**FREELANCER FINDER APPLICATION USING MERN**

**1. INTRODUCTION**

**Project Title:** Freelancer Finder Application

**Team ID :** LTVIP2025TMID53116

**Team Members:**

**Team Leader:** Pinniganti Neeharika

**Team member:** Neelam Saiteja

**Team member:** Neeima Vusa

**Team member:** Nandam Janaki

**2. PROJECT OVERVIEW**

**Purpose:**

SB Works is a full-stack freelancing platform built using the MERN stack (MongoDB, Express.js, React.js, Node.js). It connects clients with skilled freelancers, enabling seamless project posting, bidding, collaboration, and work submission. The platform prioritizes efficiency, transparency, and security, with an admin team ensuring quality control and smooth communication. SB Works aims to be a reliable hub for clients seeking talent and freelancers building their careers.

**Features:**

* **Project Posting:** Clients can create projects with detailed descriptions, budgets, and required skills.
* **Bidding System:** Freelancers browse projects, filter by skills, and submit proposals with portfolio samples.
* **Secure Communication:** An integrated chat system allows real-time collaboration between clients and freelancers, with support for file attachments.
* **Work Submission and Review:** Freelancers submit completed work via the platform; clients review and provide feedback or request revisions.
* **Admin Oversight:** Admins monitor platform activity, enforce policies, resolve disputes, and manage users.
* **User Profiles:** Freelancers showcase skills, experience, and ratings; clients view detailed profiles to select candidates.
* **Real-Time Notifications:** Users receive updates on project status, new opportunities, and messages.
* **Authentication:** Secure JWT-based authentication for user registration, login, and role-based access (client, freelancer, admin).

**Scenario based case-study:**

Sarah, a recent graduate with a degree in graphic design, is eager to showcase her skills and build a strong freelance portfolio. She stumbles upon SB Works while searching for online freelancing opportunities.

**Finding the Perfect Project**: Impressed by the user-friendly interface, Sarah browses through various project categories. She discovers a project posted by a local bakery, "Sugar Rush," seeking a logo redesign. The project description details the bakery's brand identity and target audience, giving Sarah a clear understanding of the client's needs.

**Bidding with Confidence**: Confident in her design skills, Sarah dives into the project details. SB Works allows her to review the bakery's previous marketing materials, further solidifying her design approach. She submits a compelling proposal highlighting her relevant experience and attaching a few samples from her portfolio stored securely within the platform.

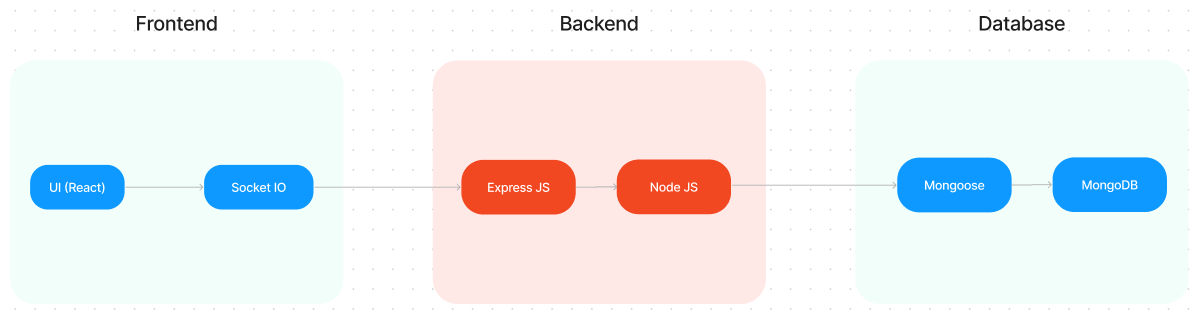
**Communication & Collaboration:** "Sugar Rush" selects Sarah's proposal based on her impressive portfolio and competitive pricing. SB Works facilitates seamless communication between Sarah and the bakery, allowing them to discuss project specifics and refine the design direction through an integrated chat system.

**Delivery & Feedback:** Once finalized, Sarah submits her logo design through the SB Works platform. "Sugar Rush" can review the design, provide feedback, and request minor revisions if needed. SB Works fosters a collaborative environment where both parties can work towards achieving the desired outcome.

**Building a Thriving Career:** Following a successful project completion and a glowing review from "Sugar Rush," Sarah's profile on SB Works gains traction. The positive experience encourages her to actively seek new projects on the platform. With a growing portfolio and strong client testimonials, Sarah is well on her way to establishing a thriving freelance career on SB Works.

**3. ARCHITECTURE**

**Technical architecture:**



The technical architecture of SB Works follows a client-server model, where the frontend serves as the client and the backend acts as the server. The frontend encompasses the user interface, presentation, and integrates the Axios library to facilitate easy communication with the backend through RESTful APIs.

**Frontend:**

The frontend is built with React.js, utilizing reusable components for modularity (e.g., ProjectCard, ChatBox, Profile). Bootstrap and Material UI ensure a polished, responsive UI. Axios facilitates HTTP requests to the backend APIs. React Context manages global state, such as user data and search filters, for efficient data sharing across components. Navigation is handled with React Router for seamless transitions between pages like the landing page, dashboards, and project details.

**Backend:**

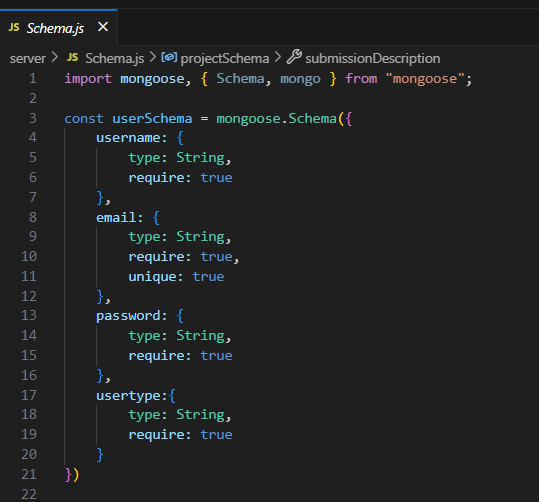
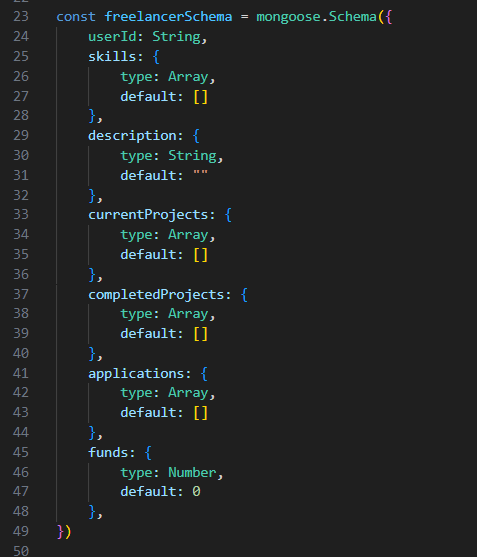
The backend leverages Node.js and Express.js to manage server-side logic and RESTful APIs. Mongoose provides structured interactions with MongoDB, defining schemas for data consistency. JWT secures user authentication, while Bcrypt hashes passwords. CORS enables cross-origin requests, and body-parser processes incoming data. Routes are organized into modules (e.g., user, project, chat) for maintainability.

