Case Study

Moveinsync- Intelligent Floor Plan Management System for a Seamless Workspace Experience

1. Introduction-

Moveinsync introduces an Intelligent Floor Plan Management System, reshaping the workspace experience. This case study highlights our commitment to seamless floor plan management, emphasizing resilience, user-friendliness, secure authentication, and efficient system recovery. The subsequent sections detail our approach, aiming to set a new standard for adaptable and efficient workspaces.

2. Solution Proposed-

The proposed solution aims to create an intelligent and user-centric Floor Plan Management System focusing on effective meeting scheduling. The system caters to three user divisions based on their roles and permissions:

2.1) Users:

Users without Meeting Scheduling Access:

- Individuals who do not have access to schedule meetings.

Users with Meeting Scheduling Access (Non-admin):

- Employees with access to schedule meetings but without administrative privileges.

Administrators (Admins):

- Users with the authority to schedule, reschedule, update, and delete multiple meetings concurrently.

2.2) Entities (Tables):

User-

```python
class User:
    def __init__(self, username, email_id, name, department, designation):
        self.username = username
        self.email_id = email_id
        self.name = name
        self.department = department
        self.designation = designation

    def __str__(self):
        return f"Username: {self.username}, EmailID: {self.email_id},
        Name: {self.name}, Department: {self.department},
        Designation: {self.designation}"
```

Meeting Rooms-

```python
class MeetingRoom:
    def __init__(self, room_id, address, capacity):
        # Parameters:
        # - room_id (int): Unique identifier for the meeting room.
        # - address (str): Address of the meeting room.
        # - capacity (int): Maximum capacity of the meeting room.

        self.RoomId = room_id
        self.Address = address
        self.Capacity = capacity
```

Meetings-

```python
class Meeting:
    # Class variable to keep track of the last assigned meeting_id
    last_meeting_id = 0
    def __init__(self, scheduled_by, attenders, start_time_date, end_time_date, meeting_room_id, status="Not_started"):
        """
        Initialize a Meeting instance.

        Parameters:
        - meeting_id (int): Unique identifier for the meeting.
        - scheduled_by (str): Username of the user who scheduled the meeting.
        - attenders (list): List of usernames of attendees.
        - start_time_date (str): Start date and time of the meeting in the format "YYYY-MM-DD HH:MM".
        - end_time_date (str): End date and time of the meeting in the format "YYYY-MM-DD HH:MM".
        - meeting_room_id (int): Foreign key referencing the Meeting Room where the meeting takes place.
        - status (str): Not started yet, Ongoing, Finished and Pending
        """
        Meeting.last_meeting_id += 1

        self.meeting_id = Meeting.last_meeting_id # new meeting id = last meeting id + 1
        self.scheduled_by = scheduled_by
        self.attenders = attenders
        self.start_time_date = start_time_date
        self.end_time_date = end_time_date
        self.meeting_room_id = meeting_room_id
        self.status = status
```

Queue-

```
class Queue:
    last_meeting_id = 0
    def __init__(self, scheduled_by, attenders, start_time_date, end_time_date, meeting_room_id, status="Pending"):

        Meeting.last_meeting_id += 1
        self.meeting_id = Meeting.last_meeting_id # new meeting id = last meeting id + 1
        self.scheduled_by = scheduled_by
        self.attenders = attenders
        self.start_time_date = start_time_date
        self.end_time_date = end_time_date
        self.status = status
```

2.3) Workflow:

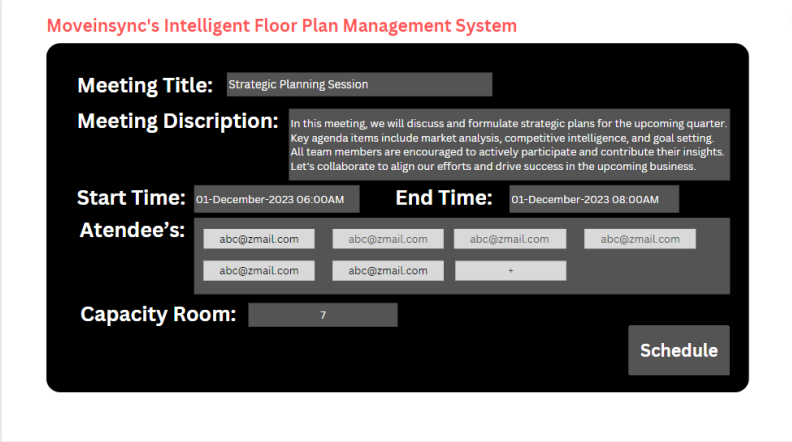
User Type 2 (Non-admin) Interaction:

- Users initiate the process by entering meeting details such as capacity, start time, end time, and member's emails.
- The algorithm identifies suitable meeting rooms with the required capacity and availability.
- A room is allocated based on the user's historical preferences, optimizing for past usage patterns, and meeting entry will get pushed into the meeting table.
- After the successful allocation, mail will be sent to all members' email addresses.

