Mangalore Institute of Technology and Engineering (MITE)

ISO 9001:2008 certification

Mangalore Institute of Technology and Engineering Moodbidri-574225

MITE

Invent Solutions

"Online Movie Ticket Booking System"

Online Movie Ticket Booking System Mini project report prescribed for the Database Management Systems (18CS53) course in 5th Semester in

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

By

Naveenkumar S R
(4MT18CS055)

Anoop UV Bhat
(4MT18CS007)

Under the Guidance of

Mr. Guru Prasad
Assistant Professor,
Dept. Of CSE,
MITE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JANUARY 2021

1. INTRODUCTION

Welcome to newly designed movie ticket booking System is a faster, cleaner and a tad more personal GUI, specially designed to make your booking experience better. Log on, navigate and find out for yourselves and if time permits leave your valuable feedback. Customers may view the contents of any movie show at any time and may book any movie ticket as needed. The program automatically calculates the subtotal and grand total. When a visitor decides to finally book the ticket, the order information including the buyer's name, address and billing instruction is stored in the database securely and payment has been made. The combo booking is also provided at the time of booking the ticket and there's a wonderful facility of delivering the combos at your seat when you are watching the movie.

2.About mini project

The project objective is to book cinema tickets in online. The Ticket Reservation System is an Internet based application that can be accessed throughout the Net and can be accessed by anyone who has a net connection. This application will reserve the tickets. This online ticket reservation system provides a website for a cinema hall where any user of internet can access it. User is required to login to the system and needs a credit card for booking the tickets. Tickets can be collected at the counter and Watching movies with family and friends in theatres is one of the best medium of entertainment after having a hectic schedule. But all this excitement vanishes after standing in hours in long queues to get tickets booked. The website provides complete information regarding currently running movies on all the screens with details of show timings, available seats. Ticket reservations are done using credit card and can be cancelled if needed. Our online tickets reservation system is one of the best opportunities for those who cannot afford enough time to get their tickets reserved standing in long queues. People can book tickets online at any time of day or night. Our reservation system also provides option to cancel the tickets which are reserved previously.

3.Technology Used

3.1 SYSTEM REQUIREMENT

Processor-8th generation Core i5
Platform - Windows 10
Visual Studio Code
Front end – Python (Tkinter module)
Backend – SQLite

3.2 Front End

Python: Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was created in the late 1980s, and first released in 1991, by Guido van Rossum as a successor to the ABC programming language. Python 2.0, released in 2000, introduced new features, such as list comprehensions, and a garbage collection system with reference counting, and was discontinued with version 2.7 in 2020.[30] Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible and much Python 2 code does not run unmodified on Python 3. With Python 2's end-of-life, only Python 3.6.x[31] and later are supported, with older versions still supporting e.g., Windows 7 (and old installers not restricted to 64-bit Windows).

Python interpreters are supported for mainstream operating systems and available for a few more (and in the past supported many more). A global

community of programmers develops and maintains CPython, a free and open-source [32] reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development

3.3 Tkinter

Graphical User Interface (GUI) is a form of user interface which allows users to interact with computers through visual indicators using items such as icons, menus, windows, etc. It has advantages over the Command Line Interface (CLI) where users interact with computers by writing commands using keyboard only and whose usage is more difficult than GUI.

Tkinter is the inbuilt python module that is used to **create GUI** applications. It is one of the most commonly used modules for creating GUI applications in Python as it is simple and easy to work with. You don't need to worry about the installation of the Tkinter module separately as it comes with Python already. It gives an object-oriented interface to the Tk GUI toolkit.

Some other Python Libraries available for creating our own GUI applications are

- Kivy
- Python Qt
- wxPython

Among all **Tkinter** is most widely used

In Python, Tkinter is a standard GUI (graphical user interface) package. Tkinter is Python's default GUI module and also the most common way that is used for GUI programming in Python. Note that Tkinter is a set of wrappers that implement the Tk widgets as Python classes.

Tkinter in Python helps in **creating GUI** Applications with a minimum hassle. Among various GUI Frameworks, Tkinter is the only framework that is built-in into **Python's Standard Library.**

- An important feature in favour of Tkinter is that it is cross-platform, so the same code can easily work on **Windows**, **macOS**, and **Linux.**

- Tkinter is a lightweight module.

- It is simple to use.

3.4 Back End (SQLite) Database

About SQLite: SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is a database, which is zero-configured, which means like other databases you do not need to configure it in your system.

SQLite engine is not a standalone process like other databases, you can link it statically or dynamically as per your requirement with your application. SQLite accesses its storage files directly

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is one of the fastest-growing database engines around, but that's growth in terms of popularity, not anything to do with its size. The source code for SQLite is in the public domain.

Why SQLite?

- SQLite does not require a separate server process or system to operate (serverless).

- SQLite comes with zero-configuration, which means no setup or administration needed.

- A complete SQLite database is stored in a single cross-platform disk file.

- SQLite is very small and light weight, less than 400KiB fully configured or less than 250KiB with optional features omitted.
- SQLite is self-contained, which means no external dependencies.
- SQLite transactions are fully ACID-compliant, allowing safe access from multiple processes or threads.
- SQLite supports most of the query language features found in SQL92 (SQL2) standard.
- SQLite is written in ANSI-C and provides simple and easy-to-use API.
- SQLite is available on UNIX (Linux, Mac OS-X, Android, iOS) and Windows (Win32, WinCE, WinRT).

- SQLite is very small and light weight, less than 400KiB fully configured or less than 250KiB with optional features omitted.
- SQLite is self-contained, which means no external dependencies.
- SQLite transactions are fully ACID-compliant, allowing safe access from multiple processes or threads.
- SQLite supports most of the query language features found in SQL92 (SQL2) standard.
- SQLite is written in ANSI-C and provides simple and easy-to-use API.
- SQLite is available on UNIX (Linux, Mac OS-X, Android, iOS) and Windows (Win32, WinCE, WinRT).

