# Campus Event Management Platform - Design Document

## 1. Data Model & Schema

### Core Data to Track

- **Event Lifecycle**: Creation, updates, cancellations
- **Student Interactions**: Registration, check-in, feedback
- **Analytics**: Attendance rates, popularity metrics, engagement
- **Administrative**: College management, staff actions

### Database Schema (PostgreSQL)

```
sql
```

```sql
-- Colleges table for multi-tenancy
CREATE TABLE colleges (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    code VARCHAR(10) UNIQUE NOT NULL, -- DSU, MIT, etc.
    domain VARCHAR(100) NOT NULL, -- dsu.edu.in
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Event categories for better organization
CREATE TABLE event_categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL, -- Technical, Cultural, Sports, etc.
    description TEXT
);

-- Core events table
CREATE TABLE events (
    id SERIAL PRIMARY KEY,
    event_id VARCHAR(20) UNIQUE NOT NULL, -- DSU_EVT_001 format
    college_id INTEGER REFERENCES colleges(id),
    category_id INTEGER REFERENCES event_categories(id),
    title VARCHAR(255) NOT NULL,
    description TEXT,
    venue VARCHAR(255),
    event_date DATE NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    capacity INTEGER DEFAULT 100,
    registration_deadline TIMESTAMP,
    image_url VARCHAR(500),
    created_by INTEGER, -- Reference to admin user
    status VARCHAR(20) DEFAULT 'active', -- active, cancelled, completed
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Students table
CREATE TABLE students (
    id SERIAL PRIMARY KEY,
    college_id INTEGER REFERENCES colleges(id),
    student_id VARCHAR(20) NOT NULL, -- DSU2023001 format
    email VARCHAR(255) UNIQUE NOT NULL,
    full_name VARCHAR(255) NOT NULL,
    phone VARCHAR(15),
    year_of_study INTEGER,
    department VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Event registrations
CREATE TABLE event_registrations (
    id SERIAL PRIMARY KEY,
    event_id INTEGER REFERENCES events(id),
    student_id INTEGER REFERENCES students(id),
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) DEFAULT 'registered', -- registered, cancelled, waitlist
    UNIQUE(event_id, student_id) -- Prevent duplicate registrations
);

-- Attendance tracking
CREATE TABLE event_attendance (
    id SERIAL PRIMARY KEY,
    registration_id INTEGER REFERENCES event_registrations(id),
    check_in_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    check_in_method VARCHAR(20) DEFAULT 'manual', -- manual, qr_code
    verified_by INTEGER, -- Admin who verified attendance
    UNIQUE(registration_id) -- One check-in per registration
);

-- Feedback collection
CREATE TABLE event_feedback (
    id SERIAL PRIMARY KEY,
    registration_id INTEGER REFERENCES event_registrations(id),
    rating INTEGER CHECK (rating >= 1 AND rating <= 5),
    comment TEXT,
    submitted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(registration_id) -- One feedback per registration
);

-- Indexes for performance
CREATE INDEX idx_events_college_date ON events(college_id, event_date);
CREATE INDEX idx_events_status ON events(status);
CREATE INDEX idx_registrations_event ON event_registrations(event_id);
CREATE INDEX idx_registrations_student ON event_registrations(student_id);
CREATE INDEX idx_attendance_registration ON event_attendance(registration_id);
```

## ER Diagram Relationships

```
Colleges (1) ──────── (N) Events
Colleges (1) ──────── (N) Students
Events (1) ──────── (N) Event_Registrations
```

```
Students (1) ──────── (N) Event_Registrations
Event_Registrations (1) ──────── (1) Event_Attendance
Event_Registrations (1) ──────── (1) Event_Feedback
Event_Categories (1) ──────── (N) Events
```

## 2. API Design

### Base URL Structure

```
Production: https://api.campusevents.edu/v1
Development: http://localhost:3000/api/v1
```

### Authentication

- **JWT Tokens** with college-specific claims
- **Admin Portal**: College staff login with role-based permissions
- **Student App**: Student email verification + JWT

