[COMP1752 COURSEWORK]

Ho Sy Minh Ha - 001306469

Lecturer: Tran Trong Minh

Ho Sy Minh Ha
001306469

Table of Contents

Stage 0: Project Structures 2

Stage 1: Basic understanding 2

- Singleton 2
- Layout 3
- Check Videos 5

Stage 2: Outline implementation – GUI 5

- Video player – application master 5
- Layout components 7
- App frames 9

Stage 3: Basic working version 11

- Configuration file 11
- Storage 12
- Video Object 13
- Tkinter Variables 15
- Event handlers 16
- General namespace 19
- Internal handlers 19

Stage 4: Testing and validation 20

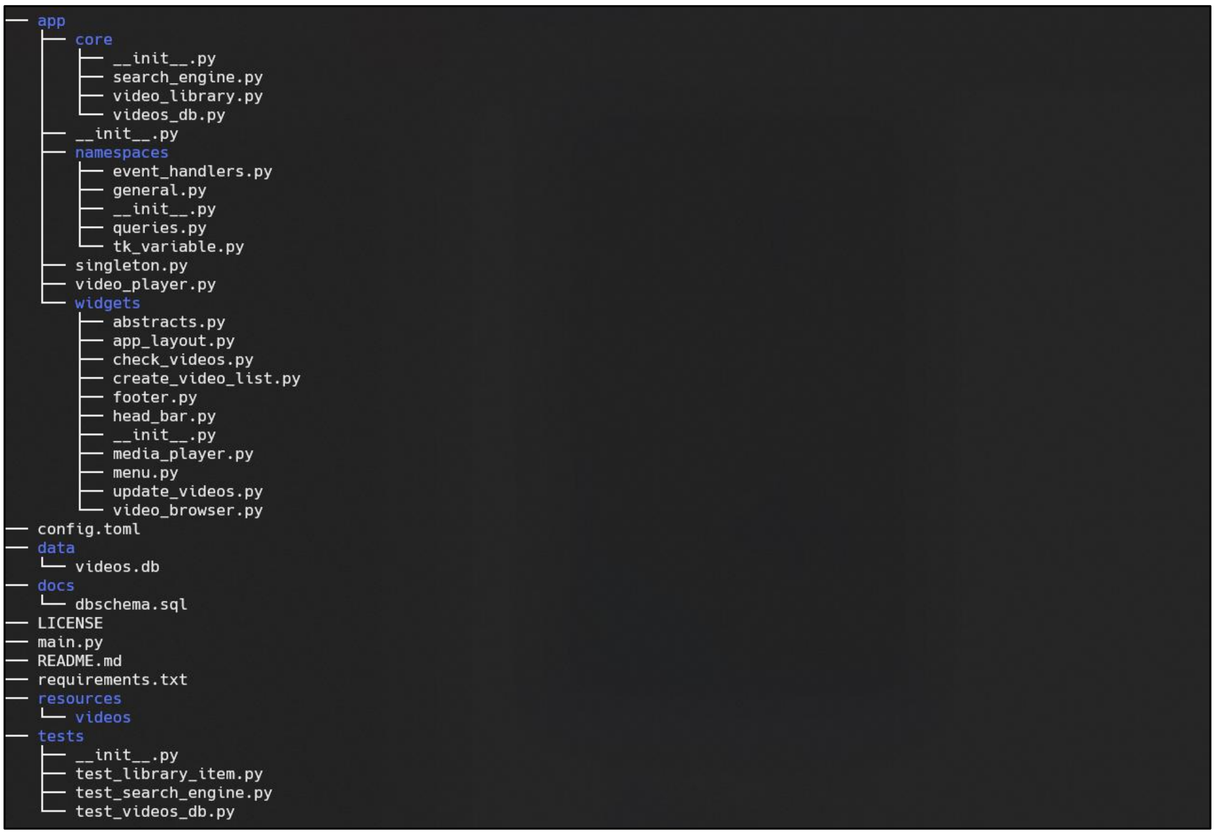
- Tests suit 1: UI Tests 20
- Test suit 2: CORE functions tests 23
- Testsuit 3: Functionalities tests 27

Stage 5: Innovation 49

- Search engine 49
- Sort order 50
- Play video 51

CONCLUSION: 57

Stage 0: Project Structures



```
── app
   ├── core
   │   ├── __init__.py
   │   ├── search_engine.py
   │   ├── video_library.py
   │   └── videos_db.py
   ├── __init__.py
   ├── namespaces
   │   ├── event_handlers.py
   │   ├── general.py
   │   ├── __init__.py
   │   ├── queries.py
   │   └── tk_variable.py
   ├── singleton.py
   ├── video_player.py
   └── widgets
       ├── abstracts.py
       ├── app_layout.py
       ├── check_videos.py
       ├── create_video_list.py
       ├── footer.py
       ├── head_bar.py
       ├── __init__.py
       ├── media_player.py
       ├── menu.py
       ├── update_videos.py
       └── video_browser.py
── config.toml
── data
   └── videos.db
── docs
   └── dbschema.sql
── LICENSE
── main.py
── README.md
── requirements.txt
── resources
   └── videos
── tests
   ├── __init__.py
   ├── test_library_item.py
   ├── test_search_engine.py
   └── test_videos_db.py
```

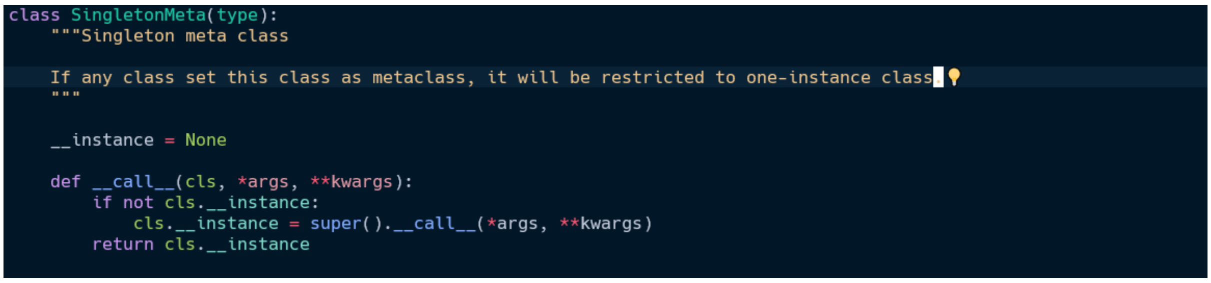
- main.py: the entry point of the application
- app/ : the main module of the application
- data/ contains data that the application yield or use
- docs/ documents
- README.md : application usage, please read it before using the application
- config.toml : application configuration
- requirements.txt: pip module requirements
- tests/ : testcases
- LICENSE: MIT

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.



```python
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 1 SingletonMeta (singleton.py) implementation

Layout

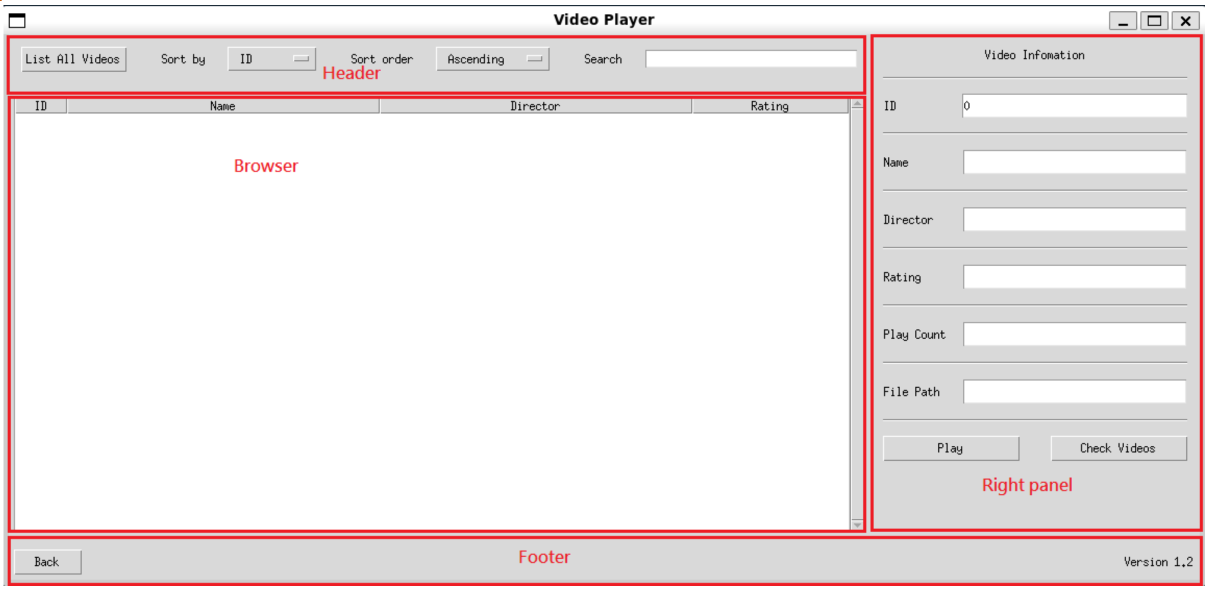
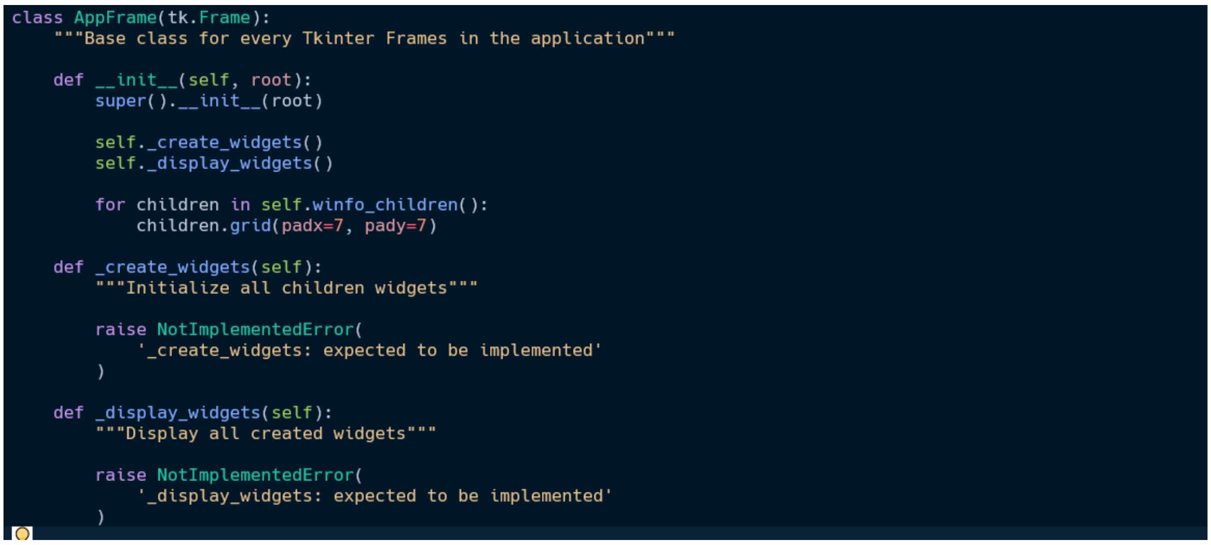


Figure 2 - Layout idea

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

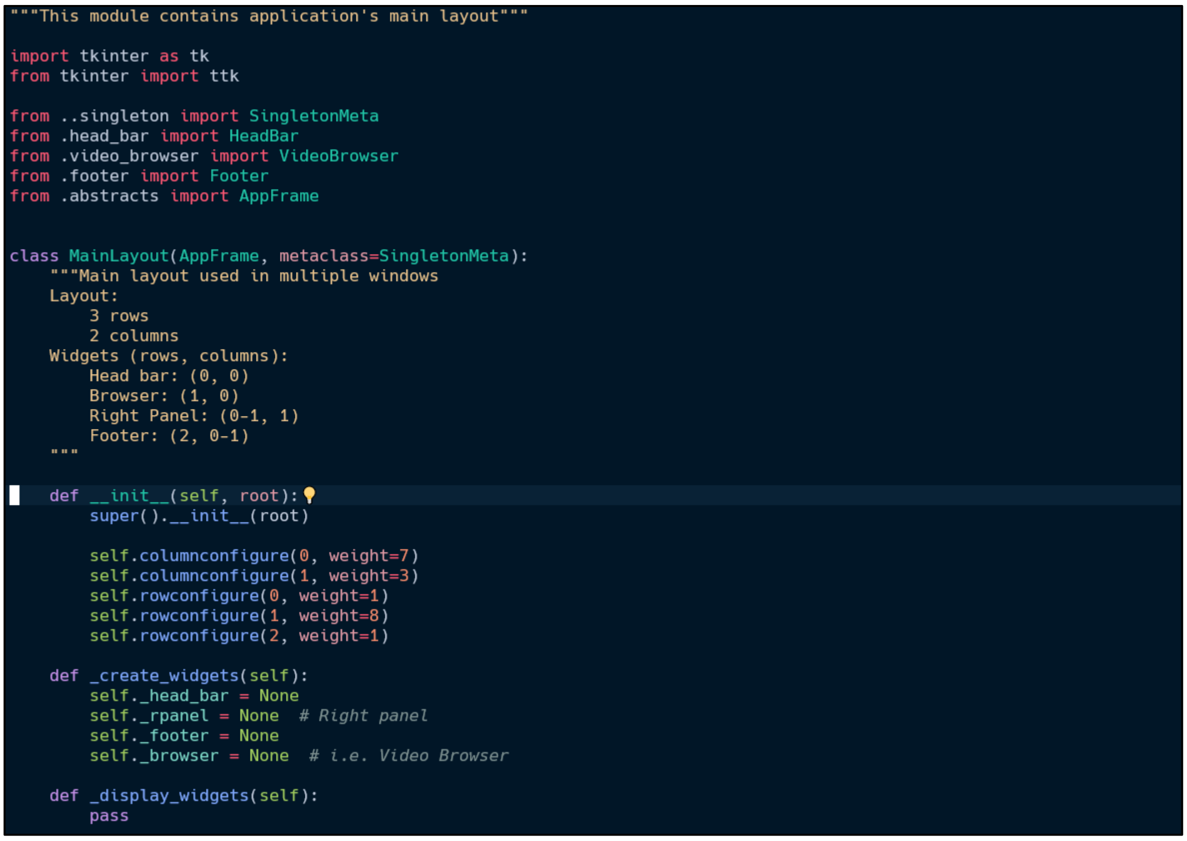
Most of the widgets will inherit from AppFrame (abstracts.py) abstract class



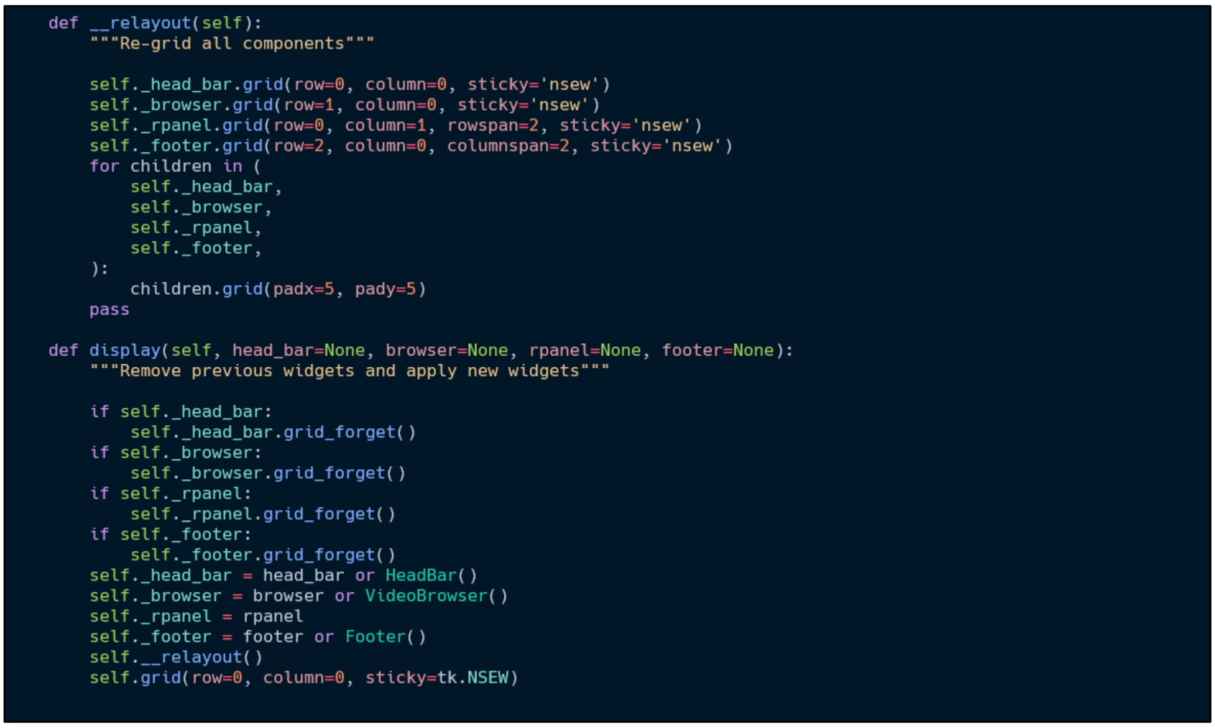
```python
class AppFrame(tk.Frame):
    """Base class for every Tkinter Frames in the application"""

    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 3 - 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
```

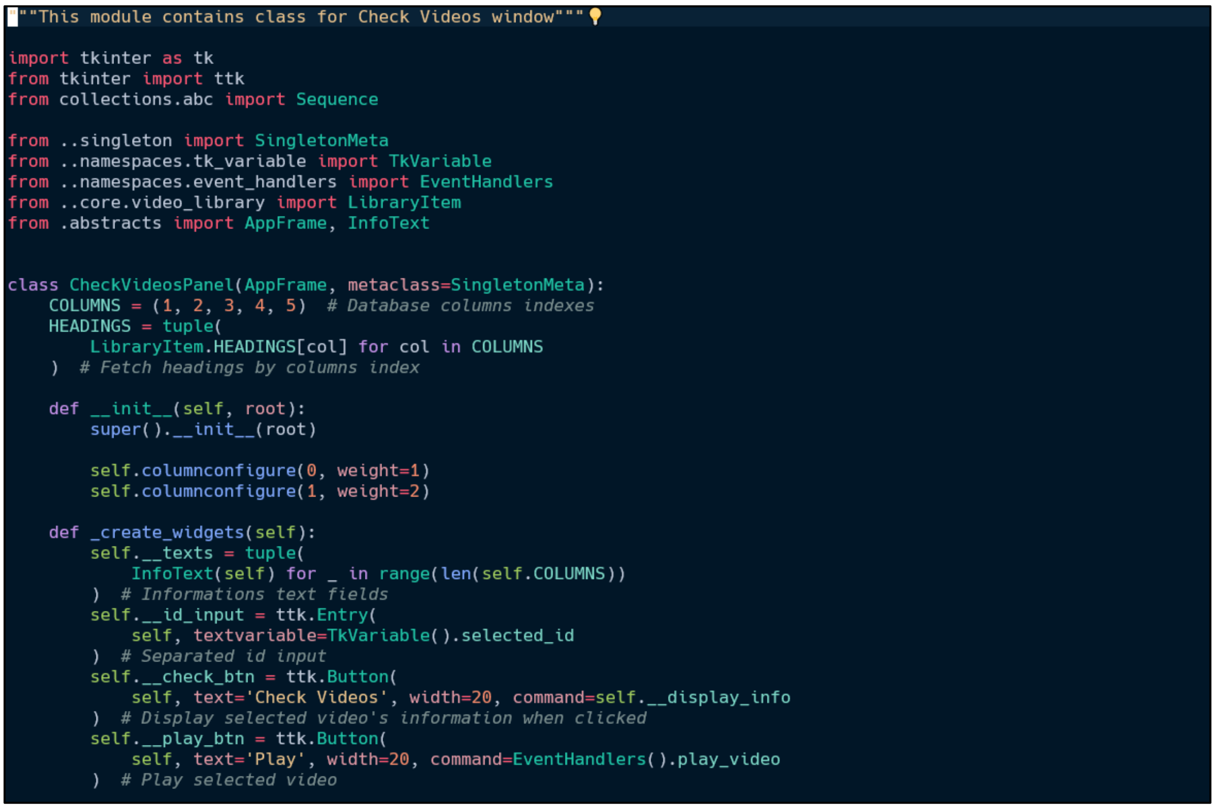


```python
    def __relayout(self):
        """Re-grid all components"""

        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)
```

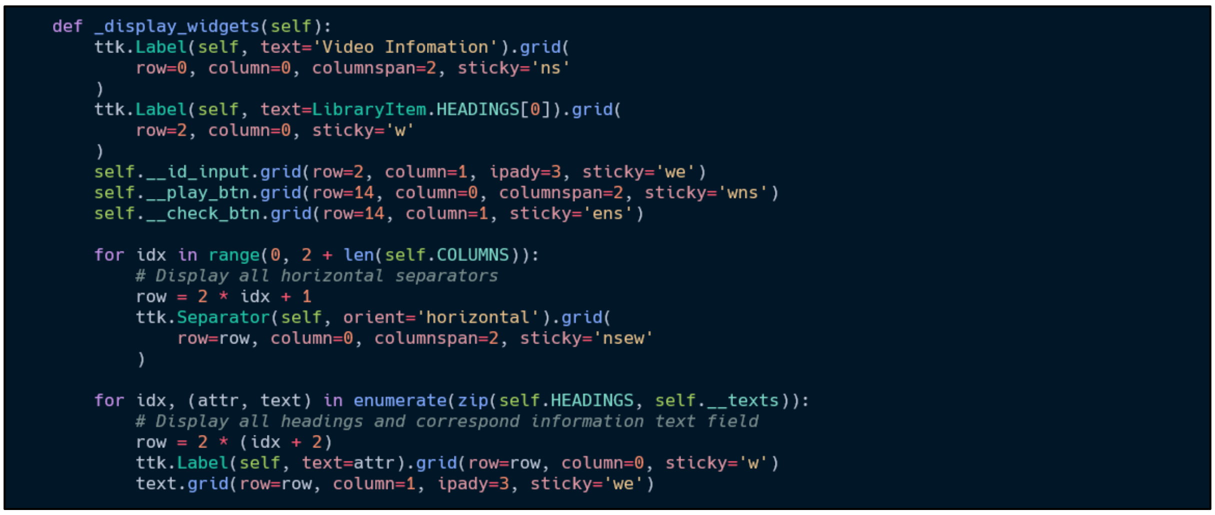
Figure 4 MainLayout (app_layout.py) implementation

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



```
"""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
```



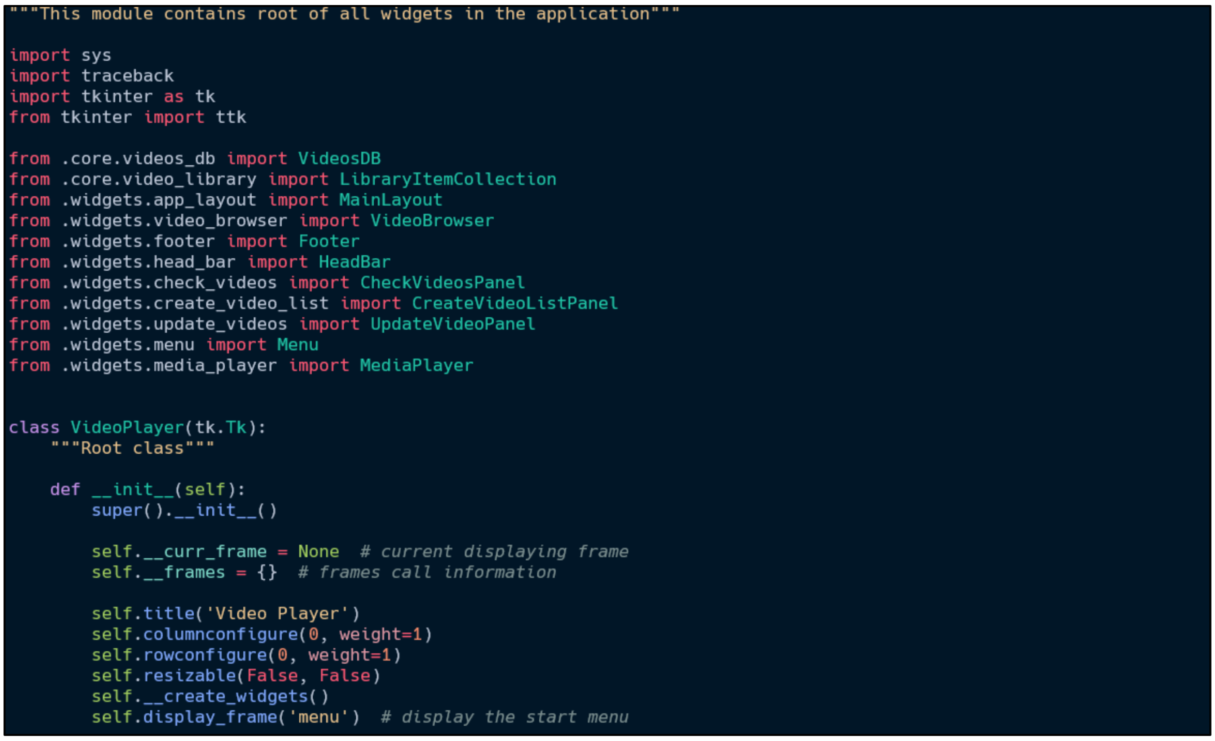
```
    def _display_widgets(self):
        ttk.Label(self, text='Video Infomation').grid(
            row=0, column=0, columnspan=2, sticky='ns'
        )
        ttk.Label(self, text=LibraryItem.HEADINGS[0]).grid(
            row=2, column=0, sticky='w'
        )
        self.__id_input.grid(row=2, 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 5 - CheckVideosPanel (check_videos.py) implementation without functionalities

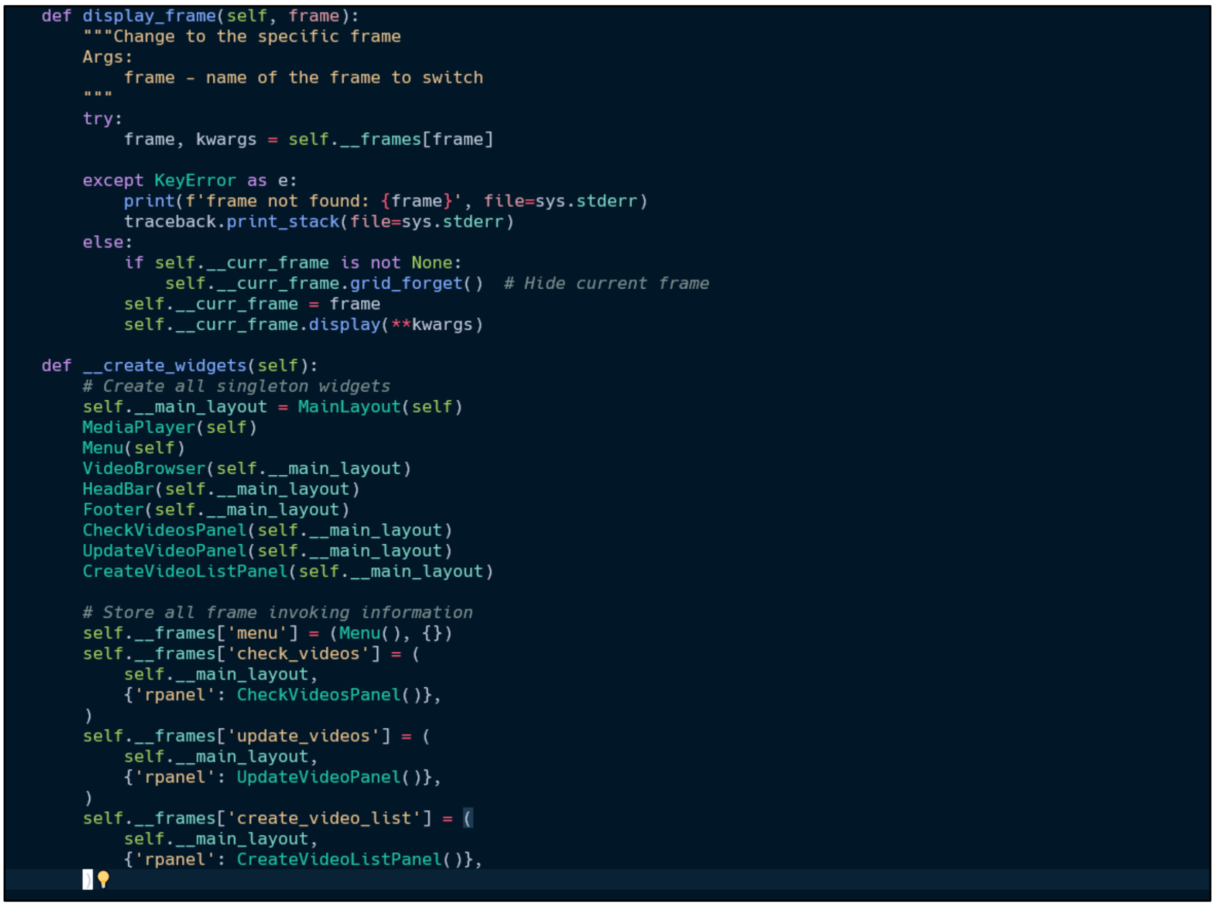
Stage 2: Outline implementation – GUI

Video player – application master

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



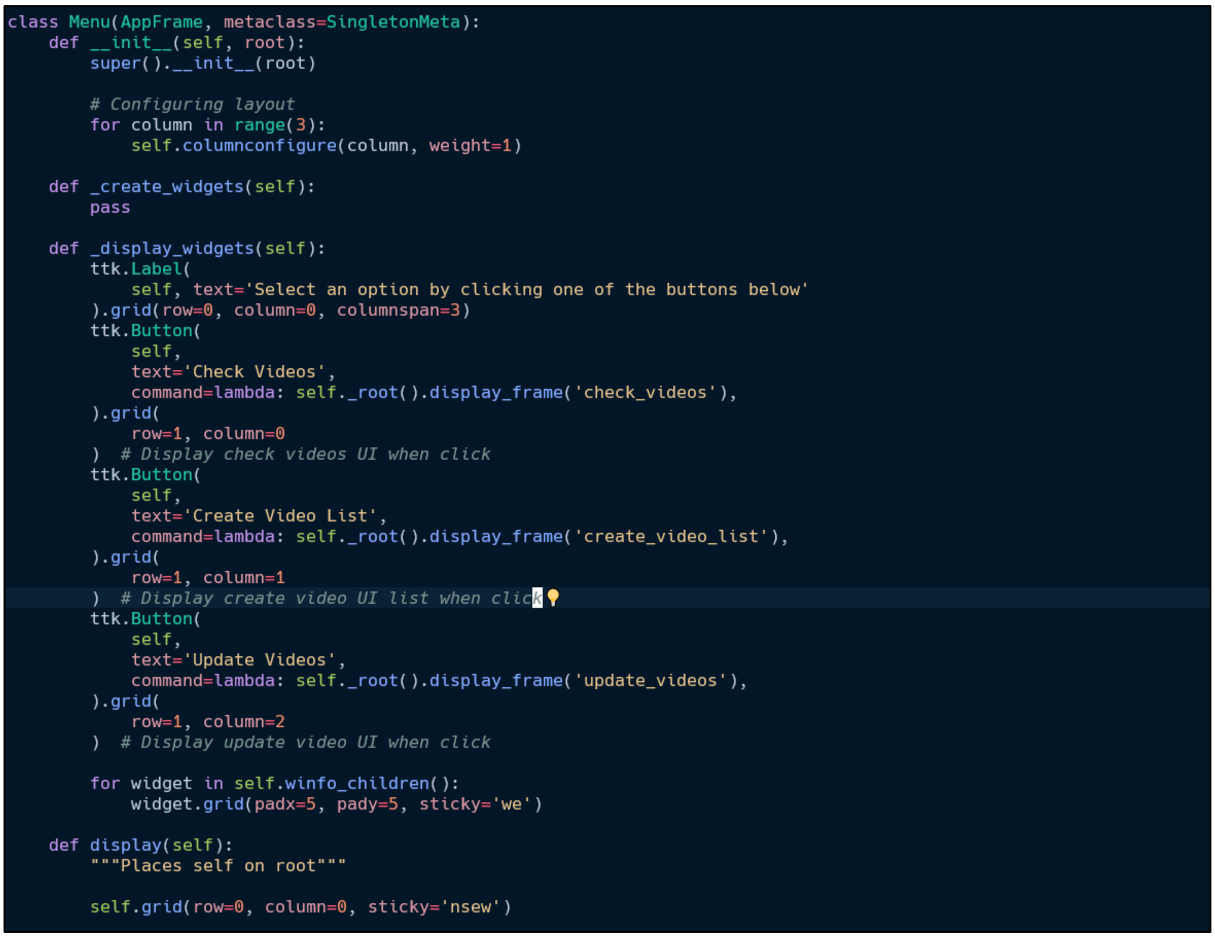
```python
"""This module contains root of all widgets in the application"""

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()},
        )
```

