

# **REC MOVIE BOOKING SYSTEM**

## **A MINI PROJECT REPORT**

**Submitted by**

**SEENUVAASAN                      220701255**

**SAFE EKUR RAHMAN            220701236**

**SARVESHWARAN                220701254**

In partial fulfillment for the award of the degree of

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE**

**RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)**

**THANDALAM**

**CHENNAI-602105**

**2023 - 24**

## **BONAFIDE CERTIFICATE**

Certified that this project report “**REC MOVIE BOOKING SYSTEM**” is the  
bonafide work of “**SEENUVAASAN(2207012550),SAFEELKUR  
RAHMAN(220701236),SARVESHWARAN(220701254) ”**  
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**

**Mrs.K.maheshmeena**  
Assistant Professor (SG),  
Computer Science and Engineering,  
Rajalakshmi Engineering College,  
(Autonomous),  
Thandalam, Chennai - 602 105

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

## **ABSTRACT**

The provided Python code utilizes Tkinter to create a Ticket Booking System GUI. It integrates with a MySQL database for ticket management.

The system allows users to view available movie tickets, select the number of tickets to book, and enter their name for reservation. Upon booking, it updates the database, generates a reservation summary, and stores it in a text file. The database script defines functions for connecting, creating, and updating ticket data.

The program ensures data integrity by managing ticket quantities and provides user-friendly feedback through error and success messages.

# **TABLE OF CONTENTS**

## **1.INTRODUCTION**

1.1INTRODUCTION

1.2OBJECTIVES

1.3MODULES

## **2. SURVEY OF TECHNOLOGIES**

2.1 SOFTWARE DESCRIPTION

2.2 LANGUAGES

2.3 SQL

2.4PYTHON

## **3.REQUIREMENTS AND ANALYSIS**

3.1REQUIREMENT SPECIFICATION

3.2HARDWARE AND SOFTWARE REQUIREMENTS

3.3ER DIAGRAM

3.4NORMALIZATION

## **4.PROGRAM CODE**

4.1 RESULTS AND DISCUSSION

4.2 CONCLUSION

REFERENCES

# CHAPTER 1

## 1. Introduction

### 1.1 Introduction

The Movie Ticket Booking System is designed to streamline the process of booking movie tickets for users. It offers an intuitive and user-friendly interface that allows users to view available movies, select the desired number of tickets, and complete the booking process with ease. This system integrates a graphical user interface (GUI) built using Tkinter and CustomTkinter, and employs MySQL for backend database operations. By leveraging these technologies, the system ensures that users have a seamless experience while booking movie tickets.

### 1.2 Objectives

The objectives of the Movie Ticket Booking System include:

**User Convenience:** To provide a simple and efficient interface for users to book movie tickets.

**Dynamic Ticket Management:** To manage and update the availability of tickets dynamically based on user bookings.

**Data Integrity:** To ensure secure storage and retrieval of movie and ticket information from the database.

**Booking Confirmation:** To generate and display booking details for user reference and record-keeping.

**Scalability:** To design a system that can be easily scaled to handle more movies and larger datasets in the future.

### 1.3 Modules

The Movie Ticket Booking System comprises several key modules:

**User Interface Module:** Implements the GUI using Tkinter and CustomTkinter, allowing users to interact with the system visually.

**Database Module:** Utilizes MySQL for storing and managing movie and ticket information, ensuring efficient data handling.

**Booking Module:** Manages the booking process, updates ticket availability in the database, and generates booking confirmation details.

**File Handling Module:** Handles the generation of text files containing booking details for user reference.

## CHATER 2

### 2. Survey of Technologies

#### 2.1 Software Description

The software for the Movie Ticket Booking System is developed using Python, which is known for its simplicity and efficiency. The graphical user interface is created using Tkinter and CustomTkinter, libraries that are popular for developing desktop applications in Python. MySQL is employed as the database management system to store and manage data related to movies and ticket availability. Together, these technologies create a robust and user-friendly application for booking movie tickets.

#### 2.2 Languages

##### 2.2.1 SQL

Structured Query Language (SQL) is a standard programming language used for managing and manipulating relational databases. In this project, SQL is used extensively to perform various database operations such as creating tables, inserting data, updating records, and fetching data from the database. The primary SQL operations involved in the project are:

**Create Tables:** SQL commands are used to define the structure of the database, creating tables to store ticket information.