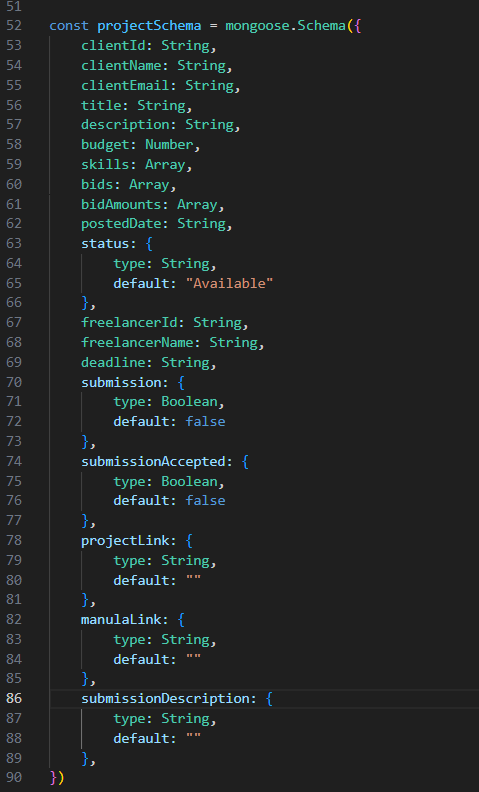
**Database:**

MongoDB, a NoSQL database, stores data in JSON-like documents for flexibility and scalability. The database includes the following collections:

* **Users:** Stores user details (name, email, password, account type).
* **Projects:** Contains project information (title, description, budget, skills).
* **Applications:** Holds freelancer proposals (proposal text, rate, portfolio links).
* **Chats:** Stores project-specific communication history.
* **Freelancers:** Extends user data with skills, experience, and ratings.

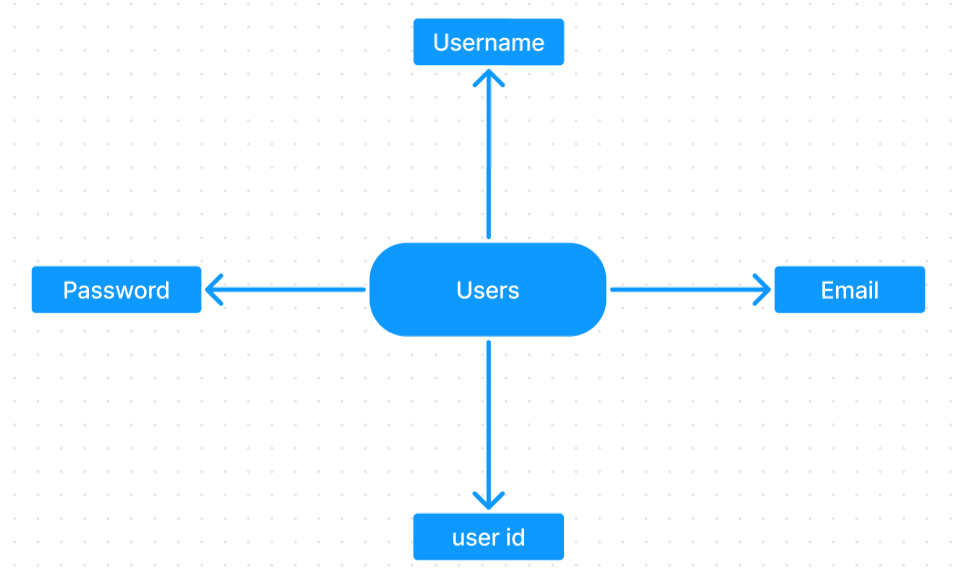
The Schemas for the database are given below

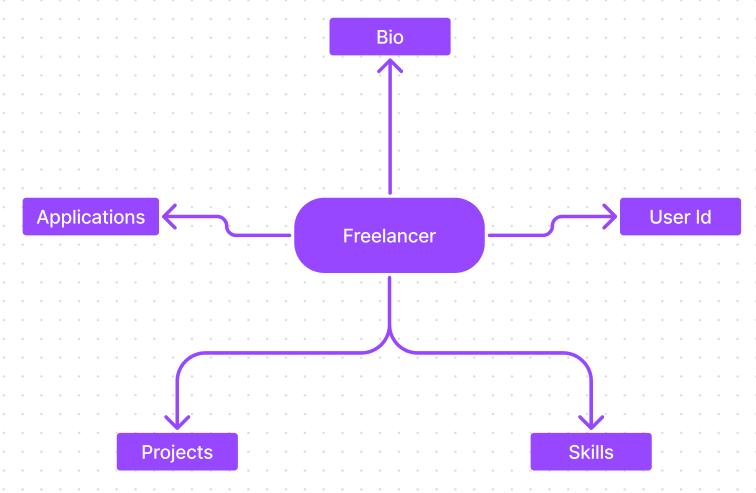
** **

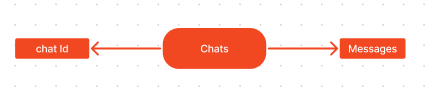
****

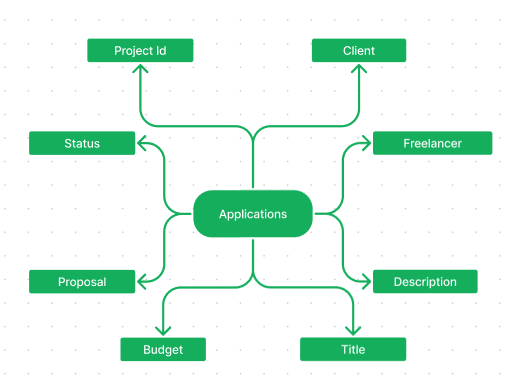
****

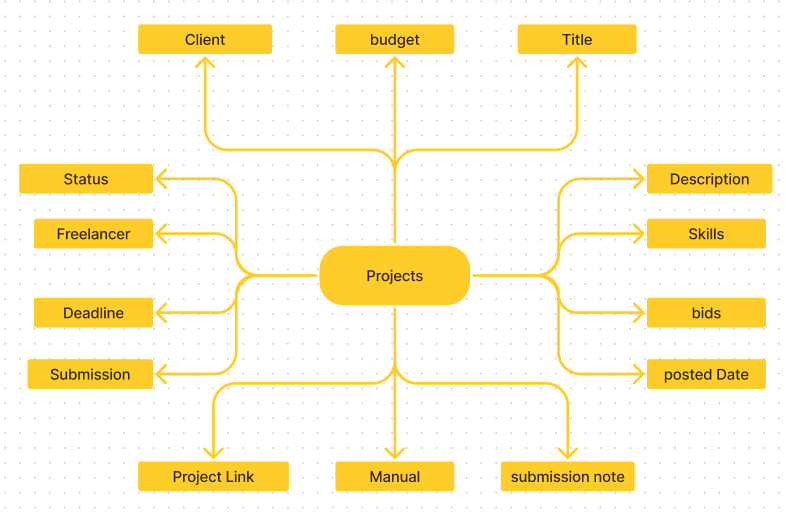
**ER-DIAGRAM:**

****

****

****

****

****

**4. SETUP INSTRUCTIONS**

**PRE-REQUISTIC:**

Here are the key prerequisites for developing a full-stack application using Express Js, MongoDB, React.js:

✔**Node.js and npm:**

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server-side. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

Download: https://nodejs.org/en/download

Installation instructions: <https://nodejs.org/en/download/package-manager/>

✔**Express.js:**

Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture.

Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware, and API development.

Installation: Open your command prompt or terminal and run the following command:

**npm install express**

✔**MongoDB:**

MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data.

Download: https://www.mongodb.com/try/download/community

Installation instructions: <https://docs.mongodb.com/manual/installation/>

✔**React.js:**

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

Follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

✔**HTML, CSS, and JavaScript**:

Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

✔**Database Connectivity**:

Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Express Js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations

✔**Front-end Framework**:

Utilize React Js to build the user-facing part of the application, including entering booking room, status of the booking, and user interfaces for the admin dashboard. For making better UI we have also used some libraries like material UI and bootstrap.

✔**Version Control**:

Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

Git: Download and installation instructions can be found at: https://git-scm.com/downloads

✔**Development Environment**: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

• **Visual Studio Code:** Download from <https://code.visualstudio.com/download>

To access the code , refer the drive link:

<https://drive.google.com/drive/folders/1RYZz_W_QJEvvpinWY7-ruQ7tDyWhBAC_?usp=sharing>

**Installations:**

• Navigate into the cloned repository directory:

cd freelancer-app-MERN

• Install the required dependencies by running the following commands:

cd client

npm install

../cd server

npm install

Start the Development Server:

• To start the development server, execute the following command:

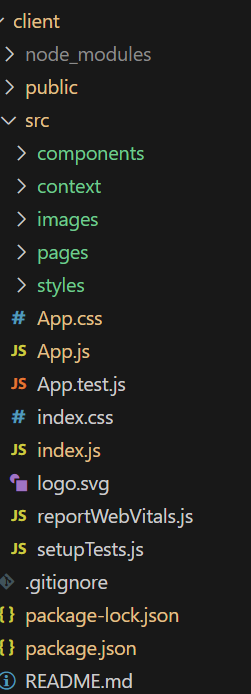
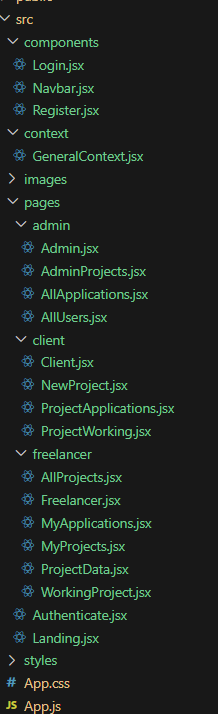
**npm start**

• The SB Works app will be accessible at <http://localhost:3000>

You have successfully installed and set up the SB Works application on your local machine. You can now proceed with further customization, development, and testing as needed.

**5. FOLDER STRUCTURE**

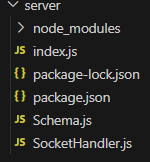
**Frontend:**

** **

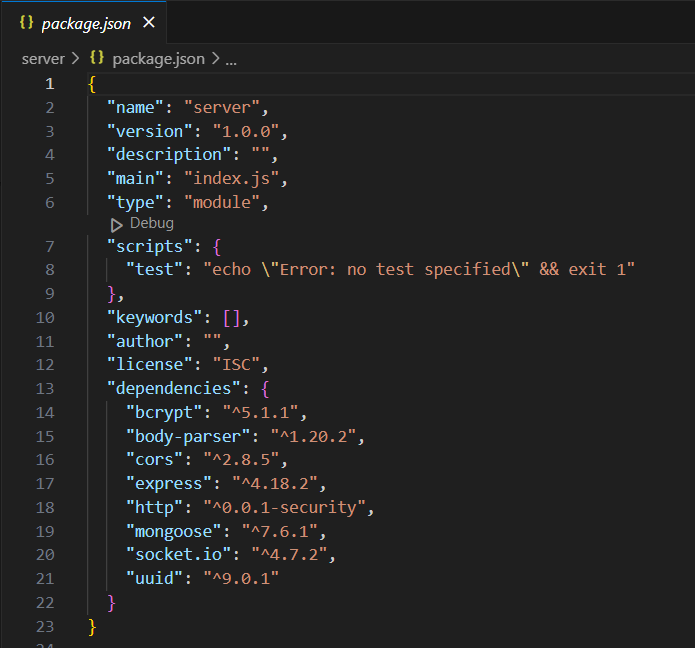
**Frontend package.json:**



**Backend:**

****

**Backend package.json:**



**6. RUNNING THE APPLICATION**

**Frontend :**

Start the frontend server:

**cd client**

**npm start**

start the backend server:

**cd server**

**npm start**

Open http://localhost:3000 in a browser to use the application.

**7. API DOCUMENTATION**

Key API endpoints (prefixed with /api):

