Full Stack Development with MERN Project Documentation format

1. Introduction

• **Project Title:** [Your Project Title]

• **Team Members:** List team members and their roles.

2. Project Overview

• **Purpose:** Briefly describe the purpose and goals of the project. • **Features:** Highlight key features and functionalities.

3. Architecture

• **Frontend:** Describe the frontend architecture using React.

• **Backend:** Outline the backend architecture using Node.js and Express.js. • **Database:** Detail the database schema and interactions with MongoDB.

4. Setup Instructions

• **Prerequisites:** List software dependencies (e.g., Node.js, MongoDB). • **Installation:** Step-by-step guide to clone, install dependencies, and set up the environment variables.

5. Folder Structure

• **Client:** Describe the structure of the React frontend.

• **Server:** Explain the organization of the Node.js backend.

6. Running the Application

• Provide commands to start the frontend and backend servers locally.

o **Frontend:** npm start in the client directory.

o **Backend:** npm start in the server directory.

7. API Documentation

• Document all endpoints exposed by the backend.

• Include request methods, parameters, and example responses.

8. Authentication

• Explain how authentication and authorization are handled in the project. • Include details about tokens, sessions, or any other methods used.

9. User Interface

• Provide screenshots or GIFs showcasing different UI features.

10. Testing

• Describe the testing strategy and tools used.

11. Screenshots or Demo

• Provide screenshots or a link to a demo to showcase the application.

12. Known Issues

• Document any known bugs or issues that users or developers should be aware of.

13. Future Enhancements

• Outline potential future features or improvements that could be made to the project.

FreelanceFinder: Discovering Opportunities, Unlocking Potential

1. Introduction

Project Title: SB Works – A Freelancing Platform

Team Leader : Nekkanti Durga Prasad

Team member : Mani Shankar Avinash Kilaparthi

Team member : Venkata Vinodh Kondaveeti

Team member : Vikash Kumar

2. Project Overview :

Purpose of the Project

SB Works is developed to create a secure, scalable, and efficient freelancing ecosystem where clients and freelancers can collaborate seamlessly. The platform aims to eliminate communication gaps, improve transparency in project management, and provide structured workflows for project execution.

In today's digital economy, freelancing platforms are essential for connecting businesses with skilled professionals. However, many platforms suffer from trust issues, unclear communication, and inefficient management. SB Works addresses these issues by implementing a structured bidding system, role-based access control, secure authentication, and integrated communication modules.

The goal of the project is to demonstrate the implementation of a full-stack web application using the MERN stack while solving a real-world problem.

Features

SB Works includes the following major features:

- User Registration and Login with secure authentication
- Role-based access control (Client, Freelancer, Admin)
- Project Posting Module for clients
- Project Browsing and Bidding Module for freelancers
- Integrated Chat System for communication
- Project Submission and Feedback Module
- Rating and Review System
- Admin Monitoring and User Management Dashboard
- Responsive and user-friendly interface

3. Architecture

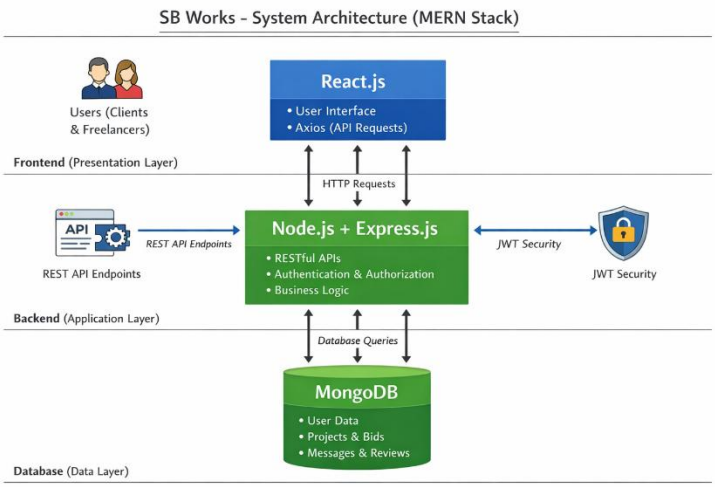

Fig 1 :- Main Architecture

3.1 Frontend Architecture

The frontend of SB Works is developed using React.js, a modern JavaScript library for building dynamic and responsive user interfaces. React follows a component-based architecture, which allows reusable UI components and improves maintainability**.**

The frontend is responsible for:

- Rendering user interfaces dynamically
- Handling user input and form validation
- Managing client-side routing using React Router
- Communicating with backend APIs using Axios
- Displaying project listings, bids, and dashboards

Key Features of Frontend:

- Component-based structure for modularity
- React Hooks for state management
- Conditional rendering based on user roles
- Responsive design using Bootstrap and Material UI
- Secure storage of authentication tokens

The frontend ensures a seamless and interactive experience for clients, freelancers, and administrators.

3.2 Backend Architecture (Node.js & Express.js)

The backend of SB Works is developed using Node.js with the Express.js framework. It serves as the core processing unit of the application, handling business logic, authentication, API routing, and communication with the database.

The backend follows a structured RESTful API architecture, ensuring clear separation of concerns and modular development.

Core Responsibilities of Backend

- Handling HTTP requests and responses
- Implementing business logic for projects and bidding
- Managing authentication and authorization
- Validating user input and data
- Interacting with MongoDB database
- Ensuring secure and protected routes

3.3 Database Architecture (MongoDB)

The database layer of SB Works is implemented using MongoDB, a NoSQL document-oriented database. MongoDB was chosen because of its flexibility, scalability, and ability to handle dynamic and structured data efficiently.

Unlike traditional relational databases, MongoDB stores data in JSON-like documents (BSON format), making it highly suitable for modern web applications built using JavaScript technologies like Node.js.

Why MongoDB Was Selected

- Schema flexibility for dynamic data
- High scalability
- Fast read and write operations
- Seamless integration with Node.js
- Efficient handling of nested objects

MongoDB supports horizontal scaling, which makes SB Works future-ready for large-scale usage.

4. Setup Instructions

• Prerequisites:

To develop a full-stack web application like SB Works using React.js, Node.js, Express.js, and MongoDB, several software tools and technologies must be installed and configured in the development environment.

