## Database Schema - Entity Relationship Diagram

```
<lov-mermaid>
erDiagram
  COLLEGES {
    int id PK
    string name
    string domain
    string address
    string contact_email
    datetime created_at
    json settings
  }

  USERS {
    int id PK
    int college_id FK
    string email UK
    string password_hash
    enum role
    string first_name
    string last_name
    string student_id
    string department
    int year_of_study
    datetime created_at
    datetime last_login
  }

  EVENTS {
    int id PK
    int college_id FK
    int created_by FK
    string title
    text description
    enum event_type
    date start_date
    date end_date
    time start_time
    time end_time
    string venue
    int capacity
    datetime registration_deadline
    enum status
    json requirements
    datetime created_at
    datetime updated_at
}

REGISTRATIONS {
    int id PK
    int user_id FK
    int event_id FK
    enum registration_status
    datetime registered_at
    int waitlist_position
    json additional_info
    boolean notification_sent
}

ATTENDANCE {
    int id PK
```

```
  int registration_id FK
  datetime check_in_time
  datetime check_out_time
  enum attendance_status
  enum check_in_method
  date session_date
  decimal location_lat
  decimal location_lng
  int verified_by FK
}

FEEDBACK {
  int id PK
  int registration_id FK
  int overall_rating
  int content_rating
  int organization_rating
  boolean would_recommend
  text comments
  text suggestions
  datetime submitted_at
  boolean is_anonymous
}

EVENT_ANALYTICS {
  int id PK
  int event_id FK
  int total_registrations
  int total_attendance
  decimal attendance_rate
  decimal average_rating
```

```
        datetime peak_registration_time
        datetime calculated_at
        json metrics
    }

    COLLEGES ||--o{ USERS : "belongs_to"
    COLLEGES ||--o{ EVENTS : "hosts"
    USERS ||--o{ EVENTS : "creates"
    USERS ||--o{ REGISTRATIONS : "makes"
    EVENTS ||--o{ REGISTRATIONS : "has"
    REGISTRATIONS ||--o| ATTENDANCE : "tracks"
    REGISTRATIONS ||--o| FEEDBACK : "provides"
    EVENTS ||--o| EVENT_ANALYTICS : "analyzed_in"
    USERS ||--o{ ATTENDANCE : "verified_by"
</lov-mermaid>
```

## API Architecture Overview

```
<lov-mermaid>
graph TB
    Client[Frontend Client] --> Auth[Authentication Layer]
    Auth --> Router[API Router]

    Router --> EventAPI[Event Management API]
    Router --> RegAPI[Registration API]
    Router --> AttendAPI[Attendance API]
    Router --> FeedAPI[Feedback API]
    Router --> ReportAPI[Reporting API]

    EventAPI --> EventDB[(Events Table)]
    RegAPI --> RegDB[(Registrations Table)]
    AttendAPI --> AttendDB[(Attendance Table)]
    FeedAPI --> FeedDB[(Feedback Table)]
    ReportAPI --> Analytics[(Analytics Engine)]

    EventDB --> MainDB[(PostgreSQL Database)]
    RegDB --> MainDB
    AttendDB --> MainDB
    FeedDB --> MainDB

    Analytics --> MainDB
    Analytics --> Cache[(Redis Cache)]

    Router --> Email[Email Service]
    Router --> Storage[File Storage]
</lov-mermaid>
```

## Student Registration Workflow

```
<lov-mermaid>
sequenceDiagram
    participant S as Student
    participant UI as Frontend
    participant API as Backend API
    participant DB as Database
    participant Email as Email Service
    participant Cache as Redis Cache

    S->>UI: Browse events page
    UI->>API: GET /api/events?college_id=1
    API->>Cache: Check cached events
```

```
alt Cache Miss
  API->>DB: SELECT events WHERE college_id=1 AND status='active'
  DB-->>API: Events list
  API->>Cache: Cache events (5 min TTL)
else Cache Hit
  Cache-->>API: Cached events
end

API-->>UI: Events data
UI-->>S: Display available events

S->>UI: Click "Register" for Event X
UI->>API: POST /api/events/X/register

API->>DB: BEGIN TRANSACTION
API->>DB: SELECT capacity, registered_count FROM events WHERE id=X
DB-->>API: Event capacity info

alt Capacity Available
  API->>DB: CHECK (user_id, event_id) not in registrations
  alt Not Already Registered
    API->>DB: INSERT INTO registrations
    API->>DB: UPDATE event registered_count
    API->>DB: COMMIT TRANSACTION
    DB-->>API: Registration successful

    API->>Email: Send confirmation email
    API->>Cache: Invalidate events cache
    API-->>UI: {success: true, message: "Registered successfully"}
    UI-->>S: Success notification
  else Already Registered
```

```
            API->>DB: ROLLBACK TRANSACTION
            API-->>UI: {success: false, error: "Already registered"}
            UI-->>S: Error message
        end
    else At Capacity
        API->>DB: INSERT INTO registrations (status='waitlisted')
        API->>DB: COMMIT TRANSACTION
        API-->>UI: {success: true, message: "Added to waitlist"}
        UI-->>S: Waitlist notification
    end
</lov-mermaid>
```

