Software Requirements Specification (SRS)

Project Title: Frolic Event Management System (MERN Stack)

---

1. Introduction

1.1 Purpose

This SRS describes the complete functional and non-functional requirements of the **Frolic Event Management System**, built using the **MERN stack (MongoDB, Express.js, React.js, Node.js)**.

The system manages:

- Institutes
- Departments
- Events
- Groups & Participants
- Event Coordinators & Users
- Winners listing

This SRS will serve as:

- A reference for developers
- A contract with stakeholders
- A blueprint for backend API design and frontend UI/UX development

---

1.2 Scope

The system provides the following major features:

Admin Features

- Manage institutes, departments, and events
- Manage event coordinators
- Approve groups, participants, and payments
- Mark attendance
- Declare winners
- Generate reports

Student / User Features

- Register & log in
- View events and event details
- Create groups
- Add participants
- Pay event fees
- View results

1.3 Definitions

Term	Meaning
MERN	MongoDB, Express.js, React.js, Node.js
Coordinator	User responsible for managing an institute, department, or event
Group	A team participating in an event
Participant	A student or user participating in an event

1.4 References

- Provided table structure document: FrolicDB.docx

  FrolicDB

---

2. Overall Description

2.1 Product Perspective

The system replaces manual event management operations with a fully digital solution.

System Architecture

- **Frontend:** React.js + Redux/Context for state management
- **Backend:** Node.js + Express.js REST APIs
- **Database:** MongoDB
- **Authentication:** JWT-based authentication
- **Deployment:** Docker or standalone deployment

2.2 User Classes

User Type	Description
Admin	Full access, manages events, departments, institutes, and users
Institute Coordinator	Manages assigned institutes
Department Coordinator	Manages assigned departments
Event Coordinator	Manages events, participants, payments
Student / Participant	Joins events, creates groups

2.3 Operating Environment

- Web Browsers: Chrome, Firefox, Safari
- Server: Node.js 18+
- Database: MongoDB 6+
- OS: Linux/Windows

---

3. System Features / Functional Requirements

Below are detailed features aligned with your database structure.

3.1 User Management

Description

Manages all users including admins and coordinators.

Database Reference

Users table

FrolicDB

Functional Requirements

1. User can register with name, email, phone, and password.
2. Passwords must be stored hashed.
3. User can log in and receive JWT token.
4. Admin can assign roles (Admin, Coordinator, etc.).
5. User can update profile.
6. Admin can deactivate/activate users.

3.2 Institute Management

Database Reference

Institutes table

FrolicDB

Functional Requirements

1. Admin can create, update, delete institutes.
2. Admin assigns an Institute Coordinator (UserID foreign key).
3. View institute list and details.
4. Upload institute image.
5. System tracks CreatedAt, ModifiedAt, ModifiedBy.

3.3 Department Management

Database Reference

Departments table

FrolicDB

Functional Requirements

1. Create/update/delete departments within an institute.
2. Assign Department Coordinator (UserID).
3. Attach images and descriptions.
4. Show all departments under each institute.

3.4 Event Management

Database Reference

Events table

FrolicDB

Functional Requirements

1. Admin/Department/Institute coordinator can add events.
2. Event details include:
    - Name, tagline, description
    - Images
    - Fees, prizes
    - Min/Max participants
    - Event location, max groups allowed
3. Assign Event Coordinator (UserID).
4. Student Coordinator details saved (name/phone/email).
5. Update/delete events.
6. Display events publicly.

3.5 Group Management

Database Reference

Groups table

FrolicDB

Functional Requirements

1. Students can create a group for an event.
2. Group fields:
    - GroupName
    - EventID
    - IsPaymentDone
    - IsPresent
3. Admin/coordinator can mark attendance and verify payment.
4. Limit number of groups per event (MaxGroupsAllowed).

3.6 Participant Management

Database Reference

Participants table

FrolicDB

Functional Requirements

1. Students can add group members (participants).
2. Participant fields include name, enrollment number, institute name, city, phone, email.

3. Mark one participant as "Group Leader".
4. System checks:
    - Max participant limit
    - Min participant limit
5. Admin/coordinator can update participant details.
6. Track CreatedAt, ModifiedAt, ModifiedBy.

3.7 Winner Management

Database Reference

EventWiseWinners table

FrolicDB

Functional Requirements

1. Event Coordinators can assign winners.
2. Each winner record stores:
    - EventID
    - GroupID
    - Sequence (1st, 2nd, 3rd)
3. Allow updating or deleting winners.
4. Display event results publicly.

