МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»

(БГТУ им. В.Г. Шухова)

Институт информационных технологий и управляющих систем

«КУРСОВАЯ РАБОТА»

По дисциплине: «Базы данных»

тема: «Приложения для поиска фильмов и актеров по фильмам»

Автор работы _____ Зарубин Даниил Евгеньевич ВТ-211

(Подпись)

Руководитель работы _____ Панченко Максим Владимирович

(Подпись)

Оценка _____

Белгород 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный мир кинематографа предлагает огромное количество фильмов и актеров, и найти интересующий фильм или информацию об актере может быть непростой задачей. В связи с этим возникает потребность в удобном инструменте для быстрого и эффективного поиска фильмов и актеров.

ЦЕЛЬ КУРСОВОЙ РАБОТЫ

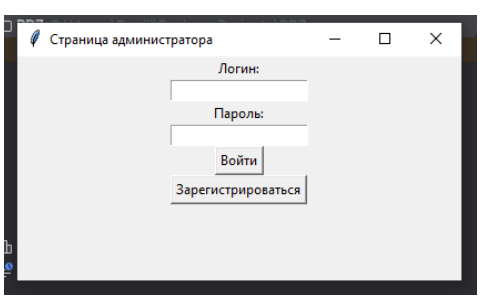
Целью данной курсовой работы является разработка приложения на Python для поиска фильмов и актеров по фильмам с использованием технологии ORM. Основными задачами приложения будут:

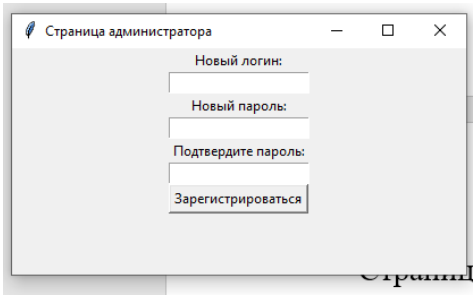
1. Поиск фильмов: Реализация механизма поиска фильмов по названию, году выпуска и другим характеристикам.
2. Просмотр информации о фильме: Возможность просмотра подробной информации о фильме, включая описание, актеров, рейтинги и рецензии.
3. Поиск актеров: Поиск актеров, участвовавших в определенном фильме, и получение информации о них.
4. Интерфейс пользователя: Создание простого и интуитивно понятного интерфейса пользователя для облегчения взаимодействия с приложением.

Это приложение будет полезным для кинолюбителей, исследователей кинематографа и всех, кто интересуется мирами кино и актерского мастерства. Использование технологии ORM (Object-Relational Mapping) позволит упростить доступ к базе данных и обеспечить более удобную работу с данными.

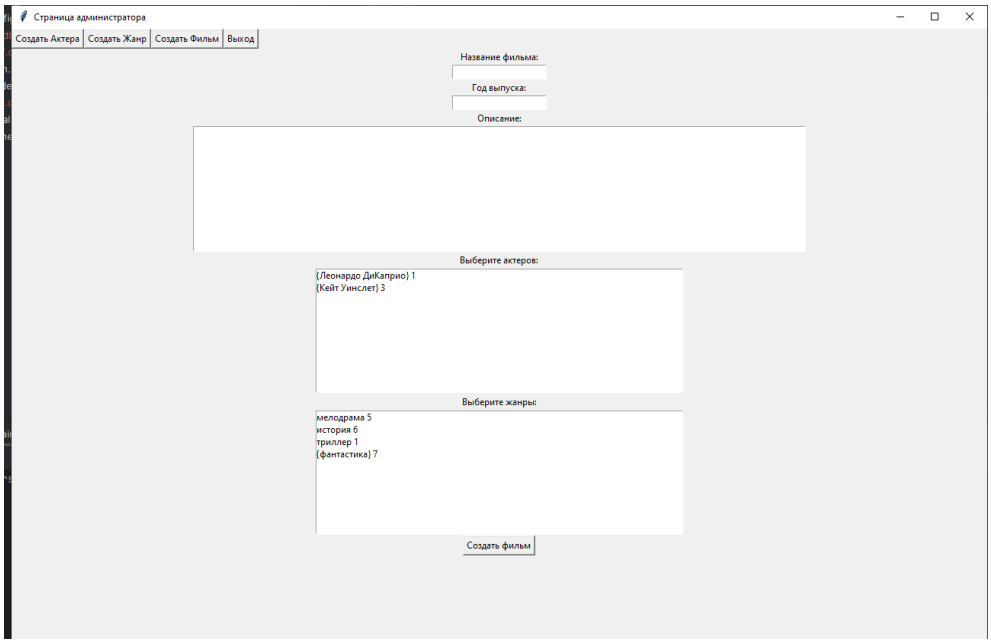
РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

