High Level Design Document

Student Event Management

Front End Design

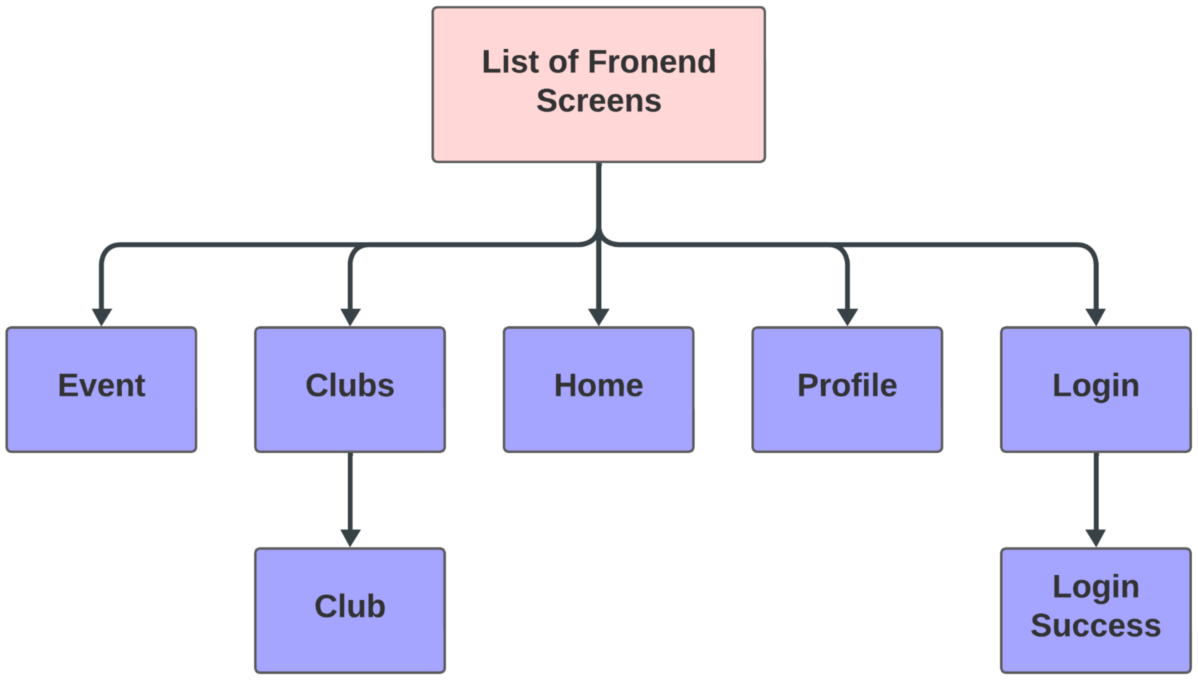
Overall Technology to be used:

- HTML (Hypertext Markup Language)
- CSS (Cascading Style Sheets)
- JavaScript
- React JS

Component library being used, including any state management:

- ReduxJS: A predictable state container for managing application state.
- TailwindCSS: a utility-first CSS framework for rapidly building modern websites.

List of screens in Front End:



The front end screens/modules are summarized in the table below:

Name	**Main Components**	**APIs Required**
Event	**Cards:** Displaying event information. **Navigation Menu:** For easy access to Home, Clubs and Profile pages. **Forms:** To register for events.	AuthenticateUser, Get Event Details, RegisterForEvent
Home	**Cards:** List of Upcoming Events. **Navigation Menu** **Pages:** Event details page.	GetAllEvents
Clubs (includes club single info page)	**Cards:** Displaying club information. **Navigation Menu** **Forms:** To register new clubs, edit an existing club's details, propose a new event or edit an existing event's details **Pages:** Club details page.	AuthenticateUser, GetAllClubs, CreateClubRequest, UpdateClubDetails, getAboutInfo, CreateEventRequest
Profile **(Includes Manage)**	**Cards:** Displaying user profile information, List of registered events, List of event proposal status, List of club proposal status, List of pending event proposals with approve and reject action(for admin only), List of pending club proposals with approve and reject action(for admin only). **Navigation Menu:** For easy access to Home, Clubs and Profile pages. **Forms:** Approve/Reject Event and Club Proposals, Change Admin(for admin only)	UserProfile, ApproveEventRequest, RejectEventRequest, ApproveClubRequest, RejectClubRequest
Login (includes login success)	**Forms:** GoogleAuthLogin **Pages:** Login Success Page(for redirecting).	OAuthCallback,

1. Events
    a. Purpose: The "Events" screen serves as the central hub for event management within the application. Its primary purpose is to empower users to efficiently manage their events. Students can browse upcoming events, register for upcoming events. Club coordinators can propose a new event and update event details.
    b. Main components:
        i. Cards: Displaying event information.
        ii. Navigation Menu: For easy access to events-related actions.
        iii. Forms: To register for events.
        iv. Pages: Events details page.
    c. Functions:

        i. Manage Events: Accessible through the navigation menu, allowing users to perform actions like registering for events or updating event details.
        ii. View Event Details: Navigating to the event details page to see comprehensive information about a specific event.