## Event Check-in Workflow

```
<lov-mermaid>
sequenceDiagram
    participant S as Student
    participant App as Mobile App
    participant QR as QR Scanner
    participant API as Backend API
    participant DB as Database
    participant Admin as Admin Dashboard

    Note over S,Admin: Event Day Check-in Process

    S->>App: Open event check-in
    App->>QR: Activate QR scanner
    S->>QR: Scan event QR code
    QR-->>App: QR data (event_id, validation_token)

    App->>API: POST /api/attendance/checkin
```

```
Note right of API: Headers: Authorization, Location
Note right of API: Body: {event_id, qr_token, location}

API->>API: Validate JWT token
API->>API: Verify QR token signature

API->>DB: SELECT registration_id FROM registrations WHERE user_id=? AND event_id=?

alt Valid Registration
    DB-->>API: Registration found
    API->>DB: SELECT * FROM attendance WHERE registration_id=? AND session_date=today

    alt First Check-in Today
        API->>DB: INSERT INTO attendance (registration_id, check_in_time, status='present')
        DB-->>API: Attendance recorded

        API->>Admin: WebSocket update (real-time attendance count)
        API-->>App: {success: true, message: "Checked in successfully", event_info}
        App-->>S: Welcome message + event details

    else Already Checked In
        DB-->>API: Existing attendance record
        API-->>App: {success: false, message: "Already checked in at [time]"}
        App-->>S: "Already present" notification
    end

else No Registration
    DB-->>API: No registration found
    API-->>App: {success: false, error: "Not registered for this event"}
    App-->>S: Registration required message
    App->>App: Show quick registration option
    end
</lov-mermaid>
```

## Reporting Data Flow

```
<lov-mermaid>
sequenceDiagram
    participant A as Admin
    participant UI as Admin Dashboard
    participant API as Reporting API
    participant DB as Database
    participant Analytics as Analytics Engine
    participant Cache as Redis Cache

    A->>UI: Request "Event Popularity Report"
    UI->>API: GET /api/reports/event-popularity?college_id=1&period=30days

    API->>Cache: Check report cache key: "popularity_report_1_30days"

    alt Cache Hit (< 1 hour old)
        Cache-->>API: Cached report data
        API-->>UI: Report JSON
    else Cache Miss or Expired
        API->>Analytics: Generate popularity report

        Analytics->>DB: Complex aggregation query
        Note right of DB: SELECT e.title, e.event_type,<br/>COUNT(r.id) as registrations,<br/>COUNT(a.id) as attendance<br/>FROM events e<br/>LEFT JOIN registrations r ON e.id = r.event_id<br/>LEFT JOIN attendance a ON r.id = a.registration_id<br/>WHERE e.college_id = 1<br/>AND e.created_at >= (NOW() - INTERVAL '30 days')<br/>GROUP BY e.id<br/>ORDER BY registrations DESC
        DB-->>Analytics: Raw aggregated data
        Analytics->>Analytics: Process data (calculate percentages, trends)
        Analytics-->>API: Processed report data

        API->>Cache: Cache report (1 hour TTL)
        API-->>UI: Report JSON
    end

    UI->>UI: Render interactive charts
    UI-->>A: Display popularity dashboard

    A->>UI: Click "Export to PDF"
    UI->>API: GET /api/reports/event-popularity/export?format=pdf
    API->>Analytics: Generate PDF report
    Analytics-->>API: PDF file buffer
    API-->>UI: PDF download response
    UI-->>A: Download PDF file
</lov-mermaid>
```

## System Architecture Overview

```
<lov-mermaid>
graph TB
    subgraph "Frontend Layer"
        Web[Web Dashboard]
        Mobile[Mobile App]
    end

    subgraph "API Gateway"
        Gateway[Load Balancer/API Gateway]
        Auth[Authentication Service]
        RateLimit[Rate Limiting]
    end

    subgraph "Application Layer"
        EventService[Event Management Service]
        RegService[Registration Service]
        AttendService[Attendance Service]
        NotifyService[Notification Service]
        ReportService[Reporting Service]
    end

    subgraph "Data Layer"
        MainDB[(Primary Database<br/>PostgreSQL)]
        Cache[(Redis Cache)]
        Queue[(Message Queue<br/>Redis/RabbitMQ)]
        FileStorage[(File Storage<br/>AWS S3/CloudFlare)]
    end

    subgraph "External Services"
        EmailService[Email Service<br/>SendGrid/SES]
        SMSService[SMS Service<br/>Twilio]
        Analytics[Analytics Service<br/>Google Analytics]
    end

    Web --> Gateway
    Mobile --> Gateway
    Gateway --> Auth
    Gateway --> RateLimit

    RateLimit --> EventService
    RateLimit --> RegService
```

```
    RateLimit --> AttendService
    RateLimit --> ReportService

    EventService --> MainDB
    RegService --> MainDB
    AttendService --> MainDB
    ReportService --> MainDB

    EventService --> Cache
    RegService --> Cache
    ReportService --> Cache

    NotifyService --> Queue
    NotifyService --> EmailService
    NotifyService --> SMSService

    EventService --> FileStorage
    ReportService --> Analytics
</lov-mermaid>
```

