**7. CISC Calendar System**

**a. Purpose**

The **CISC Calendar System** serves as a unified scheduling hub. It consolidates schedules from academic events, class timetables, student organization activities, and official college announcements. It prevents conflicts, ensures transparency, and helps all stakeholders remain informed of time-sensitive matters.

**b. Intended Users and Roles**

1. **Students**
   * View upcoming classes, deadlines, and official announcements.
   * Subscribe to faculty and organization calendars for visibility.
2. **Faculty**
   * Track official academic dates (exams, grading deadlines, faculty meetings).
   * Post class schedules, assignments, and exam schedules for students.
3. **Organization Officers**
   * Submit events for approval.
   * View approved schedules to avoid conflicts.
   * Manage organization-specific calendars layered with official ones.
4. **Admins**
   * Encode college-wide events (holidays, foundation days, seminars).
   * Supervise layering (academic, organizational, administrative).
   * Approve or reject proposed organization events.

**c. Functional Requirements**

1. **Event Management**
   * Create, edit, and delete events with details (title, date, time, description, organizer).
   * Support recurring events (weekly classes, monthly meetings).
   * Event categorization (academic, organizational, administrative).
2. **Calendar Views**
   * Monthly, weekly, and daily views.
   * Layer-based display (toggle between academic, organizational, or global events).
3. **Event Approval Workflow**
   * Organization events require admin approval before becoming visible.
   * Faculty events auto-publish under faculty calendars.
4. **Role-Based Access Control**
   * Students: read-only (view events).
   * Faculty: add/manage class-related events.
   * Organization officers: propose/manage organization events (subject to approval).
   * Admins: full control of calendar layers.
5. **Conflict Detection**
   * System automatically flags overlapping events within the same scope (class schedules, organization events, major announcements).
6. **Notifications & Reminders**
   * Email or in-app reminders for upcoming deadlines or events.
   * Push notifications for last-minute changes (desktop alerts).

**d. Technical Design**

1. **Frontend (Desktop App)**
   * Developed in **PyQt6 / PySide6**.
   * GUI layouts with Qt Designer.
   * Calendar widget integration (QCalendarWidget + custom views).
2. **Backend (Django)**
   * Event APIs for CRUD operations.
   * Role-based access endpoints with JWT authentication.
   * Business logic for event approval and conflict detection.
3. **Database (PostgreSQL)**
   * Event Table: event\_id, title, description, start\_datetime, end\_datetime, category, status, created\_by, approved\_by.
   * User Roles: student, faculty, org\_officer, admin.
   * Calendar Layers Table: links events to roles/scopes.
4. **Data Synchronization**
   * Local caching (SQLite3) for offline access.
   * Auto-sync with PostgreSQL when connected.

**e. Example Data Flow**

1. **Org Officer** creates an event → stored as pending in DB.
2. **Admin** receives notification → approves/rejects.
3. If approved → event status updated to active → reflected on all relevant calendars.
4. **Students/Faculty** see event instantly in their interface.

**f. Non-Functional Requirements**

1. **Validation**: Input (title, dates, times) must be validated on frontend & backend.
2. **Error Handling**: Graceful fallback when conflicts occur or data fails to sync.
3. **Scalability**: PostgreSQL backend supports high-volume events.
4. **Security**: Only authorized roles can modify events; RBAC enforced at API level.