OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Vladyslav Keidaliuk

IMDB SOFTWARE OF HOLLYWOOD ACTORS AND ACTRESSES. TASK 3

Faculty:	Informatik und Mathematik
Degree:	Computer Science
Submission deadline:	10.01.2024
Supervisor/examiner:	Istvan Lengyel

Table of Contents

Introduction

This report provides a detailed overview of the finalized version of the project, focusing on the functionality, structure, and implementation of the chosen solution. The accompanying source code and this report demonstrate the completed project, highlighting the modules, data structures, and tools utilized during development.

The project addresses the task of extracting and presenting information about the top 50 Hollywood actors and actresses. The final implementation includes a modular design for efficient data scraping, storage, and display through a graphical user interface (GUI).

Description of the project and the problems that have been addressed in it

The project is a comprehensive software solution designed to extract, organize, and present information about the top 50 Hollywood actors and actresses from IMDb. By leveraging web scraping and a graphical user interface (GUI), the software offers users an interactive experience to explore detailed actor profiles, movie lists, awards and ratings.

The project combines dynamic data retrieval from IMDb with local data storage to provide a seamless and efficient user experience. The graphical interface ensures that even non-technical users can access and analyze actor-related data easily.

Key Features:

- List of all available actors and actresses
- Biography the actor/actresses
- All time movie names, years, ratings, genres
- Awards to actor/actresses in different years (sorting and grouping is present)
- Movie genres of actor/actresses
- Average rating of their movies (overall and each year)
- Top 5 movies, their respective years and genre

Problems Adressed

1. Efficient Data Collection:

    - ➢ Problem: Gathering structured data about actors and their movies from a web source like IMDb can be time-consuming and error-prone.

- Solution: The software automates this process using Selenium, Selenium WebDriver to dynamically scrape actor details, including names, biographies, movies, awards, and ratings.

2. Data Storage and Reusability:

   - Problem: Repeatedly scraping the same data increases processing time and server load, while failing to persist data locally limits usability.
   - Solution: The project stores scraped data in a JSON file, allowing reuse and faster access to information without additional scraping.

3. Data Presentation:

   - Problem: Presenting large amounts of actor-related data in an intuitive way can be challenging, especially for users unfamiliar with technical tools.
   - Solution: A user-friendly GUI, built with Tkinter, displays actor lists, biographies, and movies in an organized and visually appealing manner. The GUI includes features like buttons, scrollable lists, and interactive windows for detailed exploration.

Module, data structures, tools used in the project

Data was collected from the IMDb website and stored in a JSON file format. The data includes detailed information about actors, such as their names, photos (saved locally), biographies, and a list of movies with attributes like title, release year, genres, ratings, and poster URLs.

Modules Used

The project utilizes a variety of Python modules and libraries, categorized as follows:

Core Python Modules:

- os: Handles file system operations, such as checking file existence and managing file paths.
- json: Facilitates reading and writing structured data in JSON format.
- re: Provides support for regular expressions, used for data validation and parsing.
- time and datetime: Used for time-related calculations, such as age determination.

Third-Party Libraries:

- Selenium: A browser automation tool used for web scraping.
    - selenium.webdriver: Provides WebDriver interfaces to control the browser.
    - selenium.webdriver.support.ui.WebDriverWait: Manages waiting conditions for web elements.
    - selenium.webdriver.support.expected_conditions: Defines conditions to synchronize browser interactions.
    - selenium.webdriver.common.by.By: Locates elements using different strategies, such as XPATH or CSS Selectors.
- Pillow (PIL): A Python Imaging Library used for handling images.
- Image and ImageTk: Used to process and display images in the GUI.
- Requests: Facilitates HTTP requests for downloading images and handling API calls.

Custom Helper Functions:

- helper.py: Contains reusable utility functions:
    - calculate_age: Computes actor age based on their date of birth.
    - safe_convert_rating: Safely convert the rating to a number.
    - get_unique_sorted_genres: Extracts and sorts unique genres from movie data.
    - group_movies_by_year: Groups movies by their release years for presentation.
    - fetch_movie_images: Downloads and processes movie poster images.

Graphical User Interface:

- tkinter: Provides a framework for creating the GUI.
- tk and ttk: Used for creating widgets and interactive elements like buttons, labels, and scrollbars.

The project was developed using **JetBrains PyCharm** as the primary coding environment. Web scraping was performed with the **Selenium** library to interact with the IMDb website dynamically. For data management, Python's **JSON** module was used to process and store the scraped information in structured formats such as dictionaries and lists.

To ensure a clear and maintainable structure, the project follows the **Page Object Model** design pattern. This approach organizes the scraping logic into distinct classes and methods, facilitating scalability and readability in the codebase.

Data Structures

- Dictionaries: Dictionaries are extensively used throughout the project to store and organize information. Each dictionary represents a single entity, such as an actor, movie, or award, with key-value pairs for different attributes.
- Lists : Used to store collections of dictionaries, such as:
  - The list of actors scraped from IMDb.
  - A list of movies for a specific actor.
  - A list of unique genres sorted alphabetically.
- Sets: Used for efficient operations like:
  - Eliminating Duplicates: To ensure that genres, actor names, or awards are not repeated.
  - Processed Actors: A set is used to track actors whose data has already been scraped.

Tools

- Selenium WebDriver:
  - Used for browser automation to dynamically scrape data from IMDb.
  - Enabled interaction with web elements like buttons, links, and text fields to extract actor and movie information.
- Chrome DevTools:
  - Used to inspect and identify web element locators (e.g., XPath, CSS selectors) for Selenium scripts.
  - Assisted in troubleshooting and verifying web scraping operations.
- JetBrains PyCharm:
  - The primary Integrated Development Environment (IDE) used for writing, debugging, and maintaining the project's code.

