COMP1752 COURSEWORK

Video Player

HO SY MINH HA
001306469

Video Player

Table of Contents

Introduction

This is a mini VideoPlayer project built entirely in python. The project aims to provide a user-friendly interface for interacting with video, offering features such as check video's information, create & play a playlist and update video's information.

The goal of the project is to show the understanding and implementing different python Libraries such as Tkinter for UI, tkVideoPlayer for video processing and sqlite3 for information storage. Also, it requires various OOP concepts to implement smoothly.

Design

Overview

A feature-rich multimedia tool, the Python Video Player is intended to offer a smooth and engaging video-watching experience. This project, which was created in Python, uses strong libraries like tkVideoPlayer for video processing and Tkinter for the GUI to provide a feature-rich video player.

Numerous features that improve the user's control over their multimedia material are supported by the Python Video Player. Among these characteristics are:

1. Video information: The application allows users to view and update video's information such as name, rating, author or video path.
2. Playlist creation: User can crate their own playlist and play it in order.
3. Playback controls: The application offers standard video controls such as play, pause, and select video.

This project is a great way to experiment with Python's multimedia processing capabilities. It provides a useful application of Python programming fundamentals, which makes it an invaluable educational tool for educators, students, and hobby programmers.

Project Structure

Directory Structure

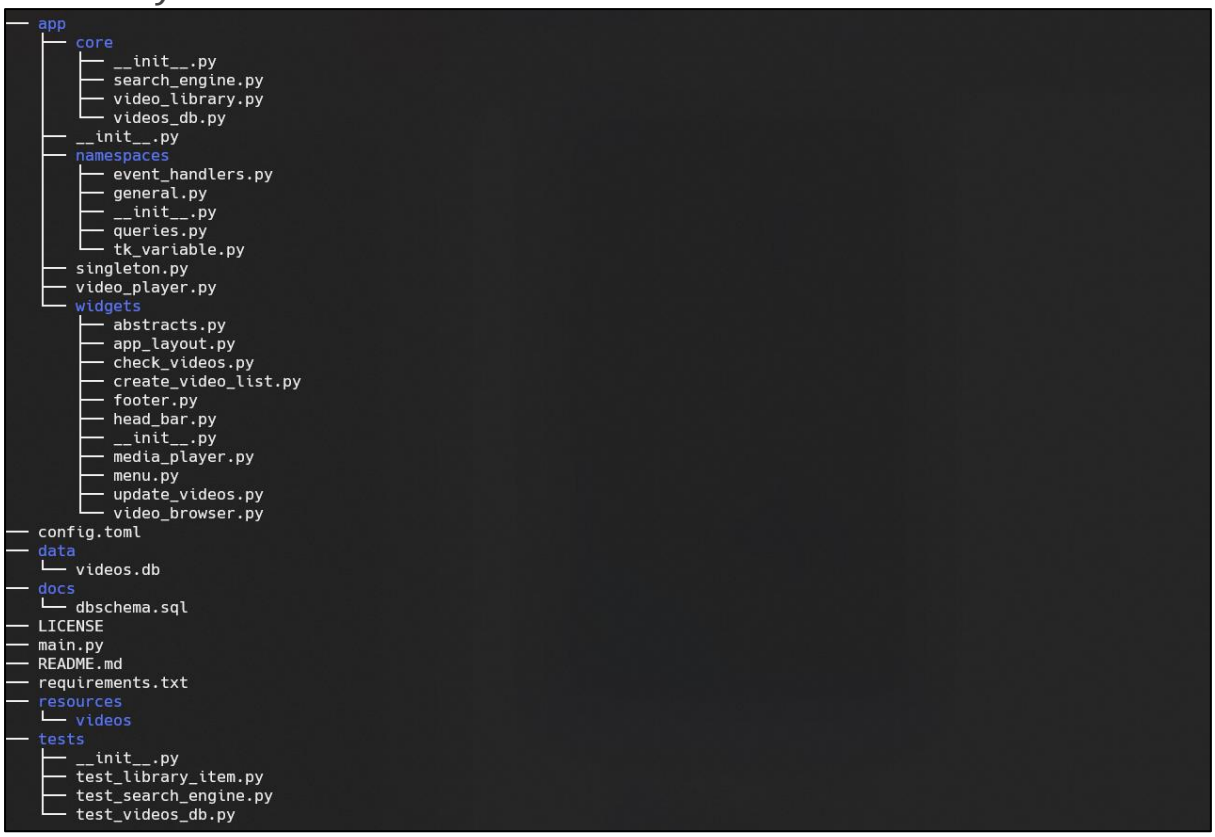


Figure 1 - Project Structure

- main.py: this is the main entry point of the application. It's where the instance of VideoPlayer class is created and start the eventloop.
- /app: this package contains main application code
    - Video_player.py: the main application class, inherited from tk.Tk
    - singleton.py: contains singleton metaclass
    - /core: this is the subpackage contains core classes such as database connection (videos_db.py)
    - /widgets: this subpackage contains all widgets used in application, each class should inherit from tk.Widget, tk.Frame or tk.Toplevel
    - /namespaces: as its name this subpackage contains namespaces, is where the global variables (such as data fetched from database) and functions (such as event_handlers) are stored as it's used by multiple factors
- config.toml: the config file
- /data: contains data that the application use of yield
- /resources: contains resources such as videos or images
- /docs: contains documents such as database schema
- /tests: contains application testcases using pytest and unittest modules
- LICENSE: MIT license
- README.md: contains information about the project, like how to install and run it

- Requirements.txt: this file lists the Python packages the project depends on.

External Dependencies

1. Python version: 3.10 (some module doesn't support lower or higher)
2. tkVideoPlayer: a pip module supports play media file
3. pytest and pytest-mock: testing modules
4. toml: a module support operating on TOML format

User Interface (UI) Design

Menu Interface

This is where the user selects the feature to use

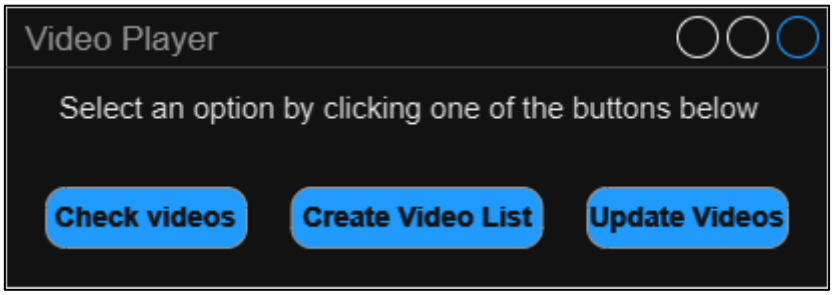


Figure 2 - Menu UI Design

Check Videos Interface

Includes options to check video information and play video

Figure 3 - Check Videos UI Design

Create Video List Interface

Offers feature to create a playlist and play it

Figure 4 - Create Video List Design

Update Video Interface

A feature to update video's information

Figure 5 - Update Video Design

Media Player Interface

An interface for play a playlist

Figure 6 - Media Player UI Design

Development

Stage 1: Basic Understanding

The project has two main kinds of windows, one of which is the window that will display the video and play it, another is for required functions such as Check Videos, Create Video List and Update Videos.

Singleton

The project highly utilizes the singleton concept, as there are similarities between Windows, that each widget which is designed to re-use across the whole project will be marked as singleton.

```
class SingletonMeta(type):
    """Singleton meta class

    If any class set this class as metaclass, it will be restricted to one-instance class
    """

    __instance = None

    def __call__(cls, *args, **kwargs):
        if not cls.__instance:
            cls.__instance = super().__call__(*args, **kwargs)
        return cls.__instance
```

Figure 7 SingletonMeta (singleton.py) implementation

Layout

Figure 8 - Layout idea

The only different thing between windows is the right panel, so check_videos.py, create_videos_list.py and update_videos.py will contains corresponding panels

Most of the widgets will inherit from AppFrame (abstracts.py) abstract class

```python
class AppFrame(tk.Frame):
    """Base class for every Tkinter Frames in the application"""

    def __init__(self, root):
        super().__init__(root)

        self._create_widgets()
        self._display_widgets()

        for children in self.winfo_children():
            children.grid(padx=7, pady=7)

    def _create_widgets(self):
        """Initialize all children widgets"""

        raise NotImplementedError(
            '_create_widgets: expected to be implemented'
        )

    def _display_widgets(self):
        """Display all created widgets"""

        raise NotImplementedError(
            '_display_widgets: expected to be implemented'
        )
```

Figure 9 - AppFrame (abstracts.py) implementation

The main idea is to use single frame with the same layout (MainLayout class) for all three windows Check Videos, Create Video List, Update Video.

```python
"""This module contains application's main layout"""

import tkinter as tk
from tkinter import ttk

from ..singleton import SingletonMeta
from .head_bar import HeadBar
from .video_browser import VideoBrowser
from .footer import Footer
from .abstracts import AppFrame


class MainLayout(AppFrame, metaclass=SingletonMeta):
    """Main layout used in multiple windows
    Layout:
        3 rows
        2 columns
    Widgets (rows, columns):
        Head bar: (0, 0)
        Browser: (1, 0)
        Right Panel: (0-1, 1)
        Footer: (2, 0-1)
    """

    def __init__(self, root):
        super().__init__(root)

        self.columnconfigure(0, weight=7)
        self.columnconfigure(1, weight=3)
        self.rowconfigure(0, weight=1)
        self.rowconfigure(1, weight=8)
        self.rowconfigure(2, weight=1)

    def _create_widgets(self):
        self._head_bar = None
        self._rpanel = None  # Right panel
        self._footer = None
        self._browser = None  # i.e. Video Browser

    def _display_widgets(self):
        pass
```

```python
    def __relayout(self):
        """Re-grid all components"""

        self._head_bar.grid(row=0, column=0, sticky='nsew')
        self._browser.grid(row=1, column=0, sticky='nsew')
        self._rpanel.grid(row=0, column=1, rowspan=2, sticky='nsew')
        self._footer.grid(row=2, column=0, columnspan=2, sticky='nsew')
        for children in (
            self._head_bar,
            self._browser,
            self._rpanel,
            self._footer,
        ):
            children.grid(padx=5, pady=5)
        pass

    def display(self, head_bar=None, browser=None, rpanel=None, footer=None):
        """Remove previous widgets and apply new widgets"""

        if self._head_bar:
            self._head_bar.grid_forget()
        if self._browser:
            self._browser.grid_forget()
        if self._rpanel:
            self._rpanel.grid_forget()
        if self._footer:
            self._footer.grid_forget()
        self._head_bar = head_bar or HeadBar()
        self._browser = browser or VideoBrowser()
        self._rpanel = rpanel
        self._footer = footer or Footer()
        self.__relayout()
        self.grid(row=0, column=0, sticky=tk.NSEW)
```

Figure 10 MainLayout (app_layout.py) implementation

Note: in the MainLayout#display method, there is four parameters in total for future development but currently it mainly uses the rpanel parameter.

Check Videos

```python
"""This module contains class for Check Videos window"""

import tkinter as tk
from tkinter import ttk
from collections.abc import Sequence

from ..singleton import SingletonMeta
from ..namespaces.tk_variable import TkVariable
from ..namespaces.event_handlers import EventHandlers
from ..core.video_library import LibraryItem
from .abstracts import AppFrame, InfoText


class CheckVideosPanel(AppFrame, metaclass=SingletonMeta):
    COLUMNS = (1, 2, 3, 4, 5)  # Database columns indexes
    HEADINGS = tuple(
        LibraryItem.HEADINGS[col] for col in COLUMNS
    )  # Fetch headings by columns index

    def __init__(self, root):
        super().__init__(root)

        self.columnconfigure(0, weight=1)
        self.columnconfigure(1, weight=2)

    def _create_widgets(self):
        self.__texts = tuple(
            InfoText(self) for _ in range(len(self.COLUMNS))
        )  # Informations text fields
        self.__id_input = ttk.Entry(
            self, textvariable=TkVariable().selected_id
        )  # Separated id input
        self.__check_btn = ttk.Button(
            self, text='Check Videos', width=20, command=self.__display_info
        )  # Display selected video's information when clicked
        self.__play_btn = ttk.Button(
            self, text='Play', width=20, command=EventHandlers().play_video
        )  # Play selected video
```

```python
    def _display_widgets(self):
        ttk.Label(self, text='Video Infomation').grid(
            row=0, column=0, columnspan=2, sticky='ns'
        )
        ttk.Label(self, text=LibraryItem.HEADINGS[0]).grid(
            row=2, column=0, sticky='w'
        )
        self.__id_input.grid(row=2, column=1, ipady=3, sticky='we')
        self.__play_btn.grid(row=14, column=0, columnspan=2, sticky='wns')
        self.__check_btn.grid(row=14, column=1, sticky='ens')

        for idx in range(0, 2 + len(self.COLUMNS)):
            # Display all horizontal separators
            row = 2 * idx + 1
            ttk.Separator(self, orient='horizontal').grid(
                row=row, column=0, columnspan=2, sticky='nsew'
            )

        for idx, (attr, text) in enumerate(zip(self.HEADINGS, self.__texts)):
            # Display all headings and correspond information text field
            row = 2 * (idx + 2)
            ttk.Label(self, text=attr).grid(row=row, column=0, sticky='w')
            text.grid(row=row, column=1, ipady=3, sticky='we')
```

Figure 11 - CheckVideosPanel (check_videos.py) implementation without functionalities

Stage 2: Outline implementation – GUI

Video player – application master

The main frame and menu frame will be managed by VideoPlayer master

```python
"""This module contains root of all widgets in the application"""

import sys
import traceback
import tkinter as tk
from tkinter import ttk

from .core.videos_db import VideosDB
from .core.video_library import LibraryItemCollection
from .widgets.app_layout import MainLayout
from .widgets.video_browser import VideoBrowser
from .widgets.footer import Footer
from .widgets.head_bar import HeadBar
from .widgets.check_videos import CheckVideosPanel
from .widgets.create_video_list import CreateVideoListPanel
from .widgets.update_videos import UpdateVideoPanel
from .widgets.menu import Menu
from .widgets.media_player import MediaPlayer


class VideoPlayer(tk.Tk):
    """Root class"""

    def __init__(self):
        super().__init__()

        self.__curr_frame = None  # current displaying frame
        self.__frames = {}  # frames call information

        self.title('Video Player')
        self.columnconfigure(0, weight=1)
        self.rowconfigure(0, weight=1)
        self.resizable(False, False)
        self.__create_widgets()
        self.display_frame('menu')  # display the start menu
```

```python
    def display_frame(self, frame):
        """Change to the specific frame
        Args:
            frame - name of the frame to switch
        """
        try:
            frame, kwargs = self.__frames[frame]

        except KeyError as e:
            print(f'frame not found: {frame}', file=sys.stderr)
            traceback.print_stack(file=sys.stderr)
        else:
            if self.__curr_frame is not None:
                self.__curr_frame.grid_forget()  # Hide current frame
            self.__curr_frame = frame
            self.__curr_frame.display(**kwargs)

    def __create_widgets(self):
        # Create all singleton widgets
        self.__main_layout = MainLayout(self)
        MediaPlayer(self)
        Menu(self)
        VideoBrowser(self.__main_layout)
        HeadBar(self.__main_layout)
        Footer(self.__main_layout)
        CheckVideosPanel(self.__main_layout)
        UpdateVideoPanel(self.__main_layout)
        CreateVideoListPanel(self.__main_layout)

        # Store all frame invoking information
        self.__frames['menu'] = (Menu(), {})
        self.__frames['check_videos'] = (
            self.__main_layout,
            {'rpanel': CheckVideosPanel()},
        )
        self.__frames['update_videos'] = (
            self.__main_layout,
            {'rpanel': UpdateVideoPanel()},
        )
        self.__frames['create_video_list'] = (
            self.__main_layout,
            {'rpanel': CreateVideoListPanel()},
        )
```

Figure 12 - VideoPlayer (video_player.py) implementation

Layout Components

```python
class Menu(AppFrame, metaclass=SingletonMeta):
    def __init__(self, root):
        super().__init__(root)

        # Configuring layout
        for column in range(3):
            self.columnconfigure(column, weight=1)

    def _create_widgets(self):
        pass

    def _display_widgets(self):
        ttk.Label(
            self, text='Select an option by clicking one of the buttons below'
        ).grid(row=0, column=0, columnspan=3)
        ttk.Button(
            self,
            text='Check Videos',
            command=lambda: self._root().display_frame('check_videos'),
        ).grid(
            row=1, column=0
        )  # Display check videos UI when click
        ttk.Button(
            self,
            text='Create Video List',
            command=lambda: self._root().display_frame('create_video_list'),
        ).grid(
            row=1, column=1
        )  # Display create video UI list when click
        ttk.Button(
            self,
            text='Update Videos',
            command=lambda: self._root().display_frame('update_videos'),
        ).grid(
            row=1, column=2
        )  # Display update video UI when click

        for widget in self.winfo_children():
            widget.grid(padx=5, pady=5, sticky='we')

    def display(self):
        """Places self on root"""

        self.grid(row=0, column=0, sticky='nsew')
```

Figure 13 - Menu (menu.py) implementation

```python
class HeadBar(AppFrame, metaclass=SingletonMeta):
    def __init__(self, root):
        super().__init__(root)

        # configuring layout
        for col in range(6):
            self.columnconfigure(col, weight=1)
        self.columnconfigure(6, weight=2, minsize=200)

    def _create_widgets(self):
        self.__list_video_btn = ttk.Button(
            self,
            text='List All Videos',
            command=VideoBrowser().display_playlist,
        )
        self.__sort_by = ttk.OptionMenu(
            self,
            TkVariable().sort_by,
            'ID    ',
            *('ID    ', 'Name  ', 'Author', 'Rating'),
            direction='below'
        )
        self.__sort_order = ttk.OptionMenu(
            self,
            TkVariable().sort_order,
            'Ascending ',
            *('Ascending ', 'Descending')
        )
        self.__search_bar = tk.Entry(
            self, textvariable=TkVariable().search_entry
        )

    def _display_widgets(self):
        ttk.Label(self, text='Sort by ').grid(row=0, column=1, sticky='e')
        ttk.Label(self, text='Sort order ').grid(row=0, column=3, sticky='e')
        ttk.Label(self, text='Search ').grid(row=0, column=5, sticky='e')

        self.__list_video_btn.grid(row=0, column=0, sticky='w')
        self.__sort_by.grid(row=0, column=2, sticky='w')
        self.__sort_order.grid(row=0, column=4, sticky='w')
        self.__search_bar.grid(row=0, column=6, sticky='we')
```

