BASAVARAJESHWARI GROUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

AUTONOMOUS INSTITUTE UNDER VTU, BELAGAVI

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

PYTHON MINI PROJECT ON

"THEATRE ASSET MANAGEMENT"

Submitted by

Name :SINCHANA.M **USN:3BR23EE095**

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited institution*

(Recognised by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to VTU, Belagavi)

"Jnana Gangotri"Campus,No.873/2,Ballari-Hospet Road,Allipur,Ballari-583104 Karnataka, India.

INTRODUCTION

Theatre asset management in Python involves creating systems and tools to efficiently track and manage the various assets used in theatrical productions. This includes items such as costumes, props, sets, and lighting equipment. By utilizing Python's libraries and frameworks, you can develop applications to inventory assets, schedule usage, and manage maintenance.

Theatre asset management is a critical aspect of managing the myriad resources involved in producing live performances. It encompasses the systematic oversight of both tangible assets—such as costumes, props, and technical equipment—and intangible assets like intellectual property and brand reputation. Effective asset management ensures that theatres operate efficiently, maintain financial health, and create engaging experiences for audiences.

In today's dynamic theatrical landscape, where productions often face budget constraints and heightened audience expectations, the need for strategic asset management is more important than ever. By optimizing resource allocation, enhancing maintenance practices, and leveraging data analytics, theatre managers can maximize the utility and lifespan of their assets.

This approach not only supports seamless production processes but also fosters sustainability and innovation within the theatre. Ultimately, effective theatre asset management is key to delivering high-quality performances while navigating the complex demands of the industry.

Key Components

Inventory Management:

- Track items like props, costumes, and equipment.
- Include details like item name, description, quantity, condition, and location.

Check-In/Check-Out System:

- Allow users to check out items for specific productions.
- Track who has what item and due dates for returns.

User Management:

- Create different user roles (e.g., admins, stage managers, cast).
- Allow users to log in and manage their assigned assets.

Reporting:

- Generate reports on asset usage, current inventory status, and overdue items.
- Provide insights into asset lifecycle and cost management.

Notifications:

- Set up alerts for overdue items or low stock levels.
- Notify users of upcoming productions and asset needs.

What a typical Theatre Asset Management POC can offer?

A typical Theatre Asset Management proof of concept (PoC) can offer several essential features to demonstrate the value of managing theatre assets effectively. Here are the key offerings you might consider:

1. Asset Inventory Management

- Cataloging:A comprehensive database of all assets, including props, costumes, lighting, and sound equipment.
- Details: Each asset entry should include fields such as name, description, quantity, condition, location, and associated production.

2. Check-In/Check-Out System

- User-Friendly Interface: Easy mechanisms for checking items in and out.
- Tracking: Record who has checked out each item, when, and when it is due back.
- Availability Status: Indicate which items are currently available and which are checked out.

3. User Management

- Role-Based Access: Different access levels for administrators, stage managers, and cast members.
- User Profiles: Ability to create and manage user accounts with associated permissions.

4. Production Management

- Production Records: Create and manage production-specific asset lists.
- Scheduling: Integrate production dates and relevant asset needs.

5. Reporting and Analytics

- Inventory Reports: Generate reports on current inventory status, usage statistics, and asset conditions.
- Utilization Tracking: Analyze which assets are most used and which may need maintenance or replacement.

6. Notifications and Alerts

- Due Date Reminders: Automated notifications for items nearing their return dates.
- Low Stock Alerts: Notify when inventory levels fall below a certain threshold.

7. Search and Filter Functionality

- Search Options: Quickly find specific assets based on keywords or categories.
- Filtering: Allow users to filter assets by type, location, condition, or availability.

8. Data Exporting

- Export Options: Ability to export inventory data and reports in formats like CSV or PDF for offline analysis and sharing.

9. Mobile Access (Optional)

- Responsive Design: If web-based, ensure it's mobile-friendly for access on the go.
- Mobile App: A simple app for checking in/out assets on-site.

10. User Feedback Mechanism

- Comments and Ratings: Allow users to provide feedback on asset condition and functionality.

Implementation Considerations

- Technology Stack: Choose appropriate technologies (Python, Flask/Django, SQLite/PostgreSQL).
- User Testing: Collect feedback from potential users to refine features.
- Scalability: Design with future growth in mind, accommodating more assets or users easily.

The Purpose of Building Code

Building a codebase for Theatre Asset Management serves several important purposes:

1. Streamlined Asset Tracking

- Efficient Management: Automate the tracking of various assets (props, costumes, equipment) to reduce manual errors and time spent searching for items.
- Real-Time Inventory: Maintain an up-to-date inventory of available and checked-out items, ensuring that everyone knows what's available.

