Code Organization – Student Event Management System

Top Level Directory Structure

```
├── client/
├── docs/
├── .gitignore
├── LICENSE
├── package.json
├── package-lock.json
└── server/
```

Brief Description:

- client folder contains all the front-end code – details given below
- docs folder contains all the documents related do this application
- server contains the code for the back-end – details given below
- Other files are self explanatory

Server Folder (back-end code)

The directory structure of the backend repository code with a brief description of the key folders is given below. (There are some files also like .gitignore, .env, package.json, etc which are not mentioned here.)

```
├── app.ts
├── auth
│   ├── google.ts
│   ├── jwt.ts
│   └── token.ts
├── config
│   ├── auth.ts
│   └── user.ts
├── controllers
│   ├── auth.ts
│   ├── club.ts
│   ├── event.ts
│   └── user.ts
├── database
│   └── mongodbConnection.ts
├── middleware
│   ├── accessMiddleware.ts
│   ├── auth.ts
│   └── error.ts
├── models
│   ├── club.ts
│   ├── event.ts
│   └── user.ts
├── routes
│   ├── auth.ts
│   ├── club.ts
│   ├── event.ts
│   └── user.ts
│
```

```
├── server.ts
├── test
│   ├── script.test.js
│   └── setup.js
└── utils
    ├── APIError.ts
    └── catchAsync.ts
```

Brief Description/Explanation:

- **app.ts**: This is the main application file that initializes and starts the Express server.
- **auth** (folder): Contains modules for authentication, such as OAuth with Google, JWT handling, and token management.
  - **google.ts**: Manages Google OAuth authentication logic.
  - **jwt.ts**: Handles JSON Web Token creation and verification.
  - **token.ts**: Provides functionality related to token generation and management.
- **config** (folder): Contains configuration files for different aspects of the application, like authentication and user settings.
  - **auth.ts**: Token secrets for authentication.
  - **user.ts**: User access level enum.
- **controllers** (folder): Houses controller files that handle client requests and return responses, typically involving business logic.
  - **auth.ts**: Manages authentication-related requests.
  - **club.ts**: Handles requests related to club functionalities.
  - **event.ts**: Manages event-related requests.
  - **user.ts**: Handles user-related requests.
- **database** (folder): Contains database-related files, usually configurations and connection setup.
  - **mongodbConnection.ts**: Sets up and configures the connection to MongoDB using the URI from an environment variable.
- **middleware** (folder): Holds middleware functions that process requests before reaching the final route handlers.
  - **accessMiddleware.ts**: Handles access control and permissions to populate the request object with user info.
  - **auth.ts**: Middleware for authentication requests to passport.js library.
  - **error.ts**: Error handling middleware (error converter and handler).
- **models** (folder): Contains models that define the data structure for the application's database.
  - **club.ts**: Defines the data model for club entities.
  - **event.ts**: The data model for event entities.
  - **user.ts**: User data model**.**
- **routes** (folder): Contains files that define various routes/endpoints of the application.
  - **auth.ts**, **club.ts**, **event.ts**, **user.ts**: Define routes related to authentication, clubs, events, and users, respectively.
- **server.ts**: the main file configuring the express server with its middlewares and routes.
- **test** (folder): Contains test scripts and configurations for application testing.
  - **script.test.js**: Contains all the test cases described using the Jest library.

    - **setup.js**: Setup configurations for tests.
- **utils** (folder): Utility functions and helpers used across the application.
    - **APIError.ts**: Defines a custom error class for API-related errors.
    - **catchAsync.ts**: Utility for handling asynchronous operations and errors.

Client Folder (Front-end code)

The directory structure of the frontend, along with a brief description of the key sub folders is given here. (As with server, there are other files which are not listed here)

```
Client Folder
├── README.md
├── package-lock.json
├── package.json
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── manifest.json
│   └── robots.txt
├── src
│   ├── components
│   │   ├── EventForm.jsx
│   │   ├── Navbar.jsx
│   │   └── RequireAuth.jsx
│   ├── features
│   │   └── user
│   │       └── user.js
│   ├── index.js
│   ├── pages
│   │   ├── Club.jsx
│   │   ├── Clubs.jsx
│   │   ├── Event.jsx
│   │   ├── Home.jsx
│   │   ├── Login.jsx
│   │   ├── LoginSuccess.jsx
│   │   └── Profile.jsx
│   ├── store
│   │   └── store.js
│   ├── styles
│   │   └── index.css
│   └── utils
│       └── baseUrl.js
└── tailwind.config.js
```

Brief Description/Explanation of source files:

- **index.js**: Sets up a React application with Redux and React Router, rendering components for different routes like login, profile etc.

- **components** (folder): Contains reusable React components like EventForm, Navbar, and RequireAuth used across the application.
    - **EventForm.jsx**: Provides a form for creating event requests, using state hooks for form inputs and Redux for accessing the user token.
    - **Navbar.jsx**: Creates a navigation bar with links to Home, Clubs, and User Profile, using Redux to access the user's handle for the profile link.
    - **RequireAuth.jsx**: checks if a user is logged in using Redux, and either renders its children or redirects to the login page.
- **features/user/user.js**: Manages user state, including login status and profile information, with an asynchronous thunk for user login.
- **pages** (folder): Contains the main React components that represent different pages of the application, such as Clubs, Event, Home, Login, Profile etc.
    - **Club.jsx:** Fetches and displays information about a club based on the club handle from the URL, and includes an EventForm for creating new events.
    - **Clubs.jsx**: Fetches and displays a list of all clubs, with each club name linking to its specific page.
    - **Event.jsx**: Fetches and displays information about a specific event based on the event handle from the URL, using the user's token for authorization.
    - **Home.jsx**: Fetches and displays a list of upcoming events from the API, with a message and image displayed if there are no upcoming events.
    - **Login.jsx**: Displays a login page with a Google sign-in option, and redirects to the home page if the user is already logged in.
    - **LoginSuccess.jsx**: handles successful login, extracting the access token and user handle from the URL parameters, updating the Redux state, and then navigating to the home page.
    - **Profile.jsx**: Fetches and displays user profile information based on the user handle from the URL, including ‘about’ info and badges for coordinator roles.
- **store/store.js**: Sets up the Redux store for the application, importing and using the userReducer to manage the user slice of the state.
- **styles/index.css**: Sets up global styles for the application and imports Tailwind CSS.
- **utils/baseUrl.js**: Defines the `apiBaseUrl` variable.