Figure 14 - Head bar (head_bar.py) implementation

```python
class Footer(ttk.Frame, metaclass=SingletonMeta):
    def __init__(self, root):
        super().__init__(root)

        self.columnconfigure(0, weight=1)
        self.columnconfigure(1, weight=1)

        self.__back_btn = ttk.Button(
            self, text='Back', command=self.__back
        )  # Back to menu when clicked

        ttk.Label(self, text='Version 2.0').grid(row=0, column=1, sticky='e')
        self.__back_btn.grid(row=0, column=0, sticky='w')

    def __back(self):
        """Back to menu"""
        self._root().display_frame('menu')
```

Figure 15 – Footer (footer.py) implementation

```python
class VideoBrowser(AppFrame, metaclass=SingletonMeta):
    COLUMNS = (0, 1, 2, 3)  # database column indexes

    def __init__(self, root):
        super().__init__(root)

        for children in self.winfo_children():
            children.grid(padx=0)

    def _create_widgets(self):
        columns_width = (50, 300, 300, 150)
        self.__browser = ttk.Treeview(
            self, show='headings', height=20
        )  # All videos will be displayed here
        self.__scrollbar = ttk.Scrollbar(
            self, orient='vertical', command=self.__browser.yview
        )  # Video Browser's scrollbar
        self.__browser.config(yscrollcommand=self.__scrollbar.set)
        self.__browser['columns'] = tuple(
            VideosDB.COLUMNS[column] for column in self.COLUMNS
        )  # Set browser columns
        for column in self.COLUMNS:
            # Config browser
            self.__browser.heading(
                VideosDB.COLUMNS[column], text=LibraryItem.HEADINGS[column]
            )
            self.__browser.column(
                VideosDB.COLUMNS[column], width=columns_width[column]
            )
        # called when an item in VideoBrowser is selected
        self.__browser.bind('<<TreeviewSelect>>', self.__item_selected)

    def _display_widgets(self):
        self.__browser.grid(row=0, column=0, sticky='nsew')
        self.__scrollbar.grid(row=0, column=1, sticky='nsew')
```

Figure 16 - Video Browser (video_browser.py) implementation without functionalities

```python
class InfoText(tk.Text):
    """Base class for Text which are designed only for containing information"""

    def __init__(self, root):
        super().__init__(root, height=1, width=35)

        self[
            'state'
        ] = 'disabled'  # Infomation texts are not allowed to change

    def display(self, value: str) -> None:
        """Display information into Textbox

        Args:
            value: an information string to be displayed

        Returns:
            None
        """

        self['state'] = 'normal'
        self.delete('1.0', tk.END)  # Remove all contents
        self.insert('1.0', value)  # Display new contents
        self['state'] = 'disabled'
```

Figure 17 - InfoText (abstracts.py) implementation

Note: This class is just a normal text field but designed to be read-only

`App frames`

```python
class CreateVideoListPanel(AppFrame, metaclass=SingletonMeta):
    COLUMNS = (1,)  # database column indexes
    HEADINGS = tuple(
        LibraryItem.HEADINGS[col] for col in COLUMNS
    )  # get corresponded column headings

    def __init__(self, root):
        super().__init__(root)

        # Name field will be updated when selected_id changes
        TkVariable().selected_id.trace_add('write', self.__display_name)

        self.columnconfigure(0, weight=2)
        self.columnconfigure(1, weight=3)
```

```python
  def _create_widgets(self):
      self.__playlist_fr = ttk.Frame(self)  # Frame for playlist listbox
      self.__playlist = tk.Listbox(
          self.__playlist_fr, width=45
      )  # Listbox contains added videos
      self.__sb = ttk.Scrollbar(
          self.__playlist_fr, orient='vertical'
      )  # playlist listbox
      self.__id_entry = ttk.Entry(
          self, textvariable=TkVariable().selected_id
      )  # Selected video_id entry
      self.__texts = tuple(
          InfoText(self) for _ in range(len(self.HEADINGS))
      )  # All information text field
      self.__add_btn = ttk.Button(
          self,
          text='Add',
          width=20,
          command=self.display_playlist(
              EventHandlers().add_selected_to_playlist
          ),
      )  # Video will be added to be playlist when clicked
      self.__remove_btn = ttk.Button(
          self,
          text='Remove',
          width=20,
          command=self.display_playlist(
              EventHandlers().remove_selected_from_playlist
          ),
      )  # Video will be removed from playlist when clicked
      self.__play_btn = ttk.Button(
          self,
          text='Play playlist',
          command=self.display_playlist(EventHandlers().play_playlist),
      )  # Playlist will be played when clicked

      self.__playlist.config(yscrollcommand=self.__sb.set)  # Set Scrollbar
      self.__sb.config(command=self.__playlist.yview)  #
```

```python
def _display_widgets(self):
    ttk.Label(self, text='Create video list').grid(
        row=0, column=0, columnspan=2
    )
    ttk.Label(self, text='ID').grid(row=4, column=0, sticky='w')
    for idx in range(0, 3 + len(self.COLUMNS)):
        row = 2 * idx + 1
        ttk.Separator(self, orient='horizontal').grid(
            row=row, column=0, columnspan=2, sticky='nsew'
        )
    self.__playlist_fr.grid(row=2, column=0, columnspan=2)
    self.__id_entry.grid(row=4, column=1, ipady=3, sticky='nsew')

    for idx, (attr, text) in enumerate(zip(self.HEADINGS, self.__texts)):
        # Display all field and corresponded heading
        row = 2 * (idx + 3)
        ttk.Label(self, text=attr).grid(row=row, column=0, sticky='w')
        text.grid(row=row, column=1, ipady=3, sticky='nsew')

    self.__add_btn.grid(column=0, row=8, columnspan=2, sticky='w')
    self.__remove_btn.grid(column=1, row=8, sticky='e')
    self.__play_btn.grid(column=0, columnspan=2, sticky='we')
    self.__playlist.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
    self.__sb.pack(side=tk.RIGHT, fill=tk.Y)
```

Figure 18 - Create Video List Panel (create_video_list.py) implementation without functionalities

```python
class UpdateVideoPanel(AppFrame, metaclass=SingletonMeta):
    COLUMNS = (1, 2, 3, 5)  # database column indexes
    HEADINGS = tuple(
        LibraryItem.HEADINGS[col] for col in COLUMNS
    )  # get corresponded column heading

    def __init__(self, root):
        super().__init__(root)

        # Attach new event, called when new video_id is selected
        TkVariable().selected_id.trace_add('write', self.__display_info)

        self.columnconfigure(0, weight=2)
        self.columnconfigure(1, weight=3)

    def _create_widgets(self):
        self.__vars = (
            tk.StringVar(),
            tk.StringVar(),
            tk.DoubleVar(),
            tk.StringVar(),
        )  # corresponded variable for updating entries
        self.__id_input = ttk.Entry(
            self, width=35, textvariable=TkVariable().selected_id
        )  # video_id input
        self.__entries = tuple(
            ttk.Entry(self, width=35, textvariable=var) for var in self.__vars
        )  # Updating entries
        self.__update_btn = ttk.Button(
            self,
            text='Update',
            width=20,
            command=lambda: EventHandlers().update_video(
                self.COLUMNS, self.__vars
            ),
        )  # Update video information when clicked
```

```python
    def _display_widgets(self):
        ttk.Label(self, text='Update Video').grid(
            row=0, column=0, columnspan=2
        )
        ttk.Label(self, text='ID').grid(row=2, column=0, sticky='w')
        for idx in range(0, 2 + len(self.COLUMNS)):
            # Display all separators
            row = idx * 2 + 1
            ttk.Separator(self, orient='horizontal').grid(
                row=row, column=0, columnspan=2, sticky='nsew'
            )
        for idx, (attr, entry) in enumerate(
            zip(self.HEADINGS, self.__entries)
        ):
            # Display all updating entries
            row = 2 * (idx + 2)
            ttk.Label(self, text=attr).grid(row=row, column=0, sticky='w')
            entry.grid(row=row, column=1, ipady=3, sticky='e')

        self.__id_input.grid(row=2, column=1, ipady=3, sticky='e')
        self.__update_btn.grid(row=15, column=1, sticky='e')
```

Figure 19 - Update Video Panel (update_video.py) implementation without functionalities

Stage 3: Basic working version

Configuration file

Configuration file (config.toml) is powered by *toml, as it's human readable and easy to refer to*

```toml
[database]
path.db = 'data/videos.db'
path.schema = 'docs/dbschema.sql'
```

Figure 20- Configuration (config.toml)

```python
# Load config
CONFIG = MappingProxyType(
    toml.load(
        Path(__file__).parent.parent / 'config.toml',
    )
)  # Using MappingProxyType to prevent change to config
```

Figure 21 - Config (app/__init__.py) is read as a global variable

Storage

Sqlite3 is used as a storage of videos information, as it is a built-in module and easy to implement

```python
"""This module contains namespace of database query strings"""

from string import Template
from dataclasses import dataclass


@dataclass(frozen=True)
class Queries:
    UPDATE = Template('UPDATE $table SET $column = ? WHERE $filter_col = ?')
    SELECT_ALL = Template('SELECT * FROM $table')
    SELECT_TABLE = (
        "SELECT name FROM sqlite_master WHERE type='table' AND name = ?"
    )
```

Figure 22 - SQL Queries (queries.py) is managed in a separated namespace

```python
"""This module contains Database connection class"""

import sqlite3 as sql
from pathlib import Path

from ..namespaces.queries import Queries
from ..singleton import SingletonMeta
from .. import CONFIG

CONFIG = CONFIG['database']
```

```python
class VideosDB(metaclass=SingletonMeta):
    COLUMNS = (
        'video_id',
        'name',
        'director',
        'rating',
        'play_count',
        'file_path',
    )  # all columns in database

    TABLE = 'videos'

    def __init__(self, db_path=None):
        db_path = db_path or Path(
            CONFIG['path']['db']
        )  # get database path from config
        if not db_path.exists():
            db_path.parent.mkdir(parents=True, exist_ok=True)
            db_path.touch()
        self.__conn = sql.connect(db_path)  # Create database connection
        self.__cursor = self.__conn.cursor()  # fetch database cursor
        self.__ensure_db()  # makesure database exists

    @property
    def cursor(self):
        return self.__cursor

    def close(self):
        """Close database connection"""

        self.__conn.commit()
        self.cursor.close()
        self.__conn.close()
```

```python
    def update(self, id: int, column: str, val: str | int) -> None:
        """Update database data
        Args:
            id - video_id
            column - column to update
            val - new value
        """
        try:
            self.cursor.execute(
                Queries.UPDATE.safe_substitute(
                    table=self.TABLE, column=column, filter_col='video_id'
                ),
                (val, id),
            )
        except sql.OperationalError as e:
            msgbox.showerror("Database error", message=e)

    def get_all(self) -> tuple:
        """Return database data"""

        ret = None
        try:
            self.__conn.commit()
            self.cursor.execute(
                Queries.SELECT_ALL.safe_substitute(table=self.TABLE)
            )
            ret = tuple(self.cursor.fetchall())
        except sql.OperationalError as e:
            msgbox.showerror("Database error", message=e)

        return ret

    def __ensure_db(self):
        """Makesure database exists"""

        if self.__cursor is None:
            # if there is no cursor
            return

        # Check table existance
        self.__cursor.execute(Queries.SELECT_TABLE, (self.TABLE,))

        if self.__cursor.fetchone() is not None:
            # If the table exists
            return

        with open(CONFIG['path']['schema'], 'r') as f:
            # Init database by database schema
            self.cursor.executescript(f.read())
```

Figure 23 - VideosDB (videos_db.py) implementation

Video Object

Video Library

This object contains a single video information

```python
class LibraryItem:
    """The class represents the video"""

    HEADINGS = (
        'ID',
        'Name',
        'Director',
        'Rating',
        'Play Count',
        'File Path',
    )  # heading corresponded to database columns

    def __init__(
        self,
        id: int,
        name: str,
        director: str,
        rating: float,
        play_count: int,
        path: str,
    ):
        values = (
            int(id),
            str(name),
            str(director),
            float(rating),
            int(play_count),
            str(path),
        )
        self.__data = {
            key: val for key, val in zip(VideosDB.COLUMNS, values)
        }  # store data by id

    def list_all(
        self, attrs: Sequence[str | int] = VideosDB.COLUMNS
    ) -> tuple[int | str]:
        """List attributes by columns
        Args:
            attrs - columns to list
        Returns:
            tuple of attributes
        """

        return tuple(self[attr] for attr in attrs)
```

```python
    def get_id(self) -> int:
        return self[0]

    def get_name(self) -> str:
        return self[1]

    def get_director(self) -> str:
        return self[2]

    def get_rating(self) -> int:
        return self[3]

    def get_play_count(self) -> int:
        return self[4]

    def get_file_path(self) -> str:
        return self[5]

    def increment_play_count(self) -> None:
        self[4] += 1

    def set_name(self, name: str) -> None:
        self[1] = str(name)

    def set_director(self, director: str) -> None:
        self[2] = str(director)

    def set_rating(self, rating: float) -> None:
        self[3] = float(rating)

    def set_file_path(self, file_path: str) -> None:
        self[5] = str(file_path)

    def __contains__(self, item: str | int):
        """Check if an attribute exists"""

        if isinstance(item, int):
            item = VideosDB.COLUMNS[item]
        return item in self.__data
```

```python
    def __getitem__(self, item: str | int) -> str | int:
        """Get item by column or index"""

        if isinstance(item, int):
            item = VideosDB.COLUMNS[item]
        if item not in self:
            raise AttributeError(f'invalid attribute: {item}')
        return self.__data[item]

    def __setitem__(self, item: str | int, new_val: str | int) -> None:
        """Set value by column or index"""

        if isinstance(item, int):
            item = VideosDB.COLUMNS[item]
        if item not in self:
            raise AttributeError('can\'t assign new attribue')
        self.__data[item] = type(self[item])(new_val)
        VideosDB().update(self.get_id(), item, self[item])
```

Figure 24 - LibraryItem (video_library.py) implementation

`LibraryItemCollection`

This object contains a list or a collection of videos

```python
class LibraryItemCollection:
    def __init__(self, videos: Sequence[LibraryItem] = None):
        if videos is None:
            videos = dict()
        videos = {video.get_id(): video for video in videos}
        self.__videos = videos

    @classmethod
    def from_sequences(cls, videos: Sequence[Sequence]):
        """Alternative class constructor
        Args:
            Videos data by columns
        Returns:
            new class instance
        """
        videos = (LibraryItem(*video) for video in videos)
        return cls(videos)

    def add(self, video: LibraryItem) -> None:
        self.__videos[video.get_id()] = video

    def remove(self, id: int) -> None:
        del self.__videos[id]

    def values(self):
        return self.__videos.values()

    def __getitem__(self, video_id) -> LibraryItem:
        return self.__videos[video_id]

    def __iter__(self):
        return iter(self.values())

    def __contains__(self, id):
        return id in self.__videos

    def __bool__(self):
        return bool(self.__videos)
```

Figure 25 - LibraryItemCollection (video_library.py) implementation

Tkinter Variables

As tkinter variables is used by multiple widgets, it is reasonable to put them in a global namespace TkVariable.

There are multiple frequently used operations on tkinter variables, it is a good point to implement it as methods such as get_selected_id, as this method including input validation which is required before the data is used in other operations.

```python
class TkVariable(metaclass=SingletonMeta):
    """Namespace contains variable, and method to access its values

    The purpose of @porperty is for readability and preventing changes
    """

    def __init__(self):
        self.__selected_id = tk.IntVar()  # value of all id entries
        self.__search_entry = tk.StringVar()  # value of search bar
        self.__sort_by = tk.StringVar()  # value of sorting selection
        self.__sort_order = tk.StringVar()  # value of sort order

    def get_selected_id(self, display_msg=True):
        """Get the current selected id
        Args:
            display_msg - a boolean to decide if a messagebox is display when the id is invalid

        Returns:
            id - integer selected id
        or
            none if id is invalid
        """

        try:
            id = self.selected_id.get()  # get value of tkvar
            if id not in General().data:  # check validity
                raise KeyError('Invalid ID')
        except Exception as e:
            if not display_msg:
                return
            tk.messagebox.showerror('Id error', message='Invalid ID')
        else:
            return id
```

```python
    def get_search_entry(self):
        """Returns search value"""
        return self.__search_entry.get().strip().lower()

    def get_sort_by(self):
        """Returns sort option"""

        return self.__sort_by.get().strip().lower()

    def get_sort_order(self):
        """Returns sort order"""

        return self.__sort_order.get().strip().lower() == 'descending'

    @property
    def selected_id(self):
        """Return the selected_id variable"""

        return self.__selected_id

    @property
    def search_entry(self):
        """Return the search_entry variable"""

        return self.__search_entry

    @property
    def sort_by(self):
        """Returns the sort_by variable"""

        return self.__sort_by

    @property
    def sort_order(self):
        """Returns the sort_order variable"""

        return self.__sort_order
```

Figure 26 - TkVariable (tk_variable.py) implementation

Event handlers

As the methods handling the events may be used in multiple widgets and namespaces, to avoid circular import and hard-read code, is reasonable to put them in single namespace EventHandlers