Страница входа и регистрации:



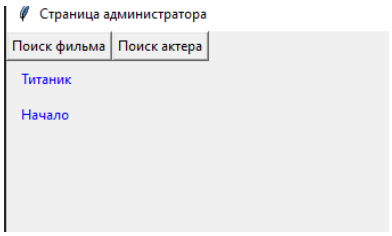


Страница администратора:

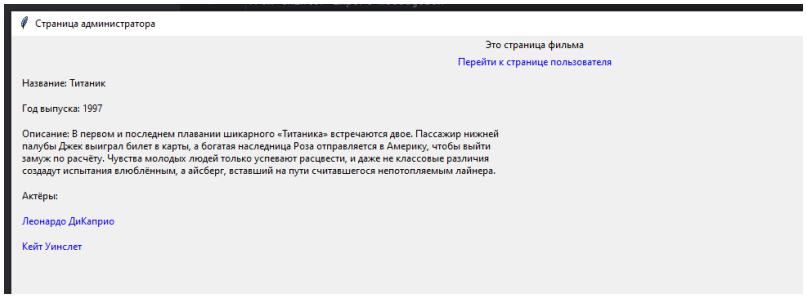


Страница пользователя:

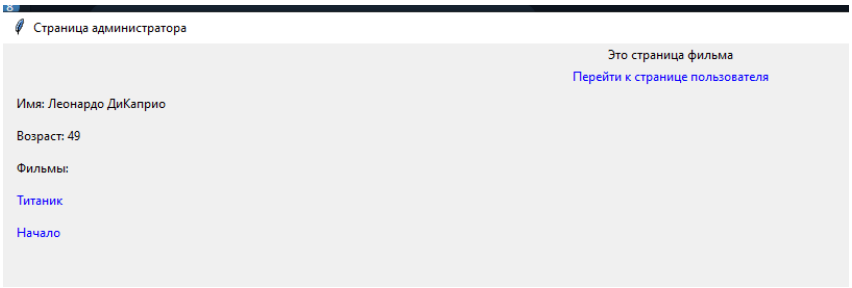
На ней сразу выводится список всех фильмов. Также доступны кнопки найти конкретный фильм или актера



Страничка фильма:



Страничка актёра:



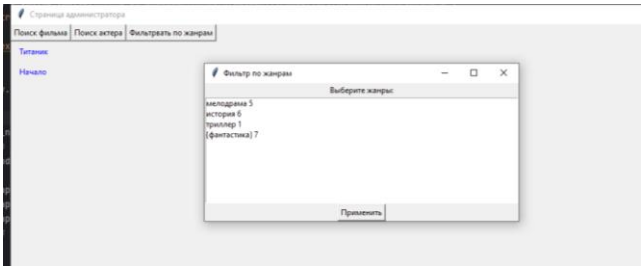
Все синии ссылки являются кликабельными.

Создание резервыной копии базы данных происходит каждый раз когда закрывается приложение.Сохранение происходит на яндекс диск.

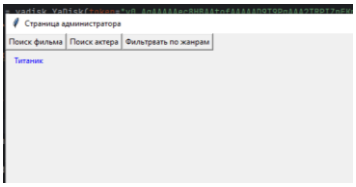
```
time_backup = time.strftime('%Y-%m-%d-%H-%M-%S')
path_file = f"backup_{time_backup}.sql"

command = "O:/PG16/bin/pg_dump -U {} -h {} -p {} -F c -b -v -f {} {}".format( *args: user, host, port, path_file, dbname)
subprocess.Popen(command, shell=True, env={'PGPASSWORD': password, 'SYSTEMROOT': os.environ['SYSTEMROOT']},
                 stdin=subprocess.PIPE)
```

На странице пользователя есть вкладка фильтр по жанрам



После выбора жанров и нажатие на кнопку применить основная страница прерисовывается оставляя те фильмы которые подходят по фильтрам а также создается pdf файл с списком фильмов.