**Insert Data:** SQL INSERT statements are used to add initial movie and ticket data into the database.

**Update Data:** SQL UPDATE statements are used to modify the number of available tickets after a booking is made

**Select Data:** SQL SELECT statements are used to retrieve the list of available tickets, which are then displayed in the GUI.

Examples of SQL queries used in the project include:

```
CREATE TABLE IF NOT EXISTS Tickets (  
    ticket_id VARCHAR(10) PRIMARY KEY,  
    movie_name VARCHAR(100),  
    available_tickets INT,  
    ticket_price FLOAT  
);
```

### **2.2.2 Python**

Python is the primary programming language used to develop the Movie Ticket Booking System. It is widely known for its readability and ease of use, making it an ideal choice for both novice and experienced developers. Python's extensive library support allows for rapid development and integration of various functionalities.

Key Python components and libraries used in the project include:

**Tkinter and CustomTkinter:** These libraries are used to create the graphical user interface (GUI) of the application. Tkinter is the standard GUI toolkit for Python, while CustomTkinter provides additional customization options for creating modern and visually appealing interfaces.

**MySQL Connector:** This is a Python library that facilitates communication between Python and MySQL databases. It allows the execution of SQL queries from within the Python code, enabling seamless database operations.

**File Handling:** Python's built-in file handling capabilities are used to generate and save text files containing booking details, providing users with a tangible record of their transactions.



## CHAPTER 3

### **REQUIREMENT ANALYSIS**

#### **3.1 Requirement Analysis**

##### **Functional Requirements:**

###### **1. Display Available Films:**

- This requirement ensures that users can view a list of movies along with their availability and ticket prices. It enhances user experience by providing relevant information for decision-making.

###### **2. Select Number of Tickets:**

- By allowing users to specify the quantity of tickets they want to book, this requirement facilitates flexibility and customization in the booking process, catering to varying needs.

###### **3. Enter Customer Information:**

- Collecting customer information such as name ensures personalization of the booking process and enables communication with customers regarding their reservations.

###### **4. Book Tickets:**

- This requirement is essential for the core functionality of the system, enabling users to confirm their ticket selections and initiate the booking process.

###### **5. Update Ticket Quantity:**

- Ensuring that the available ticket quantity is updated in the database after booking helps maintain data accuracy and prevents overselling of tickets.

###### **6. Generate Reservation Summary:**

- Providing users with a summary of their reservation enhances transparency and serves as a confirmation of their booking details. Storing this information in a text file ensures a record of all transactions for reference.

## **7. Error Handling:**

- Error handling is crucial for providing a seamless user experience. By detecting and notifying users of errors such as incomplete data or insufficient ticket availability, this requirement improves system reliability and user satisfaction.

## **Non-Functional Requirements:**

### **1. User Interface:**

- A visually appealing and user-friendly interface enhances usability and encourages user engagement. It contributes to the overall user experience and can influence user perception of the system.

### **2. Performance:**

- Efficient database operations are necessary to prevent delays or timeouts during peak usage periods. Optimizing performance ensures responsiveness and reliability, even under high load conditions.

### **3. Security:**

- Secure database connections and transaction handling are essential for protecting sensitive user data and preventing unauthorized access or manipulation of information.

### **4. Reliability:**

- Maintaining data integrity and consistency is critical for ensuring that users can trust the system to accurately process their bookings and provide reliable information.

### **5. Scalability:**

- Designing the system to be scalable allows for future growth and expansion without significant rework. It ensures that the system can accommodate increased user demand or additional features seamlessly.

### **6. Documentation:**

- Comprehensive documentation facilitates system understanding, maintenance, and support. It ensures that users and developers have access to relevant information for installation, usage, and troubleshooting.

### **7. Compatibility:**

- Compatibility with various operating systems and devices ensures broad accessibility and usability, catering to a diverse user base. It improves the reach and adoption of the system across different platforms.