The main dataflow is when a button clicked, the corresponding handler is trigger which will get the validated data from TkVariable, handling and return the result if required

```
class EventHandlers:
    """Event handers namespace

    The purpose of @static is for readbility and namespace properties
    """

    @staticmethod
    def get_brower_items() -> None:
        """Returns filtered videos"""

        _prefix = TkVariable().get_search_entry()  # Search prefix
        _data = General().search_engine.search_prefix(
            _prefix
        )  # Filter by search_prefix
        _data = (General().data[id] for id in _data)  # Fetch data
        d = {
            'id': 'video_id',
            'author': 'director',
            'name': 'name',
            'rating': 'rating',
        }  # corresponded sort option
        sort_by = TkVariable().get_sort_by()  # get sort option
        sort_order = TkVariable().get_sort_order()  # get sort direction
        return sorted(
            _data, key=lambda val: val[d[sort_by]], reverse=sort_order
        )  # sort data by sort option and direction

    @staticmethod
    def play_video():
        """Plays the selected video"""

        video = EventHandlers.get_video()
        if not video:
            return
        playlist = LibraryItemCollection((video,))  # Create playlist
        MediaPlayer().play(playlist)

    @staticmethod
    def get_video() -> LibraryItem:
        """Returns current selected video"""

        id = TkVariable().get_selected_id()
        if not id:
            return None
        return General().data[id]
```

```python
    @staticmethod
    def add_selected_to_playlist() -> bool:
        """Adds selected video to the current playlist
        Returns:
            a boolean of action status
        """

        video = EventHandlers.get_video()
        if not video:
            return False
        if video.get_id() in General().play_list:
            msgbox.showerror('Add error', 'This video has been added')
            return False
        General().play_list.add(video)
        return True

    @staticmethod
    def remove_selected_from_playlist() -> bool:
        """Remove current selected videos from the current playlist
        Returns:
            a boolean of action status
        """

        id = TkVariable().get_selected_id()
        if not id:
            return False
        if id not in General().play_list:
            # display error if the video is not in playlist
            msgbox.showerror(
                'Remove error', 'This video is not in playlist!'
            )
            return False
        General().play_list.remove(id)
        return True

    @staticmethod
    def play_playlist() -> bool:
        """Play the current playlist
        Returns:
            a boolean of action status
        """

        if not General().play_list:
            # Show error if the playlist is empty
            msgbox.showerror('Play error', 'Cannot play an empty playlist!')
            return False
        MediaPlayer().play(General().play_list)
        return True
```

```python
    @staticmethod
    def update_video(columns, new_values: Sequence[tk.Entry]):
        """Update video informations
        Args:
            columns - column indexes to be updated
            new_values - tkiter variables corresponded to columns
        """

        video = EventHandlers.get_video()
        if not video:
            return
        try:
            # Try to fetch variables values
            new_values = tuple(item.get() for item in new_values)
        except tk._tkinter.TclError as e:
            msgbox.showinfo('Update error', e)
            return
        if all(val == video[index] for index, val in zip(columns, new_values)):
            # If there is no change
            msgbox.showinfo('Update', 'Nothing to update!')
            return
        if any(not val for val in new_values if isinstance(val, str)):
            # If there are invalid values
            msgbox.showerror('Update error', 'Entry cannot be empty!')
            return

        for col, new_val in zip(columns, new_values):
            if new_val != video[col]:
                # Update if new values if different
                video[col] = new_va
```

Figure 27 - EventHandlers (event_handlers.py) implementation

General namespace

The general namespace is implemented for other normal variables and objects, such as data fetched from database, the current playlist and the search engine object. Note: these methods are only responsible for fetching data, not display data to screen which is handled by internal frame methods

```python
class General(metaclass=SingletonMeta):
    """General purpose namespace

    The purpose of @property is for readbility and preventing changes
    """

    def __init__(self):
        data = VideosDB().cursor.execute('SELECT * FROM videos;')
        self.__data = LibraryItemCollection.from_sequences(data.fetchall())
        data = []
        for item in self.__data:
            data.extend(
                (
                    (item.get_name().lower(), item.get_id()),
                    (item.get_director().lower(), item.get_id()),
                )
            )
        self.__search_engine = SearchEngine(data)
        self.__play_list = LibraryItemCollection()

    @property
    def data(self):
        return self.__data

    @property
    def search_engine(self):
        return self.__search_engine

    @property
    def play_list(self):
        return self.__play_list
```

Figure 28 - General (general.py) implementation

Internal handlers

These methods are implemented directly to each frame, they will call to the correspond event handler, and display the result to the children widgets

Checkvideos

```python
    def __display_info(self):
        """Display all information of a selected video"""

        data = EventHandlers.get_video()  # Get selected video
        if not data:
            # Clear all InfoText there is no selected video
            for text in self.__texts:
                text.display('')
            return

        for text, col in zip(self.__texts, self.COLUMNS):
            text.display(data[col])
```

Figure 29 - Internal handler of CheckVideoPanel (check_videos.py)

CreateVideoList

```python
    def __display_name(self, *ignore):
        """Display video name"""

        id = TkVariable().get_selected_id(display_msg=False)
        if not id:
            # Clear all field and exit if id is invalid
            for text in self.__texts:
                text.display('')
            return
        data = General().data[id]
        for idx, col in enumerate(self.COLUMNS):
            # Display video's information
            self.__texts[idx].display(data[col])

    def display_playlist(self, call_back):
        """Wrappers for class event handlers

        Args:
            call_back: callable - event handler
        Return:
            decorated function
        """

        def __wrapper():
            succ = call_back()  # Action status
            if not succ:
                # If the action failed, return
                return

            # Update playlist if the action (add, remove, play) success
            self.__playlist.delete(0, tk.END)
            self.__playlist.insert(
                tk.END,
                *(
                    f'{item.get_id()} - {item.get_name()}'
                    for item in General().play_list
                ),
            )

        return __wrapper
```

Figure 30 - CreateVideoList (create_video_list.py) internal handlers

Update Videos

```python
    def __display_info(self, *ignore):
        """Display video information"""

        id = TkVariable().get_selected_id(display_msg=False)
        if not id:
            # If id is invalid, reset entries and exit
            for var in self.__vars:
                var.set('' if isinstance(var, tk.StringVar) else 0)
            return
        data = General().data[id]
        for idx, col in enumerate(self.COLUMNS):
            self.__vars[idx].set(data[col])
```

Figure 31 - UpdateVideo (update_videos.py) internal hander

Video Browser

```python
def display_playlist(self) -> None:
    contents = EventHandlers.get_brower_items()  # Get all filtered videos
    # display contents
    self.__clear_contents()
    for item in contents:
        self.__browser.insert(
            '',
            tk.END,
            id=item.get_id(),
            values=item.list_all(self.COLUMNS),
        )

    # focus the first video in video browser
    id = TkVariable().get_selected_id(display_msg=False)
    if not id:
        id = int(self.__browser.get_children()[0])
    if self.__browser.exists(id):
        self.__browser.selection_set(id)
```

Stage 4: Innovation

Search engine

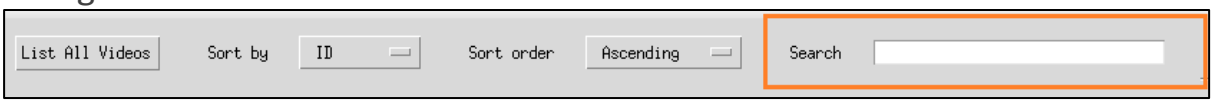


Figure 32 - Search entry placed at headbar

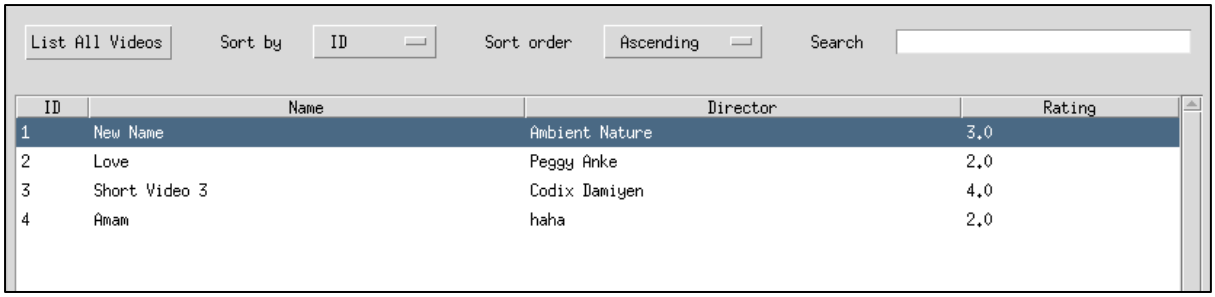
Implementation: The basic search algorithm use hashmap as the main data structure, search entry end return matches ids

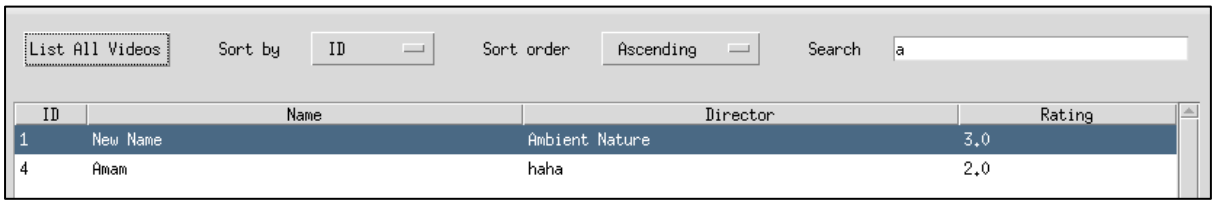
```python
class HashMap:
    """Search algorithm Hashmap"""

    def __init__(self):
        self.__data = dict()

    def insert(self, text, id):
        if text not in self.__data:
            self.__data[text] = set()
        self.__data[text].add(id)

    def search_prefix(self, prefix: str):
        found = set()
        for key, val in self.__data.items():
            if key.startswith(prefix):  # if the value match prefix
                found |= val
        return found


class SearchEngine:
    def __init__(self, data=None, /, *, data_structure=HashMap):
        self.__ds = data_structure()

        if not data:
            return
        for text, index in data:
            self.__ds.insert(text, index)

    def search_prefix(self, prefix):
        """Returns the ids that has name or author match the prefix"""

        return self.__ds.search_prefix(prefix)
```

Figure 33 - SearchEngine (search_engine.py) implementation

When *List all videos* button is clicked, the handlers will automatically call search engine and filter the data

- Test:

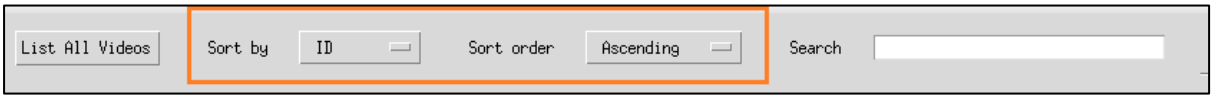




Sorting

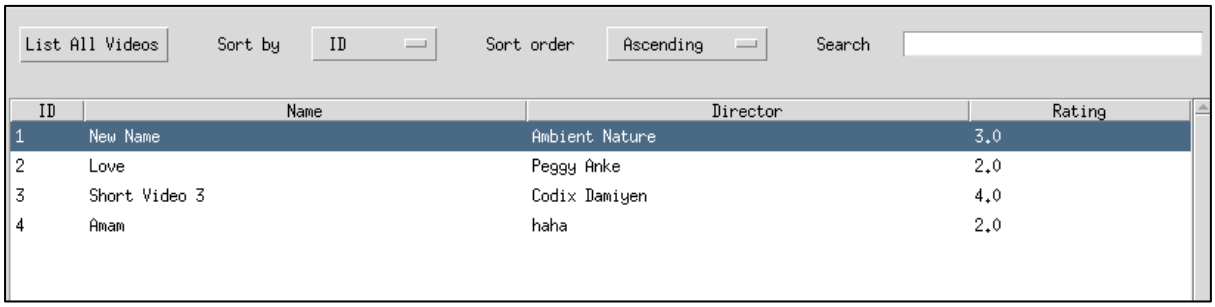
Sort by – field to sort (default: ID)

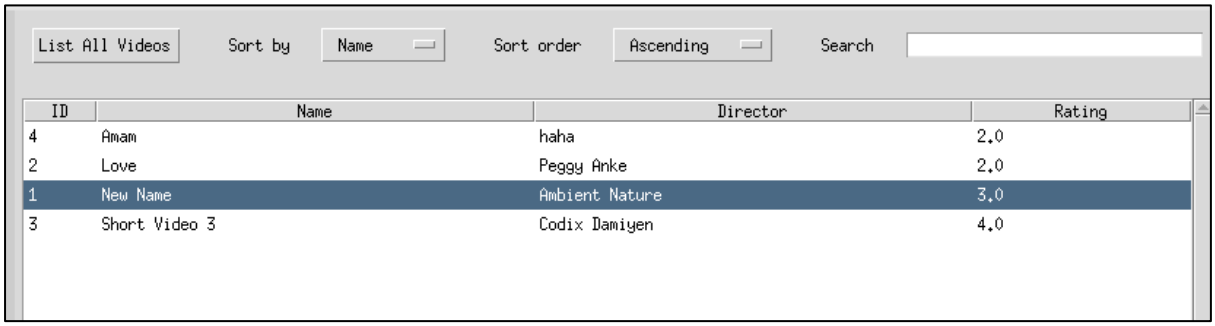
Sort order – Sort direction ascending (default) or descending

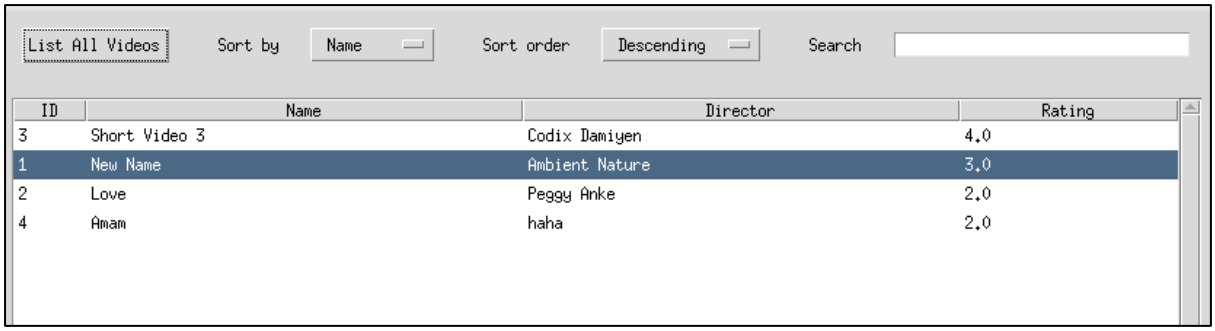


The sort will do its function when ListAllVideos button is clicked

- Test:



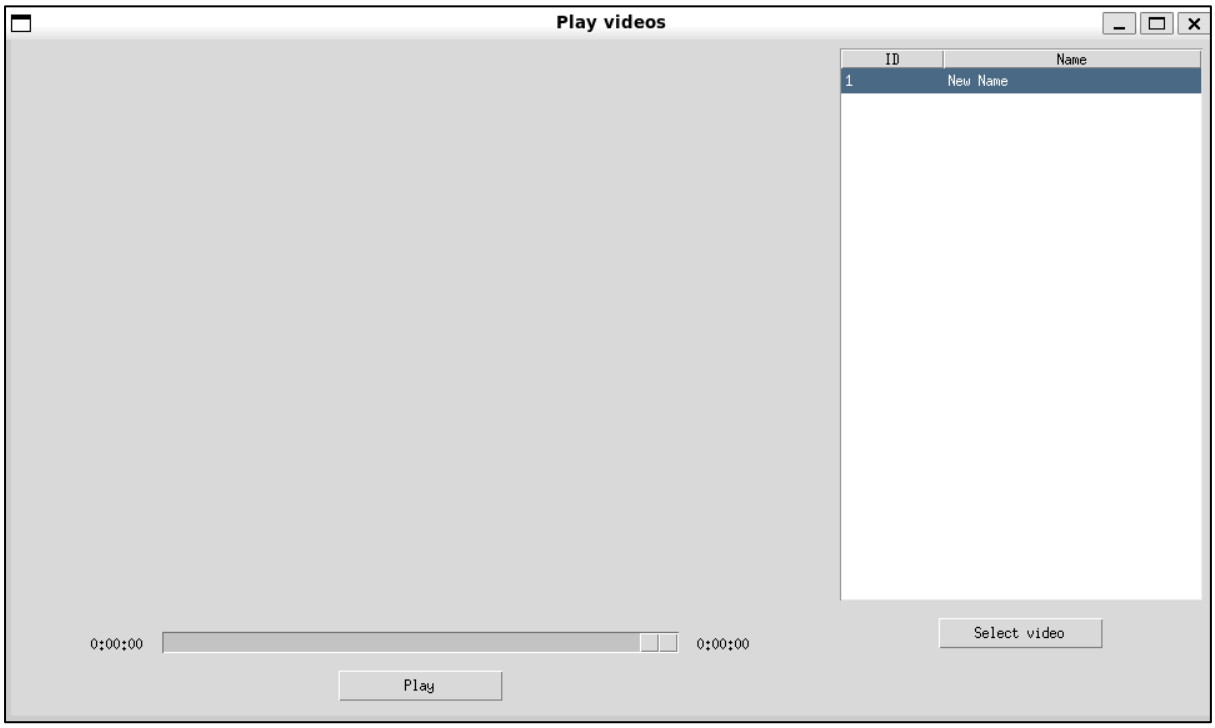




Play video

New button (Play) is added in CheckVideos UI
Use module tkVideoPlayer
When Play or Play Playlist is clicked, new TopLevel window will display containing video play area, and current playlist
Buttons:

