



**CINEBOOK**

**PODAR INTERNATIONAL SCHOOL**

Submitted by: Mahip Thakore

Sparsh Bhargava

Class: 12TH A-1

Submitted to: Mr. Mitul Bhatt

PGT(CS)

**MOVIE BOOKING SYSTEM**

COMPUTER SCIENCE PROJECT

2025-2026

INDEX

|  |
| --- |
| Certificate |
| Acknowledgement |
| Hardware & Software |
| Introduction |
| Python Source Code |
| MySQL Database |
| Output |
| Bibliography |

HARDWARE

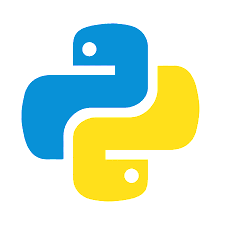
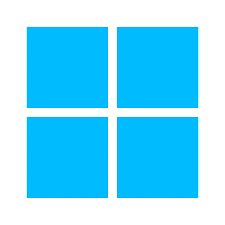
&  
SOFTWARE

HARDWARE:

1. Lenovo LOQ
2. Desktop

SOFTWARE:

1. Windows 11
2. MySQL
3. Python IDLE



INTRODUCTION

CineBook is a comprehensive **desktop-based movie ticket booking application** built with Python and Tkinter for the graphical user interface. This robust system provides an end-to-end solution for movie theatres to manage their movie listings, showtimes, seat bookings, and customer ticketing operations.

**Key Features**

**🎬 Movie Management**

* **Dual-section catalogue**: "Now Showing" and "Upcoming" movies
* **Detailed movie information**: Title, description, ratings, tags, release dates, and poster images
* **Admin panel** for adding, editing, and removing movies
* **Automatic poster management** with file copying and organization

**🎫 Ticket Booking System**

* **Interactive seat selection** with real-time availability
* **Multiple showtimes** throughout the day (10:00 AM to 10:00 PM)
* **8 auditorium support** with customizable seating layouts
* **Date-specific booking** with calendar integration

**💾 Database Architecture**

* **MySQL backend** with relational data modelling
* **Four main tables**: Movies, Shows, Seats, and Bookings
* **Referential integrity** with proper foreign key constraints
* **Automated database setup** and sample data seeding

**🎨 User Experience**

* **Modern dark-themed UI** with intuitive navigation
* **Scrollable movie grids** with high-quality poster displays
* **Rotating advertisement banners** for promotions
* **Responsive design** with smooth interactions

**📱 Admin Capabilities**

* **Complete CRUD operations** for movie management
* **Poster image handling** with automatic file organization
* **Section management** (Now Showing vs Upcoming)
* **Release date scheduling** and movie categorization

**🎟️ Digital Ticketing**

* **QR code integration** for quick verification
* **Professional ticket generation** with booking details
* **Save-to-PNG functionality** for digital tickets
* **Booking confirmation** with seat numbers and show details

**Technical Stack**

* **Frontend**: Tkinter (Python's standard GUI toolkit)
* **Backend**: MySQL with mysql-connector-python
* **Image Processing**: Pillow (PIL) for image handling
* **QR Generation**: qrcode library
* **Date Management**: tkcalendar for date pickers
* **File Management**: Built-in OS and in-build modules

**System Requirements**

* **Python 3.7+** with required packages
* **MySQL Server** (local or remote)
* **Minimum 2GB RAM**
* **Windows/Linux/macOS** compatible

**Target Users**

1. **Movie Theatres** - For managing their daily operations
2. **Cinema Chains** - Multi-auditorium support
3. **Event Managers** - For special screenings and premieres
4. **Customers** - Easy ticket booking experience

**Business Benefits**

* **Streamlined operations** with automated booking processes
* **Reduced manual errors** through digital seat management
* **Enhanced customer experience** with visual seat selection
* **Increased revenue** through efficient seat utilization
* **Marketing opportunities** via promotional ad banners

**Future Enhancements**

The modular architecture allows for easy integration of additional features such as:

* Online payment gateway integration
* Mobile app companion
* Loyalty programs and discounts
* Advanced analytics and reporting
* Multi-language support
* Social media sharing capabilities

CineBook represents a modern approach to cinema management, combining robust backend functionality with an elegant user interface to deliver a seamless movie booking experience for both theatre operators and moviegoers.

PYTHON SOURCE CODE

# main.py

import os

import shutil

import mysql.connector

from mysql.connector import errorcode

import tkinter as tk

from tkinter import ttk, messagebox, filedialog, Toplevel, Canvas

from tkcalendar import Calendar

from PIL import Image, ImageTk, ImageDraw, ImageFont

import qrcode

from datetime import datetime, date

# ---------------- CONFIG (Update your MySQL credentials here) ----------------

DB\_CONFIG = {

"host": "localhost",

"user": "root",

"password": "Hawk\_Mahip0", # <-- change

"database": "cinebook\_db"

}

# -----------------------------------------------------------------------------

# ---------------- PATHS ----------------

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

POSTERS\_DIR = os.path.join(BASE\_DIR, "posters")

ADS\_DIR = os.path.join(BASE\_DIR, "ads")

os.makedirs(POSTERS\_DIR, exist\_ok=True)

os.makedirs(ADS\_DIR, exist\_ok=True)

# ---------------- SAFE IMAGE LOADER ----------------

def load\_image\_safe(rel\_path, size):

"""Load image relative to BASE\_DIR; return ImageTk.PhotoImage or None."""

full = os.path.join(BASE\_DIR, rel\_path)

if not os.path.exists(full):

print(f"[load\_image\_safe] Missing: {full}")

return None

try:

img = Image.open(full).convert("RGBA")

img = img.resize(size, Image.LANCZOS)

return ImageTk.PhotoImage(img)

except Exception as e:

print("Error loading image:", full, e)

return None

# ---------------- PLACEHOLDER ADS ----------------

def ensure\_ads\_placeholders():

files = os.listdir(ADS\_DIR)

if len([f for f in files if f.lower().endswith((".png",".jpg",".jpeg"))]) == 0:

for i in range(1,4):

path = os.path.join(ADS\_DIR, f"ad{i}.png")

if not os.path.exists(path):

img = Image.new("RGB", (980,140), (40+i\*30, 70+i\*10, 100+i\*5))

d = ImageDraw.Draw(img)

fnt = ImageFont.load\_default()

d.text((20,60), f"Place your ad in /ads/ad{i}.png", fill="white", font=fnt)

img.save(path)

# ---------------- MYSQL SETUP ----------------

def ensure\_database\_and\_tables():

# create database if not present

try:

cnx = mysql.connector.connect(host=DB\_CONFIG["host"], user=DB\_CONFIG["user"], password=DB\_CONFIG["password"])

cur = cnx.cursor()

cur.execute(f"CREATE DATABASE IF NOT EXISTS {DB\_CONFIG['database']}")

cur.close()

cnx.close()

except mysql.connector.Error as e:

messagebox.showerror("Database error", f"Could not ensure database: {e}")

raise

# create tables

try:

cnx = mysql.connector.connect(\*\*DB\_CONFIG)

cur = cnx.cursor()

cur.execute("""

CREATE TABLE IF NOT EXISTS movies (

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255),

description TEXT,

rating VARCHAR(20),

tags VARCHAR(255),

poster\_path VARCHAR(255),

section ENUM('Now Showing','Upcoming'),

release\_date DATE

) ENGINE=InnoDB;

""")

cur.execute("""

CREATE TABLE IF NOT EXISTS shows (

id INT AUTO\_INCREMENT PRIMARY KEY,

movie\_id INT,

show\_date DATE,

show\_time TIME,

audi INT,

FOREIGN KEY (movie\_id) REFERENCES movies(id) ON DELETE CASCADE

) ENGINE=InnoDB;

""")

cur.execute("""

CREATE TABLE IF NOT EXISTS seats (

id INT AUTO\_INCREMENT PRIMARY KEY,

show\_id INT,

seat\_no VARCHAR(10),

is\_booked TINYINT(1) DEFAULT 0,

FOREIGN KEY (show\_id) REFERENCES shows(id) ON DELETE CASCADE

) ENGINE=InnoDB;

""")

cur.execute("""

CREATE TABLE IF NOT EXISTS bookings (

id INT AUTO\_INCREMENT PRIMARY KEY,

show\_id INT,

seat\_no VARCHAR(10),

customer\_name VARCHAR(255),

booked\_on DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (show\_id) REFERENCES shows(id) ON DELETE CASCADE

) ENGINE=InnoDB;

""")

cnx.commit()

cur.close()

cnx.close()

except mysql.connector.Error as e:

messagebox.showerror("Database error", f"MySQL error: {e}")

raise

# ---------------- SEED SAMPLE MOVIES (FORCE UPDATE) ----------------

def seed\_sample\_movies():

cnx = mysql.connector.connect(\*\*DB\_CONFIG)

cur = cnx.cursor()

# CLEAR EXISTING DATA

cur.execute("DELETE FROM bookings")

cur.execute("DELETE FROM seats")

cur.execute("DELETE FROM shows")

cur.execute("DELETE FROM movies")

sample = [

# Now Showing (Jan–May)

("Interstellar", "A team travels through a wormhole to find a new home for humanity.", "95%", "IMAX, Dolby Atmos", os.path.join("posters","interstellar.png"), "Now Showing", "2026-01-15"),

("Oppenheimer", "The story of J. Robert Oppenheimer.", "93%", "IMAX, Dolby Vision", os.path.join("posters","oppenheimer.png"), "Now Showing", "2026-02-12"),

("KGF", "Rise of a man from poverty to power.", "88%", "Dolby", os.path.join("posters","kgf.png"), "Now Showing", "2026-03-05"),

("Fast X", "High-octane car action.", "75%", "IMAX", os.path.join("posters","fast\_x.png"), "Now Showing", "2026-03-22"),

("Ford v Ferrari", "Racing rivalry drama.", "89%", "Dolby", os.path.join("posters","ford\_v\_ferrari.png"), "Now Showing", "2026-04-02"),

("Avatar: Way of Water", "Return to Pandora.", "90%", "IMAX, Dolby Vision", os.path.join("posters","avatar\_way\_of\_water.png"), "Now Showing", "2026-04-20"),

("Hangover", "Comedy of errors.", "80%", "Dolby", os.path.join("posters","hangover.png"), "Now Showing", "2026-05-06"),

("Ratatouille", "A rat dreams of being a chef.", "92%", "Dolby 7.1", os.path.join("posters","ratatouille.png"), "Now Showing", "2026-05-18"),

("Dishoom", "Action-cop movie.", "74%", "Dolby", os.path.join("posters","dishoom.png"), "Now Showing", "2026-05-26"),