Below are the essential prerequisites required for development:

Node.js and npm

Node.js is required to run JavaScript on the server side. It includes npm (Node Package Manager), which is used to install project dependencies.

- Download: https://nodejs.org/en/download/
- Installation Instructions: https://nodejs.org/en/download/package-manager/

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

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

• Installation instructions:https://docs.mongodb.com/manual/installation/

MongoDB

MongoDB is used as the database for storing user information, projects, bids, messages, and reviews.

You can either:

Install MongoDB locally

OR

Use MongoDB Atlas (cloud database service)

Download MongoDB Community Edition:

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

Installation Instructions:

https://docs.mongodb.com/manual/installation/

After installation, ensure MongoDB service is running.

Express.js

Express.js is a web application framework for Node.js. It is used to handle server-side routing, middleware, and API development.

To install Express.js in your project:

npm install express

Express simplifies backend development by managing HTTP requests, routing, and middleware handling.

React.js

React.js is a JavaScript library used to build the client-side interface of the application. It enables the creation of Single Page Applications (SPA).

Getting Started with React

You can create a React application using Vite (recommended modern approach).

Quick Start:

npm create vite@latest

cd my-app

npm install

npm run dev

If you've previously installed create-react-app globally via npm install -g create-react-app, we recommend you uninstall the package using npm uninstall -g create-react-app or yarn global remove create-react-app to ensure that npx always uses the latest version.

Create a new React project:

• Choose or create a directory where you want to set up your React project.

• Open your terminal or command prompt.

• Navigate to the selected directory using the cd command.

• Create a new React project by running the following command: npx create-react-app your-app-name.Wait for the project to be created:

• This command will generate the basic project structure and install the necessary dependencies

Navigate into the project directory:

• After the project creation is complete, navigate into the project directory by running the following command: cd your-app-name

Start the development server:

• To launch the development server and see your React app in the browser, run the following command: npm run dev

• The npm start will compile your app and start the development server.

• Open your web browser and navigate to https://localhost:5173 to see your React app.

You have successfully set up React on your machine and created a new React project. You can now start building your app by modifying the generated project files in the src directory.

Please note that these instructions provide a basic setup for React. You can explore more ad- vanced configurations and features by referring to the official React documentation: https://react.dev/

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 Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Library: Utilize React to build the user-facing part of the application, including products listings, booking forms, and user interfaces for the admin dashboard.

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

• Sublime Text: Download from

https://www.sublimetext.com/download

• WebStorm: Download from https://www.jetbrains.com/webstorm/download

5. Folder Structure

Frontend:

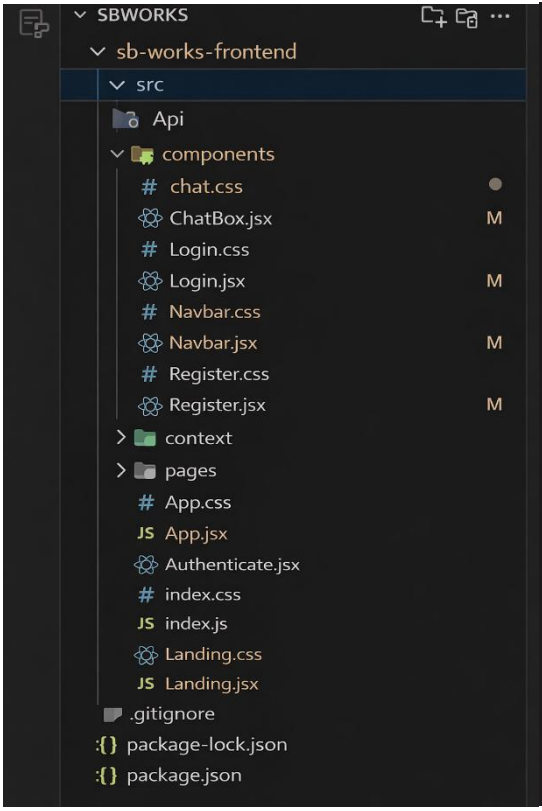

Fig 2 :- Frontend Structure

Backend:

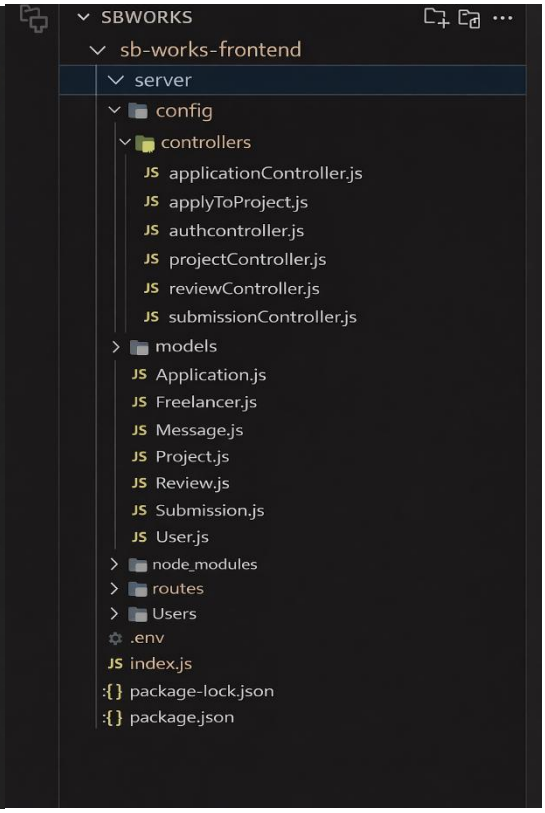

Fig 3 :- Backend Structure

Frontend (React Client) – Structure

The frontend of SB Works is developed using React.js and follows a modular, component-based architecture. This architecture ensures reusability, maintainability, and scalability of the application.

The frontend is structured into organized folders such as pages, components, context, api, and routes, each serving a specific purpose in the application.

The pages folder contains the main screens of the application, including:

- Login Page
- Registration Page
- Client Dashboard
- Freelancer Dashboard
- Admin Panel
- Project Listing Page

These pages are logically separated based on user roles.

Backend (Node.js Server) – Structure

The backend of SB Works is built using Node.js and Express.js, following a modular MVC (Model-View-Controller) architecture.

The backend is organized into folders such as:

- models

- routes
- controllers
- middleware
- config
- utils

Each folder plays a specific role in maintaining structured development.