## **3.2 HARDWARE AND SOFTWARE REQUIREMENTS**

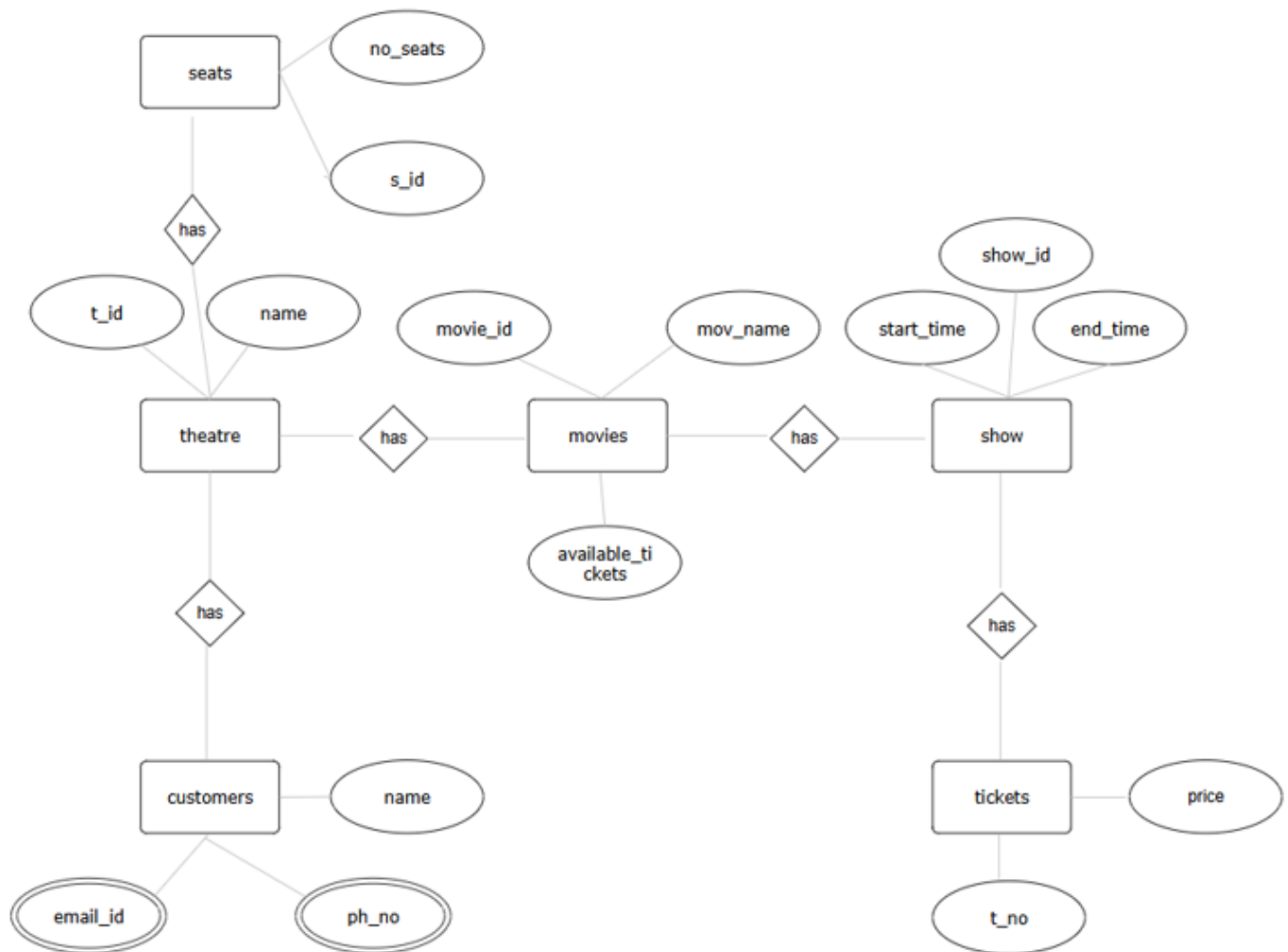
### **HARDWARE SPECIFICATION**

PROCESSOR : INTEL i3  
MEMORY SIZE : 4GB  
HDD : 256GB

### **SOFTWARE SPECTFICATION**

OPERATING SYSTEM : WINDOWS 11  
GUI INTERFACE : PYTHON  
BACKEND : MY SQL

### 3.3 ER DIAGRAM



### **3.4 NORMALIZATION**

Let's represent the `Tickets` table in terms of each normalization level:

#### **First Normal Form (1NF):**

ticket_id	movie_name	available_tickets	ticket_price
1	movie1	3	50
2	movie2	2	40
3	Movie3	4	60
4	Movie4	5	70
5	Movie5	1	65
...			