ЛИСТИНГ ПРОГРАММЫ

config.py

```
# # В файле config.py будут описаны объект приложения Flask и объект,
# # предоставляющий инструменты для взаимодействия с базой данных.

from peewee import *
```

```
db = PostgresqlDatabase('cursach', user='postgres', password='1111',
host='localhost', port='5432')
db.connect()

class BaseModel(Model):
    class Meta:
        database = db
```

models.py

```
# В файле models.py с использованием объявленной в config.py базы
# данных будут объявлены базовые модели и все модели, необходимые для работы
приложения.
import peewee as pw
from config import BaseModel, db


class UserRole(BaseModel):
    id = pw.AutoField(column_name='id_role')
    name = pw.CharField(max_length=50, null=False, column_name='name_role')

    class Meta:
        database = db
        table_name = 'user_role'


class User(BaseModel):
    name = pw.CharField(primary_key=True, max_length=50, null=False,
unique=True, column_name='name_user')
    password = pw.CharField(max_length=50, null=False,
column_name='password_user')
    role = pw.ForeignKeyField(UserRole, backref='users', null=False,
column name='role user')

    class Meta:
        database = db
        table_name = 'user'


class Actor(BaseModel):
    id = pw.AutoField(column_name='id_actor')
    name = pw.CharField(max_length=100, unique=True, null=False,
column_name='name_actor')
    age = pw.IntegerField(null=False, column_name='age_actor')

    class Meta:
        database = db
        table_name = 'actor'


class Genres(BaseModel):
    id = pw.AutoField(column_name='id_genres')
    name = pw.CharField(max_length=100, unique=True, null=False,
column_name='name_genres')

    class Meta:
        database = db
        table_name = 'genres'


class Movie(BaseModel):
```

```
    id = pw.AutoField(column_name='id_movie')
    name = pw.CharField(max_length=100, null=False, column_name='name_movie')
    releaseYear = pw.IntegerField(column_name='release_year')
    description = pw.CharField(max_length=500, column_name='description')

    class Meta:
        database = db
        table_name = 'movie'


class GenresMovie(BaseModel):
    movie_id = pw.ForeignKeyField(Movie, backref='genmovies', null=False,
column_name='movie_id')
    genres_id = pw.ForeignKeyField(Genres, backref='genmovies', null=False,
column_name='genres_id')

    class Meta:
        database = db
        table_name = 'genres_movie'
        primary_key = pw.CompositeKey('movie_id', 'genres_id')


class PlayMovie(BaseModel):
    movie_id = pw.ForeignKeyField(Movie, backref='plmovies', null=False,
column_name='movie_id')
    actor_id = pw.ForeignKeyField(Actor, backref='plmovies', null=False,
column_name='actor_id')

    class Meta:
        database = db
        table_name = 'play_movie'
        primary_key = pw.CompositeKey('movie_id', 'actor_id')




# Создание таблиц для всех моделей
with db:
    db.create_tables([
        User, UserRole, Actor, Movie, GenresMovie, Genres, PlayMovie])
```

