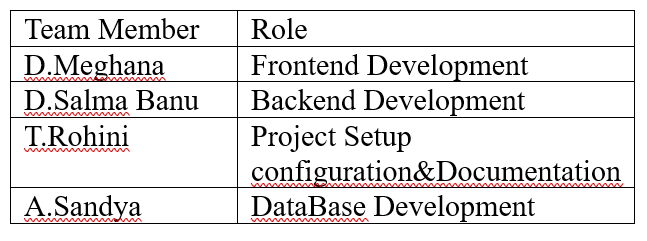
**Full Stack Development with MERN Project Documentation format**

# 1. Introduction:

* **Project Title:** Freelance Finder
* **Team Members:**

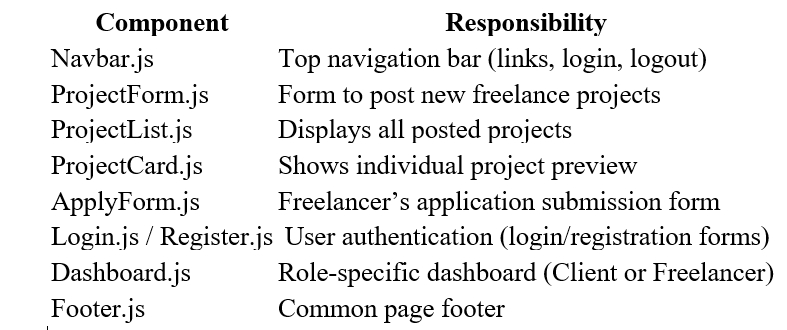


# 2. Project Overview :

* **Purpose:**
* Connect Clients with Freelancers
* Simplify the Hiring Process
* Support Remote and Flexible Work
* Enable Direct Communication
* **Features:**
* User Registration/Login
* Post Projects
* Manage Projects
* View Freelancer Applications
* Apply for Projects