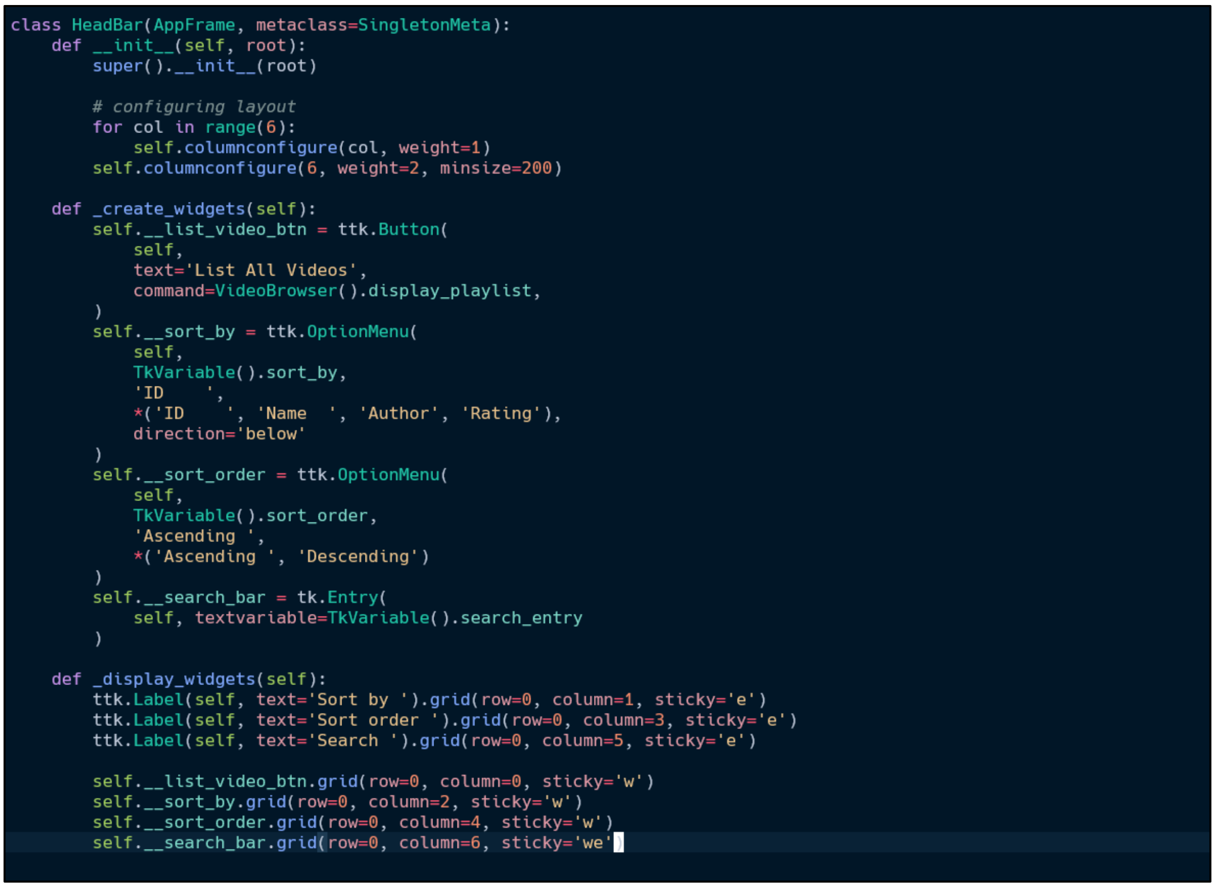
Figure 6 - VideoPlayer (video_player.py) implementation

Layout components



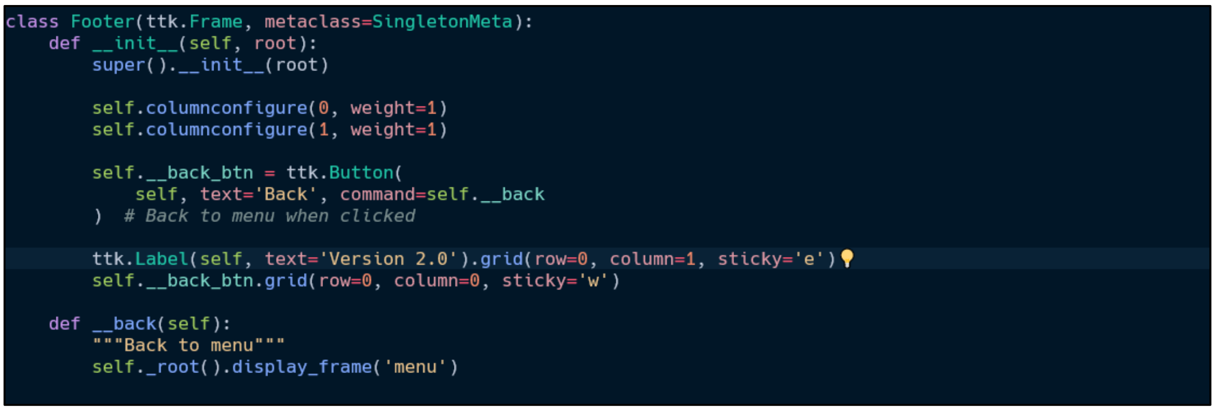
```python
class Menu(AppFrame, metaclass=SingletonMeta):
    def __init__(self, root):
        super().__init__(root)

        # 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 7 - Menu (menu.py) implementation



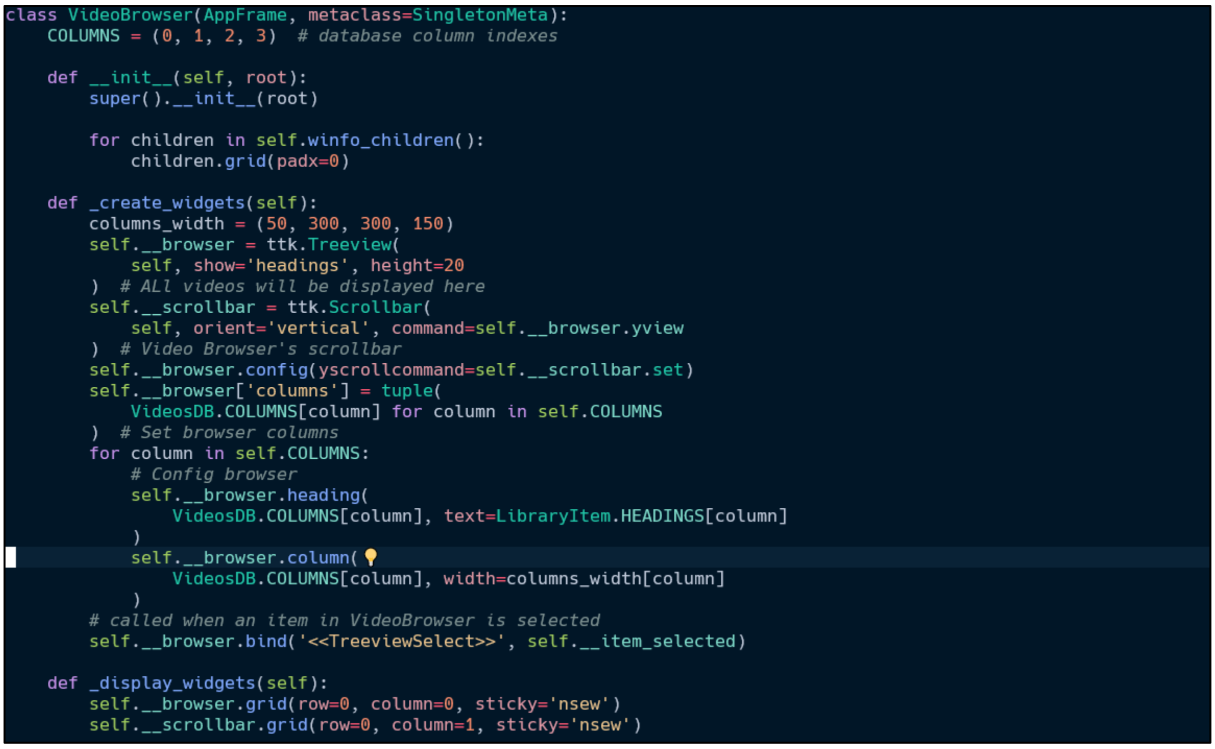
```python
class HeadBar(AppFrame, metaclass=SingletonMeta):
    def __init__(self, root):
        super().__init__(root)

        # 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 8 - Head bar (head_bar.py) implementation



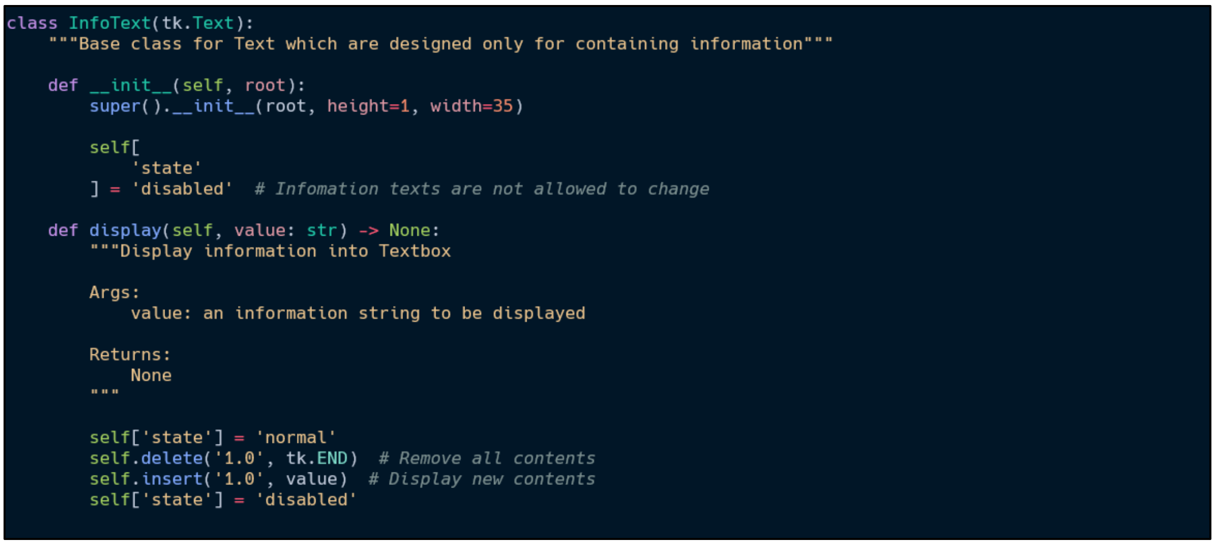
```python
class Footer(ttk.Frame, metaclass=SingletonMeta):
    def __init__(self, root):
        super().__init__(root)

        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 9 – 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 10 - Video Browser (video_browser.py) implementation without functionalities

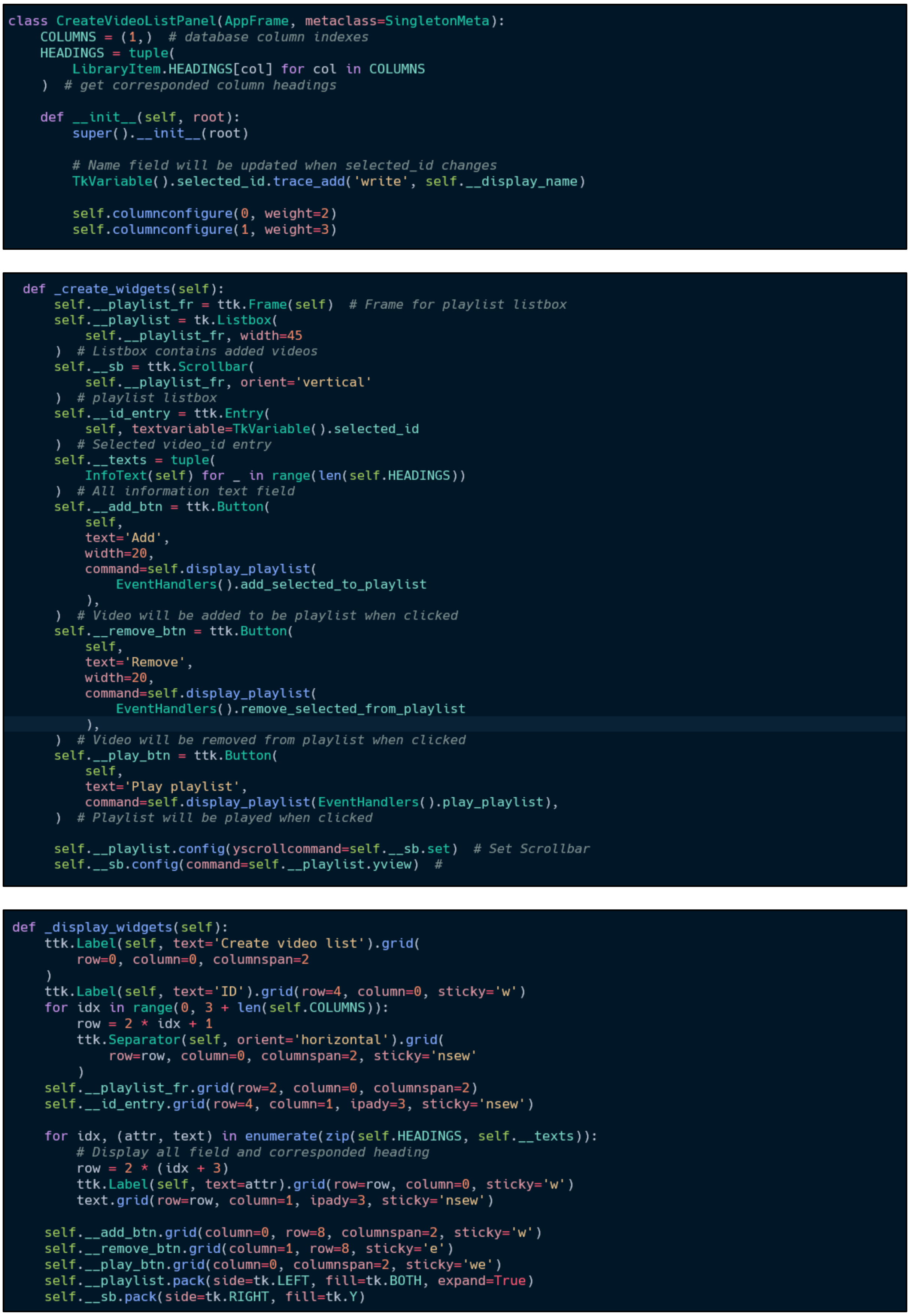


```python
class InfoText(tk.Text):
    """Base class for Text which are designed only for containing information"""

    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 11 - InfoText (abstracts.py) implementation

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

App frames

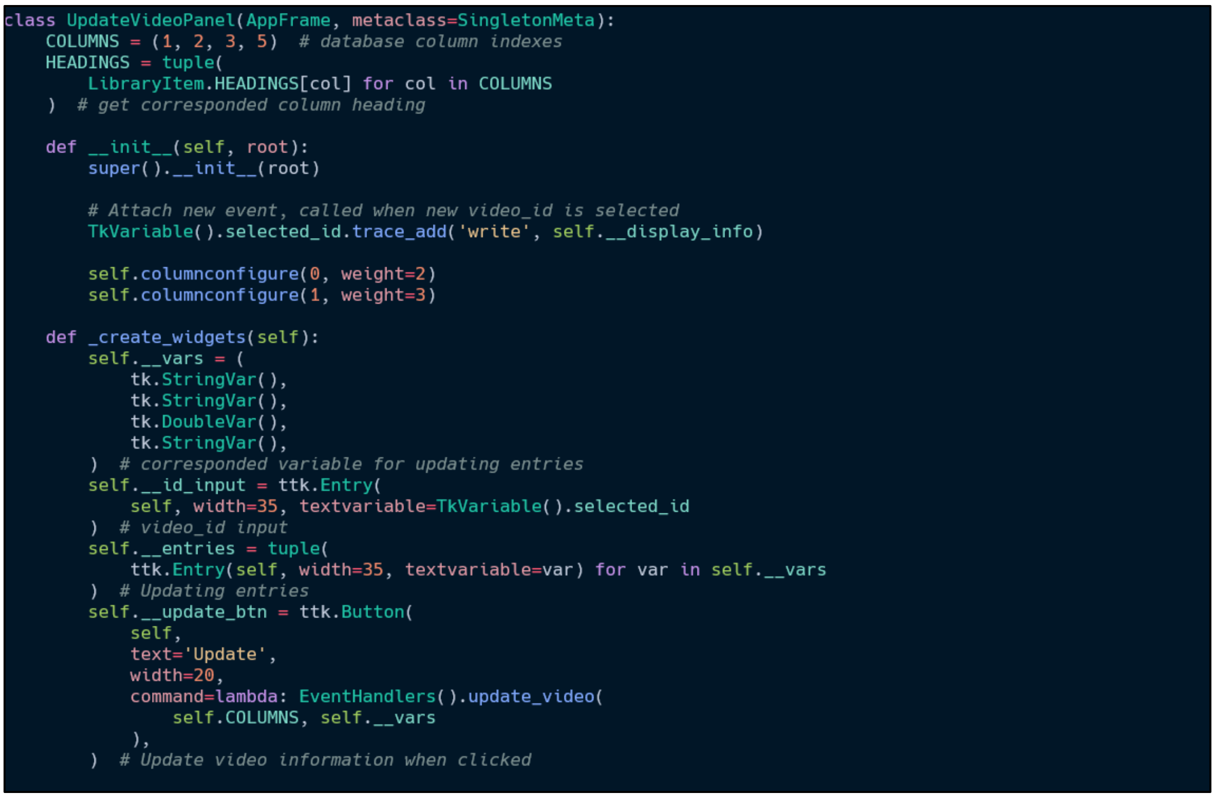


```python
class CreateVideoListPanel(AppFrame, metaclass=SingletonMeta):
    COLUMNS = (1,)  # database column indexes
    HEADINGS = tuple(
        LibraryItem.HEADINGS[col] for col in COLUMNS
    )  # get corresponded column headings

    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)
```

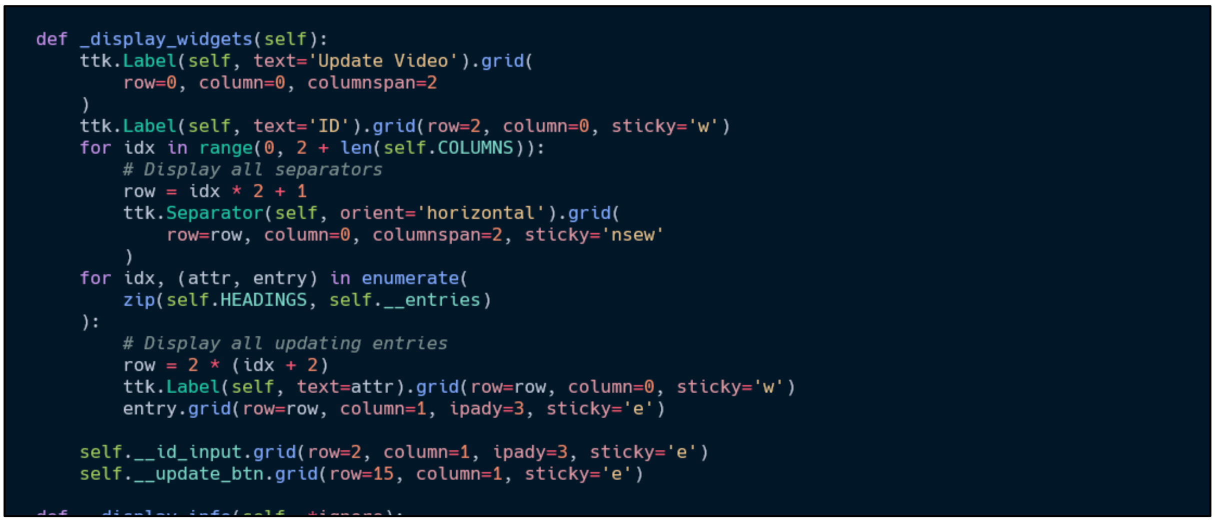
```python
  def _create_widgets(self):
      self.__playlist_fr = ttk.Frame(self)  # Frame for playlist listbox
      self.__playlist = tk.Listbox(
          self.__playlist_fr, width=45
      )  # Listbox contains added videos
      self.__sb = ttk.Scrollbar(
          self.__playlist_fr, orient='vertical'
      )  # 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 12 - 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
```



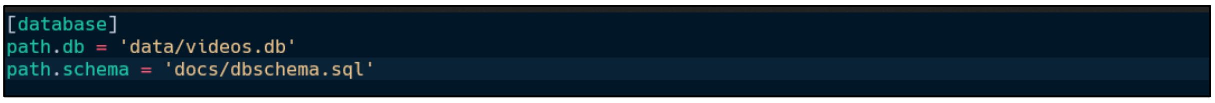
```python
    def _display_widgets(self):
        ttk.Label(self, text='Update Video').grid(
            row=0, column=0, columnspan=2
        )
        ttk.Label(self, text='ID').grid(row=2, column=0, sticky='w')
        for idx in range(0, 2 + len(self.COLUMNS)):
            # Display all separators
            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')

    def _display_info(self, *ignore):
```

Figure 13 - Update Video Panel (update_video.py) implementation without functionalities

Stage 3: Basic working version

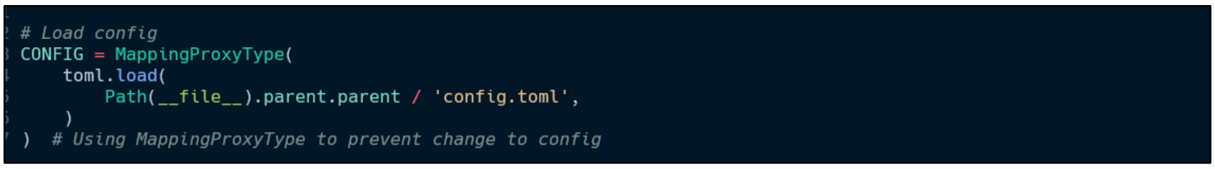
Configuration file

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 14- Configuration (config.toml)

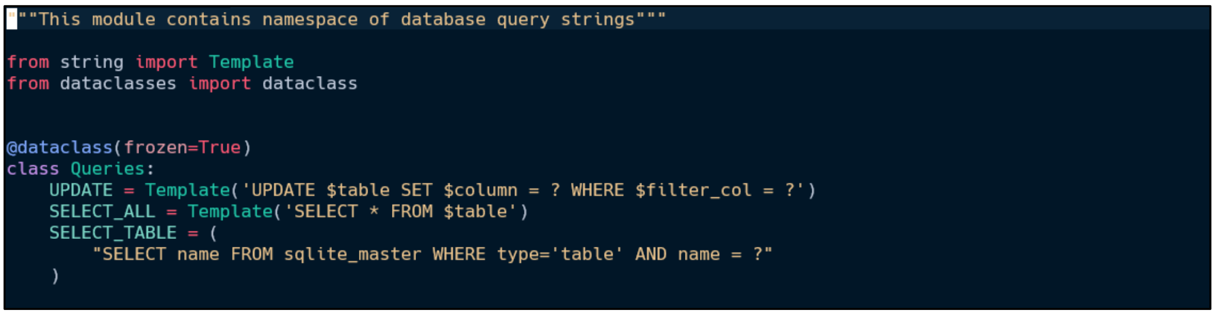


```python
# Load config
CONFIG = MappingProxyType(
    toml.load(
        Path(__file__).parent.parent / 'config.toml',
    )
)  # Using MappingProxyType to prevent change to config
```

Figure 15 - Config (app/__init__.py) is read as a global variable

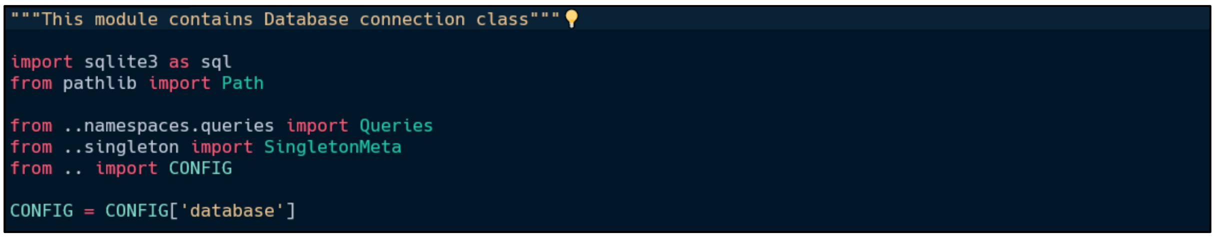
Storage

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

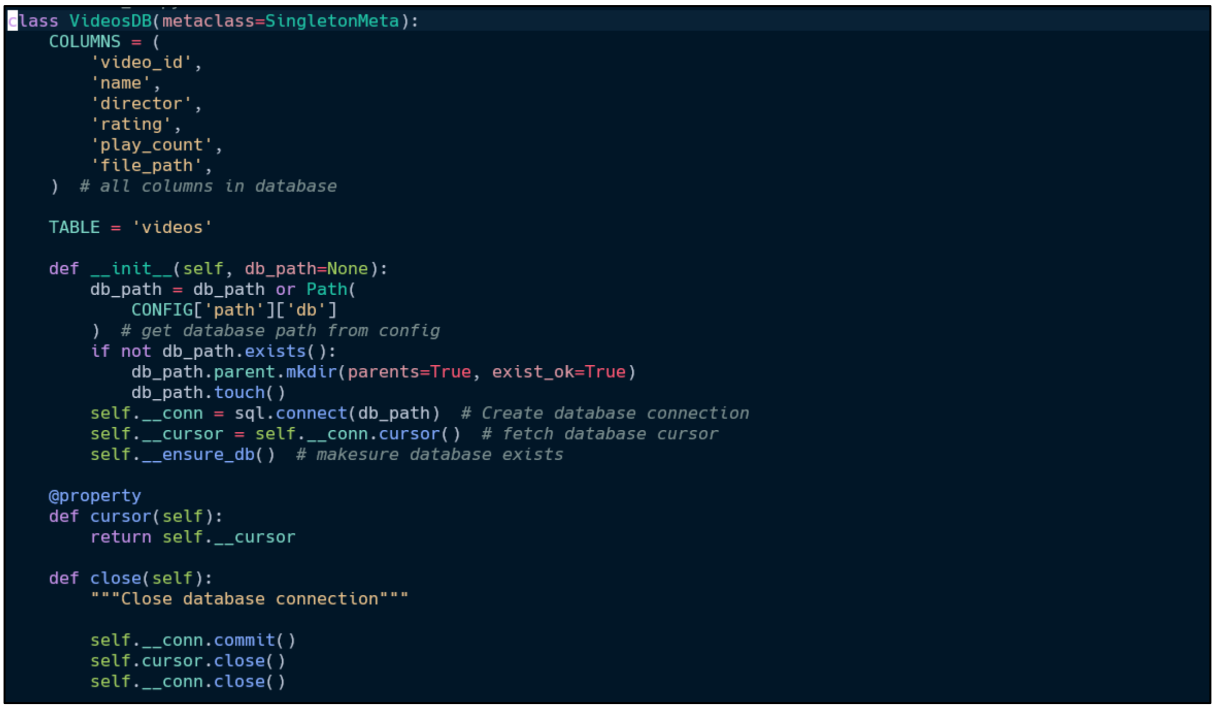


```python
"""This module contains namespace of database query strings"""

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 = ?"
    )
```