admin_page.py

```
from main import *

class AdminPage(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

        self.controller.title("Страница администратора")

        self.toolbar = tk.Frame(self)
        self.toolbar.pack(side="top", fill="x")

        button_creat_actor = tk.Button(self.toolbar, text="Создать Актера",
command=self.button_cr_ac_clicked)
        button_creat_actor.pack(side="left")

        button_creat_genres = tk.Button(self.toolbar, text="Создать Жанр",
command=self.button_cr_ge_clicked)
```

```
        button_creat_genres.pack(side="left")

        button_creat_movie = tk.Button(self.toolbar, text="Создать Фильм",
command=self.button_cr_mv_clicked)
        button_creat_movie.pack(side="left")

        button_exit = tk.Button(self.toolbar, text="Выход",
command=self.button_exit)
        button_exit.pack(side="left")

        self.form_panel = tk.Frame(self)  # Панель для формы (пока скрыта)
        self.actor_form_panel = tk.Frame(self.form_panel)  # Панель для формы
создания актера
        self.genres_form_panel = tk.Frame(self.form_panel)  # Панель для
формы создания жанров
        self.movie_form_panel = tk.Frame(self.form_panel)  # Панель для формы
создания фильма

        self.create_actor_form()
        self.create_genres_form()
        self.create_movie_form()

        self.form_panel.pack()

    def create_actor_form(self):
        # Элементы формы для создания актера
        self.actor_name_label = tk.Label(self.actor_form_panel, text="Имя
актера:")
        self.actor_name_label.pack()
        self.actor_name_entry = tk.Entry(self.actor_form_panel)
        self.actor_name_entry.pack()

        self.actor_age_label = tk.Label(self.actor_form_panel, text="Возраст
актера:")
        self.actor_age_label.pack()
        self.actor_age_entry = tk.Entry(self.actor_form_panel)
        self.actor_age_entry.pack()

        self.save_actor_button = tk.Button(self.actor_form_panel,
text="Создать актера", command=self.save_actor)
        self.save_actor_button.pack()

    def create_genres_form(self):
        self.genres_name_label = tk.Label(self.genres_form_panel,
text="Название жанра:")
        self.genres_name_label.pack()
        self.genres_name_entry = tk.Entry(self.genres_form_panel)
        self.genres_name_entry.pack()

        self.save_genres_button = tk.Button(self.genres_form_panel,
text="Создать жанр", command=self.save_genres)
        self.save_genres_button.pack()

    def create_movie_form(self):
        # Элементы формы для создания фильма
        self.movie name label = tk.Label(self.movie form panel,
text="Название фильма:")
        self.movie_name_label.pack()
        self.movie_name_entry = tk.Entry(self.movie_form_panel)
        self.movie_name_entry.pack()

        self.movie_year_label = tk.Label(self.movie_form_panel, text="Год
выпуска:")
```

```
        self.movie_year_label.pack()
        self.movie_year_entry = tk.Entry(self.movie_form_panel)
        self.movie_year_entry.pack()

        self.movie_description_label = tk.Label(self.movie_form_panel,
text="Описание:")
        self.movie_description_label.pack()
        self.movie_description_entry = tk.Text(self.movie_form_panel,
width=100, height=10)
        self.movie_description_entry.pack(fill=tk.BOTH, expand=True)
        # self.movie_description_entry =
tk.Entry(self.movie_form_panel,width=100)
        # self.movie_description_entry.pack()

        self.actor_list_label = tk.Label(self.movie_form_panel,
text="Выберите актеров:")
        self.actor_list_label.pack()

        # Создаем список для выбора актеров
        self.actor_listbox = tk.Listbox(self.movie_form_panel,
selectmode=tk.MULTIPLE, width=80, exportselection=0)
        self.actor_listbox.pack()

        # Заполняем список для выбора актеров
        self.fill_actor_listbox()

        self.genres_list_label = tk.Label(self.movie_form_panel,
text="Выберите жанры:")
        self.genres_list_label.pack()

        # Создаем список для выбора актеров
        self.genres_listbox = tk.Listbox(self.movie_form_panel,
