**Freelance finder: Discovering, Opportunities, Unlocking Potential**

**Project Title:** Freelance Finder

**Team Members:** Nagireddi. Kumari

Moram. Rashmi Sri

Mopidevi. Sivasai kumar

Mubasharunnisa

1. **Introduction:**

**Freelance Finder (SB Works)** is a modern, full-stack freelancing platform built to seamlessly connect clients and freelancers. Developed using the **MERN stack** (MongoDB, Express.js, React.js, and Node.js), it provides an intuitive interface where clients can post projects and skilled freelancers can submit competitive bids to collaborate.

**Freelance Finder** is designed to simplify the entire freelancing lifecycle — from project creation and bid submission to work progress and final delivery. Clients can review freelancer profiles and portfolios to select the most suitable candidate, while freelancers can easily highlight their skills, track projects, and communicate with clients in real-time via an integrated chat feature.

1. **Project Overview:**

The purpose of this project is to develop a fully functional, user-friendly web application tailored to the client’s specific requirements. The goal is to deliver a responsive and scalable solution that enhances user engagement, streamlines business processes, and achieves the client’s digital objectives.

**Scalability for future feature additions:**

A modern and intuitive user interface Seamless integration between frontend and backend Secure data management and user authentication Scalability for future feature additions.

**Key Features**

The application will include the following core features:

**1. Responsive Frontend Interface:** Developed using React.js to ensure a smooth and dynamic user experience across devices.

**2. User Authentication & Authorization:** Secure sign-up/login system using JWT for protected routes and role-based access control.

**3. RESTful API:** Integration Efficient communication between frontend and backend using standardized API endpoints (Node.js + Express).

**4. Database Management:** MongoDB used for structured storage of user data, content, and application configurations.

**5. Admin Panel (Optional):** A dashboard for managing content, users, and system settings with appropriate access rights.

**6. Form Validation & Notifications:** Interactive forms with real-time validation and success/error feedback.

**7. Performance Optimization:** Lazy loading, code splitting, and efficient database queries to ensure fast load times

**8. Deployment Ready:** Fully deployable on platforms like Vercel, Netlify (frontend) and Render, Heroku, or Railway (backend)

1. **Architecture**

* **Frontend:**

Built with React.js, the frontend provides a highly interactive and responsive user experience. It features dynamic components tailored for different user roles, including clients, freelancers, and administrators. The component-based structure enables code reusability and enhances maintainability, allowing developers to make updates easily without affecting other parts of the application. Axios is utilized for all API communications, ensuring a seamless flow of data between the client and the server. Error handling and loading states are thoughtfully integrated into the interface, enhancing the overall usability.Additionally, the UI leverages Material UI and Bootstrap to ensure a polished and modern design aesthetic. Material UI provides a robust set of prebuilt components that improve development speed, while Bootstrap's responsive grid system allows the app to adapt gracefully to various screen sizes.

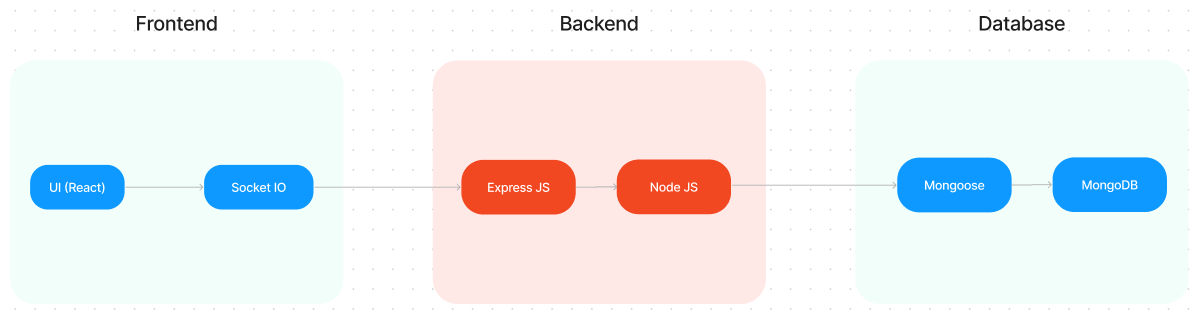
* **Backend:**

The backend is powered by Node.js and Express.js, which serve as the foundation for a scalable and efficient API. Express.js simplifies route handling and middleware integration, allowing for clear and maintainable server-side logic. The backend exposes RESTful APIs that support all core operations, from user registration and login to project management and bid submissions. This architecture enables the separation of concerns, ensuring that the backend remains modular and easily testable.Moreover, the backend integrates Socket.io to enable real-time chat functionality, allowing clients and freelancers to communicate instantly. This real-time capability significantly improves the collaborative experience, making it easier for teams to resolve issues quickly and keep everyone updated.

* **Database:**

Freelance Finder uses MongoDB as its primary database, which is well-suited for storing structured and semi-structured data. MongoDB's document-based design allows for flexibility in schema design, making it easier to iterate on new features and store diverse data types. The data models for users, projects, freelancers, and applications are carefully defined, ensuring efficient querying and robust data relationships. Indexes can also be added to optimize common search and retrieval operations. By leveraging MongoDB's built-in scalability and cloud compatibility, Freelance Finder is well-equipped to handle large datasets and increased user demand.

**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.

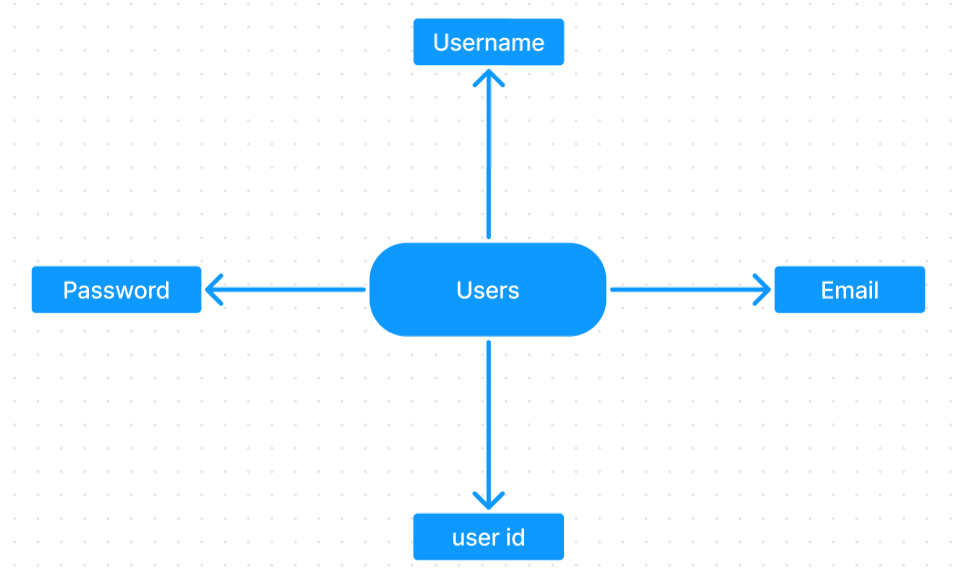
To enhance the user experience, the frontend leverages the Bootstrap and Material UI libraries, creating a real-time and visually appealing interface for users.