("John Wick 4", "Assassin saga continues.", "91%", "Dolby Atmos", os.path.join("posters","johnwick4.png"), "Now Showing", "2026-05-28"),

# Upcoming (Jun–Dec) - CHANGES MADE HERE

("Kung Fu Panda", "Po returns for more adventure.", "N/A", "Dolby Atmos", os.path.join("posters","kung\_fu\_panda.png"), "Upcoming", "2026-06-10"),

("Tron Legacy", "Sam's journey in cyberworld.", "N/A", "IMAX", os.path.join("posters","tron\_legacy.png"), "Upcoming", "2026-07-01"), # MOVED FROM SEPT 25

("Aladdin", "A street rat finds a magic lamp.", "N/A", "Dolby Vision", os.path.join("posters","aladdin.png"), "Upcoming", "2026-07-20"),

("Cruella", "Origin story of Cruella.", "N/A", "Dolby Vision", os.path.join("posters","cruella.png"), "Upcoming", "2026-08-05"),

("Hobbit", "An epic fantasy journey of Bilbo Baggins.", "N/A", "Dolby", os.path.join("posters","hobbit.png"), "Upcoming", "2026-08-25"), # CHANGED FROM LORD OF THE RINGS

("Alien Covenant", "Horror sci-fi returns.", "N/A", "IMAX", os.path.join("posters","alien\_covenant.png"), "Upcoming", "2026-09-10"),

("Inglorious Bastards", "Jewish soldiers to commit violent acts of retribution against the Nazis.", "N/A", "Dolby 7.1", os.path.join("posters","inglorious\_bastards.png"), "Upcoming", "2026-09-25"), # MOVED FROM JULY 1

("Avatar : Fire And Ash", "Pandora saga continues.", "N/A", "IMAX", os.path.join("posters","avatar3.png"), "Upcoming", "2026-10-10"),

]

insert\_q = """INSERT INTO movies (title, description, rating, tags, poster\_path, section, release\_date)

VALUES (%s,%s,%s,%s,%s,%s,%s)"""

cur.executemany(insert\_q, sample)

cnx.commit()

cur.close()

cnx.close()

print("Database updated with new movie list!")

# ---------------- ADMIN: copy poster/ad into folder and return path ----------------

def admin\_copy\_file\_to\_folder(src\_path, folder):

if not os.path.exists(src\_path):

return None

os.makedirs(folder, exist\_ok=True)

fname = os.path.basename(src\_path)

dst = os.path.join(folder, fname)

# if filename already exists, add suffix

base, ext = os.path.splitext(fname)

count = 1

while os.path.exists(dst):

dst = os.path.join(folder, f"{base}\_{count}{ext}")

count += 1

shutil.copyfile(src\_path, dst)

# return relative path from BASE\_DIR

return os.path.relpath(dst, BASE\_DIR)

# ---------------- QR + Receipt generator ----------------

def generate\_ticket\_image(movie\_title, date\_str, time\_str, audi, seats, booking\_id):

# build an image showing the info + a QR code

w, h = 800, 500

img = Image.new("RGB", (w, h), (20,20,20))

d = ImageDraw.Draw(img)

font\_title = ImageFont.load\_default()

try:

font\_title = ImageFont.truetype("arial.ttf", 28)

font\_sub = ImageFont.truetype("arial.ttf", 16)

except:

font\_title = ImageFont.load\_default()

font\_sub = ImageFont.load\_default()

d.text((30,30), f"CineBook Ticket #{booking\_id}", fill="white", font=font\_title)

d.text((30,80), f"Movie: {movie\_title}", fill="white", font=font\_sub)

d.text((30,110), f"Date: {date\_str} Time: {time\_str}", fill="white", font=font\_sub)

d.text((30,140), f"Audi: {audi} Seats: {', '.join(seats)}", fill="white", font=font\_sub)

# QR payload: basic text

qr\_payload = f"{movie\_title}|{date\_str}|{time\_str}|{audi}|{','.join(seats)}|ID:{booking\_id}"

qr = qrcode.make(qr\_payload)

qr = qr.resize((200,200))

img.paste(qr, (w-240, 60))

return img

# ---------------- GUI App ----------------

class CineBookApp:

def \_\_init\_\_(self, root):

self.root = root

root.title("CineBook — Movie Ticket Booking")

root.geometry("1200x760")

root.configure(bg="#0f0f10")

ensure\_ads\_placeholders()

# ad images

self.ad\_images = []

self.ad\_index = 0

self.load\_ads()

# top bar with Admin

top = tk.Frame(root, bg="#0f0f10")

top.pack(fill="x", padx=16, pady=10)

tk.Label(top, text="CineBook", font=("Helvetica", 26, "bold"), fg="white", bg="#0f0f10").pack(side="left")

tk.Button(top, text="Admin", command=self.open\_admin\_panel, bg="#333", fg="white").pack(side="right")

# ad banner

self.ad\_label = tk.Label(root, bg="#0f0f10")

self.ad\_label.pack(pady=(0,12))

self.rotate\_ads() # starts rotation via after()

# main content (scrollable)

content = tk.Frame(root, bg="#0f0f10")

content.pack(fill="both", expand=True, padx=12, pady=(0,18))

self.canvas = Canvas(content, bg="#0f0f10", highlightthickness=0)

vbar = ttk.Scrollbar(content, orient="vertical", command=self.canvas.yview)

self.canvas.configure(yscrollcommand=vbar.set)

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

vbar.pack(side="right", fill="y")

self.inner = tk.Frame(self.canvas, bg="#0f0f10")

self.canvas.create\_window((0,0), window=self.inner, anchor="nw")

self.inner.bind("<Configure>", lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all")))

self.canvas.bind\_all("<MouseWheel>", self.\_on\_mousewheel)

# sections

self.build\_sections()

# ---------------- ads ----------------

def load\_ads(self):

files = [f for f in os.listdir(ADS\_DIR) if f.lower().endswith((".png",".jpg",".jpeg"))]

for f in files:

rel = os.path.join("ads", f)

img = load\_image\_safe(rel, (980,140))

if img:

self.ad\_images.append(img)

def rotate\_ads(self):

if self.ad\_images:

img = self.ad\_images[self.ad\_index % len(self.ad\_images)]

self.ad\_label.config(image=img)

self.ad\_label.image = img

self.ad\_index += 1

else:

# show a simple text if no ads

self.ad\_label.config(text="Place ad images in /ads/", fg="white", bg="#0f0f10", font=("Arial", 14))

self.root.after(4000, self.rotate\_ads)

def \_on\_mousewheel(self, event):

self.canvas.yview\_scroll(int(-1\*(event.delta/120)), "units")

# ---------------- content build ----------------

def build\_sections(self):

# Now Showing

tk.Label(self.inner, text="NOW SHOWING", font=("Helvetica", 18, "bold"), fg="#ffffff", bg="#0f0f10").pack(anchor="w", pady=(6,8))

self.populate\_movie\_grid("Now Showing")

tk.Label(self.inner, text="UPCOMING", font=("Helvetica", 18, "bold"), fg="#ffffff", bg="#0f0f10").pack(anchor="w", pady=(18,8))

self.populate\_movie\_grid("Upcoming")

def populate\_movie\_grid(self, section):

grid = tk.Frame(self.inner, bg="#0f0f10")

grid.pack(padx=8, pady=6, fill="x")

cnx = mysql.connector.connect(\*\*DB\_CONFIG)

cur = cnx.cursor()

cur.execute("SELECT id, title, description, rating, tags, poster\_path, section, release\_date FROM movies WHERE section=%s", (section,))

rows = cur.fetchall()

cur.close()

cnx.close()

cols = 4

for i, row in enumerate(rows):

mid, title, desc, rating, tags, poster\_path, sect, rdate = row

card = tk.Frame(grid, bg="#141414", bd=0)

card.grid(row=i//cols, column=i%cols, padx=12, pady=12, sticky="n")

poster\_img = load\_image\_safe(poster\_path, (160,240))

if poster\_img is None:

# create simple fallback image

img = Image.new("RGB", (160,240), (30,30,30))

d = ImageDraw.Draw(img)

f = ImageFont.load\_default()

d.text((18,110), "No Poster", fill="white", font=f)

poster\_img = ImageTk.PhotoImage(img)

btn = tk.Button(card, image=poster\_img, text=title, compound="top", fg="white", bg="#141414", bd=0,

command=lambda m\_id=mid: self.open\_movie\_window(m\_id))

btn.image = poster\_img

btn.pack()

tk.Label(card, text=title, fg="white", bg="#141414", font=("Helvetica", 11, "bold")).pack(pady=(6,0))

tk.Label(card, text=f"RT: {rating}", fg="#bdbdbd", bg="#141414").pack()

tk.Label(card, text=tags, fg="#9aa0a6", bg="#141414", wraplength=160).pack(pady=(4,8))

# ---------------- movie detail window ----------------

def open\_movie\_window(self, movie\_id):

cnx = mysql.connector.connect(\*\*DB\_CONFIG)

cur = cnx.cursor()

cur.execute("SELECT id, title, description, rating, tags, poster\_path, section, release\_date FROM movies WHERE id=%s", (movie\_id,))

row = cur.fetchone()

cur.close()

cnx.close()

if not row:

messagebox.showerror("Error", "Movie not found")

return

mid, title, desc, rating, tags, poster\_path, section, rdate = row

win = Toplevel(self.root)

win.title(title)

win.geometry("900x620")

win.configure(bg="#0d0d0d")

left = tk.Frame(win, bg="#0d0d0d")

left.pack(side="left", padx=18, pady=18)

right = tk.Frame(win, bg="#0d0d0d")

right.pack(side="right", fill="both", expand=True, padx=18, pady=18)

poster\_img = load\_image\_safe(poster\_path, (320,480))

if poster\_img:

p\_lbl = tk.Label(left, image=poster\_img, bg="#0d0d0d")

p\_lbl.image = poster\_img

p\_lbl.pack()

else:

tk.Label(left, text="No Poster", fg="white", bg="#0d0d0d").pack()

tk.Label(right, text=title, font=("Helvetica", 22, "bold"), fg="white", bg="#0d0d0d").pack(anchor="nw")

tk.Label(right, text=desc or "No description", fg="#ccc", bg="#0d0d0d", wraplength=420, justify="left").pack(anchor="nw", pady=(8,12))

tk.Label(right, text=f"Rating (RT): {rating}", fg="#eee", bg="#0d0d0d").pack(anchor="nw", pady=2)

tk.Label(right, text=f"Tags: {tags}", fg="#eee", bg="#0d0d0d").pack(anchor="nw", pady=2)

tk.Label(right, text=f"Release: {rdate}", fg="#eee", bg="#0d0d0d").pack(anchor="nw", pady=2)