---

4. External Interface Requirements

4.1 User Interface Requirements

Admin Panel

- Dashboard with summary cards
- CRUD screens for institutes, departments, events
- Manage users
- Winner declaration section
- Attendance marking page
- Reports

Student UI

- Browse events
- Event detail pages
- Create group

- Add participants
- Payment page

4.2 API Interface Requirements

REST-style endpoints like:

```
POST /api/auth/login
POST /api/users
GET  /api/institutes
POST /api/events
POST /api/groups
POST /api/participants
POST /api/winners
```

All admin actions require JWT + admin role middleware.

---

5. Database Design Mapping

All tables referenced:
Users, Institutes, Departments, Events, Groups, Participants, EventWiseWinners

FrolicDB

Each table maps to one or more API modules:

Table	API Module
Users	Auth, Users
Institutes	Institutes
Departments	Departments
Events	Events
Groups	Groups
Participants	Participants
EventWiseWinners	Winners

---

6. REST API Design

6.1 Modules

- **Auth & Users**
- **Institutes**

- **Departments**
- **Events**
- **Groups**
- **Participants**
- **EventWiseWinners (Results)**

JWT authentication on all protected routes (Admin / Coordinator / User).

6.2 Endpoint Overview

6.2.1 Auth

Method	Endpoint	Description	Auth
POST	/api/auth/register	User registration	Public
POST	/api/auth/login	Login, returns JWT	Public
GET	/api/auth/me	Current logged-in user	JWT

6.2.2 Users

Method	Endpoint	Description	Auth
GET	/api/users	List users	Admin
GET	/api/users/:id	Get user by ID	Admin/self
PATCH	/api/users/:id	Update user/role	Admin/self
DELETE	/api/users/:id	Deactivate/remove user	Admin

6.2.3 Institutes

Method	Endpoint	Description	Auth
GET	/api/institutes	List institutes	Public
GET	/api/institutes/:id	Institute details	Public
POST	/api/institutes	Create institute	Admin
PATCH	/api/institutes/:id	Update institute	Admin
DELETE	/api/institutes/:id	Delete institute	Admin

6.2.4 Departments

Method	Endpoint	Description	Auth
GET	/api/departments	List all departments	Public
GET	/api/institutes/:id/departments	Departments under an institute	Public
POST	/api/departments	Create department	Admin/InstituteCoord

Method	Endpoint	Description	Auth
PATCH	/api/departments/:id	Update department	Admin/DeptCoord
DELETE	/api/departments/:id	Delete department	Admin

6.2.5 Events

Method	Endpoint	Description	Auth
GET	/api/events	List all events	Public
GET	/api/events/:id	Event details	Public
GET	/api/departments/:id/events	Events under department	Public
POST	/api/events	Create event	Admin/DeptCoord
PATCH	/api/events/:id	Update event	Admin/EventCoord
DELETE	/api/events/:id	Delete/cancel event	Admin

6.2.6 Groups

Method	Endpoint	Description	Auth
GET	/api/events/:id/groups	List groups for event	Coord/Admin
POST	/api/events/:id/groups	Create group for event	User
GET	/api/groups/:id	Get group	Coord/Admin/Owner
PATCH	/api/groups/:id	Update name, payment, presence	Coord/Admin
DELETE	/api/groups/:id	Delete group (pre-event or admin)	Owner/Admin

---

6.2.7 Participants

Method	Endpoint	Description	Auth
GET	/api/groups/:groupId/participants	List participants of group	GroupOwner/Coord
POST	/api/groups/:groupId/participants	Add participant to group	GroupOwner
PATCH	/api/participants/:id	Update participant	Owner/Coord
DELETE	/api/participants/:id	Remove participant	Owner/Coord

6.2.8 EventWiseWinners (Results)

Method	Endpoint	Description	Auth
GET	/api/events/:id/winners	Get winners for event	Public
POST	/api/events/:id/winners	Declare winners (1st/2nd/3rd)	Admin/EventCoord
PATCH	/api/winners/:id	Update winner record	Admin/EventCoord
DELETE	/api/winners/:id	Delete winner record	Admin

12-Week MERN Project Plan

Week 1 – HTML Wireframing & Static Page Structures

Topics

- HTML basics
- Creating page structure
- Layout planning of all pages needed for the project

Student Tasks (Static HTML pages)

Students must create the **base structure** of the following pages:

1. **Login Page**
2. **Admin Dashboard**
3. **Institute List Page**
4. **Department List Page**
5. **Event List Page**

Each page must include:

- Header / Title
- Sections
- Basic HTML layout
- Placeholder content

Goal: Students visualize the full project.

Week 2 – CSS Styling & Component Design

Topics

- External CSS
- Flexbox (for layouts)
- Styling tables, forms, cards

Student Tasks

Add CSS styling to previously created pages:

✓ Style Login page
✓ Style Dashboard layout
✓ Create table layouts for:

- Institutes
- Departments
- Events

✓ Create card layout for Event detail page (static)

Focus:
Looks neat, aligned, readable.