- Select Video: load the current selected video
- Play: play the loaded video



Implementation (media_player.py)

```
"""This module contains Top-Level window that plays the videos"""

import tkinter as tk
from tkinter import messagebox as msgbox
from tkinter import ttk
from pathlib import Path
import datetime

from tkVideoPlayer import TkinterVideo

from ..singleton import SingletonMeta
from ..core.videos_db import VideosDB
from ..core.video_library import LibraryItem
```

```python
class MediaPlayer(tk.Toplevel, metaclass=SingletonMeta):
    """Top level window for playing a playlist"""

    COLUMNS = (0, 1)  # database columns indexes
    COLUMNS_WIDTH = (25, 150)

    def __init__(self, root):
        super().__init__(root)
        self.title('Play videos')
        self.minsize(width=960, height=540)

        self.protocol(
            'WM_DELETE_WINDOW', self.__on_close
        )  # Redefine close button 'x'
        self.__playlist = None  # Current play list
        self.__progress_value = tk.DoubleVar(self)  # Variable of progress bar
        self.__video_playable = None  # is the current selected video playable

        self.rowconfigure(0, weight=6)
        self.rowconfigure(1, weight=1)
        self.rowconfigure(2, weight=1)
        self.columnconfigure(0, weight=1)
        self.columnconfigure(1, weight=5)
        self.columnconfigure(2, weight=1)
        self.columnconfigure(3, weight=2)

        self._create_widgets()
        self._display_widgets()

        self.withdraw()  # Hide the window
```

```python
    def _create_widgets(self):
        self.__player = None  # Media player
        self.__start_label = ttk.Label(
            self, text=self.__time_to_str()
        )  # Video duration start
        self.__end_label = ttk.Label(
            self, text=self.__time_to_str()
        )  # Video duration end
        self.__play_btn = ttk.Button(
            self, text='Play', width=20, command=self.__play_pause
        )  # Play the video when clicked
        self.__select_video_btn = ttk.Button(
            self, text='Select video', width=20, command=self.__play_video
        )  # Load the selected video when clicked
        self.__progress_slider = ttk.Scale(
            self,
            from_=0,
            to=0,
            orient='horizontal',
            variable=self.__progress_value,
            command=self.__seek,
        )  # Time frame slider
        self.__video_browser = ttk.Treeview(
            self, height=20, show='headings'
        )  # Current playlist
        self.__video_browser['columns'] = tuple(
            VideosDB.COLUMNS[column] for column in self.COLUMNS
        )
        for column in self.COLUMNS:
            # Configuring video browser which contains playlist
            self.__video_browser.heading(
                VideosDB.COLUMNS[column], text=LibraryItem.HEADINGS[column]
            )
            self.__video_browser.column(
                VideosDB.COLUMNS[column], width=self.COLUMNS_WIDTH[column]
            )

    def _display_widgets(self):
        self.__video_browser.grid(row=0, column=3, sticky='nsew')
        self.__start_label.grid(row=1, column=0, sticky='es')
        self.__progress_slider.grid(row=1, column=1, sticky='wes')
        self.__end_label.grid(row=1, column=2, sticky='ws')
        self.__select_video_btn.grid(row=1, column=3, sticky='n')
        self.__play_btn.grid(row=2, column=1, sticky='n')
        for children in self.winfo_children():
            children.grid(padx=7, pady=7)
```

```
def __load_video(self, video):
    """Load new video

    Args:
        video - LibraryItem object
    """

    if self.__player:
        # Remove current playing video
        self.__player.destroy()
    self.__player = Player(
        self,
        duration=self.__update_duration,
        update_scale=self.__update_scale,
        video_ended=self.__video_ended,
    )  # create new video player object
    file_path = video.get_file_path()
    self.__video_playable = Path(
        file_path
    ).exists()  # Validate video existance
    if not self.__video_playable:
        return
    self.__player.load(file_path)
    self.__progress_slider.config(to=0, from_=0)  # Reset the progress bar
    self.__progress_value.set(0)  #

def __seek(self, value):
    """Play video at a specific second

    Args:
        value - a time frame to play in seconds
    """

    self.__player.seek(int(float(value)))

def __play_pause(self):
    """Toggle play button between Play-Pause"""

    if not self.__video_playable:
        msgbox.showerror('Video error', message='Cannot play this video!')
        return

    if self.__player.is_paused():
        self.__player.play()
        self.__play_btn['text'] = 'Pause'
    else:
        self.__player.pause()
        self.__play_btn['text'] = 'Play'
```

```
def __video_ended(self, event):
    """Reset the progress bar and button when the video ended"""

    self.__play_btn['text'] = 'Play'
    self.__progress_slider.set(0)

def __load_video(self, video):
    """Load new video

    Args:
        video - LibraryItem object
    """

    if self.__player:
        # Remove current playing video
        self.__player.destroy()
    self.__player = Player(
        self,
        duration=self.__update_duration,
        update_scale=self.__update_scale,
        video_ended=self.__video_ended,
    )  # create new video player object
    file_path = video.get_file_path()
    self.__video_playable = Path(
        file_path
    ).exists()  # Validate video existance
    if not self.__video_playable:
        return
    self.__player.load(file_path)
    self.__progress_slider.config(to=0, from_=0)  # Reset the progress bar
    self.__progress_value.set(0)  #

def __seek(self, value):
    """Play video at a specific second

    Args:
        value - a time frame to play in seconds
    """

    self.__player.seek(int(float(value)))
```

```
def __play_pause(self):
    """Toggle play button between Play-Pause"""

    if not self.__video_playable:
        msgbox.showerror('Video error', message='Cannot play this video!')
        return

    if self.__player.is_paused():
        self.__player.play()
        self.__play_btn['text'] = 'Pause'
    else:
        self.__player.pause()
        self.__play_btn['text'] = 'Play'

def __play_video(self):
    """Get the current selected video in playlist and play"""

    id = int(
        self.__video_browser.selection()[0]
    )  # Get the current selected video in playlist
    video = tuple(self.__playlist.values())[id]
    video.increment_play_count()  # Update play count
    self.__load_video(video)  # Load video to thread

def play(self, playlist):
    if self.winfo_viewable():
        # ask to replacing new playlist if the mediaplayer has already been displayed
        replacing = msgbox.askokcancel(
            title='Replace playlist',
            message='Replace current playing playlist?',
            icon=msgbox.WARNING,
        )
        if not replacing:
            return
    self.__playlist = playlist  # replace the playlist
    self.__refresh_playlist()
    self.deiconify()  # display the window
    self.__play_video()
```

```python
class Player(TkinterVideo):
    """Video player inherits from TkinterVideo class"""

    """
    @params:
        root - widgets root
        kwargs - event handlers
    """

    def __init__(self, root, **kwargs):
        super().__init__(root, scaled=True)
        self.bind(
            "<<Duration>>", kwargs.get('duration')
        )  # Called when new duration is loaded
        self.bind(
            "<<SecondChanged>>", kwargs.get('update_scale')
        )  # Called when video frame change
        self.bind(
            "<<Ended>>", kwargs.get('video_ended')
        )  # Called when video ended

        self.grid(row=0, column=0, columnspan=3, sticky='nsew')
```

Figure 34 - MediaPlayer (media_player.py) implementation

Testing and validation

TC	Description	Expected Output	Input	Output	Evidence	Test Result
1	Test menu UI	Display a menu as designed	Start the application	A menu matched designed UI	evidence	PASSED
2	Test the check videos UI	Display the check video window as designed	Click Check videos at menu window	A check video window matched designed UI	evidence	PASSED
3	Test Create Video List UI	Display the Create Video List window as designed	Click Create Video List button at menu window	A create video list window matched designed UI	evidence	PASSED
4	Test Update Video UI	Display the Update Video window as designed	Click Update Video button at menu window	An update video window matched designed UI	evidence	PASSED
5	Test Header widget	Display the same header in all three widgets	Click all three buttons at menu window	In all three windows, the header displayed similarly and matched the designed header	Evidence1 Evidence2 Evidence3	PASSED
6	Test footer widget	Display the same footer in all three widgets	Click all three buttons at menu window	In all three windows, the footer displayed similarly and matched the designed footer	Evidence1 Evidence2 Evidence3	PASSED

7	Test browser widget	Display the same browser in all three widgets	Click all three buttons at menu window	In all three windows, the browser displayed similarly and matched the designed browser	Evidence1 Evidence2 Evidence3	PASSED
8	Library Item module test. This test the functionality of two classes in the module Test design	Pass all unittests by pytest	Automated	All tests passed	Evidence	PASSED
9	VideosDB module test. This test the database connection and queries to the database Test design	Pass all unittests by pytest	Automated	All test passed	Evidence	PASSED
10	Test the function of menu's buttons	All button displays the correspond window	Click on each button	Each buttons display its correspond window	Evidence1 Evidence2 Evidence3	PASSED
11	Back button functionality test	Return to the menu when click	Click on back button	It returned to the menu window	evidence	PASSED
12	List all video button functionality test	Browser display the filtered videos information	Click list all video button at headbar	It displayed the filtered videos information	evidence	PASSED
13	Check videos functionality test with valid id input	Display the correct video's information	ID = 1 -> click check videos button	Displayed correct video's information	evidence	PASSED
14	Check videos functionality test with invalid id input	Show an error message box containing informative message	ID = 'one' -> click check videos button	Displayed an error message box containing informative message	evidence	PASSED
15	Create videos list adding functionality test with a valid id input	Video will be added to the playlist	ID = 1 -> click add button	The video with id=1 was added to the playlist	evidence	PASSED
16	Create videos list adding functionality test with an invalid id input (e.g. an invalid id or a video that has already been added)	Show an error message box containing informative message	ID = 'one' -> click add button	Displayed an error message box containing informative message	evidence	PASSED
17	Create videos list removing functionality	Video will be removed	ID = 1 -> click	The video with id=1 was	evidence	PASSED

	test with a valid id input	from the playlist	remove button	removed from the playlist		
18	Create videos list removing functionality test with an invalid id input (e.g. an invalid id or a video that has not been added)	Show an error message box containing informative message	ID = 'one' -> click remove button	Displayed an error message box containing informative message	evidence	PASSED
19	Create videos list playing functionality test with a non-empty playlist	All video in the playlist will increase its play count by one	A playlist containing video with id=(1, 2)->click play playlist button	Every playlist videos's playcount increased by one	evidence	PASSED
20	Create videos list playing functionality test with an empty playlist	Show an error message box containing informative message	An empty playlist -> click play playlist button	Displayed an error message box containing informative message	evidence	PASSED
21	Update video functionality with valid inputs	New video information will be updated	Video ID = 2, name='New name' -> click update button	Video information has been updated successfully	evidence	PASSED
22	Update video functionality with invalid id input	Show an error message box containing informative message	Video ID = 'two' -> click update button	Displayed an error message box containing informative message	evidence	PASSED
23	Update video functionality with valid id entry and invalid information entry field (e.g. Left empty)	Show an error message box containing informative message	Id = 2, name = '' -> click update	Displayed an error message box containing informative message	evidence	PASSED
24	Search engine test with prefix = empty string	Show all videos	Search box entry = ''	Displayed all videos	evidence	PASSED
25	Search engine test with prefix = 'a'	Show videos 1 (director matches prefix), 4 (name matches prefix	Searchbox entry = 'a'	Display video 1, 4	evidence	PASSED
26	Search engine test with prefix = 'abc'	Show no videos as no video match the prefix	Search entry	Displayed no video	evidence	PASSED

27	Sort by functionality test	Show videos sord by selected attribute	Sort_by = id	Displayed videos sorted by id	evidence	PASSED
28	Sort by functionality test	Show videos sorted by selected attribute	Sort by=name	Displayed videos sorted by name	evidence	PASSED
29	Sort order functionality test	Show videos sorted ascendingly by id	Sortby=id, sort order=asce nding	Displayed videos sorted ascendingly by name	evidence	PASSED
30	Sort order functionality test	Show videos sorted descendingly by id	Sortby=id, sord_order =descendin g	Displayed videos sorted descendingly by name	evidence	PASSED
31	Play video functionality test (play button at check video window)	Show a toplevel window containing media play area and a playlist with 1 video	Id = 2 -> click play	A media player shown containing playlist of 1 video	evidence	PASSED
32	Play video functionality test (play playlist button at create video list window)	Show a toplevel window containing media play area and the created playlist	Playlist containing video id=(1, 2) - > click play button	A media player shown containing playlist of created playlist	evidence	PASSED
TOTAL						32/32 PASSED

Conclusion and Future Development

The Python Video Player project has successfully demonstrated the versatility and power of Python by creating a useful and user-friendly video player. The application's capacity to storing video information, create playlists, and update video data demonstrates Python's promise in multimedia processing.

My understanding of developing GUIs, evaluating movies, and utilising various Python libraries has improved as a result of this project. It has also proven to be an excellent teaching aid, offering a practical application of Python programming concepts.

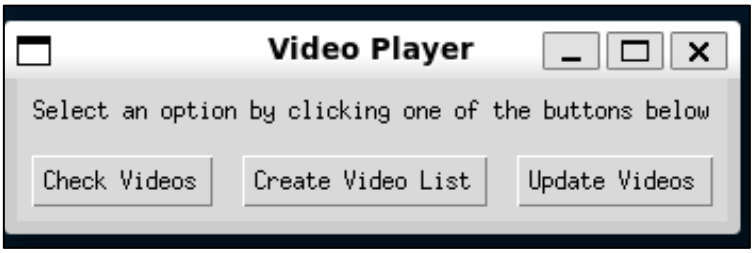
There are a number of intriguing prospects for the Python Video Player to be developed in the future:

1. Support for More Video Formats: More video formats might be supported by the programme, increasing its adaptability and utility.
2. Improved User Interface: It would be possible to improve the user interface's readability and aesthetic appeal. It might be possible to provide other features like progress bars, skin customization, and video thumbnails.
3. More Advanced Video Control: The application's video control functionality is currently restricted; to improve the video viewing experience, capabilities like

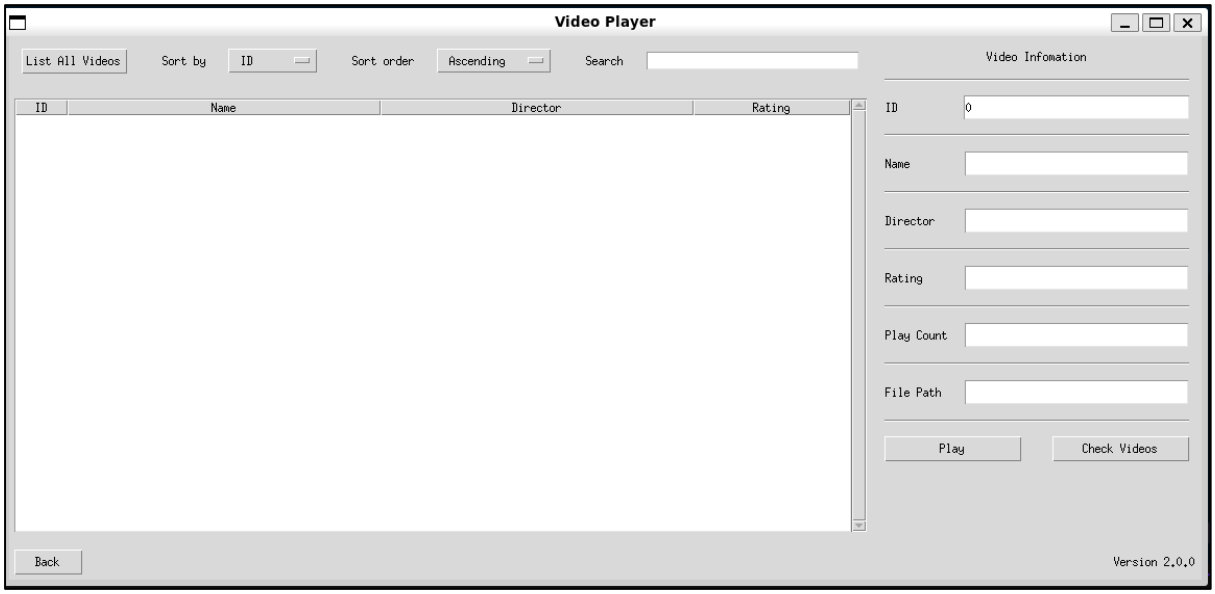
fast-forward, rewind, frame-by-frame viewing, and subtitle support might be added.