Design of the project

The project is designed using a modular approach to ensure scalability, maintainability, and clear communication between components. It employs the **Page Object Model** (POM) for web scraping and a structured flow for data processing and presentation. Here’s how the modules interact:

- BasePage:

        - Serves as the foundation for all page-specific classes, providing shared methods for element interactions (e.g., finding elements, clicking, and scrolling).
        - Enables consistent and reusable logic across other modules.
- ➢ Page-Specific Classes:
    - ActorsListPage: Extracts the list of actors and their profile links from the main IMDb page. It navigates to individual actor pages when needed.
    - ActorPage: Retrieves detailed actor data, including their name, date of birth, biography link, awards link, and movie list. It also navigates to sub-pages (biography and awards).
    - BiographyPage: Extracts the actor’s biography text.
    - AwardsPage: Collects award details, including award names, years, and associated movies.
- ➢ Web Scraping Module:
    - Collects data dynamically from IMDb using Selenium WebDriver and organizes it into structured dictionaries and lists.
    - Stores the scraped data in a JSON file (actors_data.json) for reuse and faster access.
- ➢ Data Processing Module:
    - Performs tasks such as calculating actor ages, grouping movies by year, extracting unique genres, and safely converting ratings.

GUI Module:

- ➢ Displays the processed data in an interactive, user-friendly graphical interface created with Tkinter.
- ➢ Enables users to:
    - Browse the list of actors and their profiles.
    - View detailed actor biographies.
    - Explore movies with titles, years, ratings, and genres.
- ➢ Integrates images (actor photos and movie posters) using the Pillow library for enhanced visual representation.

Module Communication

Web Scraping → Data Storage:

The web scraping module saves data into a JSON file, ensuring reusability and preventing redundant scraping.

Data Storage → Data Processing:

The data processing module reads the JSON file and transforms the raw data for easy display in the GUI.

Data Processing → GUI:

The processed data is dynamically loaded into the GUI, populating actor lists, biographies, and movie details.

User Interaction → GUI → Data Processing:

User actions in the GUI (e.g., selecting an actor or viewing movies) trigger data retrieval and additional processing if needed.

Pseudocode of all problems addressed in the project

Pseudocode for Web Scraping Functionality

Main Scraping Program Flow:

```
IMPORT necessary libraries and modules

DEFINE function scrape_data(max_retries=3)
    INITIALIZE retries to 0

    WHILE retries < max_retries
        TRY
            SET home_page_url to 'https://www.imdb.com/list/ls053501318/'
            CONFIGURE Chrome options with English language
            INITIALIZE Chrome WebDriver with the configured options
            OPEN home page URL in the browser

            CREATE ActorsListPage instance
            GET list of actors from the page
            ACCEPT cookies on the page

            IF actors_data.json exists
                LOAD actors data from file
            ELSE
                INITIALIZE empty actors data list
```

```
CREATE a set of processed actors (actors already scraped)

FOR each actor in actors
    IF actor is already processed
        SKIP to next actor

    PRINT "START scraping for [actor name]"
    CLICK on actor's name to open actor profile page

    GET actor's photo element and download image

    GET actor's date of birth
    OPEN biography page

    GET actor's biography
    GO back to actor profile page

    OPEN awards page
    GET actor's awards
    GO back to actor profile page

    CREATE actor_info dictionary with:
        - name
        - photo path
        - date of birth
        - biography
        - awards
        - empty list for movies

    EXPAND list of movies
    GET total number of movies for the actor

    FOR each movie index (1 to number_of_movies)
        GET movie data (title, year, genres, rating)
        ADD movie data to actor_info

    PRINT "Movies get successfully"
    APPEND actor_info to actors_data

    SAVE actors_data to actors_data.json

    PRINT "END scraping for [actor name]"

    GO back to home page

QUIT the driver after all actors are processed
```

```
            BREAK the loop

        CATCH exception
            INCREMENT retries by 1
            PRINT error message with attempt number
            QUIT the driver

            IF retries >= max_retries
                PRINT "Max retries reached. Exiting..."
                BREAK the loop
```

ActorsListPage:

```
CLASS ActorsListPage:
  FUNCTION get_actors():
      CALL self.find_elements(locator)
      RETURN list of actor names and links

  FUNCTION click_accept_cookies():
      CALL self.click_on(locator)

  FUNCTION click_by_title(actor_name):
      CALL self.click_on(title_with_text(actor_name))
```

ActorPage:

```
CLASS ActorPage extends BasePage:
    DEFINE constants for element locators (e.g., PHOTO, ABOUT, MOVIES, etc.)

    FUNCTION actor_img(text):
        RETURN XPath for actor's image based on the given text

    FUNCTION get_photo_url():
        FIND element PHOTO and return its 'src' attribute

    FUNCTION get_actor_name():
        FIND and return actor's name from the page

    FUNCTION get_about():
        FIND and return actor's biography text

    FUNCTION get_movies():
        FIND movie elements, extract title and year, RETURN list of movies
    FUNCTION open_bio_page():
        CLICK on biography page link

    FUNCTION get_date_of_birth():
        FIND and return actor's date of birth

    FUNCTION get_count_of_movies():
        TRY to get count of actor's movies, if not found, get count for actress

    FUNCTION expand_list_of_movies():
        TRY to click "SEE ALL" button for actor or actress movies, if not present
click "PREVIOUS PROJECTS"

    FUNCTION click_on_movie_by_index(index):
        CLICK on movie block by index (actor or actress)

    FUNCTION click_on_close_btn_movie_block():
        CLICK on close button of movie block

    FUNCTION get_rating_of_movie():
        TRY to get movie rating, if not available return "Not rated"

    FUNCTION get_movie_data_by_index(index):
        GET movie image, title, year, genres, rating
        RETURN movie data dictionary

    FUNCTION open_awards_page():
        CLICK on awards page link
```

BiographyPage:

```
CLASS BiographyPage:
  FUNCTION get_biography():
      CALL self.find_element(locator)
      RETURN text of biography element
```

