

STAR INFO (IMDB) SYSTEM

A MINI PROJECT REPORT

Submitted by

YOGESH CV **220701327**

YOGESHWARAN H **220701328**

in partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI-602105

NOV-DEC

2023

BONAFIDE CERTIFICATE

Certified that this project report “**IMDB MANAGEMENT SYSTEM**” is the bonafide work of “**YOGESH CV (220701327), YOGESHWARAN H (220701328)**” who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Dr.R.SABITHA
Professor and II Year Academic Head
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous),
Thandalam, Chennai - 602 105

SIGNATURE

Ms.V.JANANEE
Assistant Professor (SG),
Computer Science and Engineering,
Rajalakshmi Engineering College,
(Autonomous),
Thandalam, Chennai - 602 105

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

The "Star Info" project is a comprehensive database management system designed to provide detailed information about actors from both foreign and Indian cinema. The system offers a user-friendly interface for browsing and exploring biographical details, filmographies, and other relevant information about actors. Users can view an actor's complete profile, including a list of their movies, personal details, and interesting facts.

A key feature of the application is the comment section, where users can leave, update, and delete comments on actor profiles, fostering an interactive and engaging user experience. The project employs a MySQL database to store actor information, movie details, and user comments, ensuring efficient data management and retrieval.

The user interface, developed using Python's Tkinter library, provides seamless navigation between different sections, allowing users to easily switch between browsing foreign and Indian actors. The project also integrates the PIL library for handling and displaying actor images.

Overall, "Star Info" serves as a robust platform for cinephiles and entertainment enthusiasts, offering a rich repository of actor-related information and a dynamic commenting system to enhance user interaction. The project highlights effective database design, user interface development, and interactive functionalities, making it a valuable resource for anyone interested in the world of cinema.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION

1.2 OBJECTIVES

1.3 MODULES

2. SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

2.2 LANGUAGES

2.2.1 SQL

2.2.2 PYTHON

3. REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

3.3 ARCHITECTURE DIAGRAM

3.4 ER DIAGRAM

3.5 NORMALIZATION

4. PROGRAM CODE

5. RESULTS AND DISCUSSION

6.CONCLUSION

7.REFERENCES

CHAPTER 1

1. INTRODUCTION

1.1 INTRODUCTION

The "Star Info" project is an innovative and comprehensive platform designed to provide users with detailed information about actors from both foreign and Indian cinema. Leveraging a robust database management system, the application offers a seamless interface for users to explore biographical details, filmographies, and other interesting facts about their favorite actors. Developed using Python's Tkinter library for the graphical user interface and MySQL for database management, "Star Info" ensures efficient data retrieval and user-friendly navigation. This project aims to serve as a valuable resource for cinephiles and entertainment enthusiasts by delivering a wealth of information and fostering an interactive user experience through a dynamic commenting system.

1.2 OBJECTIVES

The primary objectives of the "Star Info" project are as follows:

Information Repository: To create a comprehensive database that stores detailed information about actors from foreign and Indian cinema, including their biographies, filmographies, and other relevant data.

User Interaction: To provide an interactive platform where users can leave, update, and delete comments on actor profiles, enhancing engagement and user experience.

Seamless Navigation: To develop a user-friendly interface that allows easy navigation between different sections of the application, enabling users to effortlessly switch between browsing foreign and Indian actors.

Efficient Data Management: To utilize a MySQL database for efficient storage and retrieval of actor and movie details, ensuring quick access to information.

Visual Appeal: To integrate the PIL library for handling and displaying actor images, making the interface visually appealing and informative.

1.3 MODULES

The "Star Info" project is divided into several key modules, each responsible for different aspects of the application's functionality:

1.3.1 User Interface Module

Tkinter Integration: This module handles the creation and management of the graphical user interface using Python's Tkinter library.

Navigation: Facilitates seamless navigation between the main page, actor selection pages (foreign and Indian actors), and detailed actor profile pages.

Image Handling: Uses the PIL library to load and display actor images, ensuring they are resized and presented appropriately.

1.3.2 Database Management Module

MySQL Integration: Manages the connection to the MySQL database, including establishing connections, executing queries, and handling data retrieval and updates.

Data Storage: Responsible for storing actor details, movie information, and user comments in the database.

Data Retrieval: Handles the fetching of actor profiles, movie lists, and comments from the database for display in the user interface.

1.3.3 Actor Details Module

Profile Display: Displays detailed information about each actor, including their biography, nationality, and other personal details.

Filmography: Lists the movies associated with each actor, including titles, synopses, IMDb ratings, OTT platforms, release dates, and box office collections.

1.3.4 Commenting System Module

Comment Submission: Allows users to leave comments on actor profiles, storing these comments in the database.

Comment Management: Provides functionality for users to update or delete their comments, ensuring an interactive and dynamic user experience.

Comment Display: Retrieves and displays user comments on the actor profile pages, along with options to update or delete each comment.

1.3.5 Administrative Module

Database Maintenance: Ensures the integrity and consistency of the database by handling CRUD (Create, Read, Update, Delete) operations on actor and movie data.

Error Handling: Manages errors and exceptions that may occur during database operations or user interactions, ensuring smooth application performance.

2. SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

The "Star Info" project is a sophisticated software application developed to provide comprehensive information about actors from both foreign and Indian cinema. It leverages a combination of Python, Tkinter, PIL (Python Imaging Library), and MySQL to deliver an interactive and user-friendly platform. Below is a detailed description of the software components and technologies used in the project:

2.1.1 Programming Language

Python:

The core programming language used for developing the "Star Info" application. Python was chosen for its simplicity, versatility, and wide range of libraries that facilitate GUI development and database interaction.

2.1.2 Graphical User Interface (GUI)

Tkinter:

Tkinter is Python's standard GUI library. It provides a fast and easy way to

create GUI applications. Tkinter is used to create all the windows, frames, buttons, labels, text boxes, and other interactive elements in the "Star Info" application.

Key features implemented using Tkinter include:

- **Main Window:** The starting point of the application, which provides an introduction and navigational options.
- **Actor Selection Windows:** Separate windows for browsing foreign and Indian actors.
- **Actor Details Window:** Displays detailed information about selected actors, including their biographies, filmographies, and user comments.

2.1.3 Image Handling

PIL (Python Imaging Library): PIL, now maintained under the name Pillow, is used for opening, manipulating, and displaying images. In "Star Info", PIL is utilized to:

- Load and resize actor images for display in the GUI.
- Ensure that images are displayed with the correct aspect ratio and resolution.

2.1.4 Database Management

- **MySQL:** MySQL is a popular open-source relational database management system. It is used in the "Star Info" project to store and manage data related to actors, movies, and user comments. Key aspects of MySQL integration include:
- **Database Connection:** Establishing a connection to the MySQL database using the `mysql.connector` library.

- **Data Storage:** Structuring and storing actor details, movie information, and comments in appropriate tables within the database.
- **Query Execution:** Performing SQL queries to insert, retrieve, update, and delete data as needed by the application.
- **Error Handling:** Managing exceptions and errors that occur during database operations to ensure the application's stability and reliability.

2.1.5 Software Features

The "Star Info" application includes several key features designed to enhance user experience and provide valuable information:

- **Actor Profiles:** Comprehensive profiles for each actor, including biography, nationality, and other personal details.
- **Filmographies:** Detailed lists of movies associated with each actor, including titles, synopses, IMDb ratings, OTT platforms, release dates, and box office collections.
- **Interactive Commenting System:** Allows users to leave, update, and delete comments on actor profiles, fostering engagement and interaction.
- **Dynamic Navigation:** Smooth and intuitive navigation between different sections of the application, including main pages, actor selection pages, and detailed actor profiles.

2.1.6 Software Architecture

The architecture of the "Star Info" application is designed to ensure modularity, scalability, and maintainability:

- **Modular Design:** The application is divided into distinct modules, each responsible for specific functionality (e.g., GUI management, database operations, image handling).
- **Event-Driven Programming:** Utilizes event-driven programming to

handle user interactions, such as button clicks and text entry, ensuring responsive and dynamic behavior.

- **Database Integration:** Employs a robust database backend to manage and retrieve data efficiently, supporting the application's information-rich content.

Software Development Tools

Various development tools and libraries were used to create the "Star Info" application:

- **Python IDLE:** Python's Integrated Development and Learning Environment, used for writing and debugging the application code.
- **MySQL Workbench:** A visual tool for database design and management, used to create and manage the MySQL database schema.
- **Pillow (PIL Fork):** Used for image processing tasks, such as loading and resizing actor images.

The "Star Info" project combines the power of Python, Tkinter, PIL, and MySQL to deliver a feature-rich, user-friendly application for exploring the world of cinema. Its modular design and robust architecture ensure that it is both scalable and maintainable, providing a valuable resource for users to discover and engage with information about their favorite actors and movies.

2.2 LANGUAGES

2.2.1 SQL

2.2.2 PYTHON

3.REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

1. Functional Requirements

1.1 User Roles Guest User: Can browse and search for movies and TV shows. Registered User: Can rate, review, and create watchlists. Admin: Can manage content and user accounts.

1.2 User Authentication and Authorization Registration: Users can sign up using an email address and password or via social media accounts. Login: Users can log in using their registered credentials.

Password Recovery: Users can reset their password via email. User Profiles: Users can view and edit their profiles.

1.3 Movie and TV Show Information Search and Browse: Users can search and browse movies and TV shows by various criteria (e.g., title, genre, year, rating). Details Page: Detailed pages for movies and TV shows, including: Title Synopsis Genre Release date Director, cast, and crew Ratings and reviews Trailers and images Related Content: Recommendations for similar movies and TV shows.

1.4 Ratings and Reviews Rating System: Users can rate movies and TV shows on a scale of 1 to 10.

Reviews: Users can write and read reviews. Review Moderation: Admins can moderate reviews to ensure quality and appropriateness.

2. Non-Functional Requirements

2.1 Performance Response Time:

The application should have a response time of less than 2 seconds for most operations.

Scalability: The application must handle a large number of concurrent users.

2.2 Usability Intuitive UI: The user interface should be easy to navigate.

Accessibility: The application should be accessible to users with disabilities.