Find Suitable Meeting Rooms

Collision:

- If another employee requires a room after all available rooms are booked, their request enters a queue (Queue Table).
- Requests in the queue are prioritized based on user designations.
- When a room becomes available (due to unscheduling), we try to Schedule a meeting from the queue according to the needed conditions(start, end, and capacity)
- In cases of urgency or unavailability, the Admin can intervene, contacting meeting schedulers to rearrange or reschedule meetings.



3. Working-

We will implement a locking mechanism to **handle concurrent requests for meeting room** bookings and ensure data consistency. The critical section, where availability and booking checks occur, will be protected using locks to **prevent race conditions**.

User Requests:

- When a user submits a meeting request through the main page, the backend initiates a process to check room availability.

Critical Section - Locking:

- A lock ensures that only one request can access the availability check and booking process.
- This prevents race conditions where multiple requests could simultaneously try to book the same room.
- The meeting is then booked for the user, and a new entry will be added to the Meetings Table.
- The lock is released, allowing the critical section to be accessed by the subsequent request.

Code Implementation

Queue Scheduling Algorithm:

- Unsuccessful meeting requests are added to the 'Queue' table.
- Periodically check for available meeting rooms for queued requests.
- When a room becomes available (due to unscheduling), we try to Schedule a meeting from the queue according to the needed conditions(start, end, and capacity)

- In scheduling, give priority to Designation Users.
- Notify the scheduler of successful meeting scheduling.
- Delete the entry from the 'Queue' table after successful scheduling.
- Save a new entry in the Meetings Table

Enhanced Periodic Check Function:

- Update Status of Ongoing Meetings:
- For all meetings in the 'Meetings' table:
- Mark meetings as 'Ongoing' if the current time is within the meeting's scheduled timeframe.
- Mark meetings as 'Not Started' if the current time is before the scheduled start time.
- Mark meetings as 'Finished' if the current time is after the scheduled end time.

4. Optimizations-

Handling System Failure: Log-Based Recovery

To safeguard against system failures, implement a log-based recovery system:

- Generate logs for critical operations (e.g., meeting scheduling) with timestamps.
- Store logs persistently locally for resilience against system reboots or connectivity loss.
- Log details before executing critical operations to record changes before committing to the database.
- On system restart or reconnection, process local logs to restore the system's state.
- Reapply logged transactions to the database to reflect the recorded operations.
- Implement mechanisms for resynchronization in case of deviations between the system and logs.

Caching Optimization:

Caching mechanisms will be implemented to enhance system performance, focusing on storing frequently accessed data in the cache. Specifically:

- UI Caching:
    - Cache frequently used UI components to reduce loading times and enhance user experience.
    - Store static UI elements, layouts, or templates in the cache for quick retrieval.
- Data Caching:

    - Cache frequently accessed data, such as user preferences or commonly used meeting room details.
    - Utilize caching to minimize database queries and improve overall system responsiveness.

Authentication Security Highlights:

- Store passwords securely with hashing and salting.
- Implement short-lived session tokens and reauthentication.
- Temporarily lock accounts after failed login attempts.
- Maintain logs for monitoring and response to suspicious activities.

Cost Estimation - Time and Space:

- The system employs carefully crafted algorithms, prioritizing swift execution for tasks like meeting scheduling and conflict resolution.
- Strategic use of optimized data structures minimizes computational overhead, enhancing overall time efficiency.
- Thorough evaluation and optimization of space complexity ensure judicious resource utilization and cost-efficient memory management.

5. System monitoring-

- **Comprehensive Monitoring:**
    - Implement monitoring tools to track system performance in real-time.

- **Dashboard for Admins:**
    - Develop a dedicated dashboard accessible only to admins.
    - Include critical metrics like ongoing meetings, total users in meetings, upcoming meetings, and meeting room occupancy.

- **Real-time Updates:**
    - Utilize real-time logging mechanisms to identify and address system issues promptly.
    - Ensure the dashboard provides instant updates on system status.

- **User Interaction:**
    - Introduce a user-request feature allowing users to inquire about the availability of a specific person.
    - Admins receive these requests and respond with information on the person's availability.

6. Conclusion-

Our Intelligent Floor Plan Management System ensures a secure and efficient workspace. With robust authentication, system optimizations, and user-centric features, it creates a seamless experience. Key elements include multi-factor authentication, caching for performance, and a real-time monitoring dashboard for admins. This concise solution delivers a dynamic and reliable workspace environment.

Thank you-

Bhavishya Tiwari
200003019@iiti.ac.in