

Министерство цифрового развития, связи и массовых коммуникаций Российской
Федерации
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

ОТЧЕТ

по учебной практике
на кафедре Прикладной Математики и Кибернетики

Выполнил:

студент гр. ИП-311

«5» апреля 2025г.

Мерлинский Г.Я.

Руководитель практики

доцент каф. ПМиК

«5» апреля 2025г.

Приставка П.А.

Оценка_____

Новосибирск 2025г.

Оглавление

1.Условие задачи.....	3
2. Описание используемых алгоритмов.....	4
3. Листинг программы.....	6
4. Результаты тестирования.....	22
5. Список используемых источников.....	24

1.Условие задачи

Разработать программу реализующую ввод, хранение и обработку данных о 250 лучших фильмах на основе данных сайта

<https://www.kinopoisk.ru/lists/movies/top250/>

Общие требования к программе:

1. Язык разработки: **Python версии не ниже 3.x**
2. Операционная система: определяются студентом
3. Набор свойств фильмов:

- Name – наименование
- alternativeName
- Year – год выпуска
- shortDescription – короткое описание
- movieLength- длительность
- top250 – позиция в рейтинге

4. Ввод данных

Непосредственно с главной страницы сайта

<https://www.kinopoisk.ru/lists/movies/top250/> в момент запуска программы.

Загрузка и парсинг веб-страницы производится с помощью библиотек Requests и BeautifulSoup или их аналогов. Примечание: допускается считывание строчек в количестве менее 250 (Например, 10 строчек с данными о фильмах).

5. Хранение:

Типы и структуры для хранения данных: определяются студентом

6. Обработка:

Реализовать функцию поиска информации о свойствах фильма по его названию.

2. Описание используемых алгоритмов

Я занимался следующими алгоритмами:

В слое data:

В классе **MovieRepository**:

binarySearch(self, sorted_movies: List[Movie], query: str) -> List[Movie] – берёт отсортированный массив фильмов и осуществляет бинарный поиск по переменной query, ключом в сортировке выступает название фильма.

Возвращает: список найденных по названию объектов 'Movie'.

searchMovies(self, movies: List[Movie], query: str)->List[Movie] – получает массив фильмов на вход, после сортирует их при помощи QuickSort-a, написанного моим напарником, а затем находит фильмы по названию, используя бинарный поиск.

Возвращает: список найденных по названию объектов 'Movie'.

class ImageStorage – управляет кэшированием изображений. Создает директорию для изображений при инициализации. Содержит в себе методы:

get_image(self, url: str, size=(136,204)) – Загружает изображение из директории картинок по кэш-имени. Если изображения нету в папке – то оно берётся с сайта и сохраняется в папке, кэшируясь таким образом.

class KinopoiskAPI – отвечает за сбор данных с сайта Кинопоиска. Во время инициализации берёт из файла config ссылку на сайт, с которого считывает информацию о фильмах, и api ключ, с помощью которого происходит сбор. Содержит в себе методы:

get_top250(self) – делает get запрос на сайт Кинопоиска с api ключом, получая топ 250 фильмов.

Возвращает: если запрос проходит успешно – то возвращает собранные фильмы в виде массива, иначе возвращает пустой массив.

В слое domain:

class Movie() - класс, хранящий в себе данные какого-то конкретного фильма. Содержит в себе метод:

from_dict(cls, data: Dict[str,any]) -> 'Movie' – метод, сохраняющий в классе переменные, значение которых берётся из словаря, поданного на вход.

В слое presentation:

В этом слое приложения содержатся функции, поддерживающие графическое отображение.

class MovieUI – управляет всем графическим отображением. При инициализации получает на вход объект tkinter окна и сохраняет его, как внутриклассовую переменную.

setup_ui(self) – устанавливает размер окна, его название, а также выполняет методы **setup_navigation** и **setup_scrollable_area**, реализованные моим напарником.

get_color_by_genre(self,genres: List[str]) – определяет, какой должен быть цвет у названия фильма при выводе в консоли с соответствии с жанрами этого фильма.