2. Improved Resource Utilization

- Maximize Usage: Identify which assets are frequently used and which are underutilized, enabling better resource allocation and decision-making.
- Cost Efficiency: Prevent unnecessary purchases of duplicate items by keeping a clear record of what is available.

3. Enhanced Collaboration

- Role-Based Access: Different users (admins, stage managers, cast) can access relevant information based on their roles, improving communication and coordination.
- Shared Access: Facilitate collaboration among team members, allowing for better planning and execution of productions.

CODE:

```
class Asset:
    def _init_(self, name, asset_type):
        self.name = name
        self.asset_type = asset_type
        self.maintenance_history = []

    def add_maintenance(self, date, details):
        self.maintenance_history.append((date, details))

    def _str_(self):
        return f"{self.name} ({self.asset_type})"


class Booking:
    def _init_(self, asset_name, date):
        self.asset_name = asset_name
        self.date = date

    def _str_(self):
        return f"Booking for {self.asset_name} on {self.date}"


class AssetManager:
    def _init_(self):
        self.assets = []
        self.bookings = []

    def add_asset(self, name, asset_type):
        asset = Asset(name, asset_type)
        self.assets.append(asset)
        print(f"Added asset: {asset}")

    def track_maintenance(self, asset_name, date, details):
        for asset in self.assets:
            if asset.name == asset_name:
                asset.add_maintenance(date, details)
```

```
                print(f"Logged maintenance for {asset_name}: {details}")
                return
        print("Asset not found.")

    def book_asset(self, asset_name, date):
        for asset in self.assets:
            if asset.name == asset_name:
                booking = Booking(asset_name, date)
                self.bookings.append(booking)
                print(f"Successfully booked: {booking}")
                return
        print("Asset not found.")

    def cancel_booking(self, asset_name, date):
        for booking in self.bookings:
            if booking.asset_name == asset_name and booking.date == date:
                self.bookings.remove(booking)
                print(f"Cancelled booking: {booking}")
                return
        print("Booking not found.")

    def generate_report(self):
        print("\n--- Asset Report ---")
        for asset in self.assets:
            print(f"Asset: {asset.name}, Type: {asset.asset_type}")
            if asset.maintenance_history:
                print("  Maintenance History:")
                for date, details in asset.maintenance_history:
                    print(f"    - {date}: {details}")
            else:
                print("  No maintenance history.")

        print("\n--- Bookings ---")
        if not self.bookings:
            print("  No bookings found.")
        for booking in self.bookings:
print(booking)


    def main():
        manager = AssetManager()

        while True:
            print("\n1. Add Asset")
            print("2. Track Maintenance")
```

```
        print("3. Book Asset")
        print("4. Cancel Booking")
        print("5. Generate Report")
        print("6. Exit")
        choice = input("Choose an option: ")

        if choice == '1':
            name = input("Enter asset name: ")
            asset_type = input("Enter asset type (e.g., Lighting, Sound): ")
            manager.add_asset(name, asset_type)
        elif choice == '2':
            asset_name = input("Enter asset name for maintenance: ")
            date = input("Enter maintenance date (YYYY-MM-DD): ")
            details = input("Enter maintenance details: ")
            manager.track_maintenance(asset_name, date, details)
        elif choice == '3':
            asset_name = input("Enter asset name to book: ")
            date = input("Enter booking date (YYYY-MM-DD): ")
            manager.book_asset(asset_name, date)
        elif choice == '4':
            asset_name = input("Enter asset name to cancel booking: ")
            date = input("Enter booking date (YYYY-MM-DD): ")
            manager.cancel_booking(asset_name, date)
        elif choice == '5':
            manager.generate_report()
        elif choice == '6':
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please try again.")


if _name_ == "_main_":
    main()
```

