KISHKINDA UNIVERSITY, Ballari

Mini Project Report

On

"Theater Event Calendar POC"

Department of B Tech

Submitted By:

Yasmeen Banu	**TEMPBTech-CSE024**
Jeniffer Rakshitha B	**TEMPBTech-CSE042**
Ankitha	**TEMPBTech-CSE089**
Shwetha Awati	**TEMPBTech-CSE075**
Spandana C	**TEMPBTech-CSE100**

Table of Contents:

1.Introduction:

A Theatre Event Calendar is a organized schedule that lists and promotes various theatrical performances, events, and activities over a specific period.

Purpose:

1. Provide a centralized platform for theatre enthusiasts to discover upcoming events.
2. Facilitate event planning and organization for theatre companies and venues.
3. Promote local theatre productions and support the performing arts community.

Key Features:

1. Event listings with dates, times, and descriptions.
2. Venue information, including address and seating capacity.
3. Ticket pricing and availability.
4. Cast and crew information.
5. Genre classification (e.g., drama, comedy, musical).

Types of Events:

1. Plays
2. Musicals
3. Dance performances
4. Opera productions
5. Comedy shows
6. Theatre festival

Benefits:

1. Increased visibility for theatre productions.
2. Enhanced discoverability for audiences.
3. Improved event planning and organization.
4. Support for local arts and culture.

Platforms:

1. Online calendars (e.g., websites, social media)
2. Print calendars (e.g., brochures, posters)
3. Mobile apps

2.Objective:

Primary Objectives:

1. Promote theatre events and productions to target audiences.
2. Provide a centralized platform for event information and scheduling.
3. Increase visibility and discoverability for theatre companies and venues.
4. Facilitate event planning and organization for theatre professionals.
5. Enhance audience engagement and participation.

Marketing Objectives:

1. Increase ticket sales and revenue growth.
2. Boost brand awareness and reputation.
3. Diversify audience demographics and engagement.
4. Foster partnerships with local businesses and organizations.
5. Support educational and outreach programs.

Operational Objectives:

1. Ensure accuracy and timeliness of event information.
2. Maintain a user-friendly and accessible calendar interface.
3. Provide regular updates and notifications.
4. Monitor and analyze event attendance and engagement metrics.
5. Continuously evaluate and improve the calendar's effectiveness.

Audience Development Objectives:

1. Attract new audiences and increase attendance.
2. Retain existing audiences and encourage repeat attendance.
3. Foster a sense of community among theatre enthusiasts.
4. Provide opportunities for audience engagement and participation.
5. Encourage feedback and suggestions.

Artistic Objectives:

1. Showcase diverse and high-quality theatre productions.
2. Support local and emerging artists.
3. Foster creative collaborations and partnerships.
4. Encourage innovative and experimental theatre practices.

5. Promote cultural exchange and understanding.

Financial Objectives:

1. Increase ticket revenue and sales.
2. Secure sponsorships and funding.
3. Reduce marketing and advertising expenses.
4. Improve operational efficiency and cost-effectiveness.
5. Ensure long-term financial sustainability.

Technological Objectives:

1. Develop a user-friendly and accessible calendar interface.
2. Integrate social media and online ticketing.
3. Utilize data analytics and reporting tools.
4. Ensure mobile-friendliness and responsiveness.
5. Maintain data security and integrity.

3.Methodology

Methodologies:

1. Object-Oriented Programming (OOP)
2. Model-View-Controller (MVC) architecture
3. Agile Project Management

Tools and Software:

1. Python libraries:
   - datetime for date and time management
   - calendar for calendar-related functions
   - pandas for data manipulation and analysis

2. Web frameworks:
   - Flask
   - Django
   - Pyramid
3. Database management:
   - SQLite
   - PostgreSQL
   - MySQL
4. APIs and integrations:
   - Ticketing APIs (e.g., Ticketmaster, Eventbrite)
   - Social Media APIs(e.g., Facebook, Twitter)
   - Payment Gateways (e.g., Stripe, PayPal)

Techniques:

1. Data scraping and crawling
2. Data visualization (e.g., Matplotlib, Seaborn)
3. Automated reminders and notifications
4. Search Engine Optimization (SEO)
5. User Experience (UX) design

Python Libraries for Event Calendar:

1. schedule
2. calendar data
3. event calendar
4. pytz(for time zone management)
5. icalendar(for iCalendar format support)

A theatre event calendar typically follows a structured process to manage and coordinate various events, performances, and productions. Here's an overview of the processes, algorithms, and workflows involved

Processes:

1. Event Planning: Identifying and scheduling events, performances, and productions.
2. Scheduling: Creating and managing calendars for venues, performers, and staff.
3. Ticketing: Managing ticket sales, inventory, and customer information.
4. Marketing: Promoting events through various channels (social media, email, print).
5. Logistics: Coordinating venue setup, technical requirements, and front-of-house operations.
6. Financial Management: Tracking revenue, expenses, and budgeting.