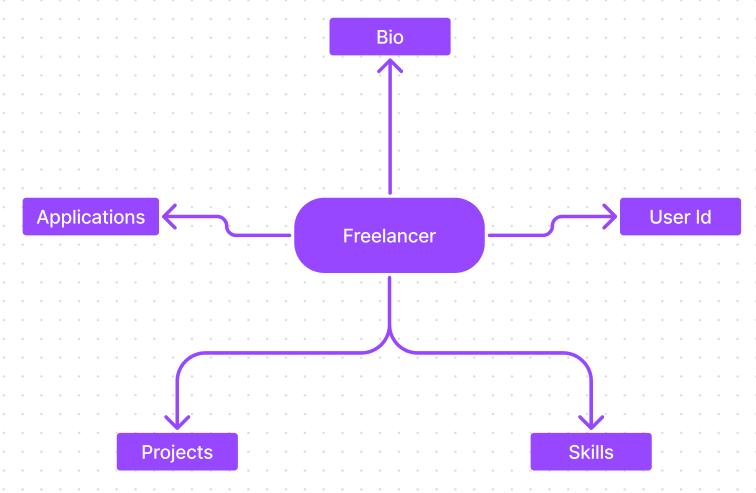
On the backend, we utilize the Express Js framework to manage server-side logic and communication. Express Js provides a robust foundation for handling requests and responses efficiently.

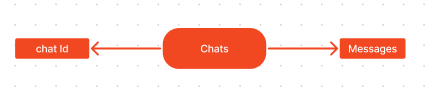
For data storage and retrieval, SB Works relies on MongoDB. MongoDB offers a scalable and efficient solution for storing various data, including user-contributed locations and images. This ensures quick and reliable access to the information needed to enrich the local tourism experience.

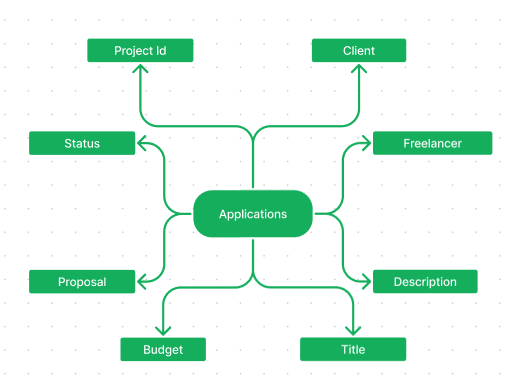
In conjunction, the frontend and backend components, complemented by Express Js, and MongoDB, together form a comprehensive technical architecture for SB Works. This architecture facilitates real-time communication, efficient data exchange, and seamless integration, ensuring a smooth and immersive experience for users contributing to and exploring their local surroundings.

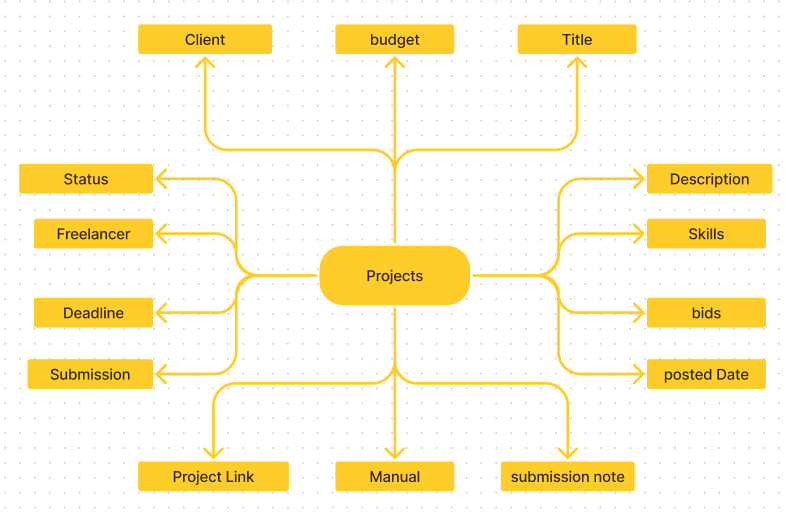
**ER DIAGRAM**

****

****

****

****

****

SB Works connects clients with skilled freelancers through a user-friendly platform. Clients can post projects with details and browse freelancer profiles to find the perfect match. Freelancers can submit proposals, collaborate with clients through secure chat, and securely submit work for review and payment. An admin team ensures quality and communication, making SB Works a go-to platform for both clients and freelancers.

1. **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. 

* **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.

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

* **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

* **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.
* **Development Environment**: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.
* **Installation Guide:**
* Visual Studio Code: Download from <https://code.visualstudio.com/download>
* **Node.js:**

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

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

* **Express.js:**

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

**npm install express**

* **MongoDB:**

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

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

* **React.js:**

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

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

* **Install Dependencies:**
* 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.

**Project setup and configuration.**

* **Folder setup:**

## Now, firstly create the folders for frontend and backend to write the respective code and install the essential libraries.