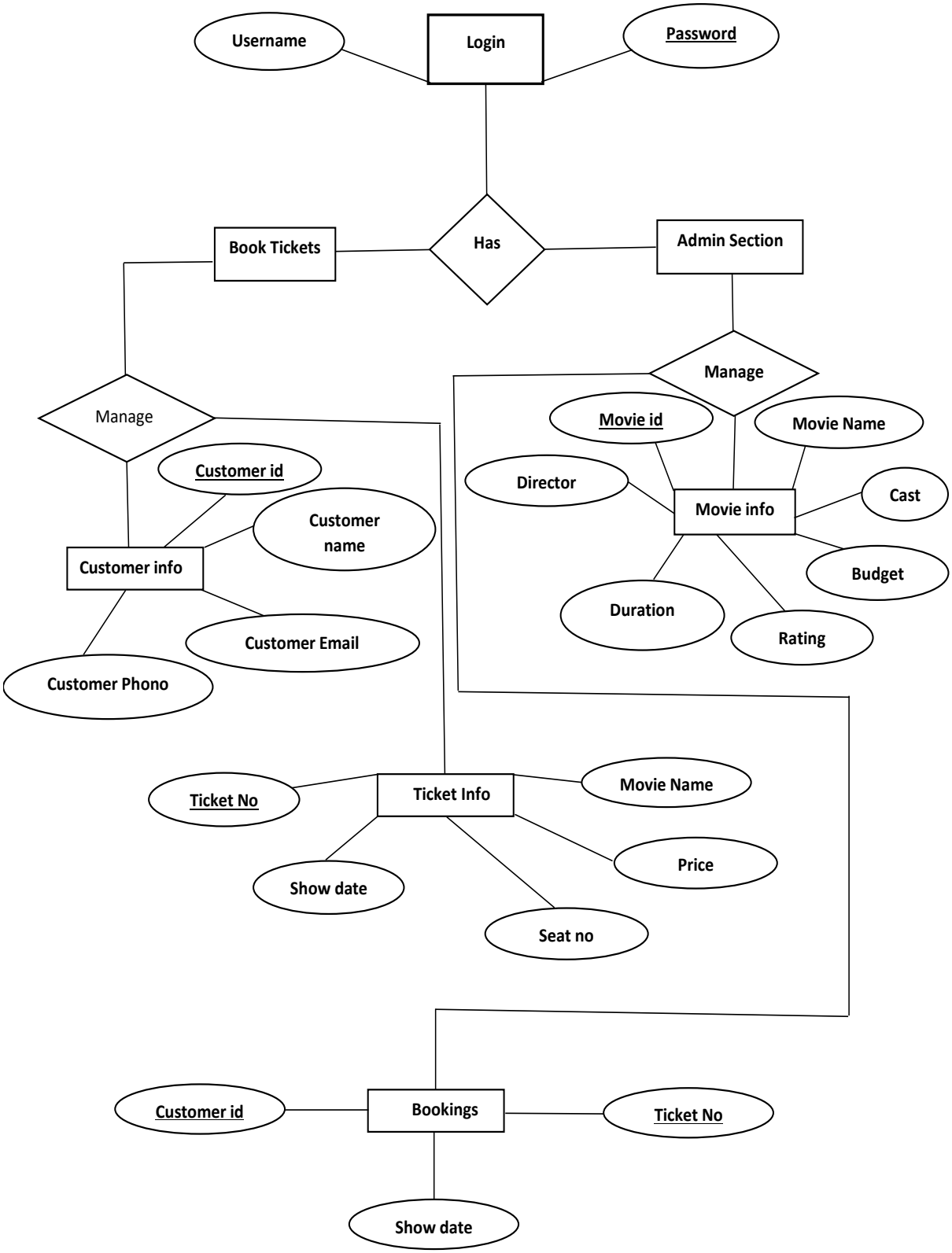
SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. SQLite database files are a recommended storage format by the US Library of Congress. Think of SQLite not as a replacement for Oracle but as a replacement for fopen()

SQLite is a compact library. With all features enabled, the library size can be less than 600KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments. Depending on how it is used, SQLite can be faster than direct filesystem I/O.
SQLite is very carefully tested prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification. An automated test suite runs millions and millions of test cases involving hundreds of millions of individual SQL statements and achieves 100% branch test coverage. SQLite responds gracefully to memory allocation failures and disk I/O errors. Transactions are ACID even if interrupted by system crashes or power failures.

4.Design

4.1 ER Diagram



4.2 Decrepitation Of ER Diagram

This ER (Entity Relationship) Diagram represent the model of Movie Ticket Booking System Entity. The entity-relationship diagram of Movie Ticket Booking System Show all the visual instrument of database tables and the relation between Customer, Payment, Movie, Shows etc. It used structure data and to define the relationships between structured data groups of Movie Ticket Booking System functionalities. The main entities of the Movie Ticket Booking System are Movie, Customer, Payments, seats and Shows

Movie Ticket Booking System entities and their attributes:

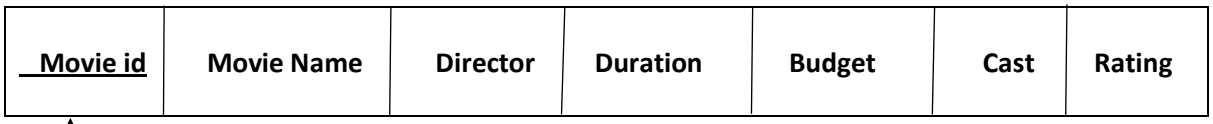
- **Movie info Entity:** Attributes of Movie are Movie_id, Movie_name, Director, Duration, Budget, Cast, Rating
- **Customer info Entity:** Attributes of Customer are Customer id Customer Name, Customer Email, Customer phono
- **Ticket info Entity:** Attributes of Ticket info are Ticket No, Movie_name, Price, Seat No, Show date
- **Booking Entity:** Attributes of Booking are Ticket No, Customer id Show date