**3. Architecture:**

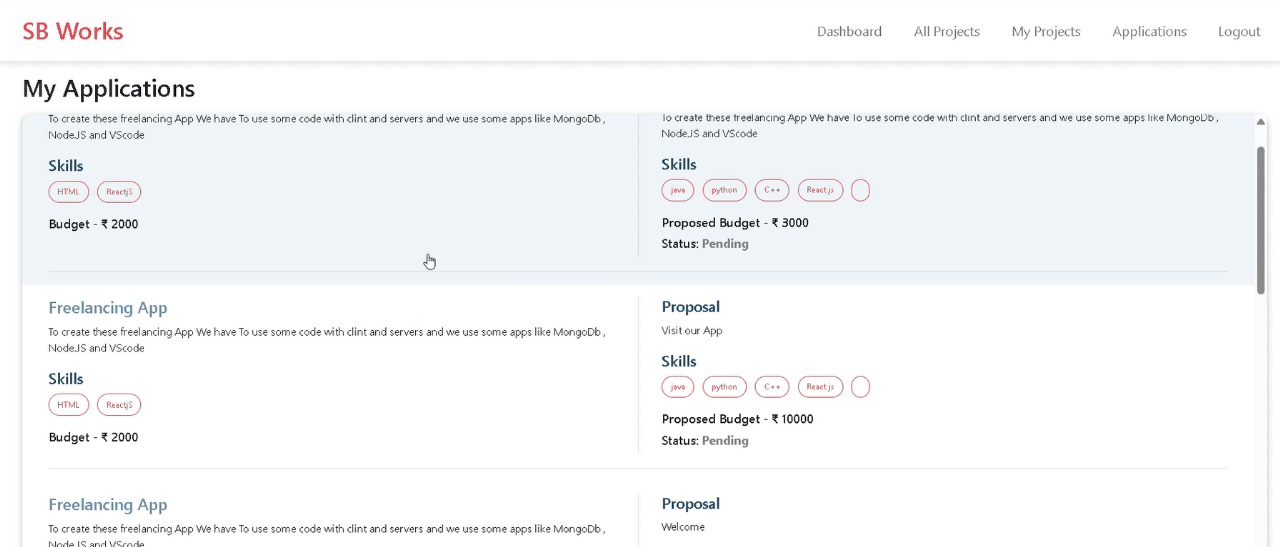
* **Frontend:**



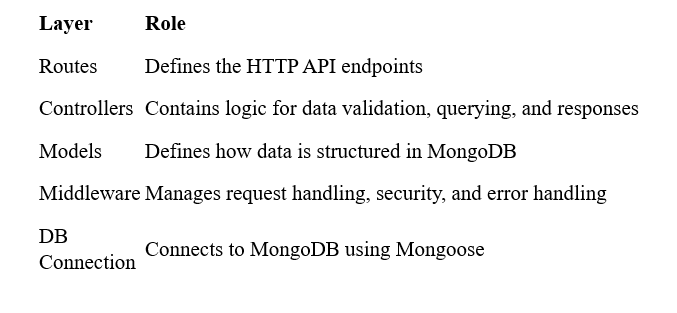


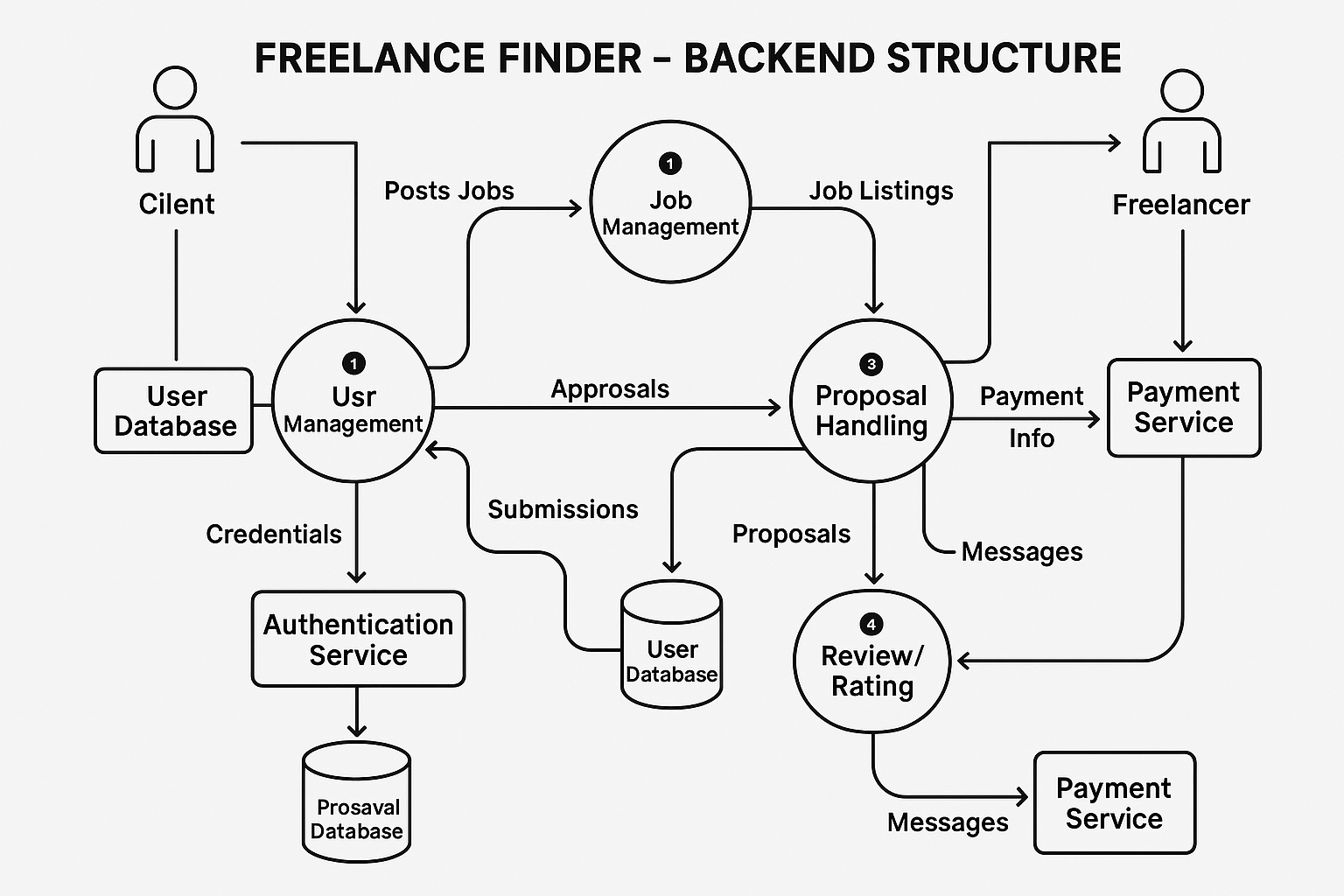
# Login/Register Interface:





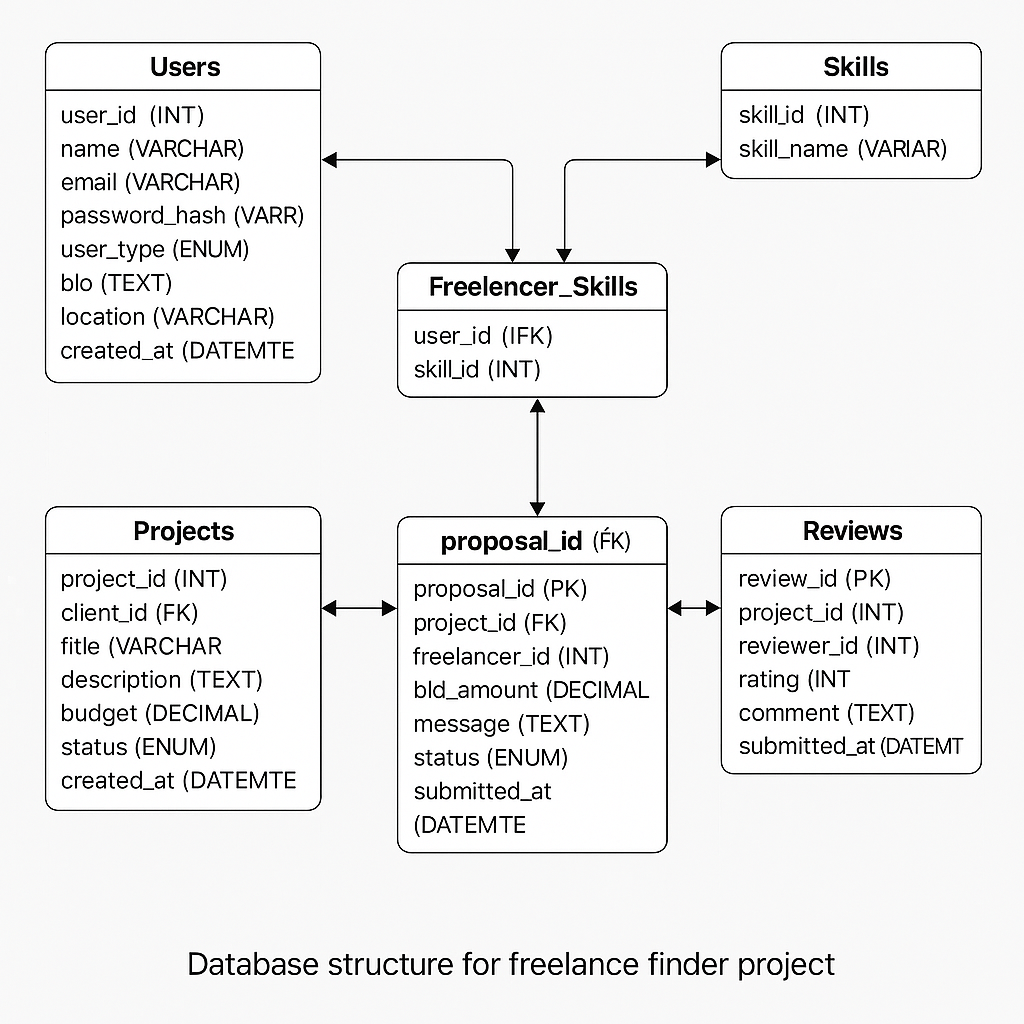
**Backend:**





**Database:**

| **Operation** | **Code Example** |
| --- | --- |
| **Create User** | User.create ({ name, email, password }) |
| **Login (Find User)** | User.findOne({ email }) |
| **Post Project** | Project.create({ title, createdBy: userId }) |
| **List Projects** | Project.find().populate('createdBy') |
| **Apply to Project** | Application.create({ projectId, freelancerId, coverLetter }) |
| **View Applications** | Application.find({ projectId }).populate('freelancerId') |