Appendix

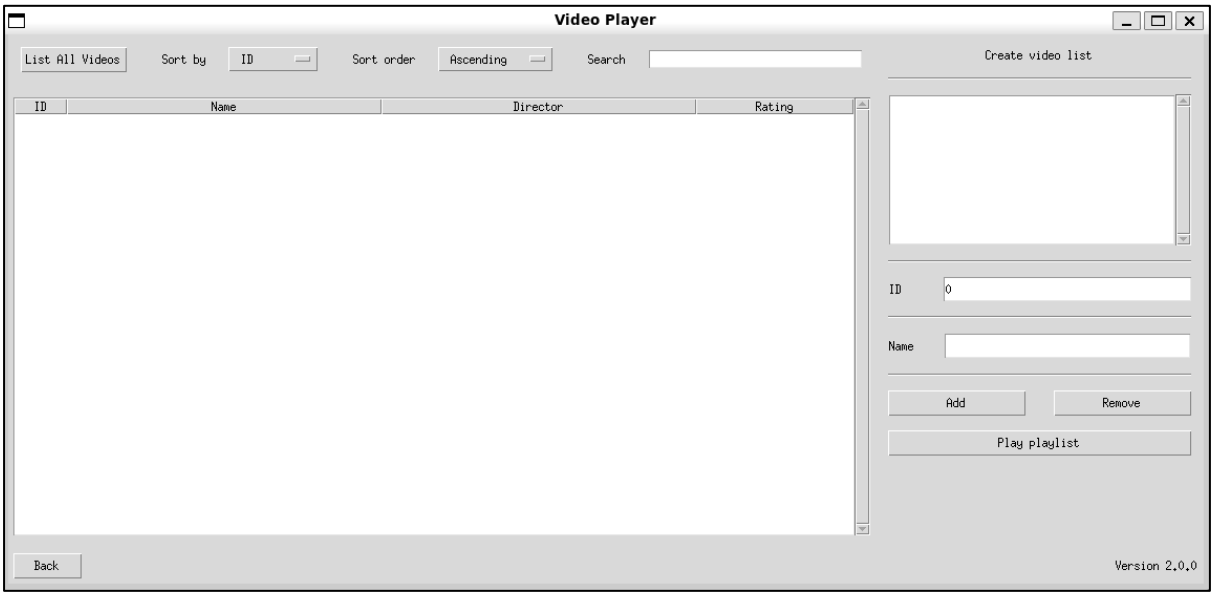
Menu



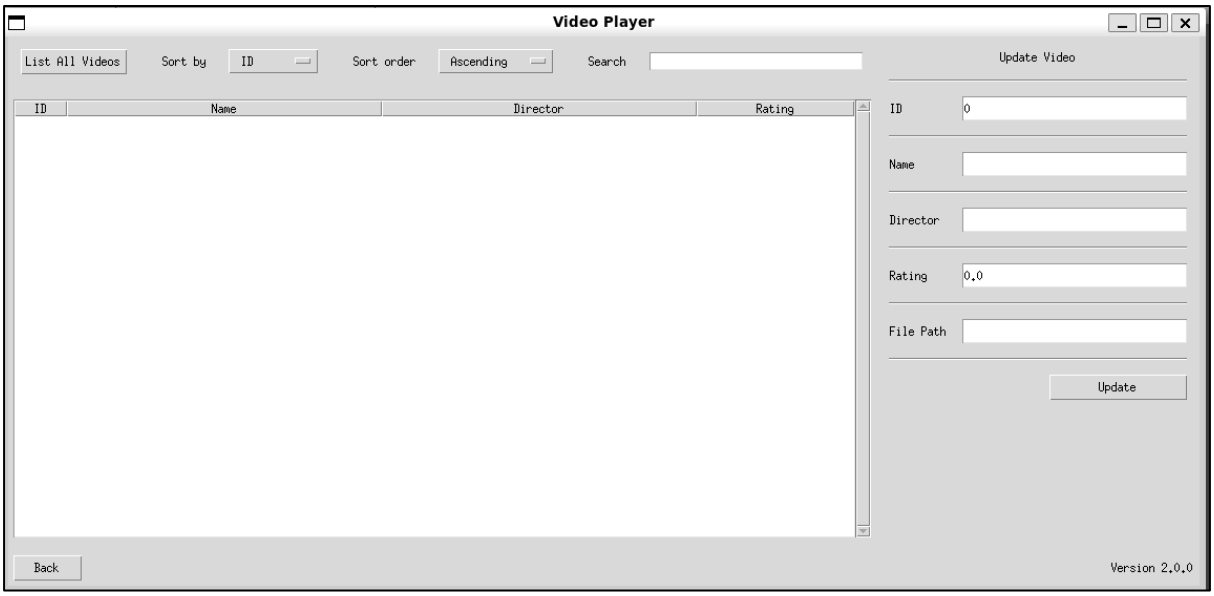
Check Videos



Create Video List UI



Update Video UI



Library Item Test

Test design

```python
class TestLibraryItem:
    item = LibraryItem(1, 'Video1', 'Director', 3, 1, '/abc/def.mp4')

    def test_list_all(self):
        assert self.item.list_all((0, 1, 2)) == (1, 'Video1', 'Director')
        assert self.item.list_all(('file_path', 'video_id', 'play_count')) == (
            '/abc/def.mp4',
            1,
            1,
        )
        assert self.item.list_all() == (
            1,
            'Video1',
            'Director',
            3,
            1,
            '/abc/def.mp4',
        )

    def test_init_value(self):
        assert self.item.get_id() == 1
        assert self.item.get_name() == 'Video1'
        assert self.item.get_director() == 'Director'
        assert self.item.get_rating() == 3
        assert self.item.get_play_count() == 1
        assert self.item.get_file_path() == '/abc/def.mp4'

    def test_index_subscription(self):
        assert self.item.get_id() == self.item[0]
        assert self.item.get_name() == self.item[1]
        assert self.item.get_director() == self.item[2]
        assert self.item.get_rating() == self.item[3]
        assert self.item.get_play_count() == self.item[4]
        assert self.item.get_file_path() == self.item[5]
```

```python
    def test_update_methods(self):
        with patch('app.core.videos_db.VideosDB.update', return_value=None):
            self.item.increment_play_count()
            assert self.item.get_play_count() == 2

            self.item.set_name('videoblah')
            assert self.item.get_name() == 'videoblah'

            self.item.set_director('Directorbuh')
            assert self.item.get_director() == 'Directorbuh'

            self.item.set_rating('4')
            assert self.item.get_rating() == 4

            self.item.set_file_path('new/path')
            assert self.item.get_file_path() == 'new/path'

    def test_update_methods_using_index_supscription(self):
        with patch('app.core.videos_db.VideosDB.update', return_value=None):
            self.item[4] += 1
            assert self.item.get_play_count() == 3

            self.item[1] = 'aha'
            assert self.item.get_name() == 'aha'

            self.item[2] = 'lmao'
            assert self.item.get_director() == 'lmao'

            self.item[3] = 5
            assert self.item.get_rating() == 5

            self.item[5] = 'another/path'
            assert self.item.get_file_path() == 'another/path'
```

```python
class TestLibraryItemCollection:
    item1 = LibraryItem(1, 'Video1', 'Director', 3, 1, '/abc/def.mp4')
    item2 = LibraryItem(2, 'Video2', 'Dir', 2, 3, 'abc/egh.mp4')

    collection = LibraryItemCollection((item1,))

    def test_membership(self):
        assert 1 in self.collection
        assert 2 not in self.collection
        assert 0 not in self.collection

    def test_add(self):
        self.collection.add(self.item2)
        assert 1 in self.collection
        assert 2 in self.collection

    def test_remove(self):
        self.collection.remove(1)
        assert 1 not in self.collection
        assert 2 in self.collection

    def test_getitem(self):
        assert self.collection[2] == self.item2

    def test_values(self):
        assert tuple(self.collection.values()) == (self.item2,)

    def test_iter(self):
        for item in self.collection:
            assert item == self.item2
```

Result

```
(cwvp) pytest tests/test_library_item.py -v
============================= test session starts ==============================
platform linux -- Python 3.10.13, pytest-7.4.3, pluggy-1.3.0 -- /home/serein/miniconda3/envs/cwvp/bin/python
cachedir: .pytest_cache
rootdir: /home/serein/programming/Projects/CW_VideoPlayer
plugins: mock-3.12.0
collected 11 items

tests/test_library_item.py::TestLibraryItem::test_list_all PASSED        [  9%]
tests/test_library_item.py::TestLibraryItem::test_init_value PASSED      [ 18%]
tests/test_library_item.py::TestLibraryItem::test_index_subscription PASSED [ 27%]
tests/test_library_item.py::TestLibraryItem::test_update_methods PASSED  [ 36%]
tests/test_library_item.py::TestLibraryItem::test_update_methods_using_index_supscription PASSED [ 45%]
tests/test_library_item.py::TestLibraryItemCollection::test_membership PASSED [ 54%]
tests/test_library_item.py::TestLibraryItemCollection::test_add PASSED   [ 63%]
tests/test_library_item.py::TestLibraryItemCollection::test_remove PASSED [ 72%]
tests/test_library_item.py::TestLibraryItemCollection::test_getitem PASSED [ 81%]
tests/test_library_item.py::TestLibraryItemCollection::test_values PASSED [ 90%]
tests/test_library_item.py::TestLibraryItemCollection::test_iter PASSED  [100%]
```

VideosDB Test

Test design

```python
import sqlite3
from pathlib import Path
from unittest import mock

import pytest
from app.core.videos_db import VideosDB
from app.namespaces.queries import Queries
from app import CONFIG


class TestVideosDB:
    @classmethod
    def setup_class(cls):
        db_path = Path(CONFIG['database']['path']['db'])
        if db_path.exists():
            db_path.unlink()
        cls.db = VideosDB()

    def test_connection(self):
        assert self.db.cursor is not None

    def test_getall(self):
        all_data = self.db.get_all()
        assert isinstance(all_data, tuple)
        assert len(all_data) == 4

    def test_close(self):
        self.db.close()
        with pytest.raises(sqlite3.ProgrammingError):
            self.db.cursor.execute(
                Queries.SELECT_ALL.safe_substitute(table=self.db.TABLE)
            )
```

Result

```
~ pytest tests/test_videos_db.py -v
============================= test session starts ==============================
platform linux -- Python 3.10.13, pytest-7.4.3, pluggy-1.3.0 -- /home/serein/miniconda3/envs/cwvp/bin/python
cachedir: .pytest_cache
rootdir: /home/serein/programming/Projects/CW_VideoPlayer
plugins: mock-3.12.0
collected 3 items

tests/test_videos_db.py::TestVideosDB::test_connection PASSED            [ 33%]
tests/test_videos_db.py::TestVideosDB::test_getall PASSED                [ 66%]
tests/test_videos_db.py::TestVideosDB::test_close PASSED                 [100%]

============================== 3 passed in 0.03s ===============================
```

Singleton Test

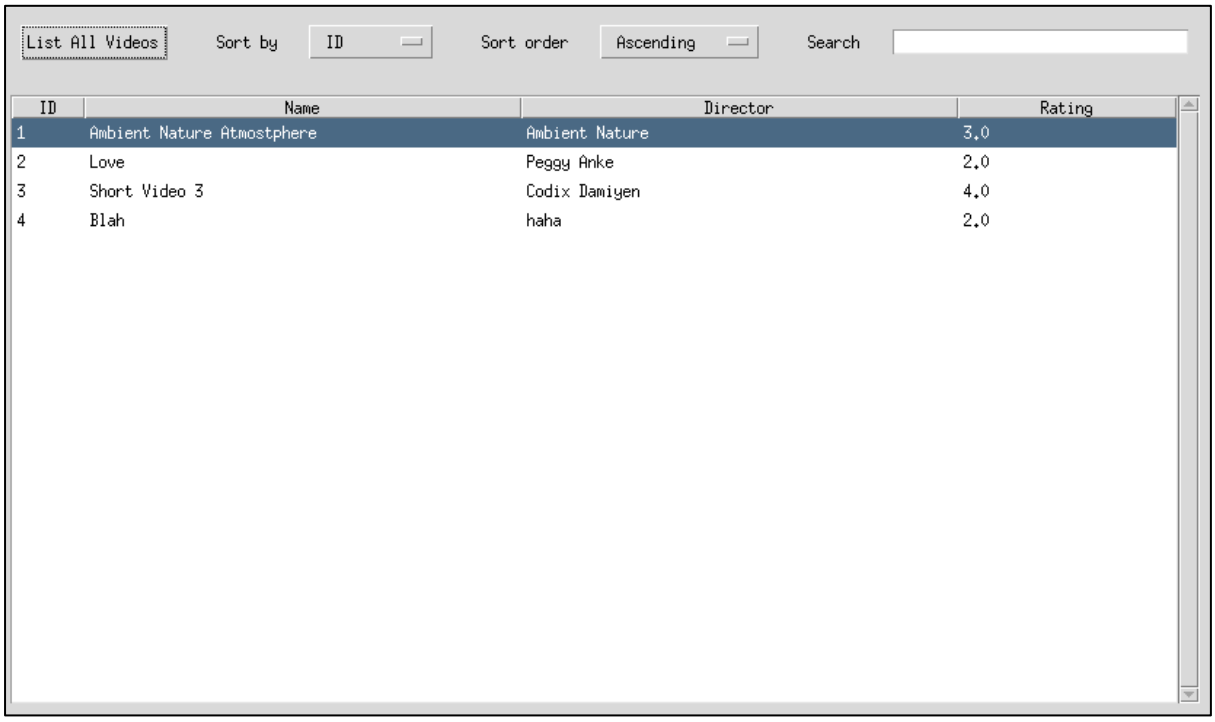
Test design

```python
import pytest
from app.singleton import SingletonMeta

def test_singleton():
    class TestObj(metaclass=SingletonMeta):
        pass
    class TestObj2(metaclass=SingletonMeta):
        pass
    assert TestObj() is TestObj()
    assert TestObj() is not TestObj2()
```

Result

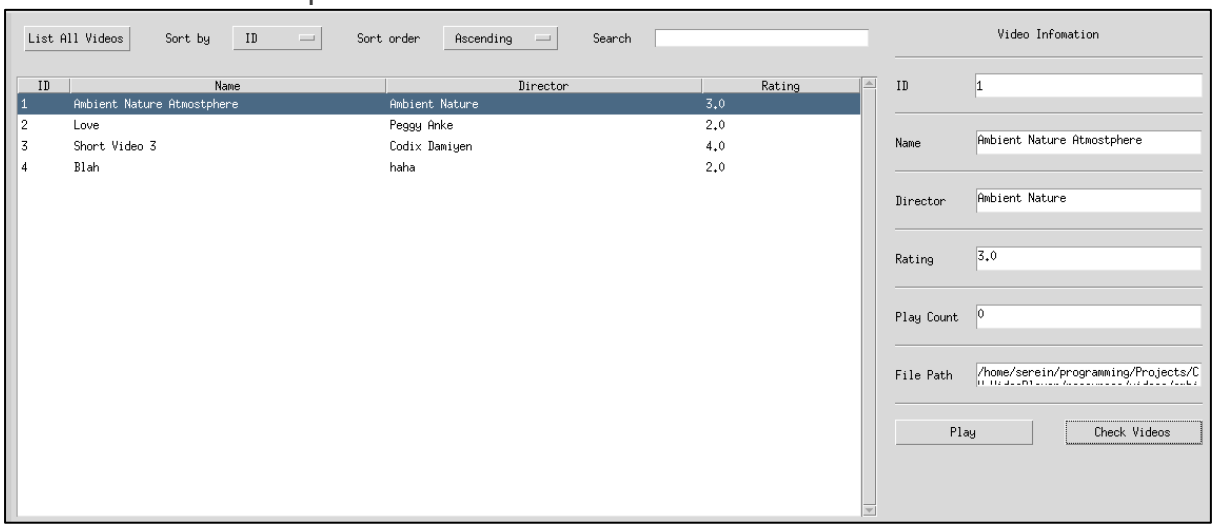
```
~ pytest tests/test_singleton.py -v
============================= test session starts ==============================
platform linux -- Python 3.10.13, pytest-7.4.3, pluggy-1.3.0 -- /home/serein/miniconda3/envs/cwvp/bin/python
cachedir: .pytest_cache
rootdir: /home/serein/programming/Projects/CW_VideoPlayer
plugins: mock-3.12.0
collected 1 item

tests/test_singleton.py::test_singleton PASSED                           [100%]

============================== 1 passed in 0.01s ===============================
```