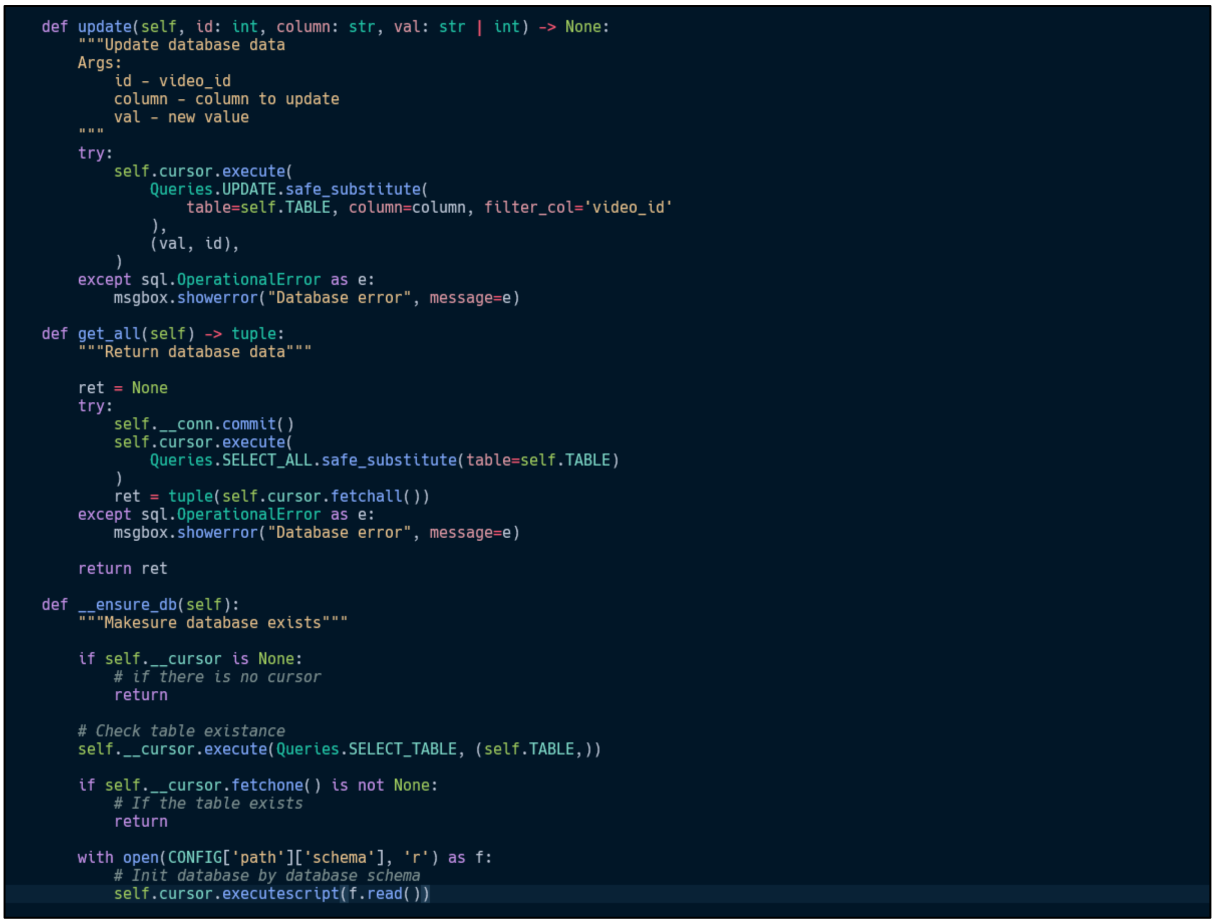
Figure 16 - SQL Queries (queries.py) is managed in a separated namespace



```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()
```



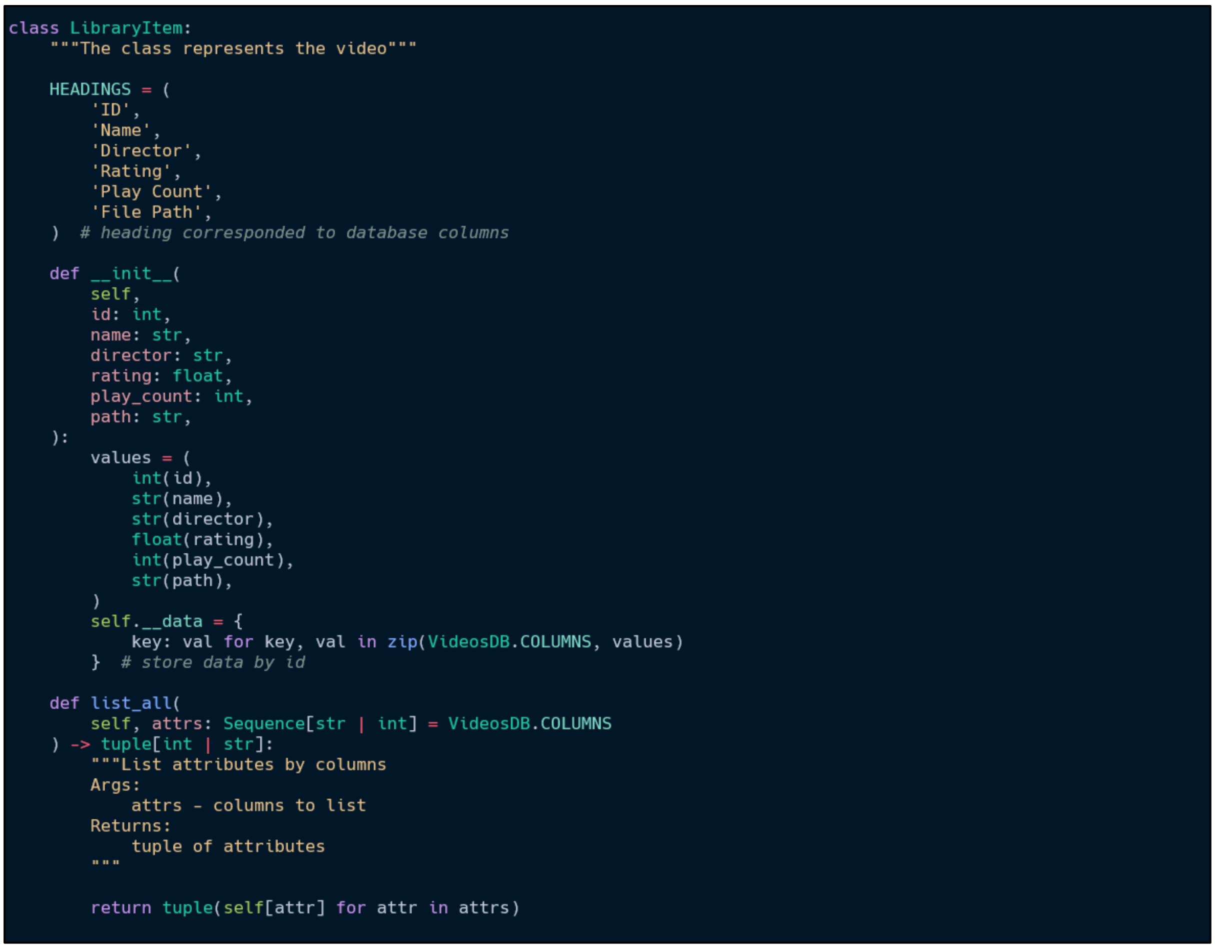
```
    def update(self, id: int, column: str, val: str | int) -> None:
        """Update database data
        Args:
            id - video_id
            column - column to update
            val - new value
        """
        try:
            self.cursor.execute(
                Queries.UPDATE.safe_substitute(
                    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 17 - VideosDB (videos_db.py) implementation

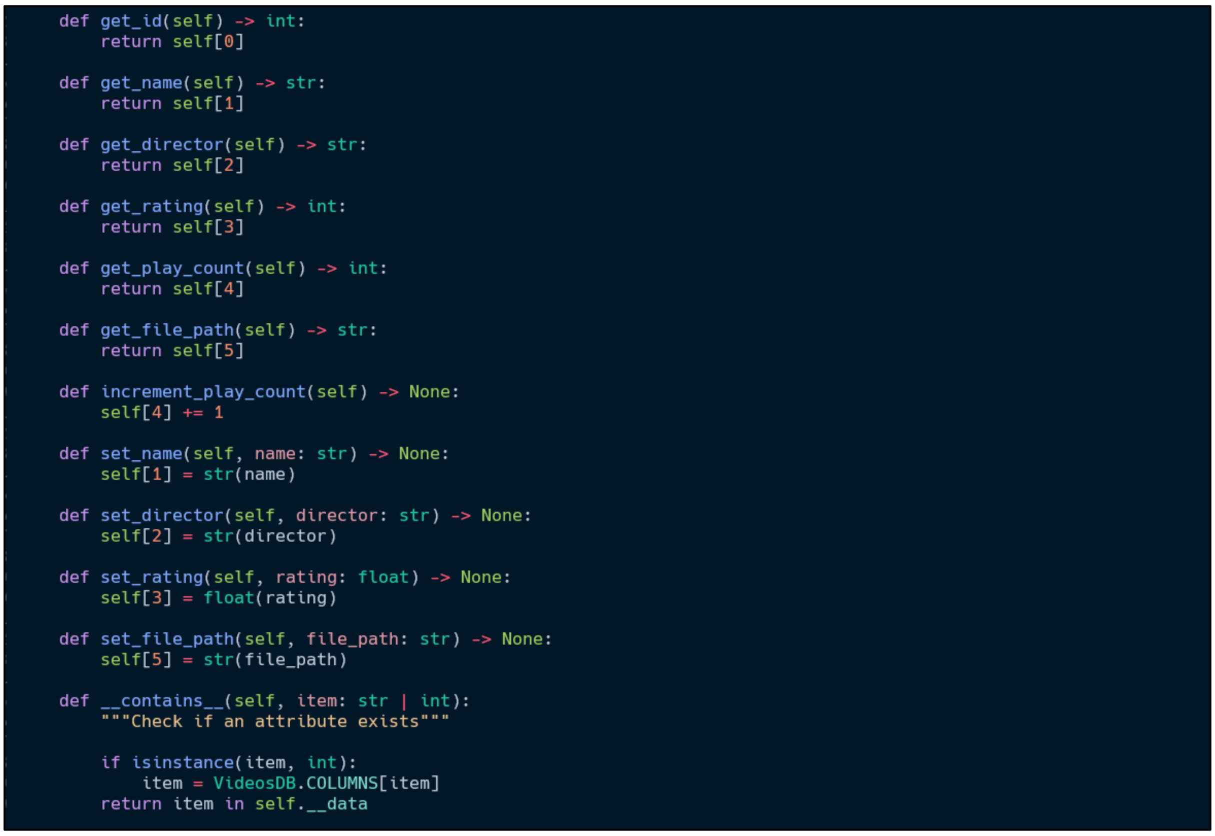
Video Object

Video Library

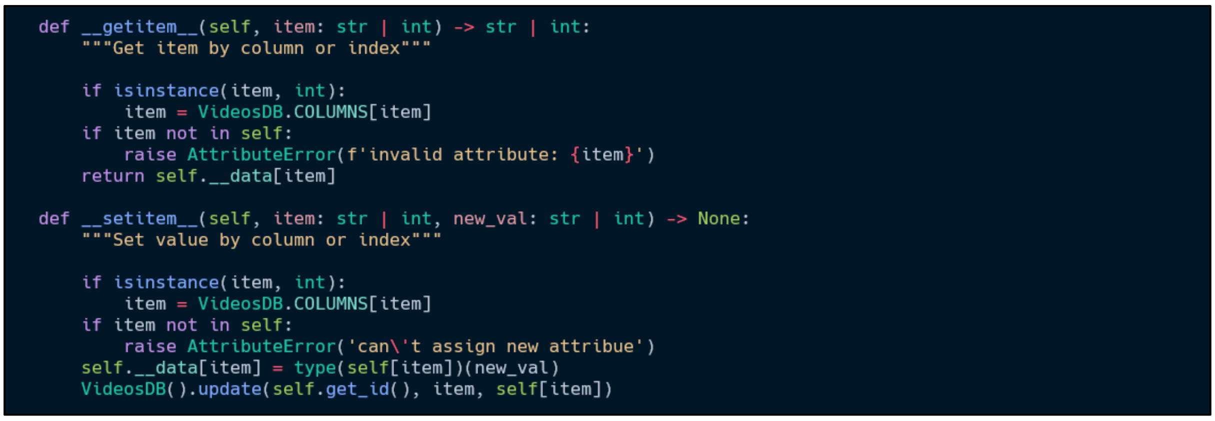
- This object contains a single video information



```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
```

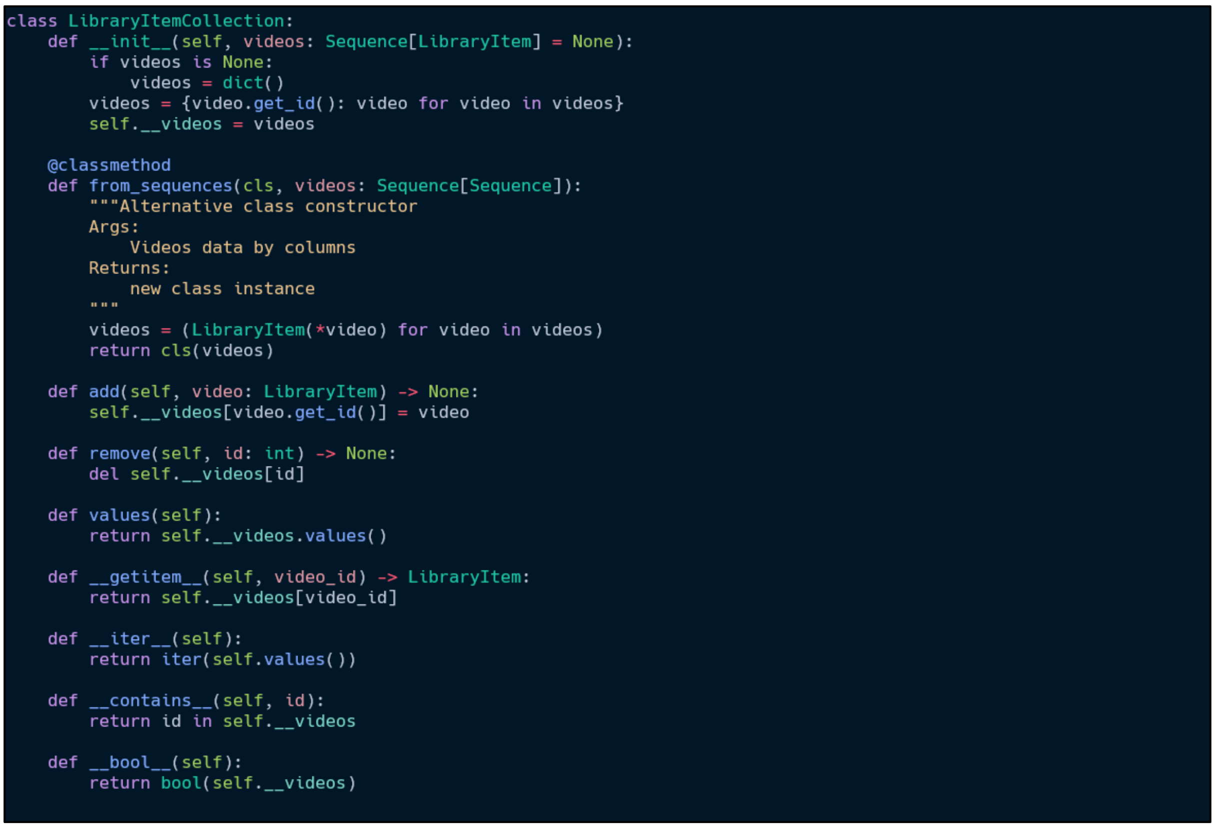


```python
    def __getitem__(self, item: str | int) -> str | int:
        """Get item by column or index"""

        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 18 - LibraryItem (video_library.py) implementation

LibraryItemCollection

This object contains a list or a collection of videos



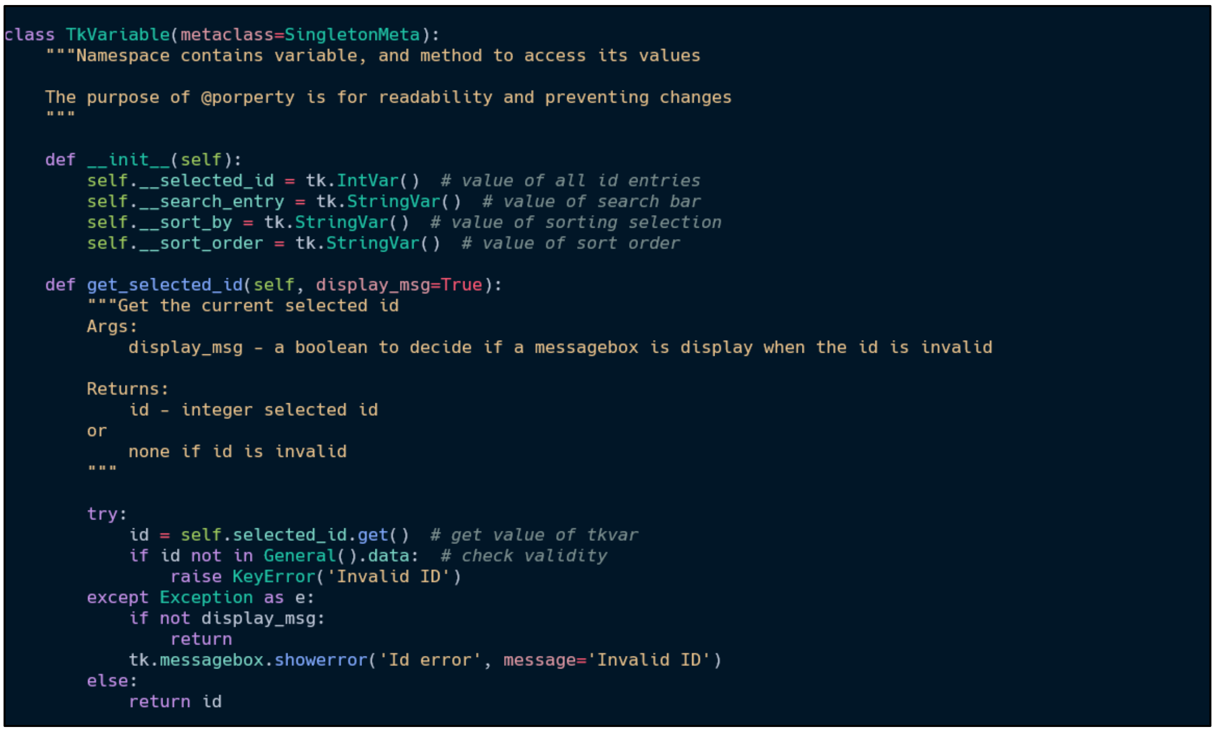
```python
class LibraryItemCollection:
    def __init__(self, videos: Sequence[LibraryItem] = None):
        if videos is None:
            videos = dict()
        videos = {video.get_id(): video for video in videos}
        self.__videos = videos

    @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 19 - LibraryItemCollection (video_library.py) implementation

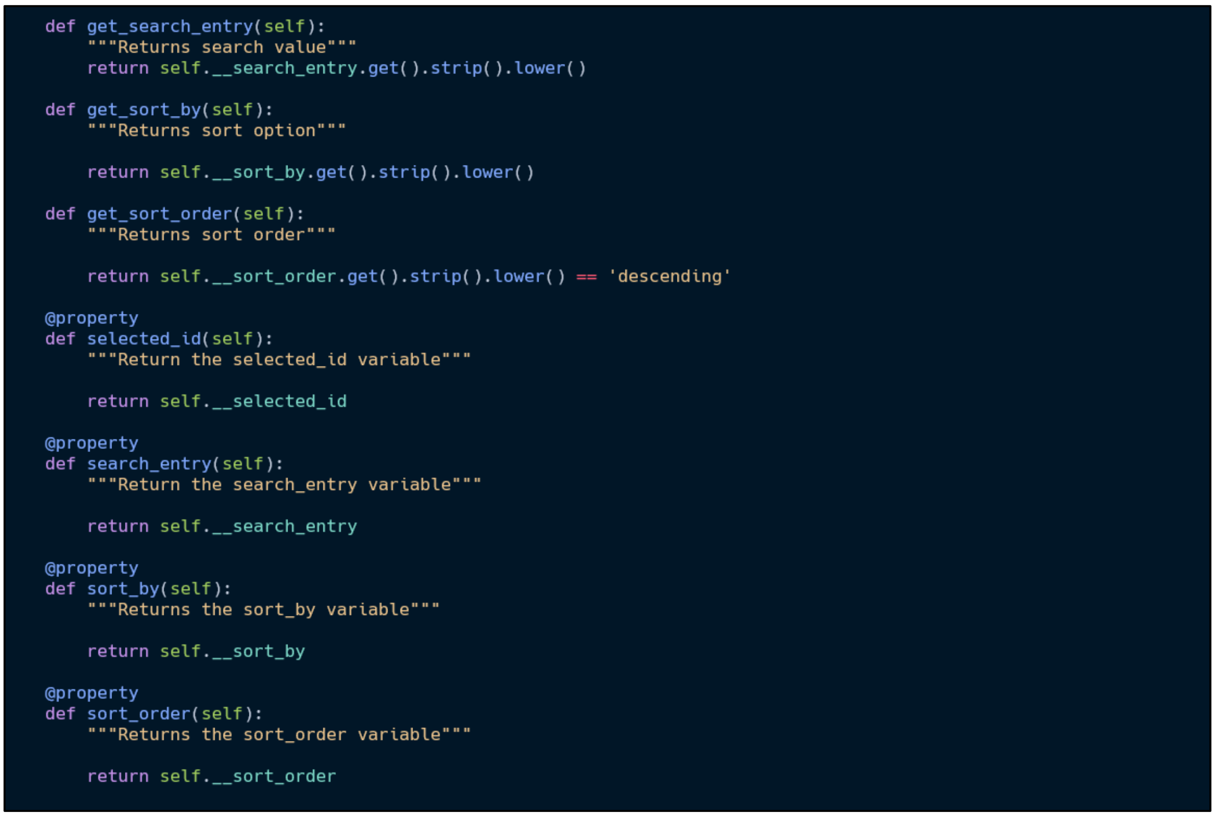
Tkinter Variables

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
```



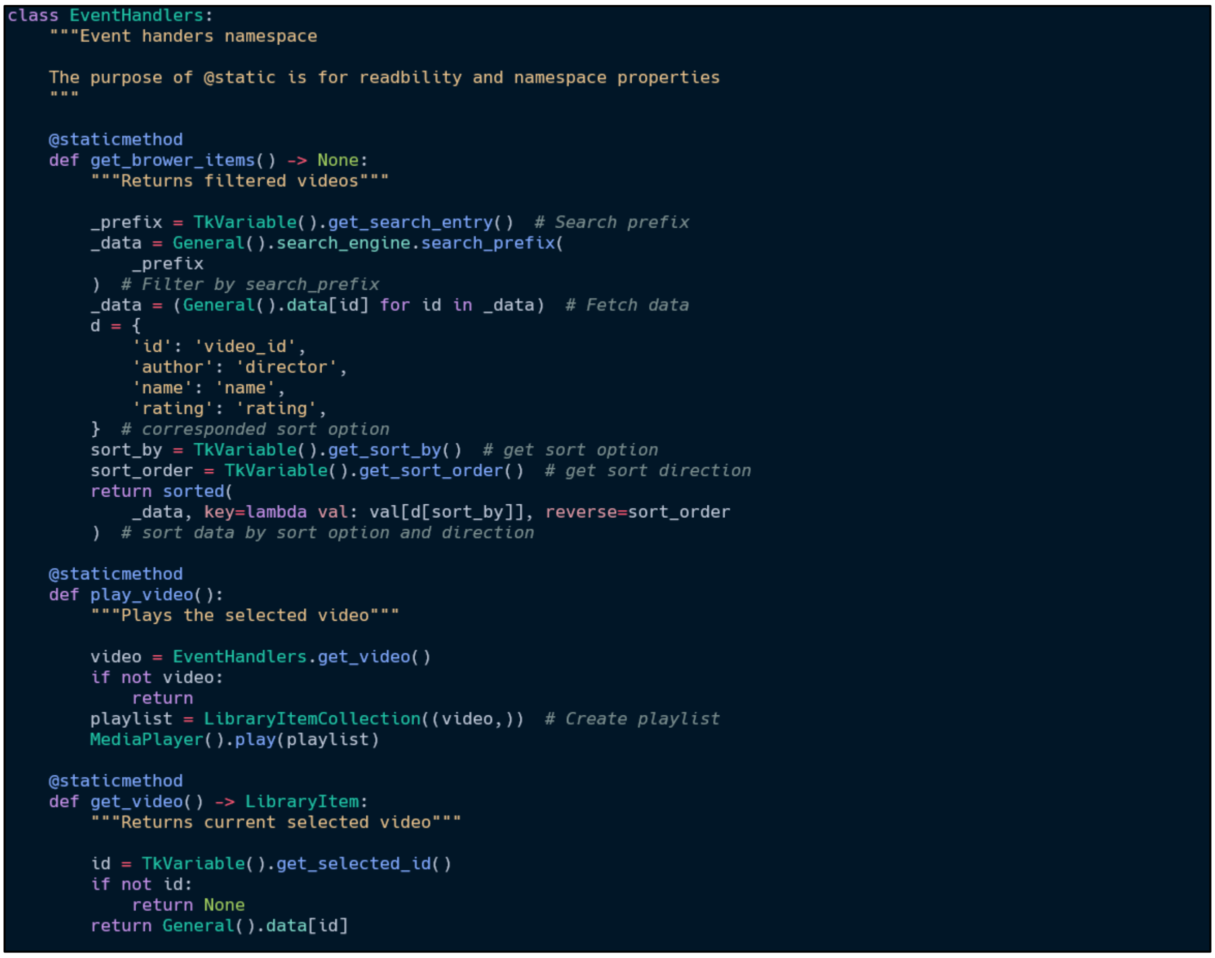
```python
    def get_search_entry(self):
        """Returns search value"""
        return self.__search_entry.get().strip().lower()

    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 20 - TkVariable (tk_variable.py) implementation

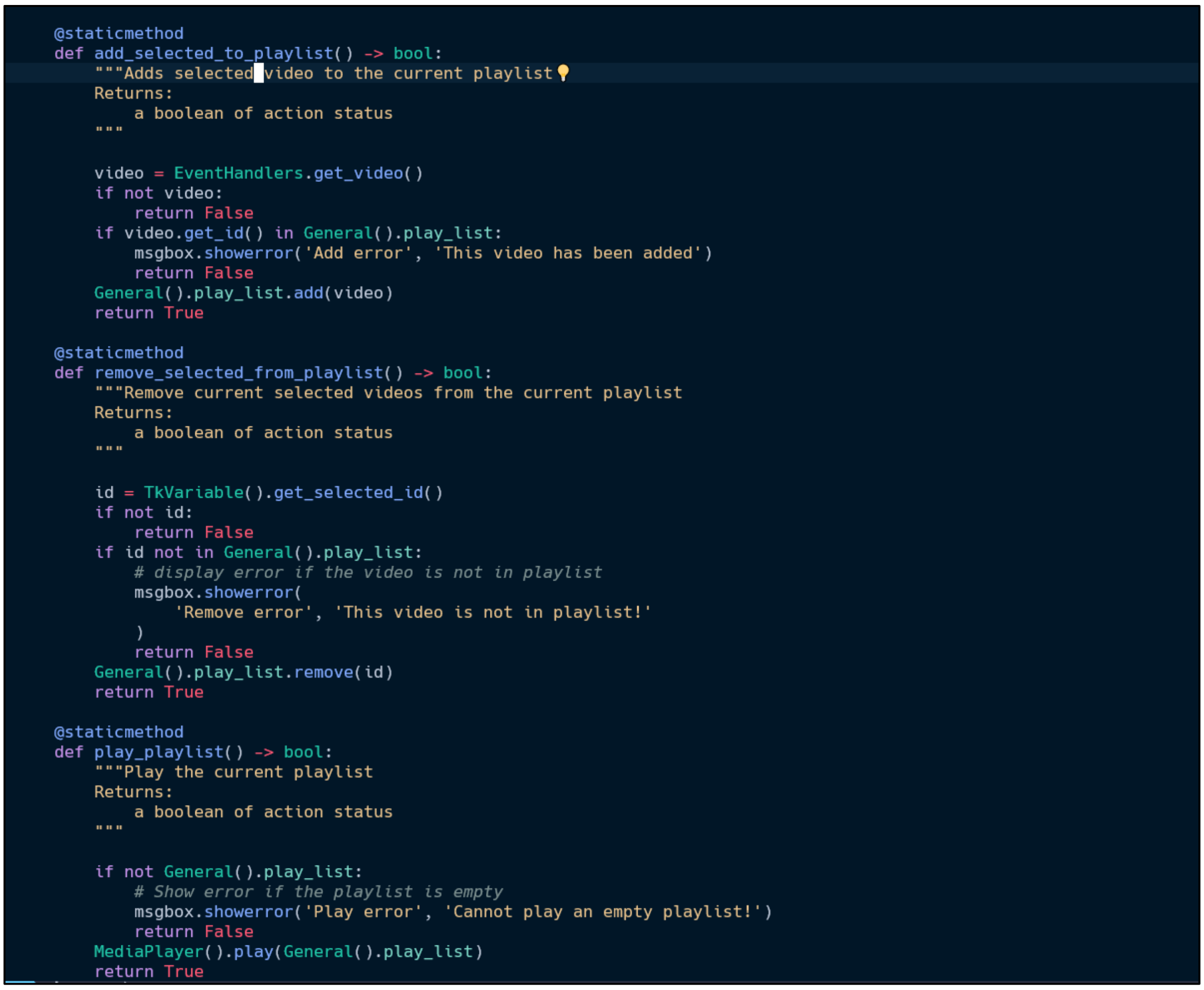
Event handlers

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

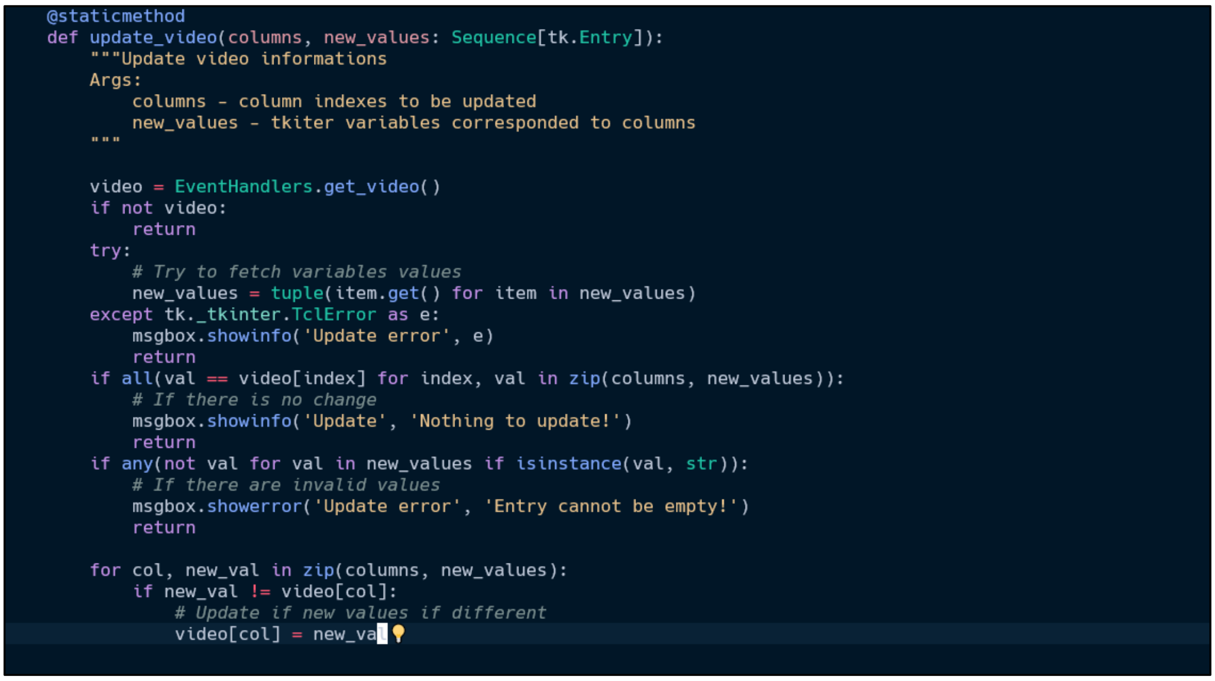
The main dataflow is when a button clicked, the corresponding handler is trigger which will get the validated data from TkVariable, handling and return the result if required