selectmode=tk.MULTIPLE, width=80, exportselection=0)
        self.genres_listbox.pack()

        # Заполняем список для выбора актеров
        self.fill_genres_listbox()

        self.save_movie_button = tk.Button(self.movie_form_panel,
text="Создать фильм", command=self.save_movie)
        self.save_movie_button.pack()

    def save_actor(self):
        # Получаем данные из полей ввода
        actor_name = self.actor_name_entry.get()
        actor_age = self.actor_age_entry.get()

        if actor_name == "":
            messagebox.showerror("Ошибка", "Имя не введено")
            return
        if actor_age == "":
            messagebox.showerror("Ошибка", "Пароль не введен")
            return

        try:
            Actor.create(name=actor_name, age=int(actor_age))
            messagebox.showinfo("Успешно", "Успешно сохранен")
        except pw.IntegrityError:
            messagebox.showerror("Ошибка", "Такой актер уже есть")

    def save_genres(self):
        # Получаем данные из полей ввода
        genres_name = self.genres_name_entry.get()
```

```
        if genres_name == "":
            messagebox.showerror("Ошибка", "Имя не введено")
            return

        try:
            Genres.create(name=genres_name)
            messagebox.showinfo("Успешно", "Успешно сохранен")
        except pw.IntegrityError:
            messagebox.showerror("Ошибка", "Такой жанр уже создан")

    def save_movie(self):
        # Получаем данные из полей ввода

        movie_name = self.movie_name_entry.get()
        movie_year = self.movie_year_entry.get()
        movie_description = self.movie_description_entry.get("1.0", tk.END)
        index_selected_actors = [self.actor_listbox.get(idx)[1] for idx in
self.actor_listbox.curselection()]
        index_selected_genres = [self.genres_listbox.get(idx)[1] for idx in
self.genres_listbox.curselection()]

        if movie_name == "":
            messagebox.showerror("Ошибка", "Имя не введено")
            return

        if movie_year == "":
            messagebox.showerror("Ошибка", "Год не введено")
            return

        if movie_description == "":
            messagebox.showerror("Ошибка", "Описание не введено")
            return

        try:
            Movie.create(name=movie_name, releaseYear=int(movie_year),
description=movie_description)

            try:
                id = Movie.get(Movie.name == movie_name, Movie.releaseYear ==
movie_year)
                for i in index_selected_genres:
                    GenresMovie.create(movie_id=id.id, genres_id=i)
                for i in index_selected_actors:
                    PlayMovie.create(movie_id=id.id, actor_id=i)
            except pw.IntegrityError:
                messagebox.showerror("Ошибка", "Проблема с созданием")

            messagebox.showinfo("Успешно", "Успешно сохранен")
        except pw.IntegrityError:
            messagebox.showerror("Ошибка", "Такой фильм уже есть")

    def fill_actor_listbox(self):
        self.actor_listbox.delete(0, tk.END)
        actors = Actor.select()
        for actor in actors:
            self.actor_listbox.insert(tk.END, (actor.name, actor))

    def fill_genres_listbox(self):
        self.genres_listbox.delete(0, tk.END)
        genres = Genres.select()
        for genr in genres:
            self.genres_listbox.insert(tk.END, (genr.name, genr))
```

```
    def button_exit(self):
        self.controller.geometry("400x200")
        self.controller.show_frame(LoginPage)

    def button_cr_ac_clicked(self):
        self.actor_form_panel.pack()
        self.genres_form_panel.pack_forget()
        self.movie_form_panel.pack_forget()

    def button_cr_ge_clicked(self):
        self.genres_form_panel.pack()
        self.actor_form_panel.pack_forget()
        self.movie_form_panel.pack_forget()

    def button_cr_mv_clicked(self):
        self.fill_actor_listbox()
        self.fill_genres_listbox()
        self.movie_form_panel.pack()
        self.actor_form_panel.pack_forget()
        self.genres_form_panel.pack_forget()
```