# 4. Setup Instructions :

* **Prerequisites:**

### **For Both Frontend and Backend:**

### **Node.js** (v16 or above) → [Download Node.js](https://nodejs.org/)

### **MongoDB** (Local or MongoDB Atlas) → [Download MongoDB](https://www.mongodb.com/try/download/community)

### **Visual Studio Code** **Git (optional)** → [Download Git](https://git-scm.com/)

**Installation:**

1️⃣ Clone or extract the project files (with client and server folders).  
2️⃣ Install backend dependencies: cd server && npm install  
3️⃣ Create .env in server/ with MongoDB URI and PORT.  
4️⃣ Install frontend dependencies: cd client && npm install  
5️⃣ Start the backend server: cd server && npm start  
6️⃣ Start the frontend app: cd client && npm start  
7️⃣ Open in browser at: http://localhost:3000

# 5. Folder Structure :

* **Client:**

client/

├── public/

│ └── index.html

├── src/

│ ├── components/

│ │ └── Navbar.js, Footer.js, etc.

│ ├── pages/

│ │ └── Home.js, Login.js, Register.js, Dashboard.js, etc.

│ ├── services/

│ │ └── api.js ← Axios calls to backend

│ ├── App.js ← Main component with routes

│ ├── index.js ← Entry point that renders App

│ └── styles/ ← (Optional) CSS files

├── package.json

└── .env

**Folder Purpose:**

* public/  
  Contains the index.html file where the React app is injected.
* src/components/  
  Holds reusable UI elements like buttons, headers, navbars**.**
* src/pages/  
  Contains full page components for different routes (like Home, Login, Register).
* src/services/  
  Manages API calls to the backend using Axios.
* App.js  
  Central component that defines all the routes.
* index.js  
  Starting point of the app; renders <App /> to the DOM.
* .env  
  Stores environment variables like the backend API URL.

This structure keeps the frontend modular, organized, and easy to maintain**.**

**Bottom of Form**

**Server:**

server/

├── models/

│ └── User.js, Project.js, Application.js # MongoDB schemas

├── routes/

│ └── userRoutes.js, projectRoutes.js # API route handlers

├── controllers/ (optional)

│ └── Logic separated from routes

├── config/

│ └── db.js # MongoDB connection setup

├── .env # Environment variables

├── index.js # Main server file (entry point)

├── package.json # Lists dependencies & scripts

**Folder/Files Description**

* **models/**  
  Contains Mongoose schemas for different data models like users, projects, and applications.
* **routes/**  
  Defines REST API endpoints (e.g., /api/users, /api/projects) and links them to logic.
* **controllers/** *(if used)*  
  Holds business logic functions (e.g., login, register, create project), keeping route files clean.
* **config/db.js**  
  Establishes connection to MongoDB using Mongoose.
* **index.js**  
  The main entry point that sets up the Express server, middleware, and routes.
* **.env**  
  Stores sensitive config values like PORT and MONGO\_URI.
* **package.json**  
  Declares project dependencies (e.g., express, mongoose, cors, dotenv).

# 6. Running the Application :

• Provide commands to start the frontend servers locally.

* **Frontend:**
* Open your terminal
* Navigate to the client directory cd client
* Start the React development server:
* npm start
* This will run the frontend at:  
  [**http://localhost:3000**](http://localhost:3000)

**Backend:**

Open Visual Studio Code or your terminal.

2️⃣ Go to the backend directory:

cd server

3️⃣ Install the required packages:

npm install

4️⃣ Create a .env file in the server folder with this content:

env

PORT=5000

MONGO\_URI=mongodb://localhost:27017/freelance\_finder

5️⃣ Start the backend server:

npm start

6️⃣ Backend is now running at:

http://localhost:5000

* **7. API Documentation:**

**1. Register User**

* Method: POST
* URL: /api/users/register
* Body: name, email, password
* Creates a new user

**2. Login User**

* Method: POST
* URL: /api/users/login
* Body: email, password
* Returns a token for login

**3. Post New Project**

* Method: POST
* URL: /api/projects
* Headers: Authorization token
* Body: title, description, budget
* Creates a project (client only)