2.3 Security Data Encryption: All sensitive data should be encrypted.

Authentication: Strong authentication mechanisms must be in place.

Authorization: Proper authorization checks to ensure users can only perform allowed actions.

2.4 AvailabilityUptime: The application should have an uptime of 99.9%.

Backup: Regular backups should be maintained.

2.5 MaintainabilityCode Quality: The codebase should follow best practices and be well-documented.

Modularity: The application should be modular to allow easy updates and additions of features.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS:

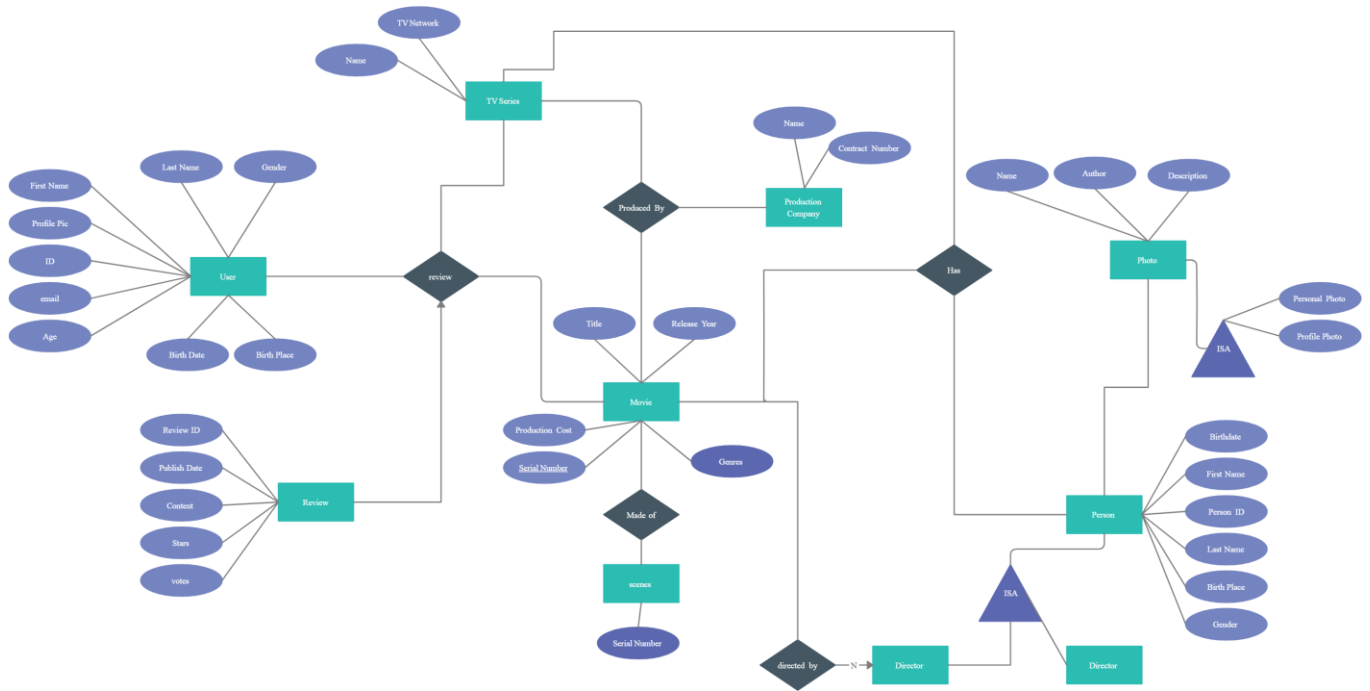
Software Requirements

- Operating System Windows 10
- Front End PYTHON
- Back End MySQL

Hardware Requirements

- Desktop PC or a Laptop
- Printer
- Operating System – Windows 10
- Intel® Core™ i3-6006U CPU @ 2.00GHz
- 4.00 GB RAM
- 64-bit operating system, x64 based processor
- 1024 x 768 monitor resolution
- Keyboard and Mouse

3.3 ER DIAGRAM



3.4 NORMALISATION

REVIEW TABLE:

Column	DataType
Review_ID	INT
UserID	INT (FK)
Movie_ID	INT (FK)
Rating	INT
Comment	TEXT
ReviewDate	DATE

MOVIE TABLE:

COLUMN	DataType
MovieID	INT
Title	VARCHAR
ReleaseDate	DATE
Language	VARCHAR

ACTOR TABLE:

COLUMN	DataType
ActorID	INT
Name	VARCHAR
DateOfBirth	DATE

4.PROGRAM CODE

4.1 CODE DETAILS AND CODE EFFICIENCY

```
from tkinter import *
from PIL import ImageTk, Image
import tkinter as tk
import mysql.connector

description_font = ("Calibri", 20)
detail_font = ("Calibri", 15)

# Sample data: actor names and image paths for foreign actors
foreign_actors = ["Robert Downey Jr.", "Keanu Reeves", "Tom Cruise", "Leonardo DiCaprio",
                  "Jackie Chan", "Brad Pitt", "Chris Evans", "Chris Hemsworth",
                  "Christian Bale", "Will Smith", "Johnny Depp", "Arnold Schwarzenegger"]
foreign_image_paths = ["rdj.jpeg", "keanu.jpeg", "tom.jpeg", "leo.jpeg", "jackie.jpeg", "brad.jpeg",
                       "chris_evans.jpeg", "chris_hemsworth.jpeg", "bale.jpeg", "will.jpeg",
                       "depp.jpeg", "arnold.jpeg"]

# Sample data: actor names and image paths for Indian actors
indian_actors = ["Shah Rukh Khan", "Vijay", "Ajith Kumar", "Kamal Hassan",
                 "Amir Khan", "Dhanush"]
indian_image_paths = ["srk.jpeg", "vijay.jpeg", "ajith.jpeg", "kamal.jpeg",
                      "amir.jpeg", "dhanush.jpeg"]

def show_actor_details(actor_id, actor_type):
```

```

def save_comment():
    comment = comment_entry.get("1.0", "end-1c") # Get comment from text box
    if comment:
        try:
            mydb = mysql.connector.connect(
                host="localhost",
                user="root",
                password="dhoni777",
                database="infostar",
                auth_plugin='mysql_native_password'
            )
            mycursor = mydb.cursor()
            sql = "INSERT INTO comments (actor_id, comment) VALUES (%s, %s)"
            val = (actor_id, comment)
            mycursor.execute(sql, val)
            mydb.commit()
            mycursor.close()
            mydb.close()
            print("Comment saved successfully:", comment)
            display_comments() # Refresh comments display
            comment_entry.delete("1.0", END) # Clear the text box
        except mysql.connector.Error as err:
            print("Error accessing database:", err)

```

```

def delete_comment(comment_id):
    try:
        mydb = mysql.connector.connect(
            host="localhost",
            user="root",
            password="dhoni777",
            database="infostar",
            auth_plugin='mysql_native_password'
        )
        mycursor = mydb.cursor()
        sql = "DELETE FROM comments WHERE id = %s"
        val = (comment_id,)
        mycursor.execute(sql, val)
        mydb.commit()
        mycursor.close()
        mydb.close()
        display_comments() # Refresh comments display
        print(f"Comment with id {comment_id} deleted successfully.")
    
```

```
except mysql.connector.Error as err:
    print("Error accessing database:", err)
```

```
def update_comment(comment_id, new_comment):
    try:
        mydb = mysql.connector.connect(
            host="localhost",
            user="root",
            password="dhoni777",
            database="infostar",
            auth_plugin='mysql_native_password'
        )
        mycursor = mydb.cursor()
        sql = "UPDATE comments SET comment = %s WHERE id = %s"
        val = (new_comment, comment_id)
        mycursor.execute(sql, val)
        mydb.commit()
        mycursor.close()
        mydb.close()
        print("Comment updated successfully:", new_comment)
        display_comments() # Refresh comments display
    except mysql.connector.Error as err:
        print("Error accessing database:", err)
```

```
def edit_comment(comment_id, current_comment):
    edit_window = tk.Toplevel()
    edit_window.title("Edit Comment")
    edit_entry = tk.Text(edit_window, height=5, width=100, wrap="word", font=detail_font)
    edit_entry.pack(pady=(10, 10))
    edit_entry.insert("1.0", current_comment)
```

```
def save_edited_comment():
    new_comment = edit_entry.get("1.0", "end-1c")
    if new_comment:
        update_comment(comment_id, new_comment)
        edit_window.destroy()
```

```
    save_button = tk.Button(edit_window, text="Save", command=save_edited_comment,
font=detail_font)
    save_button.pack(pady=(0, 10))
```

```
def display_comments():
```



```

for widget in comments_frame.winfo_children():
    widget.destroy() # Clear the existing comments display

try:
    mydb = mysql.connector.connect(
        host="localhost",
        user="root",
        password="dhoni777",
        database="infostar",
        auth_plugin='mysql_native_password'
    )
    mycursor = mydb.cursor()
    mycursor.execute("SELECT id, comment FROM comments WHERE actor_id = %s",
(actor_id,))
    comments_data = mycursor.fetchall()
    mycursor.close()
    mydb.close()

    for comment in comments_data:
        comment_id = comment[0]
        comment_text = comment[1]
        comment_frame = tk.Frame(comments_frame)
        comment_frame.pack(anchor="w", pady=(0, 5), fill="x")

        comment_label = tk.Label(comment_frame, text=comment_text, wraplength=800,
justify=tk.LEFT, font=detail_font)
        comment_label.pack(side="left", padx=(0, 10))

        update_button = tk.Button(comment_frame, text="Update Comment", command=lambda
c_id=comment_id, c_text=comment_text: edit_comment(c_id, c_text), font=detail_font)
        update_button.pack(side="left", padx=(0, 5))

        delete_button = tk.Button(comment_frame, text="Delete Comment", command=lambda
c_id=comment_id: delete_comment(c_id), font=detail_font)
        delete_button.pack(side="left")

except mysql.connector.Error as err:
    print("Error accessing database:", err)

# Connect to MySQL database
print(actor_id, actor_type)

mydb = mysql.connector.connect(