2. Clubs
    a. Purpose: The "Clubs" screen facilitates club management within the application. Its primary purpose is to empower users to efficiently manage their clubs. Students can browse existing clubs, propose a new club. Club coordinators can propose a new event and edit club details.
    b. Main components:
        i. Cards: Displaying club information.
        ii. Navigation Menu: For easy access to clubs-related actions.
        iii. Forms: To register a new club.
        iv. Pages: Clubs details page.
    c. Functions:
        i. Manage Clubs: Accessible through the navigation menu, allowing Club Coordinators to perform actions like updating club details and proposing a new event.
        ii. Create New Club: Using the form component to input club details and request a new club.
        iii. View Club Details: Navigating to the club details page to see comprehensive information about a specific club.
3. Profile
    a. Purpose: The "Profile" screen facilitates users to efficiently manage their personal details. Only Admin personnel or Student Coordinator can update the student information.
    b. Main components:
        i. Cards: Displaying user information.
        ii. Navigation Menu: For easy access to user-related actions.
    c. Functions:
        i. Manage User Information: Only Admin personnel or Student Coordinator can update the student information.
        ii. View User Details: Navigating to the user details page to see comprehensive information about your profile.

Back End Design

Backend Frameworks:

- The platform will be developed using JavaScript with Node.js for server-side
- scripting
- Express.js as the web application framework.

Components / Libraries:

- **jsonwebtoken**: package to deal with JSON web tokens
- **mongoose**: provides a straight-forward, schema-based ODM (object data model) to model your application data
- **morgan**: HTTP request logger middleware for node.js
- **passport / passport-jwt**: authentication middleware using JSON web tokens
- **passport-google-oauth20**: Google authentication library for integration with passport
- **supertest**: SuperAgent driven library for testing HTTP servers
- **typescript**: package to add typescript support to the project

Summary of Back-end APIs:

The table gives a summary of the APIs - details are given later for each of the APIs.
Name (and description), inputs, response

API Name (Description)	**Inputs**	**Response**
OAuthCallback	User credentials	Valid / Not Valid, Access Token
GetAllClubs	None	Returns a list of all clubs
CreateClubRequest	Access Token, Club data	Submit message
GetAboutInfo	Access Token, Club ID	Club data
ApproveClubRequest	Access Token, Club Proposal ID	Success message
RejectClubRequest	Access Token, Club Proposal ID	Decline message
UpdateClubDetails	Access Token, Club ID, Club data	Success message
GetAllEvents	None	Array of all upcoming events
EventInfo	Access Token, Event ID	Event data
CreateEventRequest	Access Token, Event data	Submit Message
ApproveEventRequest	Access Token, Event Proposal ID	Success message
RejectEventRequest	Access Token, Event Proposal ID	Decline message
RegisterForEvent	Access Token, Event ID	Success message / Decline message - if registration not approved
UpdateAdmin (to be implemented later)	Access Token, User ID	Success message / Decline message - if admin not updated

Further Details

Details of APIs

1. **Authentication Services**
    a. **Model Layer**: AuthModel
    b. **Functions**:
        i. oauthCallback(token): Verify the Google token and retrieve the user's email and details. If the user exists, return the user's access token; if not, create a user entry in the database and return a new access token.
    c. **Description**: Handles user authentication via Google and manages new user sign-up using institute's domain email addresses.
2. **Club Services**
    a. **Model Layer**: ClubModel
    b. **Functions**:
        i. getAllClubs(): return a list of all registered clubs with their club IDs.
        ii. proposeClub(clubDetails): create a new club entry in the database with a status field of PENDING and set the coordinator as the user.
        iii. getClubDetails(clubID): return the details of the specified club.
        iv. updateClubDetails(clubID, clubDetails): set the club details of the club with clubID. Allowed only for Student Council Clubs Coordinator and Admin and the club's coordinator.
        v. approveClubProposal(eventID): check if the user is the Student Council Clubs Coordinator/Admin and set the status field of the club to APPROVED.
        vi. rejectClubProposal(eventID): check if the user is the Student Council Clubs Coordinator/Admin and set the status field of the club to REJECTED.
    c. **Description**: Manage all the actions related to clubs
3. **Event Services**
    a. **Model Layer**: EventModel
    b. **Functions**:
        i. getAllEvents(): return a list of all upcoming events.
        ii. proposeEvent(eventDetails): create a new event entry in the database with a status field of PENDING. Allowed only for club coordinators (check if user is a coordinator of that specific club).
        iii. getEventDetails(eventID): return the details of the specified event ID
        iv. approveEventProposal(eventID): check if the user is the Student Council Clubs Coordinator and set the status field of the event to APPROVED.
        v. rejectEventProposal(eventID): check if the user is the Student Council Clubs Coordinator and set the status field of the event to REJECTED.

            vi. registerForEvent(eventID): register the user for the specific event ID by adding the user's ID to the events list of participants.