BasePage:

```
CLASS BasePage:
    FUNCTION __init__(driver):
        SET self.driver = driver

    FUNCTION find_element(locator):
        WAIT for element located by locator to be visible
        RETURN the element

    FUNCTION find_elements(locator):
        WAIT for all elements located by locator to be present
        RETURN the elements

    FUNCTION is_element_present(locator):
        FIND all elements located by locator
        RETURN True if elements are found, else False

    FUNCTION navigate_to(url):
        OPEN the given URL in the browser

    FUNCTION scroll_to_element(locator):
        FIND the element located by locator
        SCROLL the page to the element

    FUNCTION click_on(locator):
        SCROLL to the element located by locator
        FIND the element located by locator
        CLICK on the element

    FUNCTION try_click(locator, retries=3, delay=2):
        FOR each attempt from 1 to retries:
            TRY:
                IF attempt > 1:
                    PRINT attempt number and locator
                SCROLL to the element located by locator
                WAIT for element to be clickable
                CLICK on the element
                IF attempt > 1:
                    PRINT success message
                RETURN
            EXCEPT exception:
                PRINT attempt failure message
                IF attempt < retries:
                    WAIT for 'delay' seconds
                ELSE:
                    PRINT failure message and raise exception
```

Pseudocode for Tkinter-based GUI Functionality

This subsection describes the pseudocode for the Tkinter-based GUI functionality. It explains how the actor data (scraped in the previous subsection) is presented in the GUI, how actor biography, movie information, awards, top 5 movies are displayed in different windows, and how the user can interact with the interface.

```
START
  LOAD actor data from "actors_data.json"

  CREATE main window
  SET window title to "Vladyslav Keidaliuk - Project 2. Hollywood Actors and
Actresses"

  CREATE scrollable frame for actor buttons
  CALL show_actors()

  FUNCTION show_actors():
      SET columns = 5
      SET row = 0
      SET col = 0

      FOR each actor in actor_data:
          CREATE button for actor with name and photo
          SET button action to show_actor_info(actor)
          ADD button to grid in main window

          IF col >= columns:
              RESET col to 0
              INCREMENT row by 1
          ELSE:
              INCREMENT col by 1

  FUNCTION show_actor_info(actor_name):
      SET actor = find_actor_in_data(actor_name)

      IF actor is not found:
          RETURN

      CREATE new window for actor
      DISPLAY actor name and date of birth
      DISPLAY actor biography in text box

      CREATE buttons for viewing:
          - Movies
          - Awards
          - Movie Genres
          - Average Rating
          - Top 5 Movies
      SET button actions for each category
```

```
FUNCTION show_actor_movies(actor_name):
    SET actor = find_actor_in_data(actor_name)
    IF actor has no movies:
        RETURN

    CREATE new window for loading movies
    CREATE progress bar to show movie loading progress

    LOAD movie images for actor
    UPDATE progress bar while loading

    DISPLAY list of actor movies after loading

FUNCTION display_movies(actor_name, movies):
    CREATE new window for displaying actor's movies
    CREATE scrollable frame for movie list

    FOR each movie in movies:
        CREATE frame for movie
        ADD movie title, year, rating, and genres
        ADD movie image (if available)

FUNCTION show_actor_awards(actor_name):
    SET actor = find_actor_in_data(actor_name)
    IF actor has no awards:
        RETURN

    CREATE new window for displaying actor's awards
    CREATE sorting and grouping options for awards
    UPDATE awards list based on selected criteria

    FOR each award in awards:
        CREATE frame for award
        ADD award name, status, year, and category

FUNCTION show_actor_movie_genres(actor_name):
    SET actor = find_actor_in_data(actor_name)
    IF actor has no movies:
        RETURN

    GET unique sorted movie genres for actor

    CREATE new window for displaying genres
    DISPLAY list of genres

FUNCTION show_actor_average_rating(actor_name):
      SET actor = find_actor_in_data(actor_name)
      IF actor has no movies:
          RETURN

      CALCULATE average rating for all movies

      CREATE new window for displaying average rating
      DISPLAY overall average and per-year ratings

  FUNCTION show_actor_top_movies(actor_name):
      SET actor = find_actor_in_data(actor_name)
      IF actor has no movies:
          RETURN

      SORT movies by rating
      SELECT top 5 movies

      CREATE new window for displaying top 5 movies
      DISPLAY movie details (title, year, rating, genres)

  FUNCTION find_actor_in_data(actor_name):
      SEARCH for actor in actor_data by name
      RETURN actor if found, otherwise return None
END          END loop

  FUNCTION find_actor_in_data(actor_name):
      FOR actor in actor_data:
          IF actor name matches actor_name:
              RETURN actor

  RUN GUI main loop to keep window open
END
```

List of functions for each problem given in the project

update_canvas()

Purpose: Updates the canvas scroll region when the content changes in the canvas.

Parameters: event (tkinter event object).

Sample Output: None.

show_actor_movies(actor_name, actor_data)

Purpose: Opens a window with a list of the actor's movies and a progress bar to display loading status.

Parameters: actor_name (str), actor_data (list of dicts).

Sample Output: None.

display_movies(actor_name, movies)

Purpose: Displays a window with a list of the actor's movies after they have been loaded.

Parameters: actor_name (str), movies (list of dicts).

Sample Output: None.

show_actor_info(actor_name, actor_data)

Purpose: Opens a window with detailed information about the selected actor.

Parameters: actor_name (str), actor_data (list of dicts).

Sample Output: None.

create_actor_button(actor_name, actor, parent_frame)

Purpose: Creates and returns a button for an actor with a photo and name.

Parameters: actor_name (str), actor (dict), parent_frame (tkinter Frame).

Sample Output: Returns a tkinter Button element or None if no photo is found.

show_actors()

Purpose: Displays a grid of actor buttons on the main window.

Parameters: None.

Sample Output: None.

show_actor_awards(actor_name, actor_data)

Purpose: Opens a window with a list of awards the actor has won, with sorting and grouping options.

Parameters: actor_name (str), actor_data (list of dicts).

Sample Output: None.

show_actor_movie_genres(actor_name, actor_data)

Purpose: Opens a window displaying all movie genres associated with the actor.

Parameters: actor_name (str), actor_data (list of dicts).

Sample Output: None.

show_actor_average_rating(actor_name, actor_data)

Purpose: Opens a window displaying the actor's average movie rating, with year-wise breakdown.

Parameters: actor_name (str), actor_data (list of dicts).

Sample Output: None.

show_actor_top_movies(actor_name, actor_data)

Purpose: Opens a window displaying the top 5 movies of the actor based on their ratings.