```

```

host="localhost",
user="root",
password="dhoni777",
database="infostar",
auth_plugin='mysql_native_password'
)
print(mydb)

# Create cursor object to execute queries
mycursor = mydb.cursor()

try:
    # Execute select query to fetch actor details
    mycursor.execute("SELECT * FROM actors WHERE actor_id = %s", (actor_id,))
    actor_data = mycursor.fetchone()

    # Execute select query to fetch movies of the actor
    mycursor.execute("SELECT * FROM movies WHERE actor_id = %s", (actor_id,))
    movies_data = mycursor.fetchall()

    # Close cursor and database connection
    mycursor.close()
    mydb.close()

    # Display actor details and movies in a new window or dialog
    actor_details_window = tk.Toplevel()
    actor_details_window.title(actor_data[1])
    image = Image.open("./assets/"+(foreign_image_paths[actor_id-1] if actor_type=="foreign" else
indian_image_paths[actor_id-13])).resize((150, 200))
    image = ImageTk.PhotoImage(image)
    image_label = tk.Label(actor_details_window, image=image)
    image_label.image = image # Keep a reference to avoid garbage collection
    image_label.pack()

    # Set window title to actor's name

    # Create a frame to hold all the content
    frame = tk.Frame(actor_details_window)
    frame.pack(fill="both", expand=True)

    # Add a canvas to support scrollbars
    canvas = tk.Canvas(frame)

```

```

canvas.pack(side="left", fill="both", expand=True)

# Add a vertical scrollbar for the canvas
v_scrollbar = tk.Scrollbar(frame, orient="vertical", command=canvas.yview)
v_scrollbar.pack(side="right", fill="y")
canvas.config(yscrollcommand=v_scrollbar.set)

# Create another frame to hold the actual content
content_frame = tk.Frame(canvas)
canvas.create_window((0, 0), window=content_frame, anchor="nw")

# Display actor details
actor_name_label = tk.Label(content_frame, text="Name: " + actor_data[1], wraplength=900,
justify=tk.LEFT, font=description_font)
actor_name_label.pack()

actor_nationality_label = tk.Label(content_frame, text="Nationality: " + actor_data[2],
wraplength=900, justify=tk.LEFT, font=detail_font)
actor_nationality_label.pack()

actor_about_label = tk.Label(content_frame, text="About: " + actor_data[3], wraplength=900,
justify=tk.LEFT, font=detail_font)
actor_about_label.pack()

# Display movies
for movie in movies_data:
    movies_label1 = tk.Label(content_frame, text="Movie: " + movie[0] + " Synopsis: " + movie[1],
wraplength=1100, justify=tk.LEFT, font=detail_font)
    movies_label1.pack()

    movies_label2 = tk.Label(content_frame, text="IMDB Rating: " + str(movie[2]) + " OTT
Platform: " + str(movie[3]) + " Release Date: " + str(movie[4]) + " Box Office Collection: " +
str(movie[5]), wraplength=1100, justify=tk.LEFT, font=detail_font)
    movies_label2.pack()

# Comment section
comment_label = tk.Label(content_frame, text="Leave a Comment:", wraplength=900,
justify=tk.LEFT, font=detail_font)
comment_label.pack(pady=(10, 0))

comment_entry = tk.Text(content_frame, height=5, width=100, wrap="word", font=detail_font)
comment_entry.pack(pady=(0, 10))

```

```

        comment_button = tk.Button(content_frame, text="COMMENT", command=save_comment,
font=detail_font)
        comment_button.pack(pady=(0, 10))

    # Display existing comments
    comments_frame = tk.Frame(content_frame)
    comments_frame.pack(fill="both", expand=True, pady=(10, 0))

    display_comments()

    # Update the canvas to fit the full content
    content_frame.update_idletasks()
    canvas.config(scrollregion=canvas.bbox("all"))

except mysql.connector.Error as err:
    print("Error accessing database:", err)

def on_configure(canvas):
    canvas.config(scrollregion=canvas.bbox("all"))

def open_foreign_actors_page():
    foreign_actors_window = tk.Toplevel()
    foreign_actors_window.title("Foreign Actors")
    foreign_actors_window.geometry("690x350")

    # Create a frame to hold actor photos and names
    frame = tk.Frame(foreign_actors_window)
    frame.pack()

    # Add a canvas to support scrollbars
    canvas = tk.Canvas(frame, width=650, height=340)
    canvas.pack(side="left", fill="both", expand=True)

    # Add a vertical scrollbar
    v_scrollbar = tk.Scrollbar(frame, orient="vertical", command=canvas.yview)
    v_scrollbar.pack(side="right", fill="y")
    canvas.config(yscrollcommand=v_scrollbar.set)

    # Create another frame to hold the actual content
    content_frame = tk.Frame(canvas, width=650, height=340)
    content_frame.pack()