List All Videos Button Test

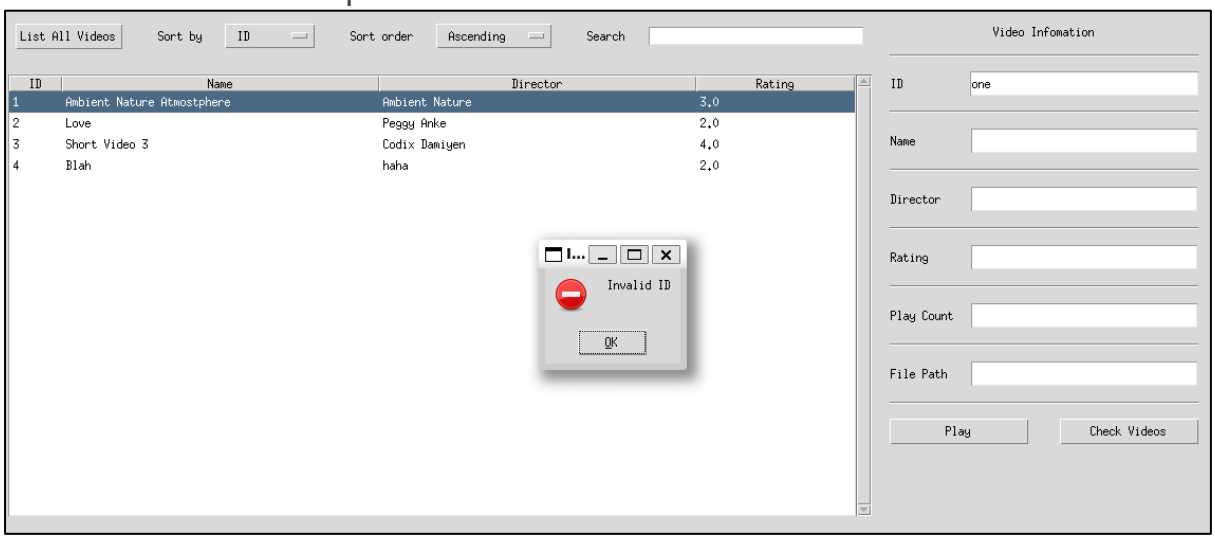


Check Videos Tests

Test1 – Valid input



Test2 - Invalid Input

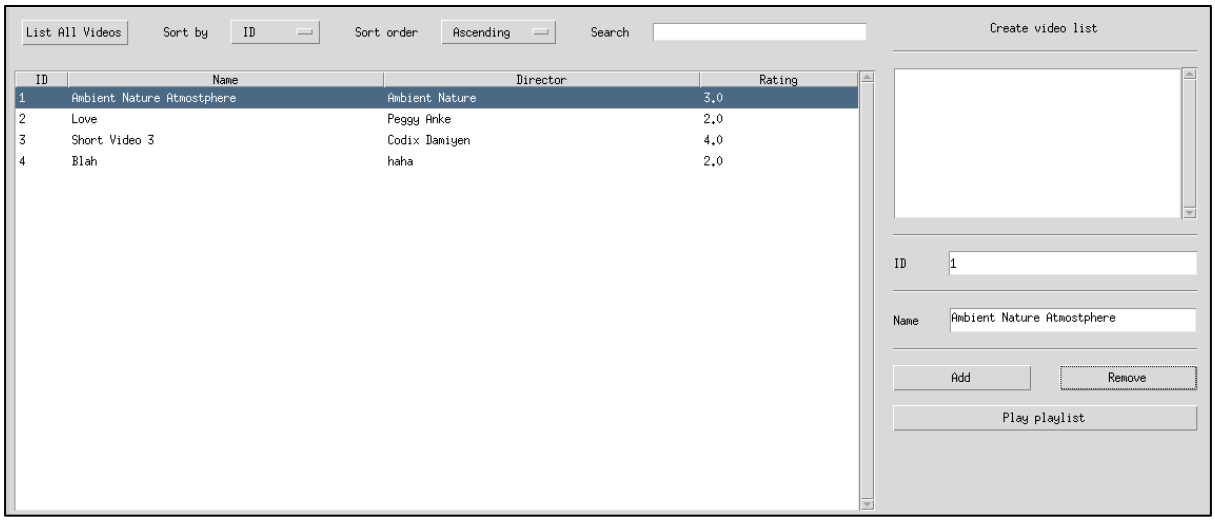


Create Video List Tests

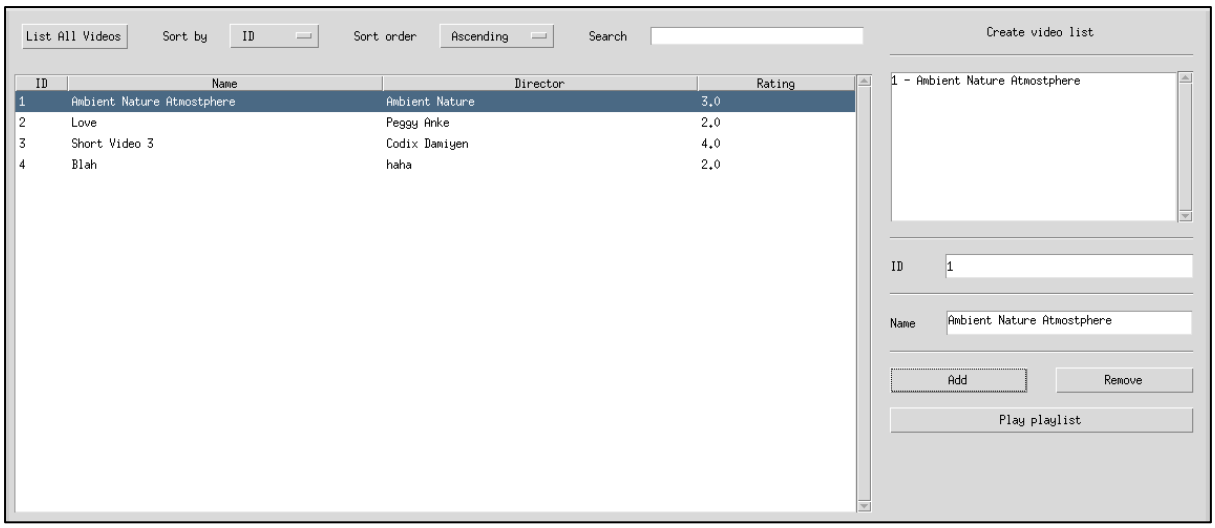
Add button

Valid Input

Before

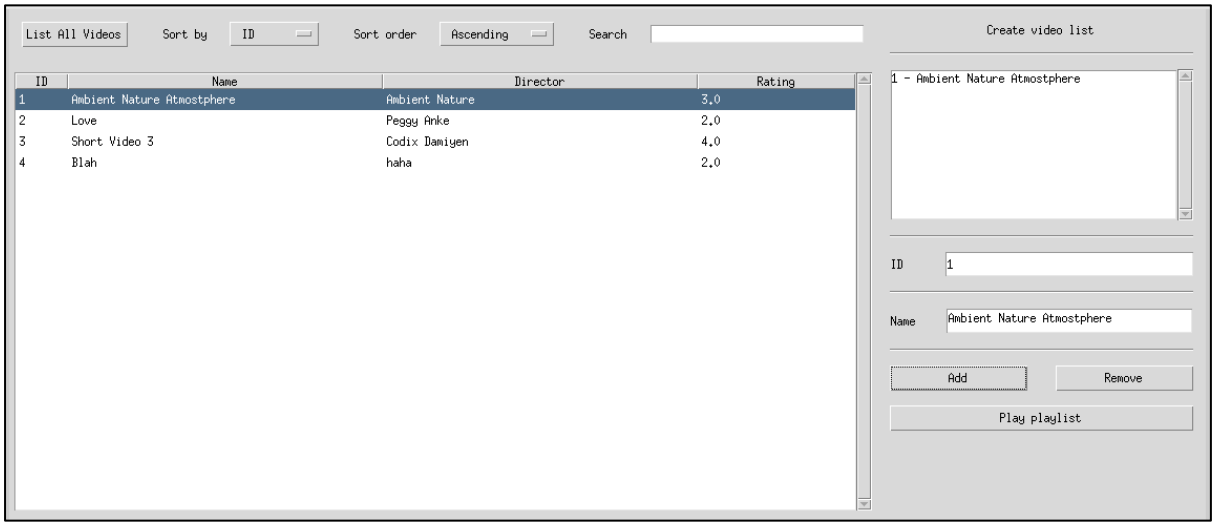


After

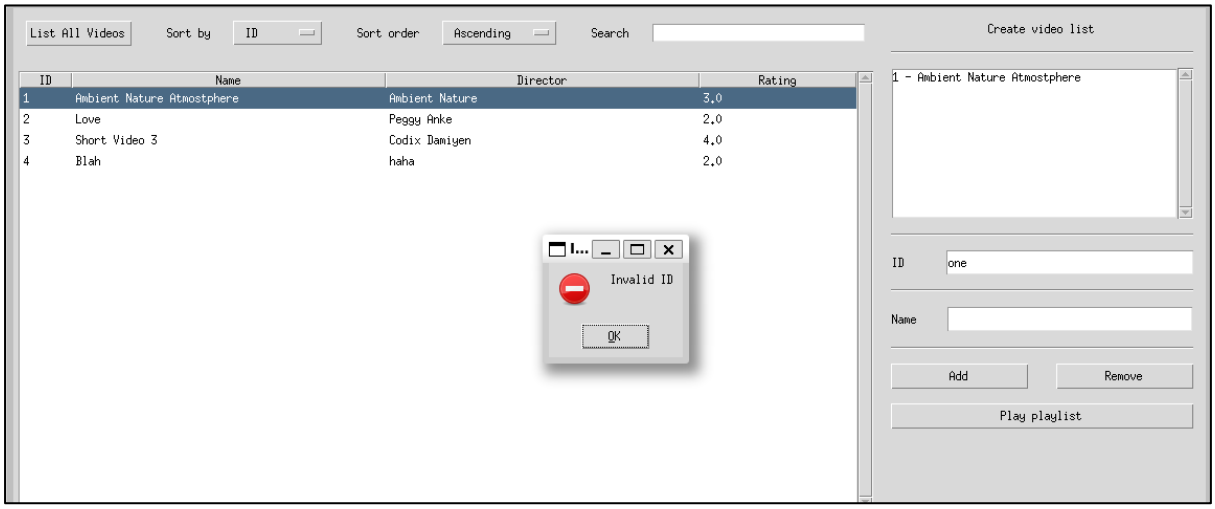


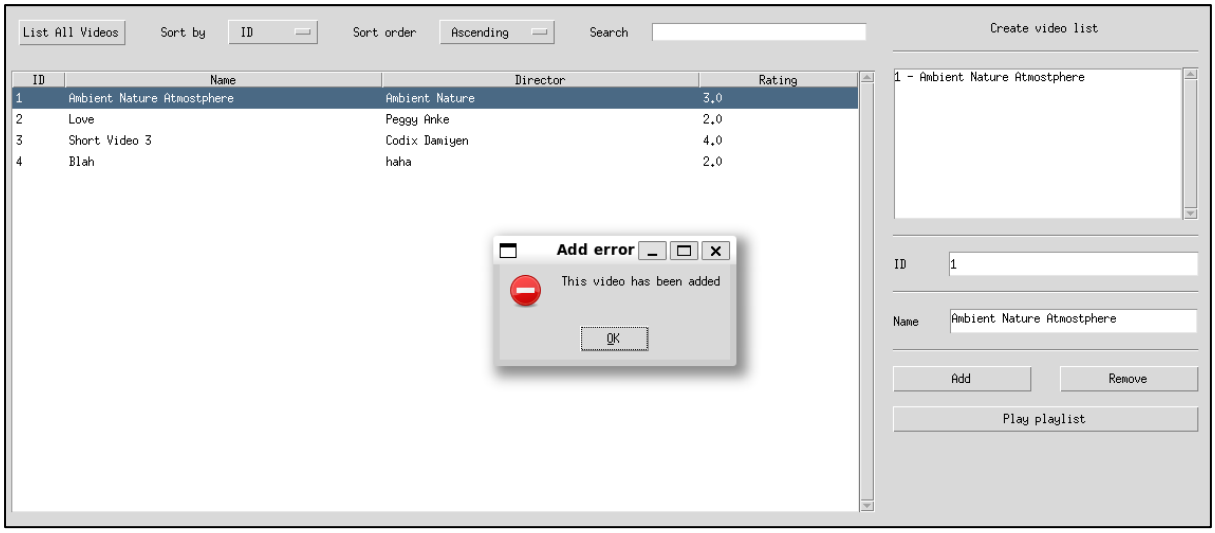
`Invalid Input`

Before



After

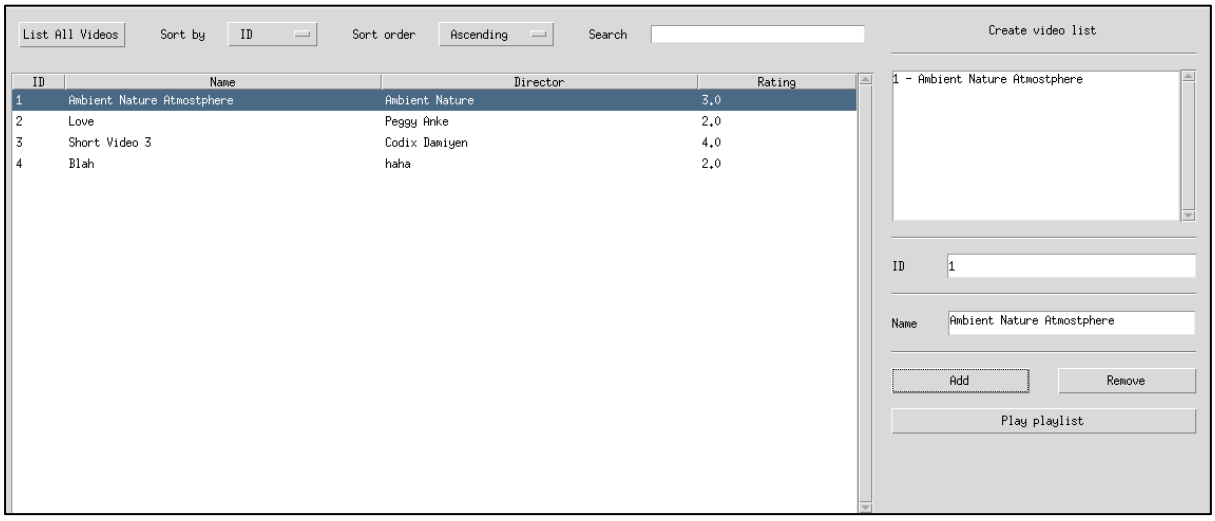




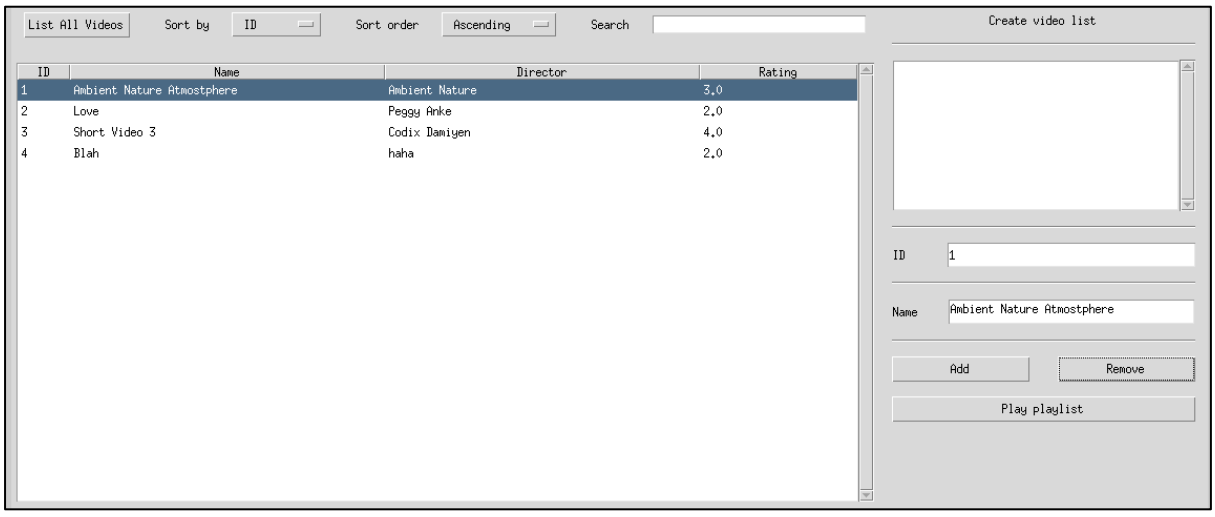
Remove Button

Valid Input

Before

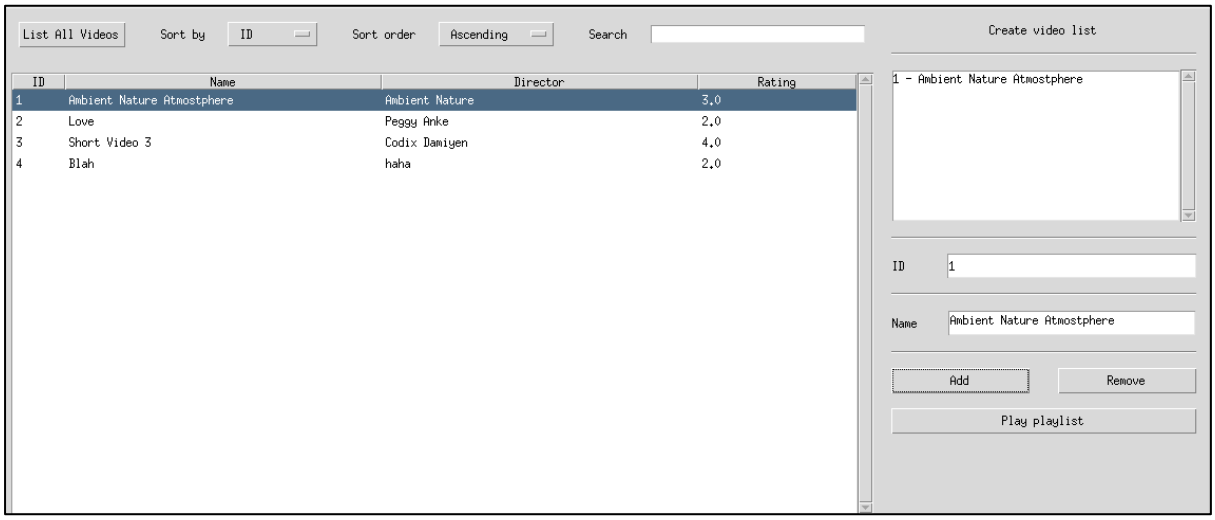


After

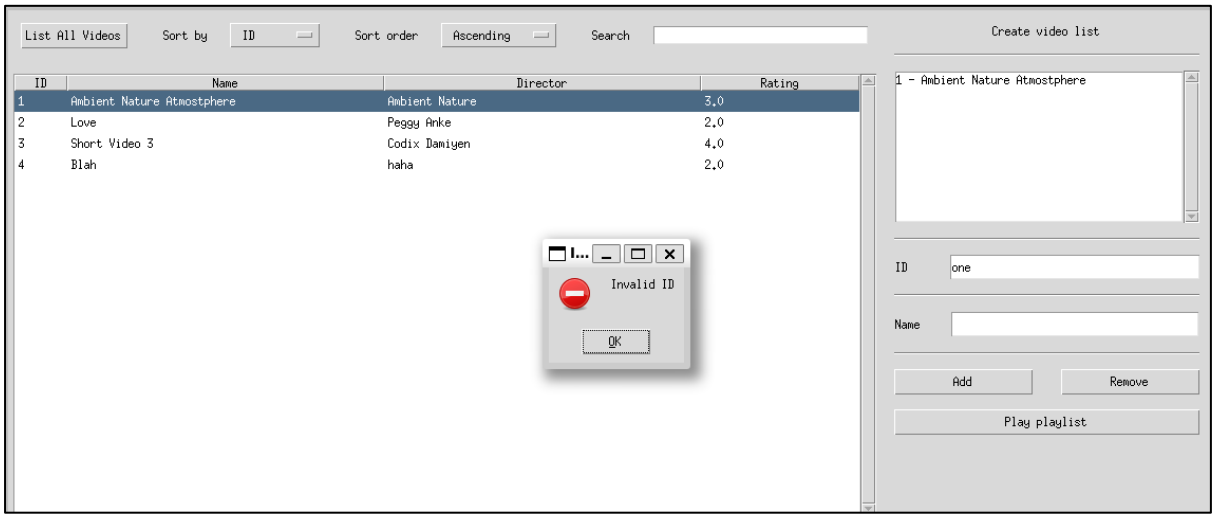


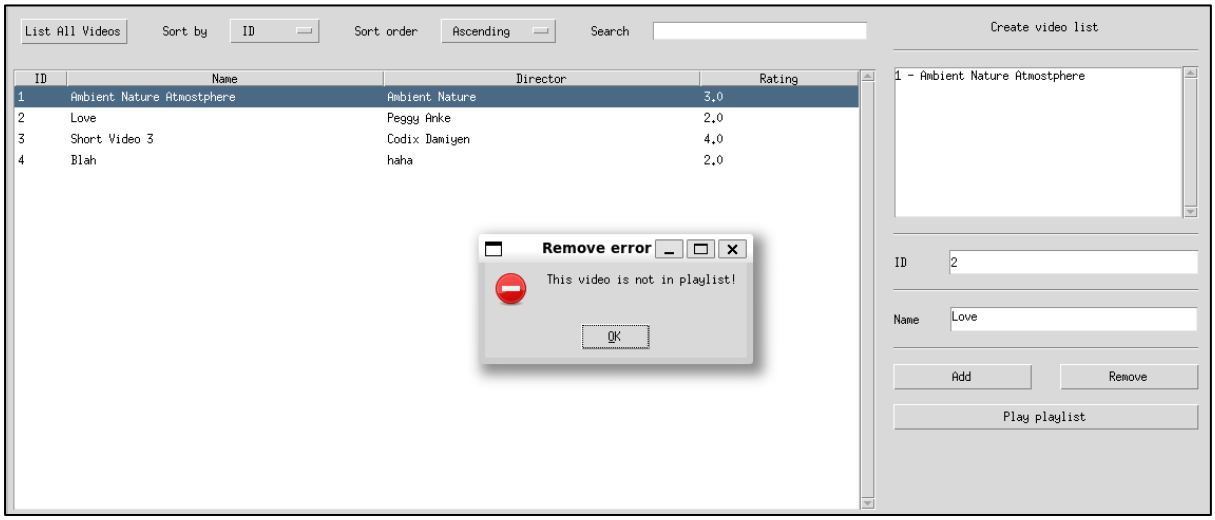
Invalid Input

Before



After

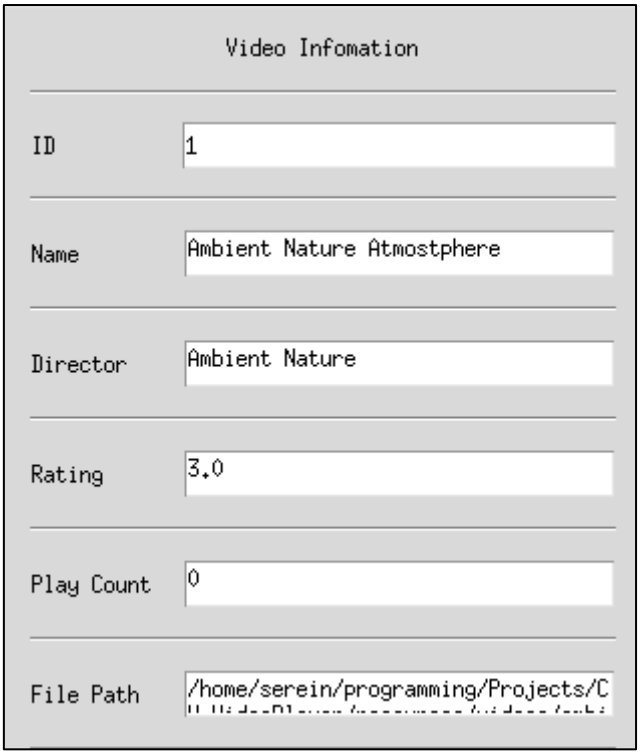


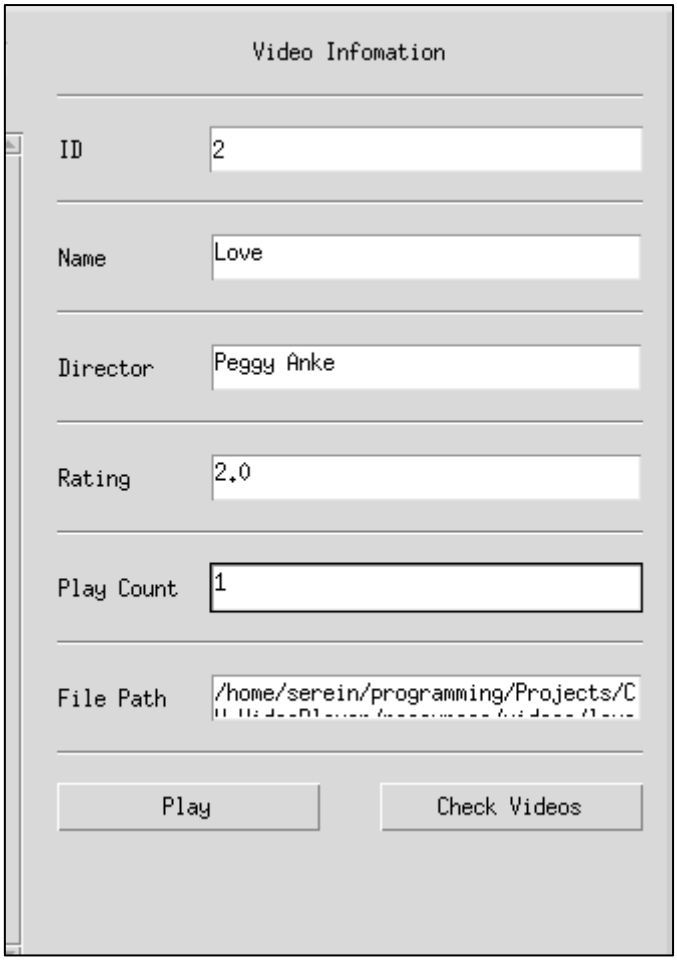