Week 3 – Bootstrap Integration & Full UI Prototype

Topics

- Bootstrap Grid
- Bootstrap Forms
- Bootstrap Navbar & Cards
- Responsive design

Student Tasks

Enhance UI using Bootstrap components:

Students must convert HTML pages into Bootstrap UI:

Pages for Students to Complete by Week 3

Page	Purpose
Login Page	User login UI
Admin Dashboard	Shows counts (dummy numbers)
Institute List Page	Bootstrap table
Department List Page	Bootstrap table
Event List Page	Card/grid layout
Event Detail Page	Event information
Group List Page (static prototype)	Table with dummy group names
Participant List Page	Table
Winner Display Page	Cards with 1st, 2nd, 3rd winners

Students Use Only Dummy Data

Week 4 – MongoDB & Mongoose + Schema Design

Topics

- Intro to MongoDB & Mongoose
- Connecting Express to MongoDB
- Designing Schema from tables (Users, Institutes, Departments, Events, Groups, Participants, Winners)

Student Tasks

- Setup MongoDB (local or Atlas)
- Create Mongoose models:
  - `User`, `Institute`, `Department`, `Event`, `Group`, `Participant`, `EventWiseWinner`
- Test inserting sample records in each collection using a test script or Postman

---

Week 5 – Institutes & Departments Backend APIs

Topics

- RESTful API design
- Relations: Institute → Departments
- Assigning coordinators (UserID as foreign key)

Student Tasks

- Implement Institute APIs:
  - `GET /api/institutes`
  - `GET /api/institutes/:id`
  - `POST /api/institutes` (Admin only)
  - `PUT /api/institutes/:id`
  - `DELETE /api/institutes/:id`
- Implement Department APIs:
  - `GET /api/departments`
  - `GET /api/institutes/:id/departments`
  - `POST /api/departments`
  - `PUT /api/departments/:id`
  - `DELETE /api/departments/:id`
- Check validation: no duplicate names in same institute (optional)

Week 6 – Events Module Backend

Topics

- Event fields & business rules
- Linking Events to Department, Coordinator, Limits (MaxGroups, Min/Max participants)

Student Tasks

- Implement Event APIs:
  - `GET /api/events`
  - `GET /api/events/:id`
  - `GET /api/departments/:id/events`
  - `POST /api/events` (Admin / DepartmentCoordinator)
  - `PUT /api/events/:id`
  - `DELETE /api/events/:id`
- Add validations:
  - `groupMinParticipants <= groupMaxParticipants`
  - `maxGroupsAllowed > 0`

Week 7 – Groups & Participants Backend

Topics

- Group registration flow
- Participant linking with Group
- Group leader concept (`isGroupLeader`)
- Min/Max participants per group

Student Tasks

- Group APIs:
  - `GET /api/events/:eventId/groups`
  - `POST /api/events/:eventId/groups`
  - `PUT /api/groups/:id` (update name, `isPaymentDone`, `isPresent`)
  - `DELETE /api/groups/:id`
- Participant APIs:
  - `GET /api/groups/:groupId/participants`
  - `POST /api/groups/:groupId/participants`
  - `PUT /api/participants/:id`
  - `DELETE /api/participants/:id`
- Add rules in backend:
  - Cannot add more participants than `groupMaxParticipants` of the event
  - Must have at least one `isGroupLeader = true`
  - Max one leader per group

Week 8 – Winners & Reporting

Topics

- Storing winners with rank (1st / 2nd / 3rd)
- Ensuring unique sequence per event
- Reporting APIs using aggregation / populate

Student Tasks

- Winner APIs:
  - `GET /api/events/:eventId/winners`
  - `POST /api/events/:eventId/winners` (Admin / EventCoordinator)
  - `PUT /api/winners/:id`
  - `DELETE /api/winners/:id`
- Validations:
  - No duplicate sequence (cannot have two 1st winners for same event)
- Reporting APIs:
  - `GET /api/events/:id/summary` → total groups, total participants
  - `GET /api/institutes/:id/summary` → events count, participants count (optional)

Week 9 – Backend Enhancements (Realistic & Clean APIs)

Topics

- Pagination & searching
- Sorting & filtering
- Error handling structure
- Security basics (rate limit, basic validation)

Student Tasks

- Add pagination to:
  - `GET /api/events` → `?page=1&limit=10`
  - `GET /api/institutes`
- Add search & filter:
  - Search events by name: `?q=code`
  - Filter events by department or location
- Implement centralized error handler middleware
- Ensure all APIs return consistent JSON format:
  - `{ success, data, message }`

Week 10 – Authentication & Authorization (Users + JWT)