# date picker

tk.Label(right, text="Choose Date:", fg="white", bg="#0d0d0d").pack(anchor="nw", pady=(16,2))

date\_var = tk.StringVar(value="Select date")

tk.Button(right, textvariable=date\_var, width=20, bg="#1f6feb", fg="white",

command=lambda: open\_calendar\_popup(win, section, lambda d: date\_var.set(d))).pack(anchor="nw")

# time picker

tk.Label(right, text="Choose Time:", fg="white", bg="#0d0d0d").pack(anchor="nw", pady=(12,2))

times = ["10:00:00", "13:30:00", "16:00:00", "19:30:00", "22:00:00"]

time\_var = tk.StringVar(value=times[0])

ttk.Combobox(right, textvariable=time\_var, values=times, width=18).pack(anchor="nw")

# audi

tk.Label(right, text="Audi:", fg="white", bg="#0d0d0d").pack(anchor="nw", pady=(12,2))

audi\_var = tk.StringVar(value="1")

ttk.Combobox(right, textvariable=audi\_var, values=[str(i) for i in range(1,9)], width=18).pack(anchor="nw")

# button seat selection

tk.Button(right, text="Open Seats", bg="#ff4b4b", fg="white", width=20,

command=lambda: open\_seat\_window(movie\_id, title, section, date\_var.get(), time\_var.get(), audi\_var.get())).pack(anchor="nw", pady=18)

# ---------------- ADMIN PANEL ----------------

def open\_admin\_panel(self):

win = Toplevel(self.root)

win.title("Admin Panel")

win.geometry("700x520")

win.configure(bg="#121212")

# left: movie list

left = tk.Frame(win, bg="#121212", width=300)

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

right = tk.Frame(win, bg="#121212")

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

tk.Label(left, text="Movies (editable)", fg="white", bg="#121212", font=("Helvetica", 12, "bold")).pack(anchor="nw")

listbox = tk.Listbox(left, width=36, height=24)

listbox.pack(pady=8)

cnx = mysql.connector.connect(\*\*DB\_CONFIG)

cur = cnx.cursor()

cur.execute("SELECT id, title FROM movies")

rows = cur.fetchall()

for r in rows:

listbox.insert("end", f"{r[0]}: {r[1]}")

cur.close()

cnx.close()

# right: edit fields

tk.Label(right, text="Edit / Add Movie", fg="white", bg="#121212", font=("Helvetica", 12, "bold")).pack(anchor="nw")

f\_title = tk.Entry(right, width=48)

f\_title.pack(pady=6)

f\_desc = tk.Text(right, height=6, width=48)

f\_desc.pack(pady=6)

f\_rating = tk.Entry(right, width=20)

f\_rating.pack(pady=6)

f\_tags = tk.Entry(right, width=48)

f\_tags.pack(pady=6)

f\_section = ttk.Combobox(right, values=["Now Showing", "Upcoming"], width=20)

f\_section.pack(pady=6)

f\_release = tk.Entry(right, width=20)

f\_release.pack(pady=6)

poster\_path\_var = tk.StringVar(value="")

def choose\_poster():

p = filedialog.askopenfilename(title="Select Poster Image", filetypes=[("Images","\*.png;\*.jpg;\*.jpeg")])

if p:

rel = admin\_copy\_file\_to\_folder(p, POSTERS\_DIR)

poster\_path\_var.set(rel)

messagebox.showinfo("Poster added", f"Poster copied to {rel}")

tk.Button(right, text="Choose Poster (copy)", command=choose\_poster, bg="#2c7be5", fg="white").pack(pady=4)

tk.Label(right, textvariable=poster\_path\_var, fg="white", bg="#121212").pack()

def on\_list\_select(evt):

sel = listbox.curselection()

if not sel: return

text = listbox.get(sel[0])

mid = int(text.split(":")[0])

cnx = mysql.connector.connect(\*\*DB\_CONFIG)

cur = cnx.cursor()

cur.execute("SELECT title, description, rating, tags, poster\_path, section, release\_date FROM movies WHERE id=%s", (mid,))

r = cur.fetchone()

cur.close(); cnx.close()

if r:

f\_title.delete(0,"end"); f\_title.insert(0,r[0])

f\_desc.delete("1.0","end"); f\_desc.insert("1.0", r[1] or "")

f\_rating.delete(0,"end"); f\_rating.insert(0,r[2] or "")

f\_tags.delete(0,"end"); f\_tags.insert(0,r[3] or "")

poster\_path\_var.set(r[4] or "")

f\_section.set(r[5] or "Now Showing")

f\_release.delete(0,"end"); f\_release.insert(0, r[6] or "")

listbox.bind("<<ListboxSelect>>", on\_list\_select)

def save\_movie():

title = f\_title.get().strip()

desc = f\_desc.get("1.0","end").strip()

rating = f\_rating.get().strip()

tags = f\_tags.get().strip()

poster\_rel = poster\_path\_var.get().strip()

section = f\_section.get().strip()

release = f\_release.get().strip() or None

if not title:

messagebox.showwarning("Missing", "Title required")

return

sel = listbox.curselection()

cnx = mysql.connector.connect(\*\*DB\_CONFIG)

cur = cnx.cursor()

if sel:

mid = int(listbox.get(sel[0]).split(":")[0])

cur.execute("""UPDATE movies SET title=%s, description=%s, rating=%s, tags=%s, poster\_path=%s, section=%s, release\_date=%s WHERE id=%s""",

(title, desc, rating, tags, poster\_rel, section, release, mid))

messagebox.showinfo("Saved", "Movie updated")

else:

cur.execute("""INSERT INTO movies (title, description, rating, tags, poster\_path, section, release\_date) VALUES (%s,%s,%s,%s,%s,%s,%s)""",

(title, desc, rating, tags, poster\_rel, section, release))

messagebox.showinfo("Saved", "Movie added")

cnx.commit()

cur.close()

cnx.close()

win.destroy()

# refresh main UI

self.refresh\_main()

def delete\_movie():

sel = listbox.curselection()

if not sel:

messagebox.showwarning("Select", "Select a movie to delete")

return

mid = int(listbox.get(sel[0]).split(":")[0])

if messagebox.askyesno("Confirm", "Delete movie and all its shows/bookings?"):

cnx = mysql.connector.connect(\*\*DB\_CONFIG)

cur = cnx.cursor()

cur.execute("DELETE FROM movies WHERE id=%s", (mid,))

cnx.commit()

cur.close(); cnx.close()

win.destroy()

self.refresh\_main()

btn\_frame = tk.Frame(right, bg="#121212")

btn\_frame.pack(pady=10)

tk.Button(btn\_frame, text="Save", command=save\_movie, bg="#2ecc71", fg="white").pack(side="left", padx=6)

tk.Button(btn\_frame, text="Delete", command=delete\_movie, bg="#ff4b4b", fg="white").pack(side="left", padx=6)

tk.Button(btn\_frame, text="Close", command=win.destroy, bg="#888", fg="white").pack(side="left", padx=6)

def refresh\_main(self):

# simply redraw the inner area

for w in self.inner.winfo\_children():

w.destroy()

self.build\_sections()

# ---------------- calendar popup (section-aware) ----------------

def open\_calendar\_popup(parent, section, on\_select):

cal\_win = Toplevel(parent)

cal\_win.title("Select Date")

cal\_win.geometry("360x360")

if section == "Now Showing":

mind = date(2026,1,1); maxd = date(2026,5,31)

else:

mind = date(2026,6,1); maxd = date(2026,12,31)

cal = Calendar(cal\_win, selectmode='day', year=mind.year, month=mind.month, day=mind.day,

mindate=mind, maxdate=maxd, date\_pattern='yyyy-mm-dd')

cal.pack(pady=16)

def confirm():

on\_select(cal.get\_date())

cal\_win.destroy()

tk.Button(cal\_win, text="Confirm", command=confirm, bg="#2c7be5", fg="white").pack(pady=10)

# ---------------- seat window, bookings, receipts ----------------

def open\_seat\_window(movie\_id, title, section, chosen\_date, chosen\_time, audi):

if not chosen\_date or chosen\_date == "Select date":

messagebox.showwarning("No date", "Please select a date first.")

return

# create/find show & seats

cnx = mysql.connector.connect(\*\*DB\_CONFIG)

cur = cnx.cursor()

cur.execute("SELECT id FROM shows WHERE movie\_id=%s AND show\_date=%s AND show\_time=%s AND audi=%s", (movie\_id, chosen\_date, chosen\_time, int(audi)))

r = cur.fetchone()

if r:

show\_id = r[0]

else:

cur.execute("INSERT INTO shows (movie\_id, show\_date, show\_time, audi) VALUES (%s,%s,%s,%s)", (movie\_id, chosen\_date, chosen\_time, int(audi)))

cnx.commit()

show\_id = cur.lastrowid

for i in range(1,21):

cur.execute("INSERT INTO seats (show\_id, seat\_no, is\_booked) VALUES (%s,%s,%s)", (show\_id, f"S{i:02}", 0))

cnx.commit()

cur.execute("SELECT seat\_no, is\_booked FROM seats WHERE show\_id=%s ORDER BY id", (show\_id,))

seat\_rows = cur.fetchall()

cur.close(); cnx.close()

# UI for seats

win = Toplevel()

win.title(f"Seats — {title}")

win.geometry("720x560")

win.configure(bg="#0f0f0f")

tk.Label(win, text=f"{title}", font=("Helvetica", 16, "bold"), fg="white", bg="#0f0f0f").pack(pady=8)

tk.Label(win, text=f"{chosen\_date} {chosen\_time} Audi {audi}", fg="#ccc", bg="#0f0f0f").pack()

gridf = tk.Frame(win, bg="#0f0f0f")

gridf.pack(pady=12)

buttons = {}

selected = set()

def toggle(seat\_no):

cur\_btn = buttons[seat\_no]

if cur\_btn["bg"] == "red":

return

if seat\_no in selected:

cur\_btn.config(bg="#333")

selected.remove(seat\_no)

else:

cur\_btn.config(bg="#2ecc71")

selected.add(seat\_no)

r = 0; c = 0

for seat\_no, is\_booked in seat\_rows:

color = "red" if is\_booked else "#333"

b = tk.Button(gridf, text=f"{seat\_no}\n🛋", width=8, height=3, bg=color, fg="white",

font=("Arial", 10), command=lambda s=seat\_no: toggle(s))

b.grid(row=r, column=c, padx=10, pady=8)

buttons[seat\_no] = b

c += 1

if c >= 5:

c = 0; r += 1

def confirm():

if not selected:

messagebox.showwarning("No seats", "Select at least one seat.")