Description of Movie Ticket Booking System Database:

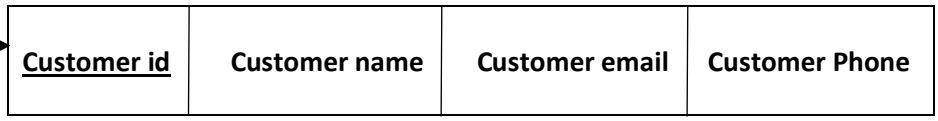
- The details of Movie are store into the Movie tables respective with all tables
- Each entity (Movie_id, Customer id, Ticket no) contain primary key and unique keys.
- The entity Booking, Ticket has binded with Movie, Customer entity with foreign key
- There is one-to-one relationship available between Tickets, Movie info
- All the entity Movie info, Ticket info, Booking are normalized and reduce delicacy of record.
- We have implemented indexing on each tables of Movie Ticket Booking System table for fast query execution.

4.3 Relation Schema Diagram:

Movie info:

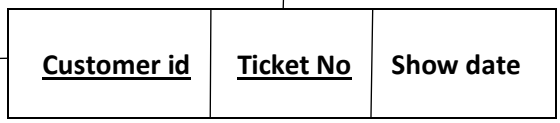


Customer info:



Ticket info:

Booking:



5.Implementation (Coding part)

5.1 Home Page Part:

```
from tkinter import *
import tkinter.messagebox
import database
import first
import booking


class Movie:



    def __init__(self, root):
        d = database.Backend()
        #d.conn


        self.root=root
        self.root.title('Movie Ticket Booking System')
        self.root.geometry('1350x750')
        self.root.config(bg="black")

        Movie_Name=StringVar()
        Movie_ID=StringVar()
        Release_Date=StringVar()
        Director=StringVar()
        Cast=StringVar()
        Budget=StringVar()
        Duration=StringVar()
        Rating=StringVar()


        def clcdata():
            self.txtMovie_ID.delete(0,END)
            self.txtMovie_Name.delete(0,END)
            self.txtRelease_Date.delete(0,END)
            self.txtDirector.delete(0,END)
            self.txtCast.delete(0,END)
```

```
    #labels
    self.lblMovie_ID=Label(LeftBody, font=('Arial', 18, 'bold'), text="Movie ID:", padx=2, pady=2, bg="black", fg="orange")
    self.lblMovie_ID.grid(row=0, column=0, sticky=W)
    self.txtMovie_ID=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Movie_ID, width=39, bg="black", fg="white")
    self.txtMovie_ID.grid(row=0, column=1)

    self.lblMovie_Name=Label(LeftBody, font=('Arial', 18, 'bold'), text="Movie Name:", padx=2, pady=2, bg="black", fg="orange")
    self.lblMovie_Name.grid(row=1, column=0, sticky=W)
    self.txtMovie_Name=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Movie_Name, width=39, bg="black", fg="white")
    self.txtMovie_Name.grid(row=1, column=1)

    self.lblRelease_Date=Label(LeftBody, font=('Arial', 18, 'bold'), text="Release Date:", padx=2, pady=2, bg="black", fg="orange")
    self.lblRelease_Date.grid(row=2, column=0, sticky=W)
    self.txtRelease_Date=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Release_Date, width=39, bg="black", fg="white")
    self.txtRelease_Date.grid(row=2, column=1)

    self.lblDirector=Label(LeftBody, font=('Arial', 18, 'bold'), text="Director:", padx=2, pady=2, bg="black", fg="orange")
    self.lblDirector.grid(row=3, column=0, sticky=W)
    self.txtDirector=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Director, width=39, bg="black", fg="white")
    self.txtDirector.grid(row=3, column=1)

    self.lblCast=Label(LeftBody, font=('Arial', 18, 'bold'), text="Cast:", padx=2, pady=2, bg="black", fg="orange")
    self.lblCast.grid(row=4, column=0, sticky=W)
    self.txtCast=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Cast, width=39, bg="black", fg="white")
    self.txtCast.grid(row=4, column=1)

    self.lblBudget=Label(LeftBody, font=('Arial', 18, 'bold'), text="Budget (Crores INR):", padx=2, pady=2, bg="black", fg="orange")
    self.lblBudget.grid(row=5, column=0, sticky=W)
    self.txtBudget=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Budget, width=39, bg="black", fg="white")
    self.txtBudget.grid(row=5, column=1)

    self.lblDuration=Label(LeftBody, font=('Arial', 18, 'bold'), text="Duration (Hrs):", padx=2, pady=2, bg="black", fg="orange")
    self.lblDuration.grid(row=6, column=0, sticky=W)
    self.txtDuration=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Duration, width=39, bg="black", fg="white")
    self.txtDuration.grid(row=6, column=1)
```

