Freelancing Application MERN

1.INTRODUCTION:

This project is presented by

Team ID : LTVIP2025TMID59364

Team Leader : Muktevi Vybhavi

Team member : Machavarapu Gnana Poojitha

Team member : Maddi Manohar

Team member : Male Ravali

TITLE:

FreelanceFinder: discovering opportunities, unlocking potential

SB Works is a freelancing platform that connects clients with skilled freelancers. It offers an intuitive interface for project posting, bidding, and streamlined collaboration. With a dedicated admin team ensuring security and smooth communication, SB Works aims to be the go-to platform for both clients and freelancers.

2. Project Overview

- **Purpose:**

SB Works, a revolutionary freelancing platform that transforms the way clients connect with skilled freelancers. Our intuitive interface provides clients with the opportunity to post diverse projects, ranging from creative endeavours to technical tasks, while freelancers can seamlessly bid on these projects based on their expertise and capabilities.

At SB Works, we prioritize efficiency and transparency in the freelancing process. Clients can review freelancer profiles, assess past work, and select the perfect candidate for their project. Once a freelancer is chosen, the client can easily communicate and collaborate with them within the platform, streamlining the entire workflow.

Our dedicated admin team ensures the integrity and security of every transaction. With stringent oversight, we guarantee the reliability and quality of the freelancers on our platform. The admin's role is not only to maintain the platform's integrity but also to facilitate smooth communication between clients and freelancers, ensuring a positive and productive working relationship.

Freelancers on SB Works benefit from a straightforward project submission process. After completing the assigned project, freelancers can submit their work directly through the platform, offering clients a hassle-free experience. Clients have the opportunity to review the work and provide feedback, fostering a collaborative environment that values excellence.

Stay informed about the latest projects and industry trends with real-time updates and notifications. SB Works aims to be the go-to platform for clients seeking reliable freelancers and freelancers looking for exciting opportunities to showcase their skills.

Join SB Works today and experience a new era of freelancing where your projects are efficiently managed, your skills are recognized, and collaborations flourish in a secure and dynamic environment.

Features:

- User Profiles: Comprehensive profiles for freelancers to display skills, portfolios, certifications, and work history, and for clients to outline their project requirements.
- Job Posting and Bidding: Clients can post detailed job listings, and freelancers can submit competitive bids with proposals tailored to project needs.
- Advanced Search and Filters: Robust search functionality with filters for skills, budget, location, and experience level to match freelancers with relevant projects.
- Secure Messaging: In-app communication system with real-time messaging and file-sharing capabilities for seamless collaboration.

- Payment Integration: Secure payment processing with escrow functionality to protect both parties until project milestones are met.
- Review and Rating System: Post-project reviews and ratings to build trust and credibility within the platform.
- Dashboard Analytics: Personalized dashboards for freelancers and clients to track ongoing projects, bids, earnings, and performance metrics.
- Notifications: Real-time alerts for new job postings, bid updates, messages, and payment statuses to keep users engaged.

3. Architecture

Frontend:

The frontend of FreelanceFinder is built using React.js, a JavaScript library for creating dynamic and interactive user interfaces. The architecture leverages a component-based approach to ensure modularity, reusability, and maintainability. Key aspects include:

- Component Structure: The frontend is organized into reusable components stored in the client/src/components directory. Core components include Navbar, JobCard, ProfileEditor, and MessageBox, which are composed to build pages like the dashboard, job listings, and user profiles.
- React Router: Utilizes React Router DOM for client-side routing, enabling seamless navigation without full page reloads. Routes are defined in App.js for pages such as /home, /jobs, /profile, and /messages.
- State Management: Employs Redux Toolkit to manage global application state, such as user authentication, job data, and notifications. Redux slices are defined in client/src/redux to handle specific domains (e.g., authSlice, jobSlice).
- Styling: Uses Tailwind CSS for responsive and utility-first styling, supplemented by Material-UI and Bootstrap components for enhanced UI elements like modals, buttons, and forms. Styles are modularized in client/src/styles to maintain consistency.

- API Integration: Axios is used for making HTTP requests to the backend API. API calls are encapsulated in utility functions within client/src/utils/api.js to handle requests for job listings, user data, and messaging.
- Performance Optimization: Implements lazy loading for non-critical components and code splitting via React Router to reduce initial load times. Memoization with React.memo and useCallback prevents unnecessary re-renders.
- Accessibility: Adheres to WCAG 2.1 standards, ensuring proper ARIA labels and keyboard navigation support for components.

Backend

The backend is developed using Node.js and Express.js, providing a robust and scalable RESTful API. The architecture is designed for modularity and efficient request handling, with the following components:

- Framework: Express.js serves as the web application framework, handling routing, middleware, and API endpoints. The entry point is server.js, which initializes the server and connects to MongoDB.
- Routing: Routes are organized by resource in the server/routes directory (e.g., auth.js, jobs.js, messages.js). Each route file defines endpoints like GET /api/jobs or POST /api/bids.
- Controllers: Business logic is encapsulated in controllers (server/controllers), with separate files for authentication, job management, bidding, and messaging. This ensures clean separation of concerns.
- Middleware: Custom middleware in server/middleware handles authentication (JWT validation), input validation, error handling, and request logging. express-validator is used for input sanitization to prevent injection attacks.