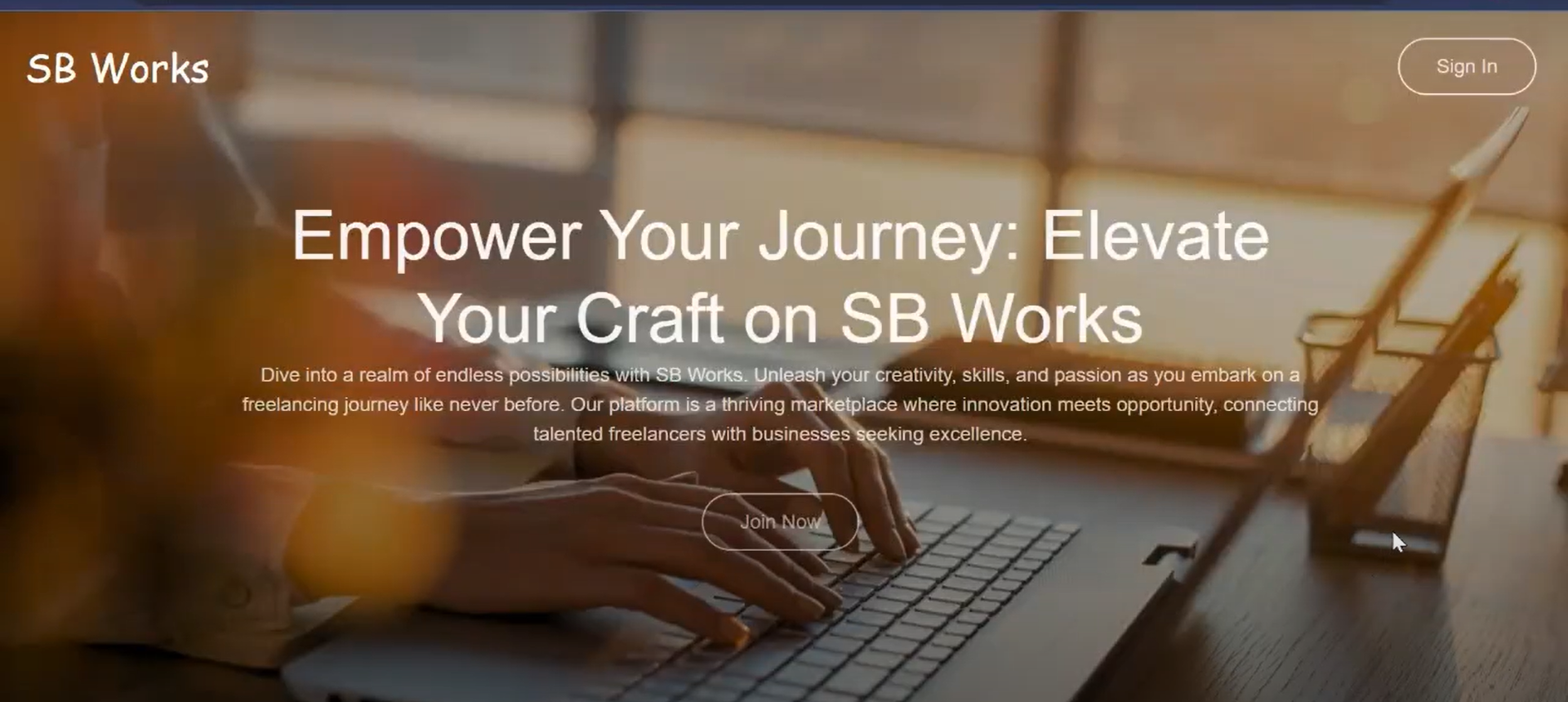
* **User Management:**
  + **POST /users/register:** Register user.
    - Body: { name, email, password, accountType }
    - Response: { userId, name, email, token }
  + **POST /users/login**: Log in.
    - Body: { email, password }
    - Response: { userId, name, email, token }
  + **GET /users/profile:** Get profile (JWT required).
    - Response: { name, email, accountType }
* **Project Management:**
  + **POST /projects:** Create project (client-only, JWT required).
    - Body: { title, description, budget, skillsRequired }
    - Response: { projectId, title, description }
  + **GET /projects**: List open projects.
    - Response: [{ projectId, title, budget, ... }]
  + **GET /projects/:id:** Project details.
    - Response: { projectId, title, description, ... }
* **Application Management:**
  + **POST /applications:** Submit proposal (freelancer-only, JWT required).
    - Body: { projectId, freelancerId, proposal, rate, portfolioLinks }
    - Response: { applicationId, status }
  + **GET /applications/project/:projectId:** List project applications (client-only).
    - Response: [{ applicationId, freelancerId, proposal, ... }]
* **Chat:**
  + **POST /chats:** Send message (JWT required).
    - Body: { projectId, senderId, message }
    - Response: { chatId, message, timestamp }
  + **GET /chats/:projectId:** Get chat history.
    - Response: { projectId, messages: [{ senderId, message, timestamp }, ...] }
* **Freelancer Profiles:**
  + **GET /freelancers/:id:** View profile.
    - Response: { userId, skills, experience, ratings, portfolio }
  + **PUT /freelancers/:id:** Update profile (freelancer-only, JWT required).
    - Body: { skills, experience, portfolio }
    - Response: { userId, skills, experience }

**8. AUTHENTICATION**

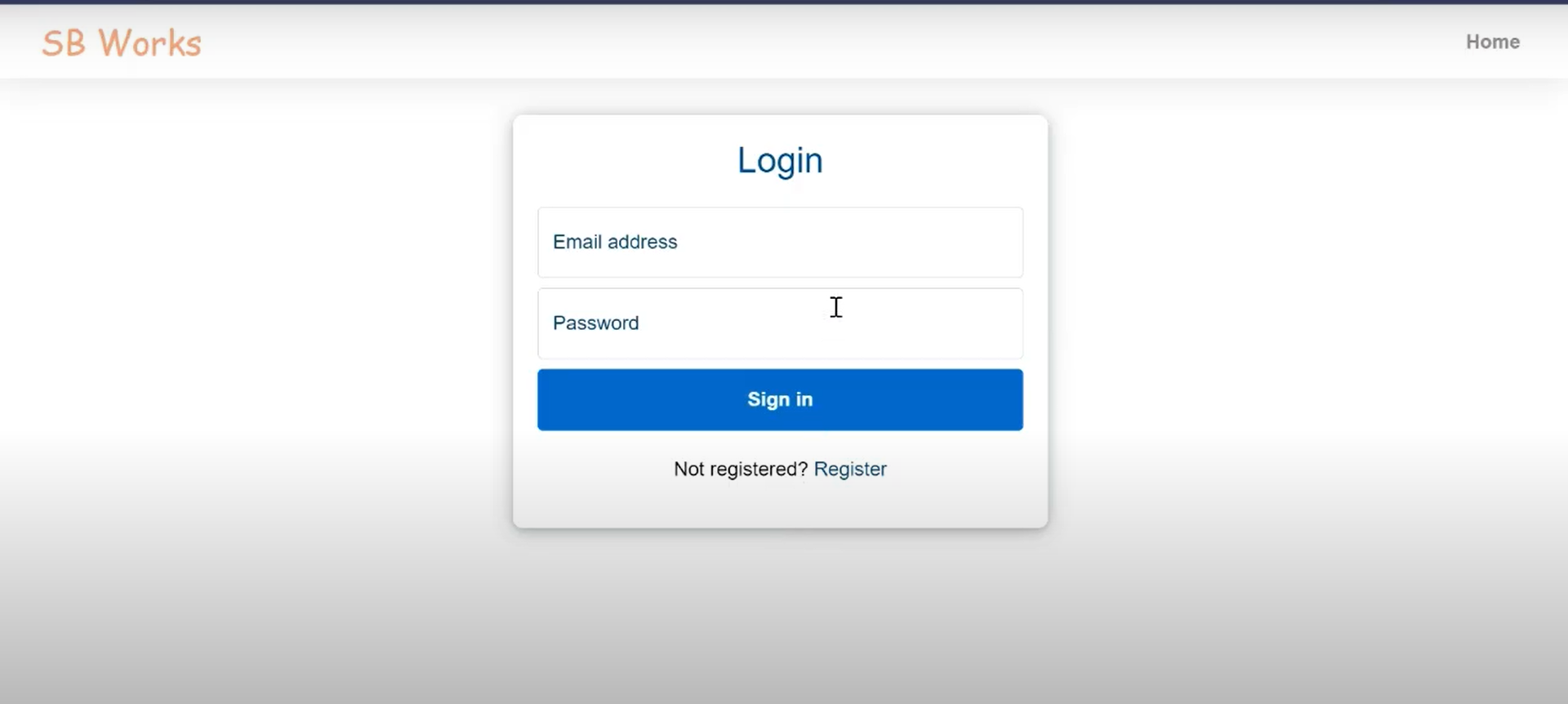
JWT handles authentication. Users receive a token upon registration/login, stored in the frontend ( System: localStorage or cookies). Protected routes (e.g., project creation, proposal submission) require the token in the Authorization header (Bearer <token>). Middleware validates tokens and enforces role-based access (client, freelancer, admin). Passwords are hashed with Bcrypt for security.