Topics

- User registration & login
- Password hashing (bcrypt)
- JWT tokens (login → token → protected route)
- Role-based fields: admin / coordinator / student

Student Tasks

- Implement:
  - `POST /api/auth/register`
  - `POST /api/auth/login`
- Store passwords as hashed
- Generate JWT token on login
- Create middleware `authMiddleware` to protect routes

Create `GET /api/auth/me` to return logged-in user info

Week 11 – React + Backend Integration (Auth & Listings)

Topics

- Axios setup
- Calling backend from React
- Storing JWT in localStorage
- Protected routes

Student Tasks

- Create basic pages:
  - Login
  - Dashboard (empty shell)
  - Institutes list page (static dummy data)
  - Events list page (static)
- Setup React Router navigation

- Implement real login using `/api/auth/login`
- After login, redirect to dashboard
- Fetch & show:
  - Institutes list from backend
  - Departments list from backend
  - Events list from backend
- Show error message on failed login

Week 12 – React Forms & CRUD for Main Entities

Topics

- Controlled forms in React
- POST/PUT/DELETE from React
- Basic form validation (required fields)
- Nested UI flows (Event → Groups → Participants)
- Conditional rendering

- Role-based UI (show certain features only to admin/coordinator)

Student Tasks
Create UI with full CRUD for:

- Institutes
- Departments
- Events
- For a selected Event:
    - Show all Groups
    - Button to “Add Group”
    - For each group: view Participants
    - Add Participant form
- Winners Page:
    - Page for Event Coordinator to select 1st/2nd/3rd from groups
    - Public page to display winners for any event

On each page:

- List items in table
- Add new using form
- Edit existing
- Delete with confirmation

Users

UserID	Int auto_increment
UserName	Varchar(100)
UserPassword	Varchar(300), hashed password
EmailAddress	Varchar(300)
PhoneNumber	Varchar(50)
IsAdmin	Boolean

Institutes

InstituteID	Int auto_increment
InstituteName	Varchar(100)
InsituteImage	Varchar(300)
InsituteDescription	Varchar(1000)
InsituteCoOrdinatorID	F.K. (Users -> UserID)
CreatedAt	Timestamp
ModifiedAt	Timestamp
ModifiedBy	F.K. (Users -> UserID)

Departments

DepartmentID	Int auto_increment
DepartmentName	Varchar(100)
DepartmentImage	Varchar(300)
DepartmentDescription	Varchar(1000)
InstituteID	F.K. (institutes -> InstitueID)
DepartmentCoOrdinatorID	F.K. (Users -> UserID)
CreatedAt	Timestamp
ModifiedAt	Timestamp
ModifiedBy	F.K. (Users -> UserID)

events

EventID	Int auto_increment
EventName	Varchar(100)
EventTagline	Varchar(300)
EventImage	Varchar(300)
EventDescription	Varchar(1000)
GroupMinParticipants	Int
GroupMaxParticipants	Int
EventFees	Int
EventFirstPrice	Varchar(300)
EventSecondPrice	Varchar(300)
EventThirdPrice	Varchar(300)
DepartmentID	F.K. (department -> DepartmentID)
EventCoOrdinatorID	F.K. (Users -> UserID)
EventMainStudentCoOrdinatorName	Varchar(100)

EventMainStudentCoOrdinatorPhone	Varchar(100)
EventMainStudentCoOrdinatorEmail	Varchar(300)
EventLocation	Varchar(100)
MaxGroupsAllowed	int
CreatedAt	Timestamp
ModifiedAt	Timestamp
ModifiedBy	F.K. (Users -> UserID)

groups

GroupID	Int auto_increment
GroupName	Varchar(100)
EventID	F.K. (Events -> EventID)
IsPaymentDone	boolean
IsPresent	boolean
CreatedAt	Timestamp
ModifiedAt	Timestamp
ModifiedBy	F.K. (Users -> UserID)

Participants

ParticipantID	Int auto_increment
ParticipantName	Varchar(100)
ParticipantEnrollmentNumber	Varchar(100)
ParticipantInsituteName	Varchar(300)
ParticipantCIty	Varchar(300)
ParticipantMobile	Varchar(100)
ParticipantEmail	Varchar(300)
IsGroupLeader	Boolean
GroupID	F.K. (Groups -> GroupID)
CreatedAt	Timestamp
ModifiedAt	Timestamp
ModifiedBy	F.K. (Users -> UserID)

EventWiseWinners

EventWiseWinnerID	Int auto_increment
EventID	F.K. (Events -> EventID)
GroupID	F.K. (Groups -> GroupID)
Sequence	Int
CreatedAt	Timestamp
ModifiedAt	Timestamp
ModifiedBy	F.K. (Users -> UserID)