- Third-Party Integrations: Integrates with Stripe for payment processing and AWS S3 for file uploads (e.g., portfolio images). These are managed in server/services.
- Scalability: Uses async/await for non-blocking I/O operations and implements rate limiting with express-rate-limit to prevent abuse. The backend is designed to scale horizontally with load balancers like Nginx.

Database

The database is powered by MongoDB, a NoSQL database that stores data in JSON-like documents, offering flexibility and scalability. Mongoose ORM is used for schema definition and data interactions. Key details include:

- Schema Design:
  - Users: Stores user details (username, email, password (hashed with bcrypt), role (freelancer/client), skills, portfolio, location, createdAt).
  - Jobs: Contains job postings (title, description, budget, skills, postedBy (ref to User), deadline, createdAt).
  - Bids: Represents bids on jobs (jobId (ref to Job), freelancerId (ref to User), amount, proposal, status, createdAt).
  - Messages: Stores communication (senderId (ref to User), recipientId (ref to User), content, createdAt).
  - Reviews: Captures feedback (userId (ref to User), rating, comment, createdAt).
  - Transactions: Tracks payments (jobId (ref to Job), freelancerId (ref to User), clientId (ref to User), amount, status, createdAt).

- Indexes: Indexes are created on frequently queried fields like Users.email, Jobs.skills, and Bids.jobId to optimize search and retrieval performance.
- Interactions: Mongoose provides a clean interface for CRUD operations. For example:
  - Creating a job: Job.create({ title, description, budget, postedBy }).
  - Fetching jobs with filters: Job.find({ skills: { $in: skillsArray } }).populate('postedBy').
  - Aggregation for analytics: Used for dashboard metrics like average bid amounts or job completion rates.
- Scalability: MongoDB supports horizontal scaling through sharding. The database is hosted on MongoDB Atlas for managed backups and high availability.
- Data Integrity: Mongoose schemas enforce required fields and data types. Transactions ensure atomicity for critical operations like bid acceptance and payment processing.

4. Setup Instructions

Prerequisites

- Node.js (v18.x+): JavaScript runtime. Download: nodejs.org.
- npm (v8.x+): Included with Node.js for dependency management.
- MongoDB (v5.0+): NoSQL database. Download: mongodb.com or use MongoDB Atlas.
- Git: Version control. Download: git-scm.com.
- Code Editor: Visual Studio Code recommended. Download: code.visualstudio.com.

Installation

The installation process involves setting up the project on a local machine by acquiring the source code, installing dependencies, and configuring the environment. The steps are as follows:

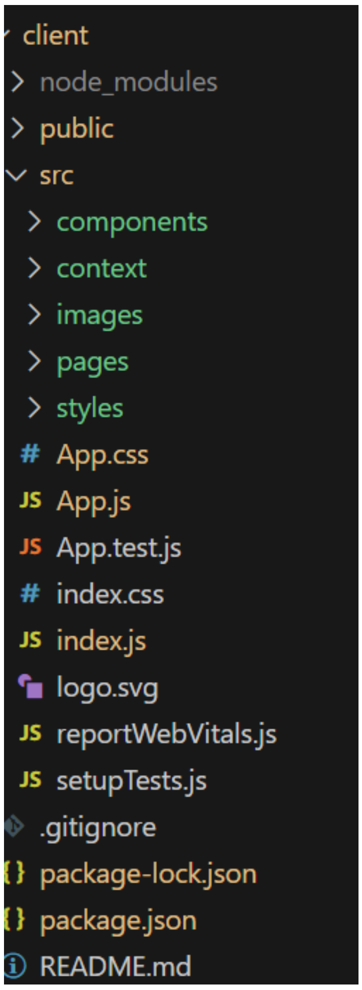
1. Repository Acquisition: The project source code is obtained by cloning the repository from a version control platform like GitHub or downloading it from a provided source, such as a shared drive link. This step ensures access to the complete codebase, including both frontend and backend components.
2. Directory Navigation: Navigate to the project's root directory, which contains separate folders for the client (frontend) and server (backend). This structure organizes the codebase for clarity and modularity.
3. Frontend Dependency Installation: Within the client directory, dependencies for the React-based frontend are installed using npm. These include libraries like React, Redux, and Tailwind CSS, which are critical for building the user interface and managing application state.
4. Backend Dependency Installation: In the server directory, dependencies for the Node.js and Express.js backend are installed via npm. This includes packages like Mongoose for MongoDB interaction and other utilities for API development and authentication.
5. Environment Configuration: Environment variables are configured to ensure the application connects to the database and external services securely. A .env file is created in the server directory to store sensitive information, such as:
    - The MongoDB connection string (local or cloud-based).
    - A secret key for JSON Web Tokens (JWT) to secure authentication.
    - A port number for the backend server.
    - API keys for third-party services, such as payment gateways (e.g., Stripe).
6. Database Setup: The MongoDB database must be operational, either by running a local MongoDB server or configuring a connection to a cloud-based MongoDB instance. This ensures the backend can perform CRUD operations on the application's data.