return

cnx = mysql.connector.connect(\*\*DB\_CONFIG)

cur = cnx.cursor()

booked = []

for s in selected:

cur.execute("SELECT is\_booked FROM seats WHERE show\_id=%s AND seat\_no=%s FOR UPDATE", (show\_id, s))

curval = cur.fetchone()

if curval and curval[0] == 1:

messagebox.showwarning("Taken", f"Seat {s} already booked.")

continue

cur.execute("UPDATE seats SET is\_booked=1 WHERE show\_id=%s AND seat\_no=%s", (show\_id, s))

cur.execute("INSERT INTO bookings (show\_id, seat\_no) VALUES (%s,%s)", (show\_id, s))

booked.append(s)

cnx.commit()

# get last inserted booking id approx (mysql doesn't give a batch lastid easily) - we'll use max id of bookings for receipt id

cur.execute("SELECT MAX(id) FROM bookings")

last\_booking\_id = cur.fetchone()[0] or 0

cur.close(); cnx.close()

if booked:

messagebox.showinfo("Booked", f"Booked seats: {', '.join(booked)}")

# show receipt with QR

receipt\_img = generate\_ticket\_image(title, chosen\_date, chosen\_time, audi, booked, last\_booking\_id)

show\_receipt\_window(receipt\_img)

win.destroy()

# refresh main UI to show booked seats reflected next time

app.refresh\_main()

else:

messagebox.showwarning("No seats", "No seats booked (maybe already taken).")

tk.Button(win, text="Confirm Booking", bg="#ff4b4b", fg="white", command=confirm).pack(pady=12)

# ---------------- receipt display/save ----------------

def show\_receipt\_window(img\_pil):

win = Toplevel()

win.title("Ticket Receipt")

win.geometry("820x560")

win.configure(bg="#222")

tk.Label(win, text="Ticket Receipt", fg="white", bg="#222", font=("Helvetica", 16, "bold")).pack(pady=8)

tk\_img = ImageTk.PhotoImage(img\_pil)

lbl = tk.Label(win, image=tk\_img, bg="#222")

lbl.image = tk\_img

lbl.pack()

def save\_png():

fname = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG Image","\*.png")], title="Save Ticket")

if fname:

img\_pil.save(fname)

messagebox.showinfo("Saved", f"Ticket saved to {fname}")

tk.Button(win, text="Save Ticket as PNG", bg="#2ecc71", fg="white", command=save\_png).pack(pady=10)

# ---------------- run initial setup and start app ----------------

if \_\_name\_\_ == "\_\_main\_\_":

try:

ensure\_database\_and\_tables()

seed\_sample\_movies() # This will now CLEAR and RESEED the database

except Exception as e:

print("DB setup error:", e)

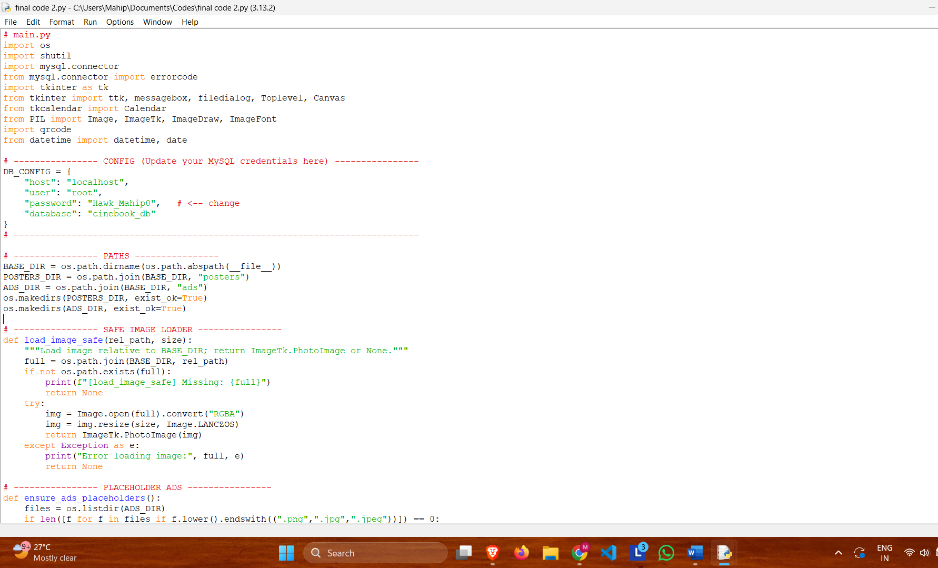
raise

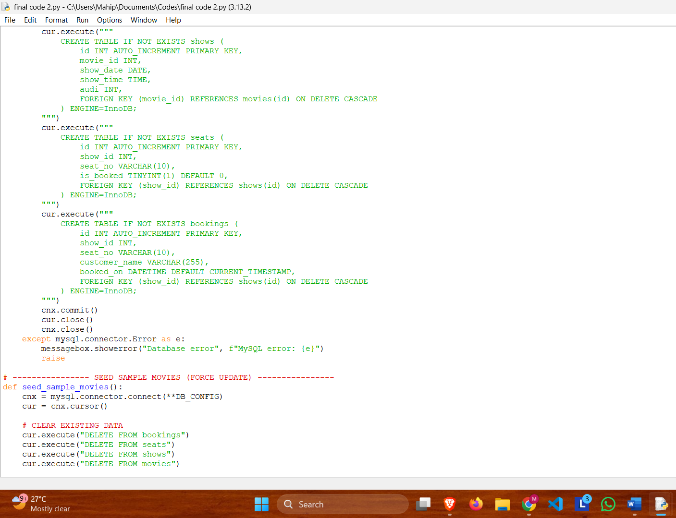
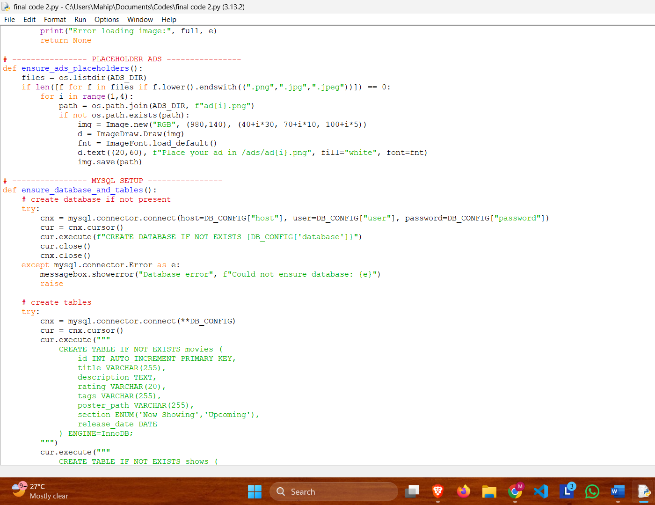
ensure\_ads\_placeholders()

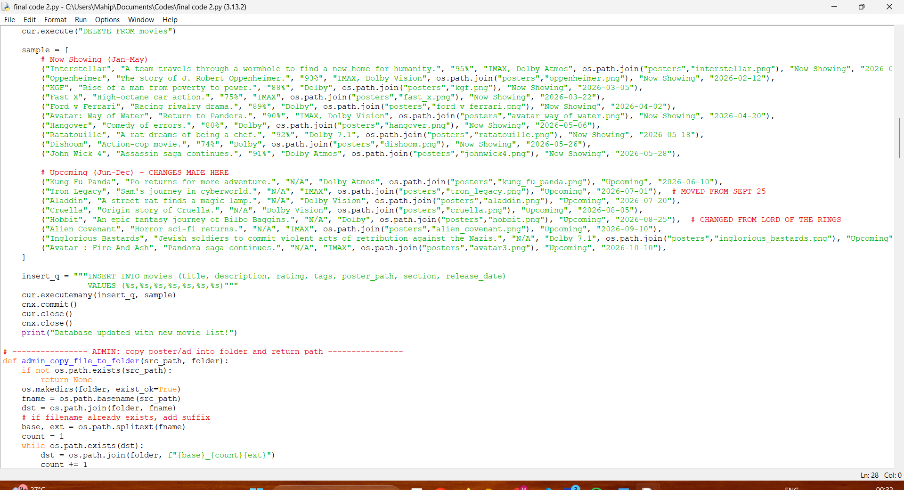
root = tk.Tk()

app = CineBookApp(root)

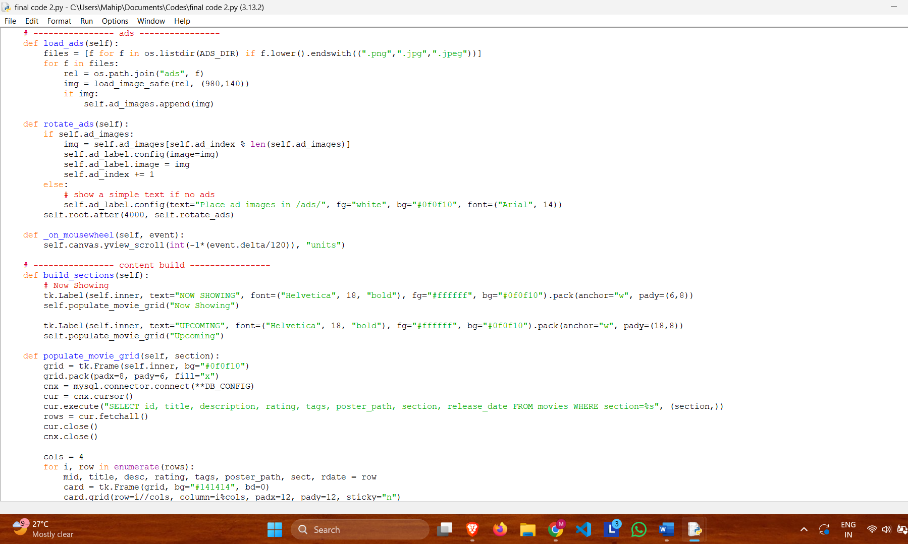
root.mainloop()