Возвращает: цвет для названия фильма в консоли в соответствии с его жанрами.

Тестирование

Нами с напарником было решено кроме реализации основного функционала сделать покрытие базовых функций тестами. Мой напарник реализовал **ui** тесты, а я – **unit** тесты. Вот, что проверяют мои тесты:

test_get_top250_cache_hit() - проверяет, корректно ли собирает информацию о фильмах класс **MovieRepository**.

test_get_top250_cache_miss() – проверяет, корректно ли собирает информацию о фильмах класс **KinopoiskAPI**.

test_quick_sort(mock_movies) – проверяет, работает ли в классе **MovieRepository** быстрая сортировка.

test_binary_movies(mock_movies) – проверяет, правильно ли работает в классе **MovieRepository** бинарный поиск.

test_search_movies(mock_movies) – проверяет, правильно ли работает поиск фильмов по имени в классе **MovieRepository**.

3. Листинг программы

```
//models.py
```

```
from dataclasses import dataclass
from typing import List, Optional, Dict, Any
```

```
@dataclass
```

```
class Movie:
    id: str
    name: str
    year: int
    rating: float
    genres: List[str]
    countries: List[str]
    poster_url: Optional[str]
```

```
    @classmethod
```

```
    def from_dict(cls, data: Dict[str, Any]) -> 'Movie':
        return cls(
            id=str(data.get('id', '')),
            name=data.get('name', ''),
            year=data.get('year', 0),
            rating=data.get('rating', {}).get('kp', 0.0),
            genres=[g.get('name', '') for g in data.get('genres', [])],
            countries=[c.get('name', '') for c in data.get('countries', [])],
            poster_url=data.get('poster', {}).get('url', None)
        )
```

```
//movie_Irepository.py
```

```
from abc import ABC, abstractmethod
from typing import List
from domain.models import Movie
```

```
class IMovieRepository(ABC):
```

```
    @abstractmethod
```

```
    def get_top250(self) -> List[Movie]:
        pass
```

```

    @abstractmethod
    def search_movies(self, movies: List[Movie], query: str) -> List[Movie]:
        pass

    @abstractmethod
    def sort_movies(self, movies: List[Movie]) -> List[Movie]:
        pass

//kinopoisk_api
import requests
from config import API_KEY, API_URL

class KinopoiskAPI:
    def __init__(self):
        self.headers = {'X-API-KEY': API_KEY}
        self.base_url = API_URL

    def get_top250(self):
        params = {
            'selectFields': ['name', 'rating.kp', 'year', 'genres.name',
            'countries.name', 'poster.url'],
            'lists': 'top250',
            'limit': 250
        }
        try:
            response = requests.get(self.base_url, headers=self.headers,
            params=params, timeout=10)
            if response.status_code == 200:
                return response.json().get('docs', [])
        except requests.RequestException:
            return []
        return []

```

```

//movie_repository.py

from typing import List

from domain.models import Movie

from infrastructure.api.kinopoisk_api import KinopoiskAPI

from infrastructure.cache.cache_manager import CacheManager

from functools import lru_cache

from domain.movie_repository import IMovieRepository


class MovieRepository(IMovieRepository):

    def __init__(self):

        self.api = KinopoiskAPI()

        self.cache = CacheManager('data/cache/movies_cache.json')

    @lru_cache(maxsize=1)

    def get_top250(self) -> List[Movie]:

        cached = self.cache.load()

        if cached:

            return [Movie.from_dict(m) for m in cached]

        data = self.api.get_top250()

        if data:

            self.cache.save(data)

            return [Movie.from_dict(m) for m in data]

        return []

    def QuickSort(self, movies: List[Movie], L: int, R: int) -> None:

        while L < R:

            i, j = L, R

            pivot = movies[L].name.lower()