```python
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]
```



```
    @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
```

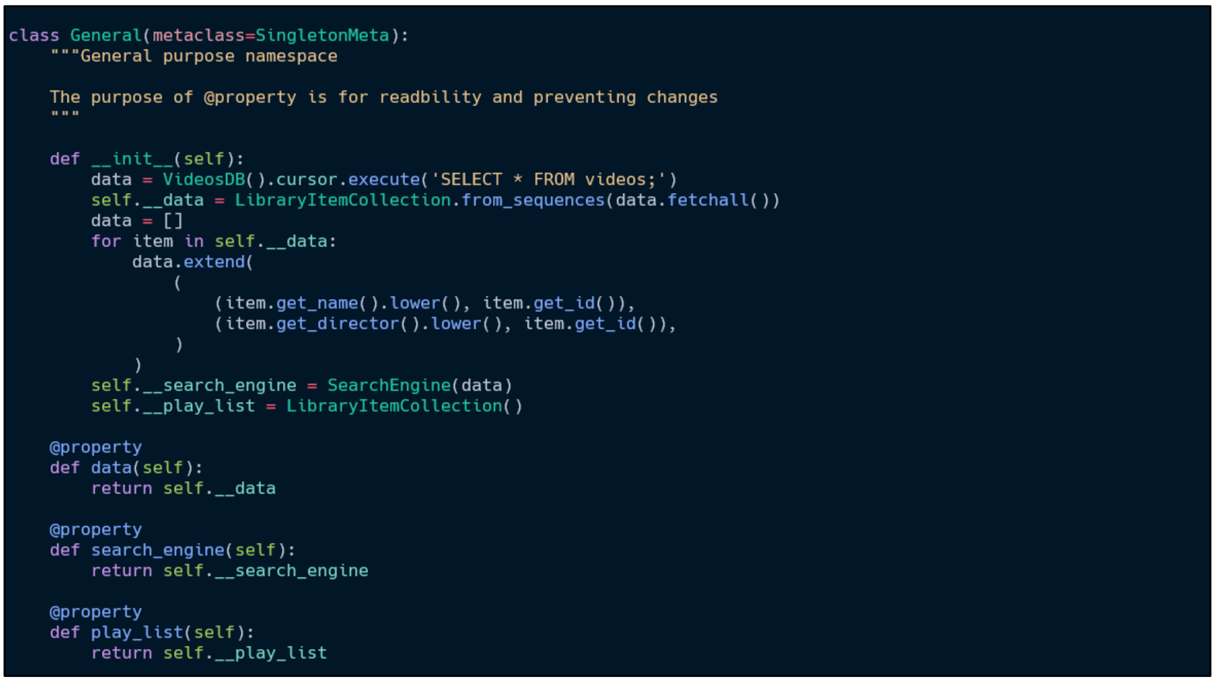


```
    @staticmethod
    def update_video(columns, new_values: Sequence[tk.Entry]):
        """Update video informations
        Args:
            columns - column indexes to be updated
            new_values - tkiter variables corresponded to columns
        """

        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_val
```

Figure 21 - 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



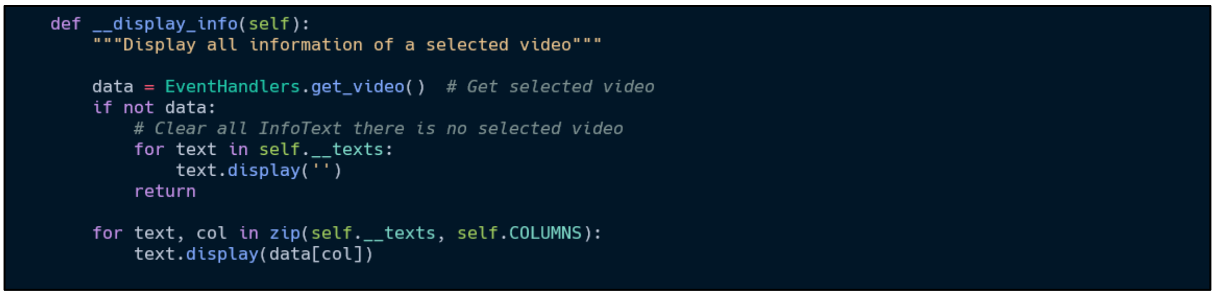
```python
class General(metaclass=SingletonMeta):
    """General purpose namespace

    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 22 - 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

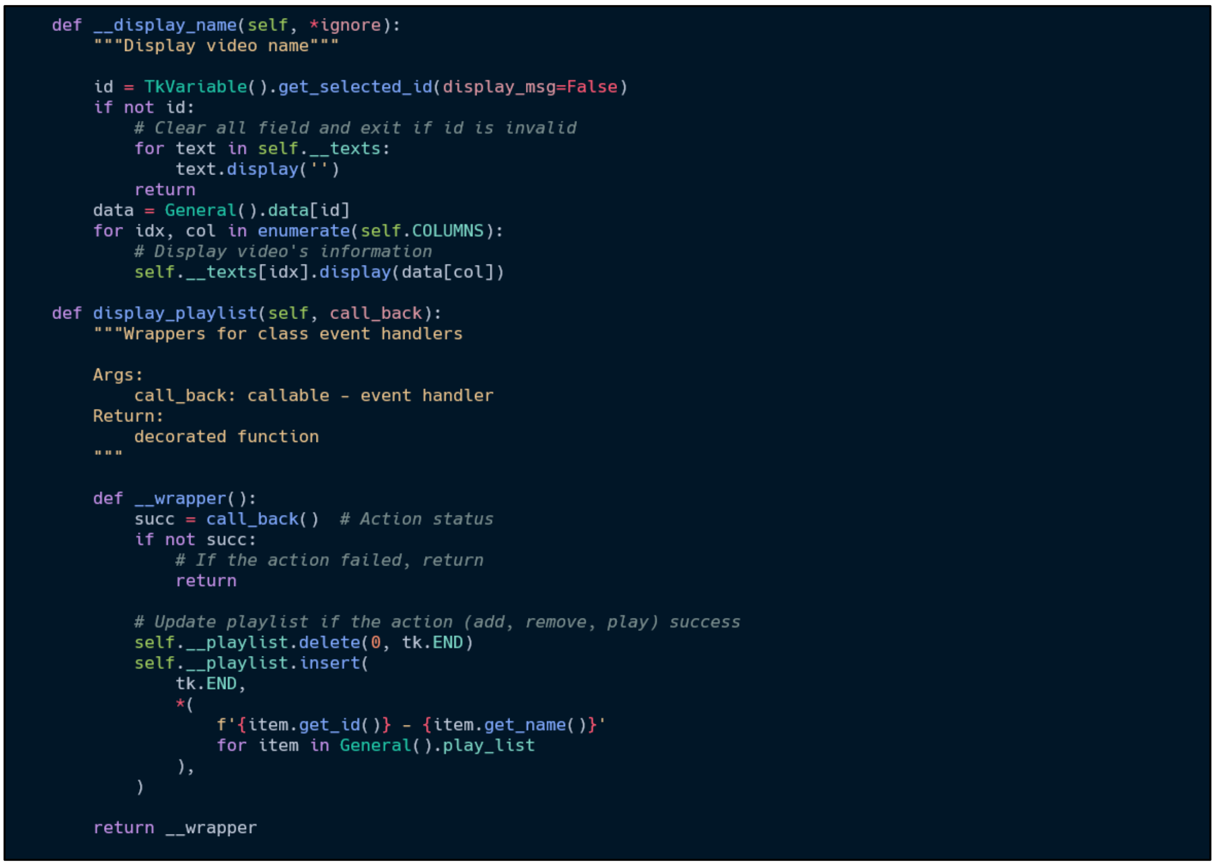
Checkvideos



```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 23 - Internal handler of CheckVideoPanel (check_videos.py)

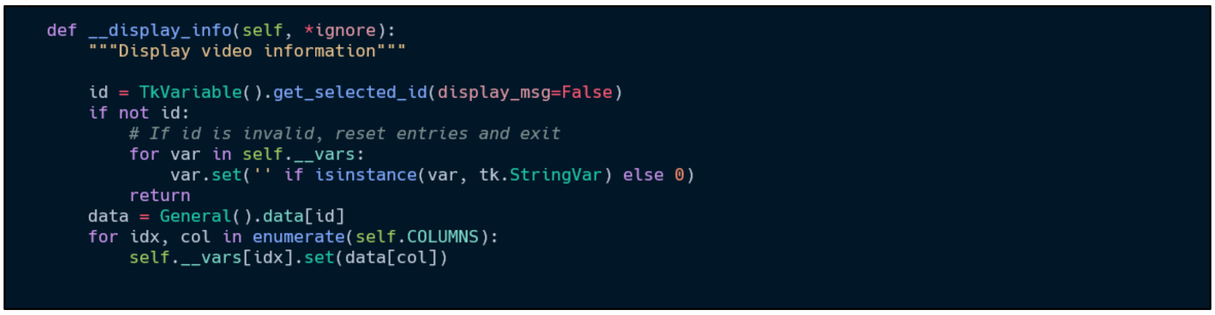
CreateVideoList



```
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 24 - CreateVideoList (create_video_list.py) internal handlers

Update Videos



```
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 25 - UpdateVideo (update_videos.py) internal hander

Stage 4: Testing and validation

Tests suit 1: UI Tests

This testsuit will covering UI testcases

- Test#1: menu UI

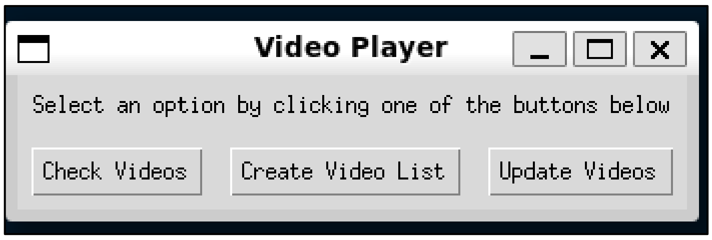


Figure 26 - UI Test#1 Menu

Result: PASSED

UI displayed as design (three buttons and label) without error or exception

- Test #2: Check Videos UI

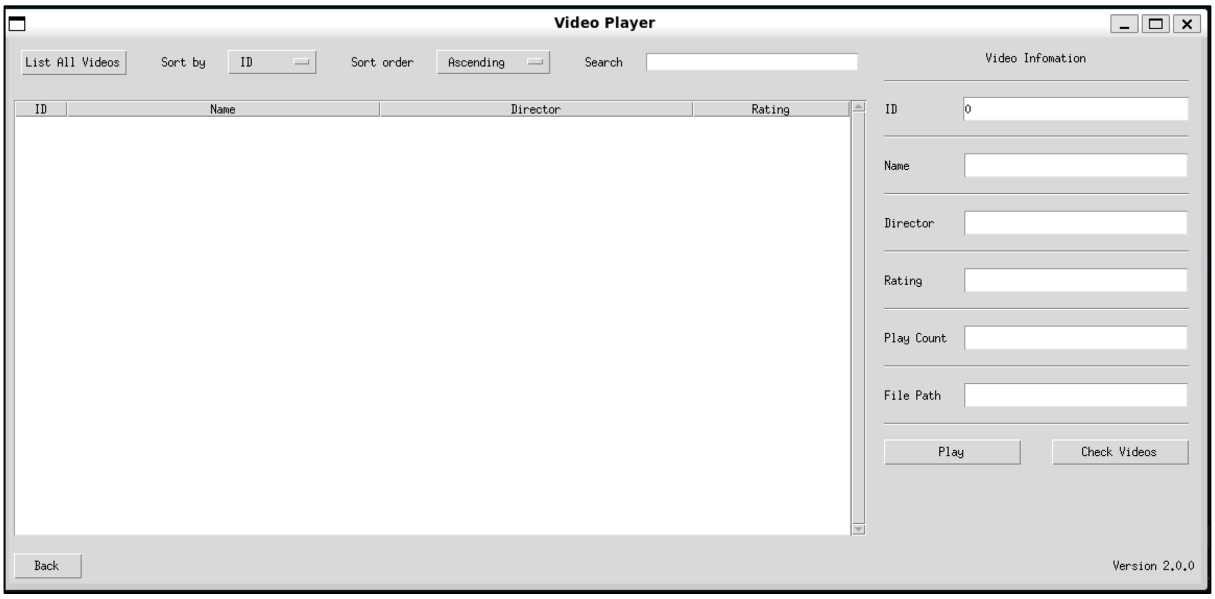


Figure 27 - UI Test#2 – CheckVideos UI

Result: PASSED

UI displayed as expected design (header, browser, panels, footer)

No error or exception raised

- Test #3: Create Videos List UI

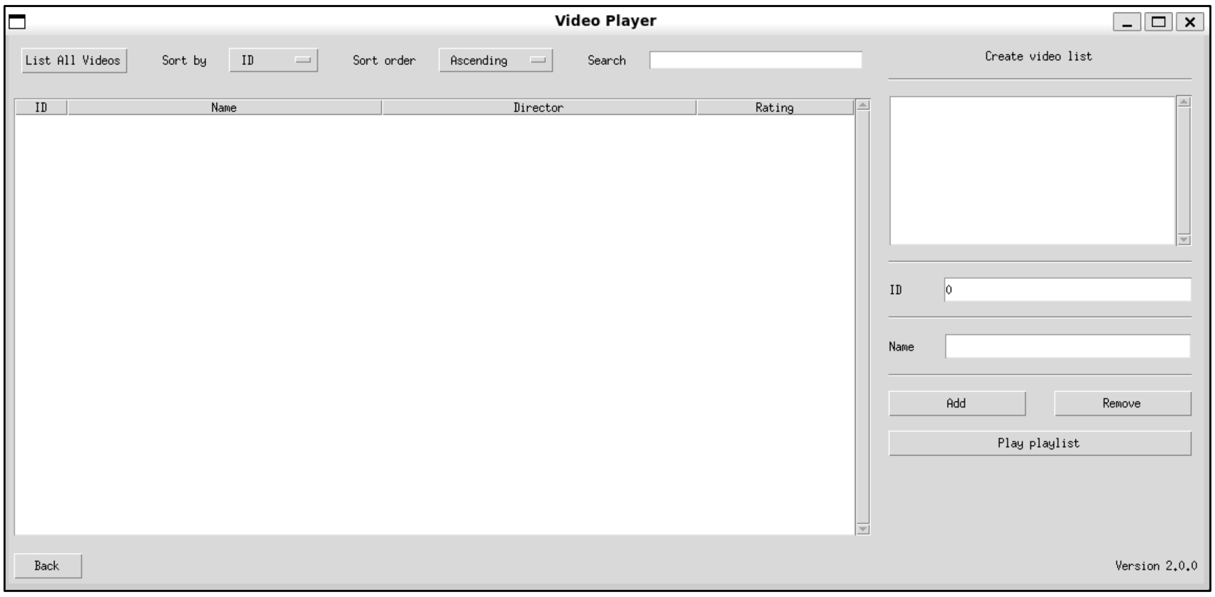


Figure 28 - UI Test#3 – Create Videos List UI

Result: PASSED

All components displayed as design

No error or exception raised

- Test #4: Update Videos UI

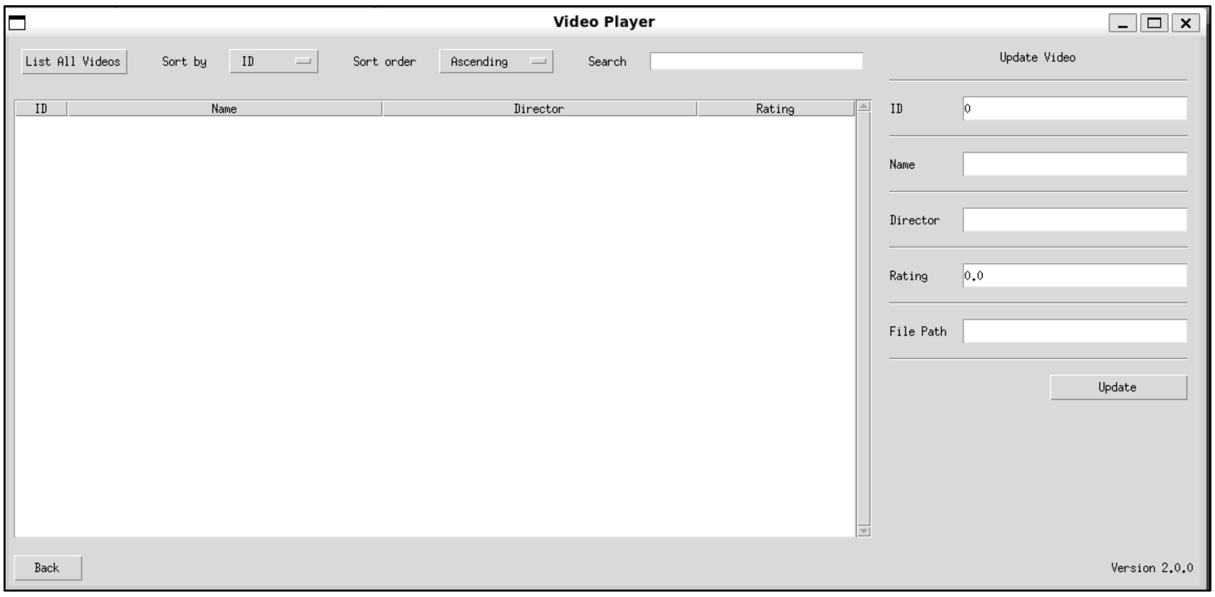


Figure 29 - UI Test#4 - Update Videos UI

Result: PASSED

All components displayed as design

No error or exception raised

- Test #5: Header

  Result: PASSED

  From 3 tests (#2, #3, #4) we can see the header component is displayed consistently as design

- Test# 6: Footer
  Result: PASSED
  From 3 tests (#2, #3, #4) we can see the footer component is displayed consistently as design

- Test# 7: browser
  Result: PASSED
  From 3 tests (#2, #3, #4) we can see the browser component is displayed consistently as design

CONCLUSION

PASSED	**7/7**
FAILED	**0/7**

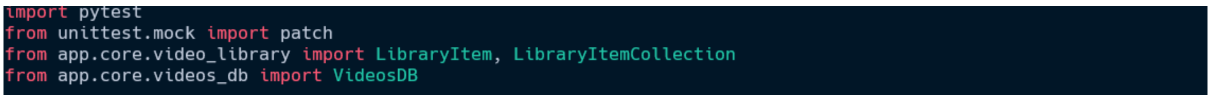
Test suit 2: CORE functions tests

This is a set of automation tests written in Python using *Pytest, unittest* modules.
The sets include **3** sub test suits:

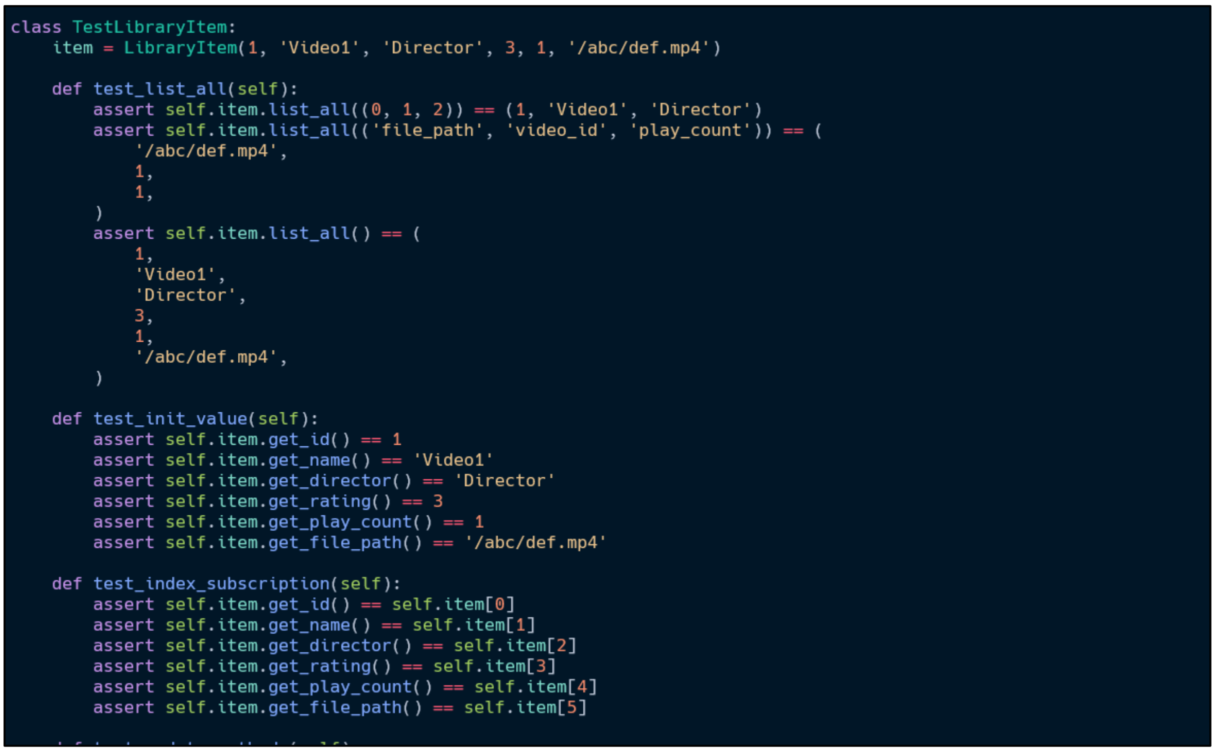
- Library Item: t11 testcases
- VideoDB: 3 testcases
- SingletonMeta: 1 testcase

All testcases is placed in *tests/* folder

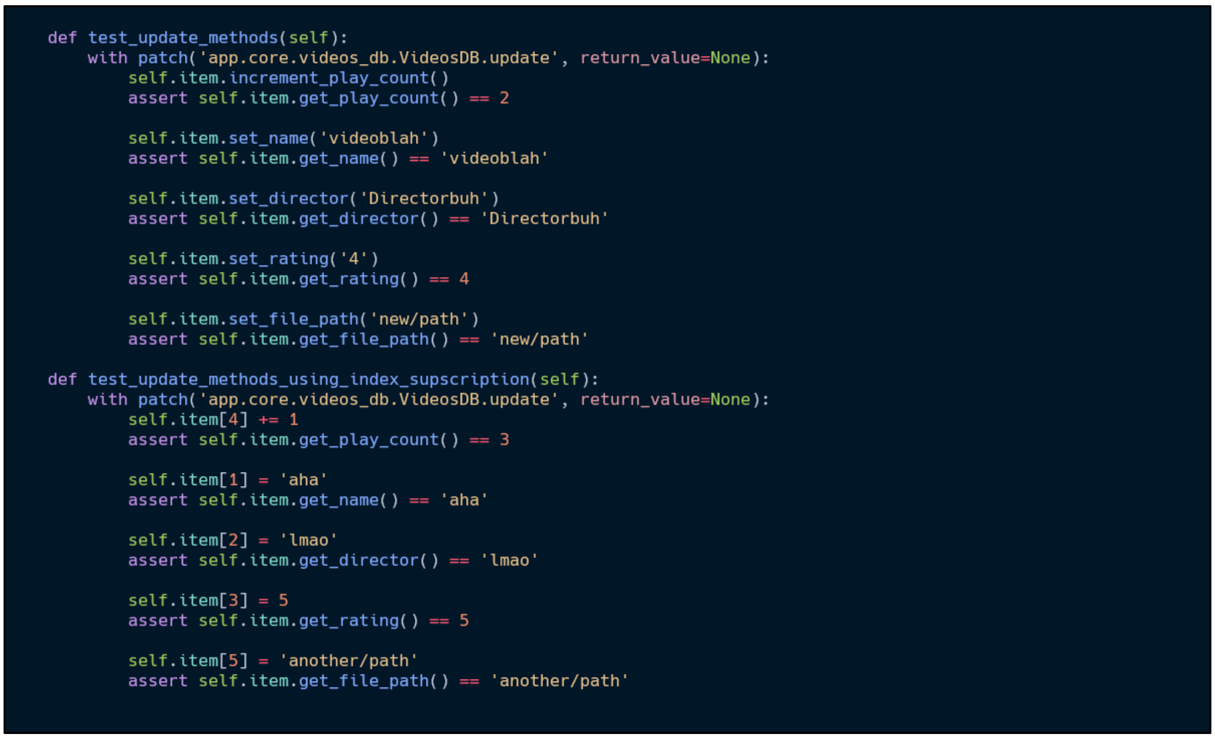
- Test#1: LibraryItem, LibraryItemCollection (test_library_item.py)
  TEST DESIGN



```python
import pytest
from unittest.mock import patch
from app.core.video_library import LibraryItem, LibraryItemCollection
from app.core.videos_db import VideosDB
```

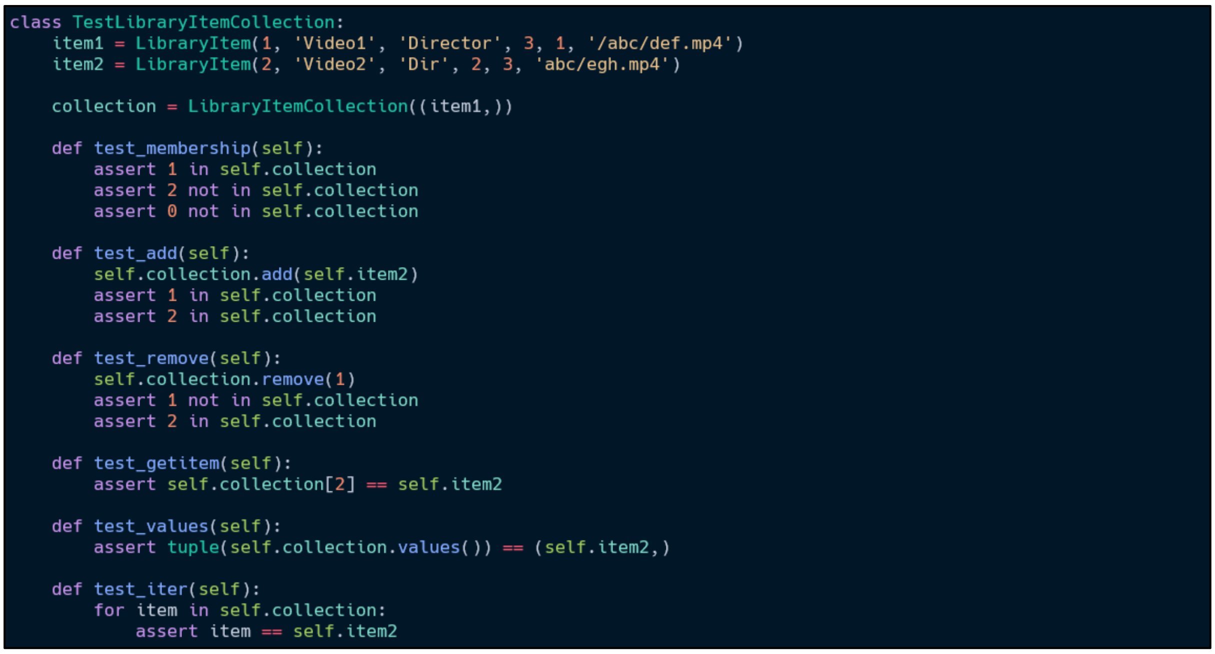


```python
class TestLibraryItem:
    item = LibraryItem(1, 'Video1', 'Director', 3, 1, '/abc/def.mp4')

    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'
```

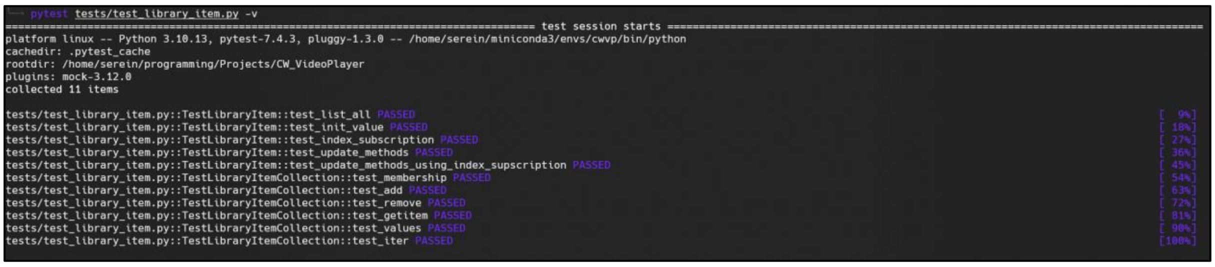
Figure 30 - LibraryItem Test Design