Algorithms:

1. Scheduling Algorithms:
   - Constraint-based scheduling (e.g., ensuring no conflicts between events).
   - Resource allocation (e.g., assigning staff, equipment).
   - Optimization algorithms (e.g., maximizing venue usage).
2. Ticketing Algorithms:
   - Seat allocation and pricing strategies.
   - Ticket availability and inventory management.
   - Dynamic pricing (adjusting prices based on demand).
3. Marketing Algorithms:
   - Target audience segmentation.
   - Personalized promotional messaging

- Social media analytics.

Event Planning Workflow:

1. Event proposal submission.
2. Review and approval.
3. Scheduling and calendar management.
4. Contract negotiation and signing.
5. Event coordination and logistics.

Scheduling Workflow:

1. Create event schedule.
2. Assign venues, performers, and staff.
3. Check for conflicts and resolve.
4. Publish schedule to stakeholders.

Ticketing Workflow:

1. Set ticket prices and availability.
2. Create ticket sales channels (online, box office).
3. Manage ticket sales and inventory.
4. Process refunds and exchanges.

Marketing Workflow:

1. Identify target audience.
2. Create promotional materials.
3. Schedule social media posts.
4. Send targeted email campaigns.
5. Monitor analytics.

Key Performance Indicators (KPIs):

1. Event attendance and revenue.
2. Ticket sales and conversion rates.
3. Customer satisfaction ratings.
4. Social media engagement metrics.
5. Financial performance (budget variance).

4.Results/Findings

Code Analysis:

1**. Event Class:**

- Well-structured and concise.
- Uses datetime module for date and time parsing.
- __repr__ method provides a readable string representation.

2.**EventCalendar Class:**

- Effectively manages events using a dictionary.
- Methods for adding, updating, deleting, and listing events.
- __repr__ method provides a summary of the calendar.

3.**OnlinePlatformSyncClass:**

- Simulates event synchronization with an online platform.
- Simple implementation, but can be extended for actual API integration.

Test Analysis:

The provided test suite (Test Event Management) covers essential scenarios:

1.test_event_creation: Verifies event object creation.

2.test_add_event: Checks event addition to the calendar.

3.test_update_event: Tests event updates.

4.test_delete_event: Confirms event deletion.

5.test_sync_events: Simulates event synchronization.

T**est Results:**

All tests pass, indicating the implementation is correct.

Suggestions for Improvement:

1.Error Handling: Enhance error handling in EventCalendar methods to provide more informative error messages.

2.Validation: Add input validation for Event attributes (e.g., date format, time range).

3.OnlinePlatformSync: Implement actual API integration or mock API calls for more realistic testing.

4. Additional Tests: Consider testing edge cases, such as:

- Duplicate event IDs.
- Invalid date formats.
- Empty event names.
- Overlapping event times.

5.Conclusion

Project Overview:

The Theatre Event Calendar project aims to design and implement a system for managing theatre events, including scheduling, ticketing, and synchronization with online platforms.

Main Points:

1. Event and Event Calendar classes for managing events.
2. Online Platform Sync class for simulating event synchronization.
3. Unit tests for ensuring correctness.
4. Implementation of event CRUD (Create, Read, Update, Delete) operations.
5. Basic error handling and validation.

Lessons Learned:

1. Importance of modular design (separate classes for events, calendar, and synchronization).
2. Benefits of unit testing for ensuring code reliability.
3. Need for robust error handling and validation.
4. Simulating real-world interactions (e.g., API calls) improves testing.

Future Work:

1. Integrate with actual online platforms (e.g., Ticketmaster) using APIs.
2. Implement advanced features:
   - Recurring events
   - Event reminders
   - Ticket sales tracking
   - Customer management
3. Enhance user interface (e.g., web, mobile app) for easier event management.
4. Explore machine learning for optimizing event scheduling and recommendation.
5. Implement security measures for protecting sensitive data.