5.2 Admin Part:

```
        self.txtRating.delete(0,END)
        self.txtDuration.delete(0,END)

    def addData():
        if (len(Movie_ID.get())!=0):
            d.addMovie(Movie_ID.get(),Movie_Name.get(),Release_Date.get(),Director.get(),Cast.get(),Budget.get(),Duration.get(),Rating.get())
            MovieList.delete(0,END)
            MovieList.insert(Movie_ID.get(),Movie_Name.get(),Release_Date.get(),Director.get(),Cast.get(),Budget.get(),Duration.get(),Rating.get())
            disData()
        else:
            tkinter.messagebox.askyesno('Enter a Movie ID')

    def disData():
        MovieList.delete(0,END)
        for row in d.ViewMovieData():
            MovieList.insert(END,row,str(''))

    def delData():
        if (len(Movie_ID.get())!=0):
            d.DeleteMovieRec(sd[0])
            clcdata()
            disData()

    def movieRec(event):
        global sd
        searchmovie=MovieList.curselection()[0]
        sd=MovieList.get(searchmovie)
        self.txtMovie_ID.delete(0,END)
        self.txtMovie_ID.insert(END,sd[0])
        self.txtMovie_Name.delete(0,END)
        self.txtMovie_Name.insert(END,sd[1])
        self.txtRelease_Date.delete(0,END)
        self.txtRelease_Date.insert(END,sd[2])
        self.txtDirector.delete(0,END)
        self.txtDirector.insert(END,sd[3])
        self.txtCast.delete(0,END)
        self.txtCast.insert(END,sd[4])
        self.txtBudget.delete(0,END)
```

```
        #labels
        self.lblMovie_ID=Label(LeftBody, font=('Arial', 18, 'bold'), text="Movie ID:", padx=2, pady=2, bg="black", fg="orange")
        self.lblMovie_ID.grid(row=0, column=0, sticky=W)
        self.txtMovie_ID=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Movie_ID, width=39, bg="black", fg="white")
        self.txtMovie_ID.grid(row=0, column=1)

        self.lblMovie_Name=Label(LeftBody, font=('Arial', 18, 'bold'), text="Movie Name:", padx=2, pady=2, bg="black", fg="orange")
        self.lblMovie_Name.grid(row=1, column=0, sticky=W)
        self.txtMovie_Name=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Movie_Name, width=39, bg="black", fg="white")
        self.txtMovie_Name.grid(row=1, column=1)

        self.lblRelease_Date=Label(LeftBody, font=('Arial', 18, 'bold'), text="Release Date:", padx=2, pady=2, bg="black", fg="orange")
        self.lblRelease_Date.grid(row=2, column=0, sticky=W)
        self.txtRelease_Date=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Release_Date, width=39, bg="black", fg="white")
        self.txtRelease_Date.grid(row=2, column=1)

        self.lblDirector=Label(LeftBody, font=('Arial', 18, 'bold'), text="Director:", padx=2, pady=2, bg="black", fg="orange")
        self.lblDirector.grid(row=3, column=0, sticky=W)
        self.txtDirector=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Director, width=39, bg="black", fg="white")
        self.txtDirector.grid(row=3, column=1)

        self.lblCast=Label(LeftBody, font=('Arial', 18, 'bold'), text="Cast:", padx=2, pady=2, bg="black", fg="orange")
        self.lblCast.grid(row=4, column=0, sticky=W)
        self.txtCast=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Cast, width=39, bg="black", fg="white")
        self.txtCast.grid(row=4, column=1)

        self.lblBudget=Label(LeftBody, font=('Arial', 18, 'bold'), text="Budget (Crores INR):", padx=2, pady=2, bg="black", fg="orange")
        self.lblBudget.grid(row=5, column=0, sticky=W)
        self.txtBudget=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Budget, width=39, bg="black", fg="white")
        self.txtBudget.grid(row=5, column=1)

        self.lblDuration=Label(LeftBody, font=('Arial', 18, 'bold'), text="Duration (Hrs):", padx=2, pady=2, bg="black", fg="orange")
        self.lblDuration.grid(row=6, column=0, sticky=W)
        self.txtDuration=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Duration, width=39, bg="black", fg="white")
        self.txtDuration.grid(row=6, column=1)
```

5.3 Customer Info Part:

```
class Custm:

    def __init__(self, root):
        d = database.Backend()
        self.root=root
        self.root.title('Movie Ticket Booking System')
        self.root.geometry('1350x750')
        self.root.config(bg="black")


        def clcdata():
            self.txtCustom_id.delete(0,END)
            self.txtCustom_name.delete(0,END)
            self.txtCustom_email.delete(0,END)
            self.txtCustom_phno.delete(0,END)

        def addData():
            if (len(Custom_id.get())!=0):
                d.addCustomerData(Custom_id.get(),Custom_name.get(),Custom_email.get(),Custom_phno.get())
                custList.delete(0,END)
                custList.insert(Custom_id.get(),Custom_name.get(),Custom_email.get(),Custom_phno.get())
                disData()
            else:
                tkinter.messagebox.askyesno('Enter a Customer ID')


        def disData():
            custList.delete(0,END)
            for row in d.ViewCustomerData():
                custList.insert(END,row,str(''))

        def delData():
            if (len(Custom_id.get())!=0):
                d.DeleteCustomRec(Custom_id.get())
                clcdata()
                disData()
```

```
HeadFrame = Frame(MainFrame,bd=1,bg='black',relief=RIDGE,padx=50,pady=10)
HeadFrame.pack(side=TOP)

self.TFrame=Label(HeadFrame, font=('Arial', 50, 'bold'), text="\u2680 MOVIE TICKET BOOKING SYSTEM\u2680", bg="black", fg="red")
self.TFrame.grid()

BottomFrame=Frame(MainFrame, bd=2, width=1350, height=70, padx=18, pady=10, bg="black", relief=RIDGE)
BottomFrame.pack(side=BOTTOM)

BodyFrame=Frame(MainFrame, bd=2, width=1300, height=500, padx=20, pady=20, bg="black", relief=RIDGE)
BodyFrame.pack(side=BOTTOM)

LeftBody=LabelFrame(BodyFrame, bd=2, width=1000, height=480, padx=20, bg="black", relief=RIDGE, font=('Arial', 15, 'bold'),text='Customer info:\n',fg='white')
LeftBody.pack()

RightBody=LabelFrame(BodyFrame, bd=2, width=350, height=480, padx=31, pady=3, bg="black", relief=RIDGE, font=('Arial', 15, 'bold'),text='Customer Details:\n',fg='
RightBody.pack(side=RIGHT)


self.lblCustom_id=Label(LeftBody, font=('Arial', 18, 'bold'), text="Customer ID:", padx=2, pady=2, bg="black", fg="orange")
self.lblCustom_id.grid(row=0, column=0, sticky=W)
self.txtCustom_id=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Custom_id, width=39, bg="black", fg="white")
self.txtCustom_id.grid(row=0, column=1)

self.lblCustom_name=Label(LeftBody, font=('Arial', 18, 'bold'), text="Customer Name:", padx=2, pady=2, bg="black", fg="orange")
self.lblCustom_name.grid(row=1, column=0, sticky=W)
self.txtCustom_name=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Custom_name, width=39, bg="black", fg="white")
self.txtCustom_name.grid(row=1, column=1)

self.lblCustom_email=Label(LeftBody, font=('Arial', 18, 'bold'), text="Customer Email:", padx=2, pady=2, bg="black", fg="orange")
self.lblCustom_email.grid(row=2, column=0, sticky=W)
self.txtCustom_email=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Custom_email, width=39, bg="black", fg="white")
self.txtCustom_email.grid(row=2, column=1)

self.lblCustom_phno=Label(LeftBody, font=('Arial', 18, 'bold'), text="Customer Ph no:", padx=2, pady=2, bg="black", fg="orange")
self.lblCustom_phno.grid(row=3, column=0, sticky=W)
self.txtCustom_phno=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Custom_phno, width=39, bg="black", fg="white")
```