In 1NF, each cell contains a single value, and there are no repeating groups within cells.

#### **Second Normal Form (2NF):**

ticket_id	movie_name	available_tickets	ticket_price
1	movie1	3	50
2	movie2	2	40
3	Movie3	4	60
4	Movie4	5	70
5	Movie5	1	65
...			

In 2NF, each non-prime attribute (`movie\_name`, `available\_tickets`, `ticket\_price`) is fully functionally dependent on the primary key (`ticket\_id`).

### **Third Normal Form (3NF):**

ticket_id	movie_name	available_tickets	ticket_price
1	movie1	3	50
2	movie2	2	40
3	Movie3	4	60
4	Movie4	5	70
5	Movie5	1	65

In 3NF, there are no transitive dependencies, meaning each non-prime attribute (`movie\_name`, `available\_tickets`, `ticket\_price`) depends only on the primary key (`ticket\_id`).

## CHAPTER 4

### PROGRAM CODE :

#### //MAIN.PY

```
from tkinter import StringVar
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox

import customtkinter
import db

app=customtkinter.CTk()
app.title("TICKET BOOKING SYSTEM")
app.geometry('1000x1000')
app.config(bg='#18161D')
app.resizable(False, False)

font1 = ('Arial', 25, 'bold')
font2 = ('Arial', 13, 'bold')
font3 = ('Arial', 18, 'bold')

def add_to_treeview():
    conn = db.connect_to_database()
    tickets = db.get_tickets(conn)
    conn.close()
    tree.delete(*tree.get_children())
    for ticket in tickets:
        if ticket[2] > 0:
            tree.insert("", tk.END, values=ticket)

def reservation(name, movie, quantity, price):
    customer_name = name
    movie_name = movie
    booked_quantity = quantity
    ticket_price = price
```

```
total_price = ticket_price * booked_quantity
```

```
frame = customtkinter.CTkFrame(app, bg_color='#18161D', fg_color='#292933', corner_radius=10,  
border_width=2, border_color='#0f0', width=200, height=130)  
frame.place(x=390, y=450)
```

```
name_label = customtkinter.CTkLabel(frame, font=font3, text=f'Name: {customer_name}',  
text_color='fff', bg_color='#18161D')  
name_label.place(x=10, y=10)
```

```
movie_label = customtkinter.CTkLabel(frame, font=font3, text=f'Movie: {movie_name}',  
text_color='fff', bg_color='#18161D')  
movie_label.place(x=10, y=50)
```

```
total_price_label = customtkinter.CTkLabel(frame, font=font3, text=f'Total: {total_price}',  
text_color='fff', bg_color='#18161D')  
total_price_label.place(x=10, y=90)
```

```
return total_price
```

```
def book():
```

```
    customer_name = name_entry.get()
```

```
    selected_item = tree.focus()
```

```
    if not selected_item:
```

```
        messagebox.showerror('Error', 'Choose a ticket to book')
```

```
    elif not customer_name:
```

```
        messagebox.showerror('Error', 'Enter Customer name')
```

```
    else:
```

```
        row = tree.item(selected_item)['values']
```

```
        if len(row) < 4:
```

```
            messagebox.showerror('Error', 'Ticket data is incomplete')
```

```
            return
```

```
        ticket_id = row[0]
```

```
        movie_name = row[1]
```

```
        ticket_price = row[3]
```



```

booked_quantity = int(variable.get())
if booked_quantity > row[2]:
    messagebox.showerror('Error', 'Not enough tickets')
else:
    conn = db.connect_to_database()
    db.update_quantity(conn, ticket_id, booked_quantity)
    conn.close()
    add_to_treeview()

total_price = reservation(customer_name, movie_name, booked_quantity, ticket_price)
with open('Ticket.txt', 'a') as file:
    file.write(f'customer_name: {customer_name}\n')
    file.write(f'movie_name: {movie_name}\n')
    file.write(f'Total: {total_price}$\n=====\\n')
messagebox.showinfo('Success', 'Tickets are booked')
print(f'Success: Booked {booked_quantity} tickets for {movie_name}')