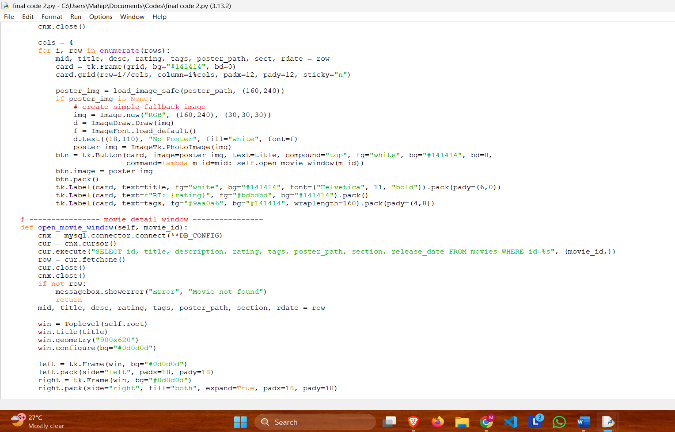


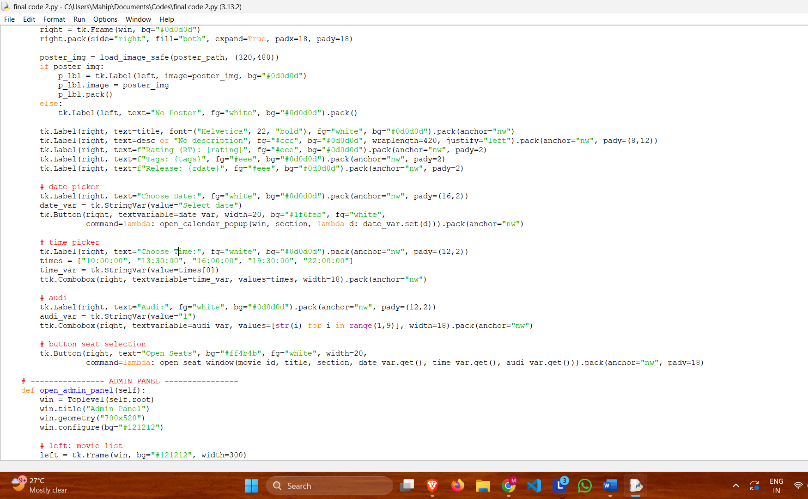


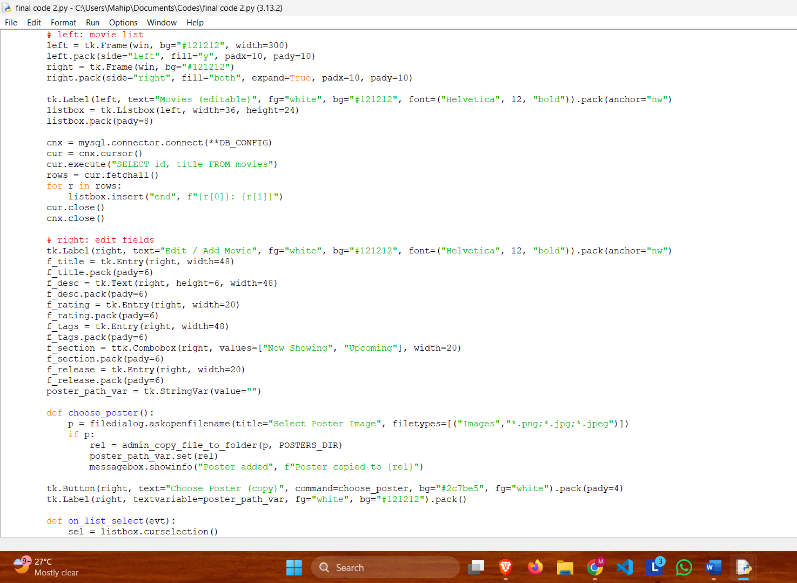


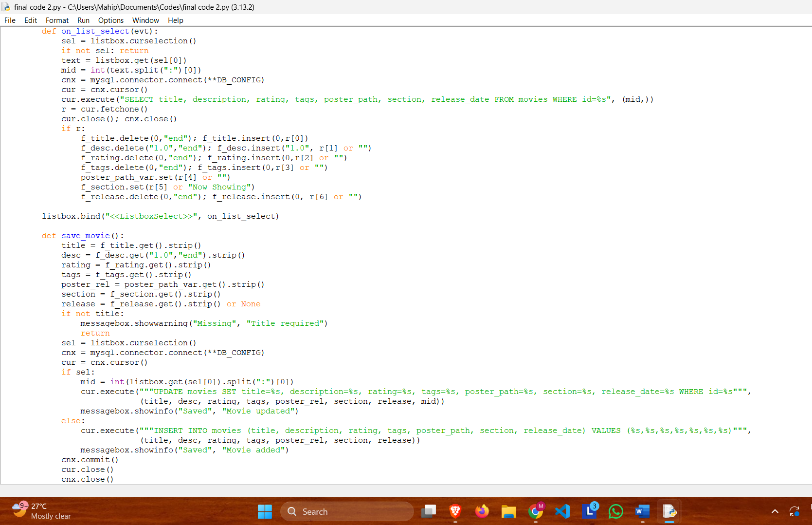


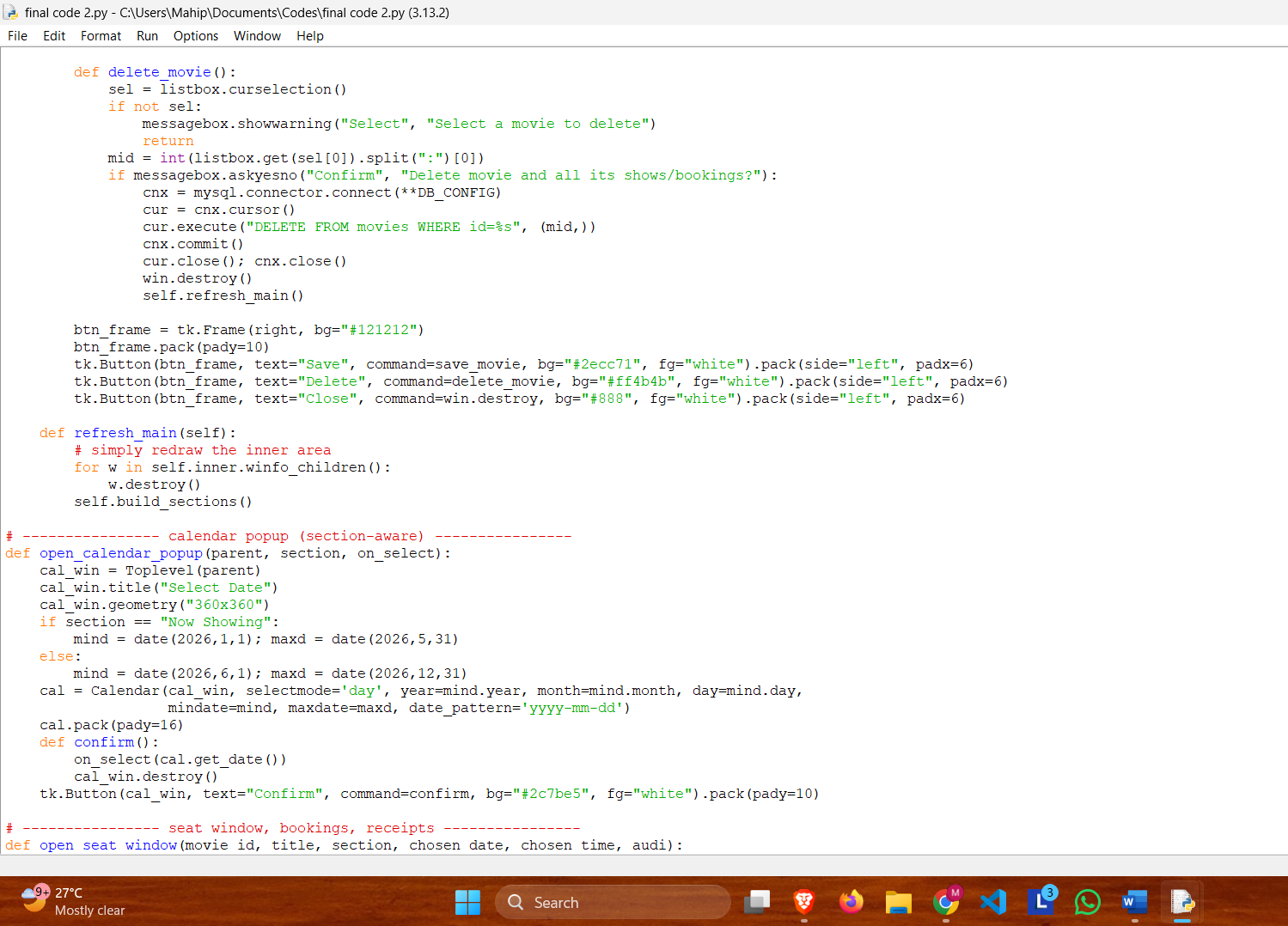


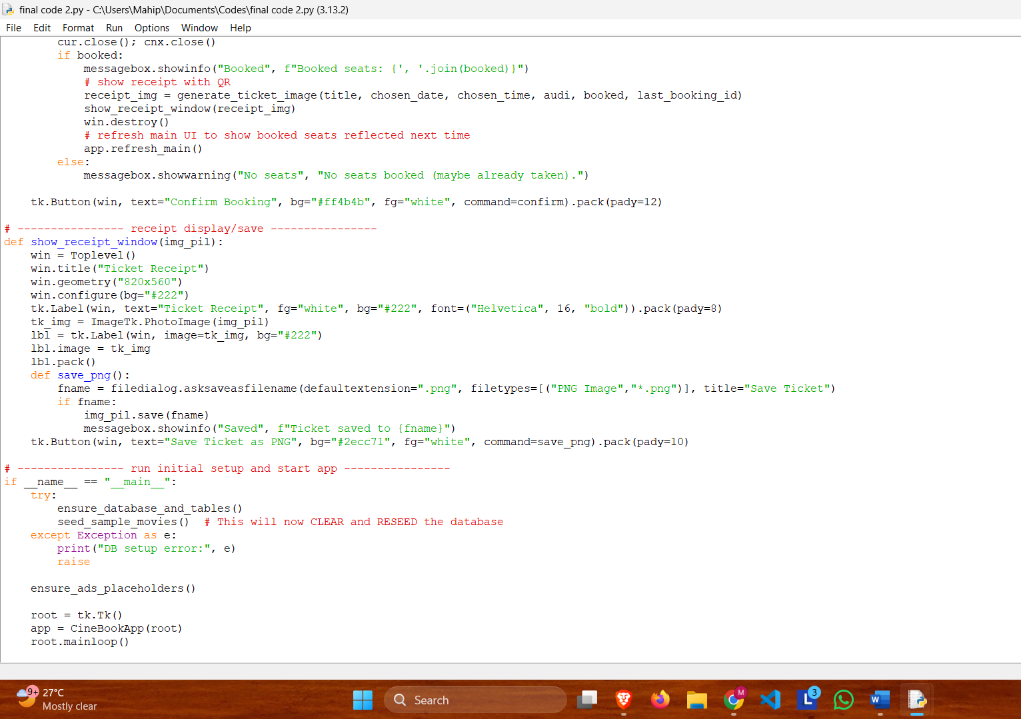




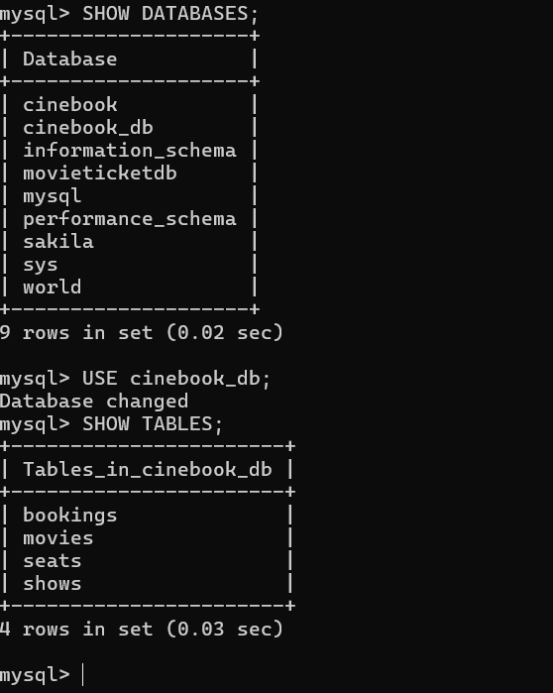
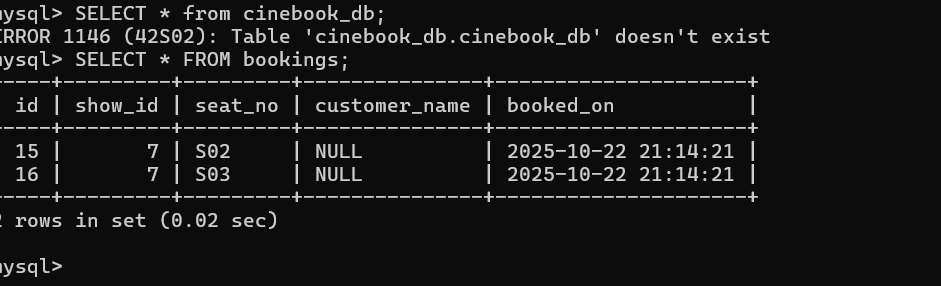