**4. Get All Projects**

* Method: GET
* URL: /api/projects
* Shows list of all projects

**5. Apply to Project**

* Method: POST
* URL: /api/applications
* Headers: Authorization token
* Body: projectId, proposal
* Freelancer applies to a project

**6. Get Applications by Project**

* Method: GET
* URL: /api/applications/:project
* Shows all applications to one project

# Authentication:

**1. User Authentication**

* Users (Clients or Freelancers) **register** using:
  + POST /api/users/register
* They **log in** using:
  + POST /api/users/login

**2. JWT Token Generation**

* When a user logs in:
  + The server checks the email & password.
  + If valid, it creates a **JWT token** using jsonwebtoken.
  + This token contains the user's ID and role (client or freelancer).

**3. Token Usage**

* The frontend stores the token (e.g., in localStorage).
* For protected API routes, the token is sent in the request headers:

**4. Route Protection**

* Middleware (authMiddleware.js) on the server:
  + Reads the token.
  + Verifies it using a secret key.
  + Grants or denies access based on role (client/freelancer).

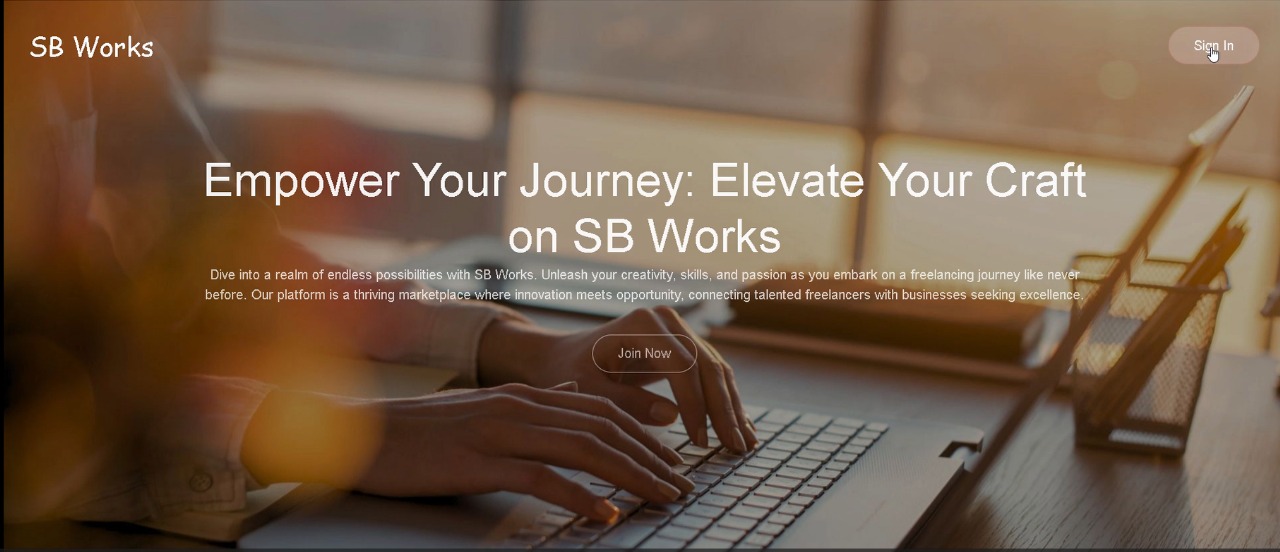
**5. Role-based Authorization**

* Only **clients** can:
  + Post projects
* Only **freelancers** can:
  + Apply to projects
* Both can:
  + View projects