OUTPUT

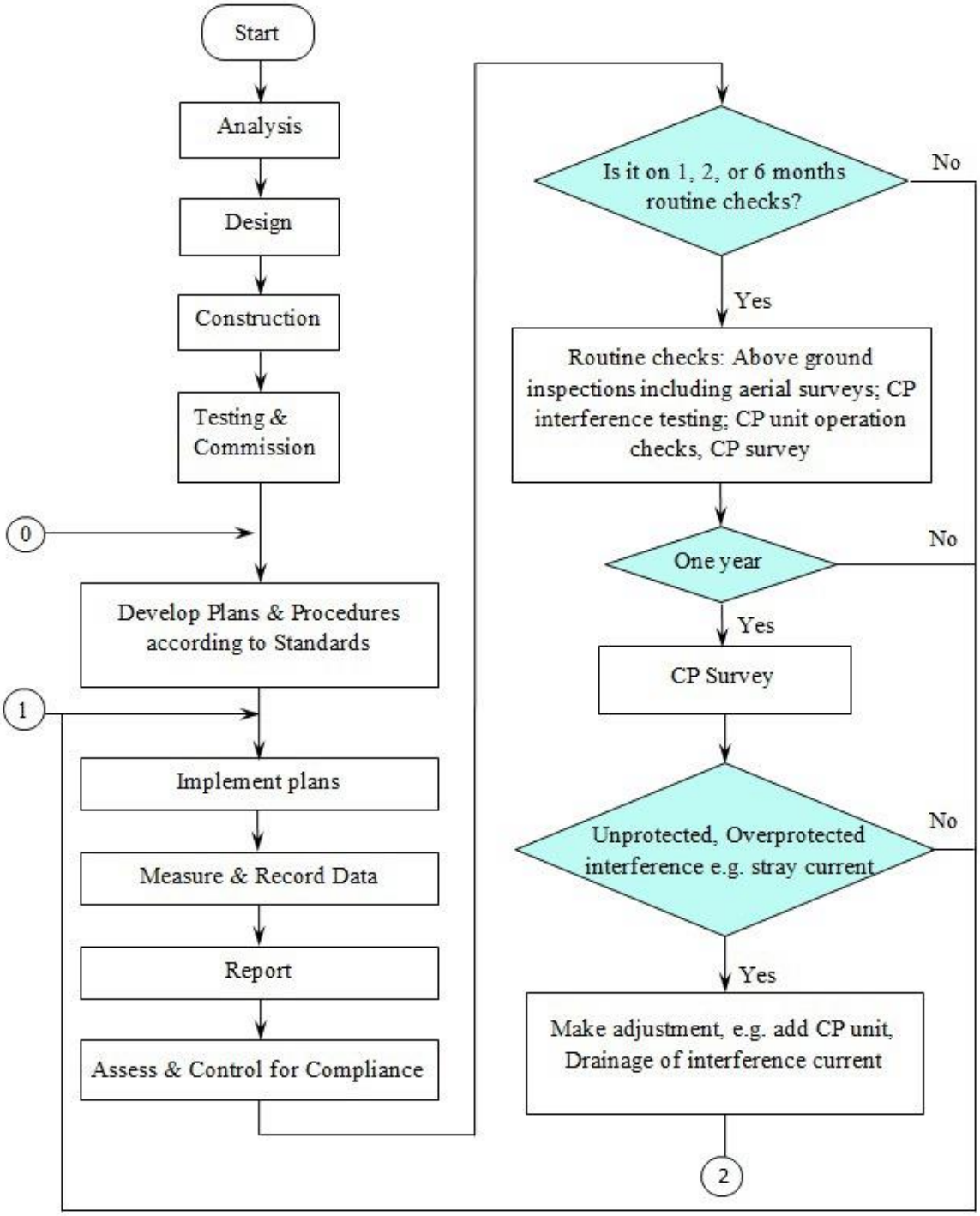
```
PS C:\Users\USER> &
C:/Users/USER/AppData/Local/Programs/Python/Python312/python.exe
c:/Users/USER/Desktop/python/hello.py

1. Add Asset
2. Track Maintenance
3. Book Asset
4. Cancel Booking
```

5. Generate Report
6. Exit
Choose an option: 2
Enter asset name for maintenance: abc
Enter maintenance date (YYYY-MM-DD): 2000-02-12
Enter maintenance details: cost
Asset not found.

1. Add Asset
2. Track Maintenance
3. Book Asset
4. Cancel Booking
5. Generate Report
6. Exit
Choose an option:

FLOWCHART:


Start
Analysis
Design
Construction
Testing & Commission
0
Develop Plans & Procedures according to Standards
1
Implement plans
Measure & Record Data
Report
Assess & Control for Compliance
Is it on 1, 2, or 6 months routine checks?
No
Yes
Routine checks: Above ground inspections including aerial surveys; CP interference testing; CP unit operation checks, CP survey
One year
No
Yes
CP Survey
Unprotected, Overprotected interference e.g. stray current
No
Yes
Make adjustment, e.g. add CP unit, Drainage of interference current
2

ALGORITHM:

The provided code doesn't implement a complex algorithm but rather a set of functions to manage assets and bookings. Here's a breakdown of the logic:

Data Structures:

- **Asset:** Represents an asset with properties like name, type, and a list to store maintenance history.
- **Booking:** Represents a booking for an asset with its name and date.
- **AssetManager:** Manages a collection of assets and bookings.