Potential Extensions:

1. Integrate with payment gateways for ticket sales.
2. Develop reporting and analytics tools for event performance.
3. Create a mobile app for event management and ticket sales.
4. Implement accessibility features for diverse audiences.

Best Practices:

1. Follow SOLID principles for maintainable code.
2. Use design patterns (e.g., Repository Pattern) for data management.
3. Continuously refactor and improve code quality.
4. Write comprehensive documentation for future development

6.References

Event Management Platforms:

1. Eventbrite
2. Ticketmaster
3. Live Nation
4. Song kick

Theatre and Performing Arts Resources:

1. International Association of Venue Managers
2. National Theatre Conference
3. Theatre Communications Group
4. Performing Arts Alliance

Calendar and Scheduling Tools:

1. Google Calendar
2. Microsoft Exchange
3. iCal
4. Calendly

APIs and Integration Resources:

1. Eventbrite API
2. Ticketmaster API
3. Google Calendar API
4. Open API Initiative

Code :

```
from datetime import datetime


class Event:
    def init(self, event_id, name, start_time, end_time):
        self.event_id = event_id
        self.name = name
        self.start_time = datetime.strptime(start_time, '%Y-%m-%d %H:%M')
        self.end_time = datetime.strptime(end_time, '%Y-%m-%d %H:%M')


    def repr(self):
        return f"Event({self.event_id}, '{self.name}', '{self.start_time}',
'{self.end_time}')"
##


### 2. The EventCalendar Class


#python
class EventCalendar:
    def init(self, calendar_id):
        self.calendar_id = calendar_id
        self.events = {}


    def add_event(self, event):
        if event.event_id in self.events:
            raise ValueError("Event ID already exists")
        self.events[event.event_id] = event
```

```
    def get_event(self, event_id):
        return self.events.get(event_id)

    def update_event(self, event_id, **kwargs):
        if event_id not in self.events:
            raise ValueError("Event not found")

        event = self.events[event_id]
        if 'name' in kwargs:
            event.name = kwargs['name']
        if 'start_time' in kwargs:
            event.start_time = datetime.strptime(kwargs['start_time'], '%Y-%m-
%d %H:%M')
        if 'end_time' in kwargs:
            event.end_time = datetime.strptime(kwargs['end_time'], '%Y-%m-
%d %H:%M')

    def delete_event(self, event_id):
        if event_id in self.events:
            del self.events[event_id]
        else:
            raise ValueError("Event not found")

    def list_events(self):
        return list(self.events.values())

    def repr(self):
```

```
        return f"EventCalendar('{self.calendar_id}', Events={len(self.events)})"
#

### 3. The OnlinePlatformSync Class

#python
class OnlinePlatformSync:
    def sync_events_with_platforms(self, calendar, platform_data):
        # Fake synchronization logic
        print("Syncing events:")
        for event in calendar.list_events():
            print(f" - {event.name} synced to platform
{platform_data['platform_name']}")
#

### Unit Tests

#We need to ensure our classes work correctly. We'll use unittest for writing the
test cases.

#python
import unittest

class TestEventManagement(unittest.TestCase):
    def setUp(self):
        self.calendar = EventCalendar('Theater123')
        self.event = Event('1', 'Hamlet', '2023-04-10 18:00', '2023-04-10 21:00')
        self.calendar.add_event(self.event)
```

```
    def test_event_creation(self):
        self.assertEqual(repr(self.event), "Event(1, 'Hamlet', '2023-04-10 18:00:00', '2023-04-10 21:00:00')")

    def test_add_event(self):
        self.assertEqual(len(self.calendar.events), 1)

    def test_update_event(self):
        self.calendar.update_event('1', name='Hamlet Revised')
        self.assertEqual(self.calendar.get_event('1').name, 'Hamlet Revised')

    def test_delete_event(self):
        self.calendar.delete_event('1')
        self.assertEqual(len(self.calendar.events), 0)

    def test_sync_events(self):
        sync = OnlinePlatformSync()
        with self.assertLogs() as cm:
            sync.sync_events_with_platforms(self.calendar, {'platform_name': 'Ticketmaster'})
            self.assertIn('Hamlet synced', cm.output[0])
if _name_ == 'main':
    unittest.main()
```