7. Optional Containerized Setup: For environments using Docker, the application can be containerized to ensure consistency. This involves building and running Docker containers for the frontend and backend, with configurations defined in a docker-compose file.
8. Verification: After installation, verify that all dependencies are correctly installed and the environment variables are properly set. This ensures the application is ready to run without runtime errors.

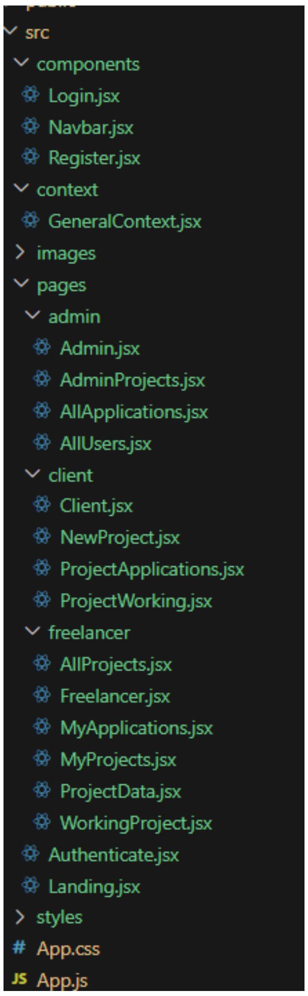
This theoretical setup process establishes a functional development environment for FreelanceFinder, enabling developers to proceed with running, testing, or further customizing the application.

5. Folder Structure

Client:

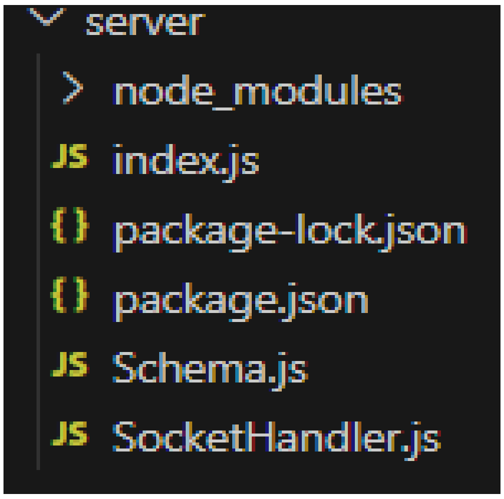


```
client
  > node_modules
  > public
  v src
    > components
    > context
    > images
    > pages
    > styles
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
```



```
v src
  v components
    Login.jsx
    Navbar.jsx
    Register.jsx
  v context
    GeneralContext.jsx
  > images
  v pages
    v admin
      Admin.jsx
      AdminProjects.jsx
      AllApplications.jsx
      AllUsers.jsx
    v client
      Client.jsx
      NewProject.jsx
      ProjectApplications.jsx
      ProjectWorking.jsx
    v freelancer
      AllProjects.jsx
      Freelancer.jsx
      MyApplications.jsx
      MyProjects.jsx
      ProjectData.jsx
      WorkingProject.jsx
    Authenticate.jsx
    Landing.jsx
  > styles
  App.css
  App.js
```

Server:



```
server
  node_modules
  index.js
  package-lock.json
  package.json
  Schema.js
  SocketHandler.js
```

6. Running the Application

To run the FreelanceFinder application locally, both the frontend and backend servers must be started. Below are the commands to initiate each server**:**

- **Frontend:**

  Navigate to the client directory: **cd client**

  Start the React development server: **npm start**

- **Backend:**

  Navigate to the server directory**: cd server**

  Start the Node.js/Express.js server**: npm start**

- **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.

Ensure MongoDB is running (locally or via MongoDB Atlas) and the .env file in the server directory is correctly configured with the required environment variables (e.g., MONGO_URI, JWT_SECRET) before starting the servers.

7. API Documentation

The FreelanceFinder platform exposes a comprehensive set of RESTful API endpoints that power the user experience for clients, freelancers, and administrators. These APIs handle everything from user authentication to project management, bidding, submissions, and chat functionalities**.**

User Authentication

 Register User: Allows new users to register with details such as username, email, password, and user type (client/freelancer).

 Login User: Authenticates a user by verifying credentials and returns user data upon successful login.

Freelancer Management

 Fetch Freelancer: Retrieves freelancer profile data based on the associated user ID.

 Update Freelancer Profile: Enables freelancers to update their skill set and personal description for better project matching.

Project Management

 Fetch Single Project: Fetches detailed information about a specific project using its unique identifier.