```

```

# Display actor photos and names
for i, actor in enumerate(foreign_actors):
    # Load actor photo
    image = Image.open("./assets/"+foreign_image_paths[i]).resize((200, 300))
    image = ImageTk.PhotoImage(image)
    image_label = tk.Label(content_frame, image=image)
    image_label.image = image # Keep a reference to avoid garbage collection
    image_label.grid(row=(i // 3)*2, column=i % 3, padx=10, pady=10)

    # Create label with actor name
    name_label = tk.Label(content_frame, text=actor)
    name_label.grid(row=(i // 3)*2 + 1, column=i % 3)

    # Bind click event to show actor details
    image_label.bind("<Button-1>", lambda event, actor_id=i+1, actor_type="foreign":
show_actor_details(actor_id, actor_type))
    name_label.bind("<Button-1>", lambda event, actor_id=i+1, actor_type="foreign":
show_actor_details(actor_id, actor_type))

# Configure the scrollbars to scroll with the canvas
canvas.create_window((0, 0), window=content_frame, anchor="nw")
canvas.bind("<Configure>", lambda event, canvas=canvas: on_configure(canvas))

def open_indian_actors_page():
    indian_actors_window = tk.Toplevel()
    indian_actors_window.title("Indian Actors")
    indian_actors_window.geometry("690x350")

    # Create a frame to hold actor photos and names
    frame = tk.Frame(indian_actors_window)
    frame.pack()

    # Add a canvas to support scrollbars
    canvas = tk.Canvas(frame, width=650, height=340)
    canvas.pack(side="left", fill="both", expand=True)

    # Add a vertical scrollbar
    v_scrollbar = tk.Scrollbar(frame, orient="vertical", command=canvas.yview)
    v_scrollbar.pack(side="right", fill="y")
    canvas.config(yscrollcommand=v_scrollbar.set)

    # Create another frame to hold the actual content

```

```

content_frame = tk.Frame(canvas, width=650, height=340)
content_frame.pack()

# Display actor photos and names
for i, actor in enumerate(indian_actors):
    # Load actor photo
    image = Image.open("./assets/"+indian_image_paths[i]).resize((200, 300))
    image = ImageTk.PhotoImage(image)
    image_label = tk.Label(content_frame, image=image)
    image_label.image = image # Keep a reference to avoid garbage collection
    image_label.grid(row=(i // 3)*2, column=i % 3, padx=10, pady=10)

    # Create label with actor name
    name_label = tk.Label(content_frame, text=actor)
    name_label.grid(row=(i // 3)*2 + 1, column=i % 3)

    # Bind click event to show actor details
    image_label.bind("<Button-1>", lambda event, actor_id=i+13, actor_type="indian":
show_actor_details(actor_id, actor_type))
    name_label.bind("<Button-1>", lambda event, actor_id=i+13, actor_type="indian":
show_actor_details(actor_id, actor_type))

# Configure the scrollbars to scroll with the canvas
canvas.create_window((0, 0), window=content_frame, anchor="nw")
canvas.bind("<Configure>", lambda event, canvas=canvas: on_configure(canvas))

def open_choose_page():
    choose_window = tk.Toplevel()
    choose_window.title("Choose")

    # Custom fonts
    title_font = ("Calibri", 24, "bold")
    button_font = ("Calibri", 20)

    # Title
    title_label = tk.Label(choose_window, text="Choose", font=title_font)
    title_label.pack(pady=20)

    # Button for Foreign Actors
    foreign_actors_button = tk.Button(choose_window, text="Foreign Actors",
command=open_foreign_actors_page, font=button_font)
    foreign_actors_button.pack(pady=10)

