Campus Event Management Platform - Approach Document

1. Problem Analysis & Assumptions

Core Problem

Design and implement a basic event reporting system for Dayananda Sagar University that connects:

- **Admin Portal (Web)**: Staff create and manage events
- **Student App (Mobile)**: Students browse, register, and check-in to events

Key Assumptions

1. **Scale**: ~50 colleges, ~500 students each, ~20 events per semester
2. **Event Types**: Hackathons, workshops, tech talks, fests, cultural events
3. **Multi-tenant**: Each college operates independently but uses same platform
4. **Data Privacy**: College data should be isolated
5. **Offline Capability**: Basic check-in should work with poor connectivity
6. **Academic Calendar**: Events are semester-based

Decision: Multi-College Architecture

Chosen Approach: Single database with college_id partitioning

- **Pros**: Easier maintenance, shared infrastructure, cross-college analytics possible
- **Cons**: Potential security concerns, requires careful access control
- **Alternative**: Separate databases per college (more complex deployment)

2. Solution Approach

Architecture Overview

```
┌─────────────────┐    ┌─────────────────┐
│  Admin Portal   │    │   Student App   │
│    (Web)        │    │   (Mobile)      │
└────────┬────────┘    └────────┬────────┘
         │                      │
         └──────────┬───────────┘
                    │
            ┌───────▼───────┐
            │   REST API    │
            │  (Node.js)    │
            └───────┬───────┘
                    │
```

```
┌─────────▼─────────┐
│ Database │
│ (PostgreSQL) │
└───────────────────┘
```

Data Flow Strategy

1. **Event Creation** → Admin creates → Students browse
2. **Registration** → Student registers → Confirmation sent
3. **Check-in** → QR/Manual → Attendance marked
4. **Feedback** → Post-event → Analytics generated
5. **Reporting** → Real-time dashboards → Export capabilities

3. Technical Decisions

Database Choice: PostgreSQL

Reasoning:

- Better JSON support for flexible event metadata
- Strong ACID properties for registration conflicts
- Excellent reporting/analytics capabilities
- Handles concurrent registrations well

Alternative Considered: SQLite (too limited for multi-college scale)

API Design Philosophy

- **RESTful**: Standard HTTP methods and status codes
- **Versioned**: `/api/v1/` prefix for future compatibility
- **Authenticated**: JWT tokens with college-specific permissions
- **Paginated**: All list endpoints support pagination

Event ID Strategy

Decision: Composite IDs (college_prefix + sequential)

- Format: `DSU_EVT_001`, `MIT_EVT_001`
- **Pros**: Human-readable, college-identifiable, sortable
- **Cons**: Slightly more complex than UUIDs
- Ensures uniqueness across colleges while maintaining readability

4. Edge Cases & Handling

Registration Edge Cases

1. **Duplicate Registrations**: Database unique constraint + API validation
2. **Event Capacity**: Real-time capacity checking with buffer
3. **Late Cancellations**: Grace period + waitlist management
4. **Network Issues**: Offline-first mobile design with sync

Attendance Edge Cases

1. **Multiple Check-ins**: Only first valid check-in counts
2. **Proxy Attendance**: QR codes expire after 5 minutes
3. **Event Changes**: Real-time notifications to registered students
4. **Missing Feedback**: Optional but encouraged, default neutral rating

Reporting Edge Cases

1. **Cancelled Events**: Excluded from popularity metrics
2. **Partial Attendance**: Distinguish registered vs attended
3. **Cross-Semester**: Date-based filtering in all reports
4. **Data Consistency**: Nightly reconciliation jobs

5. Scalability Considerations

Database Optimization

- Indexed columns: college_id, event_date, student_id
- Partitioning by college_id for large datasets
- Read replicas for reporting workloads

Caching Strategy

- Redis for session management
- Event list caching (invalidate on updates)
- Registration counts cached with TTL

Mobile Optimization

- Offline-first registration sync
- Image compression for event photos
- Progressive loading for event lists

6. Security Considerations

Authentication & Authorization

- College-specific admin accounts
- Student authentication via college email
- Role-based permissions (admin, staff, student)
- API rate limiting per college

Data Protection

- College data isolation
- Personal data encryption at rest
- Audit logs for admin actions
- GDPR-compliant data retention

7. Implementation Priority

Phase 1 (MVP - Current Focus)

- Basic CRUD for events
- Student registration system
- Simple attendance marking
- Core reporting (registrations, attendance %)

Phase 2 (Enhancement)

- Mobile app UI/UX
- QR code generation
- Real-time notifications
- Advanced analytics

Phase 3 (Scale)

- Multi-college deployment
- Advanced reporting dashboard
- Integration APIs
- Performance optimization

8. Success Metrics

Technical Metrics

- API response time < 200ms
- 99.9% uptime during events
- Zero data loss incidents
- Mobile app crash rate < 1%

Business Metrics

- Event registration rate > 70%
- Attendance rate > 80%
- Student app adoption > 60%
- Admin satisfaction score > 4.0/5.0

9. Next Steps

1. **Database Schema Design** → Define tables and relationships
2. **API Specification** → Document all endpoints
3. **Prototype Implementation** → Build core functionality
4. **UI Wireframes** → Design user interfaces
5. **Testing Strategy** → Unit and integration tests

This approach balances simplicity with scalability, ensuring we can build a working prototype while keeping future enhancements feasible.