- **Fetch All Projects:** Returns a list of all available projects for browsing by freelancers.
- **Create New Project**: Allows clients to post new freelance opportunities with details like title, description, budget, and required skills.

Bidding System

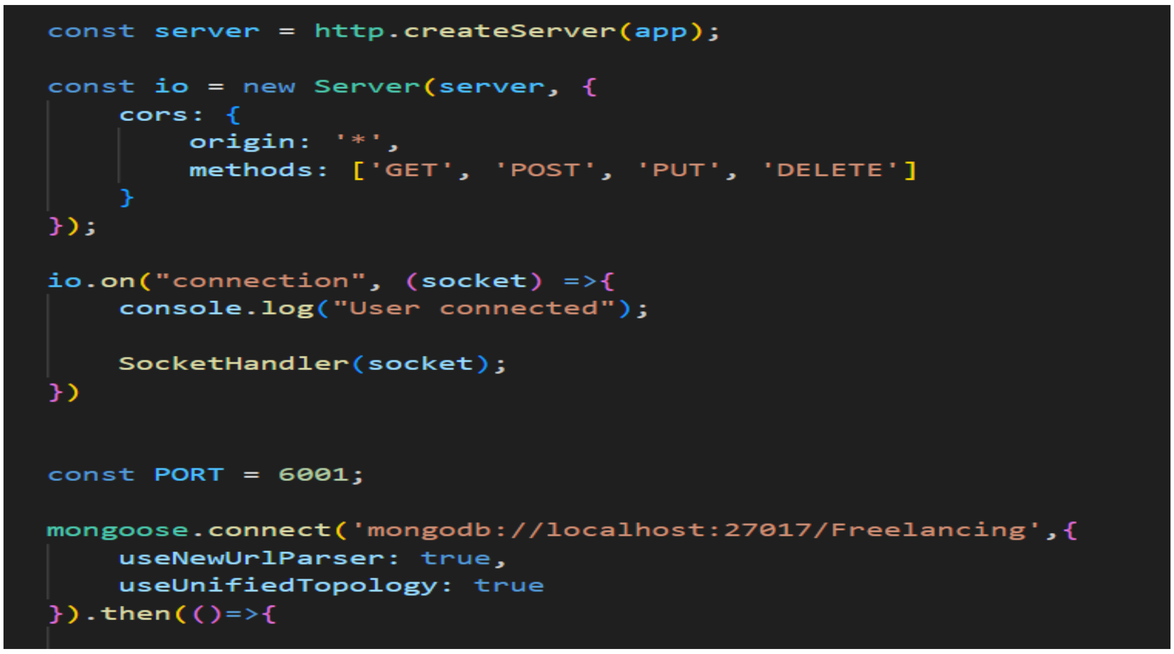
- **Submit Bid:** Enables freelancers to place bids on projects, including a proposal, estimated time, and bid amount**.**
- **Fetch Applications**: Retrieves all applications submitted for projects.
- **Approve Application:** Assigns a project to a freelancer by approving their application.
- **Reject Application: Rejects a specific freelancer’s application for a project.**

Project Submission

- **Submit Project:** Freelancers submit completed work, including project links, documentation, and notes.
- **Approve Submission:** The client approves the submitted project, marking it as completed and releasing payment to the freelancer.
- **Reject Submission:** The client rejects a submission if requirements are not met, resetting submission status.

User & Chat Management

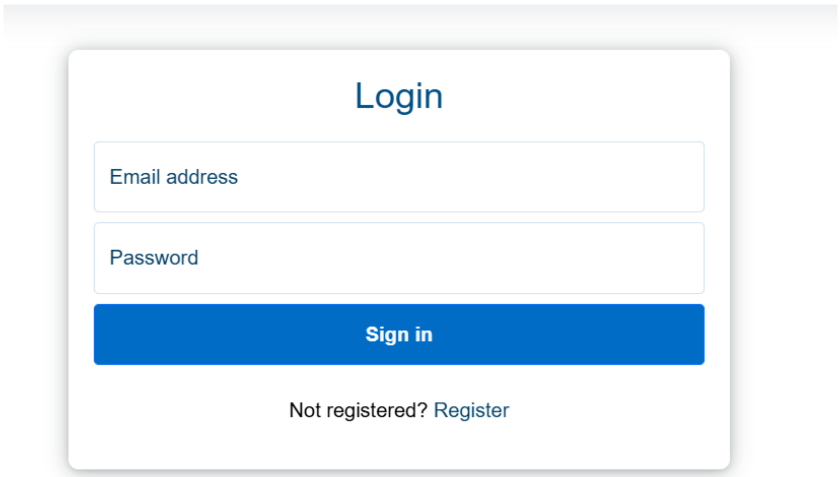
- **Fetch Users:** Returns a list of all registered users on the platform.
- **Fetch Chats:** Retrieves the chat history between clients and freelancers based on chat ID.



```
const server = http.createServer(app);

const io = new Server(server, {
    cors: {
        origin: '*',
        methods: ['GET', 'POST', 'PUT', 'DELETE']
    }
});

io.on("connection", (socket) =>{
    console.log("User connected");

    SocketHandler(socket);
})


const PORT = 6001;

mongoose.connect('mongodb://localhost:27017/Freelancing',{
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(()=>{
```