```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
```

Figure 31 - LibraryItemCollection Test Design

RUN TEST:

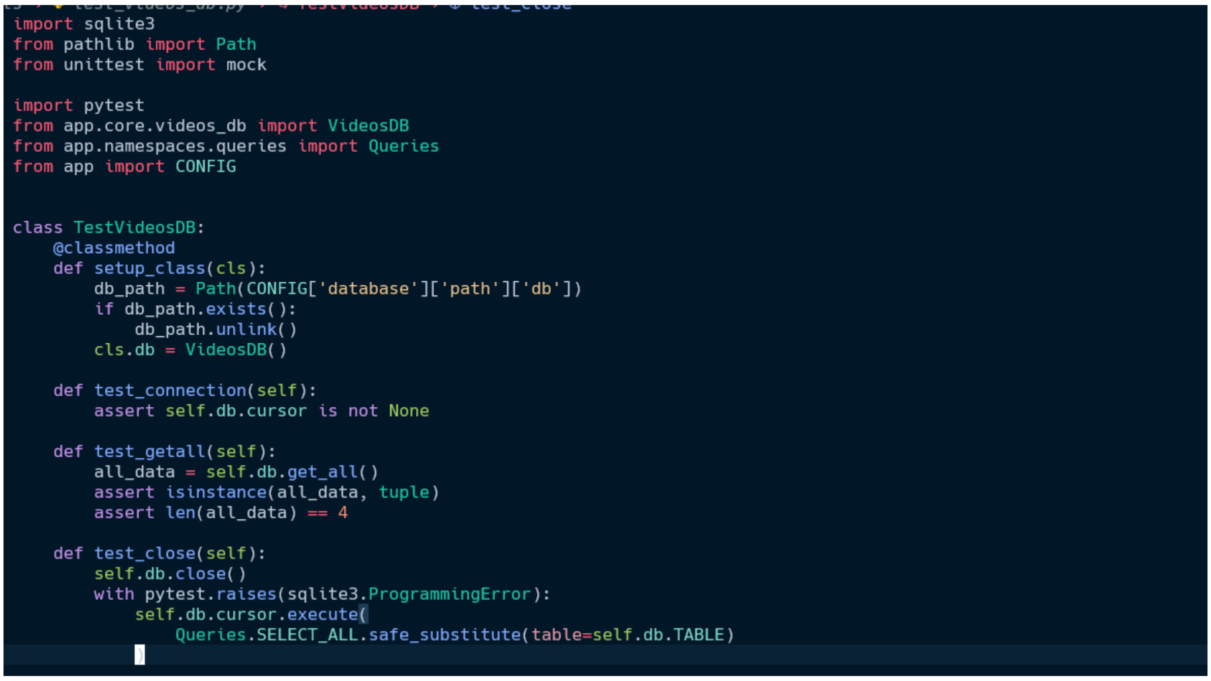


```
└── pytest tests/test_library_item.py -v
============================================================================================= test session starts ==============================================================================================
platform linux -- Python 3.10.13, pytest-7.4.3, 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%]
```

- RESULT: PASSED ALL

- TEST# 2: VideosDB

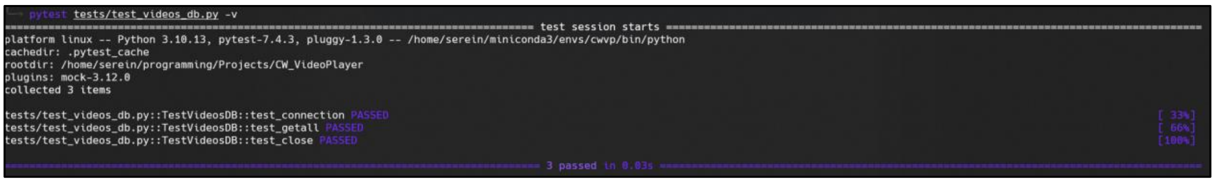
TEST DESIGN



```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)
            )
```

Figure 32 - VideosDB Test Design

RUN TESTS

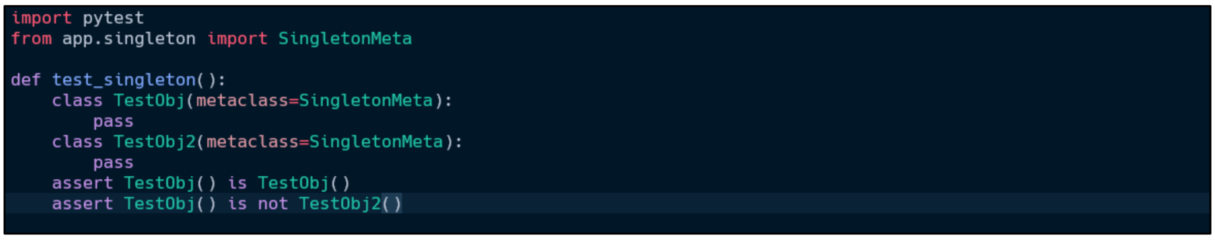


```
pytest tests/test_videos_db.py -v
=============================== test session starts ===============================
platform linux -- Python 3.10.13, pytest-7.4.3, pluggy-1.3.0 -- /home/serein/miniconda3/envs/cwvp/bin/python
cachedir: .pytest_cache
rootdir: /home/serein/programming/Projects/CW_VideoPlayer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