5.4 BackEnd Part:

```
import sqlite3

class Backend():

    def datas(self):
        conn = sqlite3.connect('movie.db')
        conn.execute("PRAGMA foreign_keys = 1")
        c = conn.cursor()

        c.execute("""CREATE TABLE IF NOT EXISTS customer (
                    cid INTEGER PRIMARY KEY,
                    c_name text,
                    email_id text,
                    phone_no text
                )""")

        c.execute(""" CREATE TABLE IF NOT EXISTS movie_data(
                    m_id INTEGER PRIMARY KEY,
                    m_name text,
                    release_date text,
                    director text,
                    actors text,
                    budget integer,
                    duration int,
                    rating int
                )""")

        c.execute(""" CREATE TABLE IF NOT EXISTS tickets(
                    ticket_no INTEGER PRIMARY KEY,
                    m_name text,
                    price int,
                    seat_no int,
                    show_date text
                )""")

        c.execute(""" CREATE TABLE IF NOT EXISTS booking(

                    cid INTEGER,
```

```
def addMovie(self,m_id,m_name,release_date,director,actors,budget,duration,rating):
    conn = sqlite3.connect('movie.db')
    c = conn.cursor()
    c.execute('INSERT INTO movie_data VALUES(?,?,?,?,?,?,?,?)',(m_id,m_name,release_date,director,actors,budget,duration,rating))
    conn.commit()
    conn.close()

def ViewMovieData(self):
    conn=sqlite3.connect("movie.db")
    c=conn.cursor()
    c.execute("SELECT * FROM movie_data")
    rows=c.fetchall()
    conn.commit()
    conn.close()
    return rows

def DeleteMovieRec(self,m_id):
    conn=sqlite3.connect("movie.db")
    c=conn.cursor()
    c.execute("DELETE FROM movie_data WHERE m_id=?", (m_id,))
    conn.commit()
    conn.close()

def addCustomerData(self,cid,c_name,email_id,phone_no):
    conn=sqlite3.connect("movie.db")
    c=conn.cursor()
    c.execute('INSERT INTO customer VALUES(?,?,?,?)',(cid,c_name,email_id,phone_no))
    conn.commit()
    conn.close()

def ViewCustomerData(self):
    conn=sqlite3.connect("movie.db")
    c=conn.cursor()
    c.execute("SELECT * FROM customer")
    rows=c.fetchall()
    conn.commit()
```

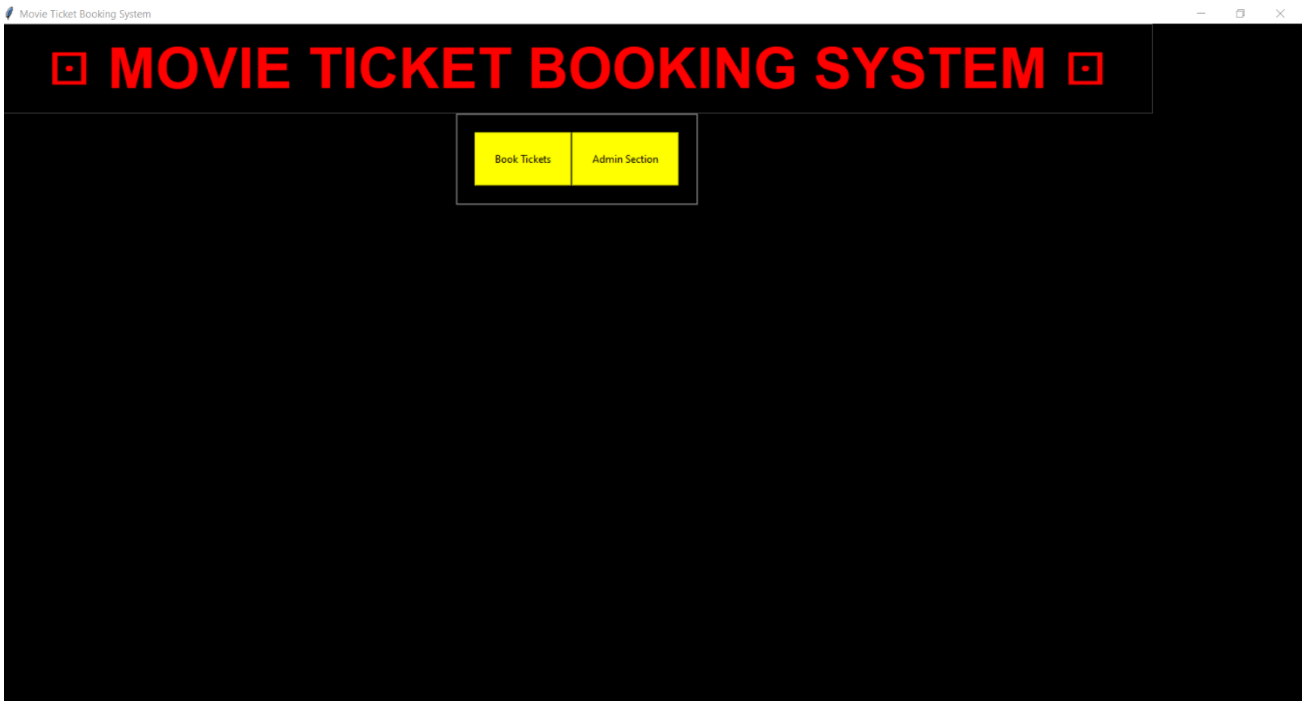
5.5 Ticket Info Part:

```
from tkinter import *
import tkinter.messagebox
import database


class Ticket:

    def __init__(self, root):

        d = database.Backend()
        self.root=root
        self.root.title('Movie Ticket Booking System')
        self.root.geometry('1350x750')
        self.root.config(bg="black")


        Ticket_no = StringVar()
        Movie_name = StringVar()
        Price = StringVar()
        Seat_no = StringVar()
        Show_date = StringVar()

        def clcdata():
            self.txtTicket_no.delete(0,END)
            self.txtMovie_name.delete(0,END)
            self.txtPrice.delete(0,END)
            self.txtSeat_no.delete(0,END)
            self.txtShow_date.delete(0,END)

        def disData():
            MovieList.delete(0,END)
            for row in d.ViewMovieData():
                MovieList.insert(END,row,str(''))

        def addData():
            if (len(Ticket_no.get())!=0):
                d.addTickets(Ticket_no.get(),Movie_name.get(),Price.get(),Seat_no.get(),Show_date.get())
                #MovieList.delete(0,END)
```

