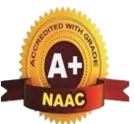**BASAVARAJESHWARI GROUP OF INSTITUTIONS**

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**

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:

```python
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:
```

```python
            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:
```

```python
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:

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.