```

```

ticket_label = customtkinter.CTkLabel(app, font=font1, text='Available Films', text_color='#fff',
bg_color='#18161D')
ticket_label.place(x=360, y=20)

```

```

name_label = customtkinter.CTkLabel(app, font=font3, text='Customer name:', text_color='#fff',
bg_color='#18161D')
name_label.place(x=330, y=300)

```

```

name_entry = customtkinter.CTkEntry(app, font=font3, text_color='#000', fg_color='#fff',
border_color='#AA04A7', border_width=2, width=160)
name_entry.insert(0, "Enter your name")
name_entry.place(x=490, y=300)

```

```

number_label = customtkinter.CTkLabel(app, font=font3, text='No. of Tickets:', text_color='#fff',
bg_color='#18161D')

```

```

number_label.place(x=350, y=350)

```

```
variable = StringVar()
```

```
option = ['1', '2', '3']
```

```
duration_option = customtkinter.CTkComboBox(app, font=font3, text_color='#000', fg_color='#fff',  
dropdown_hover_color='#AA04A7', button_color='#AA04A7', button_hover_color='#AA04A7',  
border_color='#AA04A7', width=160, variable=variable, values=option, state='readonly')
```

```
duration_option.set('1')
```

```
duration_option.place(x=490, y=350)
```

```
book_button = customtkinter.CTkButton(app, font=font3, text_color='#fff', text='Book tickets',  
fg_color='#AA04A7', bg_color='#18161D', cursor='hand2', corner_radius=15, width=200,  
command=book)
```

```
book_button.place(x=390, y=400)
```

```
style = ttk.Style(app)
```

```
style.theme_use('clam')
```

```
style.configure('TreeView', font=font2, foreground='#fff', background='#000',  
fieldbackground='#292933')
```

```
style.map('Treeview', background=[('selected', '#AA04A7')])
```

```
tree = ttk.Treeview(app, height=8)
```

```
tree['columns'] = ('Ticket ID', 'Movie Name', 'Available Tickets', 'Ticket Price')
```

```
tree.heading('#0', text="", anchor=tk.CENTER)
```

```
tree.heading('Ticket ID', text='Ticket ID', anchor=tk.CENTER)
```

```
tree.heading('Movie Name', text='Movie Name', anchor=tk.CENTER)
```

```
tree.heading('Available Tickets', text='Available Tickets', anchor=tk.CENTER)
```

```
tree.heading('Ticket Price', text='Ticket Price', anchor=tk.CENTER)
```

```
tree.column('#0', width=0, stretch=tk.NO)
```

```
tree.column('Ticket ID', anchor=tk.CENTER, width=100)
```

```
tree.column('Movie Name', anchor=tk.CENTER, width=100)
```

```
tree.column('Available Tickets', anchor=tk.CENTER, width=100)
```

```
tree.column('Ticket Price', anchor=tk.CENTER, width=100)
```

```
tree.place(x=290, y=95)
```

```
add_to_treeview()
```

```
app.mainloop()
```

### **//DB.PY**

```
import mysql.connector
```

```
def connect_to_database():
```

```
    conn = mysql.connector.connect(
```

```
        host="localhost",
```

```
        user="root",
```

```
        password="safee123",
```

```
        database="movie_booking_db"
```

```
    )
```

```
    return conn
```

```
def drop_tables(conn):
```

```
    cursor = conn.cursor()
```

```
    cursor.execute("DROP TABLE IF EXISTS Tickets")
```

```
    conn.commit()
```

```
    cursor.close()
```

```
def create_tables(conn):
```

```
    cursor = conn.cursor()
```

```
    cursor.execute("""
```

```
        CREATE TABLE IF NOT EXISTS Tickets (
```

```
            ticket_id VARCHAR(10) PRIMARY KEY,
```

```
            movie_name VARCHAR(100),
```

```
            available_tickets INT,
```

```
            ticket_price FLOAT
```

```
        )
```

```
    """)
```

```
    conn.commit()
```

```
cursor.close()

def clear_table(conn):
    cursor = conn.cursor()
    cursor.execute("DELETE FROM Tickets")
    conn.commit()
    cursor.close()

def insert_tickets(conn):
    cursor = conn.cursor()
    tickets_data = [
        ('1', 'movie1', 3, 50),
        ('2', 'movie2', 2, 40),
        ('3', 'Movie3', 4, 60),
        ('4', 'Movie4', 5, 70),
        ('5', 'Movie5', 1, 65)
    ]
    cursor.executemany("INSERT INTO Tickets (ticket_id, movie_name, available_tickets,
ticket_price) VALUES (%s, %s, %s, %s)", tickets_data)
    conn.commit()
    cursor.close()

def get_tickets(conn):
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM Tickets')
    tickets = cursor.fetchall()
    cursor.close()
    return tickets

def update_quantity(conn, id, reserved_quantity):
    cursor = conn.cursor()
    cursor.execute('UPDATE Tickets SET available_tickets = available_tickets - %s WHERE ticket_id
= %s', (reserved_quantity, id))
    conn.commit()

    cursor.close()
```

```
if __name__ == "__main__":  
    conn = connect_to_database()  
    drop_tables(conn) # Drop existing tables  
    create_tables(conn) # Create tables from scratch  
    insert_tickets(conn) # Insert initial ticket data  
    tickets = get_tickets(conn) # Retrieve tickets  
    update_quantity(conn, 'T1', 2) # Update ticket quantity  
    conn.close()
```

