Label widget for "Show All Videos"
        self.show_all_videos_lbl = tk.Label(self, text="Video Index",
font=("Helvetica", 10), bg="#deeff5")
        self.show_all_videos_lbl.grid(row=0, column=0, padx=10, pady=10)

        # Create a Label widget for the video number input prompt
        self.enter_lbl = tk.Label(self, text="Enter Video Number :", font=("Helvetica",
10),bg="#deeff5")
        self.enter_lbl.grid(row=0, column=1, padx=10, pady=10)

        # Create a StringVar object to store the video number input
        self.video_num_var = tk.StringVar()

        # Add a trace to the StringVar object to automatically update the video details
when the input changes
        self.video_num_var.trace_add("write", self.update_video_details)

        # Create an Entry widget for video number input
        self.input_txt = tk.Entry(self, width=3, textvariable=self.video_num_var,
font=("Helvetica", 10),bg="#ffeecc")
```

```
        self.input_txt.grid(row=0, column=2, padx=10, pady=10)

        #Code was commeneted out because of the live search function (Create a Button
widget for checking a specific video's details
        #self.check_video_btn = tk.Button(self, text="Check Video", font=("Helvetica",
10), command=self.update_video_details)
        #self.check_video_btn.grid(row=0, column=3, padx=10, pady=10)

        # Create a Button widget for updating a video's rating
        self.update_video_btn = tk.Button(self, text="Update Video", font=("Helvetica",
10), command=self.update_video, bg="#eebfd5", fg="black")
        self.update_video_btn.grid(row=0, column=4, padx=10, pady=10)

        # Create a Label widget for the new rating input prompt
        self.enter_lbl2 = tk.Label(self, text="Enter New Rating :", font=("Helvetica",
10), bg="#deeff5", )
        self.enter_lbl2.grid(row=0, column=5, padx=10, pady=10)

        # Create an Entry widget for new rating input
        self.input_rating_txt = tk.Entry(self, width=3, font=("Helvetica",
10),bg="#ffeecc")
        # Place the new rating input Entry widget on the grid
        self.input_rating_txt.grid(row=0, column=6, padx=10, pady=10)

        # Create instances of VideoList, VideoDetails, and StatusBar classes
        self.video_list = VideoList(master=self)
        self.video_details = VideoDetails(master=self)
        self.status_bar = StatusBar(master=self)

    # Function to update video details based on the input video number
    def update_video_details(self, *args, **kwargs):
        key = self.video_num_var.get()
        if not key.isdigit():
            self.status_bar.update_status("invalid video format")
            return
        name = lib.get_name(key)
        if name is not None:
            director = lib.get_director(key)
            rating = lib.get_rating(key)
            play_count = lib.get_play_count(key)
            video_details = f"{name}\nDirector: {director}\nRating: {rating}\nPlay
Count: {play_count}"
            self.video_details.update_details(video_details)
            self.status_bar.update_status("")
        else:
            self.status_bar.update_status("Invalid video number")
    # Function to display the list of all videos
    def list_videos_clicked(self):
        video_list = lib.list_all()
        self.video_list.update_list(video_list)

    # Function to update a video's rating
    def update_video(self):
        key = self.video_num_var.get()
        if not key.isdigit():
            self.status_bar.update_status("Invalid video format")
            return
        new_rating = self.input_rating_txt.get()
        # Validate and convert the new rating input to an integer
        try:
            new_rating = int(new_rating)
        except ValueError:
            self.status_bar.update_status("Invalid rating value")
            return
        # Update the video's rating and display the updated information
        if lib.update_rating(key, new_rating):
            self.status_bar.update_status("Video updated successfully")
            self.list_videos_clicked()  # Update the video list to reflect the change
in rating
            self.update_video_details(None)
        else:
            self.status_bar.update_status("Invalid video number or rating")

# Create and run the main application

if __name__ == "__main__":
    root = tk.Tk()
    root.title("Video Library")
    app = CheckVideos(master=root)
    app.mainloop()
```