* + Client folders.
  + Server folders
* **Installation of required tools:**

1. Open the frontend folder to install necessary tools

For frontend, we use:

* + - React
    - Bootstrap
    - Material UI
    - Axios
    - react-bootstrap

2. Open the backend folder to install necessary tools

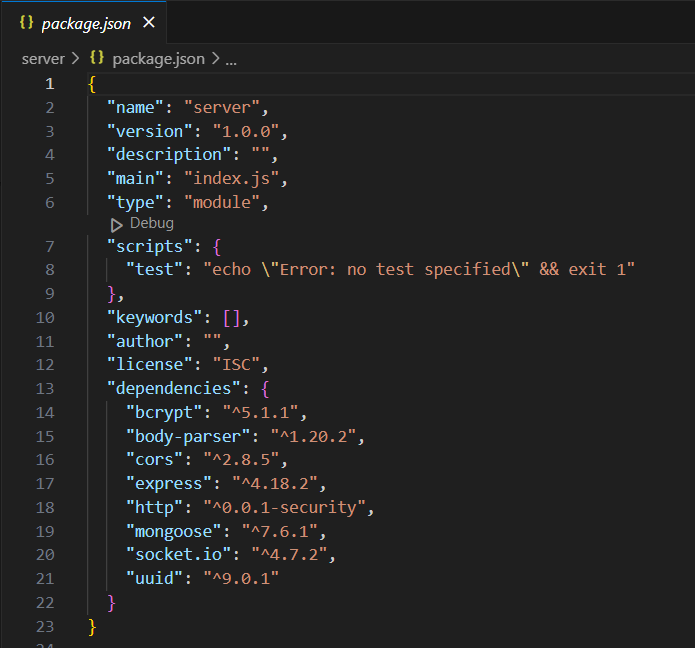
For backend, we use:

* + - Express Js
    - Node JS
    - MongoDB
    - Mongoose
    - Cors
    - Bcrypt

After the installation of all the libraries, the package.json files for the frontend looks like the one mentioned below.



After the installation of all the libraries, the package.json files for the backend looks like the one mentioned below.



**Backend Development**

**1. Project Setup:**

* Create a project directory and initialize it using npm init.
* Install required dependencies like Express.js, Mongoose, body-parser, and cors.

**2. Database Configuration:**

* Set up a MongoDB database (locally or using a cloud service like MongoDB Atlas).
* Create collections for:
* Users (storing user information, account type)
* Projects (project details, budget, skills required)
* Applications (freelancer proposals, rate, portfolio link)
* Chat (communication history for each project)
* Freelancer (extended user details with skills, experience, ratings)

**3. Express.js Server:**

* Create an Express.js server to handle HTTP requests and API endpoints.
* Configure body-parser to parse request bodies and cors for cross-origin requests.

**4. API Routes:**

* Define separate route files for user management, project listing, application handling, chat functionality, and freelancer profiles.
* Implement route handlers using Express.js to interact with the database:
* User routes: registration, login, profile management.
* Project routes: project creation, listing, details retrieval.
* Application routes: submit proposals, view applications.
* Chat routes: send and receive messages within projects.
* Freelancer routes: view and update profiles, showcase skills.

**5. Data Models:**

* Define Mongoose schemas for each data entity:
* User schema
* Project schema
* Application schema
* Chat schema
* Freelancer schema (extends User schema with skills, experience)
* Create Mongoose models to interact with the MongoDB database.
* Implement CRUD operations for each model to manage data.

**6. User Authentication:**

* Implement user authentication using JWT or session-based methods.
* Create routes and middleware for user registration, login, and logout.
* Use authentication middleware to protect routes requiring user authorization (e.g., applying for projects).

**7. Project Management:**

* Allow clients to post projects with details and budget.
* Enable freelancers to browse projects, search by skills, and submit proposals.
* Implement a system for clients to review applications and choose freelancers.

**8. Secure Communication & Collaboration:**

* Integrate a secure chat system within projects for communication between clients and freelancers.
* Allow file attachments and feedback exchange to facilitate collaboration.

**9. Admin Panel (Optional):**

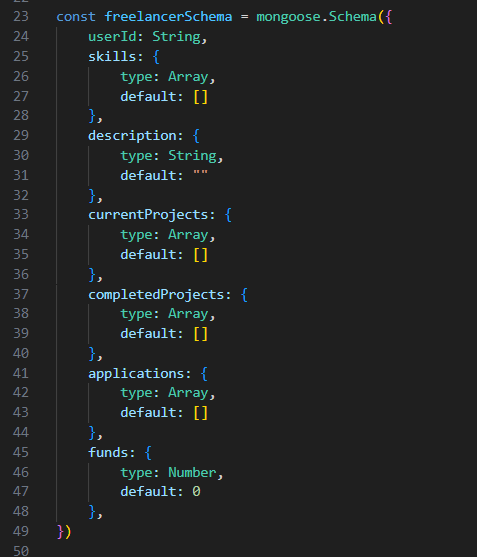
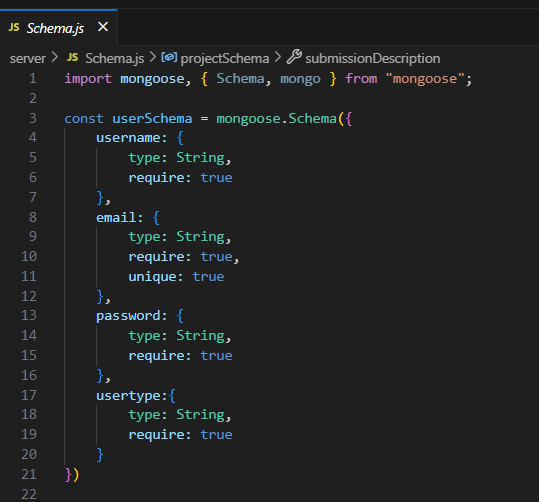
* Implement an admin panel with functionalities like:
* Managing users
* Monitoring project updates and applications
* Accessing transaction history