## CHAPTER 5

### RESULT:

#### 1.Ticket interface

### Available Films

Ticket ID	Movie Name	Available Tickets	Ticket Price
1	movie1	3	50.0
2	movie2	2	40.0
3	Movie3	4	60.0
4	Movie4	5	70.0
5	Movie5	1	65.0

Customer name:

Enter your name

No. of Tickets:

1

Book tickets

## 2.choose ticket

### Available Films

Ticket ID	Movie Name	Available Tickets	Ticket Price
1	movie1	3	50.0
2	movie2	2	40.0
3	Movie3	4	60.0
4	Movie4	5	70.0
5	Movie5	1	65.0

Customer name:

No. of Tickets:

▼

Book tickets

### 3,book ticket

### Available Films

Ticket ID	Movie Name	Available Tickets	Ticket Price
1	movie1	3	50.0
3	Movie3	4	60.0
4	Movie4	5	70.0
5	Movie5	1	65.0

Success

Tickets are booked

OK

Customer name:

ram

No. of Tickets:

1

Book tickets

Name: ram

Movie: movie2

Total: 40.0



#### 4.Not enough ticket


### Available Films

Ticket ID	Movie Name	Available Tickets	Ticket Price
1	movie1	3	50.0
3	Movie3	4	60.0
4	Movie4	4	70.0
5	Movie5	1	65.0

Customer name:

No. of Tickets:

Error

 Not enough tickets

OK

## CHAPTER 6

### **CONCLUSION:**

The Ticket Booking System project presents a comprehensive solution for streamlining the process of booking movie tickets. Leveraging Python with Tkinter for the graphical user interface and MySQL for database management, this system offers users an intuitive platform to browse available films, select desired ticket quantities, and provide their information for reservation. By integrating error handling mechanisms and ensuring data integrity through normalization up to Third Normal Form (3NF), the system provides a seamless and reliable booking experience. The user-friendly interface, coupled with detailed documentation and modular code organization, enhances usability and maintenance. With its scalability and potential for future enhancements, the Ticket Booking System stands as a versatile tool for effectively managing movie ticket reservations, catering to the needs of both users and administrators alike.

## REFERENCES

**ADD LINKS ,BOOKS,REFERED TO DEVELOP THIS PROJECT**

## **REFERENCES**

- 1 Manish Kumar Srivastava, A.K Tiwari, “A Study of Behavior of Maruti SX4 and Honda City Custo Jaipur”, Pacific Business Review- Quarterly Referred Journal, Zenith International Journal of Multi disciplinary Research Vol.4, Issue 4, pp. 77-90, Apr2011.**
- 1 M.Prasanna Mohan Raj, Jishnu Sasikumar, S.Sriram , “A Study of Customers Brand Preference in SUVs and MUVs: Effect on Marketing Mix Variables”, International Referred Research Journal Vol.- IV, Issue-1, pp. 48-58, Jan2013.**
- 2 Nikhil Monga, Bhuvender Chaudhary, “Car Market and Buying behavior - study on Consumer Perception”, IJRMEC Vol.2, Issue-2, pp. 44-63, Feb2012.**