```



```

        while i <= j:
            while movies[i].name.lower() < pivot:
                i += 1
            while movies[j].name.lower() > pivot:
                j -= 1
            if i <= j:
                movies[i], movies[j] = movies[j], movies[i]
                i += 1
                j -= 1

        if (j - L) < (R - i):
            self.QuickSort(movies, L, j)
            L = i
        else:
            self.QuickSort(movies, i, R)
            R = j

    def binary_search(self, sorted_movies: List[Movie], query: str) ->
List[Movie]:
        if not query:
            return []

        query = query.lower()
        left, right = 0, len(sorted_movies) - 1

        while left < right:
            mid = (left + right) // 2
            if sorted_movies[mid].name.lower() >= query:
                right = mid
            else:

```

```

        left = mid + 1

    result = []

    while left < len(sorted_movies) and query in
sorted_movies[left].name.lower():

        result.append(sorted_movies[left])

        left += 1

    return result

def search_movies(self, movies: List[Movie], query: str) -> List[Movie]:

    if not query:

        return []

    sorted_movies = movies.copy()

    self.QuickSort(sorted_movies, 0, len(sorted_movies) - 1)

    return self.binary_search(sorted_movies, query)

def sort_movies(self, movies: List[Movie]) -> List[Movie]:

    sorted_movies = movies.copy()

    self.QuickSort(sorted_movies, 0, len(sorted_movies) - 1)

    return sorted_movies

//image_storage.py

import os

from hashlib import md5

from PIL import Image, ImageTk

from io import BytesIO

import requests

class ImageStorage:

```

```

def __init__(self, cache_dir: str):
    self.cache_dir = cache_dir
    os.makedirs(cache_dir, exist_ok=True)

def get_image(self, url: str, size=(136, 204)):
    if not url:
        return None

    cache_name =
f"{md5(url.encode()).hexdigest()}_{size[0]}x{size[1]}.png"
    cache_path = os.path.join(self.cache_dir, cache_name)

    if os.path.exists(cache_path):
        try:
            return ImageTk.PhotoImage(Image.open(cache_path))
        except Exception:
            os.remove(cache_path)

    try:
        response = requests.get(url, timeout=10)
        if response.status_code == 200:
            img = Image.open(BytesIO(response.content))
            img = img.resize(size, Image.LANCZOS)
            img.save(cache_path, "PNG")
            return ImageTk.PhotoImage(img)
    except Exception:
        return None

```

//movie_service.py

```

from domain.movie_Irepository import IMovieRepository
from domain.models import Movie

```

```

from typing import List

class MovieService:
    def __init__(self, repository: IMovieRepository):
        self.repository = repository

    def fetch_movies(self) -> List[Movie]:
        return self.repository.get_top250()

    def search_movies(self, query: str) -> List[Movie]:
        movies = self.repository.get_top250()
        return self.repository.search_movies(movies, query)

    def get_sorted_movies(self) -> List[Movie]:
        movies = self.repository.get_top250()
        return self.repository.sort_movies(movies)

//movie_ui.py

from tkinter import messagebox, Frame, Label, Button, Entry, Canvas,
Scrollbar

from typing import List

from domain.models import Movie

from infrastructure.storage.image_storage import ImageStorage

from colorama import Fore, Style, init

from infrastructure.repository.movie_repository import MovieRepository

class MovieUI:
    def __init__(self, root):
        self.root = root

        self.movies: List[Movie] = []

        self.movies_for_search: List[Movie] = []