The models folder defines MongoDB schemas using Mongoose for:

- Users
- Projects
- Applications
- Submissions
- Reviews
- Messages

The routes folder handles API endpoint definitions and maps them to their corresponding controller functions. All routes follow RESTful conventions.

6. Running the Application

To run the application successfully without any errors , we need to run the both backend and frontned at the same time , both should be in the running state simulatanously. To run these both , we need the commands , the commands are :

For Backend : First navigate into the folder using the command " cd folder_name(backend)

Next , run the command " npm start ".

For Frontend : First navigate into the folder using the command " cd folder_name(frontend)

Next , run the command " npm run dev ".

7. API Documentation

The backend of SB Works provides multiple RESTful API endpoints for handling authentication, project management, bidding, submissions, and reviews.

All APIs follow this base structure:

http://localhost:5000/api/

7.1 Authentication APIs

Register User

Endpoint:

POST /api/auth/register

Description:

Registers a new user (Client or Freelancer).

Request Body:

{
"name": "Nani",
"email": "nani@gmail.com",
"password": "123456",
"role": "freelancer"
}

Response:

{
"message": "User registered successfully",
"token": "jwt_token_here"
}

Login User

Endpoint:

POST /api/auth/login

Description:

Authenticates user and generates JWT token.

Request Body:

{
"email": "nani@gmail.com",
"password": "123456"
}

Response:

{
"message": "Login successful",
"token": "jwt_token_here"
}

7.2 Project APIs

Create Project

Endpoint:

POST /api/projects

Authorization: Required (Client)

Request Body:

```
{
  "title": "Website Design",
  "description": "Need modern UI design",
  "budget": 5000
}
```

Response:

```
{
  "message": "Project created successfully"
}
```

Get All Projects

Endpoint:

GET /api/projects

Description:

Retrieves all available projects.

Get Single Project

Endpoint:

GET /api/projects/:id

Description:

Retrieves details of a specific project.

7.3 Bid APIs

Submit Bid

Endpoint:

POST /api/bids

Authorization: Required (Freelancer)

Request Body:

```
{
  "projectId": "project_id_here",
  "bidAmount": 4500,
  "proposal": "I can complete this in 5 days"
}
```

Get Bids for Project

Endpoint:

GET /api/bids/:projectId

Description:

Returns all bids submitted for a specific project.

7.4 Submission APIs

Submit Project Work

Endpoint:

POST /api/submissions

Authorization: Required

Request Body:

```
{
  "projectId": "project_id",
  "submissionLink": "drive_link"
}
```

7.5 Review APIs

Add Review

Endpoint:

POST /api/reviews

Authorization: Required

Request Body:

```
{
  "projectId": "project_id",
  "rating": 5,
  "feedback": "Excellent work!"
}
```

7.6 Admin APIs

Get All Users

GET /api/admin/users

Manage Projects

DELETE /api/admin/project/:id

Admin routes are protected using role-based middleware.

API Security

All protected routes require:

Authorization: Bearer <JWT_TOKEN>

Middleware verifies token validity before granting access.

8. Authentication

🔐 Authentication and Authorization in ShopSmart

The ShopSmart application implements secure authentication and role-based authorization using JSON Web Tokens (JWT). The system ensures that only authenticated users can access protected resources, and only administrators can access admin-specific functionalities.

🔹 Authentication Process

Authentication is handled using a token-based mechanism. When a user logs in successfully using valid email and password credentials, the backend verifies the user details stored in the MongoDB database. If the credentials are valid, the server generates a JSON Web Token (JWT) that contains the user's unique ID and role information.

This token is digitally signed using a secret key and sent back to the client. The frontend stores the token securely (typically in local storage) and includes it in future API requests for protected routes.

Example header format:

Authorization: Bearer <JWT_Token>

The backend middleware verifies this token before granting access to secured endpoints.

🔹 JWT (JSON Web Token) Usage

The JWT contains encoded user information such as:

- User ID
- User Role (user/admin)
- Token expiration time

The token ensures:

- Stateless authentication (no session stored on server)
- Secure request validation
- Reduced database calls for identity verification

Each protected route uses an authentication middleware that:

1. Extracts the token from request headers.
2. Verifies it using the secret key.
3. Decodes the payload.
4. Attaches user information to the request object.

If the token is invalid or expired, access is denied.

- **Authorization (Role-Based Access Control)**

Authorization is implemented using role-based access control (RBAC). Each user in the system has a role field, typically:

- user
- admin

After authentication, an additional middleware checks the user role before allowing access to certain endpoints.

For example:

- Only admins can add, edit, or delete products.
- Only admins can update order status.
- Only logged-in users can add items to cart or place orders.

If a non-admin user attempts to access admin routes, the server returns an unauthorized error response.

Session Handling

The project primarily uses stateless authentication through JWT instead of traditional server-side sessions. This improves scalability and performance, as the server does not need to store session data.

In case social login (OAuth) is implemented, temporary sessions may be used during the authentication handshake process, but the final authentication is handled through JWT tokens.

Security Measures Implemented

- Passwords are hashed before storing in the database.
- JWT tokens are signed using a secure secret key.
- Protected routes verify token authenticity.
- Role-based middleware restricts sensitive operations.
- Token expiration ensures automatic session invalidation.