Parameters: actor_name (str), actor_data (list of dicts).

Sample Output: None.

calculate_age(birth_date)

Purpose: Calculates the age of a person based on their birth date.

Parameters: birth_date (str) - The birth date in "Month day, Year" format.

Sample Output: The age (int).

parse_award_info(input_string)

Purpose: Parses a string containing award information and returns the parsed details.

Parameters: input_string (str) - A string in the format '2000 Winner SFFCC Award'.

Sample Output: A tuple containing (year, award_status, award_name) or raises a ValueError if the string doesn't match the expected format.

download_img(image_element, act_name)

Purpose: Downloads an image from a URL provided by the image element and saves it to a local directory.

Parameters: image_element (WebElement) - The image element from which the URL is extracted.

act_name (str) - The name of the actor used to generate the image file name.

Sample Output: The path where the image is saved.

safe_convert_rating(rating)

Purpose: Attempts to convert a rating to a float. If unsuccessful, returns 0.

Parameters: rating (str or float) - The rating to be converted.

Sample Output: The converted rating (float) or 0 if the conversion fails.

get_unique_sorted_genres(actor)

Purpose: Returns a sorted list of unique genres from the actor's movie list.

Parameters: actor (dict) - A dictionary containing the actor's data, including a list of movies.

Sample Output: A sorted list of unique genres (list of strings).

group_movies_by_year(movies)

Purpose: Groups movies by their release year and stores the ratings for each year.

Parameters: movies (list) - A list of movie dictionaries containing year and rating information.

Sample Output: A dictionary where the keys are years (str) and the values are lists of ratings (float).

fetch_movie_images(movies, progress_callback)

Purpose: Downloads and resizes movie images, calling the callback function to update the progress.

Parameters: movies (list) - A list of movie dictionaries, each containing an image link.

progress_callback (function) - A callback function to update the progress during image loading.

Sample Output: A list of movies with the "image" field updated with the image data.

navigate_to(self, url)

Purpose: Navigates to the given URL in the browser.

Parameters: url (str) - The URL to navigate to.

Sample Output: None.

scroll_to_element(self, locator)

Purpose: Scrolls the page to the specified element.

Parameters: locator (tuple) - The locator of the element to scroll to.

Sample Output: None.

click_on(self, locator)

Purpose: Scrolls to the element and clicks on it.

Parameters: locator (tuple) - The locator of the element to click on.

Sample Output: None.

try_click(self, locator, retries=3, delay=2)

Purpose: Tries to click on an element, with retries if it fails.

Parameters:

locator (tuple) - The locator for the element.

retries (int) - Number of retry attempts.

delay (int) - Delay between retries in seconds.

Sample Output: None (raises an exception if all attempts fail).

actor_img(text)

Purpose: Generates a locator for the actor's image based on the provided text.

Parameters: text (str) - The text to match in the image's alt attribute.

Sample Output: By.XPATH, a locator for the actor's image.

actor_movie_image_by_index(self, index)

Purpose: Returns a locator for the movie image of an actor at the specified index.

Parameters: index (int) - The index of the movie.

Sample Output: By.XPATH, a locator for the actor's movie image.

get_movie_image_by_index(self, index)

Purpose: Retrieves the movie image URL of an actor or actress at the specified index.

Parameters: index (int) - The index of the movie.

Sample Output: URL of the movie image (str), or "No image" if not found.

get_photo_url(self)

Purpose: Retrieves the URL of the actor's photo.

Sample Output: URL of the photo (str).

get_movie_data_by_index(self, index)

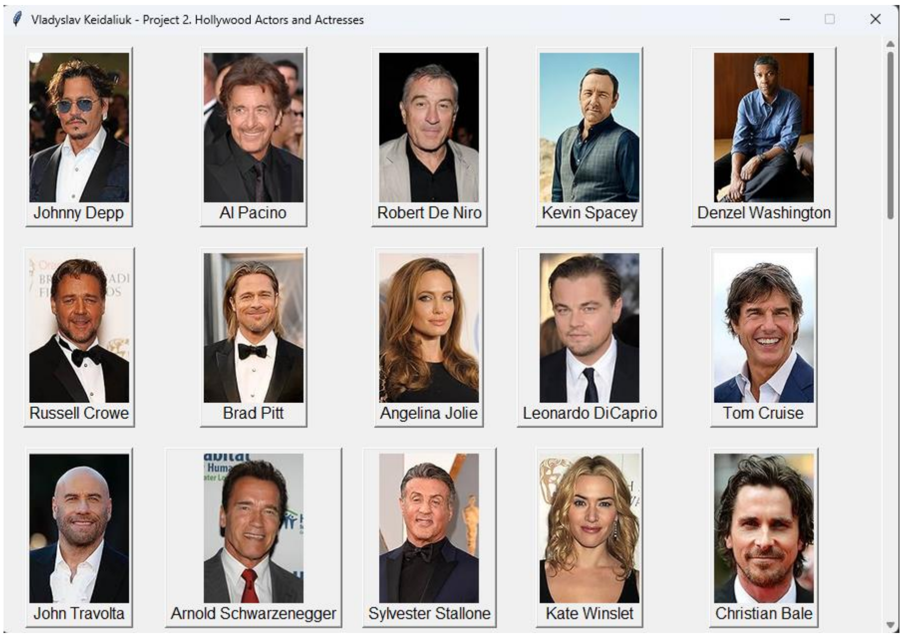
Purpose: Retrieves detailed movie data (title, year, genres, image, rating) for a movie at the specified index.

Parameters: index (int) - The index of the movie.

Sample Output: Movie data (dictionary with title, year, image, rating, genres).