RESULT: PASSED ALL

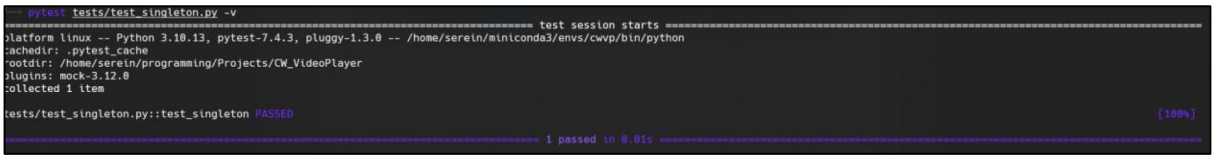
- Test#3: Singleton tests

  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()
```

RUN TEST



```
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 ===============================
```

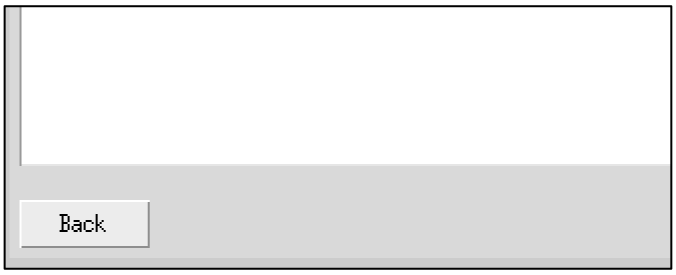
RESULT: PASSED ALL

CONCLUSION

TEST SUIT	PASSED	FAILED
LibraryItem	11/11	0/11
VideosDB	3/3	0/3
Singleton	1/1	0/1

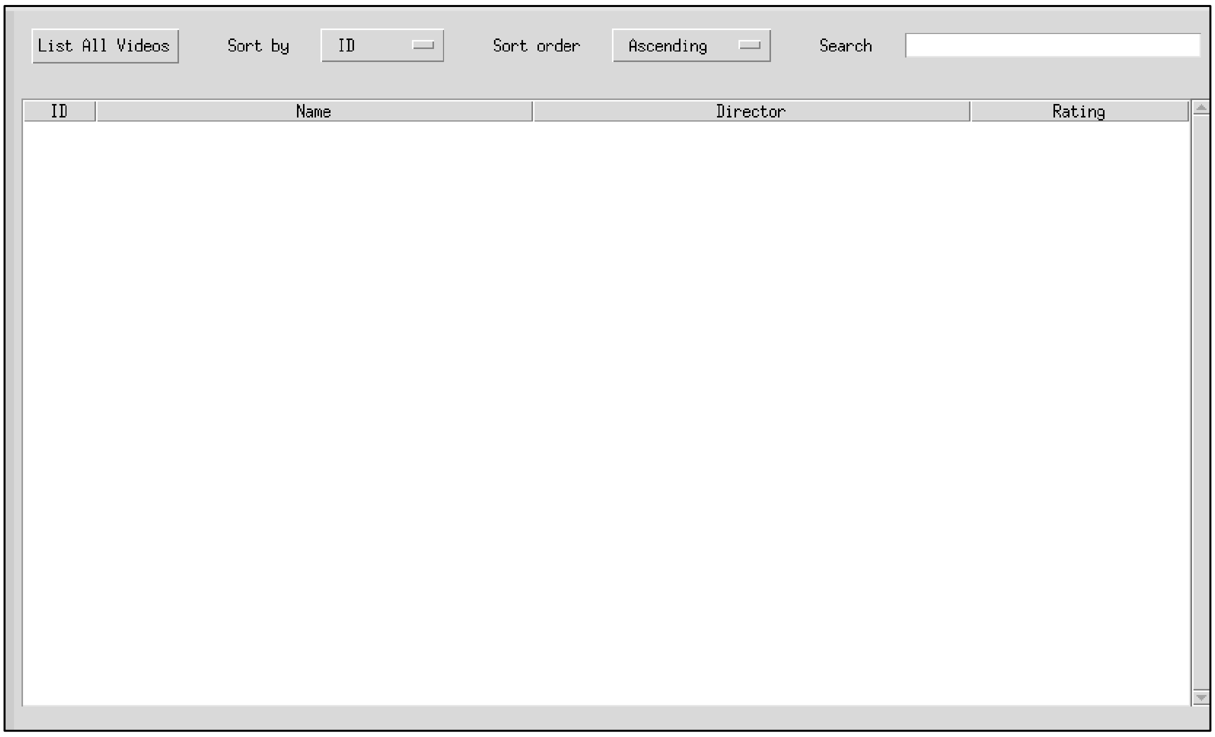
Testsuit 3: Functionalities tests

- Test#1: Menu buttons test
  - Results: PASSED – 3/3 buttons work (evidences shown in test suit 1)
- Test#2: Back button test

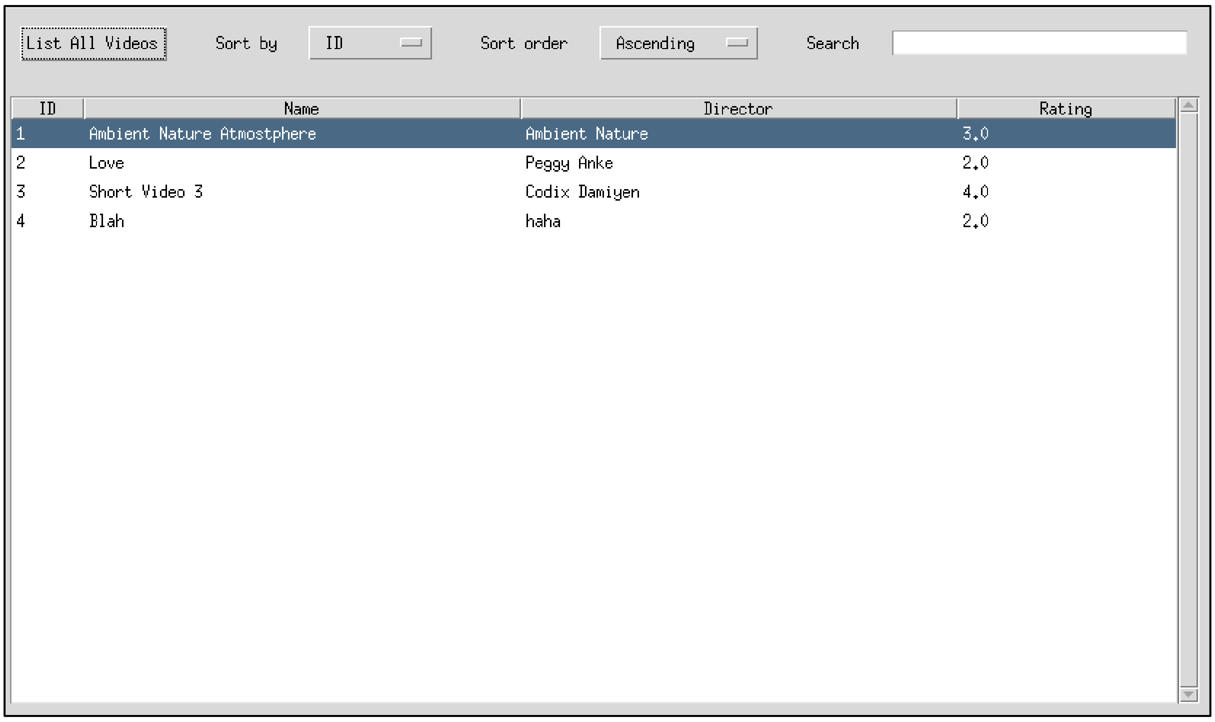


  - Result: PASSED – menu shown after clicked
- Test#3: List all videos button test

Before click



After Click



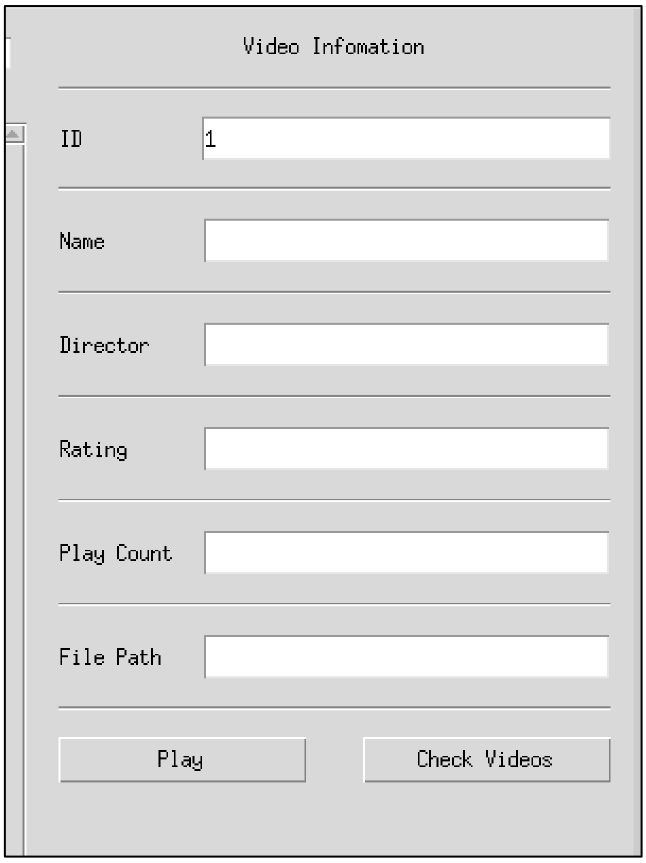
ID	Name	Director	Rating
1	Ambient Nature Atmostphere	Ambient Nature	3.0
2	Love	Peggy Anke	2.0
3	Short Video 3	Codix Damiyen	4.0
4	Blah	haha	2.0

- Result: PASSED

  The data shown match the data initialized in database

  No error or exception raised

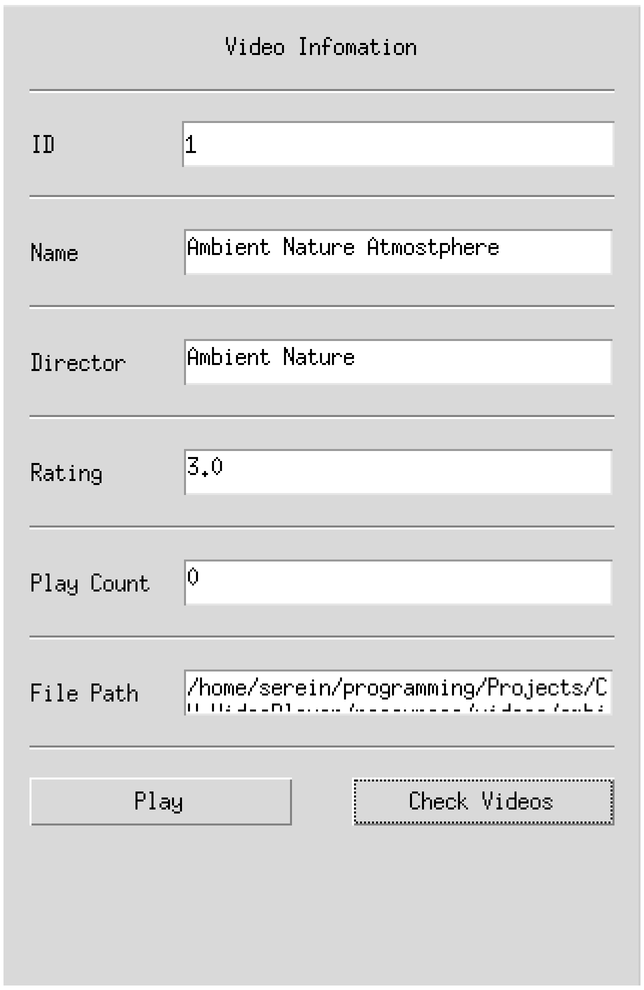
- Test# 4: Check videos button test – valid input (id = 1)
  Before click



Video Infomation

ID	1
Name	
Director	
Rating	
Play Count	
File Path	

After clicked

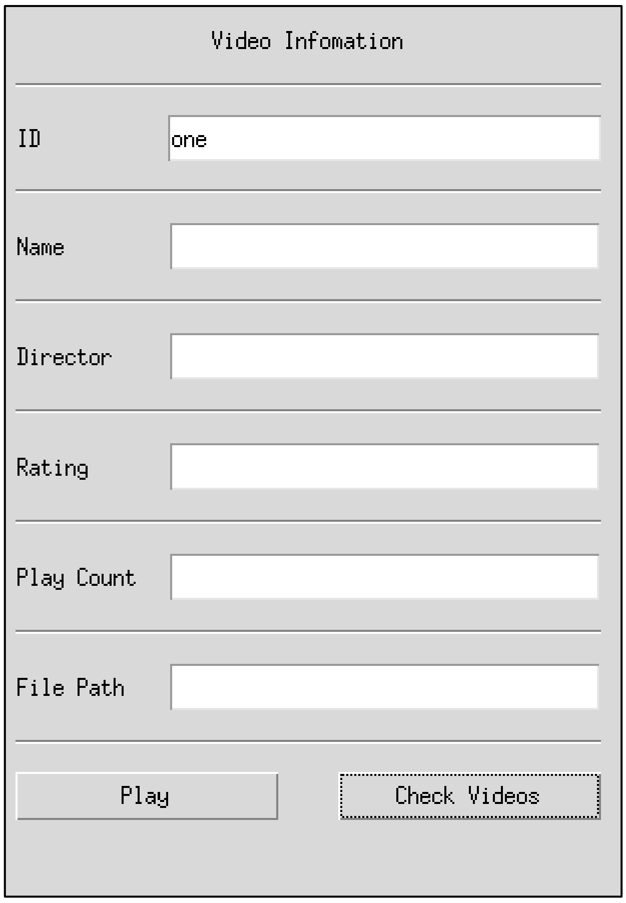


Video Infomation

ID	1
Name	Ambient Nature Atmostphere
Director	Ambient Nature
Rating	3.0
Play Count	0
File Path	/home/serein/programming/Projects/C

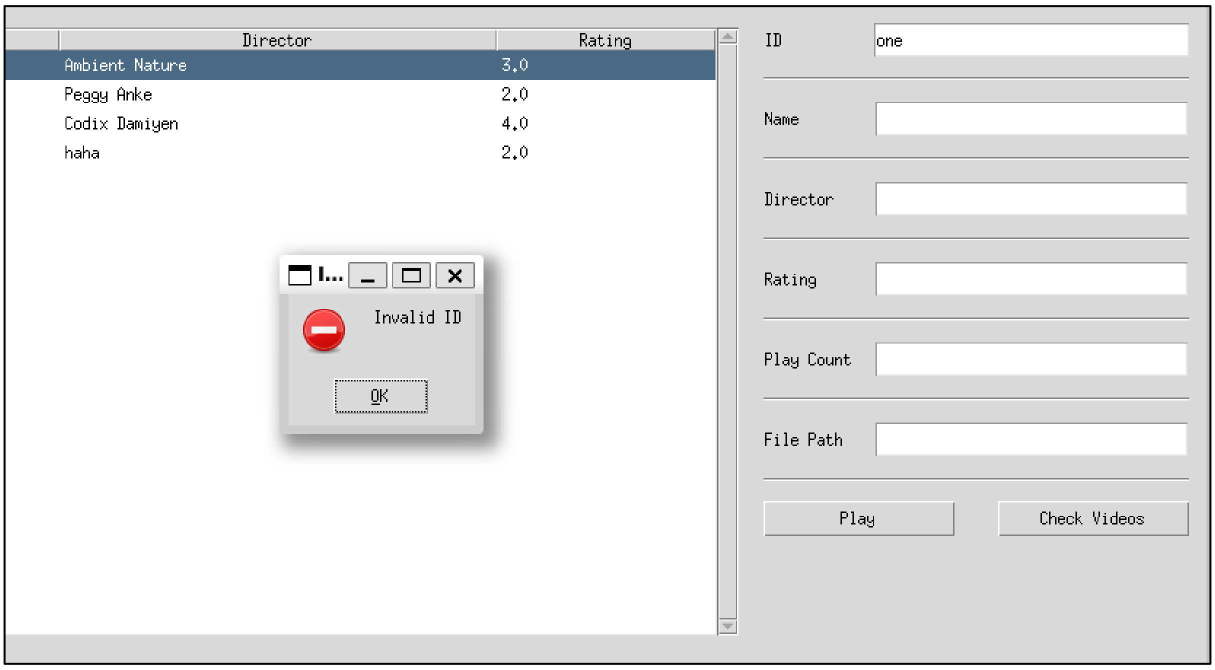
- Result: PASSED
  Displayed information match with database data
  No error or exception raised

- Test# 5: Check videos button – invalid input (id = 'one')
  Before click


Video Infomation

ID	one
Name	
Director	
Rating	
Play Count	
File Path	

After click



	Director	Rating
	Ambient Nature	3.0
	Peggy Anke	2.0
	Codix Damiyen	4.0
	haha	2.0

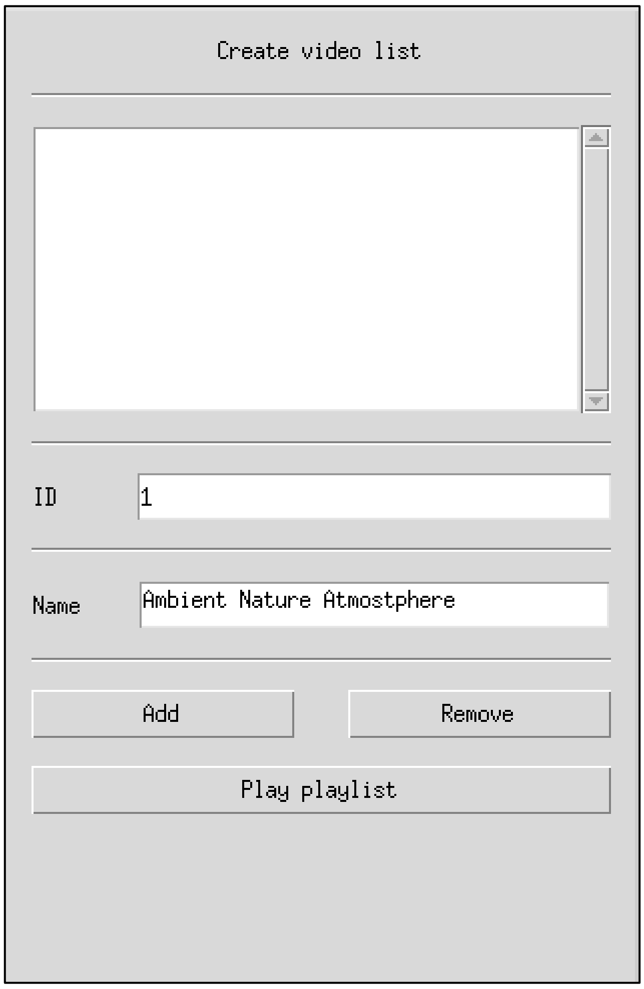
Invalid ID

ID	one
Name	
Director	
Rating	
Play Count	
File Path	

- Result: PASSED
  Invalid message box shown
  No information displayed
  No error or exception raised

- Test# 6: Check videos button – valid input (id = 1)

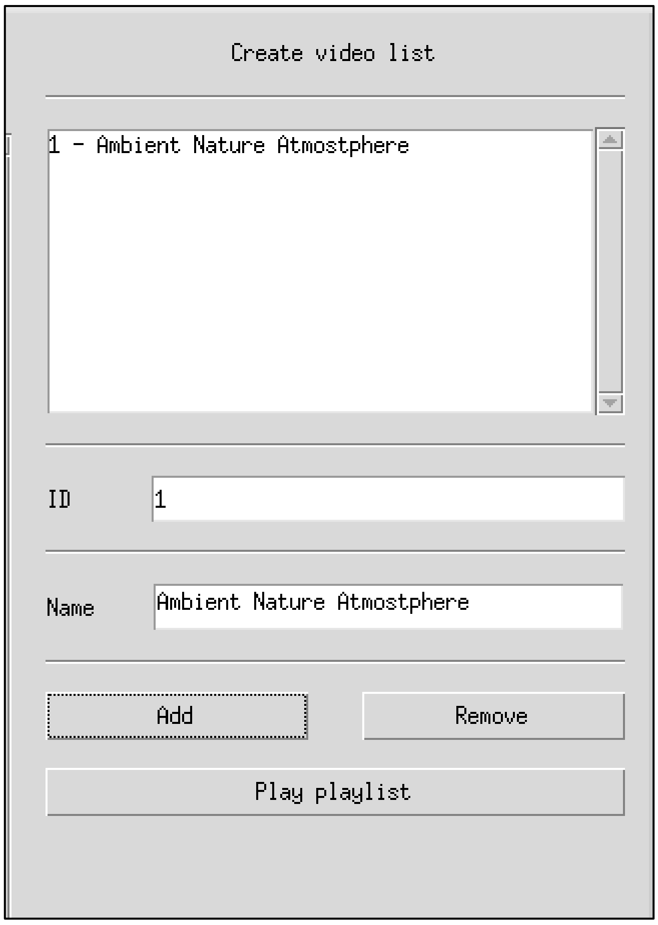
Before click


Create video list

ID 1

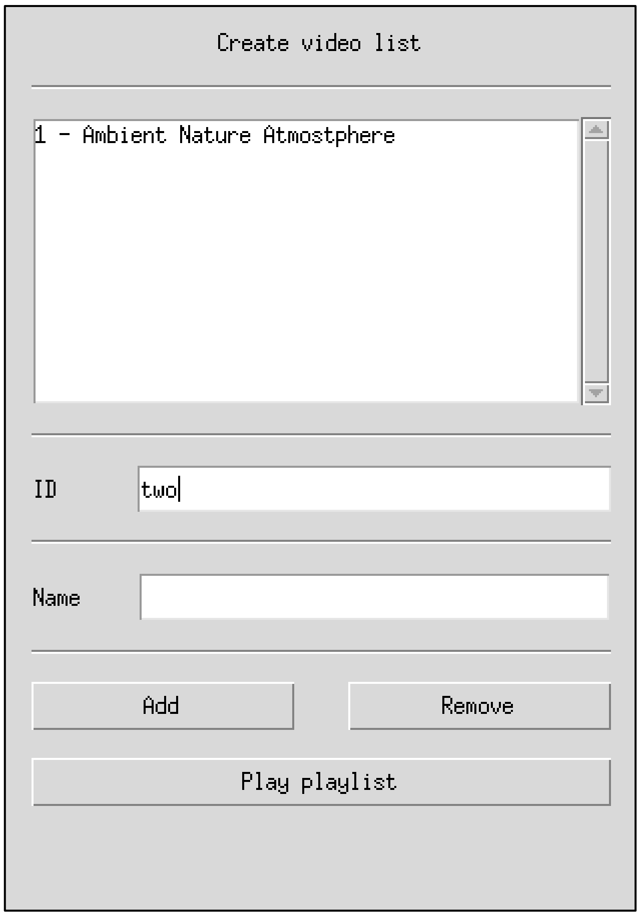
Name Ambient Nature Atmostphere

After click



- Result: PASSED
  The video name automatically displayed
  The video with id = 1 added to the playlist box
  No error or exception raised

- Test#7: Add video button test – invalid input (id = 'two')
  Before click

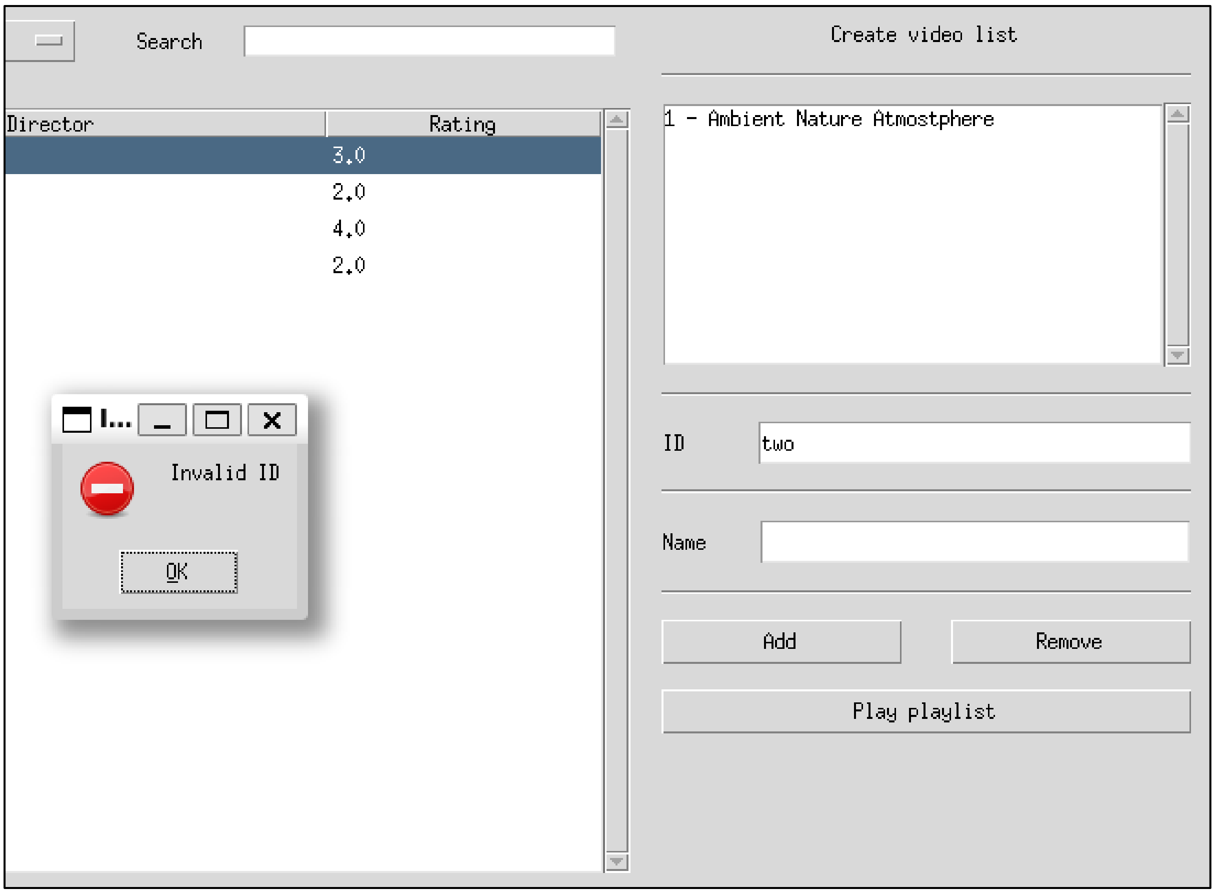

Create video list

1 - Ambient Nature Atmostphere

ID two

Name

After click



Search

Create video list

Director	Rating
	3.0
	2.0
	4.0
	2.0

1 - Ambient Nature Atmostphere

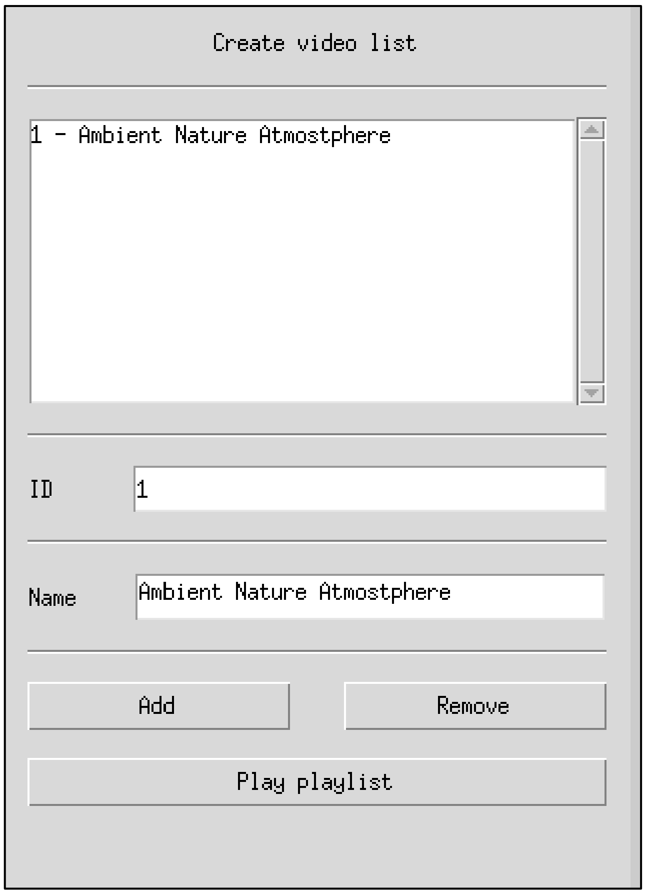
ID two

Name

Invalid ID

- Result: PASSED
  The name is not displayed as the id is invalid
  An messagebox with the informative content is shown
  No error or exception raised

- Test#8: Add button test – added video (id = 1)
  Before click

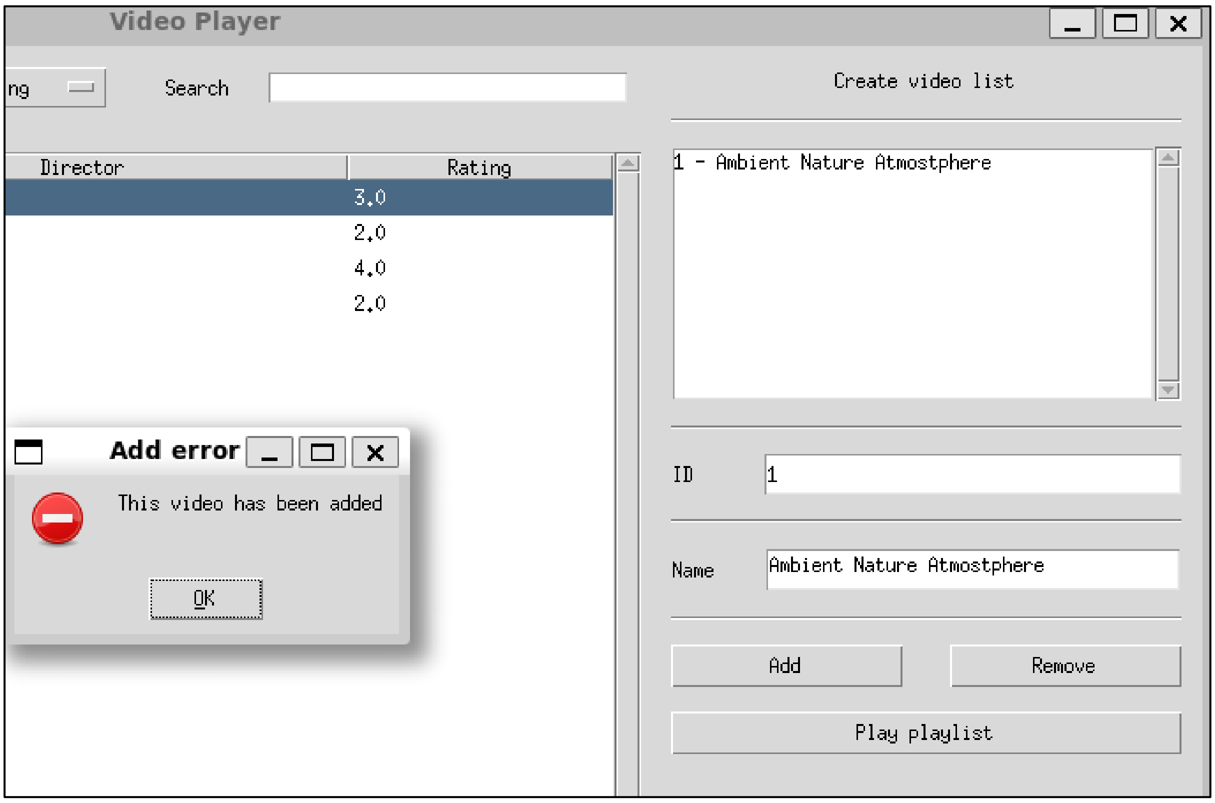

Create video list

1 - Ambient Nature Atmostphere

ID 1

Name Ambient Nature Atmostphere

After click


Video Player

Search

Director	Rating
	3.0
	2.0
	4.0
	2.0

Add error

This video has been added

Create video list

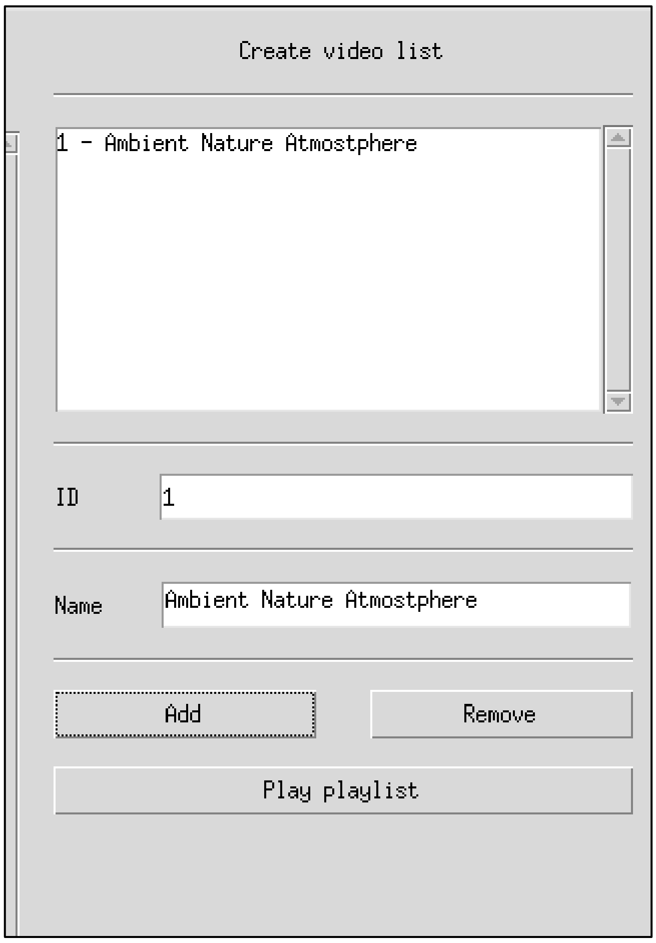
1 - Ambient Nature Atmostphere

ID 1

Name Ambient Nature Atmostphere

- Test#9: Remove button test – valid id, has been added to playlist (id = 1)

Before click

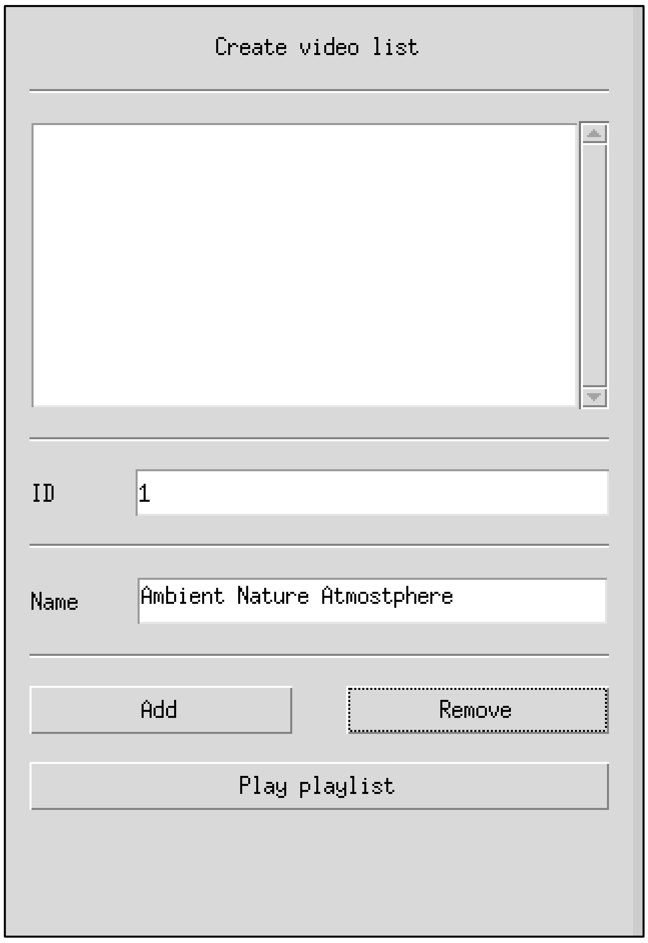


1 - Ambient Nature Atmostphere

1

Ambient Nature Atmostphere

After click

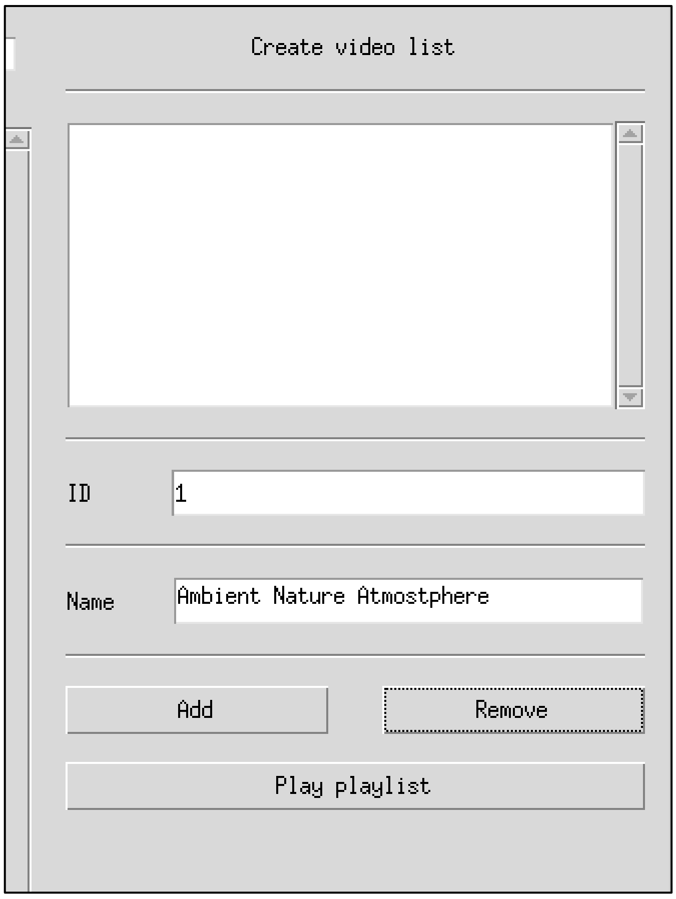

Create video list

ID 1

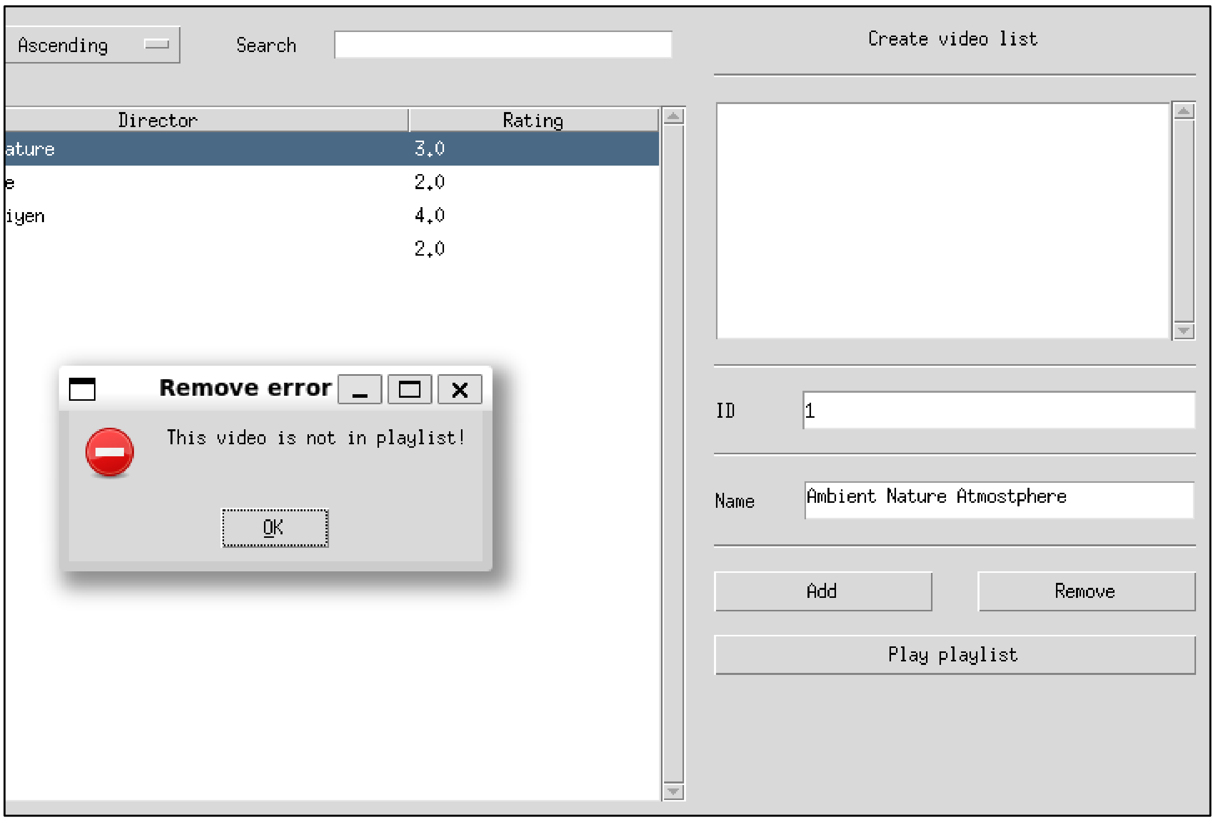
Name Ambient Nature Atmostphere

- Result: PASSED
  The video has been removed from playlist
  No error or exception raised

- Test#10: Remove button – vailid id but not added to playlist (id=1)
  Before click



After click

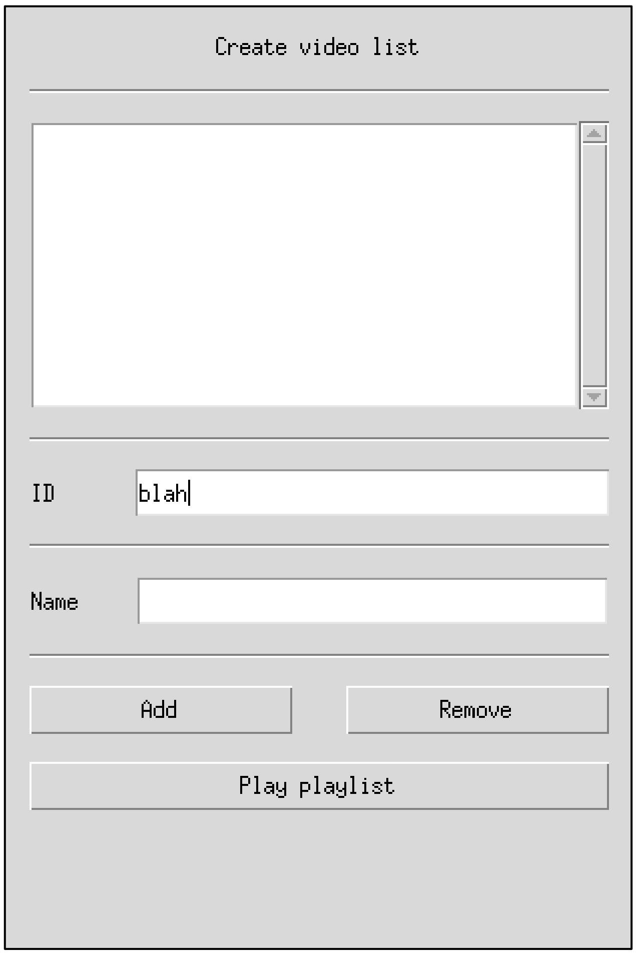


This video is not in playlist!

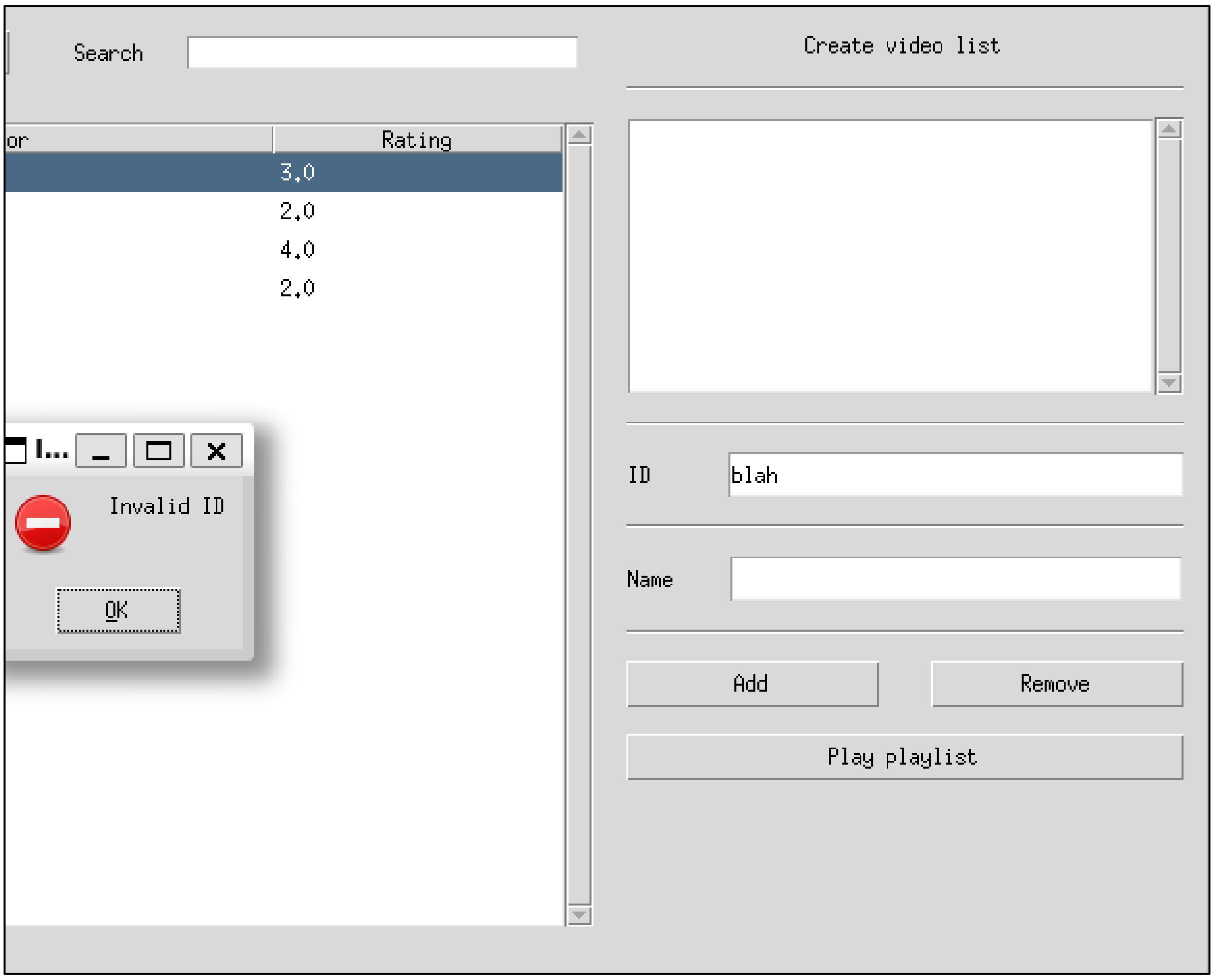
- Result: PASSED

A message box with informative contents displayed
No error or exception raised

- Test#11: invalid id input (id = 'blah')
  Before click



After click



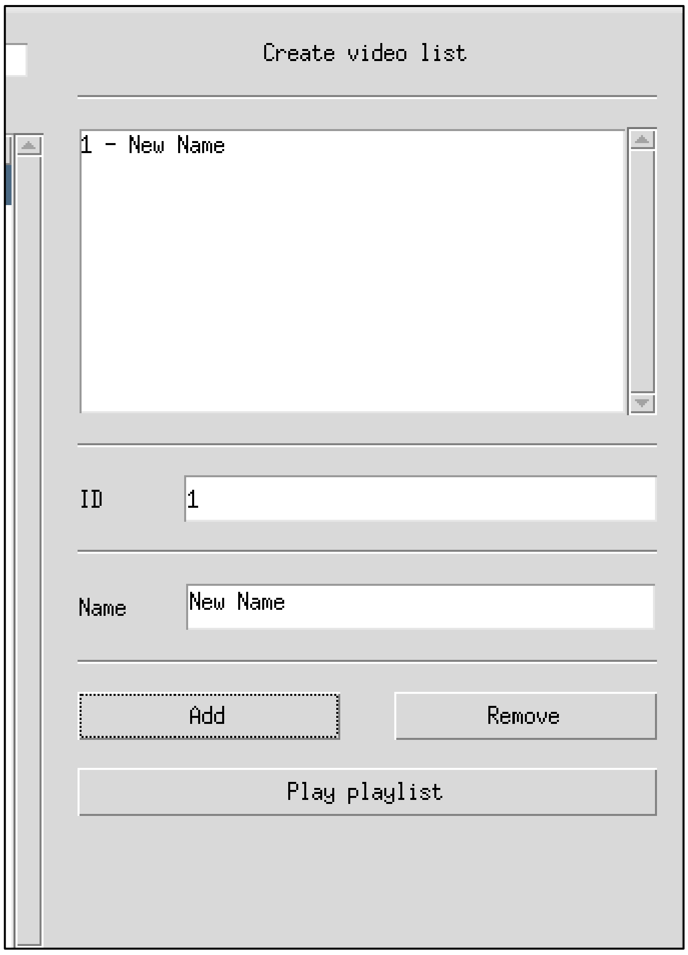
- Result: PASSED

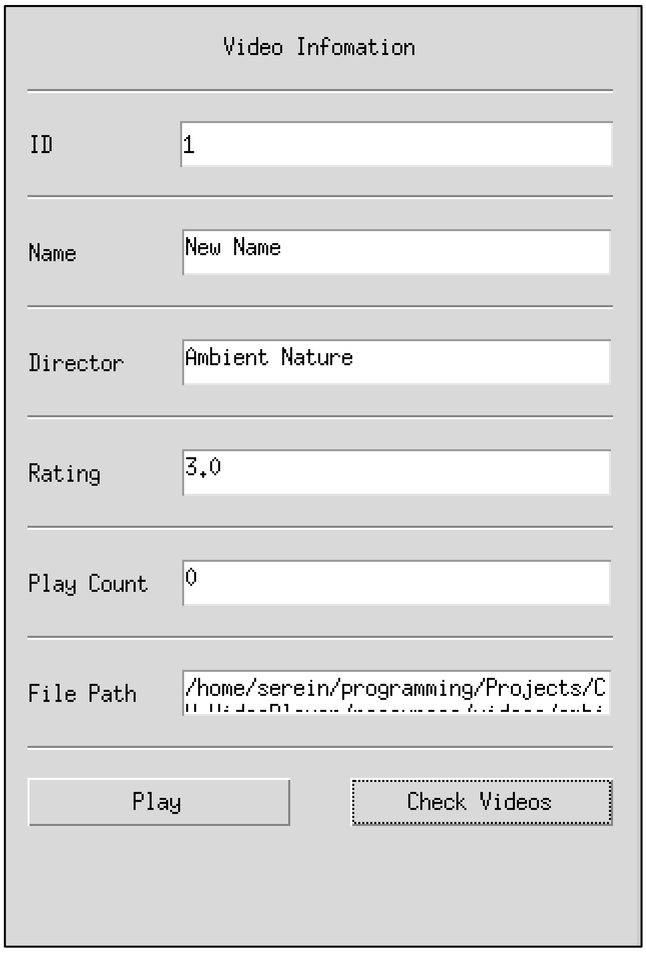
  A message box with informative contents displayed

  No error or exception raised

- Test#14: Play playlist button test – increase the play count (id = 1)

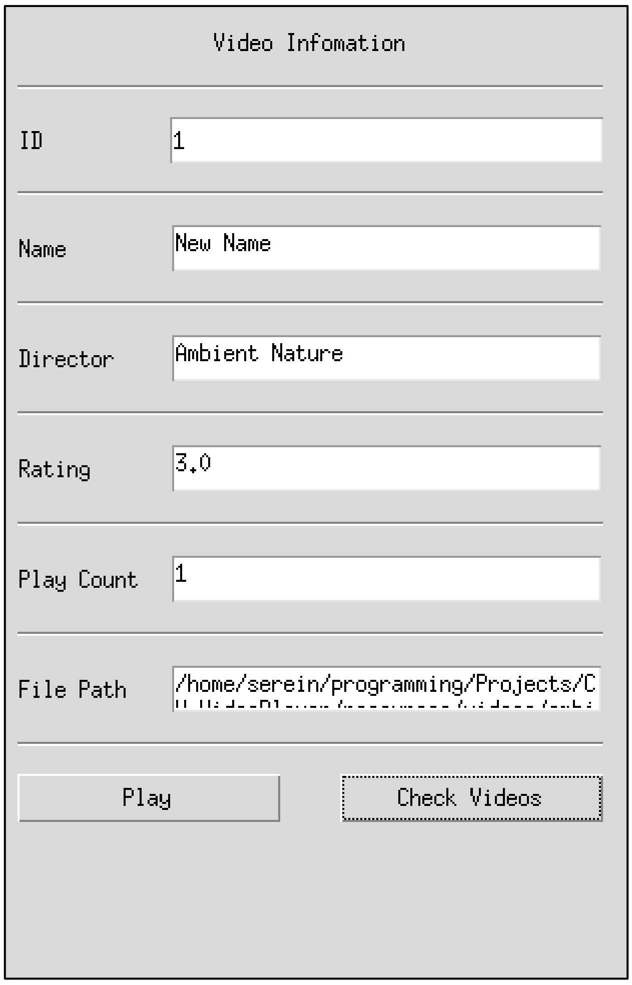
  Before click


Create video list
1 - New Name
ID
1
Name
New Name
Add
Remove
Play playlist


Video Infomation

ID	1
Name	New Name
Director	Ambient Nature
Rating	3.0
Play Count	0
File Path	/home/serein/programming/Projects/C

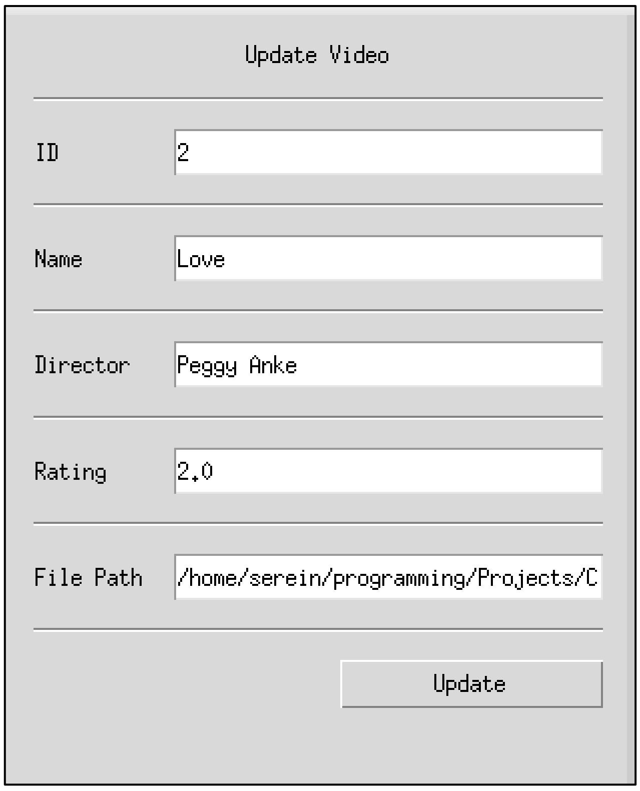
After click (playcount 0 -> 1)


Video Infomation

ID	1
Name	New Name
Director	Ambient Nature
Rating	3.0
Play Count	1
File Path	/home/serein/programming/Projects/C

- Result: PASSED
  The play count increased by one
  No error or exception raised

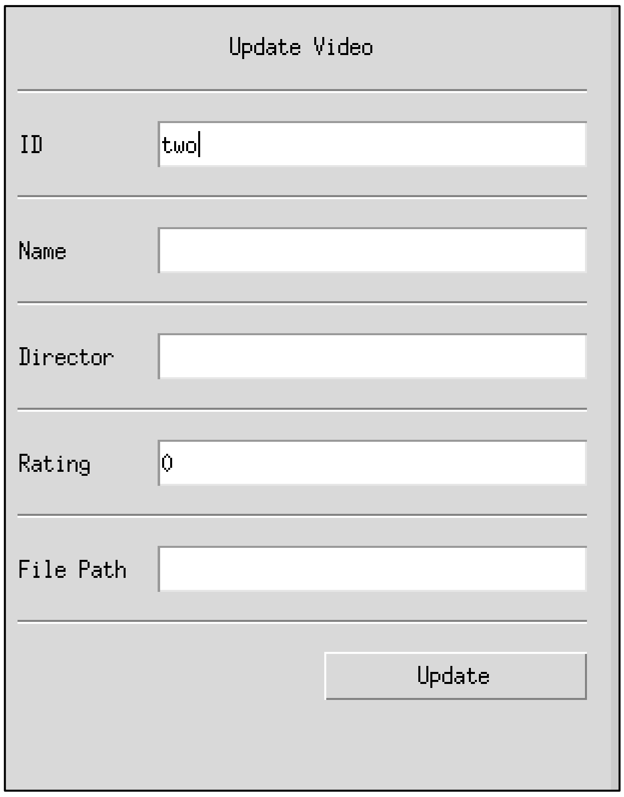
- Test#13: Update video auto display information – valid id input (input = 2)


Update Video

ID	2
Name	Love
Director	Peggy Anke
Rating	2.0
File Path	/home/serein/programming/Projects/C

- Result: PASSED
  Information displayed in right textbox and match with the database data

- Test#14: Update video auto display information – invalid id input (input = 'two')


Update Video

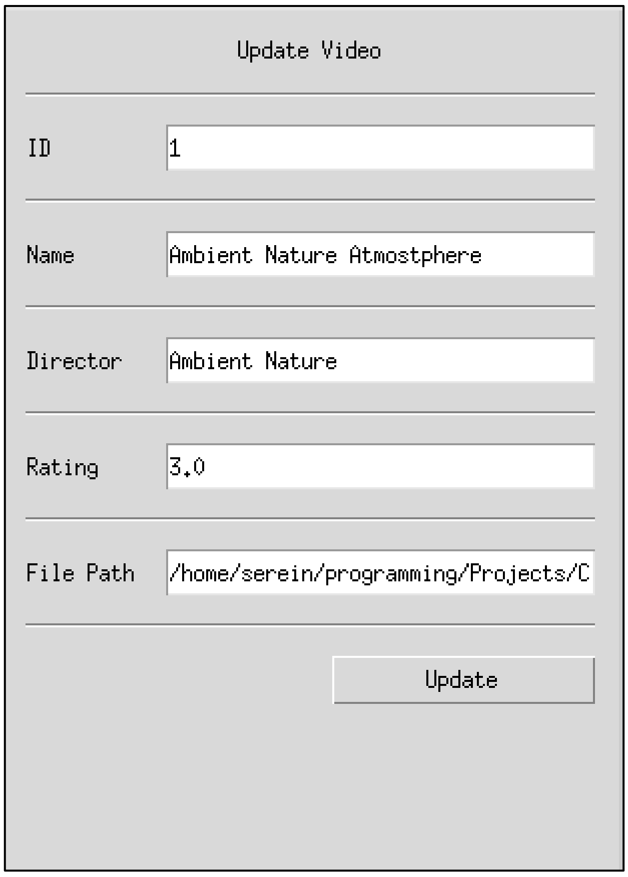
ID	two
Name	
Director	
Rating	0
File Path	

- Result: PASSED
  No information displayed (note: the zero at rating field is not the data in database, it is the default value of tk.IntVar object when reset)

No error or exception raised

- Test#15: Update video (id = 1, name='new name')

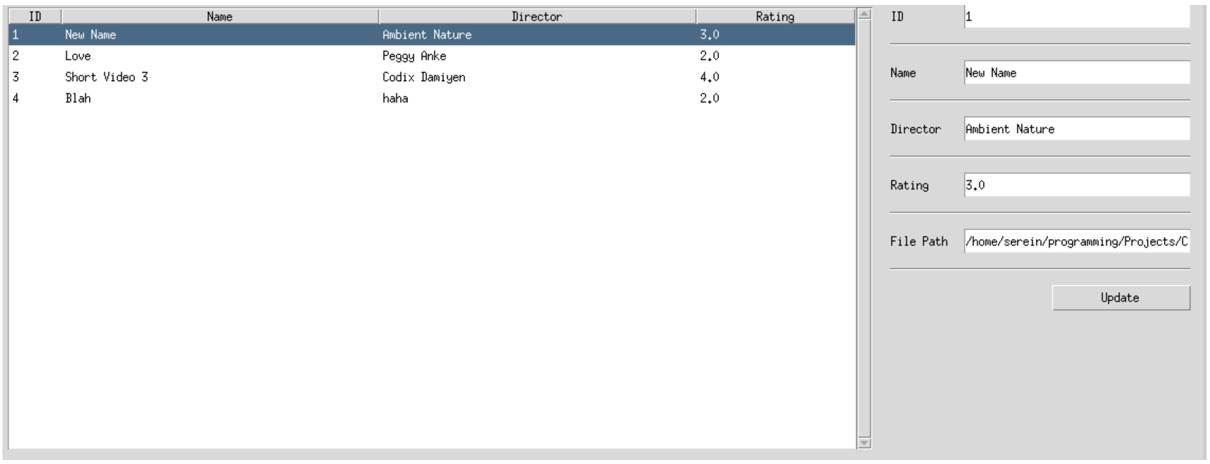
  Before update



Update Video

ID	1
Name	Ambient Nature Atmostphere
Director	Ambient Nature
Rating	3.0
File Path	/home/serein/programming/Projects/C

  After update



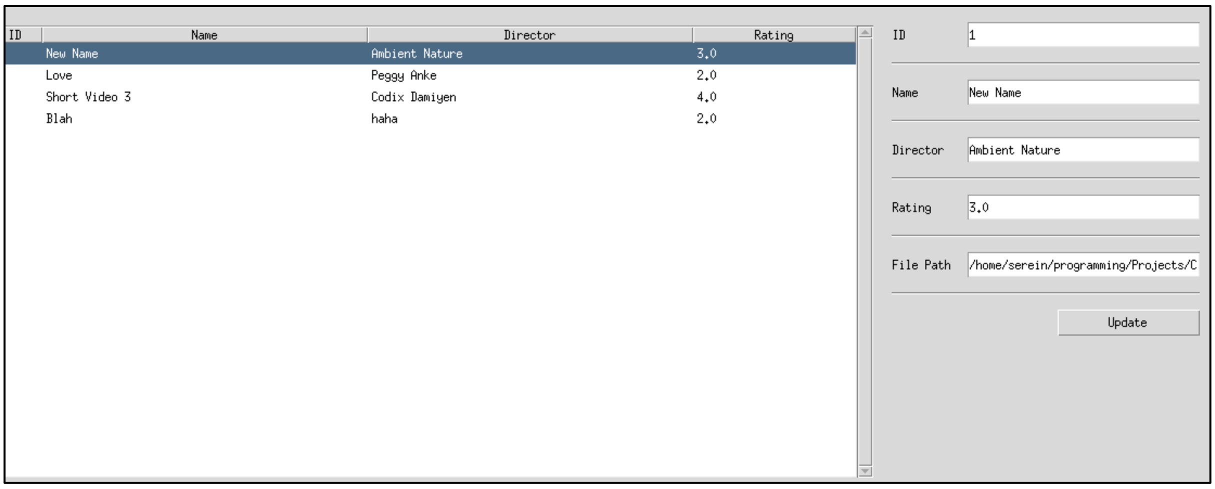
ID	Name	Director	Rating
1	New Name	Ambient Nature	3.0
2	Love	Peggy Anke	2.0
3	Short Video 3	Codix Damiyen	4.0
4	Blah	haha	2.0

ID	1
Name	New Name
Director	Ambient Nature
Rating	3.0
File Path	/home/serein/programming/Projects/C

  - Result: PASSED
    The value of the field is updated
    No error or exception raised

- Test#16: Update video – invalid new data (id = 1, Rating='a')

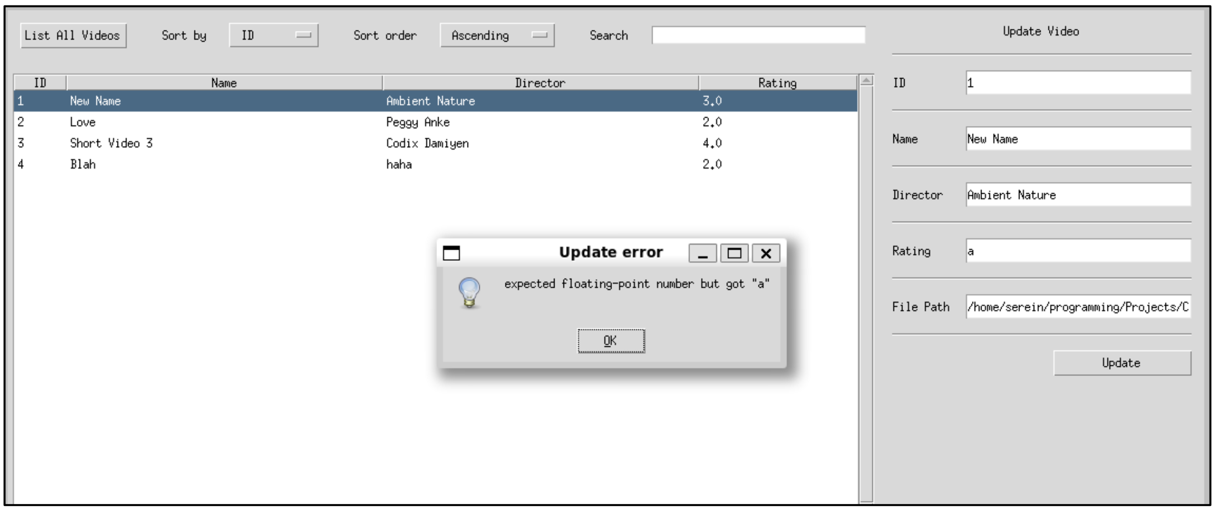
  Before update



ID	Name	Director	Rating
	New Name	Ambient Nature	3.0
	Love	Peggy Anke	2.0
	Short Video 3	Codix Damiyen	4.0
	Blah	haha	2.0

ID: 1
Name: New Name
Director: Ambient Nature
Rating: 3.0
File Path: /home/serein/programming/Projects/C

After update



ID	Name	Director	Rating
1	New Name	Ambient Nature	3.0
2	Love	Peggy Anke	2.0
3	Short Video 3	Codix Damiyen	4.0
4	Blah	haha	2.0

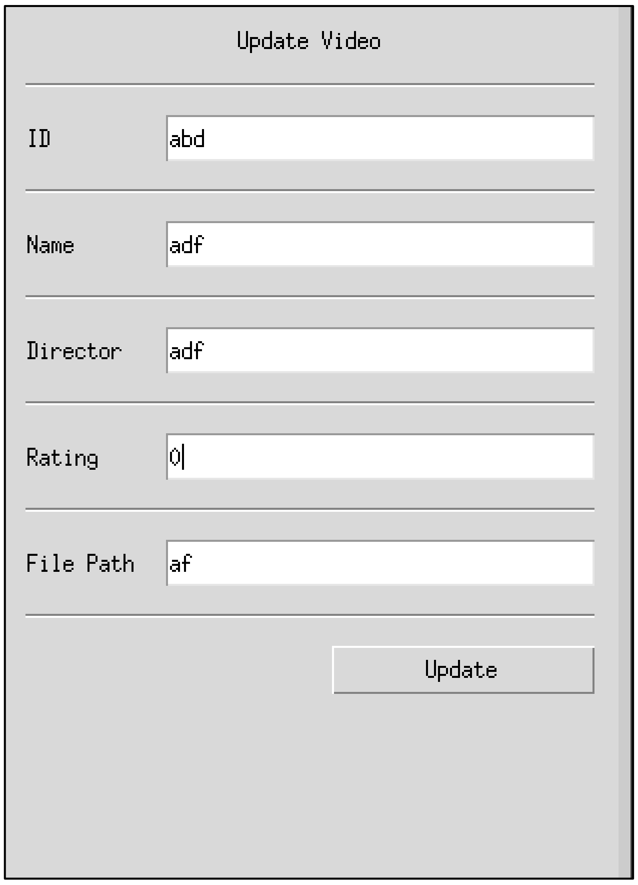
Update error

expected floating-point number but got "a"

ID: 1
Name: New Name
Director: Ambient Nature
Rating: a
File Path: /home/serein/programming/Projects/C

- Result: PASSED
  A message box with informative content is displayed
  No information is updated
  No error or exception raised

- Test#17: Update invalid id (id = 'abd')
  Before update



Update Video

ID abd

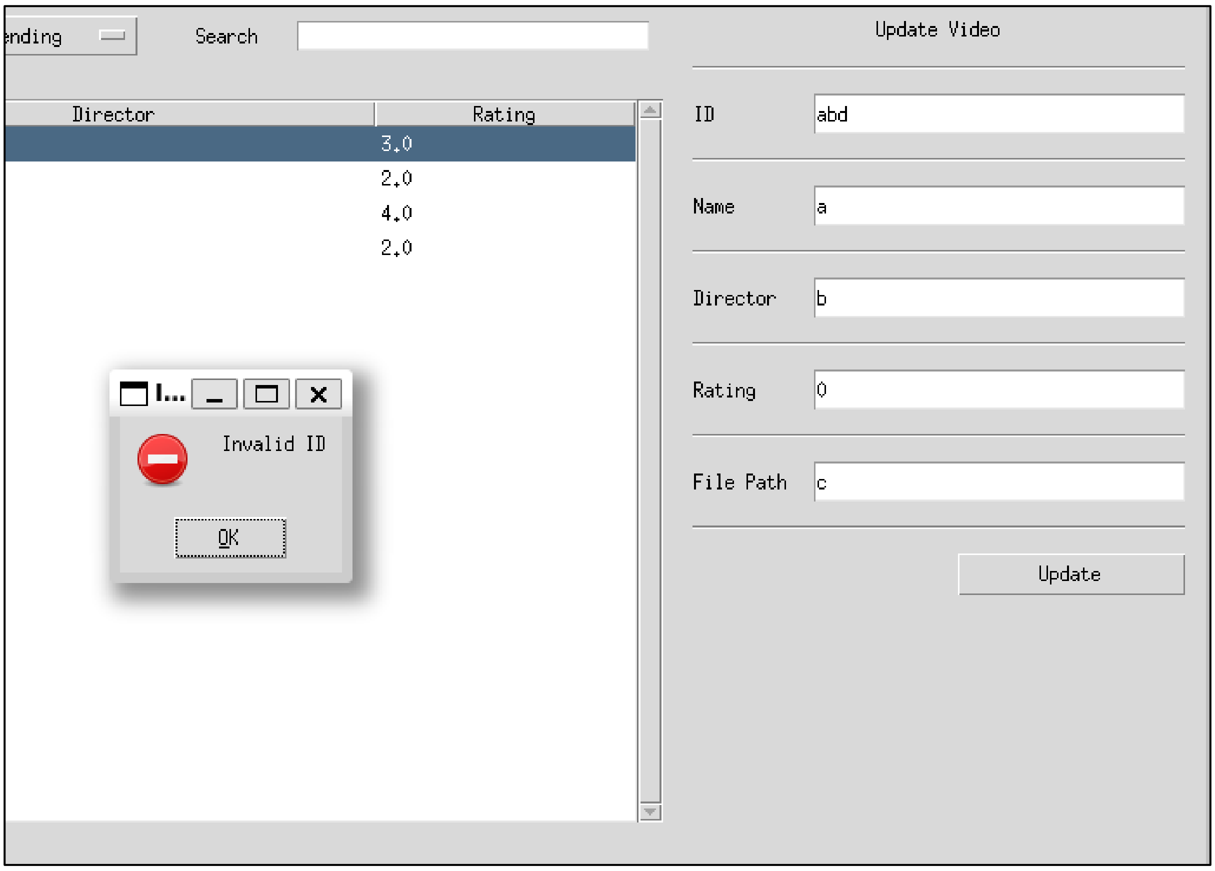
Name adf

Director adf

Rating 0

File Path af

After update



Director	Rating
	3.0
	2.0
	4.0
	2.0

Invalid ID

Update Video

ID abd

Name a

Director b

Rating 0

File Path c

- Result: PASSED

A message box with informative content is displayed
No information is updated
No error or exception raised

CONCLUSION

Testsuit	PASSED	FAILED
UI	7/7	0/7
Core functions (automation)	15/15	0/15
Functionalities	17/17	0/17
Total	39/39	0/39

Every testcases is PASSED, the application is well designed and run smoothly, functionalities are bind exactly to buttons, informative message is displayed if necessary.

Stage 5: Innovation

Search engine

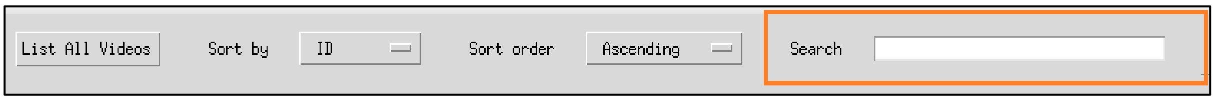
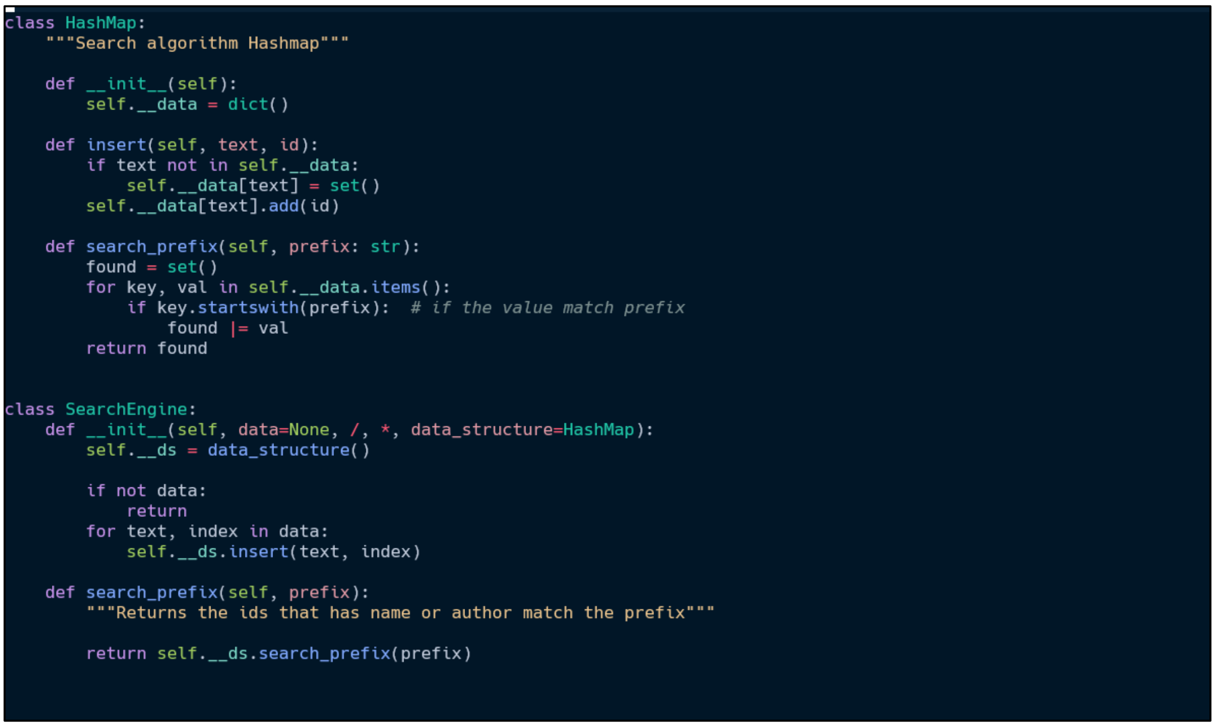


Figure 33 - Search entry placed at headbar

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

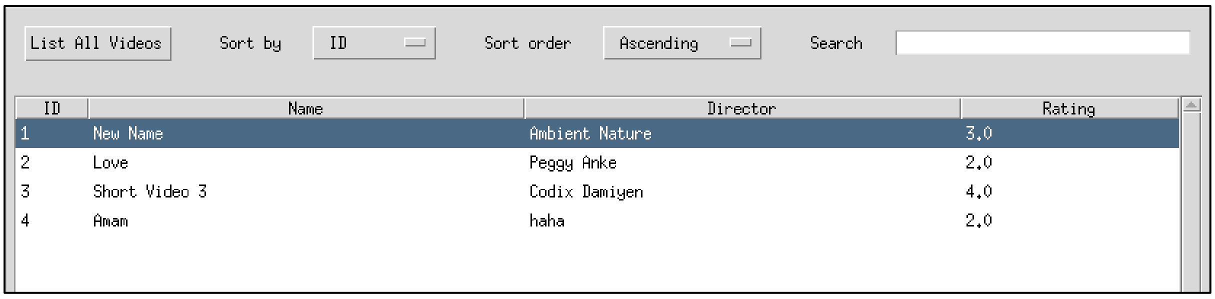