**9. USER INTERFACE**

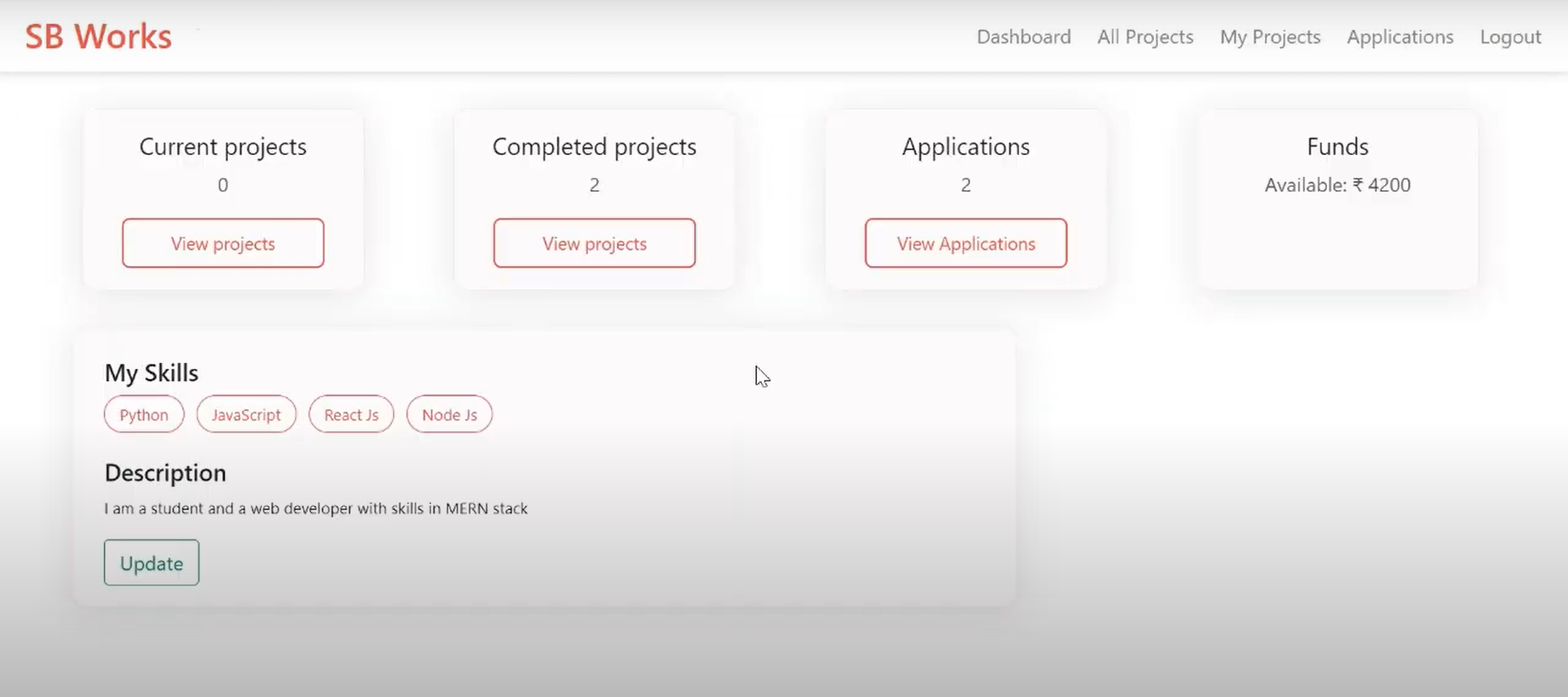
**Landing page:**

****

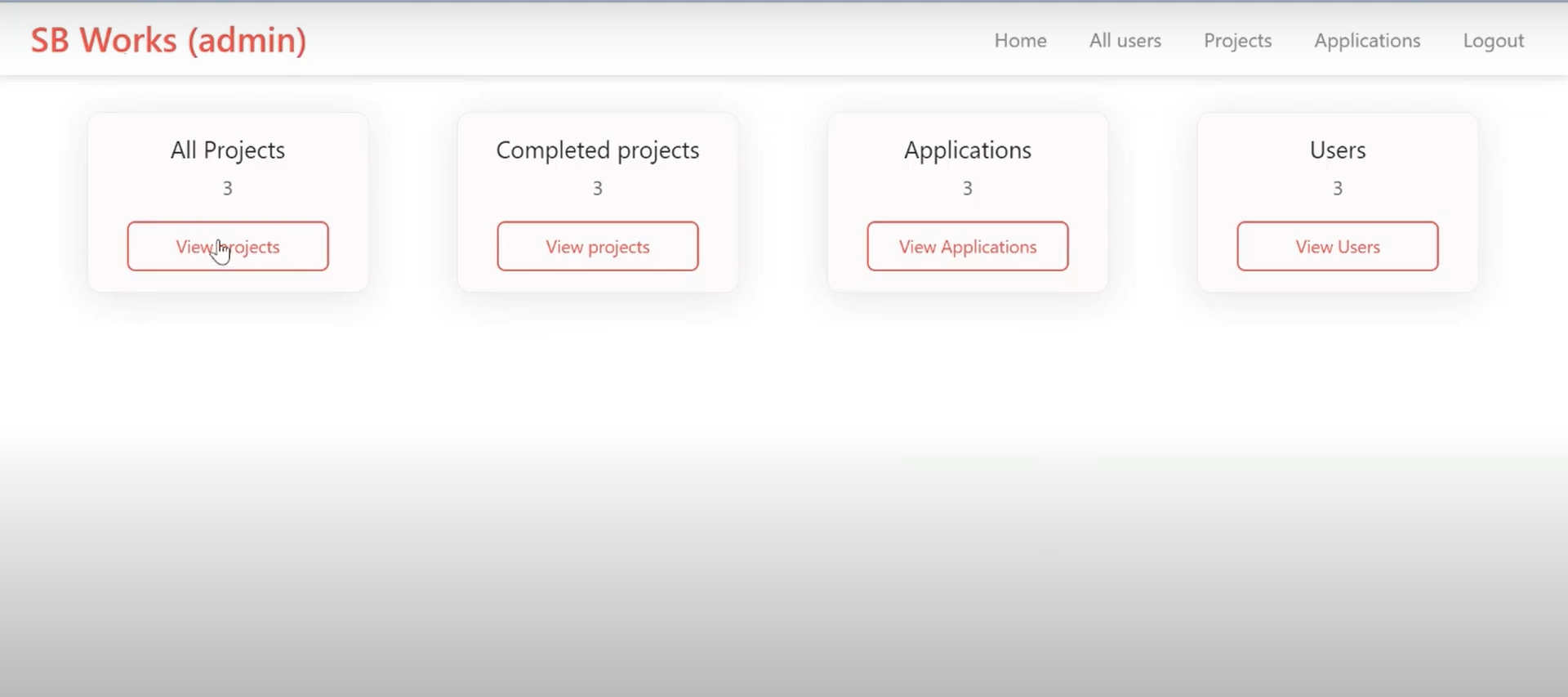
**Authentication:**

****

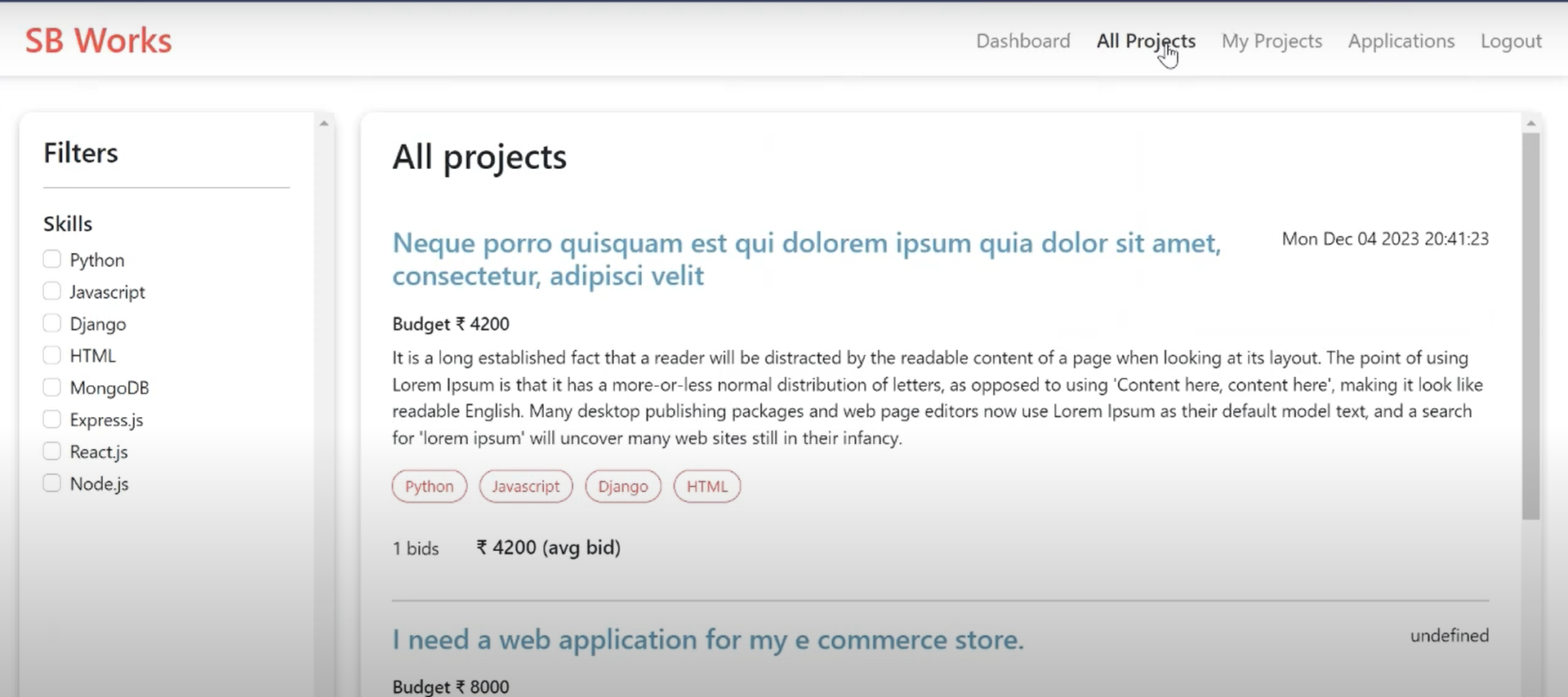
**Freelancer dashboard:**

****

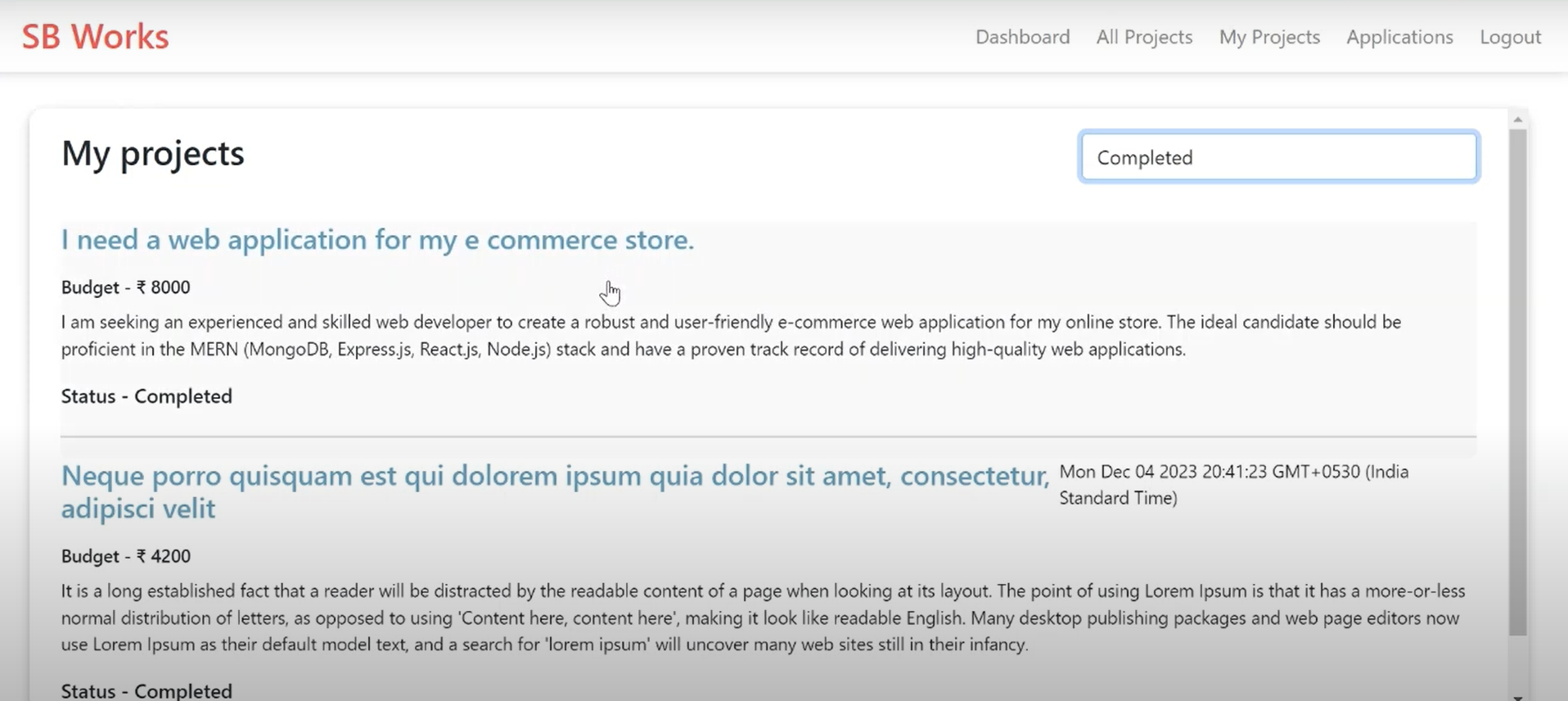
**Admin dashboard:**

****

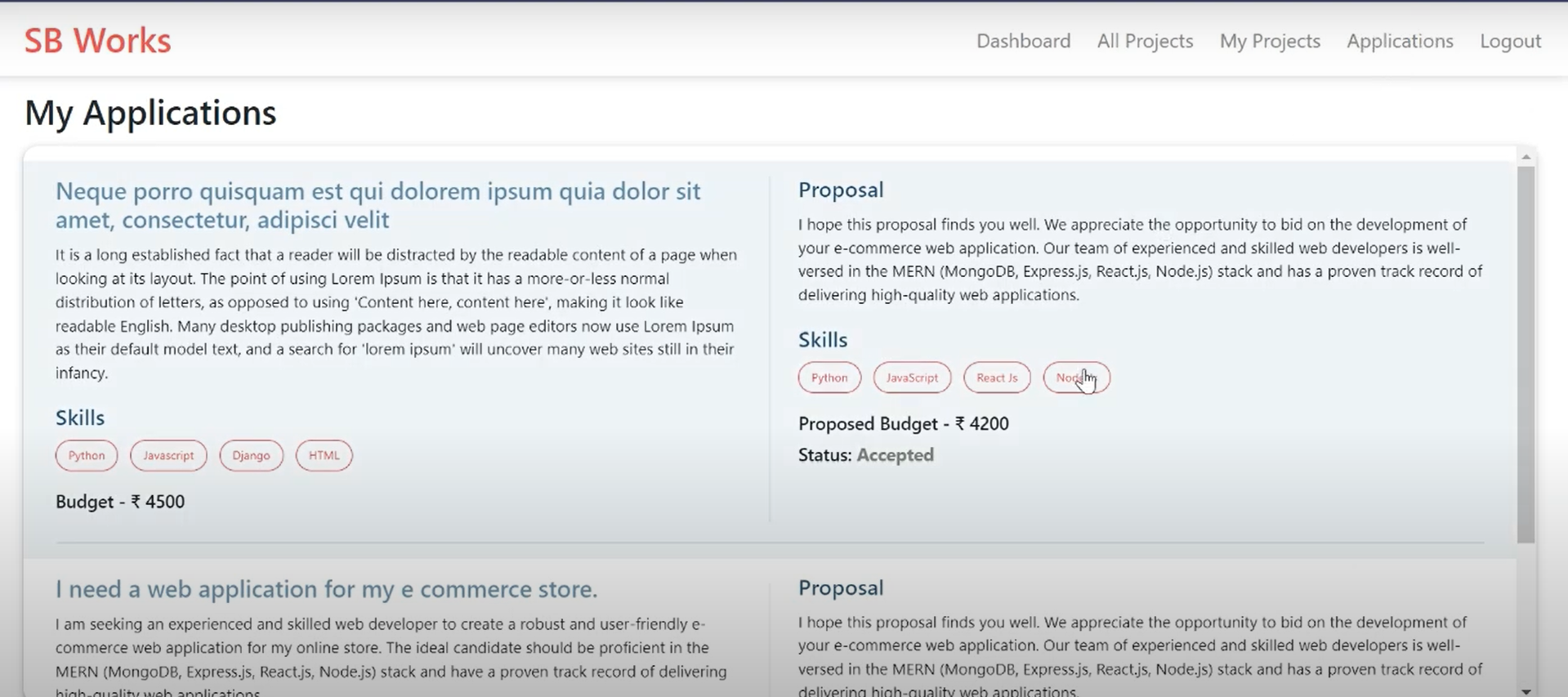
**All projects:**

****

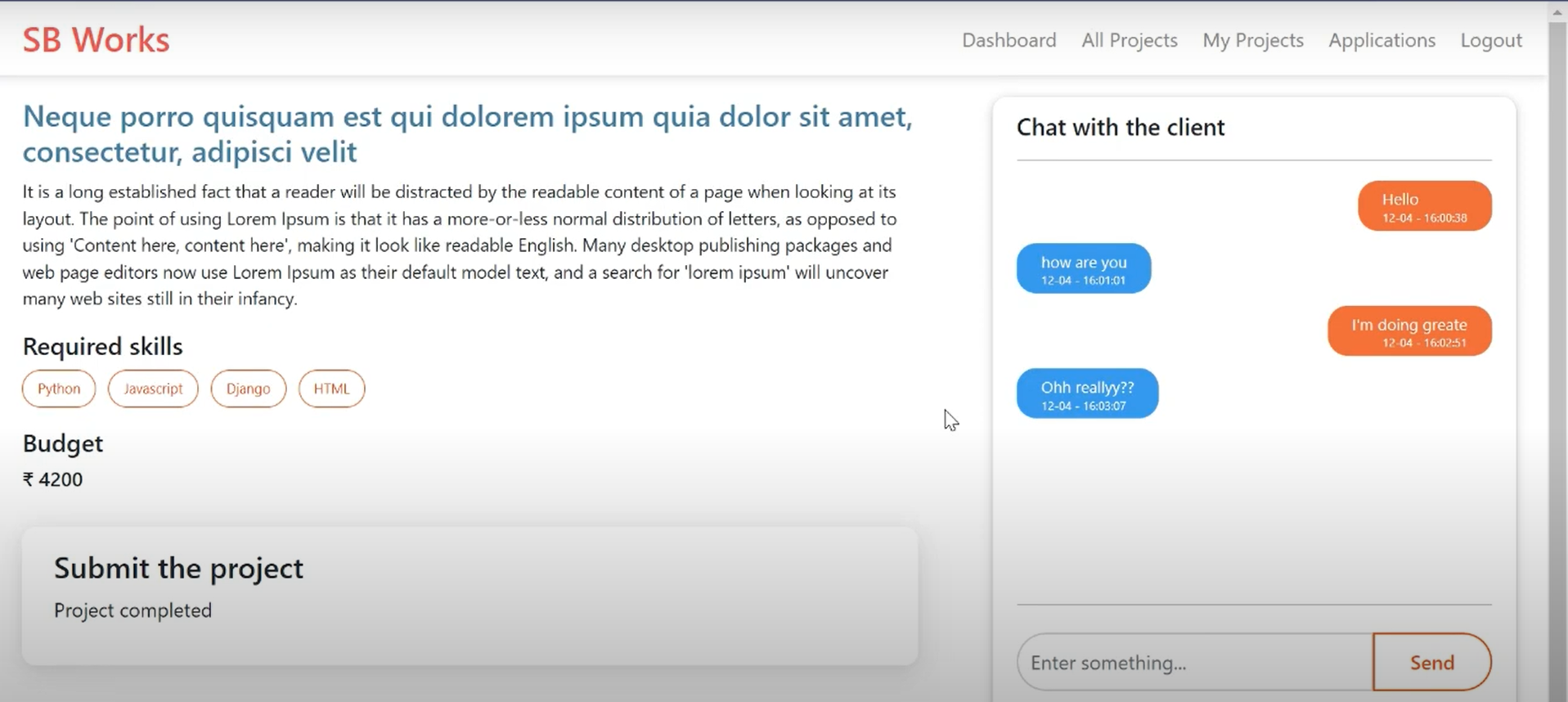