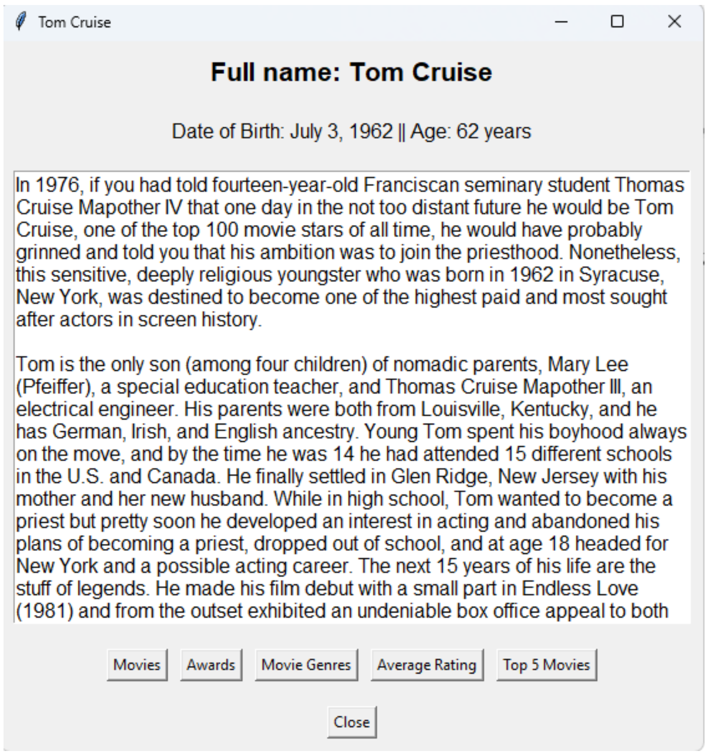
Solution screenshots

Graphical User Interface



Picture 1. Main Page of application

After clicking on a particular actor/actress (e.g. Tom Cruise), a page will open with their biography and the ability to go to a list of movies, list of awards, movie genres, average rating and top 5 movies.



Full name: Tom Cruise

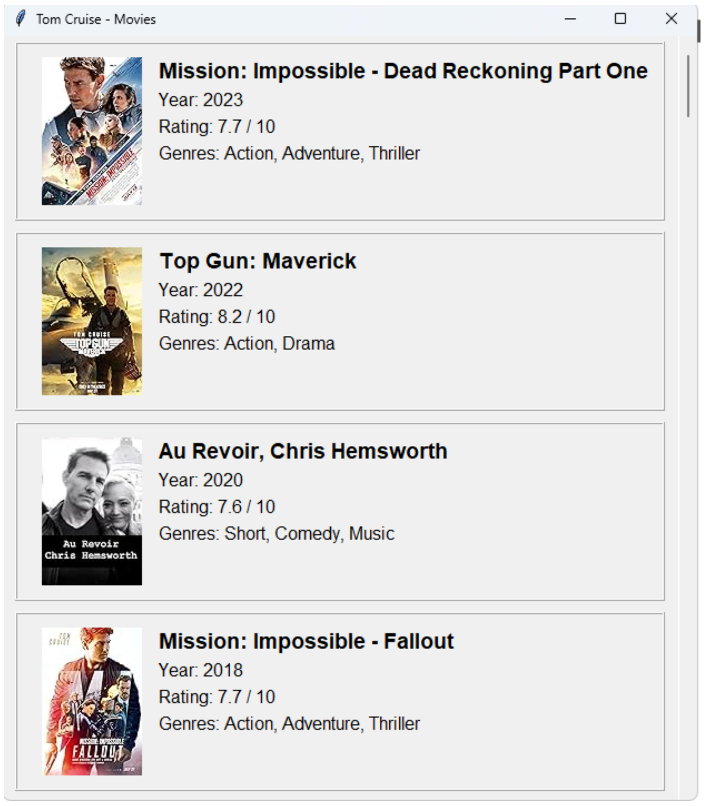
Date of Birth: July 3, 1962 || Age: 62 years

In 1976, if you had told fourteen-year-old Franciscan seminary student Thomas Cruise Mapother IV that one day in the not too distant future he would be Tom Cruise, one of the top 100 movie stars of all time, he would have probably grinned and told you that his ambition was to join the priesthood. Nonetheless, this sensitive, deeply religious youngster who was born in 1962 in Syracuse, New York, was destined to become one of the highest paid and most sought after actors in screen history.

Tom is the only son (among four children) of nomadic parents, Mary Lee (Pfeiffer), a special education teacher, and Thomas Cruise Mapother III, an electrical engineer. His parents were both from Louisville, Kentucky, and he has German, Irish, and English ancestry. Young Tom spent his boyhood always on the move, and by the time he was 14 he had attended 15 different schools in the U.S. and Canada. He finally settled in Glen Ridge, New Jersey with his mother and her new husband. While in high school, Tom wanted to become a priest but pretty soon he developed an interest in acting and abandoned his plans of becoming a priest, dropped out of school, and at age 18 headed for New York and a possible acting career. The next 15 years of his life are the stuff of legends. He made his film debut with a small part in Endless Love (1981) and from the outset exhibited an undeniable box office appeal to both

Picture 2. Information Page about actor/actress

If you click on the “Movies” button, a list of movies will be displayed.



Mission: Impossible - Dead Reckoning Part One
Year: 2023
Rating: 7.7 / 10
Genres: Action, Adventure, Thriller

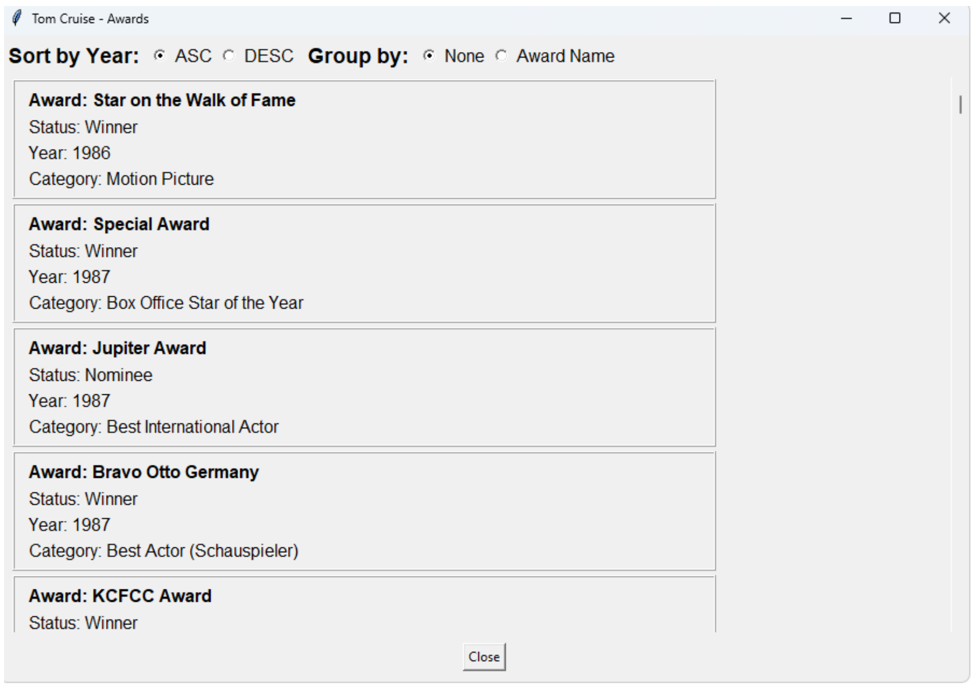
Top Gun: Maverick
Year: 2022
Rating: 8.2 / 10
Genres: Action, Drama