Functions:

- **add_asset:** Creates a new Asset object and adds it to the manager's list.
- **track_maintenance:** Finds an asset by name, adds a new maintenance record with date and details to its history.
- **book_asset:** Finds an asset by name, creates a Booking object with the date, and adds it to the manager's list.
- **cancel_booking:** Finds a booking by asset name and date, and removes it from the manager's list.
- **generate_report:** Loops through all assets, prints their details and maintenance history (if any). Then loops through bookings and prints their information.

Main Loop:

- Presents a menu with options to add assets, track maintenance, book assets, cancel bookings, generate a report, or exit.
- Based on user input, calls the corresponding function with user-provided details.

Overall, the code doesn't involve a specific algorithm beyond standard data structure manipulation and basic search functionalities.

Here are some potential algorithmic improvements you could consider:

- **Availability Check:** When booking an asset, check if it's already booked on the chosen date.
- **Search Optimization:** Use data structures like dictionaries for faster asset and booking lookups.

- **Alerts and Notifications:** Implement logic to send alerts for upcoming maintenance or conflicting bookings.

These are just a few ideas, and the specific algorithms you use would depend on the desired functionalities of your asset management system.

Upload an image

This prompt requires an image that you need to add. Tap the image button to upload an image.

Got it

Do you need a little help with this prompt?

Power up your prompt and Gemini will expand it to get you better results

Got it

PSEUDOCODE:

Pseudocode for Asset Management System

Data Structures:

Asset: Stores information about an asset (name, type, maintenance history).

name: String representing the asset name.

asset_type: String representing the asset type (e.g., Lighting, Sound).

maintenance_history: List of tuples containing maintenance date and details.

Booking: Stores information about a booking (asset name, date).

asset_name: String representing the asset name.

date: String representing the booking date (YYYY-MM-DD).

AssetManager: Manages assets and bookings.

assets: List of Asset objects.

bookings: List of Booking objects.

Functions:

add_asset(name, asset_type):

Create a new Asset object with provided name and type.

Add the new asset to the assets list of the AssetManager.

Print a message confirming the addition.

track_maintenance(asset_name, date, details):

Loop through each Asset in the assets list.

If the asset.name matches the provided asset_name:

Add a new tuple containing date and details to the asset.maintenance_history list.

Print a message confirming the maintenance tracking.

If no matching asset is found, print a message indicating the asset was not found.

book_asset(asset_name, date):

Loop through each Asset in the assets list.

If the asset.name matches the provided asset_name:

Create a new Booking object with the provided asset_name and date.

Add the new booking to the bookings list of the AssetManager.

Print a message confirming the successful booking.

If no matching asset is found, print a message indicating the asset was not found.

cancel_booking(asset_name, date):

Loop through each Booking in the bookings list.

If the booking.asset_name matches the provided asset_name and booking.date matches the provided date:

Remove the booking from the bookings list.

Print a message confirming the cancellation.

If no matching booking is found, print a message indicating the booking was not found.

generate_report():

Print a header for the asset report.

Loop through each Asset in the assets list.

Print the asset name and type.

If the asset.maintenance_history list is not empty:

Print a sub-header for maintenance history.

Loop through each tuple in asset.maintenance_history:

Print the date and details of the maintenance event.

Otherwise, print a message indicating no maintenance history exists.

Print a header for bookings.

If the bookings list is empty, print a message indicating no bookings found.

Loop through each Booking in the bookings list:

Print information about the booking.

Main Program:

Create an instance of AssetManager.

Enter a loop that continues until the user chooses to exit.

Display a menu with options for adding assets, tracking maintenance, booking assets, cancelling bookings, generating reports, and exiting.

Get the user's choice.

Based on the user's choice, call the corresponding function from the AssetManager with appropriate arguments.

When the user chooses to exit, print a message and exit the program.

CONCLUSION

In conclusion, a Theatre Asset Management system is essential for enhancing the efficiency and effectiveness of theatre operations. By implementing a structured approach to managing assets—such as props, costumes, and equipment—this system can significantly improve inventory control, accountability, and resource utilization.

REFERENCE:

1. https://asapsystems.com/case-studies/horton-plaza-theaters-foundation-case-study/
2. https://www.geeksforgeeks.org/movie-tickets-booking-management-system-in-python/
3. https://dataheadhunters.com/academy/how-to-create-an-asset-management-tool-in-python-for-businesses/