```
class HashMap:
    """Search algorithm Hashmap"""

    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)
```

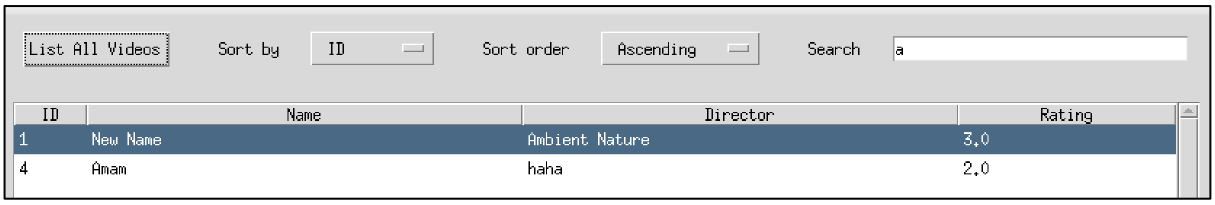
Figure 34 - SearchEngine (search_engine.py) implementation

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

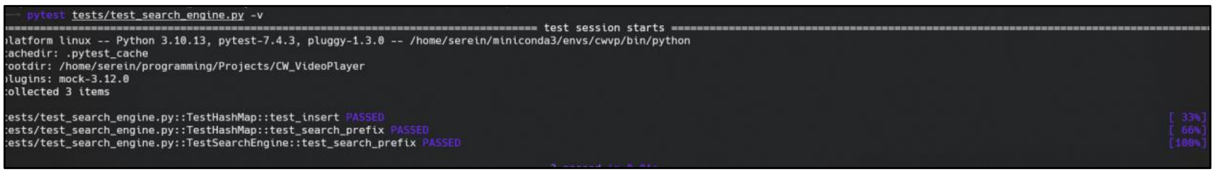
- Test:



ID	Name	Director	Rating
1	New Name	Ambient Nature	3.0
2	Love	Peggy Anke	2.0
3	Short Video 3	Codix Damiyen	4.0
4	Amam	haha	2.0



ID	Name	Director	Rating
1	New Name	Ambient Nature	3.0
4	Amam	haha	2.0

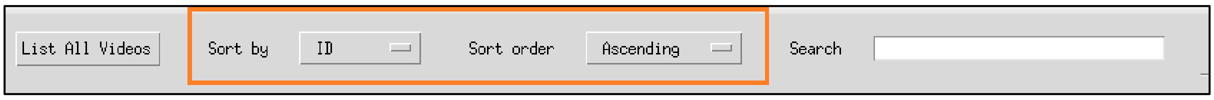


```
pytest tests/test_search_engine.py -v
============================================ test session starts ============================================
latform linux -- Python 3.10.13, pytest-7.4.3, pluggy-1.3.0 -- /home/serein/miniconda3/envs/cwvp/bin/python
achedir: .pytest_cache
ootdir: /home/serein/programming/Projects/CW_VideoPlayer
lugins: mock-3.12.0
ollected 3 items