9. User Interface

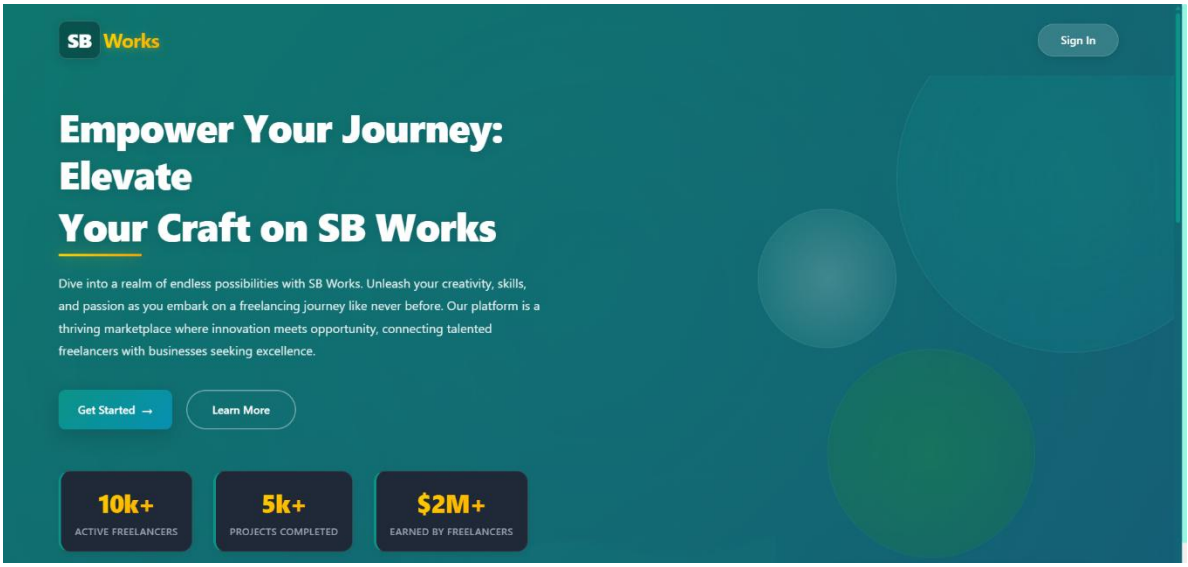

Fig 1 :- Public Home Page

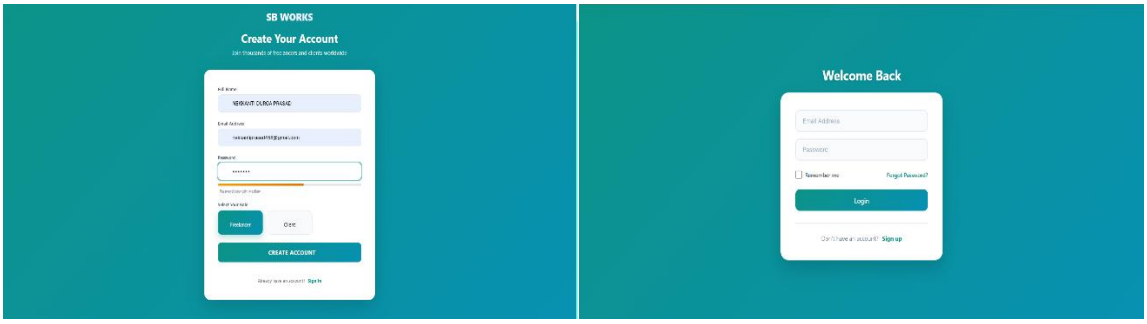

Fig 2 :- Register & Login Pages

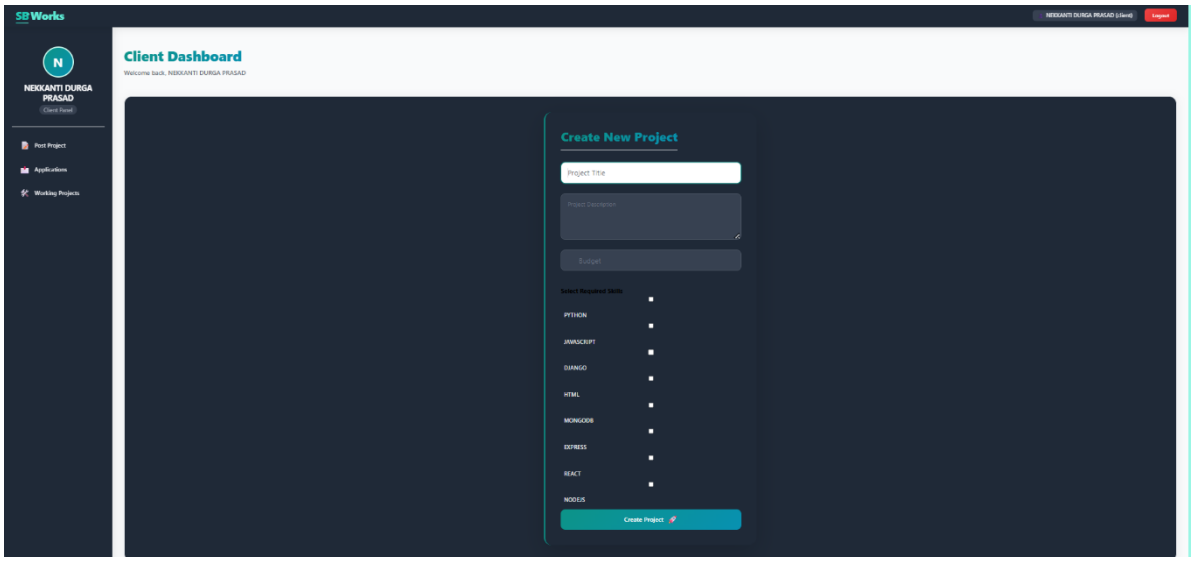

Fig 3 :- Client Home Page

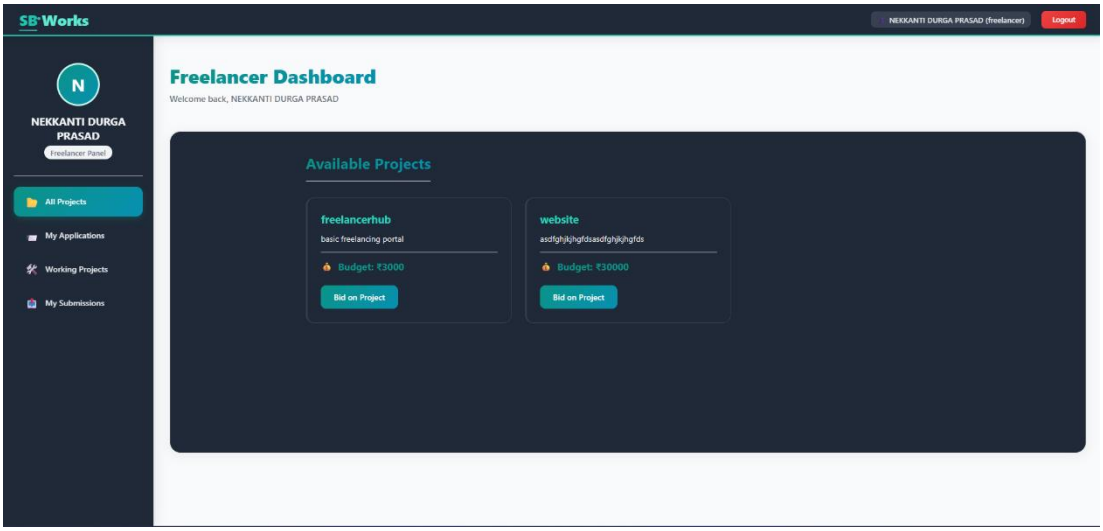

Fig 4 :- Freelancer Page

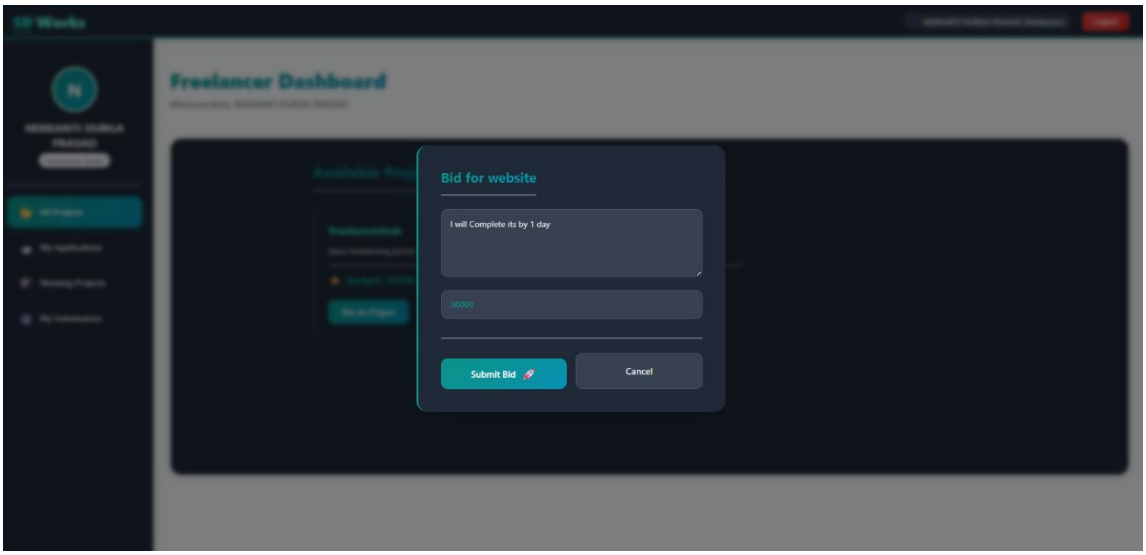

Fig 5 :- Bidding page

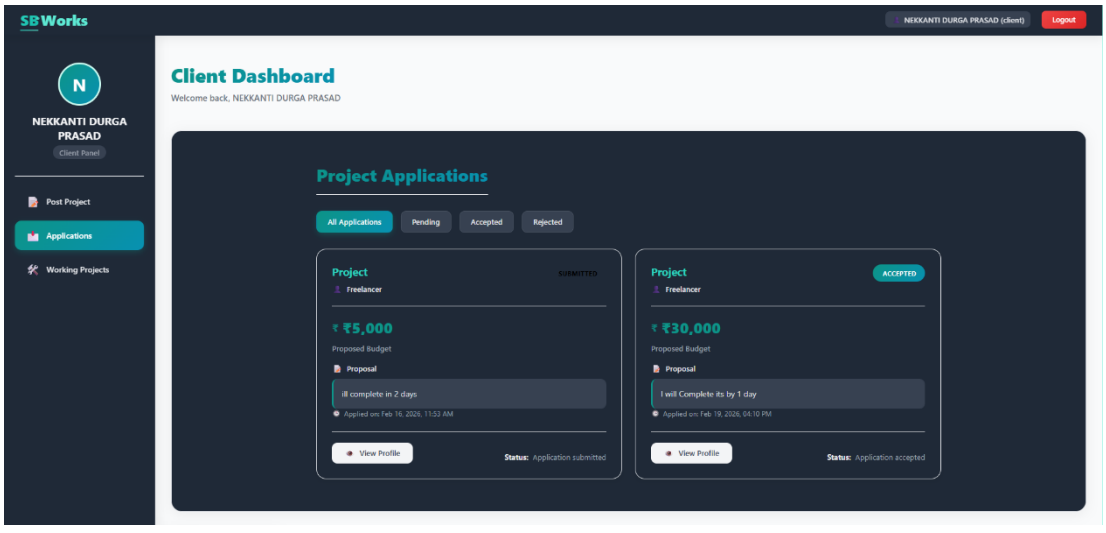

Fig 6 :- Client project Confirmation page

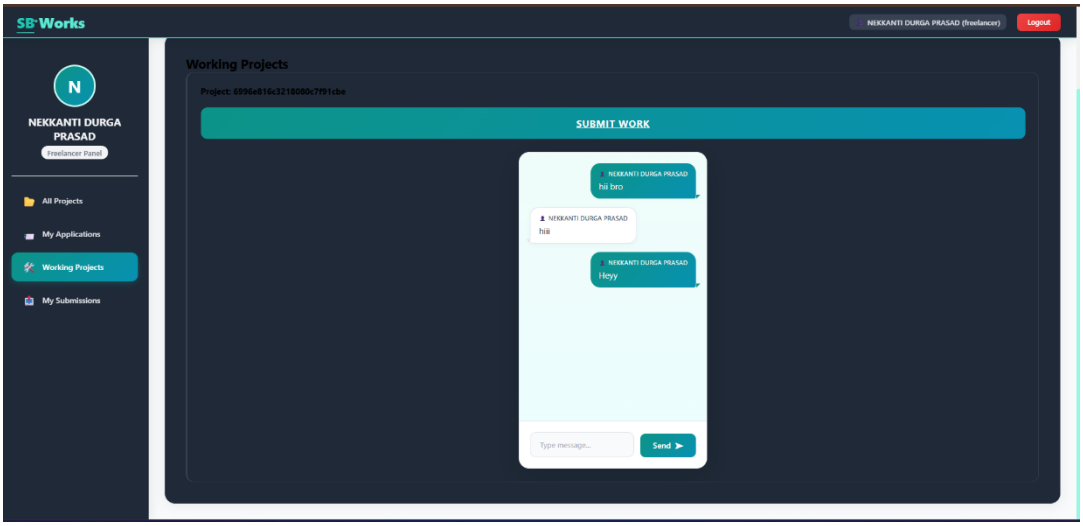

Fig 7 :- Chatting page

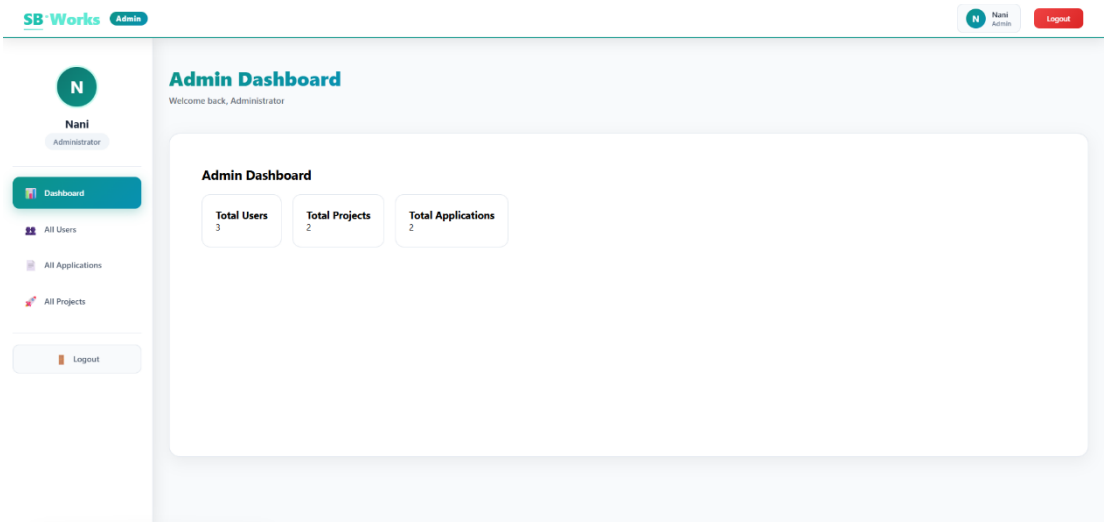

Fig 8 :- Admin Home Page

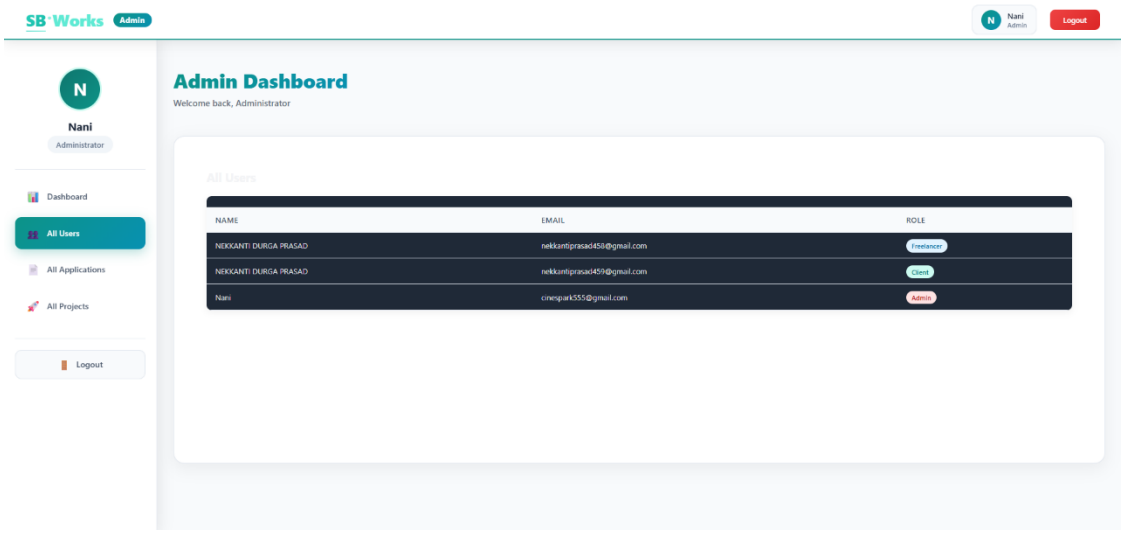

Fig 9 :- Admin Dashboard Page

10. Testing

Testing is a critical phase in the development of SB Works to ensure that all modules function correctly and meet the specified requirements. Both functional and performance testing were conducted.

10.1 Functional Testing

