Prototype Implementation :

Frontend Prototype

Admin Portal Features

- **Dashboard**: Event statistics and overview
- **Event Management**: Create, view, and manage events
- **Reports**: Comprehensive analytics and reporting
- **Student Management**: View student participation data

Student Portal Features

- **Event Browser**: Search and filter events
- **Registration**: One-click event registration
- **My Events**: Personal event tracking
- **Profile Management**: Account settings

Components Implemented

- `EventCard`: Reusable event display component
- `Navigation`: Role-based navigation system
- `AdminDashboard`: Statistics and quick actions
- `EventBrowser`: Event listing with filters
- `ReportsPage`: Analytics and reporting interface
- `CreateEventForm`: Event creation form

Backend Implementation Requirements

For full functionality, the system requires backend implementation with:

Database Requirements

```sql
-- Core tables needed
CREATE TABLE colleges (
  id UUID PRIMARY KEY,
  name VARCHAR NOT NULL,
  location VARCHAR,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE events (
  id UUID PRIMARY KEY,
  title VARCHAR NOT NULL,
  description TEXT,
  date DATE NOT NULL,
  time VARCHAR NOT NULL,
  location VARCHAR NOT NULL,
  type VARCHAR NOT NULL,
  capacity INTEGER NOT NULL,
  college_id UUID REFERENCES colleges(id),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  created_by UUID NOT NULL
);

CREATE TABLE students (
  id UUID PRIMARY KEY,
  name VARCHAR NOT NULL,
  email VARCHAR UNIQUE NOT NULL,
  college_id UUID REFERENCES colleges(id),
  student_id VARCHAR NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE event_registrations (
  id UUID PRIMARY KEY,
  event_id UUID REFERENCES events(id),
  student_id UUID REFERENCES students(id),
  registered_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  attended BOOLEAN DEFAULT FALSE,
  check_in_time TIMESTAMP,
  UNIQUE(event_id, student_id)
);

CREATE TABLE event_feedback (
  id UUID PRIMARY KEY,
  event_id UUID REFERENCES events(id),
  student_id UUID REFERENCES students(id),
  rating INTEGER CHECK (rating >= 1 AND rating <= 5),
  comment TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(event_id, student_id)
);
```

API Endpoints Required

```javascript
// Event Management
POST /api/events - Create new event
GET /api/events - List events with filters
PUT /api/events/:id - Update event
DELETE /api/events/:id - Delete event

// Registration Management
POST /api/events/:id/register - Register for event
DELETE /api/events/:id/register - Cancel registration
GET /api/students/:id/events - Get student's events
```

```
// Attendance Tracking
POST /api/events/:id/checkin - Mark attendance
GET /api/events/:id/attendance - Get attendance list

// Feedback System
POST /api/events/:id/feedback - Submit feedback
GET /api/events/:id/feedback - Get event feedback

// Analytics & Reports
GET /api/analytics/events/popularity - Event popularity report
GET /api/analytics/students/participation - Student participation
GET /api/analytics/students/top-active - Most active students
GET /api/analytics/events/by-type - Event type distribution
```

Sample Report Queries

Event Popularity Report
```sql
SELECT
  e.title as event_name,
  e.type,
  COUNT(er.id) as registrations,
  (COUNT(er.id)::FLOAT / e.capacity * 100) as fill_percentage
FROM events e
LEFT JOIN event_registrations er ON e.id = er.event_id
GROUP BY e.id, e.title, e.type, e.capacity
ORDER BY registrations DESC;
```

Student Participation Report

```sql
SELECT
  s.name as student_name,
  COUNT(er.id) as total_registrations,
  COUNT(CASE WHEN er.attended = true THEN 1 END) as events_attended,
  ROUND(
    COUNT(CASE WHEN er.attended = true THEN 1 END)::FLOAT /
    COUNT(er.id) * 100, 2
  ) as attendance_percentage
FROM students s
LEFT JOIN event_registrations er ON s.id = er.student_id
GROUP BY s.id, s.name
HAVING COUNT(er.id) > 0
ORDER BY events_attended DESC;
```

Top 3 Most Active Students

```sql
SELECT
  s.name,
  COUNT(CASE WHEN er.attended = true THEN 1 END) as events_attended,
  CASE
    WHEN COUNT(CASE WHEN er.attended = true THEN 1 END) >= 10 THEN 'Super Learner'
    WHEN COUNT(CASE WHEN er.attended = true THEN 1 END) >= 7 THEN 'Event Enthusiast'
    WHEN COUNT(CASE WHEN er.attended = true THEN 1 END) >= 5 THEN 'Active Participant'
    ELSE 'Beginner'
  END as badge
FROM students s
JOIN event_registrations er ON s.id = er.student_id
WHERE er.attended = true
```

```
GROUP BY s.id, s.name
ORDER BY events_attended DESC
LIMIT 3;
```

Mock Data Implementation

The current prototype uses comprehensive mock data to demonstrate:

- 6 sample events across different types
- Realistic registration numbers and ratings
- Sample analytics data for reports
- Student participation data

Setup Instructions

1. **Clone/Download Project**
2. **Install Dependencies**: `npm install`
3. **Run Development Server**: `npm run dev`
4. **Access Application**: Open `http://localhost:5173`

Testing the Prototype

Admin Features

1. Navigate to admin dashboard
2. View event statistics
3. Access reports section
4. Test event creation form

Student Features

1. Browse available events
2. Test registration functionality

3. View different event types

4. Check event details and ratings

Next Steps for Full Implementation

1. **Backend Setup**: Implement database and API endpoints

2. **Authentication**: Add user login/registration system

3. **Real-time Updates**: WebSocket for live capacity updates

4. **Mobile Optimization**: Responsive design improvements

5. **Testing**: Unit and integration tests

6. **Deployment**: Production deployment setup