Au Revoir, Chris Hemsworth
Year: 2020
Rating: 7.6 / 10
Genres: Short, Comedy, Music

Mission: Impossible - Fallout
Year: 2018
Rating: 7.7 / 10
Genres: Action, Adventure, Thriller

Picture 3. Page with list of movies

If you click on the “Awards” button, a list of awards will be displayed.



Sort by Year: [✓ ASC] DESC **Group by:** [✓ None] Award Name

Award: Star on the Walk of Fame
Status: Winner
Year: 1986
Category: Motion Picture

Award: Special Award
Status: Winner
Year: 1987
Category: Box Office Star of the Year

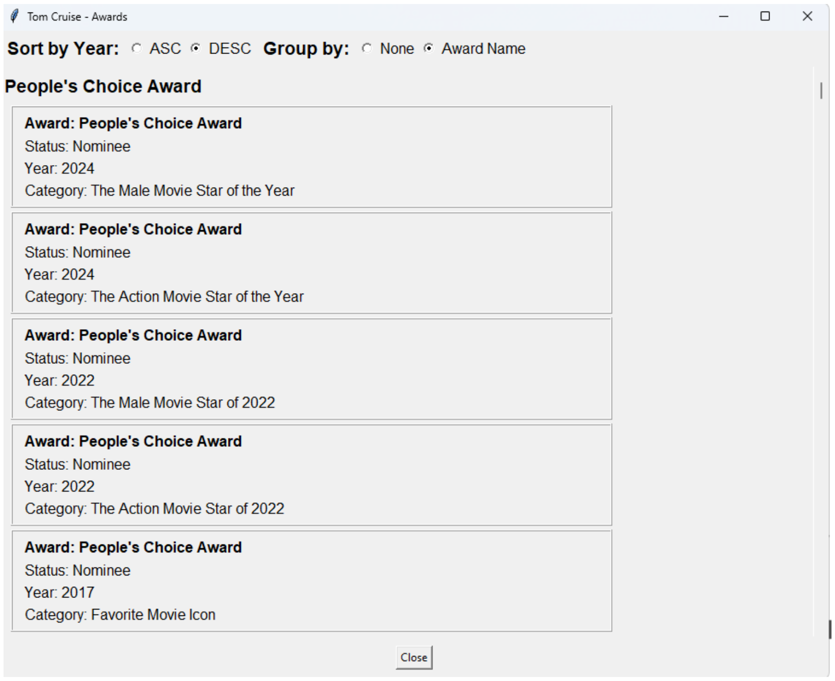
Award: Jupiter Award
Status: Nominee
Year: 1987
Category: Best International Actor

Award: Bravo Otto Germany
Status: Winner
Year: 1987
Category: Best Actor (Schauspieler)

Award: KCFCC Award
Status: Winner

Picture 4. Page with list of awards

If you click on the radio buttons “Desc” and “Award Name”, the appropriate sorting and grouping will apply


Tom Cruise - Awards

Sort by Year: ASC [✓ DESC] Group by: None [✓ Award Name]

People's Choice Award

Award: People's Choice Award
Status: Nominee
Year: 2024
Category: The Male Movie Star of the Year

Award: People's Choice Award
Status: Nominee
Year: 2024
Category: The Action Movie Star of the Year

Award: People's Choice Award
Status: Nominee
Year: 2022
Category: The Male Movie Star of 2022

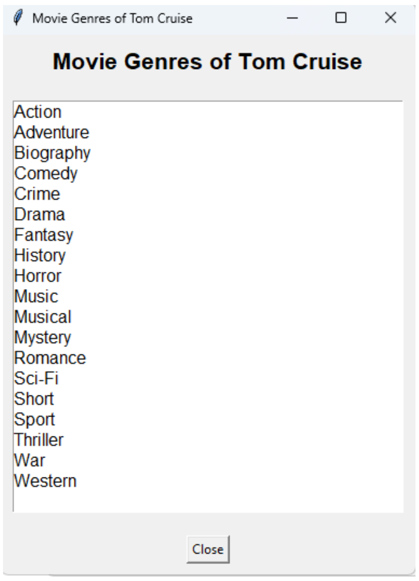
Award: People's Choice Award
Status: Nominee
Year: 2022
Category: The Action Movie Star of 2022

Award: People's Choice Award
Status: Nominee
Year: 2017
Category: Favorite Movie Icon

Close

Picture 5. Page with list of awards with applied sorting and grouping

If you click on the “Movie Genres” button, a list of unique genres for concrete actor/actress will be displayed.