```

```

self.page = 1

self.print_in_terminal = True


self.image_storage = ImageStorage('data/images')

self.setup_ui()

init(autoreset=True)


def setup_ui(self):

    self.root.title("Кинопоиск: Топ 250 фильмов")

    self.root.geometry("650x700")


    self.setup_navigation()


    self.setup_scrollable_area()


def setup_navigation(self):

    nav_frame = Frame(self.root, bg="lightgray", pady=10)

    nav_frame.pack(fill="x")


    Button(nav_frame, text="← Предыдущая", command=self.prev_page,
font=("Arial", 8)).pack(side="left", padx=5)


    Button(nav_frame, text="Следующая →", command=self.next_page,
font=("Arial", 8)).pack(side="left", padx=0)


    self.page_entry = Entry(nav_frame, width=10)

    self.page_entry.pack(side="left", padx=10)


    Button(nav_frame, text="Перейти",
command=self.go_to_page).pack(side="left", padx=10)


    self.page_label = Label(nav_frame, text=f"Номер страницы:
{self.page}", font=("Arial", 8))

    self.page_label.pack(side="left")

```

```

        Button(nav_frame, text="Поиск", command=self.start_search,
font=("Arial", 8)).pack(side="left", padx=5)

        Button(nav_frame, text="Окончить поиск", command=self.end_search,
font=("Arial", 6)).pack(side="left")

self.search_entry = Entry(nav_frame, font=("Arial", 12), width=10)
self.search_entry.pack(side="left", padx=5)

def setup_scrollable_area(self):
    self.canvas = Canvas(self.root, bg="white")
    self.scrollbar = Scrollbar(self.root, orient="vertical",
command=self.canvas.yview)

    self.canvas.configure(yscrollcommand=self.scrollbar.set)
    self.canvas.bind('<Configure>', lambda e:
self.canvas.configure(scrollregion=self.canvas.bbox("all")))

    self.scrollbar.pack(side="right", fill="y")
    self.canvas.pack(side="left", fill="both", expand=True)

    self.movie_frame = Frame(self.canvas, bg="white")
    self.canvas.create_window((0, 0), window=self.movie_frame,
anchor="nw")

def create_movie_cards(self, movies: List[Movie]):
    for widget in self.movie_frame.winfo_children():
        widget.destroy()

    for movie in movies:
        self.create_movie_card(movie)
        if self.print_in_terminal:
            self.print_movie_to_console(movie)

```

```

def create_movie_card(self, movie: Movie):

    movie_item_frame = Frame(self.movie_frame, bg="white", bd=2,
relief="groove")

    movie_item_frame.pack(fill="x", pady=5, padx=10, expand=True)

    poster_image = self.image_storage.get_image(movie.poster_url)

    poster_frame = Frame(movie_item_frame, bg="white")

    poster_frame.pack(side="left", padx=10, pady=10)

    if poster_image:

        poster_label = Label(poster_frame, image=poster_image,
bg="white")

        poster_label.image = poster_image

        poster_label.pack()

    info_frame = Frame(movie_item_frame, bg="white")

    info_frame.pack(side="left", fill="both", expand=True, padx=10,
pady=10)

    name_label = Label(info_frame, text=movie.name, bg="white",
font=("Arial", 12, "bold"))

    name_label.pack(anchor="w")

    rating_frame = Frame(info_frame, bg="white")

    rating_frame.pack(anchor="w")

    Label(rating_frame, text=f"Рейтинг: {movie.rating}", bg="white",
font=("Arial", 12)).pack(side="left")

    Label(rating_frame, text="□", fg="gold", bg="white", font=("Arial",
12)).pack(side="left", padx=5)

    info_text = (

        f"Год: {movie.year}\n"

        f"Жанры: {' '.join(movie.genres)}\n"

```

```

        f"Страны: {' , '.join(movie.countries)}"
    )

    Label(info_frame, text=info_text, bg="white", justify="left",
font=("Arial", 12)).pack(anchor="w")

```

```

def print_movie_to_console(self, movie: Movie):
    color = self.get_color_by_genre(movie.genres)
    print(f"{color}{movie.name}")
    print(f"Рейтинг: {movie.rating}")
    print(f"Год: {movie.year}")
    print(f"Жанры: {' , '.join(movie.genres)}")
    print(f"Страны: {' , '.join(movie.countries)}")
    print("-" * 40)