ests/test_search_engine.py::TestHashMap::test_insert PASSED                                          [ 33%]
ests/test_search_engine.py::TestHashMap::test_search_prefix PASSED                                   [ 66%]
ests/test_search_engine.py::TestSearchEngine::test_search_prefix PASSED                              [100%]
```

Sort order

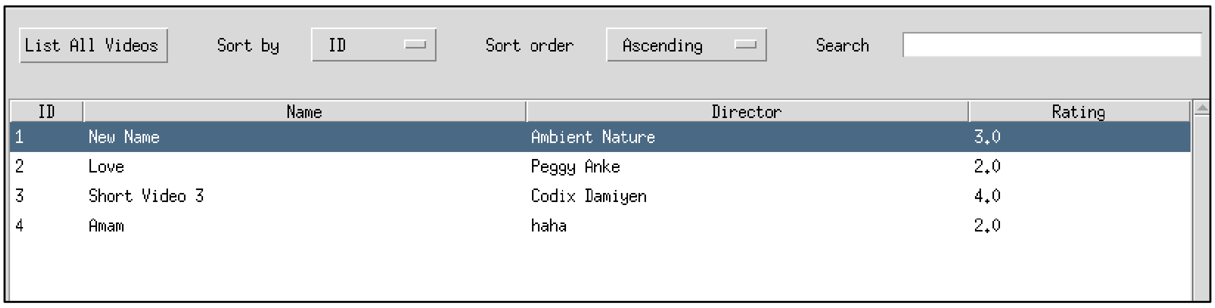
Sort by – field to sort (default: ID)

Sort order – Sort direction ascending (default) or descending

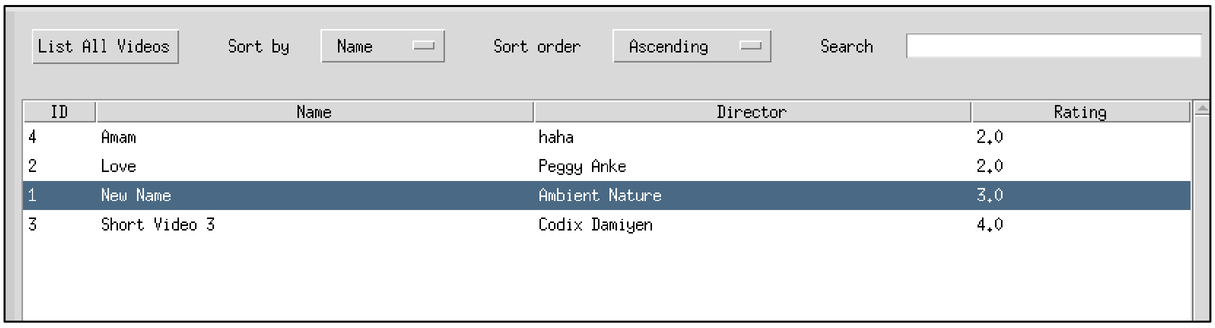


The sort will do its function when ListAllVideos button is clicked

- Test:

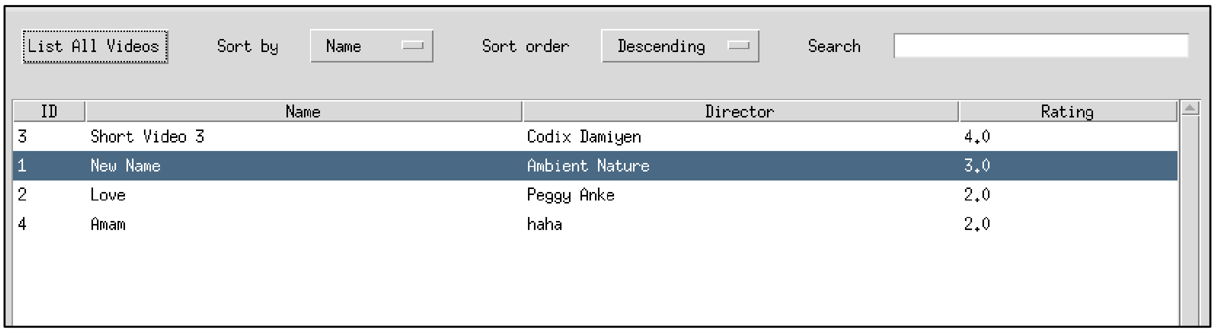


ID	Name	Director	Rating
1	New Name	Ambient Nature	3.0
2	Love	Peggy Anke	2.0
3	Short Video 3	Codix Damiyen	4.0
4	Amam	haha	2.0



List All Videos | Sort by: Name | Sort order: Ascending | Search

ID	Name	Director	Rating
4	Amam	haha	2.0
2	Love	Peggy Anke	2.0
1	New Name	Ambient Nature	3.0
3	Short Video 3	Codix Damiyen	4.0



List All Videos | Sort by: Name | Sort order: Descending | Search

ID	Name	Director	Rating
3	Short Video 3	Codix Damiyen	4.0
1	New Name	Ambient Nature	3.0
2	Love	Peggy Anke	2.0
4	Amam	haha	2.0

Play video

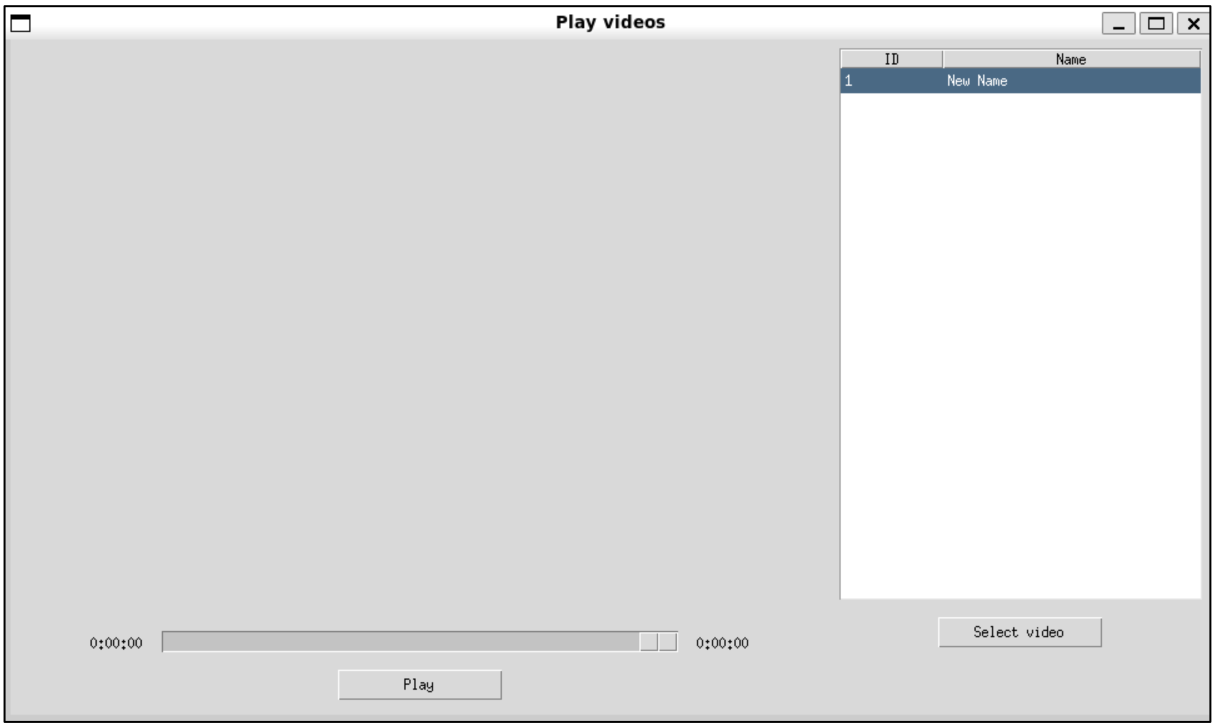
New button (Play) is added in CheckVideos UI

Use module tkVideoPlayer

When Play or Play Playlist is clicked, new TopLevel window will display containing video play area, and current playlist

Buttons:

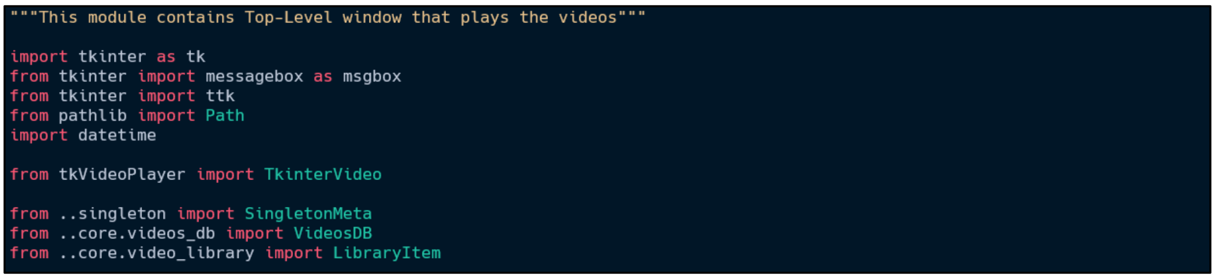
- Select Video: load the current selected video
- Play: play the loaded video



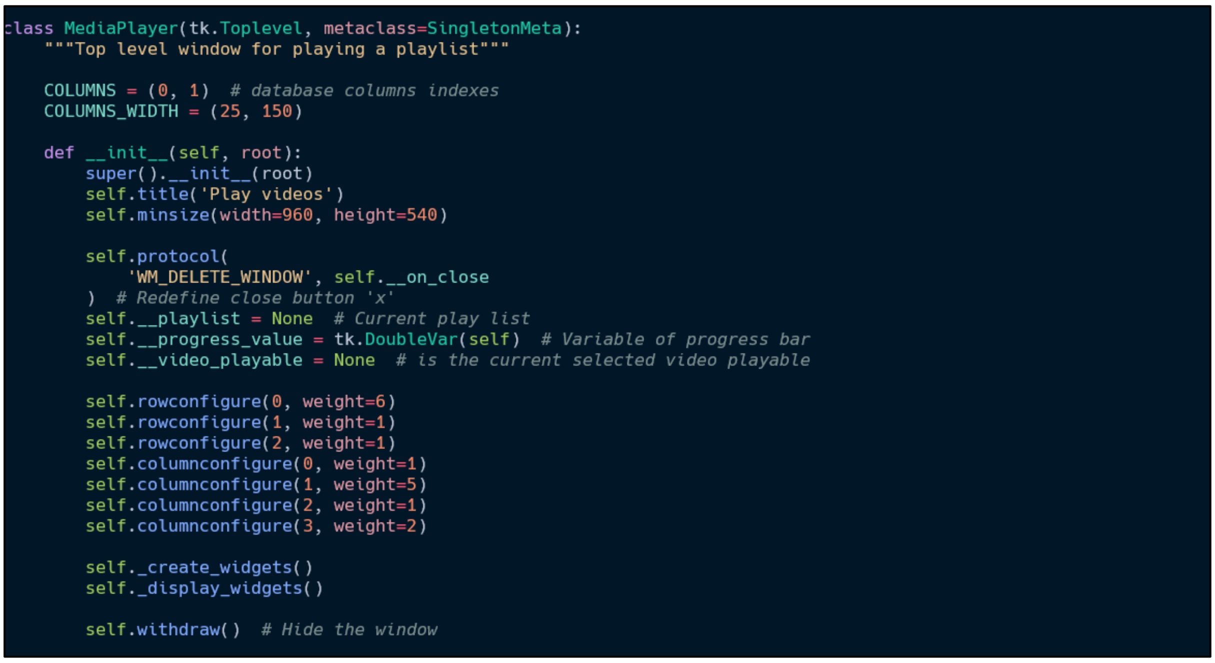
Play videos

ID	Name
1	New Name

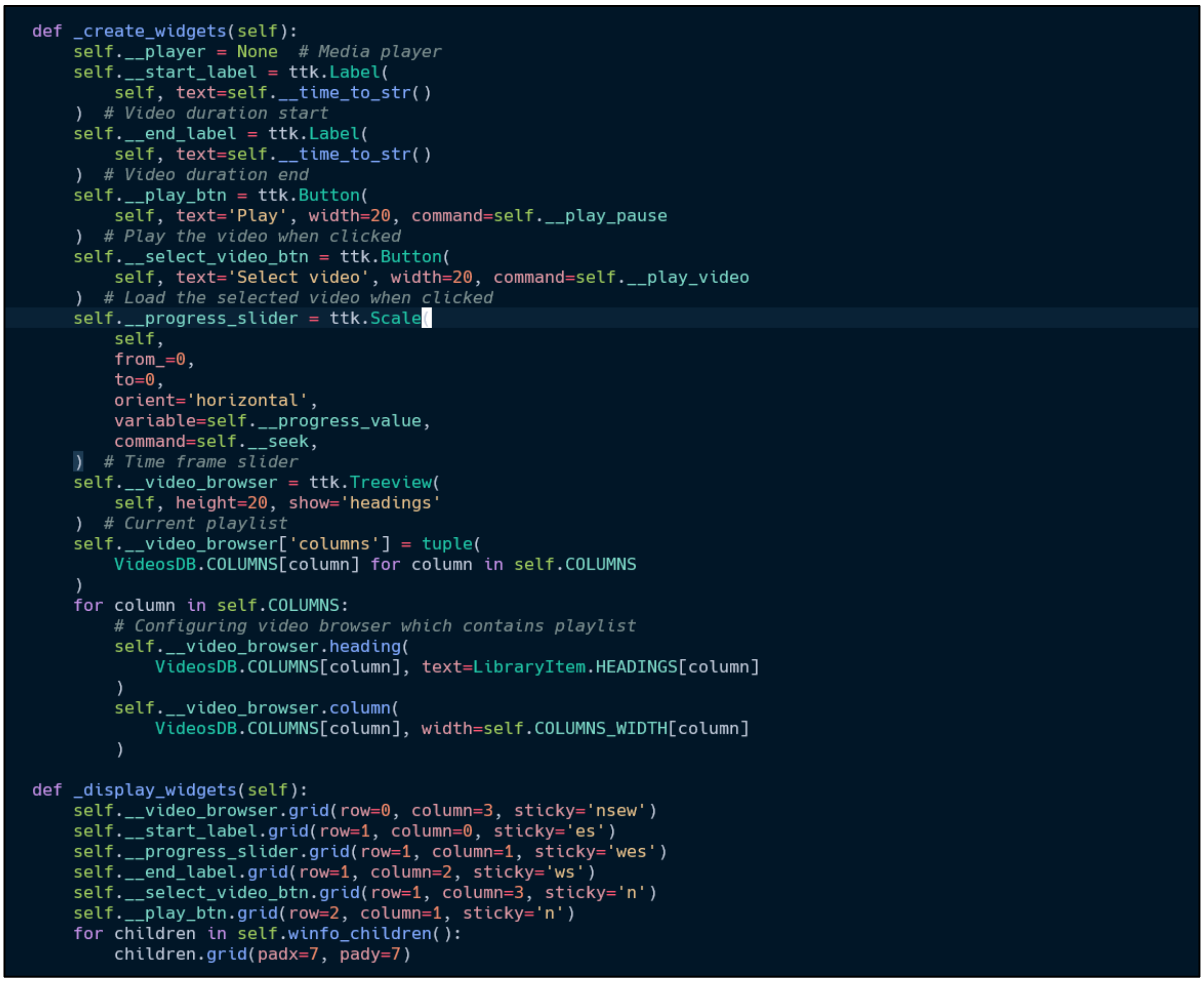
Implementation (media_player.py)



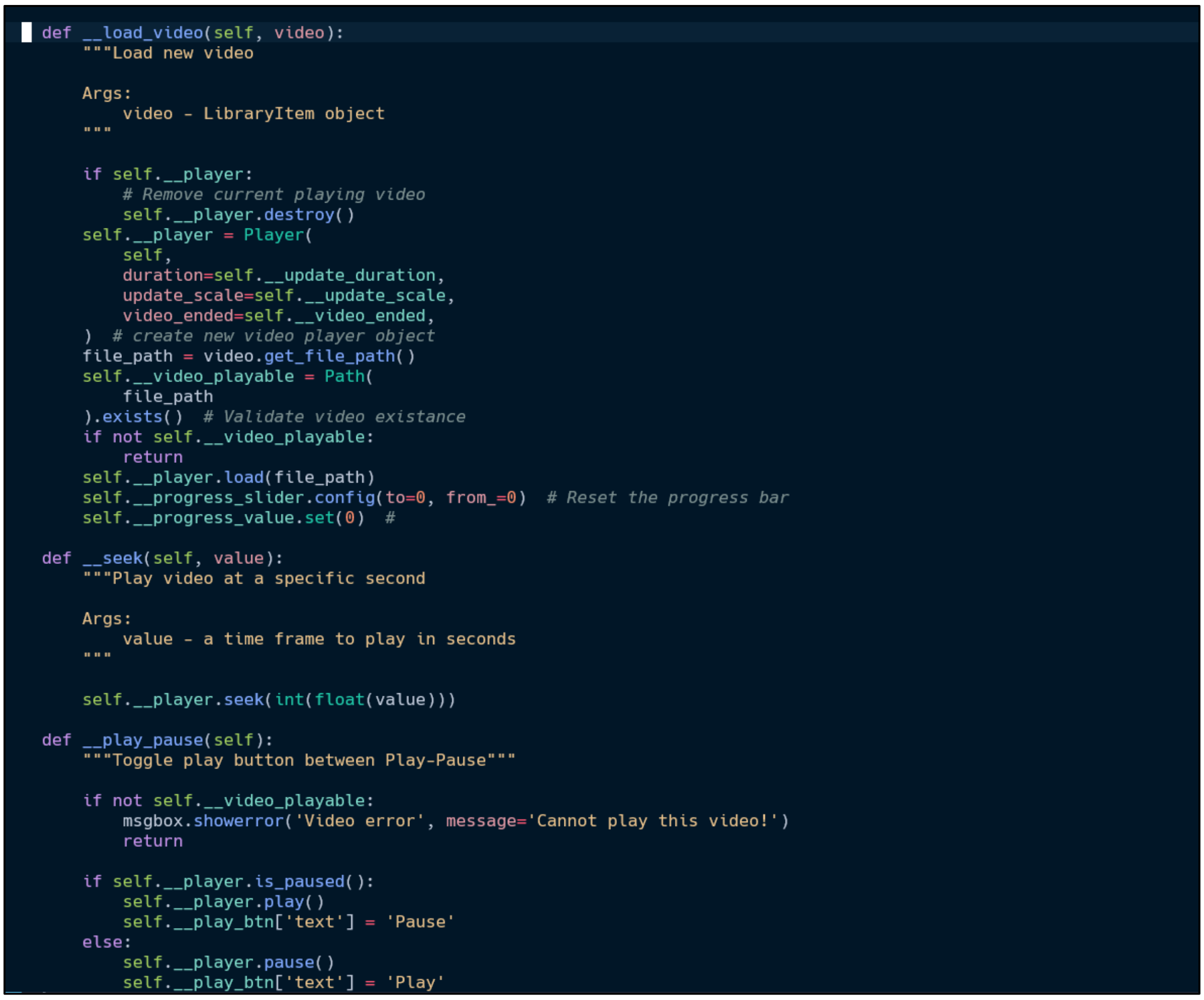
```python
"""This module contains Top-Level window that plays the videos"""

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
```



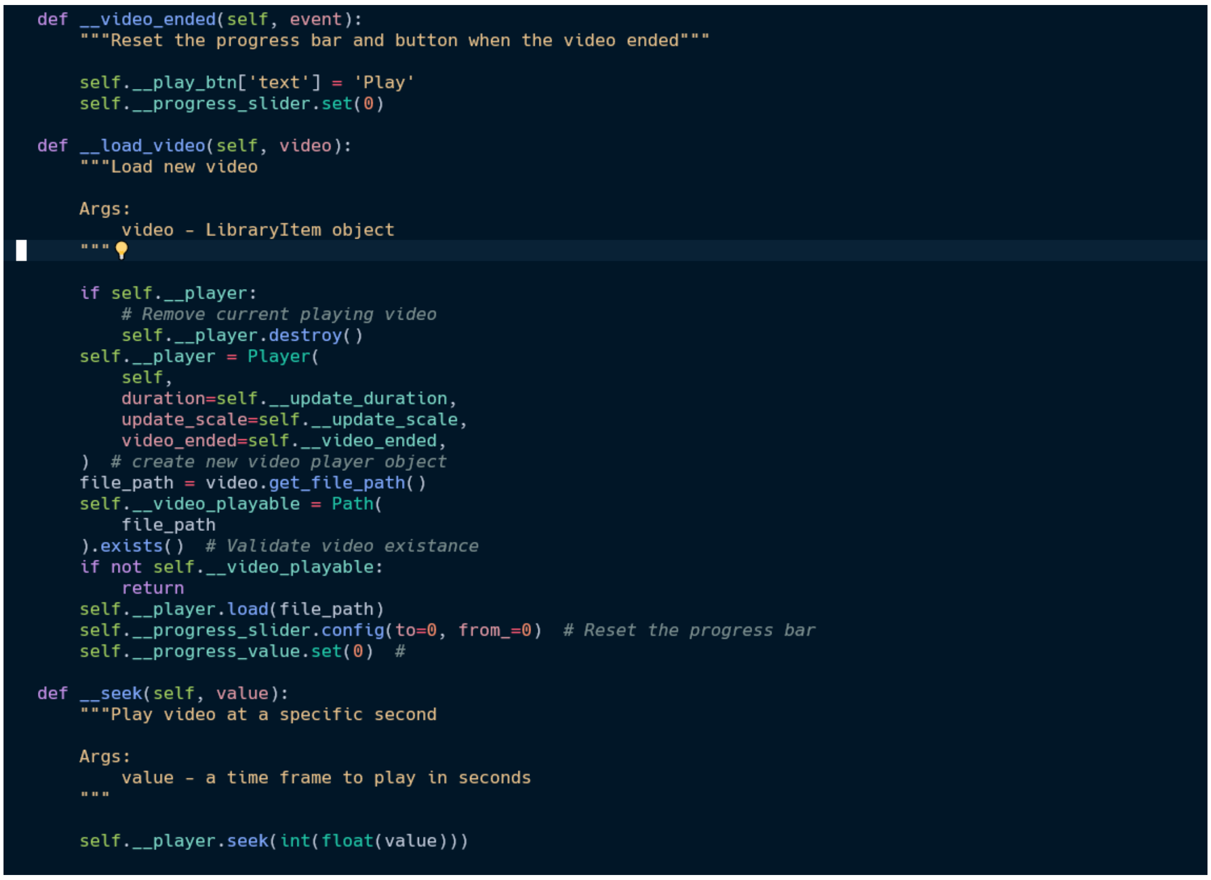
```python
class MediaPlayer(tk.Toplevel, metaclass=SingletonMeta):
    """Top level window for playing a playlist"""

    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
```



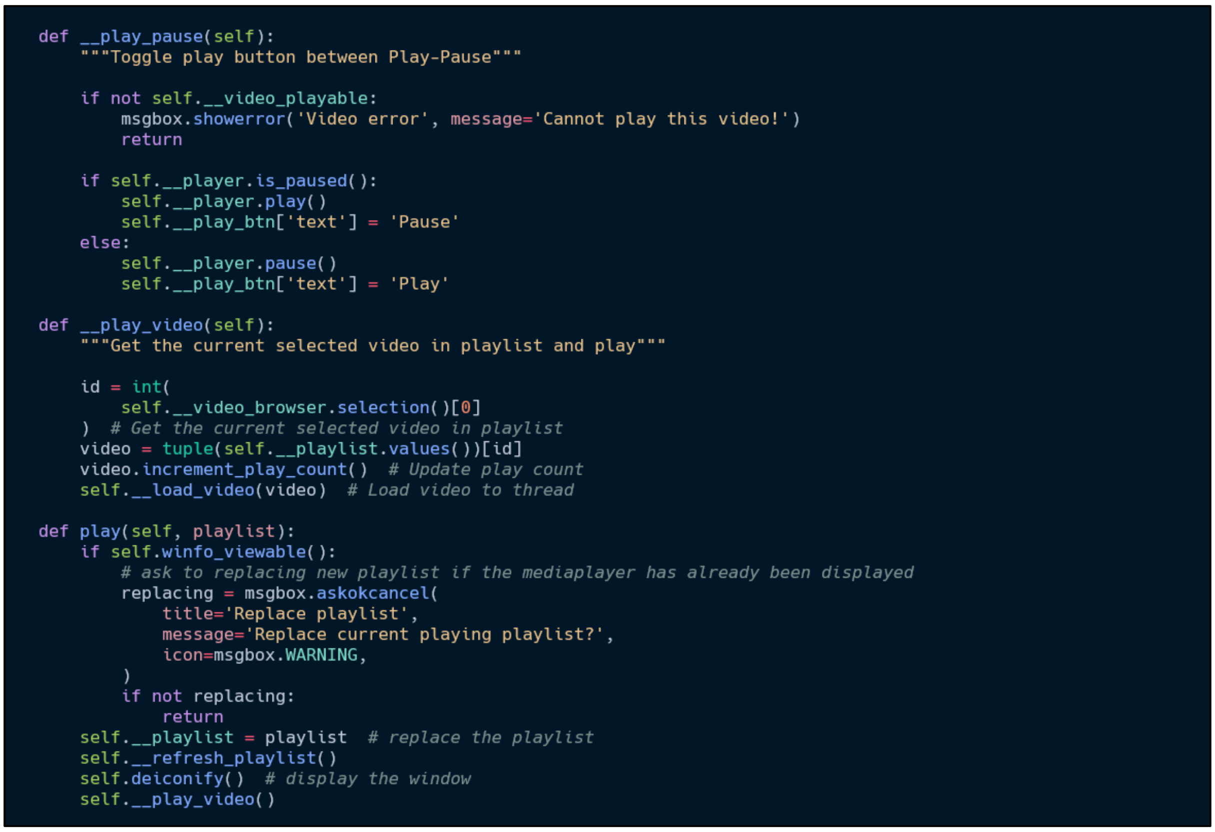
```python
def _create_widgets(self):
    self.__player = None  # Media player
    self.__start_label = ttk.Label(
        self, text=self.__time_to_str()
    )  # Video duration start
    self.__end_label = ttk.Label(
        self, text=self.__time_to_str()
    )  # Video duration end
    self.__play_btn = 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)
```



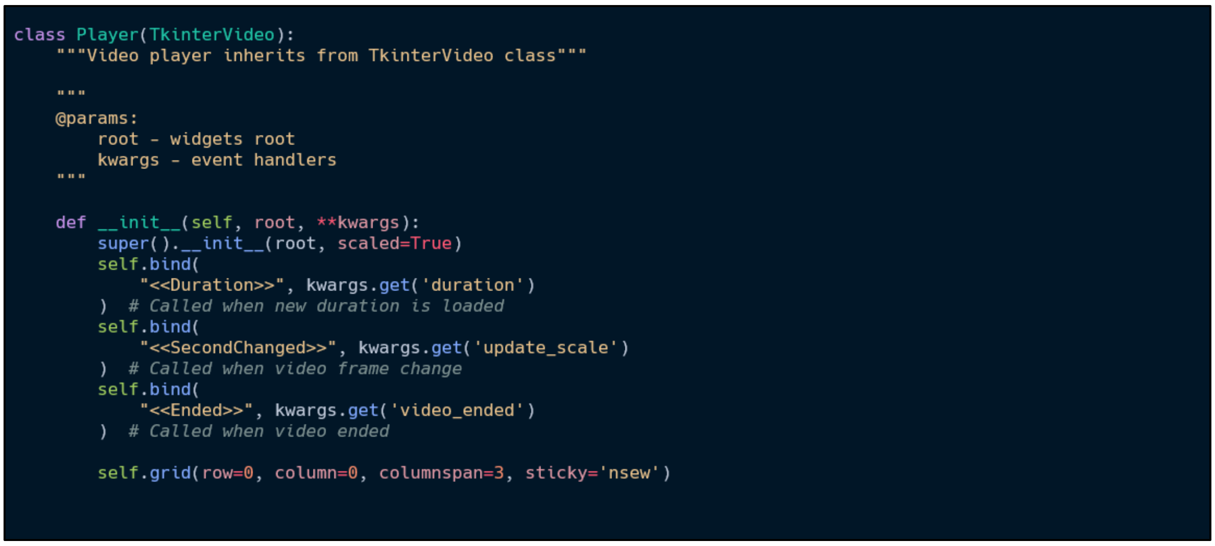
```
def __load_video(self, video):
    """Load new video

    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'
```



```python
    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)))
```



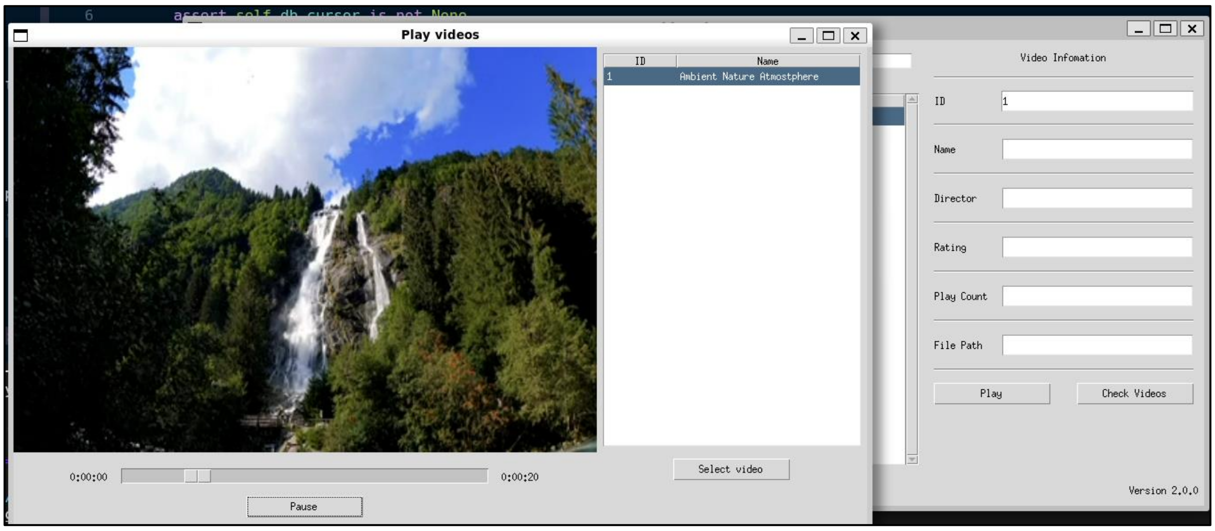
```python
    def __play_pause(self):
        """Toggle play button between Play-Pause"""

        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()
```



```
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 35 - MediaPlayer (media_player.py) implementation

- Test:



Play videos

ID	Name
1	Ambient Nature Atmostphere

0:00:00 0:00:20

Pause

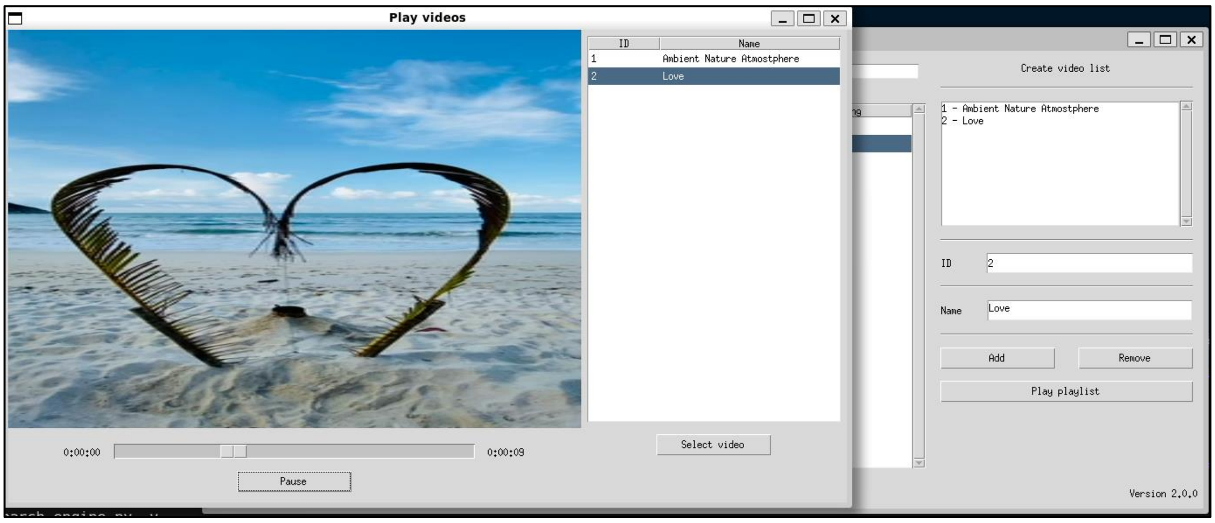
Select video

Video Infomation

ID	1
Name	
Director	
Rating	
Play Count	
File Path	

Play Check Videos

Version 2.0.0



Play videos

ID	Name
1	Ambient Nature Atmostphere
2	Love

0:00:00 0:00:09

Pause

Select video

Create video list

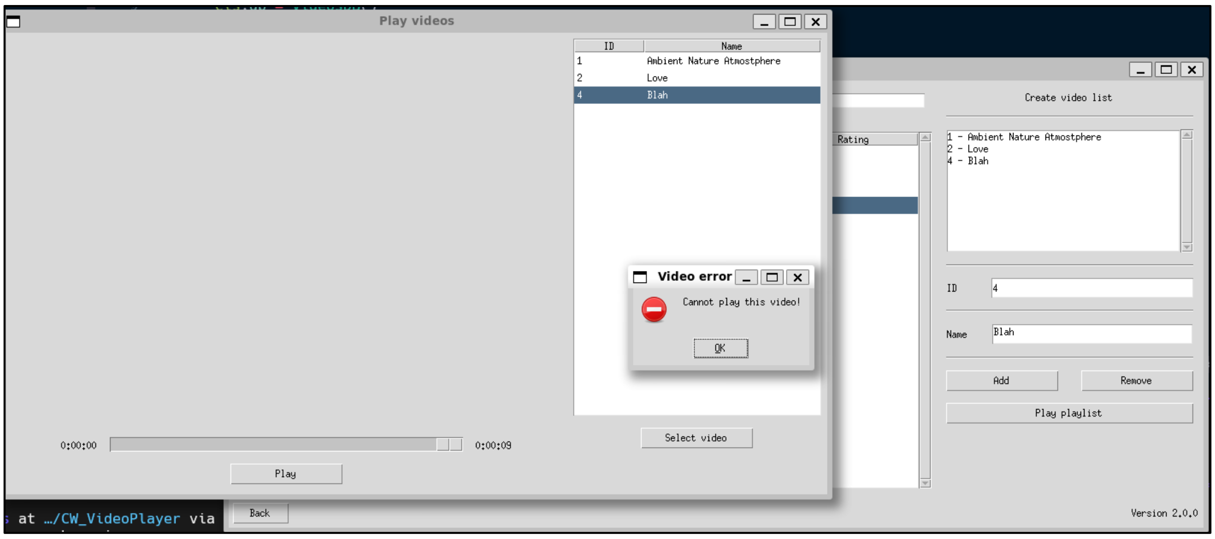
1 - Ambient Nature Atmostphere
2 - Love

ID	2
Name	Love

Add Remove

Play playlist

Version 2.0.0



CONCLUSION:

All required features are implemented correctly, with maintainable code and strictly validating and high demand innovation, the application satisfy the user requirement.