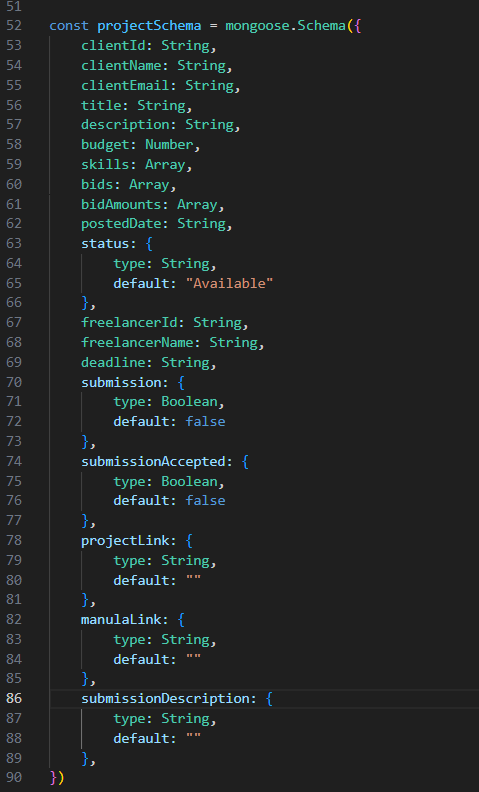
**3: Database development**

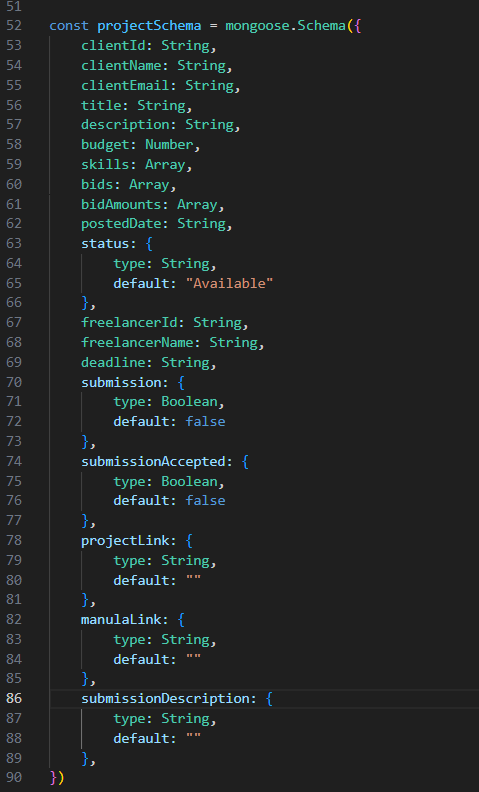
* Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas.
* Create a database and define the necessary collections for users, freelancer, projects, chats, and applications.
* Connect the database to the server with the code provided below.

****

The Schemas for the database are given below

****

****

****

**Frontend development**

**1. Setting the Stage:**

The SB Works frontend thrives on React.js. To get started, we'll:

* Create the initial React application structure.
* Install essential libraries for enhanced functionality.
* Organize project files for a smooth development experience.
* This solid foundation ensures an efficient workflow as we bring the SB Works interface to life.

**2. Crafting the User Experience:**

Next, we'll focus on the user interface (UI). This involves:

* Designing reusable UI components like buttons, forms, and project cards.
* Defining the layout and styling for a visually appealing and consistent interface.
* Implementing navigation elements for intuitive movement between features.
* These steps will create a user-friendly experience for both freelancers and clients.

**3. Bridging the Gap:**

The final stage connects the visual interface with the backend data. We'll:

* Integrate the frontend with SB Works' API endpoints.
* Implement data binding to ensure dynamic updates between user interactions and the displayed information.

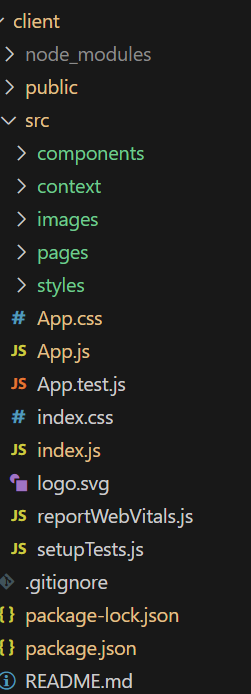
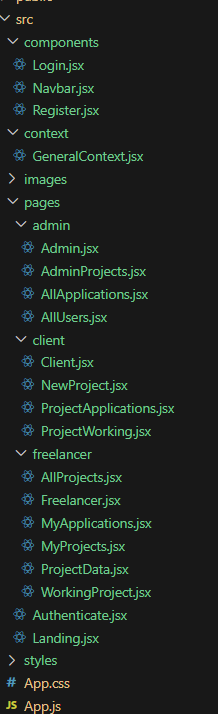
This completes the frontend development, bringing the SB Works platform to life for users.

1. **Folder Structure:**

* **Client:**

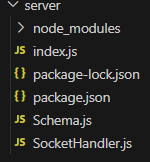
SB Works leverages React.js for the user interface. The client-side code likely consists of reusable components for profiles, projects, and chat, assembled into pages like project browsing or freelancer profiles. Shared data like user info or search filters might be managed with React Context.

* **Client Folder:**

****

* **Server:**

. On the server side, Node.js handles API requests for user management, project actions, and communication. Mongoose models ensure structured interaction with the MongoDB database. This breakdown provides a foundational understanding of SB Works' architecture.

****

1. **Running the Application:**

Provide commands to start the frontend and backend servers locally.

* **Frontend:** npm start in the client directory.
* **Backend:** npm start in the server directory.

1. **API Documentation:**

### **User Authentication**

The /register (**POST**) endpoint allows new users to sign up. The request body must include username, email, password, and usertype. If the user is a freelancer, a new freelancer profile is also created. On success, it returns the created user as JSON.  
The /login (**POST**) endpoint enables existing users to log in. It accepts email and password in the body. On successful authentication, the user object is returned. Otherwise, an error message such as “User does not exist” or “Invalid credentials” is returned.

* **Freelancer Management**

The /fetch-freelancer/:id (GET) endpoint returns the freelancer profile associated with the given userId as the Url parameter.

The /update-freelancer (POST) endpoint updates freelancer information. The request body must include freelancerId, updateSkills, and description. It parses the updateSkills string into an array and updates the freelancer record, then returns the updated profile as JSON.

* **Project Management**

The /fetch-project/:id (GET) endpoint retrieves a single project’s details by its MongoDB \_id.

The /fetch-projects (GET) endpoint returns an array of all projects available in the database.

The /new-project (POST) endpoint allows clients to post a new project. It requires title, description, budget, skills (as a comma-separated string), and client information (clientId, clientName, clientEmail). The project is then saved and returns a confirmation message

* **Bidding and Applications**

The /make-bid (POST) endpoint enables a freelancer to bid on a project. The request body must contain clientId, freelancerId, projectId, proposal, bidAmount, and estimatedTime. This endpoint adds a new Application record and updates the project's bids. On success, it returns a confirmation message.

The /fetch-applications (GET) endpoint returns all applications.