```

```

def get_color_by_genre(self, genres: List[str]):
    genre_colors = {
        'драма': Fore.RED,
        'комедия': Fore.GREEN,
        'боевик': Fore.YELLOW,
        'фантастика': Fore.BLUE,
        'триллер': Fore.MAGENTA,
        'мелодрама': Fore.CYAN,
        'детектив': Fore.WHITE,
    }

    for genre in genres:
        if genre.lower() in genre_colors:
            return genre_colors[genre.lower()]

    return Fore.WHITE

```

```

def update_movies_display(self, movies: List[Movie]):
    start_idx = (self.page - 1) * 10

```



```

end_idx = self.page * 10
movies_to_show = movies[start_idx:end_idx]

if movies_to_show:
    self.page_label.config(text=f"Номер страницы: {self.page}")
    self.create_movie_cards(movies_to_show)
    return True
else:
    messagebox.showinfo("Информация", "Фильмы закончились или
произошла ошибка.")
    return False

def prev_page(self):
    if self.page > 1:
        self.page -= 1
        self.update_movies_display(self.movies)

def next_page(self):
    self.page += 1
    if not self.update_movies_display(self.movies):
        self.page -= 1

def go_to_page(self):
    try:
        new_page = int(self.page_entry.get()) if
self.page_entry.get().isdigit() else 0
        if 0 < new_page <= (len(self.movies) // 10 + 1):
            self.page = new_page
            self.update_movies_display(self.movies)
            return True
    except:
        pass
    else:

```

```

        messagebox.showwarning("Ошибка", "Некорректный номер
страницы.")

        return False

    except ValueError:

        messagebox.showwarning("Ошибка", "Введите число.")

        return False


def start_search(self):

    key = self.search_entry.get()

    if key:

        if not self.movies_for_search:

            self.movies_for_search = self.movies.copy()

            repo = MovieRepository()

            repo.QuickSort(self.movies_for_search, 0,
len(self.movies_for_search) - 1)

            repo = MovieRepository()

            found_movies = repo.binary_search(self.movies_for_search, key)

            self.create_movie_cards(found_movies)


def end_search(self):

    self.search_entry.delete(0, 'end')

    self.update_movies_display(self.movies)

//test_ui.py

import pytest

from unittest.mock import Mock, patch, MagicMock

from ui.movie_ui import MovieUI

import tkinter as tk

from tkinter import Tk

from domain.models import Movie

@pytest.fixture

def root_window():

```

```

    root = Tk()

    yield root

    root.destroy()

def test_navigation_buttons(root_window):

    ui = MovieUI(root_window)

    nav_frame = [w for w in root_window.wininfo_children() if isinstance(w,
tk.Frame)][0]

    buttons = [w for w in nav_frame.wininfo_children() if isinstance(w,
tk.Button)]

    assert len(buttons) >= 4, "Navigation buttons are missing"


def test_scrollable_area(root_window):

    ui = MovieUI(root_window)

    canvas = [w for w in root_window.wininfo_children() if isinstance(w,
tk.Canvas)]

    assert canvas, "Canvas for scrollable area is missing"


def test_scrollbar(root_window):

    ui = MovieUI(root_window)

    assert any(isinstance(widget, tk.Scrollbar) for widget in
root_window.wininfo_children()), "Scrollbar is missing"


def test_movie_frame(root_window):

    ui = MovieUI(root_window)

    canvas = [w for w in root_window.wininfo_children() if isinstance(w,
tk.Canvas)]

    assert canvas, "Movie frame is missing"


@patch('ui.movie_ui.ImageStorage')

@patch('ui.movie_ui.Label')

def test_load_image_from_url(mock_label, mock_imagestorage, root_window):