Movie Genres of Tom Cruise

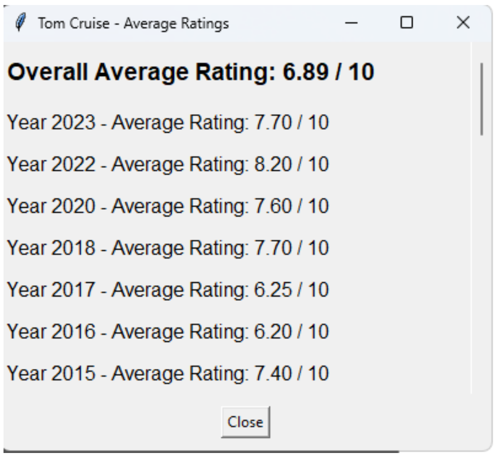
Movie Genres of Tom Cruise

Action
Adventure
Biography
Comedy
Crime
Drama
Fantasy
History
Horror
Music
Musical
Mystery
Romance
Sci-Fi
Short
Sport
Thriller
War
Western

Close

Picture 6. Page with list of unique genres for concrete actor/actress

If you click on the “Average Rating” button, a list of average rating overall and each year for concrete actor/actress will be displayed.


Tom Cruise - Average Ratings

Overall Average Rating: 6.89 / 10

Year 2023 - Average Rating: 7.70 / 10

Year 2022 - Average Rating: 8.20 / 10

Year 2020 - Average Rating: 7.60 / 10

Year 2018 - Average Rating: 7.70 / 10

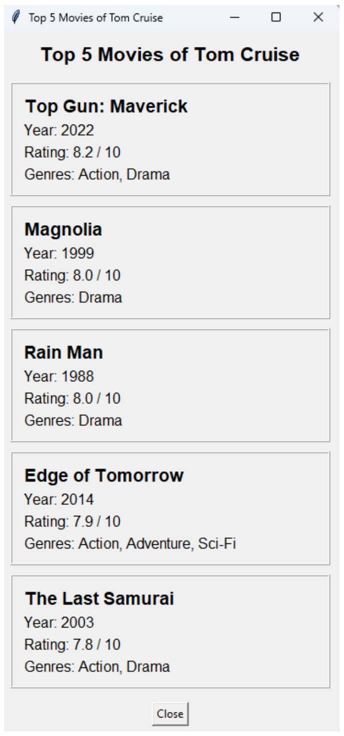
Year 2017 - Average Rating: 6.25 / 10

Year 2016 - Average Rating: 6.20 / 10

Year 2015 - Average Rating: 7.40 / 10

Picture 7. Page with list of average rating overall and each year

If you click on the “Top 5 Movies” button, a list of top 5 movies for concrete actor/actress will be displayed.


Top 5 Movies of Tom Cruise

Top 5 Movies of Tom Cruise

Top Gun: Maverick
Year: 2022
Rating: 8.2 / 10
Genres: Action, Drama

Magnolia
Year: 1999
Rating: 8.0 / 10
Genres: Drama

Rain Man
Year: 1988
Rating: 8.0 / 10
Genres: Drama

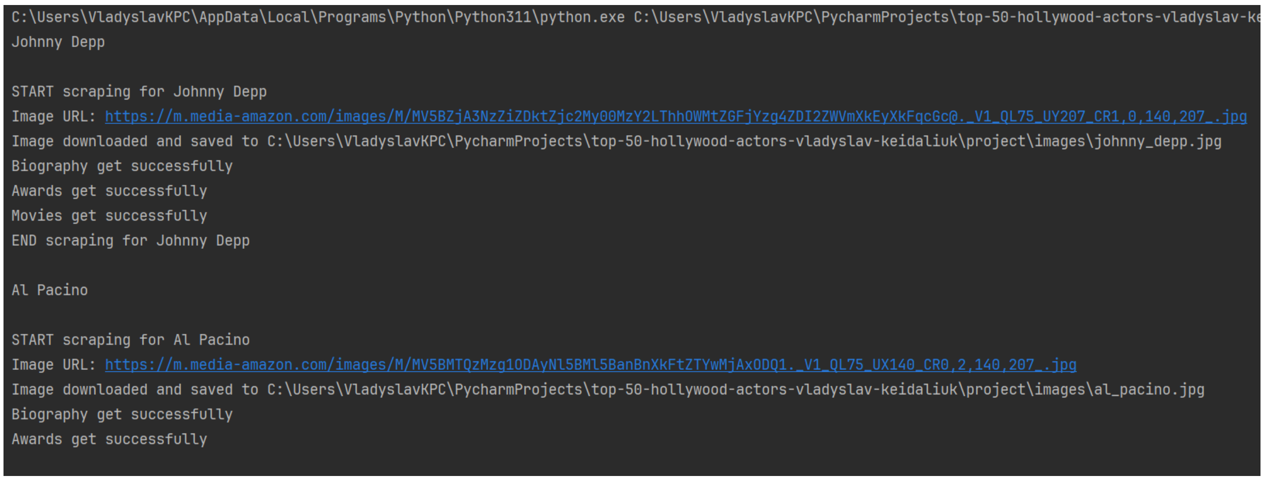
Edge of Tomorrow
Year: 2014
Rating: 7.9 / 10
Genres: Action, Adventure, Sci-Fi

