KISHKINDA UNIVERSITY, Ballari

Mini Project Report

On

"Theater Renovation Planner POC"

Department of MCA

Submitted By:

Ramanjineyalu KUB23MCA013

Introduction

Project Overview:

- The Theatre Renovation Planner POC aims to streamline the process of managing and planning renovations for theatre's. It serves as a solution for theatre owners and managers to plan renovations, schedule them effectively, and monitor the impacts on theatre performance, such as audience satisfaction and attendance rates.

- This project uses Python and Object-Oriented Programming (OOP) principles to achieve efficient management of renovation plans, incorporating CRUD (Create, Read, Update, Delete) operations, scheduling capabilities, and impact monitoring.

Problem Statement:

- Aging theatre's often require renovations to stay competitive and provide audiences with a better experience. Managing multiple renovation projects while tracking their impacts on audience behaviour is challenging when done manually. There is a need for an automated system that can handle renovation schedules, store multiple renovation plans, and monitor the outcomes to assess the renovations' success.

Objective

The primary goals of this POC are:

- Efficient Management of Renovation Plans: To provide CRUD functionality for adding, updating, deleting, and viewing renovation plans for multiple theatres.
- Scheduling and Planning: To plan and schedule renovations for theatre's, particularly focusing on aging theatre's that require timely updates.
- Monitoring Impacts: To track the effects of renovations on theatre attendance and customer satisfaction, using data analytics to assess the impact of the changes.
- Develop a CRUD system for renovation plans.
- Plan and schedule theatre renovations based on specific criteria (e.g., age of theatre, audience satisfaction).
- Monitor renovation impacts on theatre attendance and audience feedback

Software Requirements:

- Python 3.12(64-bit)
- Visual Studio Code

Hardware Requirements:

- Processor : Intel i5
- Ram : 16 GB
- Hard Disk : 500GB

Methodology

Classes Definition

- Renovation Plan Class:

This class represents an individual renovation plan. Each plan includes details like the renovation project ID, description, start date, end date, and budget.

- Attributes:

Project_id: Unique identifier for each renovation project.

description: Description of the renovation work.

start_date: Planned start date of the renovation.

end_date: Expected end date of the renovation.

budget: Estimated budget for the renovation project.

- Methods:

- __init__ (self, project_id, description, start_date, end_date, budget): Initializes a renovation plan with the given attributes.
- update_plan (self, description, start_date, end_date, budget): Updates the details of the renovation plan.
- display_plan(self): Displays the details of the renovation plan.
- TheaterRenovationPlanner Class:

This class handles the overall management of renovation plans, including CRUD operations, project planning, and monitoring the renovation impacts.

➢ Methods:

- add_plan (self, project_id, description, start_date, end_date, budget): Adds a new renovation plan.
- remove_plan (self, project_id): Deletes an existing renovation plan by its ID.
- update_plan (self, project_id, description, start_date, end_date, budget): Updates a renovation plan by its ID.
- list_all_plans (self): Lists all renovation plans.
- plan_theater_renovations (self, project_id): Schedules the renovation based on the aging condition of the theatre.

Data Structures:

- Lists:

Used to store multiple renovation plans. Each renovation plan is represented by an object and stored within the renovation_plans list.

- Dictionaries:

For tracking impacts of renovations (e.g., audience satisfaction and attendance), dictionaries can be used to map project IDs to impact data.

- Search & Sorting:

To manage large collections of renovation plans, sorting algorithms can be implemented based on the renovation start date or budget. Search functionality allows users to quickly locate a specific renovation plan by project ID.