```

```

mock_image = MagicMock()
mock_imagestorage.return_value.get_image.return_value = mock_image

ui = MovieUI(root_window)
test_movie = Movie(
    id="1",
    name="Test Movie",
    year=2020,
    rating=8.0,
    genres=["Drama"],
    countries=["USA"],
    poster_url="test_url"
)

ui.create_movie_card(test_movie)

mock_imagestorage.return_value.get_image.assert_called_once_with("test_url")
mock_label.assert_called()

//main.py
import tkinter as tk
from tkinter import messagebox
from ui.movie_ui import MovieUI
from infrastructure.repository.movie_repository import MovieRepository

def main():
    root = tk.Tk()
    repo = MovieRepository()
    movies = repo.get_top250()
    if not movies:

```

```
        messagebox.showwarning("Ошибка", "Не удалось загрузить данные.")
    return

app = MovieUI(root)
app.movies = movies
app.movies_for_search = None
app.update_movies_display(movies)

root.mainloop()

if __name__ == "__main__":
    main()
```

4. Результаты тестирования

1. Консольный вывод данных

1+1

Рейтинг: 8.846

Год: 2011

Жанры: драма, комедия

Страны: Франция

Джентльмены

Рейтинг: 8.626

Год: 2019

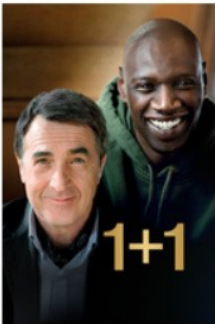
Жанры: криминал, комедия, боевик

Страны: США, Великобритания, Франция, Япония, Чехия


2. Графический вывод (первая страница)

Кинопоиск: Топ 250 фильмов


← Предыдущая Следующая → Номер страницы: 1



1+1
Рейтинг: 8.846 ★
Год: 2011
Жанры: драма, комедия
Страны: Франция



Джентльмены
Рейтинг: 8.626 ★
Год: 2019
Жанры: криминал, комедия, боевик
Страны: США, Великобритания, Франция, Япония, Чехия




Брат
Рейтинг: 8.338 ★
Год: 1997
Жанры: драма, криминал, боевик
Страны: Россия


3. Переход на конкретную страницу

Кинопоиск: Топ 250 фильмов


← Предыдущая Следующая → 11 Перейти Номер страницы: 11 Поиск Окончить поиск



Ворошиловский стрелок
Рейтинг: 7.947 ★
Год: 1999
Жанры: драма, криминал
Страны: Россия



Король говорит!
Рейтинг: 8.046 ★
Год: 2010
Жанры: драма, биография, история
Страны: Великобритания, США




12 лет рабства
Рейтинг: 7.893 ★
Год: 2013
Жанры: драма, биография, история
Страны: Великобритания, США

4. Поиск фильма по названию

Кинопоиск: Топ 250 фильмов

← Предыдущая Следующая → Перейти Номер страницы: 11 Поиск Окончить поиск Побег



Побег из Шоушенка
Рейтинг: 9.109 ★
Год: 1994
Жанры: драма
Страны: США

5. Список используемых источников

1. Руководство по языку программирования Python: сайт – URL: <https://metanit.com/python/tutorial/> (дата обращения: 15.02.2025)
2. Объектно-ориентированное программирование на Python: сайт – URL: <https://metanit.com/python/tutorial/7.1.php/> (дата обращения: 18.02.2025)
3. Словари в языке Python: сайт – URL: <https://metanit.com/python/tutorial/3.3.php/> (дата обращения: 25.02.2025)
4. Python Testing с pytest. Начало работы с pytest: сайт – URL: <https://habr.com/ru/articles/448782/> (дата обращения: 1.03.2025)
5. functools — Higher-order functions and operations on callable objects: сайт – URL: <https://docs.python.org/3/library/functools.html> (дата обращения: 9.03.2025)
6. Руководство по Tkinter: сайт – URL: <https://metanit.com/python/tkinter/1> (дата обращения: 15.03.2025)
7. Библиотека requests сайт – URL: <https://education.yandex.ru/handbook/python/article/module-requests> (дата обращения: 28.03.2025)