Functional testing was performed to verify that each feature operates as intended.

✅ **Authentication Testing**

- Verified user registration process
- Tested login with valid and invalid credentials
- Checked JWT token generation
- Ensured role-based access control

Result: Authentication module functioned correctly.

✅ **Project Module Testing**

- Tested project creation by client
- Verified project listing display
- Checked project update and status change
- Confirmed database storage

Result: Project module worked as expected.

✅ **Bidding Module Testing**

- Verified freelancer bid submission
- Checked bid retrieval
- Ensured client could view bids

Result: Bidding workflow operated successfully.

✅ **Submission & Review Testing**

- Tested project submission feature
- Verified review and rating system
- Checked feedback storage in database

Result: Submission and review module functioned properly.

✅ **Admin Module Testing**

- Tested user monitoring
- Verified role-based access
- Checked data management functions

Result: Admin panel performed effectively.

🔹 10.2 API Testing

API endpoints were tested using **Postman**.

- Verified correct request and response format
- Checked HTTP status codes
- Validated error handling

All APIs returned appropriate JSON responses.

🔹 10.3 Performance Testing

Performance testing ensured system responsiveness.

- Measured API response time
- Tested multiple user access
- Verified database query efficiency

Result: The system maintained stable performance under normal load conditions.

🔹 10.4 Security Testing

- Verified password hashing
- Checked token verification
- Ensured protected route access
- Validated role-based authorization

Result: System maintained secure data access and authentication control.

Overall, SB Works passed all major testing scenarios and demonstrated stable, secure, and reliable system behavior.