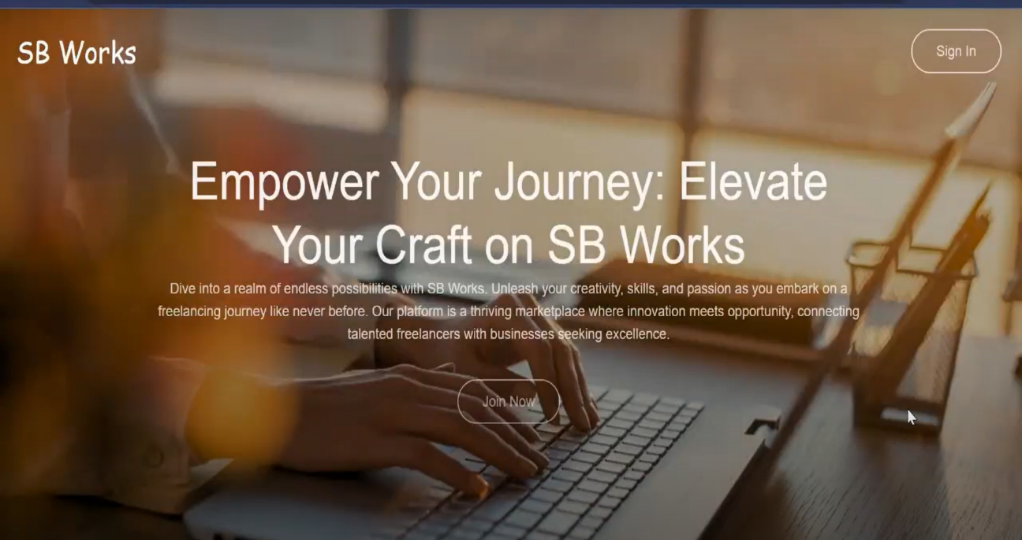
The /approve-application/:id (GET) and /reject-application/:id (GET) endpoints allow the admin/client to accept or reject a freelancer's application using the application’s :id. Approving an application updates the project's status to "Assigned", sets the freelancer for the project, and moves all other pending applications to "Rejected". Rejected applications simply change the application status to "Rejected".

1. **Authentication:**

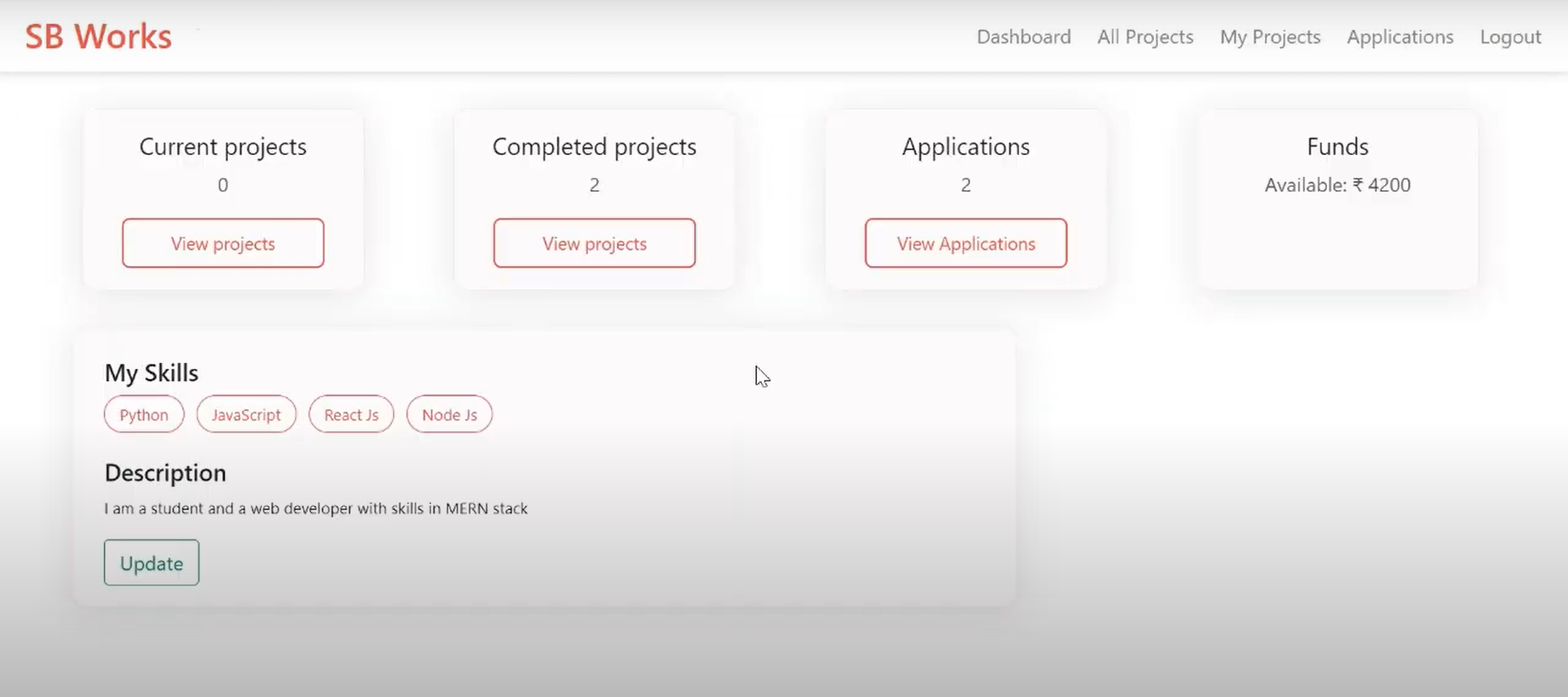
* In the FreelanceFinder application, authentication is handled using a **standard login and registration process** powered by **Express.js** and **bcrypt** for password hashing. When a new user signs up through the /register endpoint, their plain-text password is securely salted and hashed before being saved to the MongoDB database. This ensures that no passwords are stored in plain text, which enhances the overall security of the application.
* When a user attempts to log in via the /login endpoint, the provided credentials are validated against the corresponding record in the database. The hashed password is compared to the one saved in MongoDB using bcrypt.compare. Upon successful validation, the backend returns the user’s data to the client-side application. The client (React frontend) then stores this data — specifically the userId and usertype — in **local storage** to maintain the logged-in state across page reloads and to personalize the interface for each role. Unlike traditional server sessions or JWT tokens, this implementation keeps the authentication lightweight and relies on the client to persist session information.
* While there are no server-side sessions or JSON Web Tokens (JWTs) involved in this version of the application, **authorization** is implemented based on the usertype field. Throughout the frontend, conditional rendering is used to control access to different routes and features — for instance, only freelancers can view and submit bids on projects, while clients can post new projects or review submissions. Backend routes also verify the current user’s role before allowing updates or approvals. This ensures that only authorized users can perform role-specific actions like submitting a project as a freelancer, or approving an application as a client or admin.