main.py

```
# from workBD import BD
import os

import time
import zipfile
import tkinter as tk
from tkinter import messagebox
from admin_page import *
from tkinter.simpledialog import Dialog
import yadisk
from models import *
import subprocess

from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
from reportlab.pdfbase.ttfonts import TTFont
from reportlab.pdfbase import pdfmetrics
from reportlab.platypus import SimpleDocTemplate, Paragraph
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.styles import ParagraphStyle


class LoginPage(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

        username_label = tk.Label(self, text="Логин:")
        username_label.pack()
        self.username_entry = tk.Entry(self)
        self.username_entry.pack()

        password_label = tk.Label(self, text="Пароль:")
        password_label.pack()
        self.password_entry = tk.Entry(self, show="*")
        self.password_entry.pack()
```

```
        login_button = tk.Button(self, text="Войти", command=self.login)
        login_button.pack()

        register_button = tk.Button(self, text="Зарегистрироваться",
command=self.register)
        register_button.pack()

    def login(self):
        username = self.username_entry.get()
        password = self.password_entry.get()

        # формирует запрос на поиск в бд аккаунта
        try:
            user = User.get(User.name == username)
            if user.password == password:
                if user.role.name == 'user':
                    self.controller.geometry("1280x800")
                    self.controller.show_frame(UserPage)
                else:
                    self.controller.geometry("1280x800")
                    self.controller.show_frame(AdminPage)
            else:
                messagebox.showerror("Ошибка", "Неверный пароль")
        except User.DoesNotExist:
            messagebox.showerror("Ошибка", "Пользователь не найден")

    def register(self):
        self.controller.show_frame(RegisterPage)


class RegisterPage(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

        username_label = tk.Label(self, text="Новый логин:")
        username_label.pack()
        self.new_username_entry = tk.Entry(self)
        self.new_username_entry.pack()

        password_label = tk.Label(self, text="Новый пароль:")
        password_label.pack()
        self.new_password_entry = tk.Entry(self, show="*")
        self.new_password_entry.pack()

        confirm_password_label = tk.Label(self, text="Подтвердите пароль:")
        confirm_password_label.pack()
        self.confirm_password_entry = tk.Entry(self, show="*")
        self.confirm_password_entry.pack()

        register_button = tk.Button(self, text="Зарегистрироваться",
command=self.do_register)
        register_button.pack()

    def do_register(self):
        new username = self.new username entry.get()
        new_password = self.new_password_entry.get()
        confirm_password = self.confirm_password_entry.get()

        if new_username == "":
            messagebox.showerror("Ошибка", "Имя не введено")
            return
        if new_password == "":
```

```
            messagebox.showerror("Ошибка", "Пароль не введен")
            return

        if new_password == confirm_password:
            try:
                User.create(name=new_username, password=confirm_password,
role=2)
                messagebox.showinfo("Успешно", "Регистрация завершена")
                self.controller.show_frame(LoginPage)
            except pw.IntegrityError:
                messagebox.showerror("Ошибка", "Пользователь с даным именем
сущесвует")
        else:
            messagebox.showerror("Ошибка", "Пароли не совпадают")


class UserPage(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

        self.toolbar = tk.Frame(self)
        self.toolbar.pack(side="top", fill="x")

        button_search_movie = tk.Button(self.toolbar, text="Поиск фильма",
command=self.button_sh_mv_clicked)
        button_search_movie.pack(side="left")

        button_search_actor = tk.Button(self.toolbar, text="Поиск актера",
command=self.button_sh_ac_clicked)
        button_search_actor.pack(side="left")

        button_filter_genres = tk.Button(self.toolbar, text="Фильтрвать по
жанрам", command=self.button_fl_gn_clicked)
        button_filter_genres.pack(side="left")

        self.films_labl = []
        films = Movie.select()
        for film in films:
            film_label = tk.Label(self, text=film.name, fg="blue",
cursor="hand2")
            film_label.pack(anchor="w", padx=10, pady=5)
            film_label.bind("<Button-1>", lambda event, filmm=film:
self.show_movie_page(filmm))
            self.films_labl.append(film_label)

    def fill_genres_listbox(self, genres_listbox):
        genres_listbox.delete(0, tk.END)
        genres = Genres.select()
        for genr in genres:
            genres_listbox.insert(tk.END, (genr.name, genr))

    def button_fl_gn_clicked(self):
        genre_filter_window = tk.Toplevel(self)
        genre_filter_window.title("Фильтр по жанрам")

        genres_list_label = tk.Label(genre_filter_window, text="Выберите
жанры:")
        genres_list_label.pack()

        # Создаем список для выбора актеров
        genres_listbox = tk.Listbox(genre_filter_window,