11. Screenshots or Demo

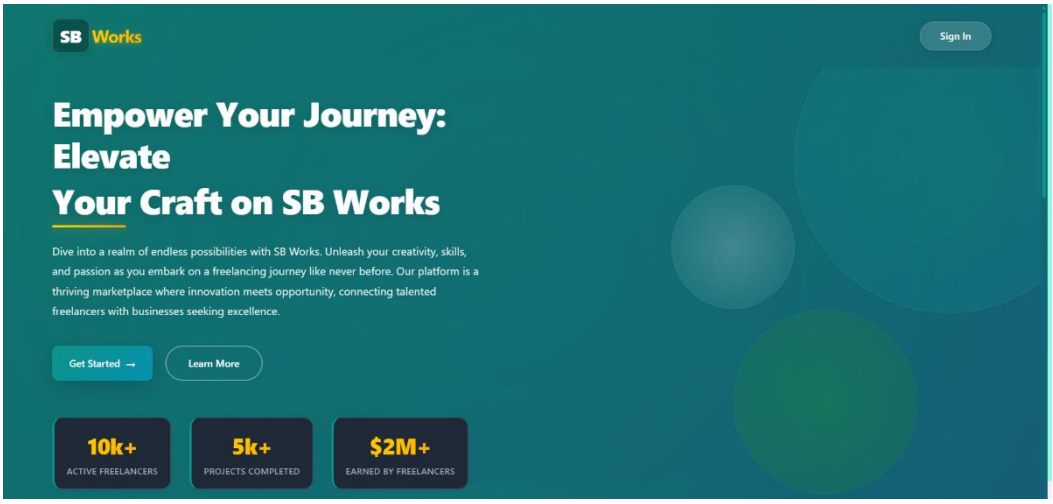

Fig 10 :- Public Home Page.