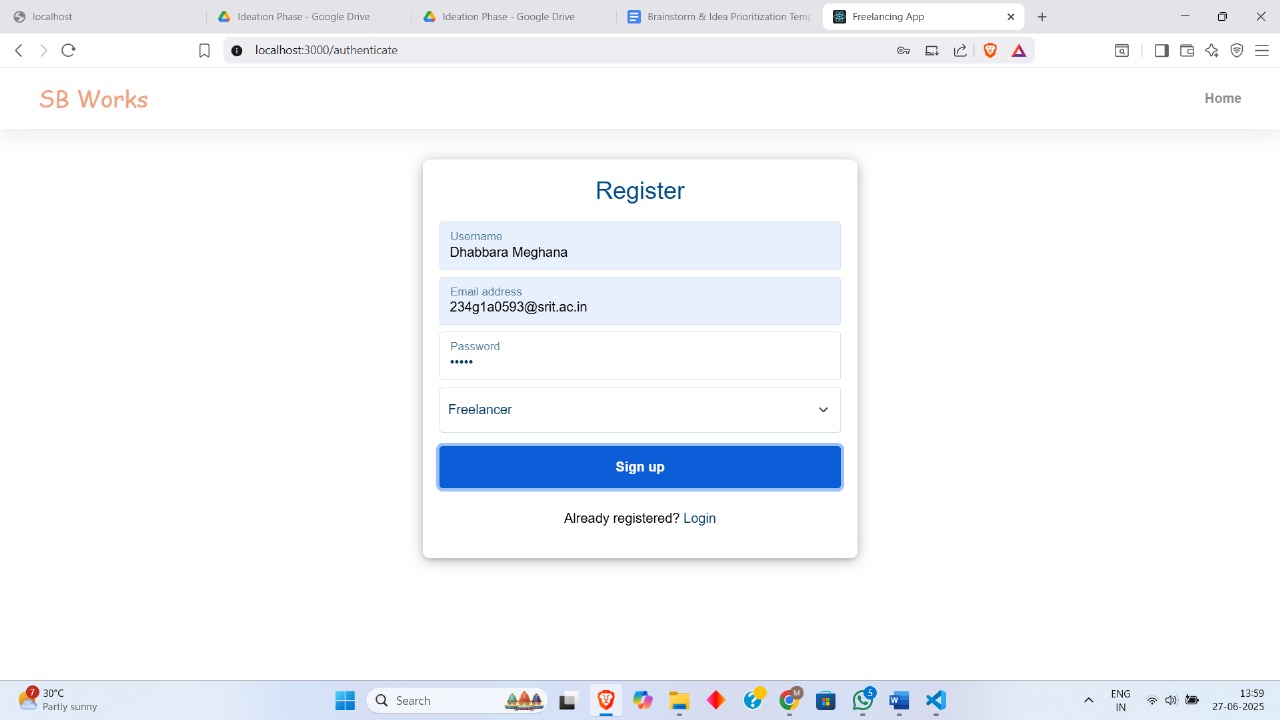
**Technologies Used**

| **Purpose** | **Tool Used** |
| --- | --- |
| Password Hashing | bcryptjs |
| Token Creation | jsonwebtoken |
| Middleware Logic | Express.js |