1. **User Interface:**

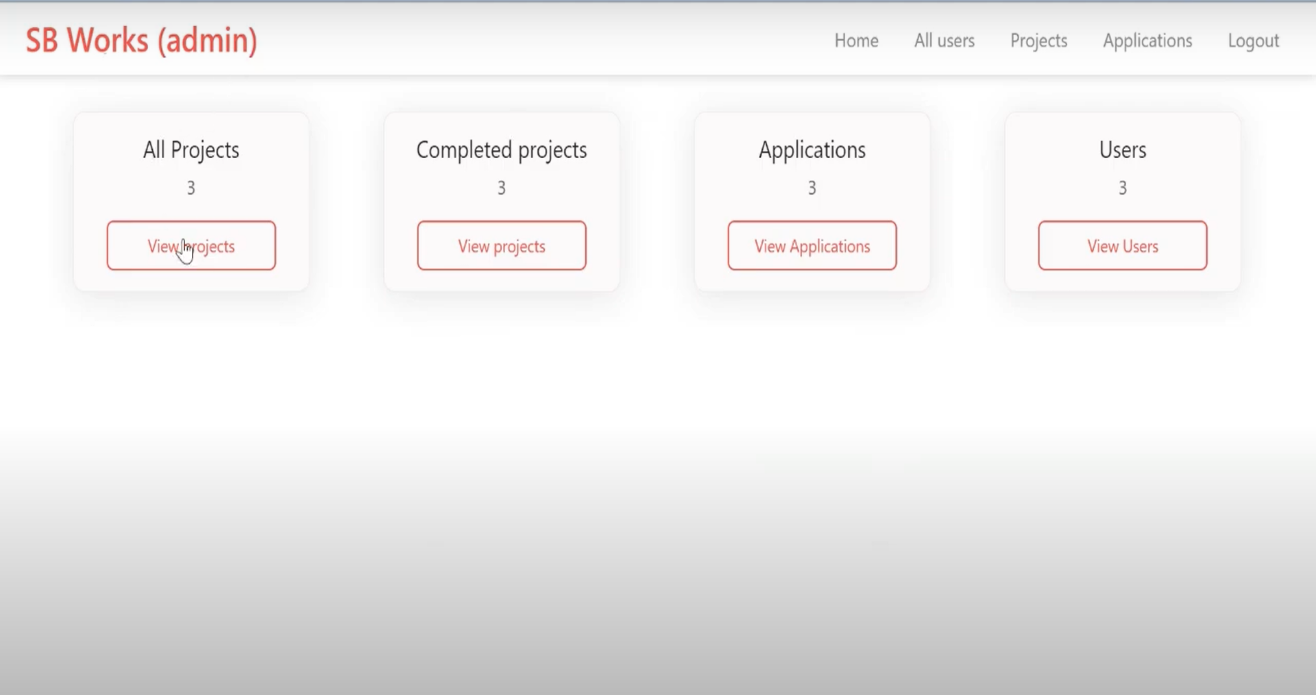
**Landing page:**

****

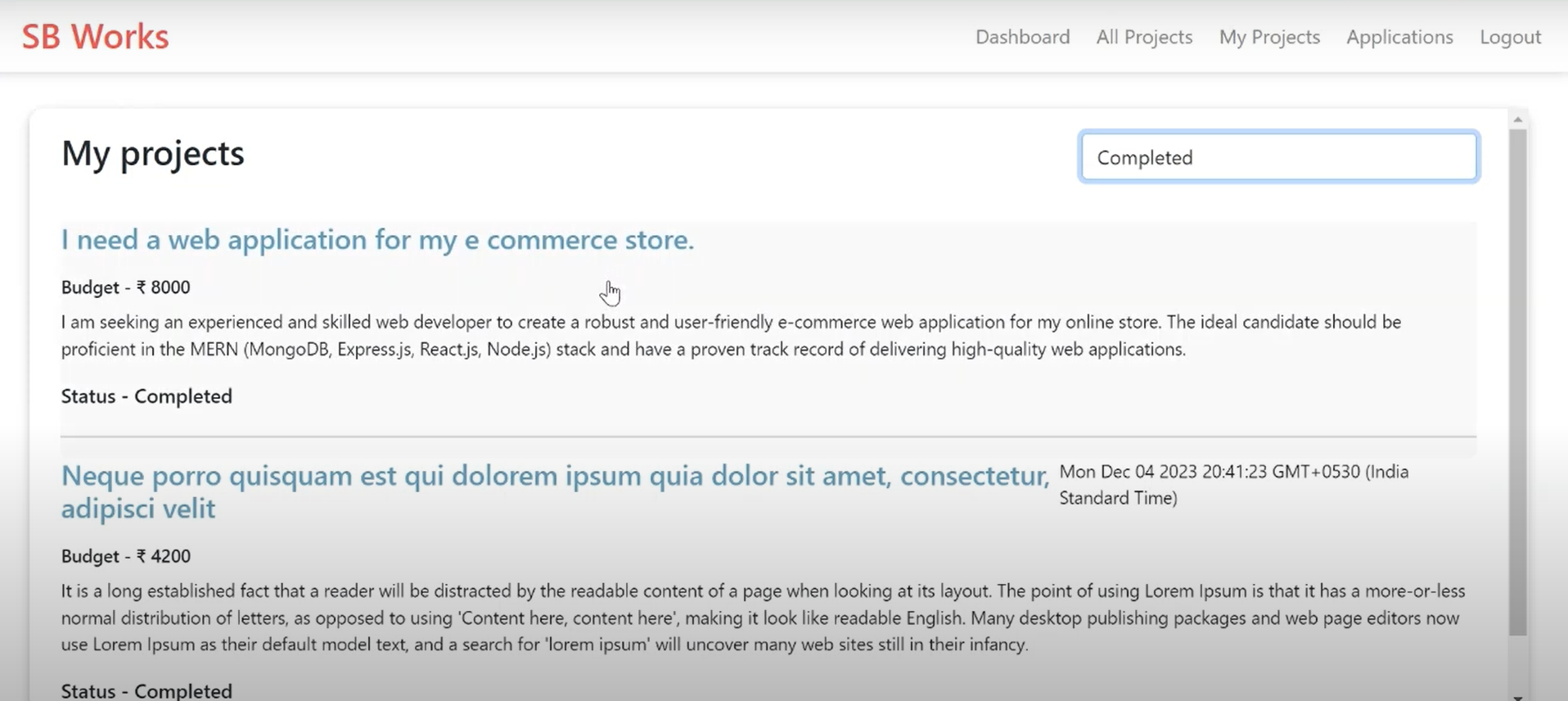
**Freelancer dashboard:**

****

**Admin dashboard:**

****

**Freelance projects:**

****

1. **Testing:**

* **User Acceptance Testing (UAT)**

**Project Overview:**

**Project Name:** Freelance Finder

**Description:** A web-based platform that connects freelancers with clients. It includes profile Project creation, job postings, application tracking, messaging, and payment integration.