Result

Code:

```
class RenovationPlan:
    def __init__(self, project_id, description, start_date, end_date, budget):
    self.project_id = project_id
    self.description = description
    self.start_date = start_date
    self.end_date = end_date
    self.budget = budget

    def update_plan (self, description, start_date, end_date, budget):
    self.description = description
    self.start_date = start_date
    self.end_date = end_date
    self.budget = budget

def display_plan(self):
    return f"Project ID: {self. project_id}, Description: {self.description}, " \
        f"Start Date: {self.start_date}, End Date: {self.end_date}, Budget: {self.budget}"
```

```
# Defining the TheaterRenovationPlanner class to manage renovation plans and monitor impacts
 class TheaterRenovationPlanner:
    def__init__(self):
        self. renovation_plans = []  # List to store renovation plans


    # Add a new renovation plan
    def add_plan (self, project_id, description, start_date, end_date, budget, user_input=False):
        new_plan = RenovationPlan (project_id, description, start_date, end_date, budget)
        self.renovation_plans.append(new_plan)
        if user_input:
            print(f"\nThank you! Your renovation plan has been successfully added: {new_plan.display_plan()}")
        else:
            print(f"\nRenovation Plan Added: {new_plan.display_plan()}")


    # Remove a renovation plan by project_id
    def remove_plan(self, project_id):
        self.renovation_plans = [plan for plan in self.renovation_plans if plan.project_id != project_id]
```

```
        print(f"\nRenovation Plan with Project ID {project_id} has been removed.")
    def menu():
    planner = TheaterRenovationPlanner()


    # Predefined plans (5-7 predefined plans)
    predefined_plans = [
        (1, "Stage renovation", "2023-01-01", "2023-03-01", 50000),
        (2, "Lighting upgrade", "2023-04-01", "2023-05-01", 30000),
        (3, "Seating replacement", "2023-06-01", "2023-07-01", 75000),
        (4, "Sound system upgrade", "2023-08-01", "2023-09-15", 45000),
        (5, "Lobby renovation", "2023-10-01", "2023-11-01", 60000),
        (6, "Projection system upgrade", "2023-11-15", "2023-12-15", 40000),
        (7, "Fire safety upgrade", "2023-12-20", "2024-01-30", 35000)
    ]
}
menu()
```

Output Screen Layouts:

```
------ Theater Renovation Planner Menu ------
1. Add a new renovation plan
2. Update an existing renovation plan
3. Remove a renovation plan
4. List all renovation plans
5. Plan and schedule a renovation
6. Monitor renovation impacts
7. Exit

Select an option (1-7):
```

```
Select an option (1-7): 4

Listing All Renovation Plans:
Project ID: 1, Description: Stage renovation, Start Date: 2023-01-01, End Date: 2023-03-01, Budget: 5000
0
Project ID: 2, Description: Lighting upgrade, Start Date: 2023-04-01, End Date: 2023-05-01, Budget: 3000
0
Project ID: 3, Description: Seating replacement, Start Date: 2023-06-01, End Date: 2023-07-01, Budget: 7
5000
Project ID: 4, Description: Sound system upgrade, Start Date: 2023-08-01, End Date: 2023-09-15, Budget:
45000
```

```
Enter the number of the plan you want to select: 3

Planning Renovation for Project ID: 3
Start Date: 2023-06-01, End Date: 2023-07-01, Budget: 75000
Renovation for Project 3 has been scheduled.
```

Conclusion

The Theatre Renovation Planner POC was developed with the goal of streamlining and automating the management of renovation plans for aging theatre's. Through the use of Python and object-oriented programming.

we implemented key functionalities such as CRUD operations for managing renovation plans, scheduling and planning theatre renovations, and monitoring the impact of these renovations on theatre attendance and customer satisfaction. The project allows users to efficiently manage both predefined and user-added renovation plans, thus enhancing flexibility and user engagement.

It improves operational efficiency, audience experience, and resource management for theaters.

Future Enhancement

- Advanced Database Management: Use PostgreSQL or MongoDB for scalable data storage.

- AI for Predictive Maintenance: Implement machine learning to predict renovation needs based on trends.

- 3D Visualization of Renovations: Provide a 3D view of the theatre layout, allowing users to visualize renovation plans, seating arrangements, and aesthetic changes before implementation.

- NLP-Based User Interaction: Integrate voice or text-based AI chatbots for easy interaction

- AI-Powered Data Analytics: Use AI for advanced analytics on renovation impact, aiding decision-making.

References

- Python Software Foundation. Python Documentation. Available at: https://docs.python.org/
- Lutz, M. (2013). Learning Python, 5th Edition. O'Reilly Media.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd Edition). The MIT Press.
- Python Testing with unittest. Available at: https://docs.python.org/3/library/unittest.html
- Sommerville, I. (2015). Software Engineering, 10th Edition. Pearson.
- Elmasri, R., & Navathe, S. B. (2015). Fundamentals of Database Systems, 7th Edition. Pearson.
- https://www.greeksforgeeks.org