selectmode=tk.MULTIPLE, width=80, exportselection=0)
```

```
        genres_listbox.pack()

        # Заполняем список для выбора актеров
        self.fill_genres_listbox(genres_listbox)

        def apply_filter(genres_listbox):
            index_selected_genres = [genres_listbox.get(idx)[1] for idx in
genres_listbox.curselection()]
            if len(index_selected_genres) != 0:
                for label in self.films_labl:
                    label.destroy()
                self.films_labl = []
                index_film = []
                for i in index_selected_genres:
                    gm = GenresMovie.select().where(GenresMovie.genres_id ==
i)
                    for j in gm:
                        index_film.append(str(j.movie_id))
                index_film = set(index_film)
                text = []
                for i in index_film:
                    film = Movie.get(Movie.id == i)
                    text.append(str(film.name))

                    film_label = tk.Label(self, text=film.name, fg="blue",
cursor="hand2")
                    film_label.pack(anchor="w", padx=10, pady=5)
                    film_label.bind("<Button-1>", lambda event, filmm=film:
self.show_movie_page(filmm))
                    self.films_labl.append(film_label)

                self.create_pdf(text)

            genre_filter_window.destroy()

        apply_button = tk.Button(genre_filter_window, text="Применить",
command=lambda: apply_filter(genres_listbox))
        apply_button.pack()

    def create_pdf(self, text):
        print(text)
        file_name = f"filter_{time.strftime('%Y-%m-%d-%H-%M-%S')}.pdf"
        # c = canvas.Canvas(file_name, pagesize=letter)
        # c.setFont('Times-Roman', 12)
        # for i in text:
        #     c.drawString(100, 750, i)  # Установка кодировки UTF-8
        # c.save()
        doc = SimpleDocTemplate(file_name, pagesize=letter)
        styles = getSampleStyleSheet()
        flowables = []
        # Определение стиля с нужным шрифтом
        style = ParagraphStyle(name='TimesNewRoman', fontName='Times-Roman')
        for line in text:
            para = Paragraph(line, style=style)
            flowables.append(para)

        doc.build(flowables)

    def show_movie_page(self, film):
        self.controller.show_frame(MoviePage)
        self.controller.frames[MoviePage].set_movie_info(film)

    def show_actor_page(self, actor):
```

```
        self.controller.show_frame(ActorPage)
        self.controller.frames[ActorPage].set_actor_info(actor)

    def button_sh_mv_clicked(self):
        # Создаем диалоговое окно для ввода названия фильма
        movie_name = tk.simpledialog.askstring("Поиск фильма", "Введите
название фильма:")
        if movie_name:
            # Выполняем поиск фильмов по введенному названию
            found_movies = Movie.select().where(Movie.name == movie_name)
            if found_movies:
                # Отображаем найденные фильмы в новом окне
                self.show_search_results_movies(found_movies)
            else:
                messagebox.showinfo("Результаты поиска", "Фильм с таким
названием не найден")

    def show_search_results_movies(self, movies):
        # Создаем новое окно для отображения результатов поиска
        search_results_window = tk.Toplevel(self)
        search_results_window.title("Результаты поиска")

        if movies:
            for movie in movies:
                text = f"{movie.name} : {movie.releaseYear}"
                movie_label = tk.Label(search_results_window, text=text,
fg="blue", cursor="hand2")
                movie_label.pack()
                movie_label.bind("<Button-1>", lambda event, film=movie:
self.show_movie_page(film))
        else:
            no_results_label = tk.Label(search_results_window, text="Фильмы с
таким названием не найдены")
            no_results_label.pack()

    def button_sh_ac_clicked(self):
        actor_name = tk.simpledialog.askstring("Поиск фильма", "Введите
название фильма:")
        if actor_name:
            found_actors = Actor.select().where(Actor.name == actor_name)
            if found_actors:
                self.show_search_results_actors(found_actors)
            else:
                messagebox.showinfo("Результаты поиска", "Фильм с таким
названием не найден")

    def show_search_results_actors(self, actors):
        search_results_window = tk.Toplevel(self)
        search_results_window.title("Результаты поиска")

        if actors:
            for actor in actors:
                text = f"{actor.name} : {actor.age}"
                movie_label = tk.Label(search_results_window, text=text,
fg="blue", cursor="hand2")
                movie label.pack()
                movie_label.bind("<Button-1>", lambda event, film=actor:
self.show_actor_page(film))
        else:
            no_results_label = tk.Label(search_results_window, text="Фильмы с
таким названием не найдены")
            no_results_label.pack()
```

```
class MoviePage(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

        label = tk.Label(self, text="Это страница фильма")
        label.pack()

        user_label = tk.Label(self, text="Перейти к странице пользователя",
fg="blue", cursor="hand2")
        user_label.pack()