   c. **Description**: Manage all the actions related to events

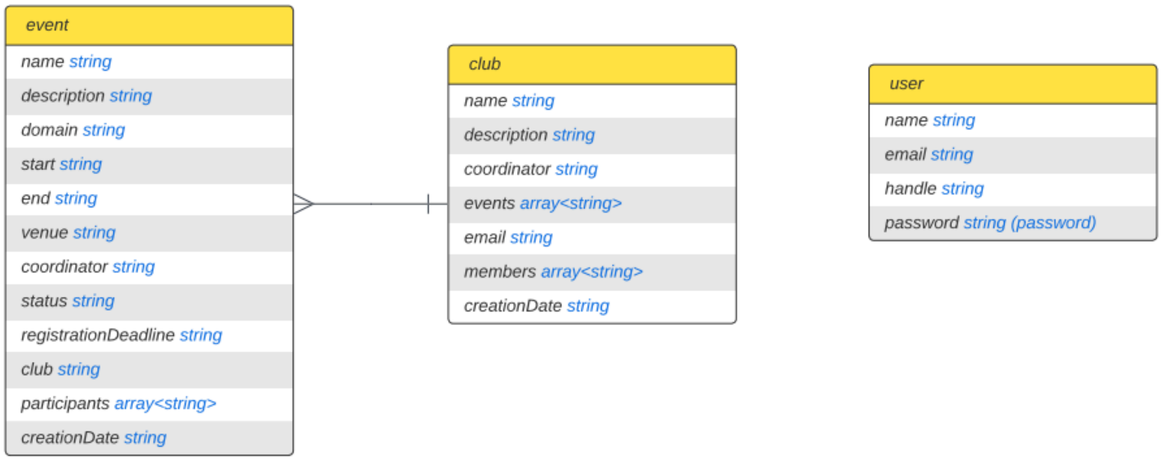
4. **User Services**
   a. **Model Layer**: UserModel
   b. **Functions**:
      i. getUserDetails(userID): returns the profile details of the specified user with a list of registered events by that user by doing a join with the events table.
      ii. updateStudentCouncilClubsCoordinator(userID): check if the user is Admin and update the new SC Clubs Coordinator to userID, and remove the previous SCCC's field from the user database.
   c. **Description**: Manage all the actions related to the user of the website

Database Design

Database Used: MongoDB

- MongoDB will serve as the database for the application. Its NoSQL, document-oriented nature is well-suited for handling diverse data types, such as event details, user profiles, and club information.
- MongoDB's flexibility allows for efficient data retrieval and storage, which is essential for managing event requests and club information.

Schema Design



Detailed Service descriptions of some APIs in OpenAPI Specification

```
openapi: 3.0.0
```

```
info:
 title: Club and Event Management API
 version: 1.0.0
paths:
 /oauth/callback:
   post:
     summary: Handles the callback after an OAuth authentication
flow.
     requestBody:
       required: true
       content:
         application/json:
           schema:
             type: object
             properties:
               userCredentials:
                 type: string
     responses:
       '200':
         description: Authentication successful, returns an
access token.
         content:
           application/json:
             schema:
               type: object
               properties:
                 valid:
                   type: boolean
                 accessToken:
                   type: string

 /clubs:
```

```yaml
  get:
    summary: Retrieves a list of all clubs.
    responses:
      '200':
        description: A list of all clubs is returned.
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Club'

/clubs/request:
  post:
    summary: Submits a request to create a new club.
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ClubRequest'
    responses:
      '202':
        description: Club creation request is submitted.

/clubs/{clubId}/about:
  get:
    summary: Retrieves information about a specific club.
    parameters:
      - in: path
        name: clubId
        required: true
```

```yaml
          schema:
            type: integer
      responses:
        '200':
          description: Detailed information of a club is
provided.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Club'

  /clubs/{clubId}/approve:
    post:
      summary: Approves a club creation or update request.
      parameters:
        - in: path
          name: clubId
          required: true
          schema:
            type: integer
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                proposalId:
                  type: integer
      responses:
        '200':
          description: Club request has been approved.
```

```yaml
/clubs/{clubId}/reject:
  post:
    summary: Rejects a club creation or update request.
    parameters:
      - in: path
        name: clubId
        required: true
        schema:
          type: integer
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              proposalId:
                type: integer
    responses:
      '200':
        description: Club request has been rejected.

/events:
  get:
    summary: Retrieves a list of all upcoming events.
    responses:
      '200':
        description: An array of all upcoming events is
returned.
        content:
          application/json:
             schema:
               type: array
               items:
                 $ref: '#/components/schemas/Event'

 /events/{eventId}:
   get:
     summary: Retrieves information about a specific event.
     parameters:
       - in: path
         name: eventId
         required: true
         schema:
           type: integer
     responses:
       '200':
         description: Detailed information of an event is
provided.
         content:
           application/json:
             schema:
               $ref: '#/components/schemas/Event'

components:
 schemas:
   Club:
     type: object
     properties:
       clubId:
         type: integer
       clubName:
         type: string
```

```
        clubDescription:
          type: string
    Event:
      type: object
      properties:
        eventId:
          type: integer
        eventName:
          type: string
        eventDescription:
          type: string
    ClubRequest:
      type: object
      properties:
        accessToken:
          type: string
        clubData:
          $ref: '#/components/schemas/Club'
```