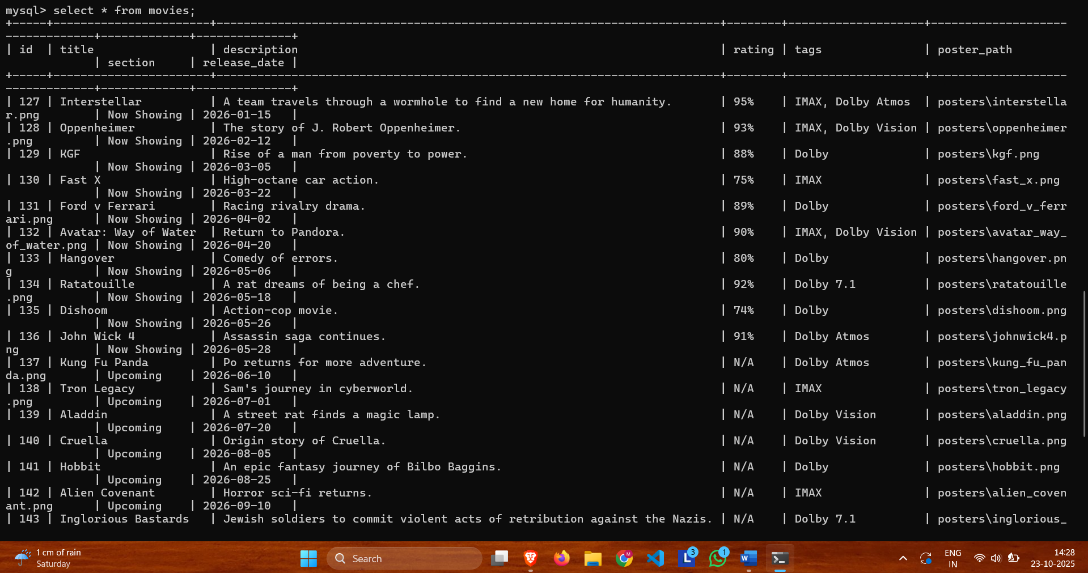


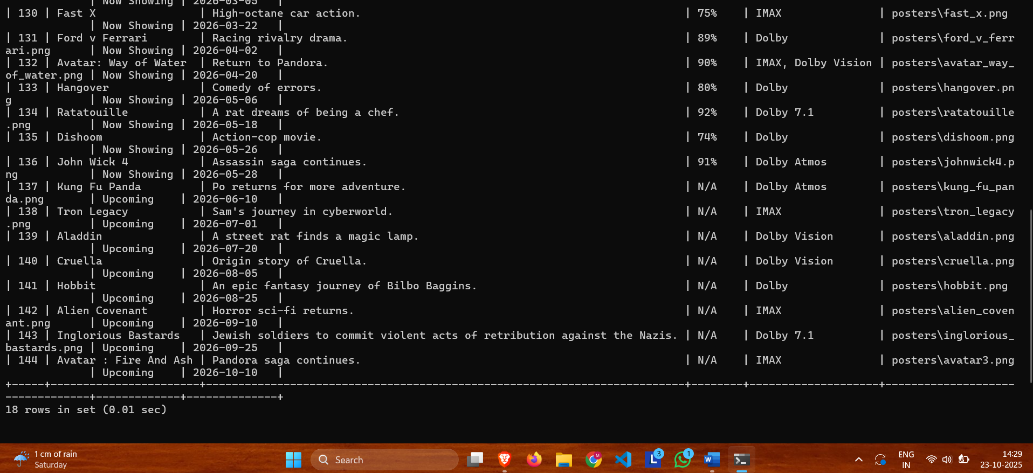


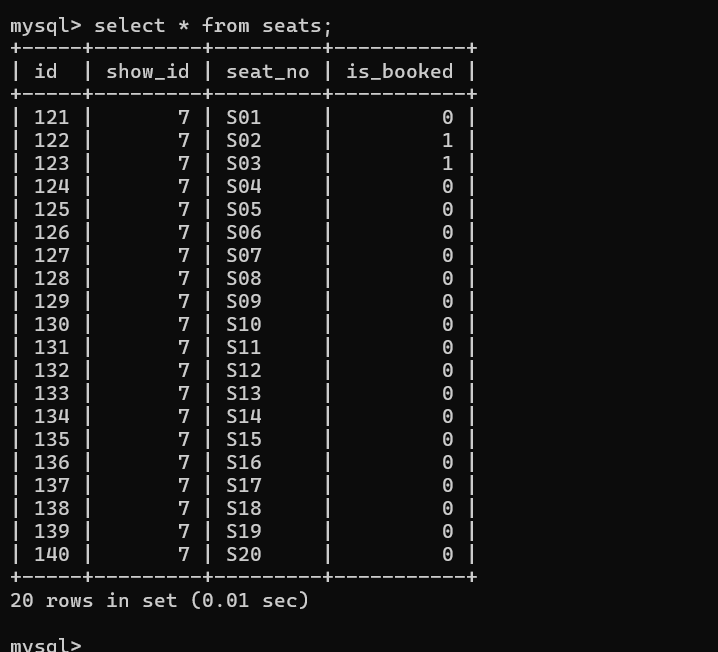


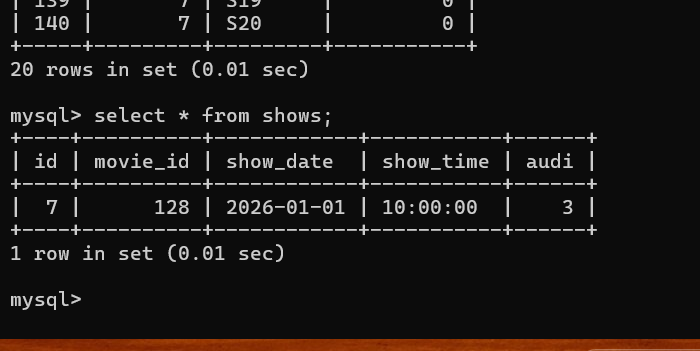
**MYSQL DATABASE**

** **

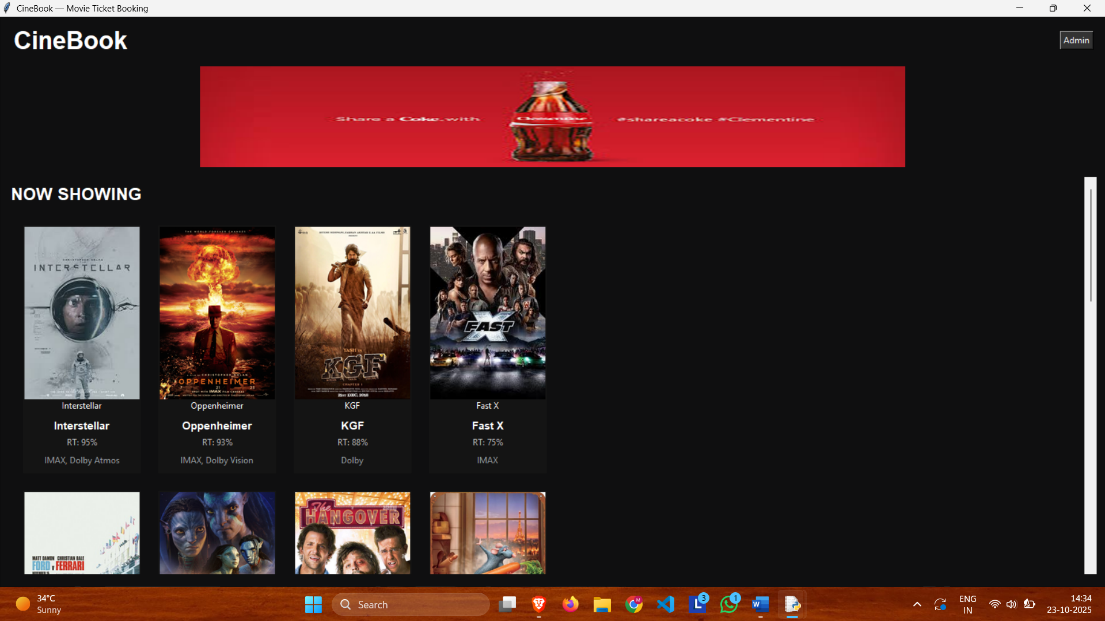
****

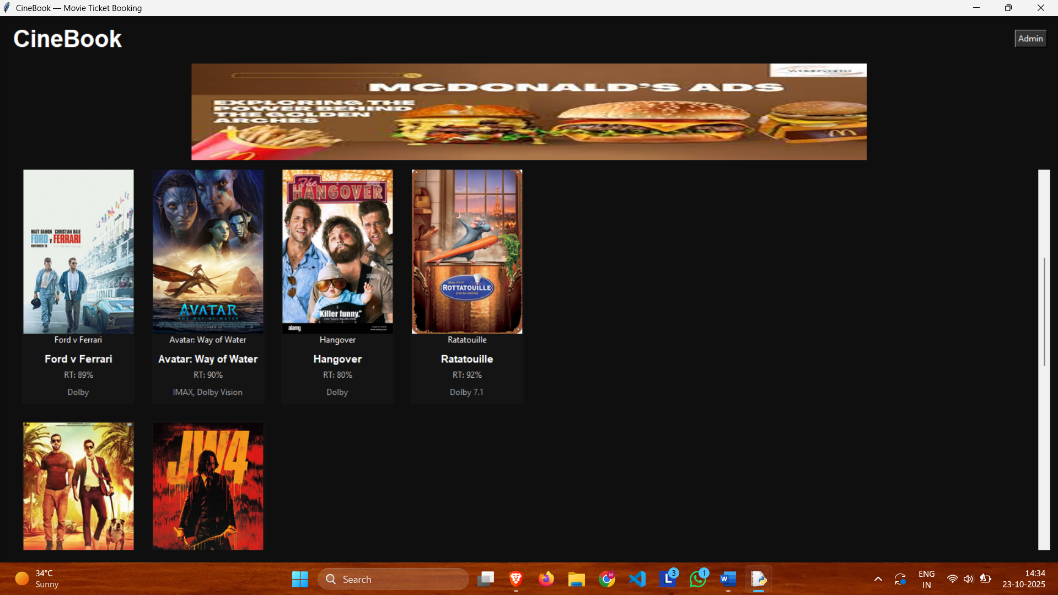