## Data Flow for Event Creation

```
<lov-mermaid>
flowchart TD
    Start([Admin Creates Event]) --> Validate{Validate Input}

    Validate -->|Invalid| Error[Return Validation Error]
    Validate -->|Valid| CheckAuth{Check Authorization}

    CheckAuth -->|Unauthorized| AuthError[Return 401 Unauthorized]
    CheckAuth -->|Authorized| CreateEvent[Insert Event Record]
    CreateEvent --> GenerateQR[Generate QR Code]
    GenerateQR --> StoreFiles[Store Event Images/Files]
    StoreFiles --> CacheInvalidate[Invalidate Related Caches]
    CacheInvalidate --> SendNotifications[Queue Notification to Interested Students]
    SendNotifications --> LogActivity[Log Admin Activity]
    LogActivity --> Success[Return Event Created Response]

    Error --> End([End])
    AuthError --> End
    Success --> End
</lov-mermaid>
```

## Error Handling and Edge Cases Flow

```
<lov-mermaid>
flowchart TD
    Request[Incoming API Request] --> RateCheck{Rate Limit Check}

    RateCheck -->|Exceeded| RateError[429 Too Many Requests]
    RateCheck -->|OK| AuthCheck{Authentication Check}

    AuthCheck -->|Invalid| AuthError[401 Unauthorized]
    AuthCheck -->|Valid| Validation{Input Validation}

    Validation -->|Invalid| ValidationError[400 Bad Request]
    Validation -->|Valid| BusinessLogic{Business Logic Check}

    BusinessLogic -->|Event Full| CapacityError[409 Conflict - Event Full]
    BusinessLogic -->|Duplicate Registration| DuplicateError[409 Conflict - Already Registered]
    BusinessLogic -->|Event Cancelled| CancelledError[410 Gone - Event Cancelled]
    BusinessLogic -->|Registration Deadline Passed| DeadlineError[410 Gone - Registration Closed]
    BusinessLogic -->|Valid| ProcessRequest[Process Request]

    ProcessRequest --> DBTransaction{Database Transaction}
    DBTransaction -->|DB Error| DBError[500 Internal Server Error]
    DBTransaction -->|Success| LogSuccess[Log Successful Operation]

    LogSuccess --> CacheUpdate[Update Cache]
    CacheUpdate --> SendResponse[Send Success Response]

    RateError --> LogError[Log Error]
    AuthError --> LogError
    ValidationError --> LogError
    CapacityError --> LogError
    DuplicateError --> LogError
    CancelledError --> LogError
    DeadlineError --> LogError
    DBError --> LogError

    LogError --> ErrorResponse[Send Error Response]
    ErrorResponse --> End([End])
    SendResponse --> End
</lov-mermaid>
```

## Key Design Decisions & Rationale

### Database Design Decisions

1. **PostgreSQL Choice**: ACID compliance, excellent JSON support, robust indexing
2. **Normalization**: 3NF to minimize redundancy while maintaining query performance
3. **Soft Deletes**: Use status fields instead of DELETE operations for audit trails
4. **UUID vs Integer IDs**: Integer for performance, UUID for public-facing identifiers

### API Design Principles

1. **RESTful Design**: Standard HTTP methods and status codes

2. **Consistent Response Format**: Unified JSON structure across all endpoints

3. **Pagination**: Cursor-based pagination for large result sets

4. **Versioning**: URL versioning (/api/v1/) for backward compatibility

### Caching Strategy

1. **Event Lists**: 5-minute TTL, invalidated on event changes

2. **User Registrations**: 1-minute TTL, invalidated on registration changes

3. **Reports**: 1-hour TTL, regenerated on demand

4. **Static Content**: CDN caching with long TTL

### Security Considerations

1. **Authentication**: JWT with refresh tokens

2. **Authorization**: Role-based access control (RBAC)

3. **Input Validation**: Server-side validation for all inputs

4. **Rate Limiting**: Per-user and per-IP rate limits

5. **SQL Injection Prevention**: Parameterized queries only

### Scalability Considerations

1. **Horizontal Scaling**: Stateless application servers

2. **Database Sharding**: By college_id for multi-tenant isolation

3. **Read Replicas**: For reporting and analytics queries

4. **Message Queues**: Asynchronous processing for notifications

5. **CDN**: Global content delivery for static assets