```

```

# Button for Indian Actors
indian_actors_button = tk.Button(choose_window, text="Indian Actors",
command=open_indian_actors_page, font=button_font)
indian_actors_button.pack(pady=10)

def main():
    root = tk.Tk()
    root.title("Star Info")

    # Custom fonts
    title_font = ("Calibri", 36, "bold")

    # Title
    title_label = tk.Label(root, text="STAR INFO", font=title_font)
    title_label.pack(pady=20)

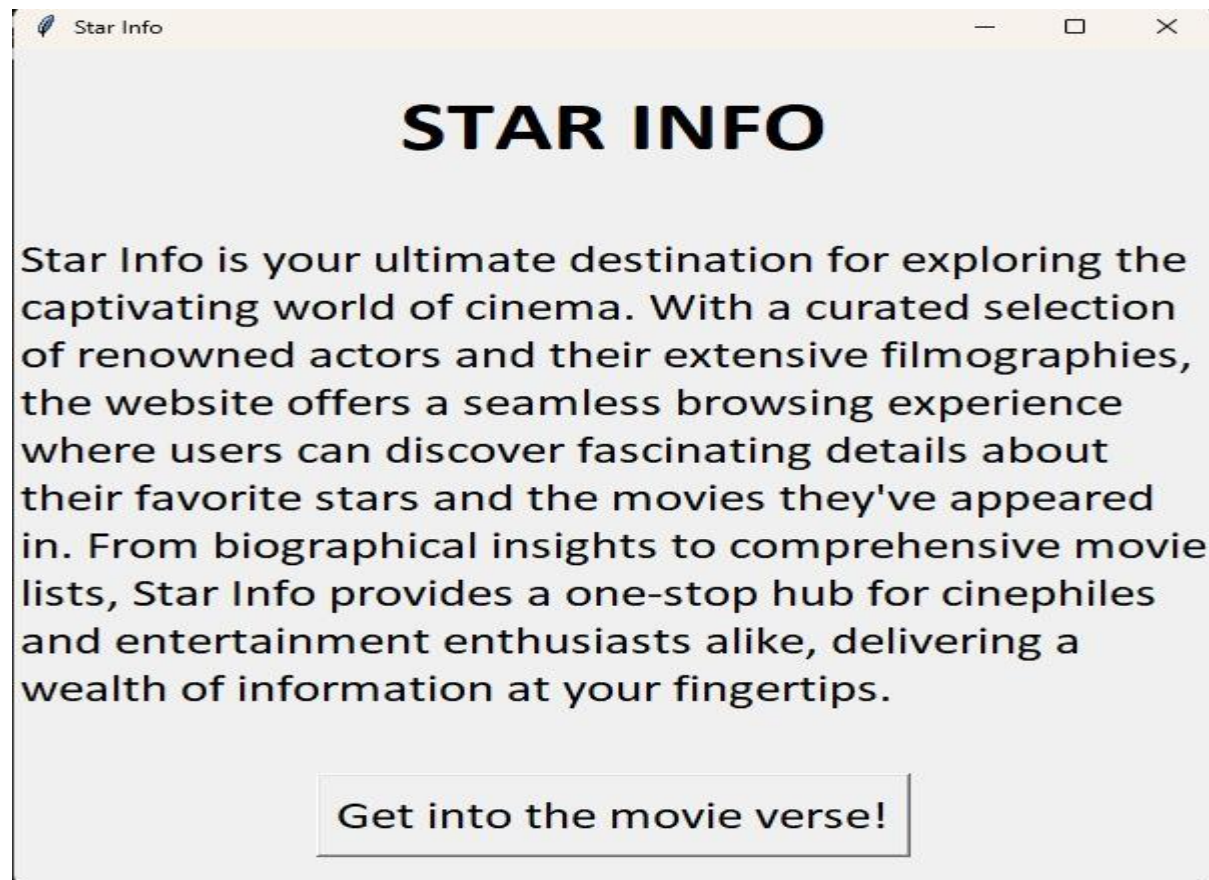
    # Description
    description = ("Star Info is your ultimate destination for exploring the captivating world of cinema. "
        "With a curated selection of renowned actors and their extensive filmographies, "
        "the website offers a seamless browsing experience where users can discover fascinating "
        details "
        "about their favorite stars and the movies they've appeared in. "
        "From biographical insights to comprehensive movie lists, "
        "Star Info provides a one-stop hub for cinephiles and entertainment enthusiasts alike, "
        "delivering a wealth of information at your fingertips.")
    description_label = tk.Label(root, text=description, wraplength=600, justify=tk.LEFT,
font=description_font)
    description_label.pack(pady=20)
    # Button to navigate to the "choose" panel
    button = tk.Button(root, text="Get into the movie verse!", command=open_choose_page,
font=("Calibri", 20))
    button.pack(pady=20)
    root.mainloop()

if __name__ == "__main__":
    main()

```

5.RESULT AND DISCUSSION

5.1 USER DOCUMENTATION





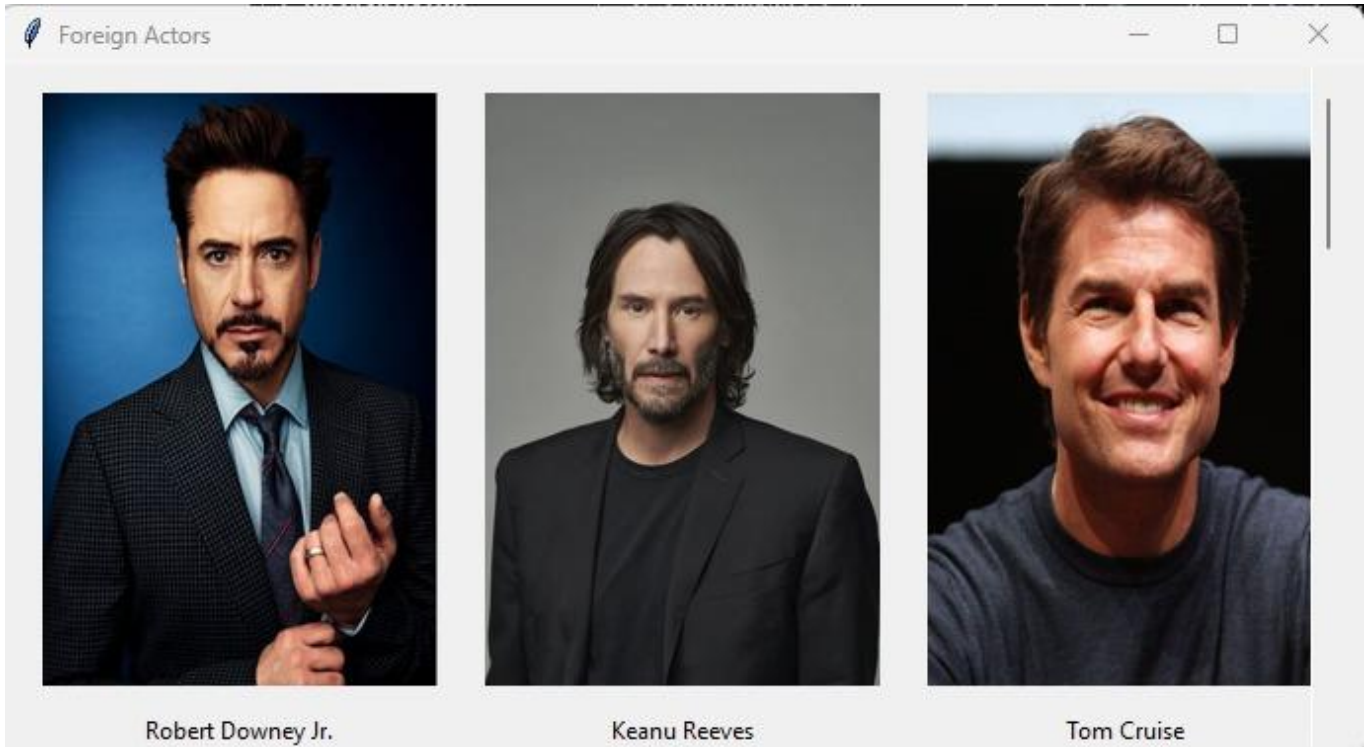
C.