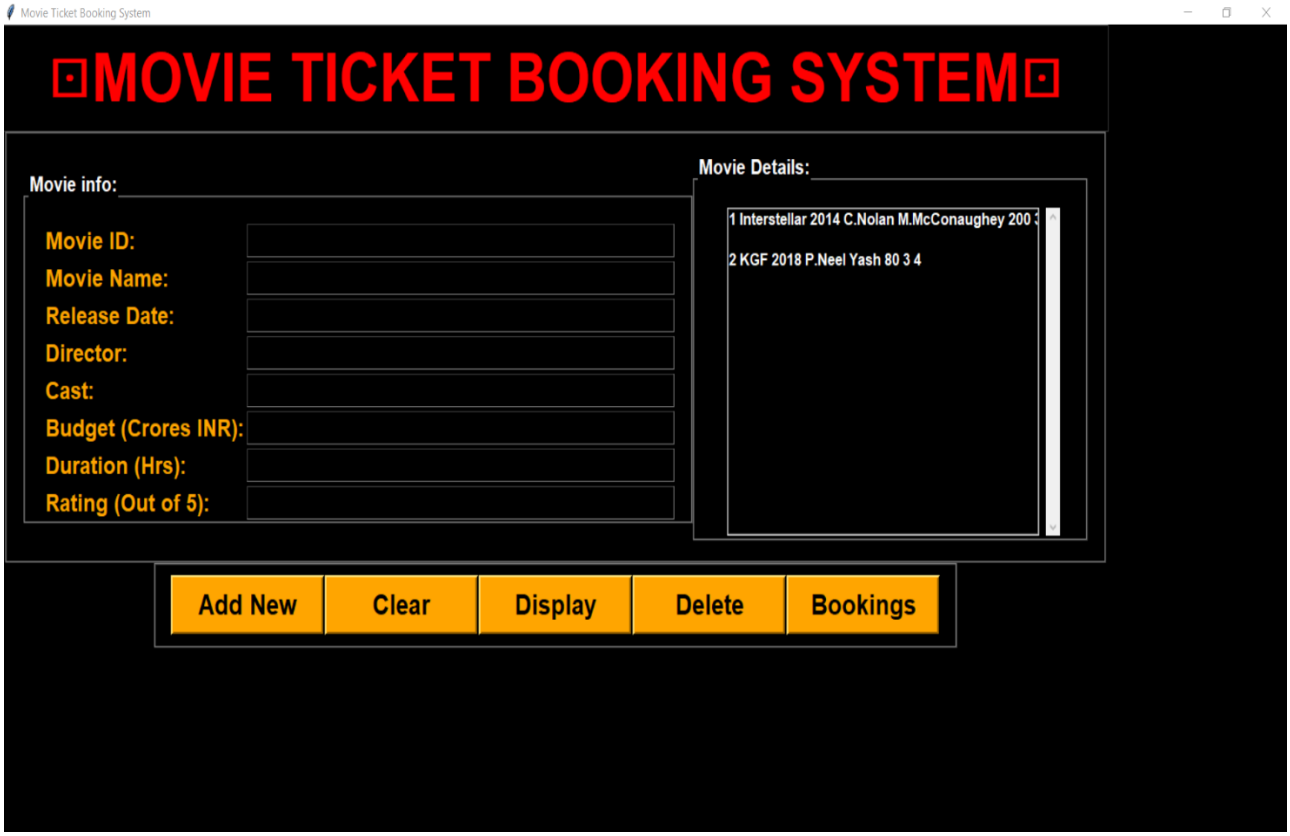
```
        self.lblSeat_no=Label(LeftBody, font=('Arial', 18, 'bold'), text="Seat No.:", padx=2, pady=2, bg="black", fg="orange")
        self.lblSeat_no.grid(row=3, column=0, sticky=W)
        self.txtSeat_no=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Seat_no, width=39, bg="black", fg="white")
        self.txtSeat_no.grid(row=3, column=1)

        self.lblShow_date=Label(LeftBody, font=('Arial', 18, 'bold'), text="Show Date.:", padx=2, pady=2, bg="black", fg="orange")
        self.lblShow_date.grid(row=4, column=0, sticky=W)
        self.txtShow_date=Entry(LeftBody, font=('Arial', 18, 'bold'), textvariable=Show_date, width=39, bg="black", fg="white")
        self.txtShow_date.grid(row=4, column=1)

        #Scroll bar
        scroll=Scrollbar(RightBody)
        scroll.grid(row=0, column=1, sticky='ns')

        MovieList=Listbox(RightBody, width=41, height=16, font=('Arial', 12, 'bold'), bg="black", fg="white", yscrollcommand=scroll.set)
        #MovieList.bind('<<ListboxSelect>>', movierec)
        MovieList.grid(row=0, column=0, padx=8)
        scroll.config(command=MovieList.yview)

        #Buttons

        self.btnadd=Button(BottomFrame, text="Add", font=('Arial', 20, 'bold'), width=10, height=1, bd=4, bg="orange",command=addData)
        self.btnadd.grid(row=0, column=0)

        self.btnclear=Button(BottomFrame, text="Clear", font=('Arial', 20, 'bold'), width=10, height=1, bd=4, bg="orange",command=clcdata)
        self.btnclear.grid(row=0, column=1)   #column 2

        self.btndisp=Button(BottomFrame, text="Display", font=('Arial', 20, 'bold'), width=10, height=1, bd=4, bg="orange",command=disData)
        self.btndisp.grid(row=0, column=2)

if __name__=='__main__':
    root=Tk()
    database = Ticket(root)
    root.mainloop()
```