8. Authentication

- Authentication is handled using username and password login.
- Passwords are hashed using bcrypt before storing in the database.
- On registration, user details and hashed password are stored in MongoDB.
- Login validates the hashed password against user input.
- User roles (client, freelancer, admin) define access privileges.
- Role-based authorization is implemented on the frontend using local storage.
- Session management is done using local storage to retain user data.
- No JWT tokens are used currently, but structure allows future integration.
- Protected routes validate user actions on the backend.
- CORS is enabled for secure cross-origin API communication**.**

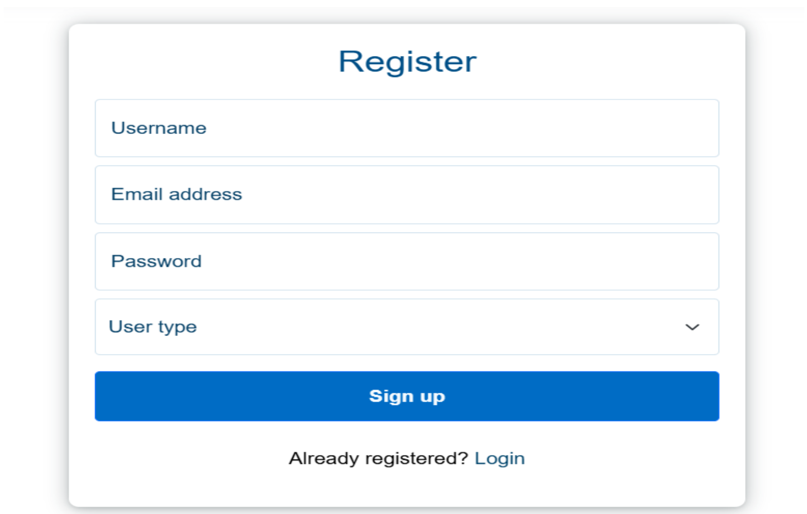

Login

Email address

Password

Sign in

Not registered? Register


Register

Username

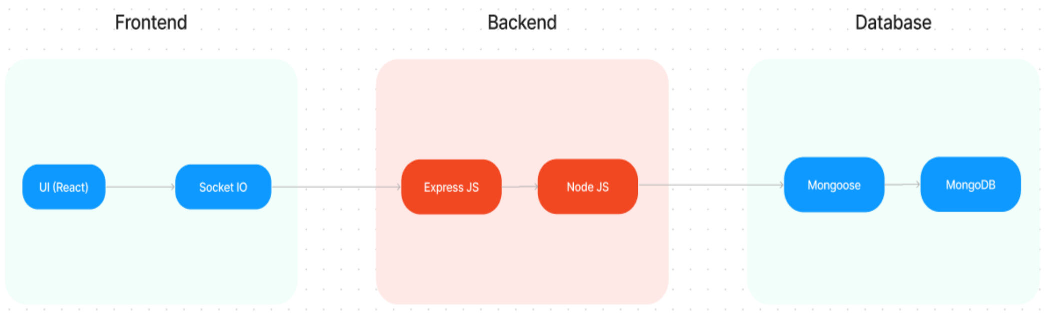
Email address

Password

User type

Sign up

Already registered? Login



9. User Interface

The user interface is designed to be clean, responsive, and role-specific. Clients can easily post projects, freelancers can browse and bid on jobs, and both parties have access to a real-time chat interface. Screenshots of login, registration, dashboards, project forms, and the messaging system demonstrate the intuitive flow of the application.

10. Testing

The testing strategy for FreelanceFinder focused on ensuring functionality, performance, and user experience across all core modules. The application was thoroughly tested across three primary layers: **manual testing**, **performance testing**, and **future readiness for automation**.

Manual Testing

Each module—including registration, login, project posting, bidding, chat, and project submission—was manually tested with various input scenarios to validate correct behavior and edge cases. Role-based access (admin, client, freelancer) was also verified for correct redirection and data access.

Performance Testing

Simulated multiple users posting projects, bidding, and using the chat system simultaneously to monitor backend responsiveness and real-time performance. The server remained stable with minimal delay under concurrent usage.

