**Project Documentation: Student Community Website**

**Table of Contents**

1. **Introduction**
   * **Purpose:** The Student Community Website aims to foster collaboration and interaction among students within a college or university. It provides a platform for students to connect, share experiences, and pursue common interests.
   * **Features:**
     + User authentication for secure interactions.
     + Community pages to facilitate student discussions and engagement.
     + Hobbies section to join or create interest groups.
   * **Technologies Used:**
     + MongoDB: A NoSQL database for storing user data, posts, and hobbies.
     + Express.js: A web application framework for building the backend.
     + React.js: A JavaScript library for building the user interface.
     + Node.js: A runtime environment for executing server-side code.

start:frontend

1. **Architecture Overview**
   * **Frontend (React.js):**
     + The frontend is structured using React components and follows a modular design for better maintainability.
     + State management is handled using React hooks and context.
     + React Router is used for client-side routing.
   * **Backend (Node.js & Express.js):**
     + The backend follows the Model-View-Controller (MVC) architecture.
     + Middleware functions are implemented for authentication and error handling.
     + Express routes are organized to handle different API endpoints.
   * **Database (MongoDB):**
     + MongoDB is used as a NoSQL database to store user data, posts, and hobbies.
     + Mongoose is used as an Object Data Modeling (ODM) library for MongoDB and Node.js.
2. **Folder Structure**
   * **Frontend:**
     + **src/components**: Contains reusable React components.
     + **src/pages**: Houses React components that represent different pages of the application.
     + **src/services**: Frontend API services for communicating with the backend.
   * **Backend:**
     + **src/controllers**: Express.js route controllers that handle requests.
     + **src/models**: MongoDB schema models for defining data structures.
     + **src/routes**: Backend API routes for routing HTTP requests.
     + **src/config**: Configuration files for setting up the application.
3. **Database Schema**
   * **Users:**
     + **username**: String, unique username for each user.
     + **email**: String, unique email address for each user.
     + **password**: String, hashed password for user authentication.
   * **Posts:**
     + **title**: String, title of the post.
     + **content**: String, content of the post.
     + **user\_id**: ObjectId, references the user who created the post.
   * **Hobbies:**
     + **name**: String, name of the hobby.
     + **description**: String, description of the hobby.
4. **Features and Functionality**
   * **User Authentication:**
     + Users can register by providing a unique username, email, and password.
     + Authentication is implemented using JWT (JSON Web Tokens).
     + Users can log in and log out securely.
   * **Student Community Pages:**
     + Users can view posts created by others.
     + They can create new posts and comment on existing ones.
   * **Interactions and Activities:**
     + Users can like posts to show appreciation.
     + Following other users enables personalized feeds and notifications.
   * **Joining Hobbies:**
     + Users can explore available hobbies.
     + They can join existing hobby groups or create new ones.
5. **User Interface**
   * **Landing Page:**
     + Welcomes users and provides a brief overview of the platform.
   * **Student Community Page:**
     + Displays a feed of posts from the student community.
     + Allows users to interact with posts, like, and comment.
   * **User Profile Page:**
     + Shows user details, posts, and followers/following information.
   * **Hobbies Page:**
     + Lists available hobbies and allows users to join or create new ones.
6. **Deployment**
   * **Frontend Deployment:**
     + Deploy the React.js application to platforms like Netlify or Vercel.
   * **Backend Deployment:**
     + Deploy the Node.js and Express.js server to platforms like Heroku.
   * **Database Deployment:**
     + Host the MongoDB database on cloud services like MongoDB Atlas.
7. **Testing**
   * **Unit Testing:**
     + Use Jest and React Testing Library for frontend unit tests.
     + Mocha and Chai can be employed for backend unit tests.
   * **Integration Testing:**
     + Test interactions between frontend and backend components.
     + Use tools like Supertest for API endpoint testing.
   * **User Acceptance Testing:**
     + Engage real users for testing the application in a production-like environment.
8. **Security**
   * **Authentication and Authorization:**
     + Secure user authentication using JWT.
     + Implement role-based access control for different user privileges.
   * **Data Validation:**
     + Validate user inputs on both frontend and backend to prevent injection attacks.
   * **HTTPS Implementation:**
     + Ensure all communication between clients and the server is encrypted.