Choose

Foreign Actors

Indian Actors





Name: Robert Downey Jr.

Nationality: American

About: Robert John Downey Jr. is an American actor, producer, and singer. He has been featured in numerous movies and is best known for his portrayal of Tony Stark / Iron Man in the Marvel Cinematic Universe.

Movie: AVENGERS:INFINITY-WAR Synopsis: The Avengers must stop Thanos, an intergalactic warlord, from getting his hands on all the infinity stones. However, Thanos is prepared to go to any lengths to carry out his insane plan.

IMDB Rating: 8.4 OTT Platform: Hotstar Release Date: 2018-APR-27 Box Office Collection: 204835975.40

Movie: IRON MAN Synopsis: When Tony Stark, an industrialist, is captured, he constructs a high-tech armoured suit to escape. Once he manages to escape, he decides to use his suit to fight against evil forces to save the world.


IMDB Rating: 7.9 OTT Platform: AMAZON Release Date: 2008-MAY-01 Box Office Collection: 140000000.00

Movie: SHERLOCK HOLMES Synopsis: Detective Sherlock Holmes and his partner, Dr Watson, send Blackwood, a serial killer, to the gallows. However, they are shocked to learn that he is back from the dead and must pursue him again.

IMDB Rating: 7.6 OTT Platform: AMAZON Release Date: 2010-JAN-08 Box Office Collection: 62304277.00

Movie: TROPIC THUNDER Synopsis: A films crew is in Southeast Asia shooting a movie when one of the actors, Tugg, is kidnapped by a local thug. They are forced to be together till they find Tugg while being stalked by drug dealers.

IMDB Rating: 7.1 OTT Platform: JIO-CINEMA Release Date: 2008-AUG-13 Box Office Collection: 195700000.00



IMDB Rating: 8.4 OTT Platform: HUTSTAR Release Date: 2019-APR-26 Box Office Collection: 279500000.00

Movie: OPPENHEIMER Synopsis: During World War II, Lt. Gen. Leslie Groves Jr. appoints physicist J. Robert Oppenheimer to work on the top-secret Manhattan Project. Oppenheimer and a team of scientists spend years developing and designing the atomic bomb. Their work comes to fruition on July 16, 1945, as they witness the worlds first nuclear explosion, forever changing the course of history.

IMDB Rating: 8.3 OTT Platform: JIO CINEMA Release Date: 2023-JUL-21 Box Office Collection: 953800000.00

Leave a Comment:

COMMENT

He is one of the best actors

He got Oscar for Oppenheimer

6.TESTING

6.1 Unit Testing Unittesting is a testing technique in which modules are tested individually. Small individual units of source code are tested to determine whether it is fit to use or not. Different modules of games are put to test while the modules are being developed. Here modules refer to individual levels, players, scenes

6.2 Integration Testing Integration testing is the technique in which individual components or modules are grouped together and tested. It occurs after testing. The inputs for the integrated testing are the modules that have already been unit tested.

6.3 System TestingSystem testing is conducted on the entire system as a whole to check whether the system meets its requirements or not. software was installed on different systems and any errors or bugs that occurred were fixed.

6.4 Acceptance Testing User Acceptance is defined as a

type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon. This testing happens in the final phase of testing before moving the software application to the Market or Production environment.

7.CONCLUSION

The IMDb application for documents, designed to manage and organize extensive collections of documents, has demonstrated substantial efficacy in enhancing productivity and streamlining workflow. By leveraging IMDb's robust framework, the application ensures efficient categorization, easy retrieval, and comprehensive metadata management. This not only reduces the time spent searching for documents but also improves overall document management processes. Its user-friendly interface and powerful search capabilities make it an invaluable tool for individuals and organizations seeking to optimize their document handling procedures. Ultimately, the IMDb application for documents represents a significant advancement in document management technology, providing users with a reliable and efficient solution.