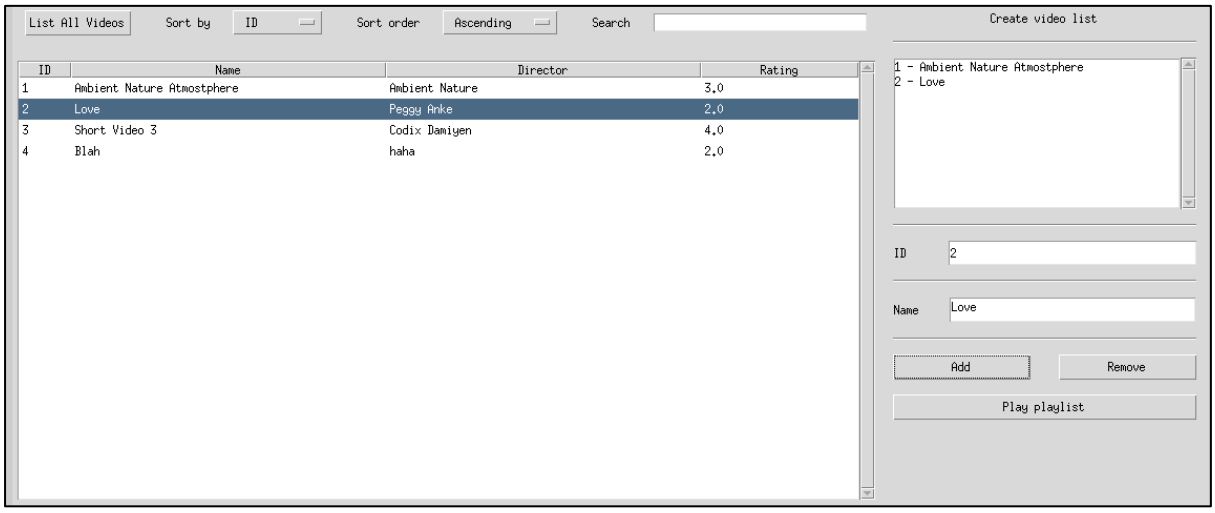
Play Playlist Button

Valid Playlist

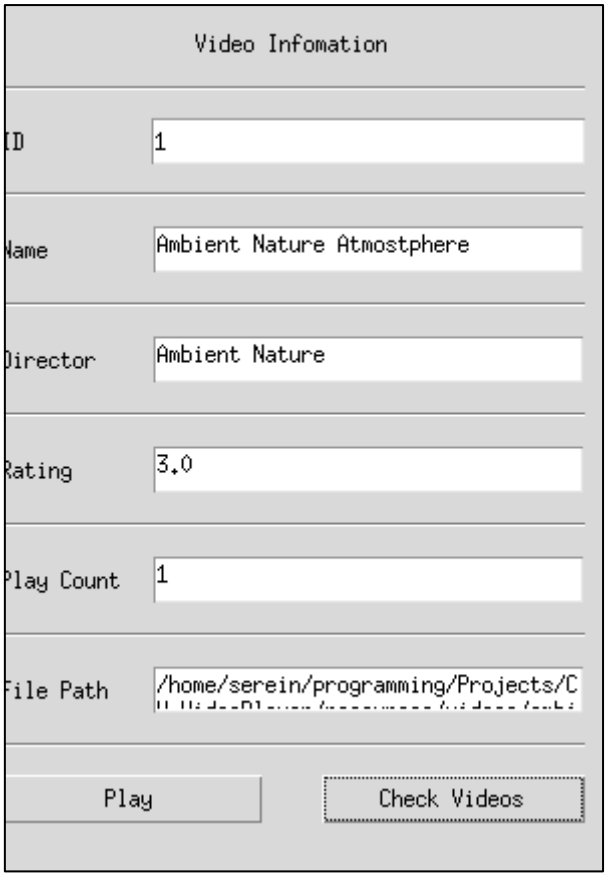
Before

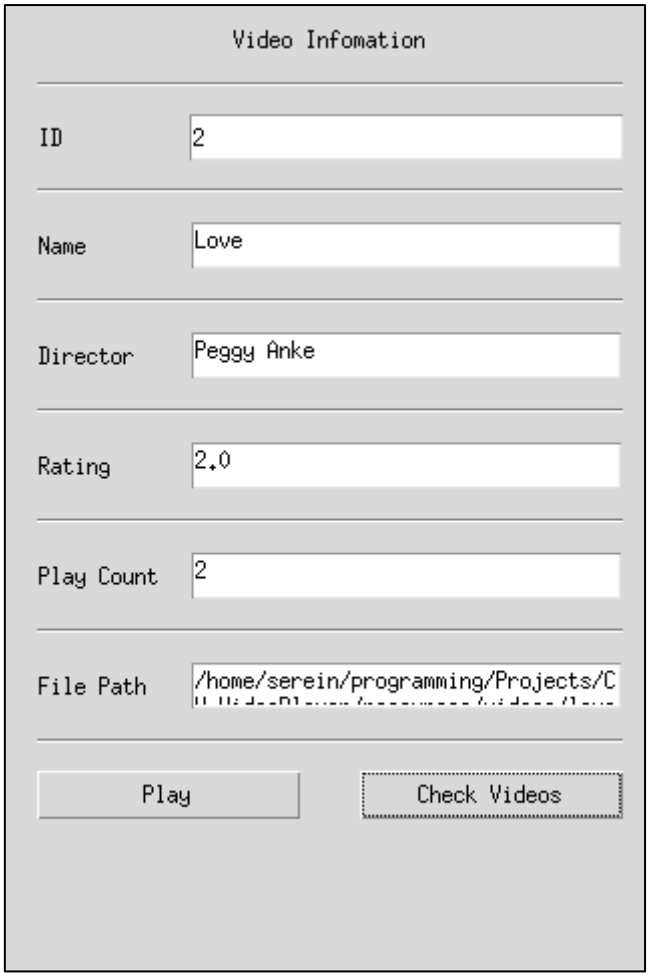



Video Infomation
ID
2
Name
Love
Director
Peggy Anke
Rating
2.0
Play Count
1
File Path
/home/serein/programming/Projects/C
Play
Check Videos


List All Videos
Sort by
ID
Sort order
Ascending
Search
Create video list
ID
Name
Director
Rating
1
Ambient Nature Atmostphere
Ambient Nature
3.0
2
Love
Peggy Anke
2.0
3
Short Video 3
Codix Damiyen
4.0
4
Blah
haha
2.0
1 - Ambient Nature Atmostphere
2 - Love
ID
2
Name
Love
Add
Remove
Play playlist

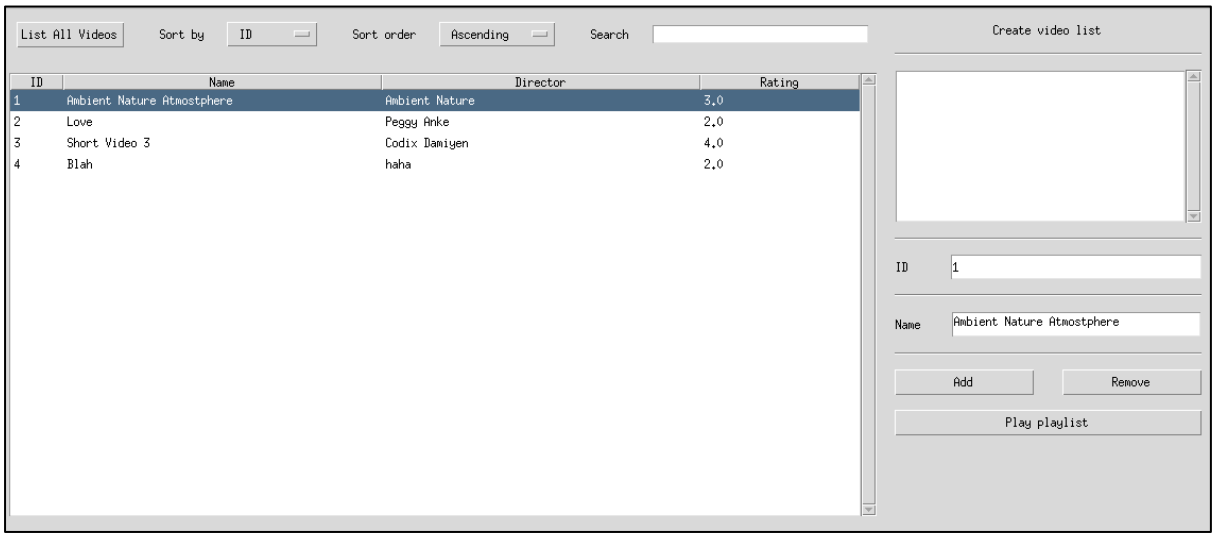
After


Video Infomation
ID
1
Name
Ambient Nature Atmostphere
Director
Ambient Nature
Rating
3.0
Play Count
1
File Path
/home/serein/programming/Projects/C
Play
Check Videos