### Core Endpoints

#### Event Management (Admin)

```
http
```

```
# Create new event
POST /api/v1/events
Authorization: Bearer <admin_token>
Content-Type: application/json

{
  "title": "AI/ML Workshop",
  "description": "Hands-on machine learning workshop",
  "category_id": 1,
  "venue": "Auditorium A",
  "event_date": "2024-03-15",
  "start_time": "10:00",
  "end_time": "16:00",
  "capacity": 150,
  "registration_deadline": "2024-03-10T23:59:59Z"
}

# Get all events for college
GET /api/v1/events?page=1&limit=20&status=active
Authorization: Bearer <admin_token>

# Update event
PUT /api/v1/events/{event_id}
Authorization: Bearer <admin_token>

# Cancel event
PATCH /api/v1/events/{event_id}/cancel
Authorization: Bearer <admin_token>
```

### Student Registration

```
http
```

```
# Browse events (public or student token)
GET /api/v1/events/public?college_id=1&date_from=2024-03-01

# Register for event
POST /api/v1/events/{event_id}/register
Authorization: Bearer <student_token>

# Cancel registration
DELETE /api/v1/events/{event_id}/register
Authorization: Bearer <student_token>

# Get my registrations
GET /api/v1/students/me/registrations
Authorization: Bearer <student_token>
```

### Attendance Management

http

```
# Mark attendance (Admin)
POST /api/v1/events/{event_id}/attendance
Authorization: Bearer <admin_token>
{
  "student_ids": [123, 456, 789],
  "check_in_method": "manual"
}

# Bulk attendance via CSV upload
POST /api/v1/events/{event_id}/attendance/bulk
Authorization: Bearer <admin_token>
Content-Type: multipart/form-data

# Get attendance list
GET /api/v1/events/{event_id}/attendance
Authorization: Bearer <admin_token>
```

### Feedback Collection

http

```
# Submit feedback (Student)
POST /api/v1/events/{event_id}/feedback
Authorization: Bearer <student_token>
{
  "rating": 4,
  "comment": "Great workshop, learned a lot!"
}

# Get event feedback (Admin)
GET /api/v1/events/{event_id}/feedback
Authorization: Bearer <admin_token>
```

### Reporting Endpoints

```http
# Event popularity report
GET /api/v1/reports/events/popularity?semester=2024-spring

# Student participation report
GET /api/v1/reports/students/participation?student_id=123

# Top active students
GET /api/v1/reports/students/top-active?limit=10

# Attendance analytics
GET /api/v1/reports/events/{event_id}/analytics
```

## 3. Workflows & Sequence Diagrams

### Event Registration Flow

```mermaid
```

```
sequenceDiagram
    participant S as Student App
    participant API as REST API
    participant DB as Database
    participant N as Notification Service

    S->>API: GET /events/public
    API->>DB: Query active events
    DB-->>API: Event list
    API-->>S: Event details

    S->>API: POST /events/{id}/register
    API->>DB: Check capacity & duplicates

    alt Capacity Available
        DB-->>API: Registration successful
        API->>N: Send confirmation email
        API-->>S: 201 Created
        N-->>S: Email confirmation
    else Capacity Full
        DB-->>API: Capacity exceeded
        API-->>S: 409 Conflict - Waitlist option
    end
```

### Attendance Marking Flow

```mermaid
sequenceDiagram
    participant A as Admin Portal
    participant API as REST API
    participant DB as Database
    participant S as Student App

    A->>API: GET /events/{id}/registrations
    API->>DB: Fetch registered students
    DB-->>API: Registration list
    API-->>A: Display student list

    A->>API: POST /events/{id}/attendance
    API->>DB: Mark attendance + timestamp
    DB-->>API: Attendance recorded
    API-->>A: Success confirmation

    API->>S: Push notification (attendance marked)
```