**Freelance projects:**

****

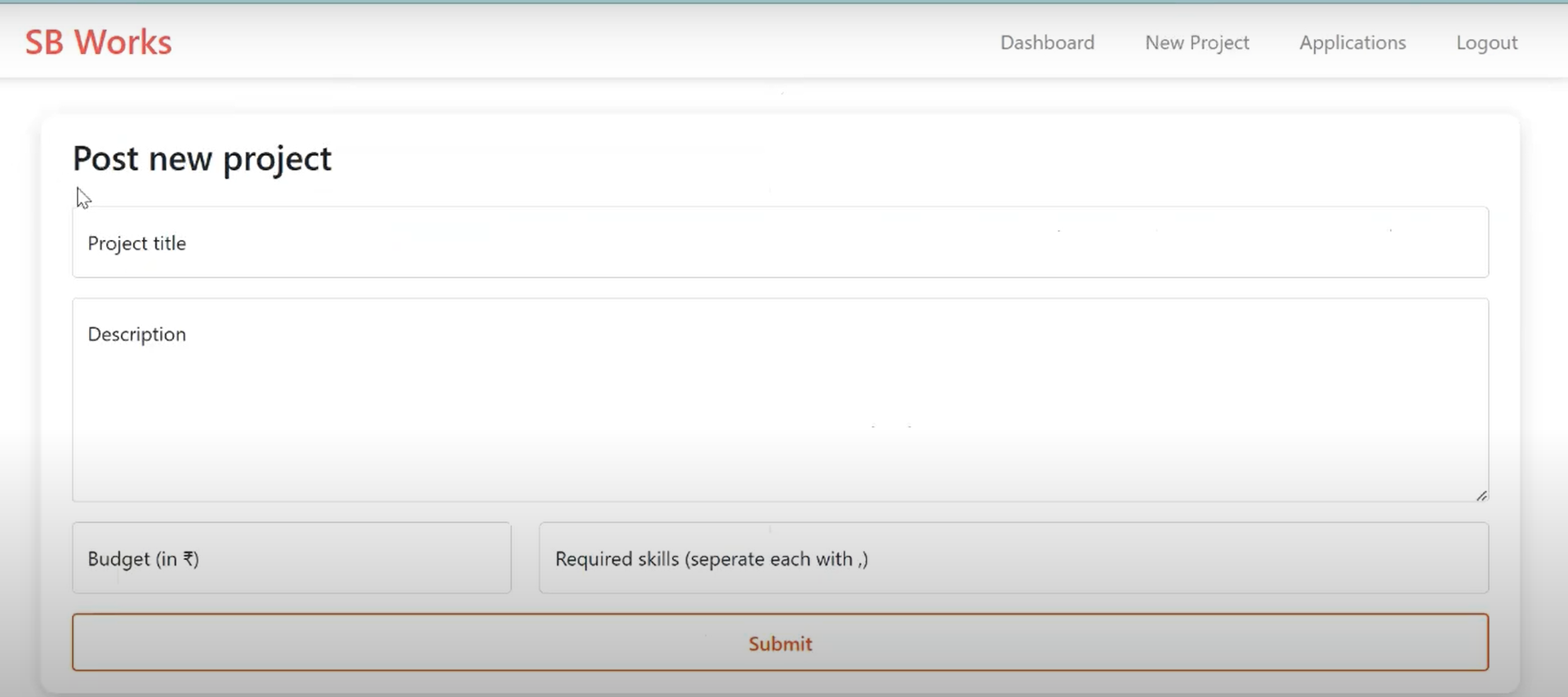
**Applications:**

****

**Project page:**

****

**New project:**

****

**10. TESTING**

**Manual testing verified:**

* User registration, login, and profile updates.
* Project creation, bidding, and freelancer selection.
* Real-time chat and file uploads.
* Admin features (user management, dispute resolution).
* Cross-browser and device compatibility.

Future testing could use Jest for unit tests and Cypress for end-to-end tests.

**11. DEMO**

Watch the demo here:<https://drive.google.com/file/d/1nEXaygs0Cb4NuxyhJHcPj3DMZXRK6nXj/view?usp=sharing>

**12. KNOWN ISSUES**

* **Chat Latency:** Delays in real-time updates under high load.
* **Mobile Responsiveness:** Minor UI alignment issues on small screens.
* **Admin Analytics:** Lacks advanced reporting features.
* **File Uploads:** Large files may impact performance.

**13. FUTURE ENHANCEMENTS**

* Add automated testing with Jest and Cypress.
* Integrate payment processing (e.g., Stripe).
* Enhance admin panel with analytics and reports.
* Optimize mobile UI and develop a native app.
* Implement AI-based project recommendations.
* Add multi-language support.