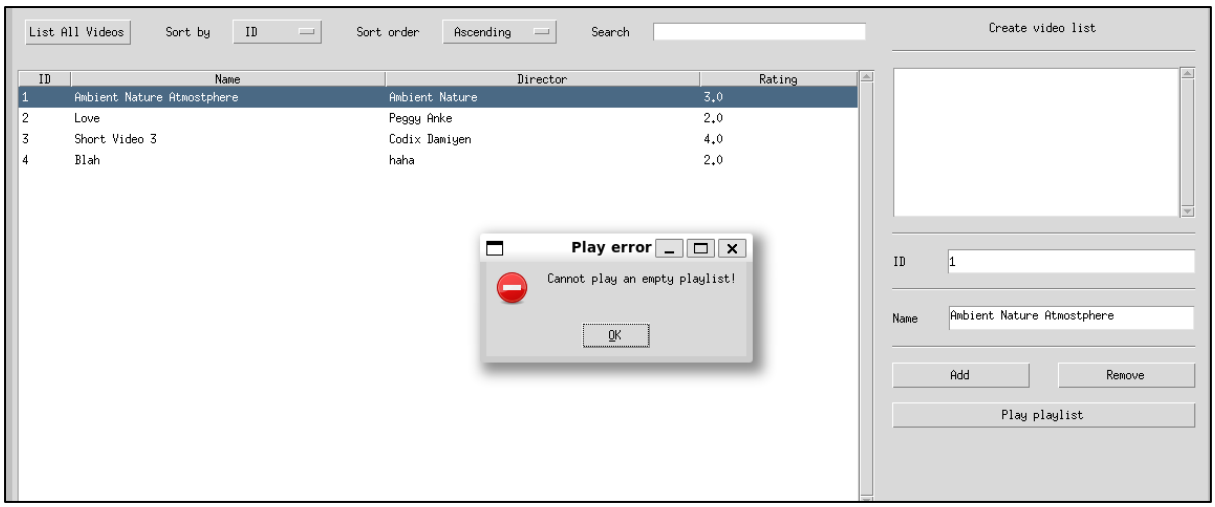


`Empty Playlist`

Before



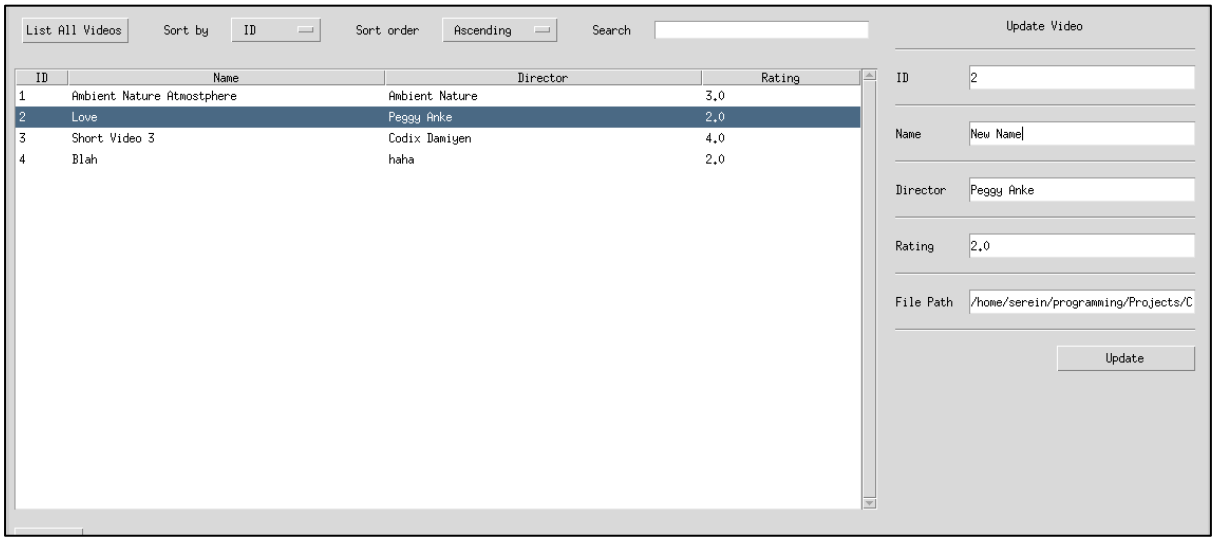
After



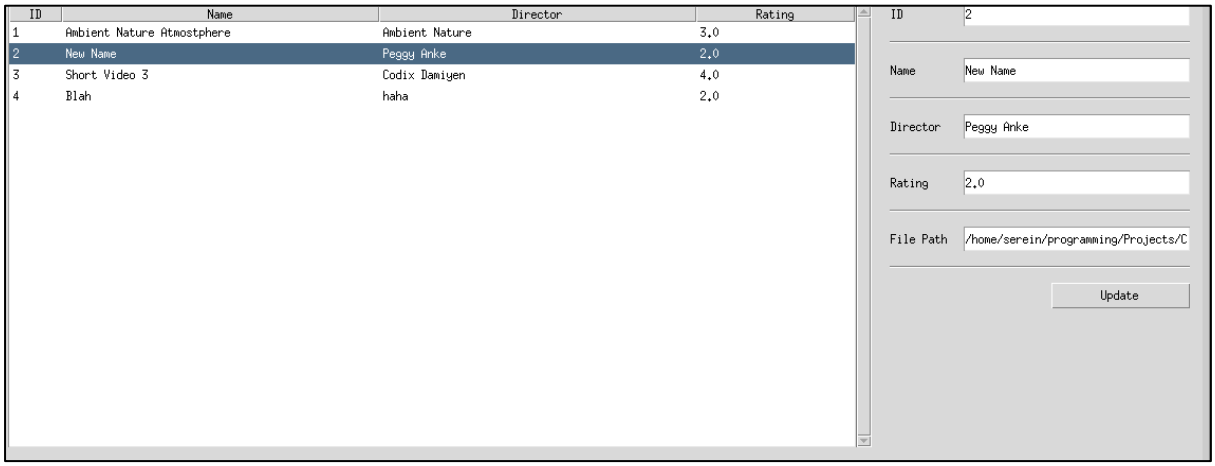
Update Video Test

Valid Input

Before

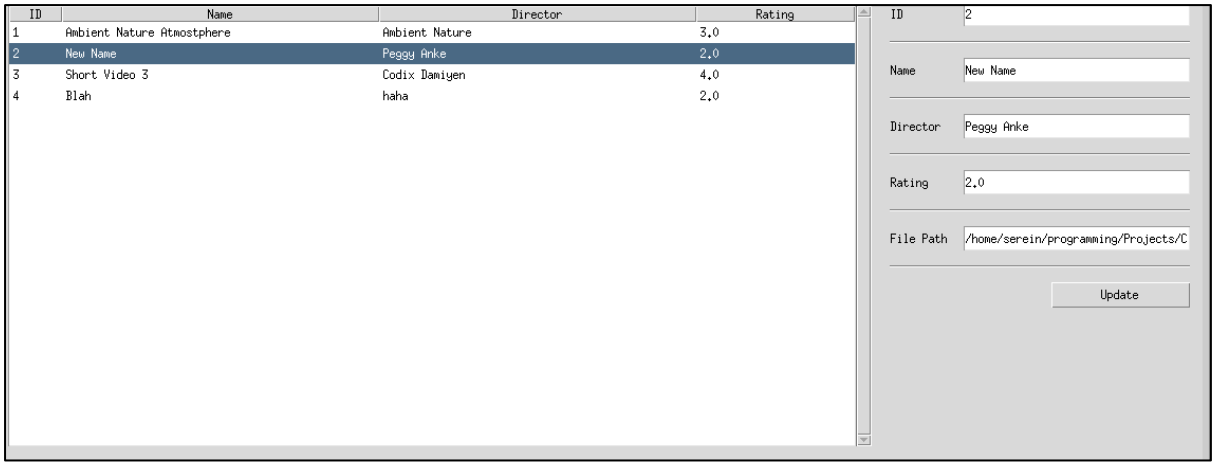


After

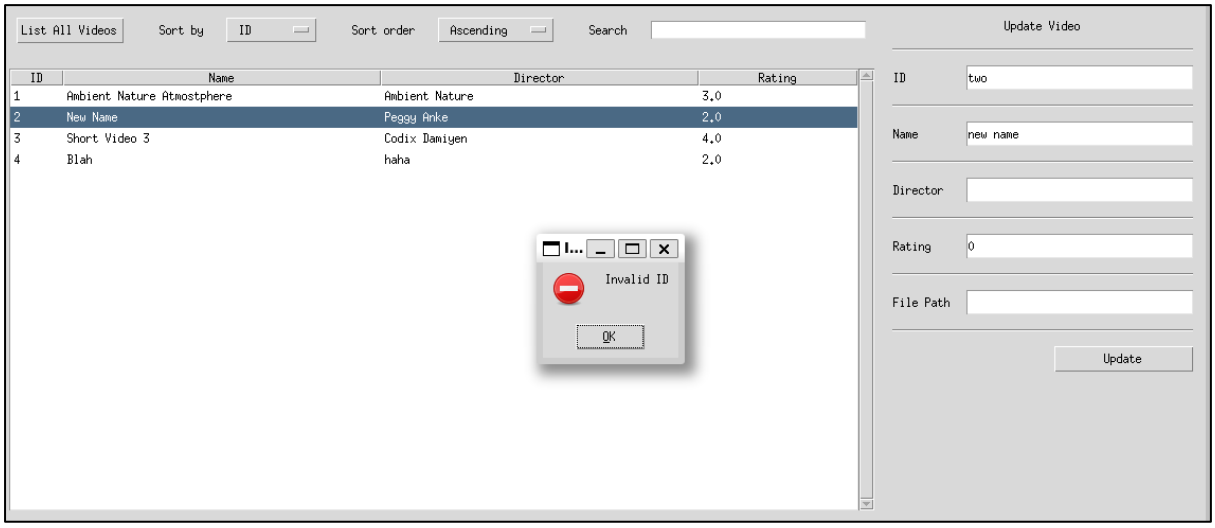


Invalid Id Input

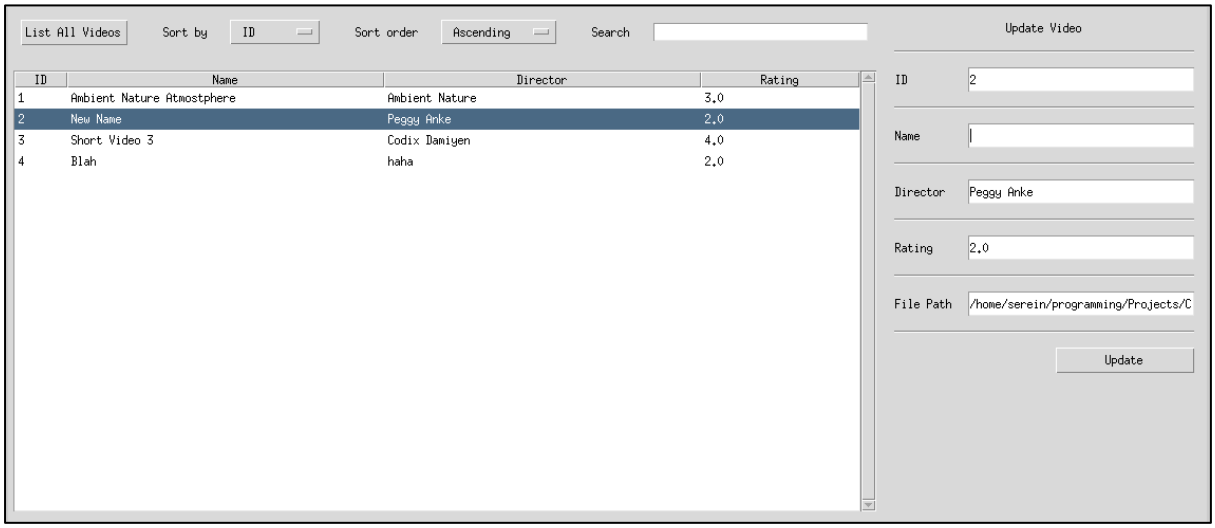
Before



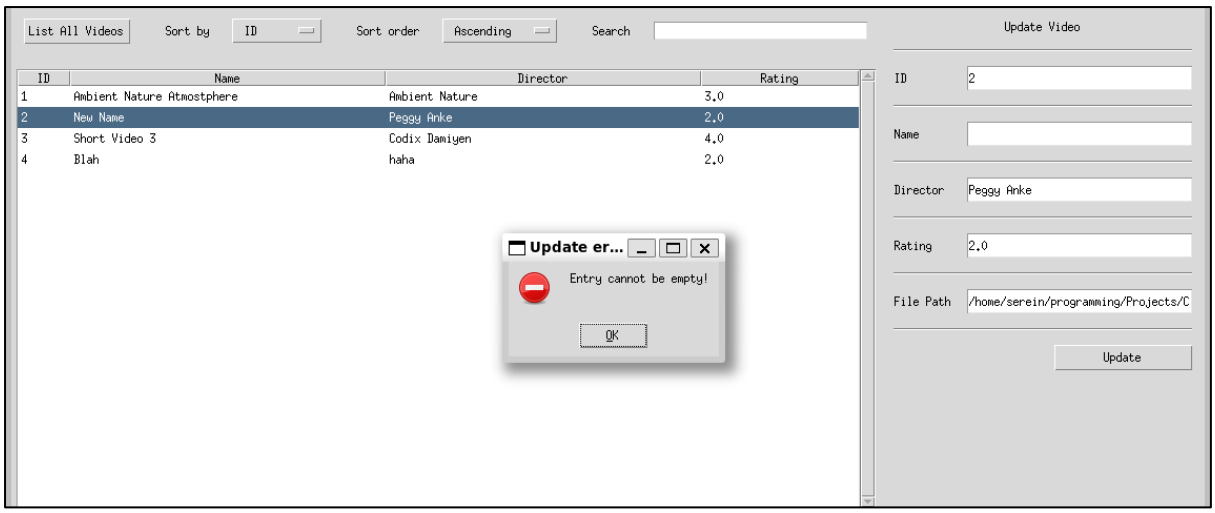
After



`Invalid Field Input`
Before



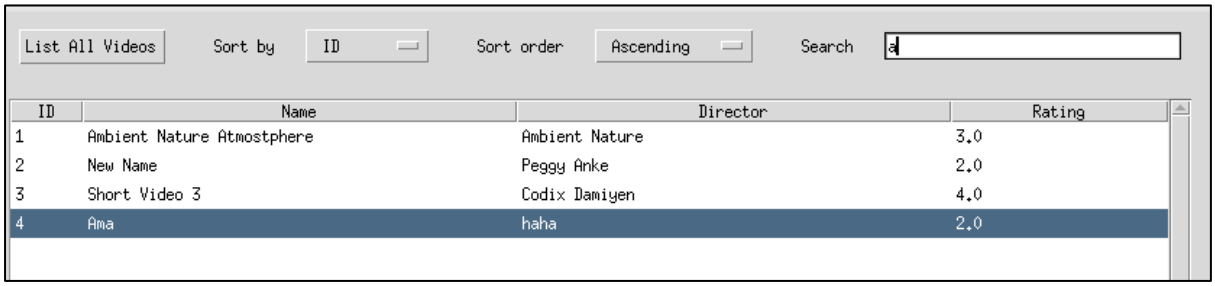
After



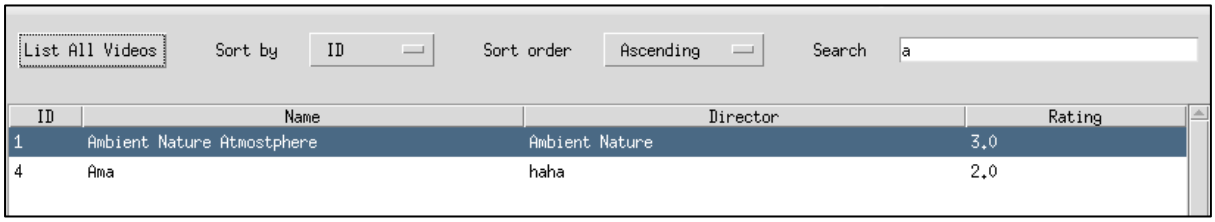
Search Engine Tests

Test1

Before

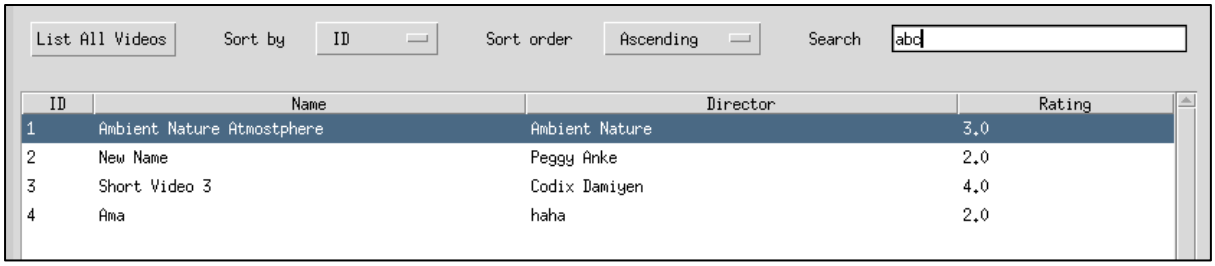


After



Test2

Before

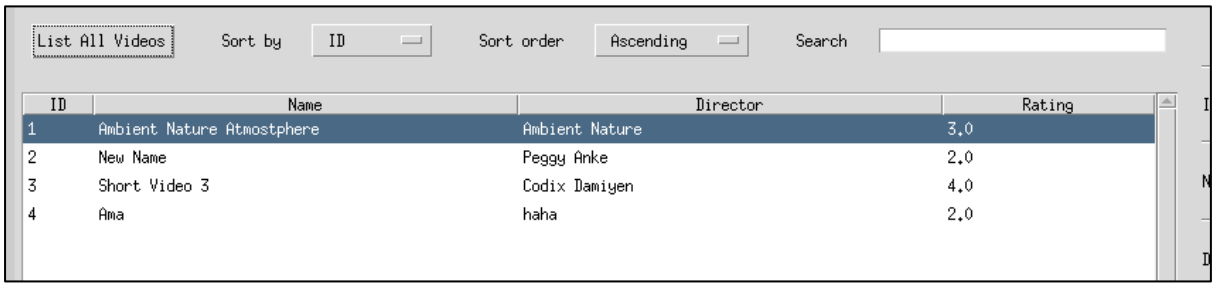


After

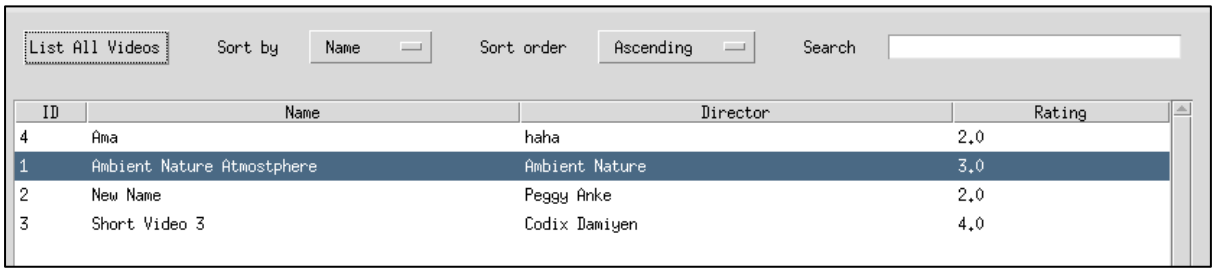
Sorting Tests

Test1

Before

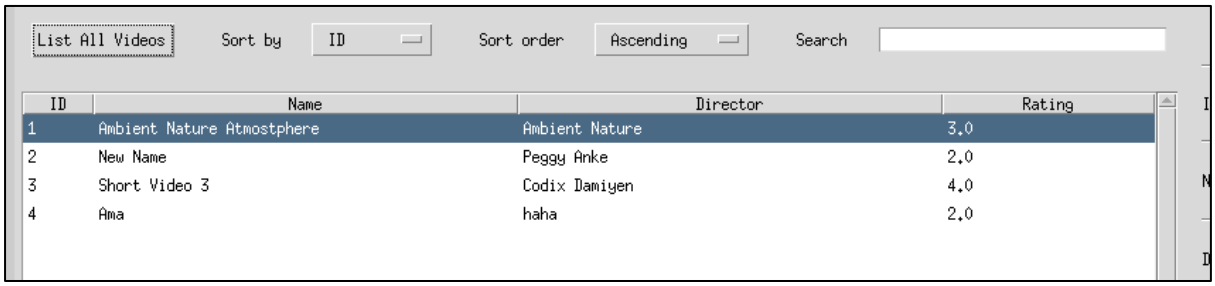


After

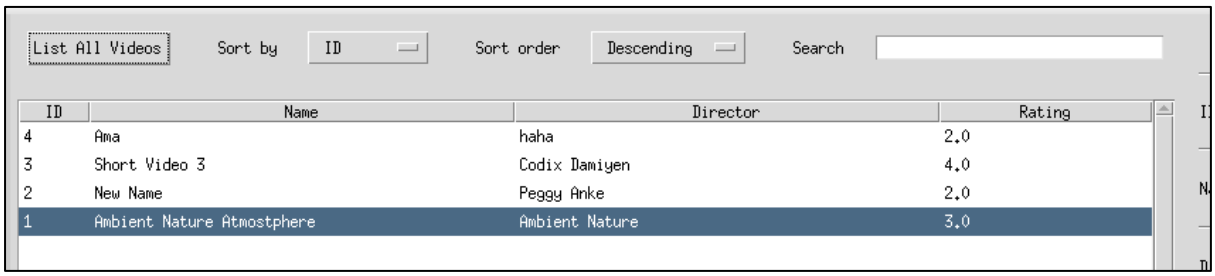


Test2

Before

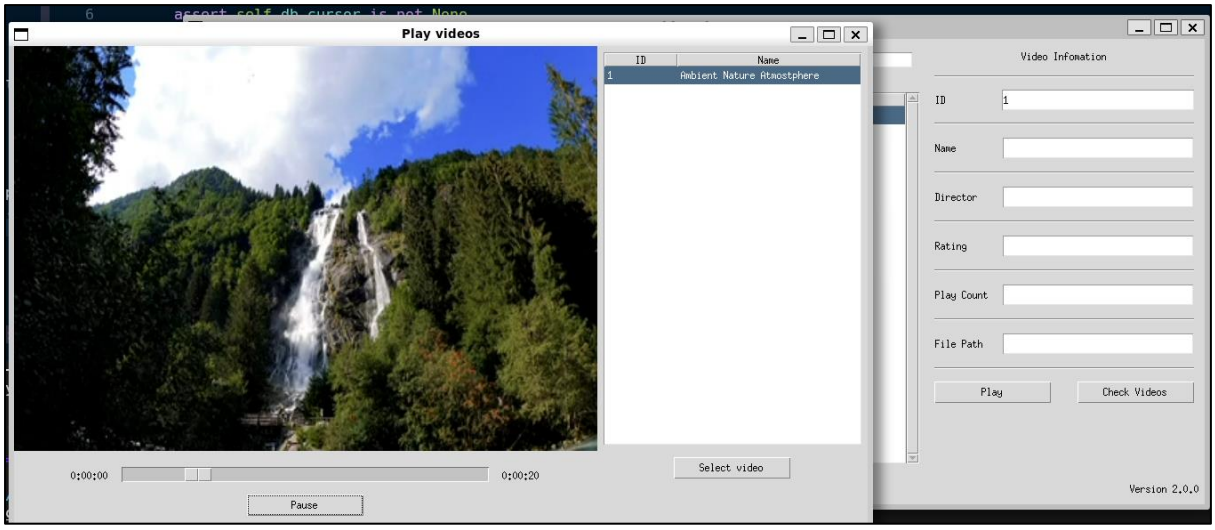


After



Play Functionality Tests

Test 1



Test 2