****

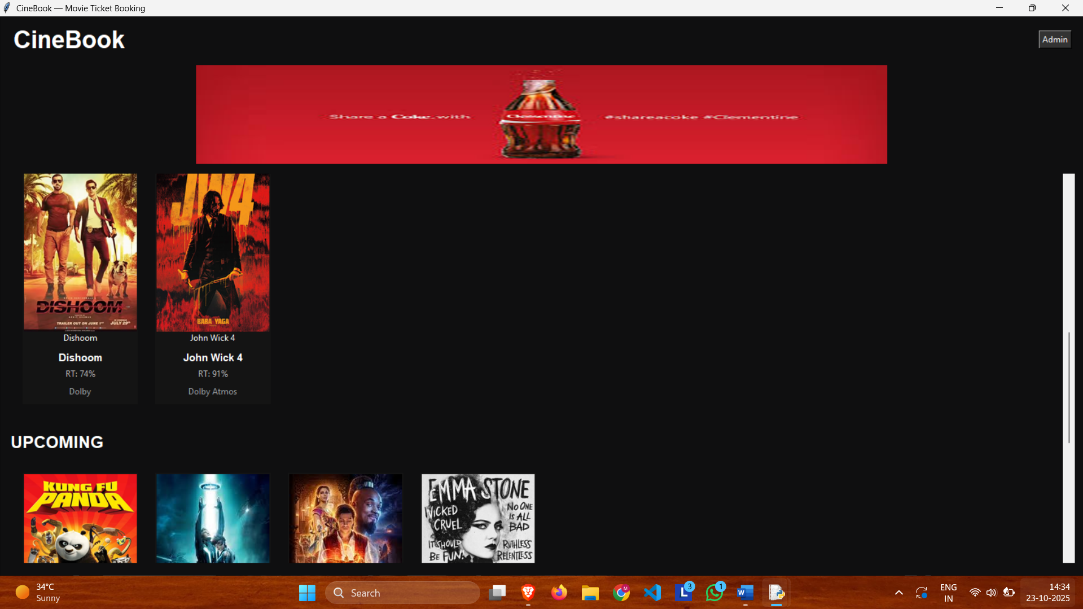
****

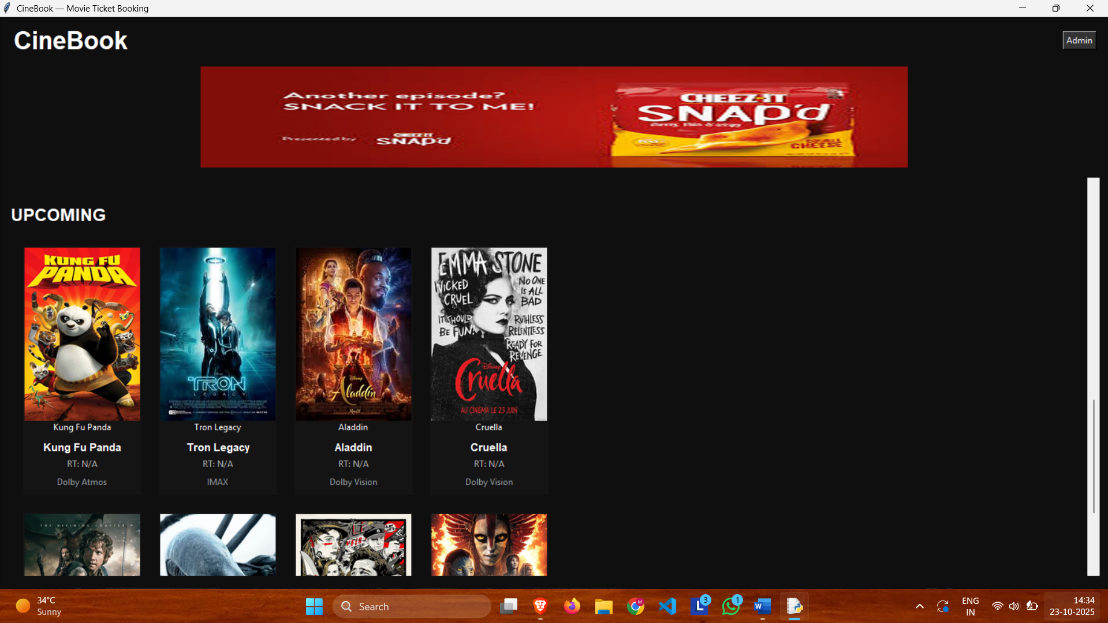
****

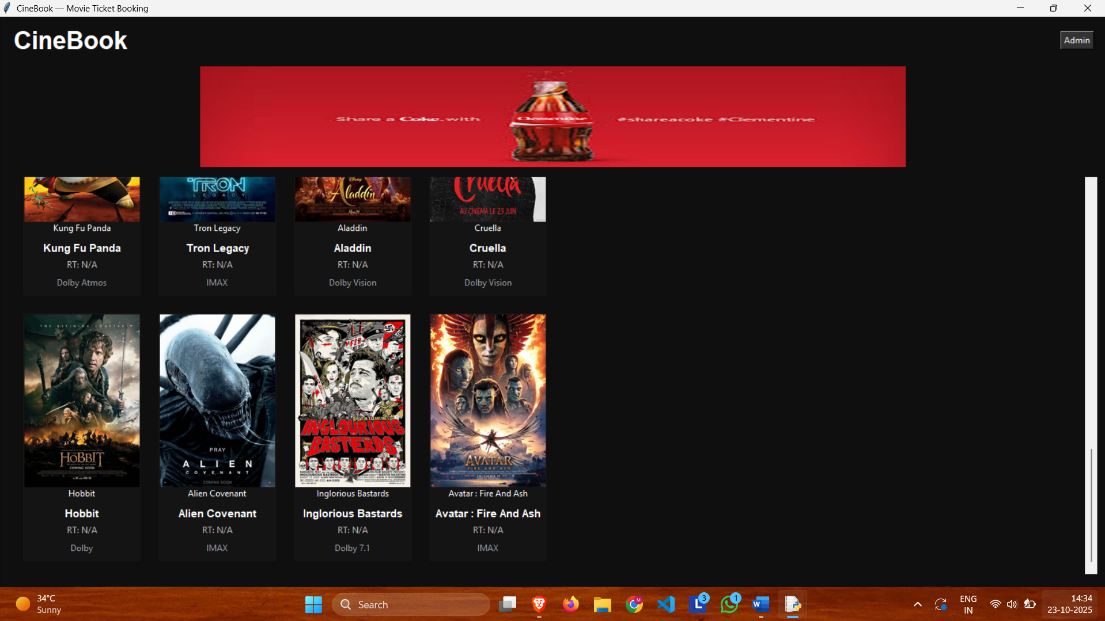
OUTPUT

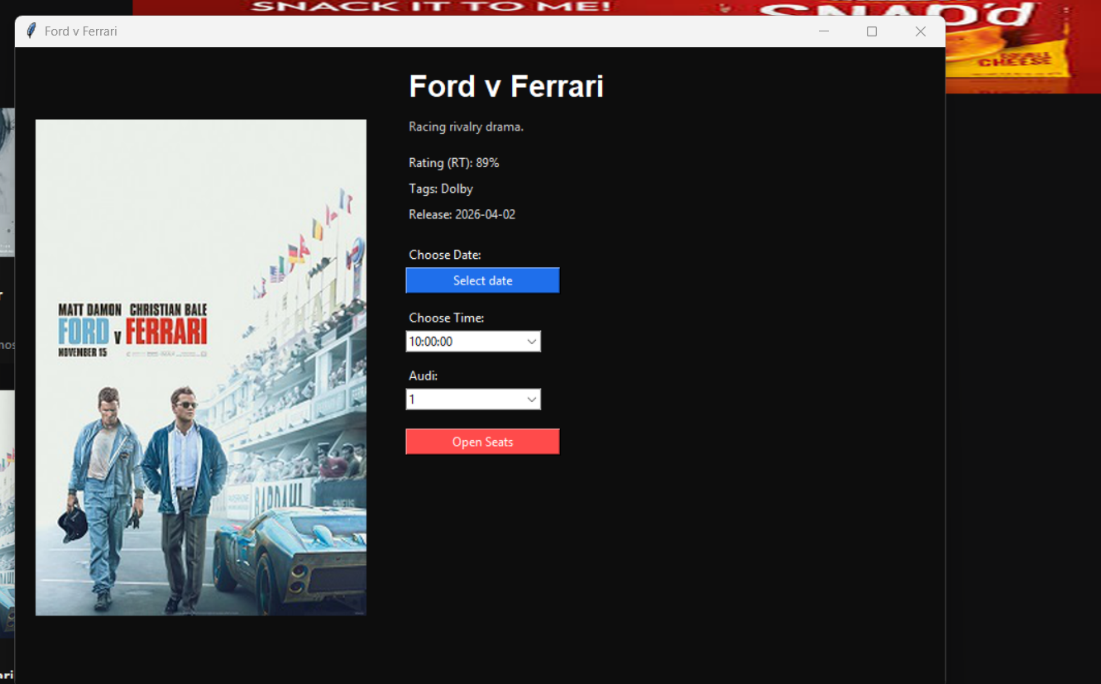


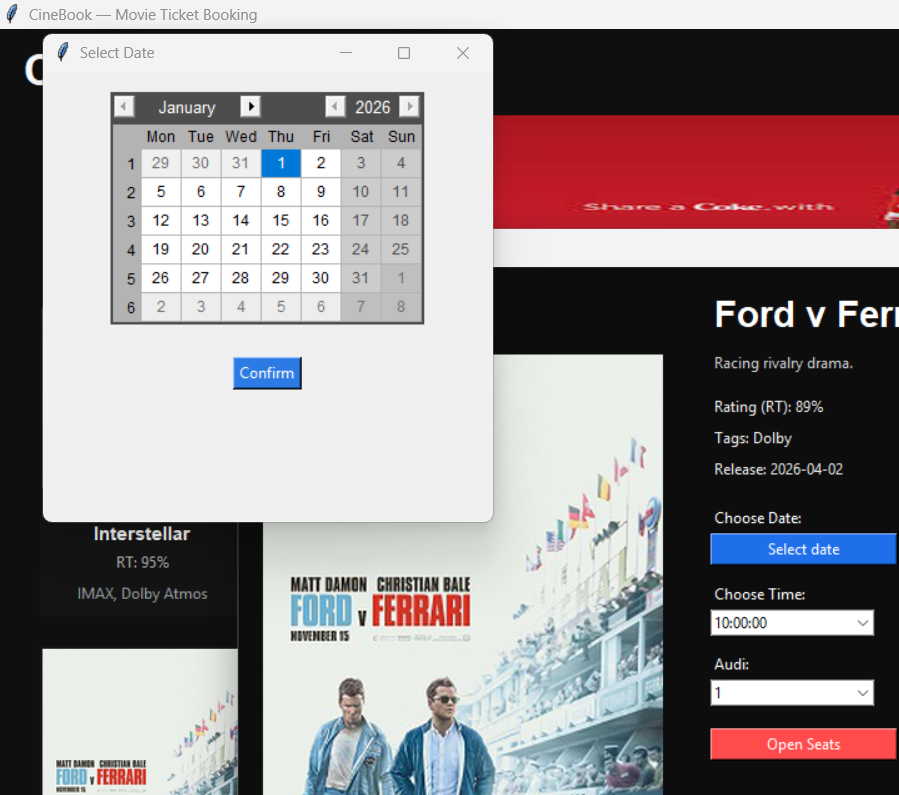


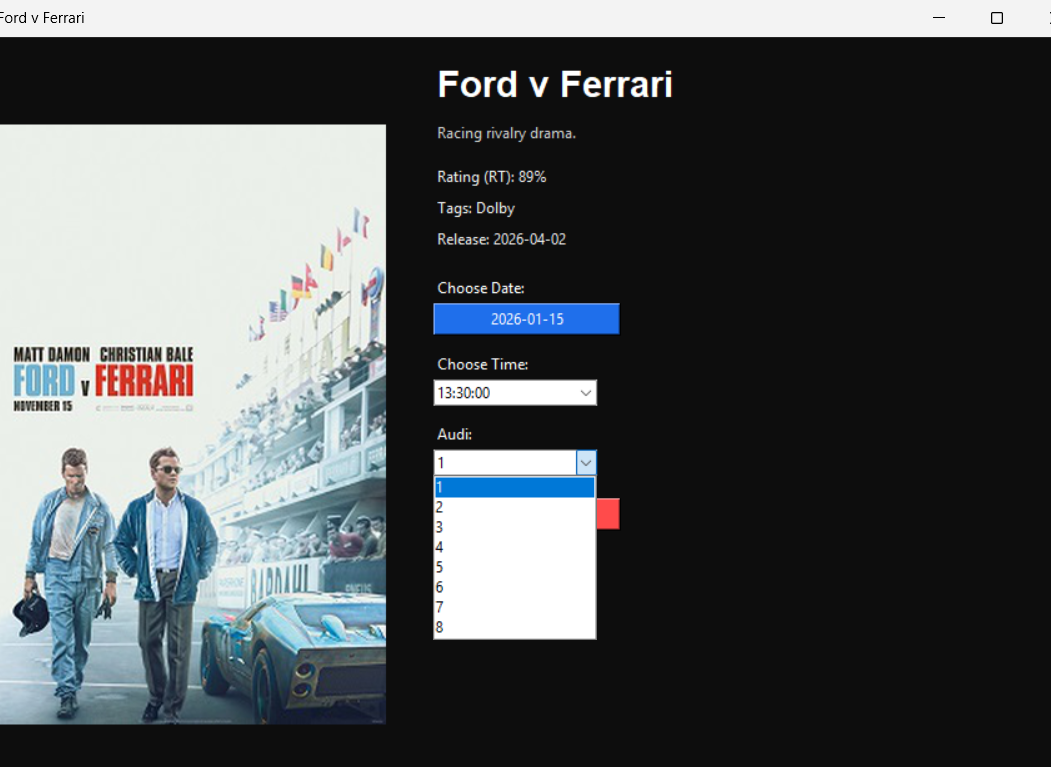