Test Scenarios Included

- Valid and invalid login attempts
- Posting a project with and without required fields
- Bidding on a project multiple times
- Freelancer and client communication via chat
- Submission and approval workflows

Tools Used

- **Postman**: For testing API endpoints (POST, GET, etc.)
- **Browser DevTools**: For inspecting frontend requests and network responses
- **Console logs and server-side logs**: For real-time error tracking and validation
- **Manual Test Case Spreadsheet**: To track inputs, outputs, and expected behaviors

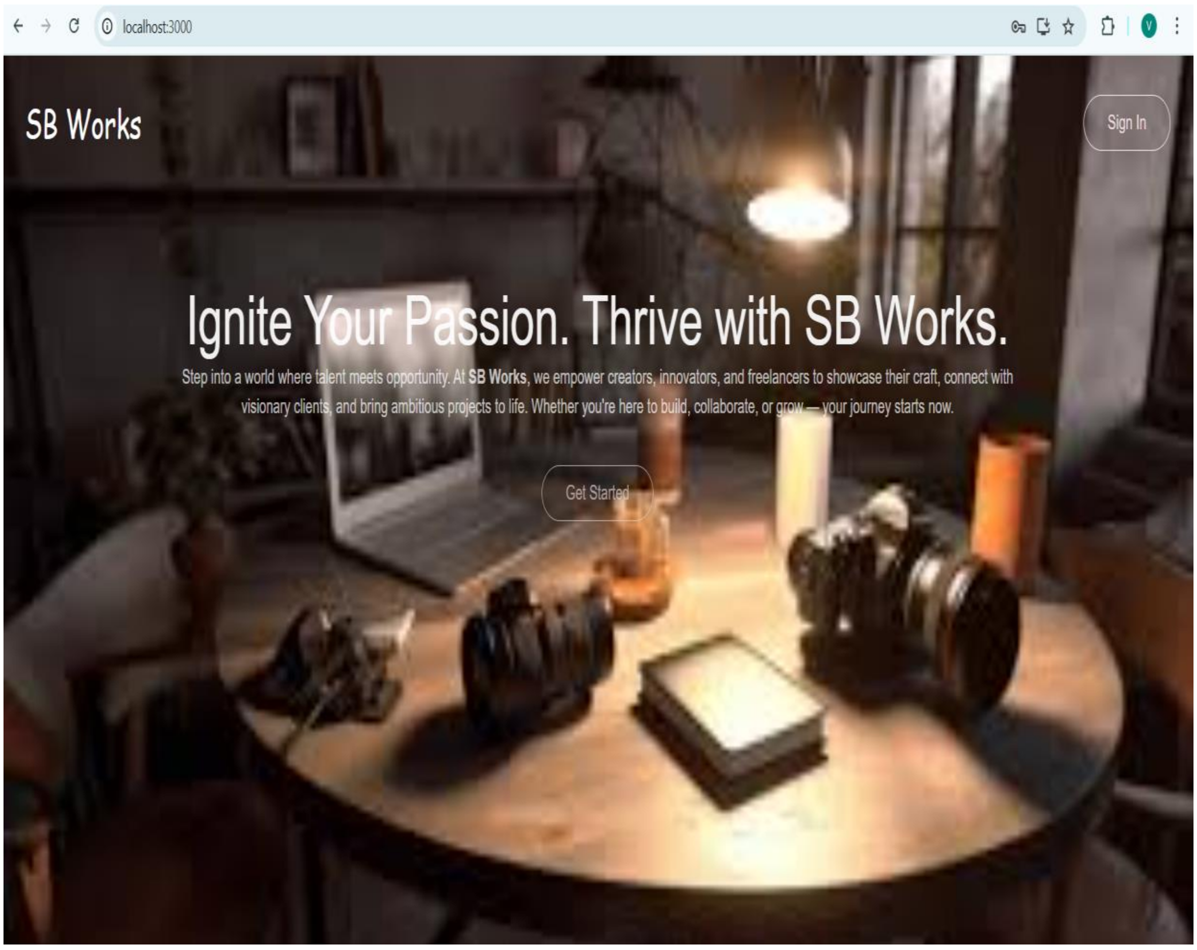
Future Scope of Testing

Automated testing frameworks like **Jest** for unit testing and **Supertest** for API testing will be integrated in future versions to ensure test coverage and continuous integration.

11.Project Implementation Screenshots of execution:

On completing the development part, we then run the application one last time to verify all the functionalities and look for any bugs in it. The user interface of the application looks a bit like the images provided below.

Landing page:



localhost:3000

SB Works

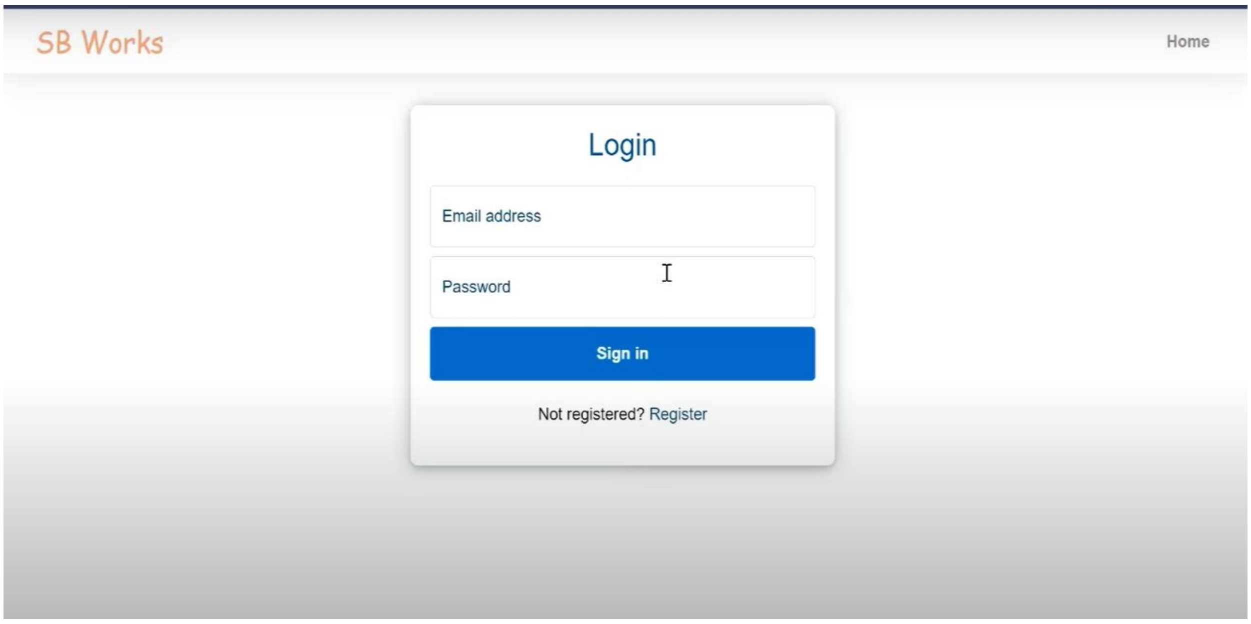
Sign In

Ignite Your Passion. Thrive with SB Works.

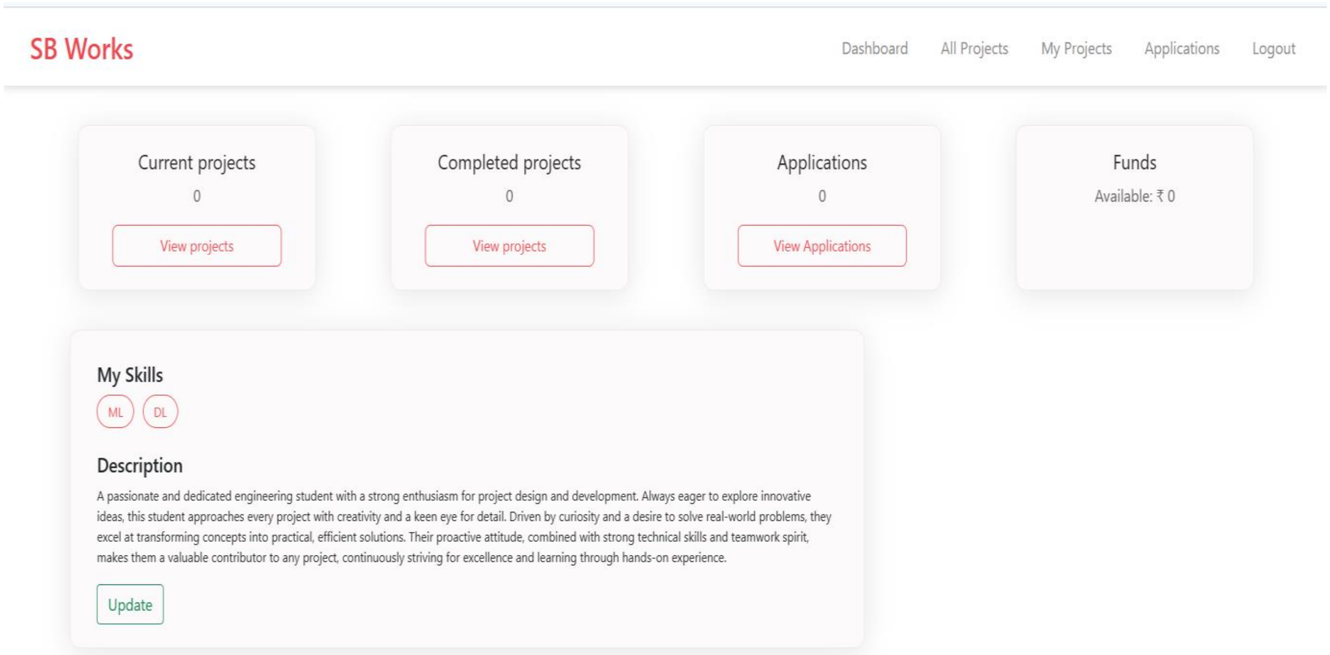
Step into a world where talent meets opportunity. At **SB Works**, we empower creators, innovators, and freelancers to showcase their craft, connect with visionary clients, and bring ambitious projects to life. Whether you're here to build, collaborate, or grow — your journey starts now.