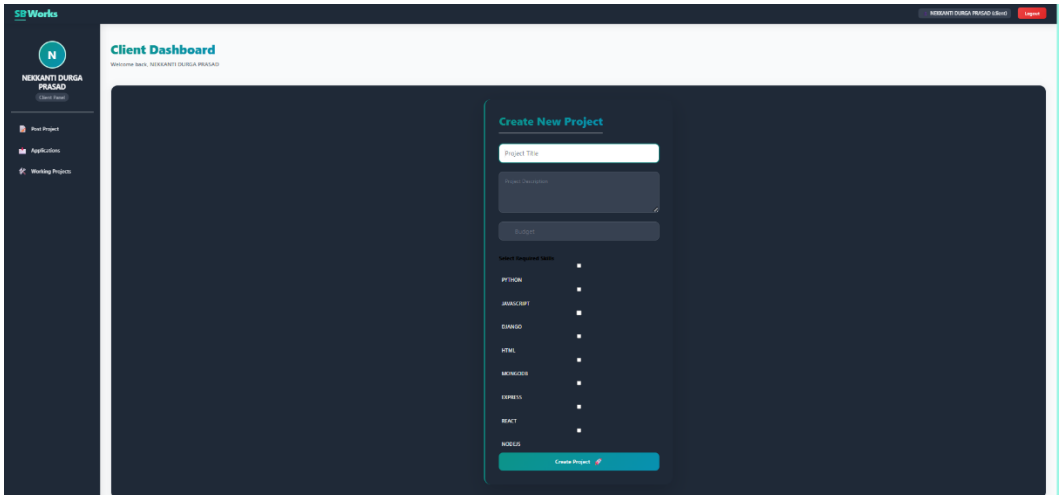

Fig 11 :- Client Submitting Project

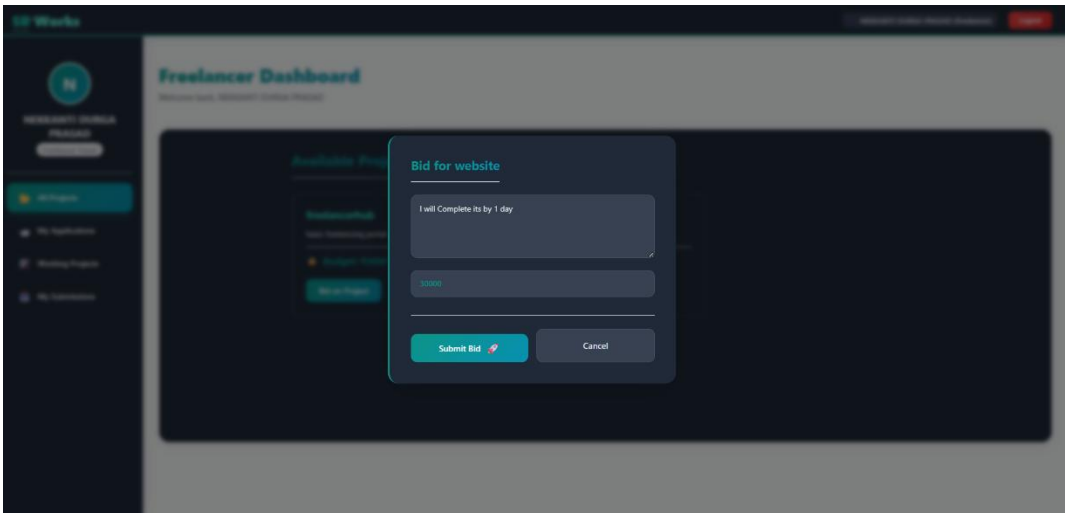

Fig 12 :- Freelancer Bidding for Project

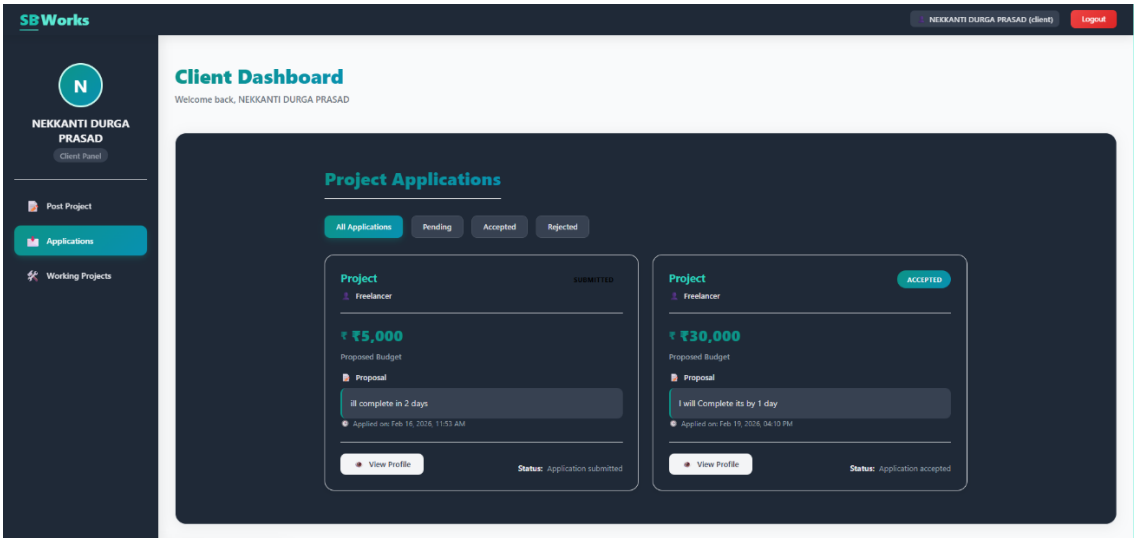

Fig 13 :- Client Approving for bid

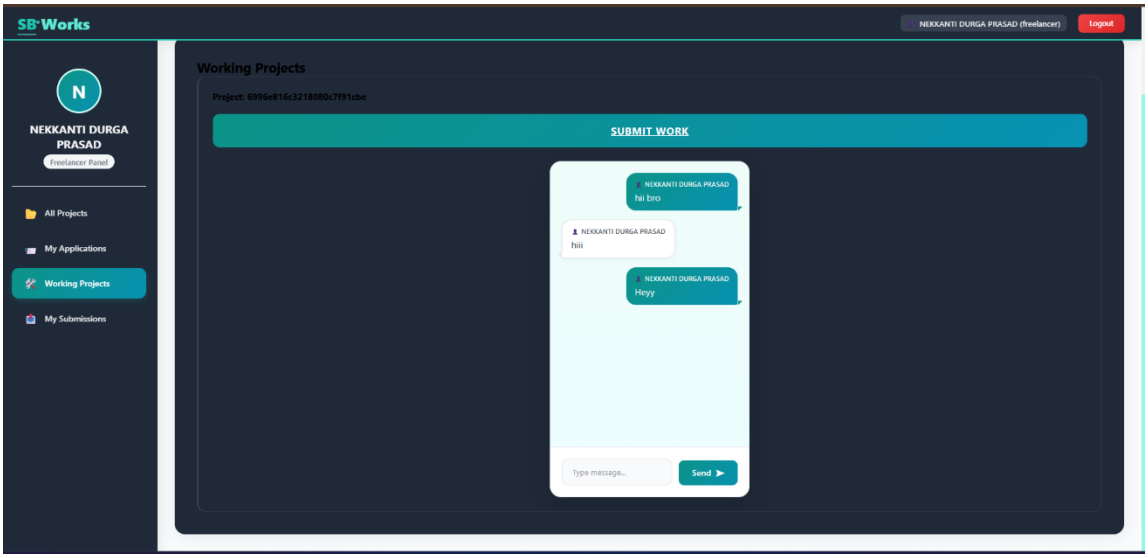

Fig 14 :- Chat module

12. Known Issues

Although SB Works is fully functional and meets its core objectives, there are certain limitations and known issues that can be improved in future updates.

---

1. Payment Gateway Not Integrated

Currently, the platform does not include a real-time payment gateway integration such as Stripe or Razorpay. Payments are assumed to be handled externally.

Future improvement: Implement secure escrow-based payment system.

---

2. Real-Time Chat Enhancement

The current chat functionality is database-based. It does not use WebSockets for real-time communication.

Future improvement: Implement Socket.io for real-time messaging.

3. **Limited Scalability Testing**

The application has been tested under normal usage conditions but has not undergone high-load stress testing.

Future improvement: Perform load testing using tools like JMeter.

---

4. **No Mobile Application**

SB Works currently operates as a web application only.

Future improvement: Develop Android and iOS mobile applications.

---

5. **Basic Analytics**

Admin dashboard includes basic monitoring but lacks advanced analytics and reporting features.

Future improvement: Integrate analytics dashboard with graphical reports.

Despite these limitations, the core functionality of SB Works operates efficiently and securely.

13. Future Enhancements

Although SB Works successfully implements the core functionalities of a freelancing platform, there are several enhancements that can be incorporated in future versions to improve performance, security, and user experience.

---

1. AI-Based Recommendation System

An intelligent recommendation system can be implemented to:

- Suggest suitable freelancers to clients based on skills and ratings
- Recommend relevant projects to freelancers
- Analyze past performance using machine learning

This would improve matching accuracy and reduce manual effort.

---

2. Secure Payment Gateway Integration

Future development can include:

- Integration with payment gateways (Stripe, Razorpay, PayPal)
- Escrow-based payment system
- Automated payment release after approval

This would increase financial security and trust among users.

---

3. Real-Time Chat Using WebSockets

Currently, messaging is database-driven. Implementing **Socket.io** would enable:

- Instant messaging
- Live typing indicators
- Real-time notifications

This would enhance collaboration efficiency.

---

4. Mobile Application Development

Developing Android and iOS applications would:

- Increase accessibility
- Improve user engagement
- Enable push notifications

This would expand platform reach significantly.

---

5. Advanced Admin Analytics Dashboard

Future improvements may include:

- Revenue tracking
- User growth statistics
- Project completion rates
- Graphical reports and insights

This would support data-driven decision-making.

---

6. Two-Factor Authentication (2FA)

Enhancing security with:

- OTP-based login
- Email verification
- Multi-factor authentication

This would further strengthen user data protection.

---

7. Cloud Deployment & Scalability

Deploying on cloud platforms such as:

- AWS
- Azure
- Google Cloud

Would allow auto-scaling and high availability.

---

8. Blockchain-Based Smart Contracts (Advanced Feature)

Incorporating blockchain for secure contract management could ensure:

- Transparent agreements
- Tamper-proof contracts
- Secure payment releases

The Demo of the App is available at :

https://drive.google.com/file/d/15VNxpXmzW4HsSYiBJqAcp6atDuzQD0PS/view?usp=sharing

***** Thank You *****