The Last Samurai
Year: 2003
Rating: 7.8 / 10
Genres: Action, Drama

Picture 8. Page with list of top 5 movies for concrete actor/actress

Command Line Interface

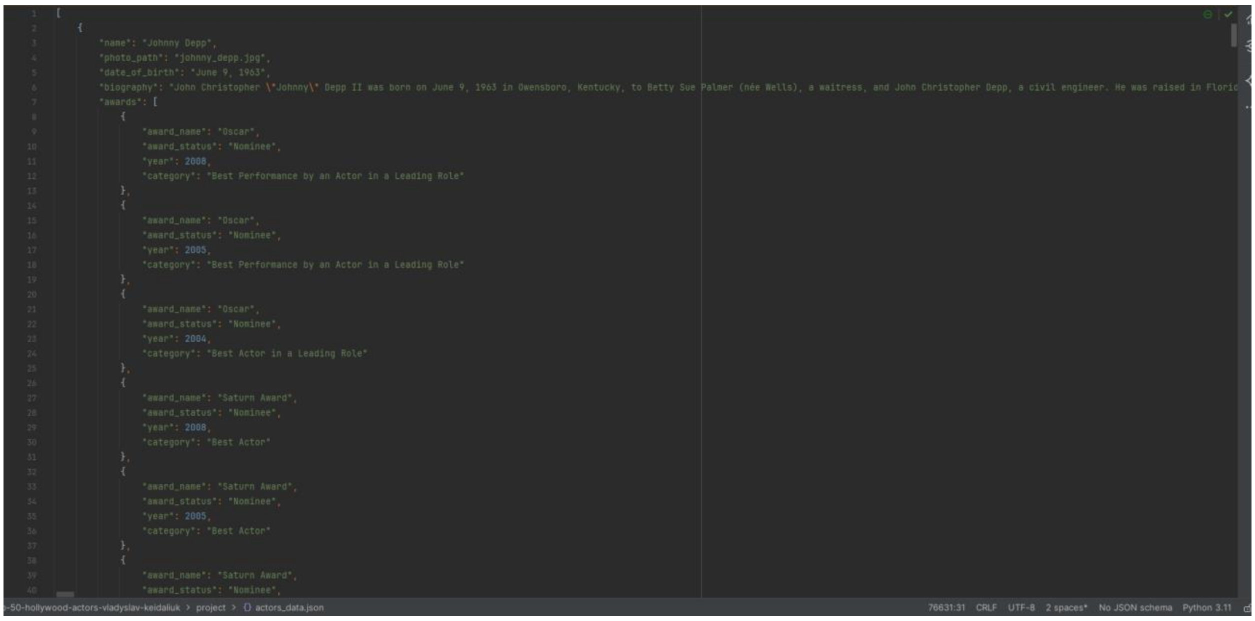
This user interface is intended when data scraping is used (scrape_all_data.py). Rather, in this case, the log output function is being performed.



```
C:\Users\VladyslavKPC\AppData\Local\Programs\Python\Python311\python.exe C:\Users\VladyslavKPC\PycharmProjects\top-50-hollywood-actors-vladyslav-ke
Johnny Depp

START scraping for Johnny Depp
Image URL: https://m.media-amazon.com/images/M/MV5BZjA3NzZiZDktZjc2My00MzY2LThhOWMtZGFjYzg4ZDI2ZWVmXkEyXkFqcGc@._V1_QL75_UY207_CR1,0,140,207_.jpg
Image downloaded and saved to C:\Users\VladyslavKPC\PycharmProjects\top-50-hollywood-actors-vladyslav-keidaliuk\project\images\johnny_depp.jpg
Biography get successfully
Awards get successfully
Movies get successfully
END scraping for Johnny Depp

Al Pacino

START scraping for Al Pacino
Image URL: https://m.media-amazon.com/images/M/MV5BMTQzMzg1ODAyNl5BMl5BanBnXkFtZTYwMjAxODQ1._V1_QL75_UX140_CR0,2,140,207_.jpg
Image downloaded and saved to C:\Users\VladyslavKPC\PycharmProjects\top-50-hollywood-actors-vladyslav-keidaliuk\project\images\al_pacino.jpg
Biography get successfully
Awards get successfully
```

Picture 9. Example of log output

File Output Screenshots



```
[
    {
        "name": "Johnny Depp",
        "photo_path": "johnny_depp.jpg",
        "date_of_birth": "June 9, 1963",
        "biography": "John Christopher \"Johnny\" Depp II was born on June 9, 1963 in Owensboro, Kentucky, to Betty Sue Palmer (née Wells), a waitress, and John Christopher Depp, a civil engineer. He was raised in Floric
        "awards": [
            {
                "award_name": "Oscar",
                "award_status": "Nominee",
                "year": 2008,
                "category": "Best Performance by an Actor in a Leading Role"
            },
            {
                "award_name": "Oscar",
                "award_status": "Nominee",
                "year": 2005,
                "category": "Best Performance by an Actor in a Leading Role"
            },
            {
                "award_name": "Oscar",
                "award_status": "Nominee",
                "year": 2004,
                "category": "Best Actor in a Leading Role"
            },
            {
                "award_name": "Saturn Award",
                "award_status": "Nominee",
                "year": 2008,
                "category": "Best Actor"
            },
            {
                "award_name": "Saturn Award",
                "award_status": "Nominee",
                "year": 2005,
                "category": "Best Actor"
            },
            {
                "award_name": "Saturn Award",
                "award_status": "Nominee",
```

-50-hollywood-actors-vladyslav-keidaliuk > project > {} actors_data.json
76631:31 CRLF UTF-8 2 spaces* No JSON schema Python 3.11

Picture 10. JSON file with all needed actors/actresses data

Testing Methodology and Evaluation

The testing methodology for this project involved executing the program with different retrieved data and observing the behavior. A major component of the system is **logging**, where detailed logs are written to the console to trace execution steps, making it easier to debug any issues.

To ensure reliability, the program is designed with **try-except** blocks to handle unexpected scenarios. For instance, when attempting to retrieve movie data (like the rating, genres, or year), the program checks for the presence of the data and handles cases where the information may be missing (e.g., if a movie doesn't have a rating, genre, or release year).

Additionally, the program implements a **retry pattern**, especially for situations where elements might not be immediately clickable or visible due to network latency or page load delays. If a failure occurs while attempting to interact with the elements, the program retries the action a specified number of times, with a delay between attempts.

If an error occurs (e.g., incorrect actor/actress names or missing data), the program logs the error in the console, allowing the user to identify the issue. In case the actor/actress cannot be found, an error message is shown, and the program retries the find a specified number of times.

The program has been thoroughly **tested for various edge cases**, including missing data for movies (like "Not rated" or "N/A" genres), and it behaves as expected. The console logs provide useful feedback to the user in case of issues, making it user-friendly and resilient. All retrieved data (such as movie titles, images, ratings, and genres) are displayed clearly and visually, ensuring a smooth user experience

Conclusion

In this report, the finalized version of the project for collecting and displaying information about Hollywood actors and actresses from IMDb was presented. The project uses Selenium WebDriver for web scraping and Tkinter for the graphical user interface. Key features include retrieving actor details, movie information, and handling missing data (e.g., ratings, genres). The system also implements a retry pattern for reliability.

The design focuses on user-friendly navigation, error handling, and efficient data display. Overall, the project successfully provides a robust and interactive platform for exploring actor information.