**Project Version**: v1.0.0

**Testing Period:** 2025-05-26 to 2025-06-02

**Testing Scope:**

* User registration and login
* Freelancer and client profile creation
* Job posting and job application
* Messaging system
* Payment gateway integration
* Job status updates (e.g., applied, hired, completed)
* Search and filter functionalities
* Review and rating system

**Requirements to be Tested:**

* As a freelancer, I want to apply for jobs easily.
* As a client, I want to post jobs and hire freelancers.
* As a user, I want secure login and payments.

**Testing Environment**

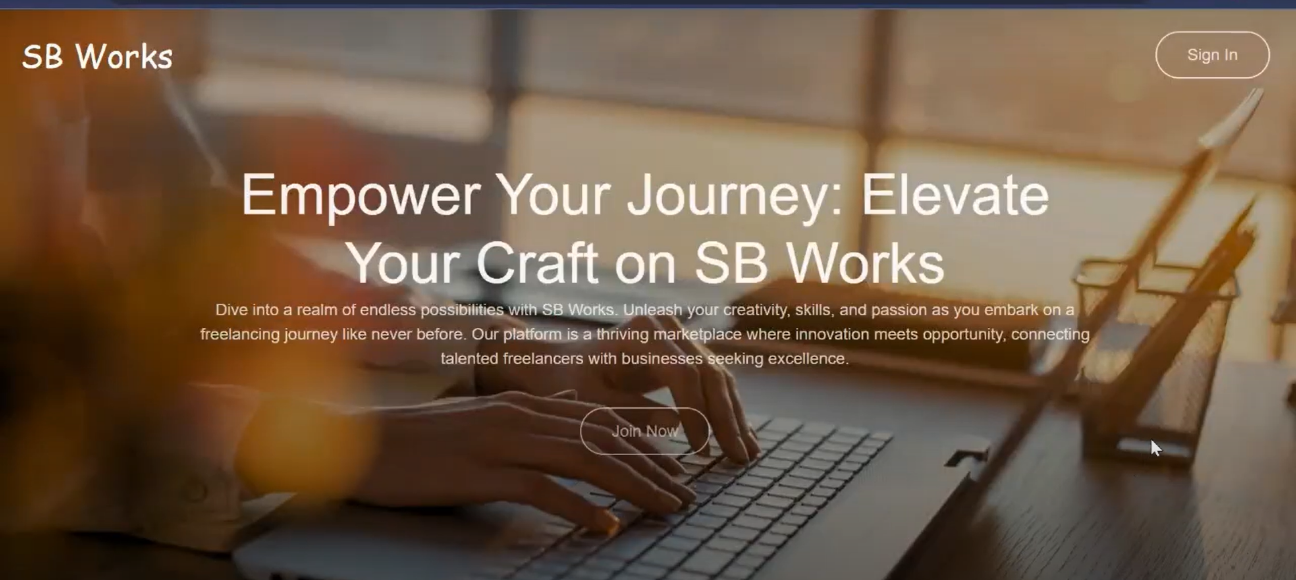
* URL: <https://freelancefinder.example.com>

**Credentials:**

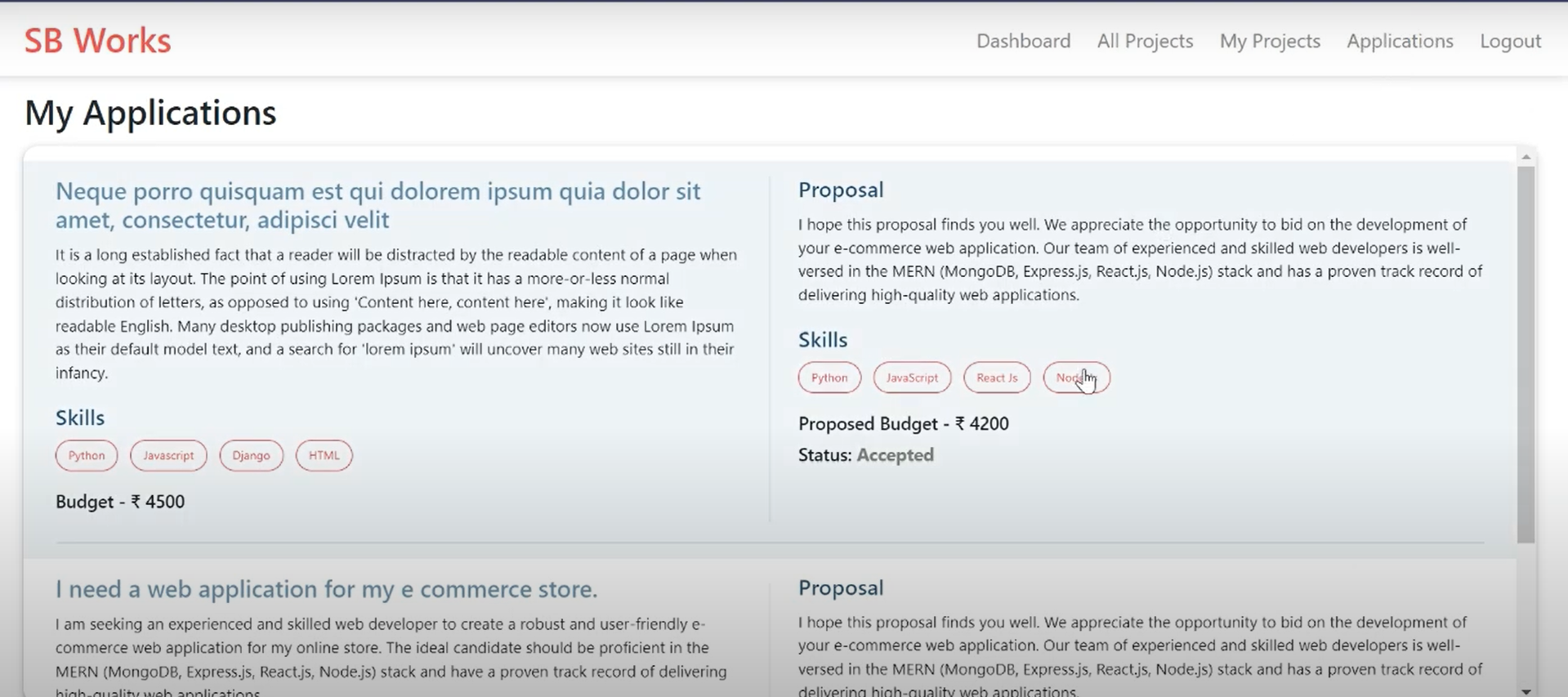
* Freelancer: test.freelancer@example.com / password123
* Client: test.client@example.com / password123
* Admin: admin@example.com / adminpass