        user_label.bind("<Button-1>", lambda event:
controller.show_frame(UserPage))

        self.movie_name_label = tk.Label(self, text="Название:")
        self.movie_name_label.pack(anchor="w", padx=10, pady=5)

        self.movie_release_label = tk.Label(self, text="Год выпуска:")
        self.movie_release_label.pack(anchor="w", padx=10, pady=5)

        self.movie_description_label = tk.Label(self, text="Описание:",
wraplength=600, justify="left")
        self.movie_description_label.pack(anchor="w", padx=10, pady=5)

        self.movie_actors_label = tk.Label(self, text="Актёры:",
wraplength=500, justify="left")
        self.movie_actors_label.pack(anchor="w", padx=10, pady=5)

        self.actor_labels = []

    def show_actor_page(self, actor):
        self.controller.show_frame(ActorPage)
        self.controller.frames[ActorPage].set_actor_info(actor)

    def set_movie_info(self, film):
        self.movie_name_label.config(text=f"Название: {film.name}")
        self.movie_release_label.config(text=f"Год выпуска:
{film.releaseYear}")
        self.movie_description_label.config(text=f"Описание:
{film.description}")

        for label in self.actor_labels:
            label.destroy()
        self.actor_labels = []

        actors = PlayMovie.select().where(PlayMovie.movie_id == film.id)
        for i in actors:
            actor = Actor.get(Actor.id == i.actor_id)
            actor_label = tk.Label(self, text=actor.name, fg="blue",
cursor="hand2")
            actor_label.pack(anchor="w", padx=10, pady=5)
            actor_label.bind("<Button-1>", lambda event, actorr=actor:
self.show_actor_page(actorr))
            self.actor_labels.append(actor_label)


class ActorPage(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

        label = tk.Label(self, text="Это страница фильма")
```

```
        label.pack()

        user_label = tk.Label(self, text="Перейти к странице пользователя",
fg="blue", cursor="hand2")
        user_label.pack()
        user_label.bind("<Button-1>", lambda event:
controller.show_frame(UserPage))

        self.actor_name_label = tk.Label(self, text="Имя:")
        self.actor_name_label.pack(anchor="w", padx=10, pady=5)

        self.actor_age_label = tk.Label(self, text="Возраст:")
        self.actor_age_label.pack(anchor="w", padx=10, pady=5)

        self.actor_movies_label = tk.Label(self, text="Фильмы:",
wraplength=500, justify="left")
        self.actor_movies_label.pack(anchor="w", padx=10, pady=5)

        self.film_labels = []

    def show_movie_page(self, film):
        self.controller.show_frame(MoviePage)
        self.controller.frames[MoviePage].set_movie_info(film)

    def set_actor_info(self, actor):
        self.actor_name_label.config(text=f"Имя: {actor.name}")
        self.actor_age_label.config(text=f"Возраст: {actor.age}")

        for label in self.film_labels:
            label.destroy()
        self.film_labels = []

        films = Movie.select()
        for film in films:
            film_label = tk.Label(self, text=film.name, fg="blue",
cursor="hand2")
            film_label.pack(anchor="w", padx=10, pady=5)
            film_label.bind("<Button-1>", lambda event, filmm=film:
self.show_movie_page(filmm))
            self.film_labels.append(film_label)


class SampleApp(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)

        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}

        for F in (LoginPage, RegisterPage, UserPage, AdminPage, MoviePage,
ActorPage):
            frame = F(container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(LoginPage)

    def show_frame(self, cont):
        frame = self.frames[cont]
```

```
        frame.tkraise()


def do2():
    host = 'localhost'
    port = '5432'
    user = 'postgres'
    password = '1111'
    dbname = 'cursach'

    time_backup = time.strftime('%Y-%m-%d-%H-%M-%S')
    path_file = f"backup_{time_backup}.sql"

    command = "O:/PG16/bin/pg_dump -U {} -h {} -p {} -F c -b -v -f {}
{}".format(user, host, port, path_file, dbname)
    subprocess.Popen(command, shell=True, env={'PGPASSWORD': password,
'SYSTEMROOT': os.environ['SYSTEMROOT']},
                     stdin=subprocess.PIPE)

    YANDEX_DIR = "/backup/"
    # ZIP_NAME = f"backup_{time_backup}.zip"

    y =
yadisk.YaDisk(token="y0_AgAAAAec8HBAAtofAAAAAD9T9PqAAA2TRPIZnFKmoZoeYu08EvoI
R6w-A")
    try:
        y.mkdir(f"{YANDEX_DIR}")
    except:
        pass

    y.upload(path_file, f"{YANDEX_DIR} {path_file}")


if __name__ == '__main__':
    # Загрузка шрифта
    pdfmetrics.registerFont(TTFont('Times-Roman', 'times.ttf'))

    app = SampleApp()
    app.geometry("400x200")
    app.mainloop()
    do2()
```

ВЫВОД О ПРОДЕЛАНОЙ РАБОТЕ

Разработанное приложение на Python с использованием технологии ORM предоставляет удобный интерфейс для поиска фильмов и актеров. Оно обеспечивает простой доступ к обширной базе данных и предоставляет широкий набор функциональных возможностей для удобного поиска и изучения информации о фильмах и актерах. Создание приложения на основе ORM упрощает работу с данными и делает его более эффективным и масштабируемым. Таким образом, разработанное приложение представляет

собой полезный инструмент для кинолюбителей и исследователей киноиндустрии.

Ссылка на github: https://github.com/Astolfen/Kur