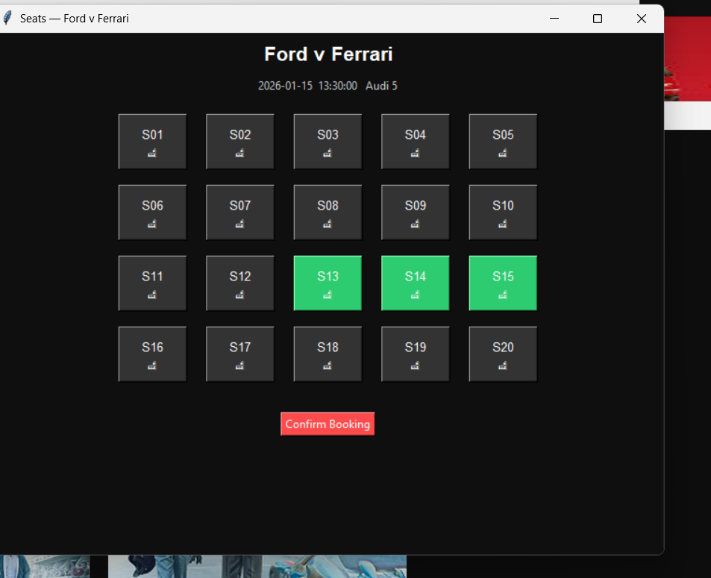


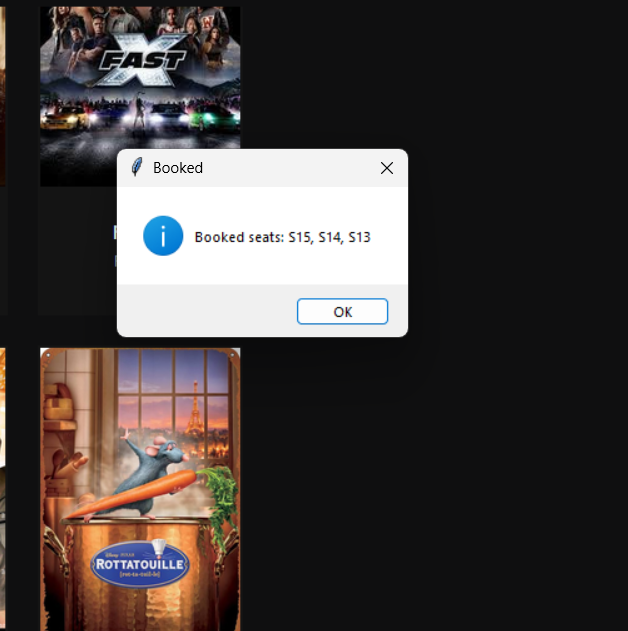


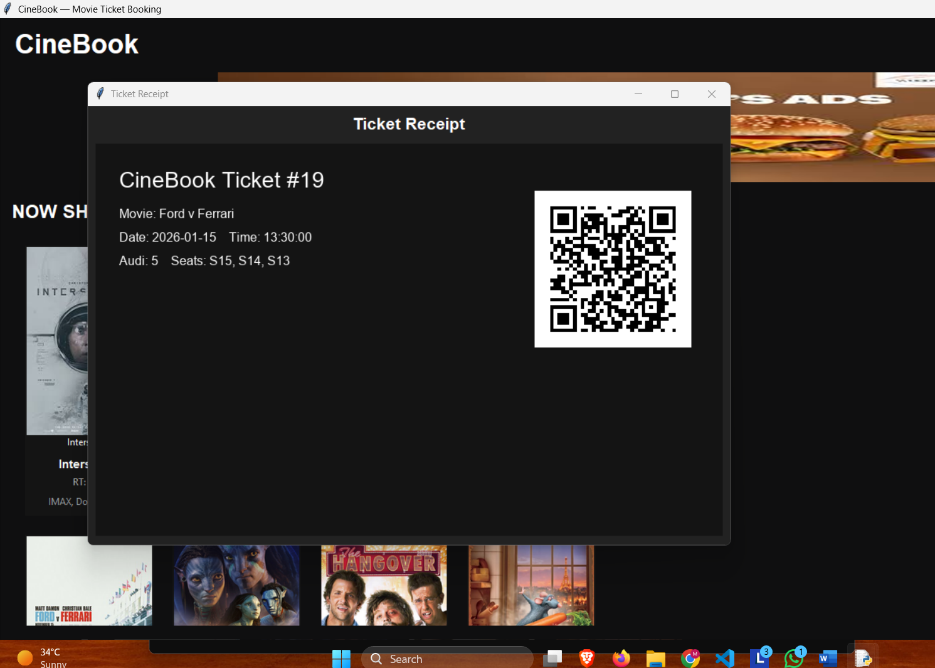












BIBLIOGRAPHY

* 1. GeeksForGeeks
  2. LeetCode
  3. The Big Book of Small Projects
  4. Python Crash Course: Handbook