### Reporting Generation Flow

```mermaid
sequenceDiagram
    participant A as Admin
    participant API as REST API
    participant Cache as Redis Cache
    participant DB as Database

    A->>API: GET /reports/events/popularity
    API->>Cache: Check cached report

    alt Cache Hit
        Cache-->>API: Cached data
    else Cache Miss
        API->>DB: Complex analytics query
        DB-->>API: Raw data
        API->>API: Process & format
        API->>Cache: Store result (TTL: 1 hour)
    end

    API-->>A: JSON report + CSV export option
```

## 4. Assumptions & Edge Cases

### Key Assumptions

1. **Internet Connectivity**: Students have internet during registration, limited during events
2. **Email Verification**: All students have valid college email addresses
3. **Event Timing**: Events don't span multiple days (single-day focus)
4. **Capacity Management**: Overbooking allowed up to 10% for no-shows
5. **Feedback Timeline**: Feedback collection window is 7 days post-event

### Critical Edge Cases

#### Duplicate Registration Handling

```sql
```

```
-- Database constraint prevents duplicates
ALTER TABLE event_registrations
ADD CONSTRAINT unique_event_student
UNIQUE (event_id, student_id);

-- API response for duplicate attempt
{
  "error": "DUPLICATE_REGISTRATION",
  "message": "You are already registered for this event",
  "registration_date": "2024-03-01T10:30:00Z"
}
```

### Event Capacity Management

```javascript
// Real-time capacity checking with buffer
const checkCapacity = async (eventId) => {
  const event = await Event.findById(eventId);
  const registrationCount = await Registration.count({ event_id: eventId });

  if (registrationCount >= event.capacity) {
    return { available: false, waitlist: true };
  }
  return { available: true, remaining: event.capacity - registrationCount };
};
```

### Missing Feedback Handling

```sql
-- Report query handling missing feedback
SELECT
  e.title,
  COUNT(er.id) as registrations,
  COUNT(ea.id) as attendance,
  COUNT(ef.id) as feedback_count,
  ROUND(AVG(ef.rating), 2) as avg_rating,
  ROUND((COUNT(ef.id)::float / COUNT(ea.id)) * 100, 2) as feedback_rate
FROM events e
LEFT JOIN event_registrations er ON e.id = er.event_id
LEFT JOIN event_attendance ea ON er.id = ea.registration_id
LEFT JOIN event_feedback ef ON er.id = ef.registration_id
WHERE e.college_id = $1 AND e.status = 'completed'
GROUP BY e.id, e.title;
```

#### Event Cancellation Impact

```javascript
// Cascade effects of event cancellation
const cancelEvent = async (eventId) => {
  await db.transaction(async (trx) => {
    // Update event status
    await trx('events').where('id', eventId).update({ status: 'cancelled' });

    // Update all registrations
    await trx('event_registrations')
      .where('event_id', eventId)
      .update({ status: 'cancelled' });

    // Send notifications to registered students
    const registrations = await trx('event_registrations')
      .join('students', 'students.id', 'event_registrations.student_id')
      .where('event_id', eventId)
      .select('students.email', 'students.full_name');

    // Queue notification emails
    await notificationService.sendCancellationEmails(registrations);
  });
};
```

## 5. Performance Considerations

### Database Optimization

- **Partitioning**: Partition large tables by college_id
- **Indexing**: Composite indexes on frequently queried columns
- **Archiving**: Move completed events to archive tables quarterly

### Caching Strategy

- **Event Lists**: Cache for 5 minutes, invalidate on updates
- **Registration Counts**: Real-time updates with Redis pub/sub
- **Reports**: Cache complex reports for 1 hour

### Mobile Optimization

- **Offline-First**: Cache event list for offline browsing
- **Progressive Loading**: Load events in batches of 20
- **Image Optimization**: Compress and lazy-load event images

This design balances simplicity with scalability, ensuring robust handling of edge cases while maintaining good performance characteristics.