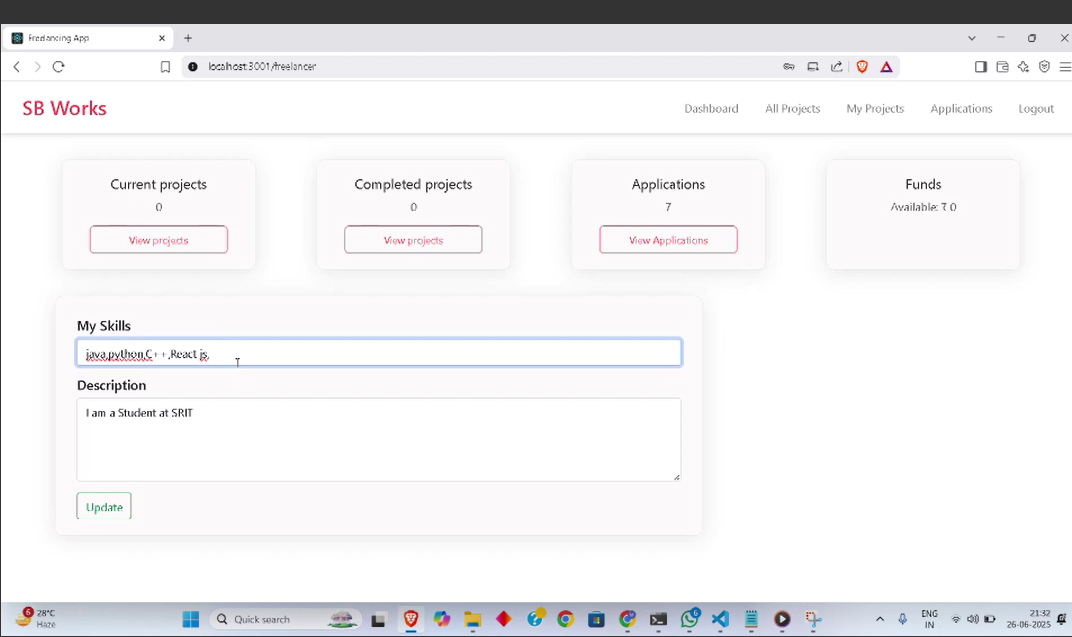
**10.User Interface: Home Page:**



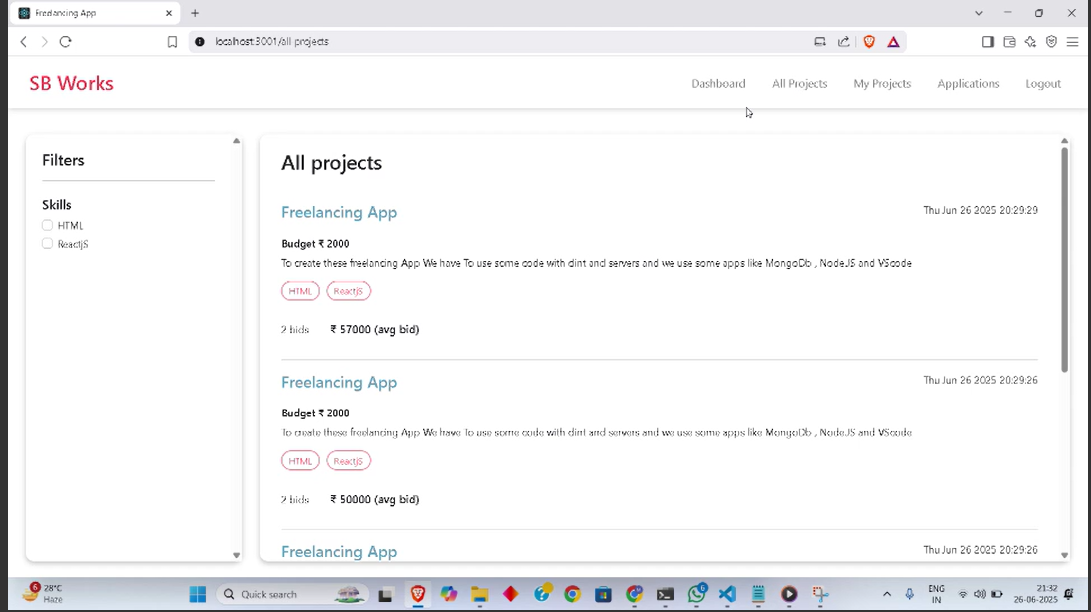
**User Registration / Login:**



**Client Dashboard:**



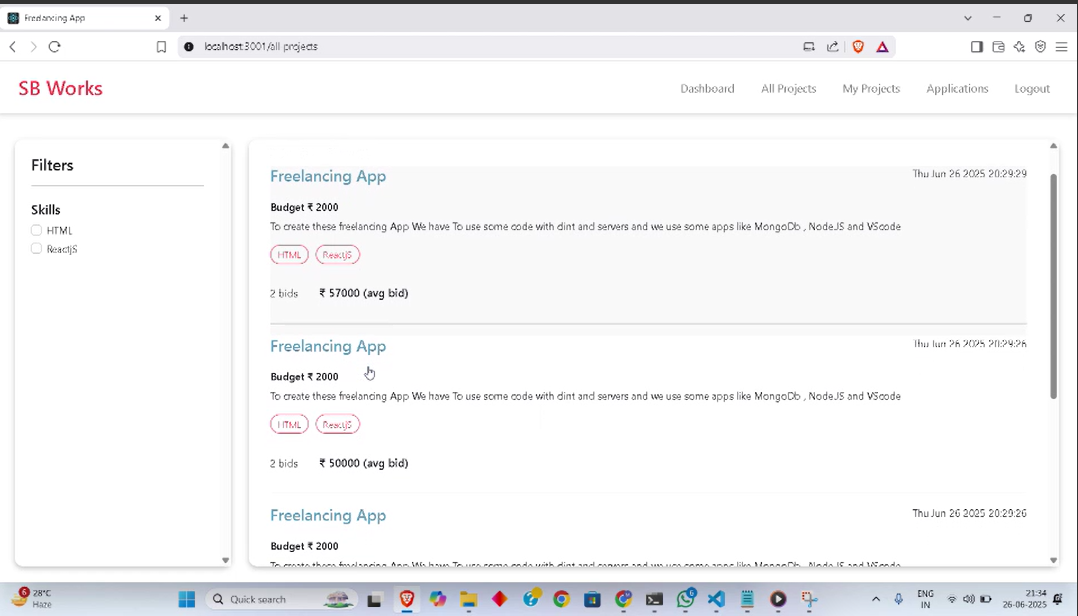
**Freelancer Dashboard:**



**Project Details:**

****

**Responsive Design:**

****

**10 .Testing :**

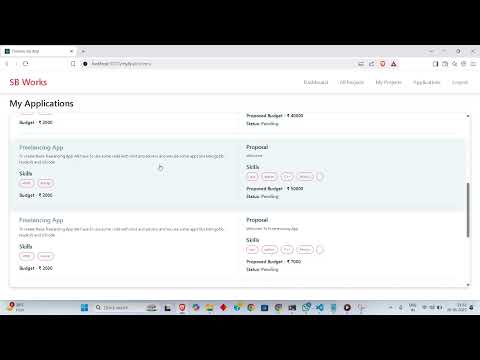
**Frontend Testing (React)**

* **Type:** Manual Testing
* **What was tested:**
  + Form inputs (registration, login, project posting)
  + Button clicks and page navigation
  + Conditional rendering based on user roles (client/freelancer)
* **Tool Used:**
  + React Developer Tools (browser extension)
  + Browser console logs and alerts

**Backend Testing (Node.js/Express):**

* **Type:** Basic Automated Testing (Optional) + Manual API Testing
* **Tools:**
  + **Postman** – for sending test requests to backend routes.
  + **MongoDB Compass** – to verify stored data.
* **What was tested:**
  + User registration/login endpoints
  + Token-based route protection
  + Project creation and retrieval
  + Application submission

**11. Screenshots or Demo:**

[](https://www.youtube.com/embed/3XL_qHvRON4?feature=oembed)

**12. Known Issues:**

**Form Validation**

* ❗ **Issue:** Forms do not always show clear error messages if required fields are left blank.
* 💡 **Fix Suggestion:** Add client-side validation using libraries like Formik or Yup.

**Token Expiry Handling**

* ❗ **Issue:** Expired JWT tokens are not always handled properly; users remain on the dashboard until a manual refresh.
* 💡 **Fix Suggestion:** Add automatic logout or token refresh mechanism on expiry.