6.Result and Discussion

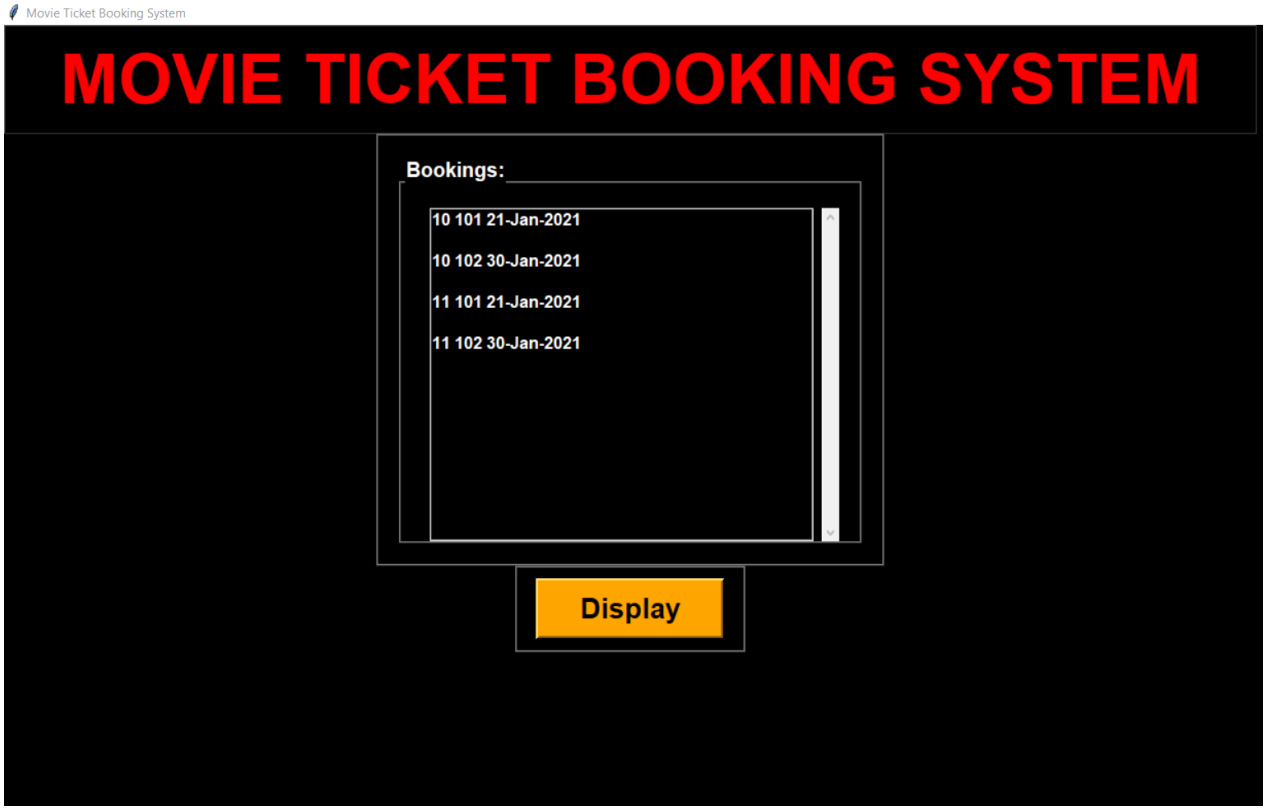
6.1 Home Page:



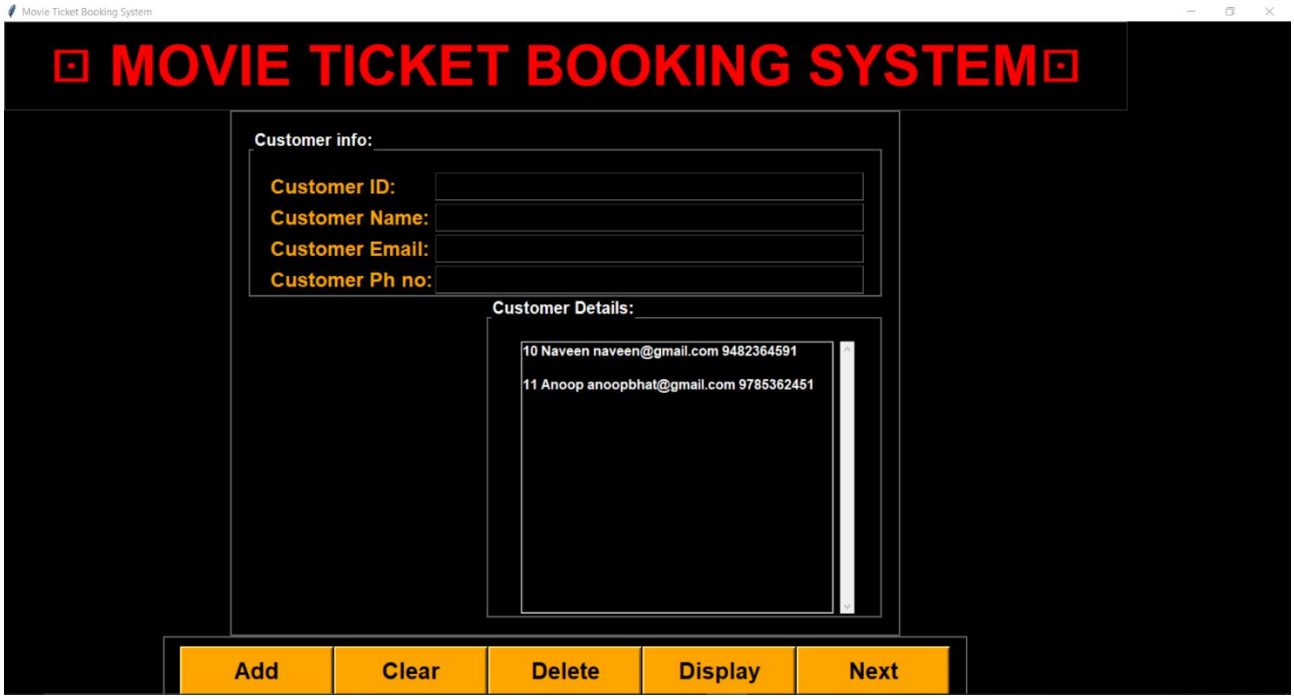
6.2 Admin page:



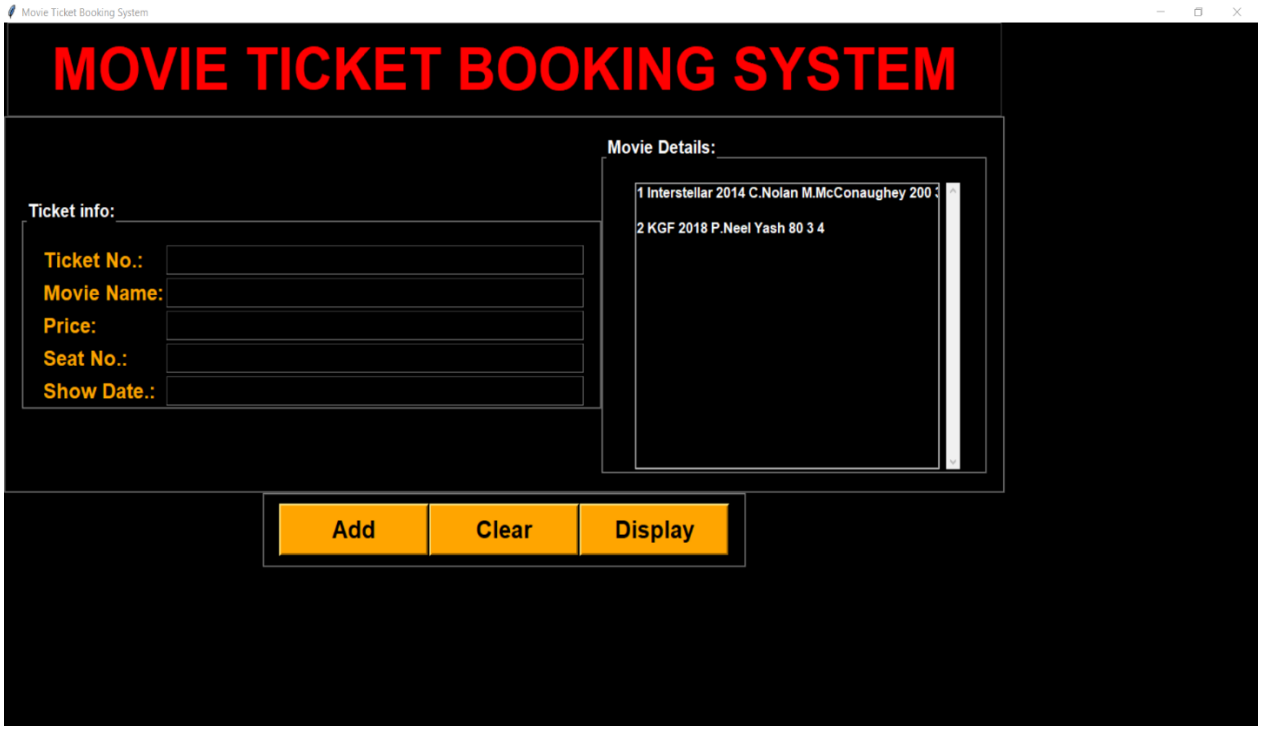
6.3 Booking Page:



6.4 Customer Information Page:



6.5 Ticket Booking Page:



7.Conclusions:

This project is developed successfully and the performance is found to be satisfactory. This project is designed to meet the requirements of assigning jobs. It has been developed in Python and the database has been built in SQLite keeping in mind the specifications of the system. The user will be able to book the ticket using this GUI. The relationship between company manager, employee, and customer satisfy a good communication to complete ticketing process.

We have designed the project to provide the user with easy retrieval of data, details of theatre and necessary feedback as much as possible. In this project, the user is provided with a GUI that can be used to book movie tickets online. To implement this as a GUI application we used Python as the technology. SQLite has advantages such as enhanced performance, scalability, built- in security and simplicity. To build any GUI application using Python we need a programming language such as Python and so on. SQLite was used as back-end database since it is one of the most popular opensource databases, and it provides fast data access, easy installation and simplicity. For front end we used Python and Tkinter.