1. **Screenshots or Demo:**

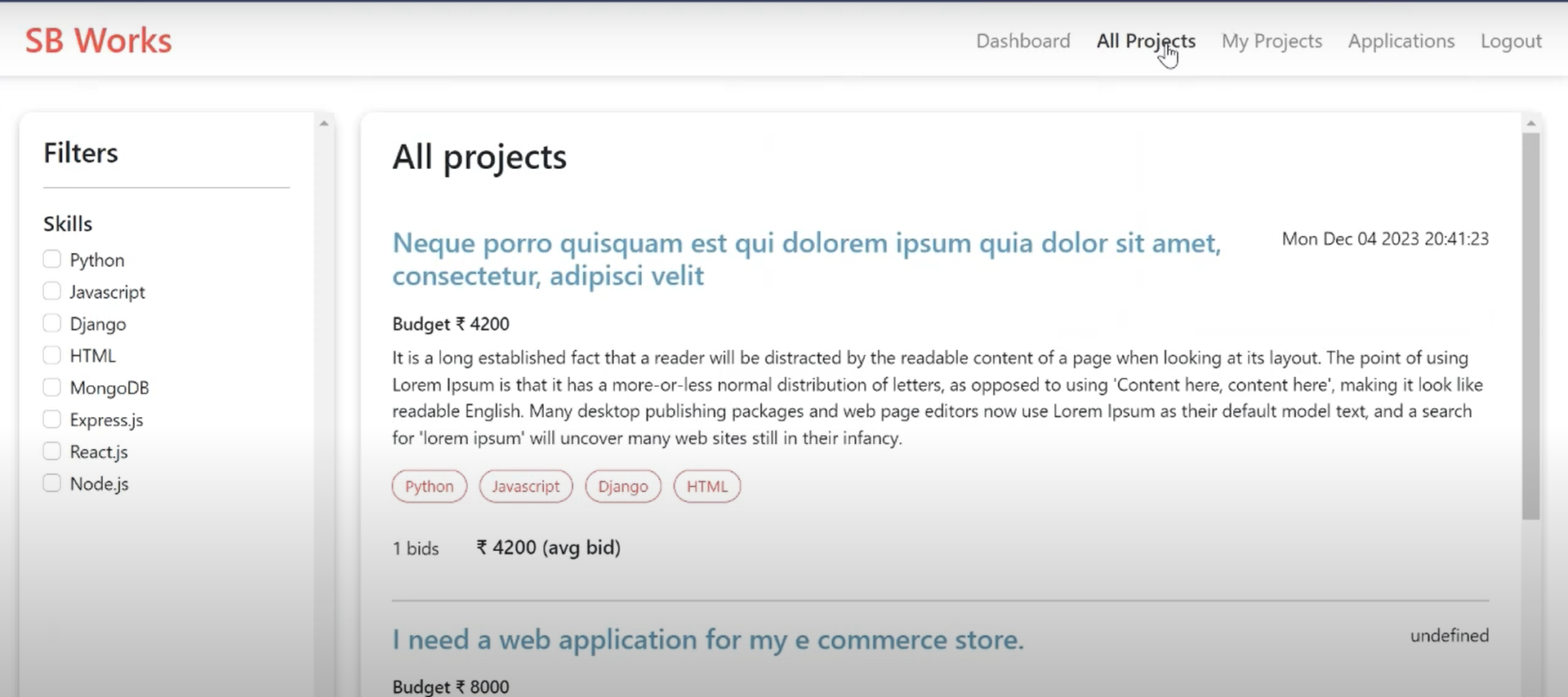
**Landing page:**



**Applications:**

****

**All projects:**

****

**Video Link:**

<https://drive.google.com/drive/folders/1zRt6hjr7MZzgijKNAUJMJ_Q7hfWsV3EH>

1. **Known Issues:**

* **Bug Tracking:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Bug ID** | **Bug Description** | **Steps to reproduce** | **Severity** | **Status** | **Additional feedback** |
| BG-001 | |  | | --- | | Error on job posting form |  |  | | --- | |  | | 1. Login as client  2. Submit empty form | Medium | Open | Form should show validation |

1. **Future Enhancements:**

* The FreelanceFinder application provides a strong foundation for connecting clients and freelancers, offering features like real-time chat, bidding, project tracking, and user role management. However, to further enhance functionality, scalability, and user experience, several future improvements can be introduced.
* One major enhancement would be the **integration of token-based authentication using JWT (JSON Web Tokens)**. Currently, user sessions are handled via browser local storage, which is suitable for basic use but lacks secure token expiration and refresh logic. Implementing JWT would improve both **security and scalability**, especially when expanding to a production-grade deployment or mobile compatibility.
* In addition, introducing **email verification and password reset functionality** via nodemailer or third-party services like SendGrid would make the registration process more secure and professional. This helps prevent spam accounts and provides a better user recovery experience.
* On the user interface side, the application could be enhanced with **file upload support** for portfolios, resumes, and work samples. Freelancers could upload sample work directly to their profile, while clients could attach reference files to project postings. Integrating cloud storage services like Cloudinary or Firebase Storage could make this scalable.
* Finally, building an **admin dashboard with analytics** (such as active users, most active freelancers, revenue tracking, etc.) would make platform management more insightful and data-driven. This could be further extended with monthly reports and downloadable summaries.
* Together, these enhancements would transition FreelanceFinder from a functional demo to a fully professional freelancing platform, offering richer features and stronger security for real-world usage.