**No Role-Based UI Restriction**

* ❗ **Issue:** UI does not always hide unauthorized buttons (e.g., freelancers might see "Post Project").
* 💡 **Fix Suggestion:** Add role checks on the frontend components.

**No Password Reset Option**

* ❗ **Issue:** There is no "Forgot Password" or reset functionality.
* 💡 **Fix Suggestion:** Implement password reset via email using NodeMailer or similar.

**Limited Mobile Responsiveness**

* ❗ **Issue:** Some pages may not display properly on smaller screens.
* 💡 **Fix Suggestion:** Improve mobile layout with responsive CSS or a framework like Bootstrap/Tailwind.

# 13. Future Enhancements:

**1. Advanced Search & Filters**

* Add filters for category, budget, experience level, and location to improve project and freelancer discovery.

**2. Email Notifications**

* Send automated emails for actions like application received, project accepted, or account registered.

**3. Real-Time Chat**

* Integrate a messaging system so clients and freelancers can communicate directly on the platform.

**4. Mobile App**

* Build a mobile version using **React Native** for better accessibility on phones and tablets.

**5. Profile Enhancements**

* Add profile pictures, skill tags, portfolio uploads, and ratings/reviews for freelancers and clients.

**6. Payment Integration**

* Implement payment gateways like **Stripe** or **Razorpay** for secure in-app payments between clients and freelancers.

**7. Admin Dashboard**

* Create an admin panel to manage users, projects, and monitor system usage.

**8. Two-Factor Authentication (2FA)**

* Improve account security with SMS/email-based 2FA during login.