Get Started

Authentication:

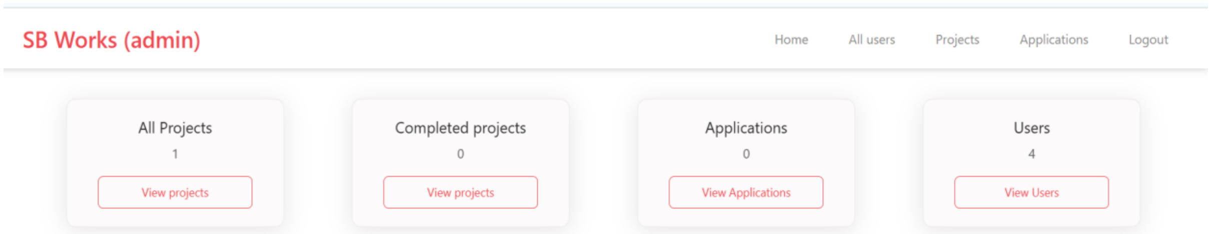


Freelancer dashboard:



A passionate and dedicated engineering student with a strong enthusiasm for project design and development. Always eager to explore innovative ideas, this student approaches every project with creativity and a keen eye for detail. Driven by curiosity and a desire to solve real-world problems, they excel at transforming concepts into practical, efficient solutions. Their proactive attitude, combined with strong technical skills and teamwork spirit, makes them a valuable contributor to any project, continuously striving for excellence and learning through hands-on experience.

Admin dashboard:



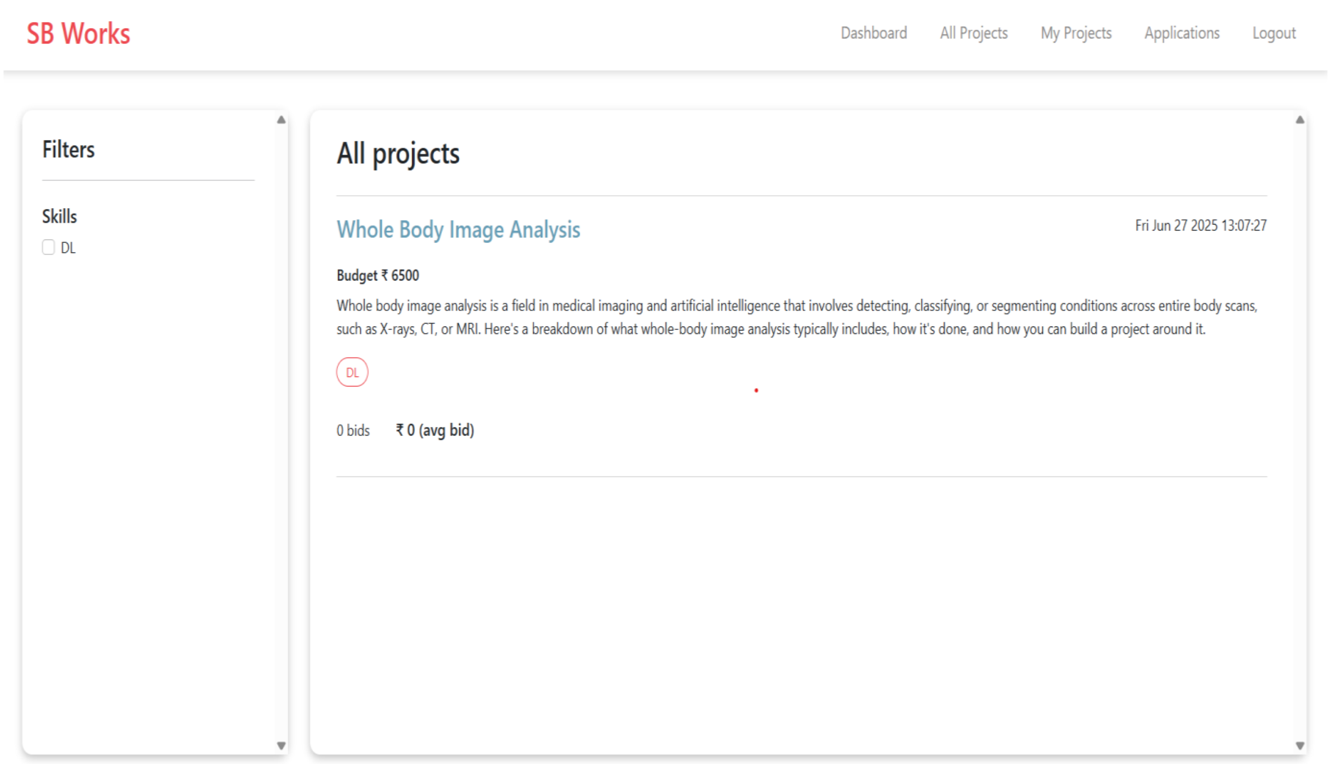
All Projects
1
View projects

Completed projects
0
View projects

Applications
0
View Applications

Users
4
View Users

All projects:



All projects

Whole Body Image Analysis

Fri Jun 27 2025 13:07:27

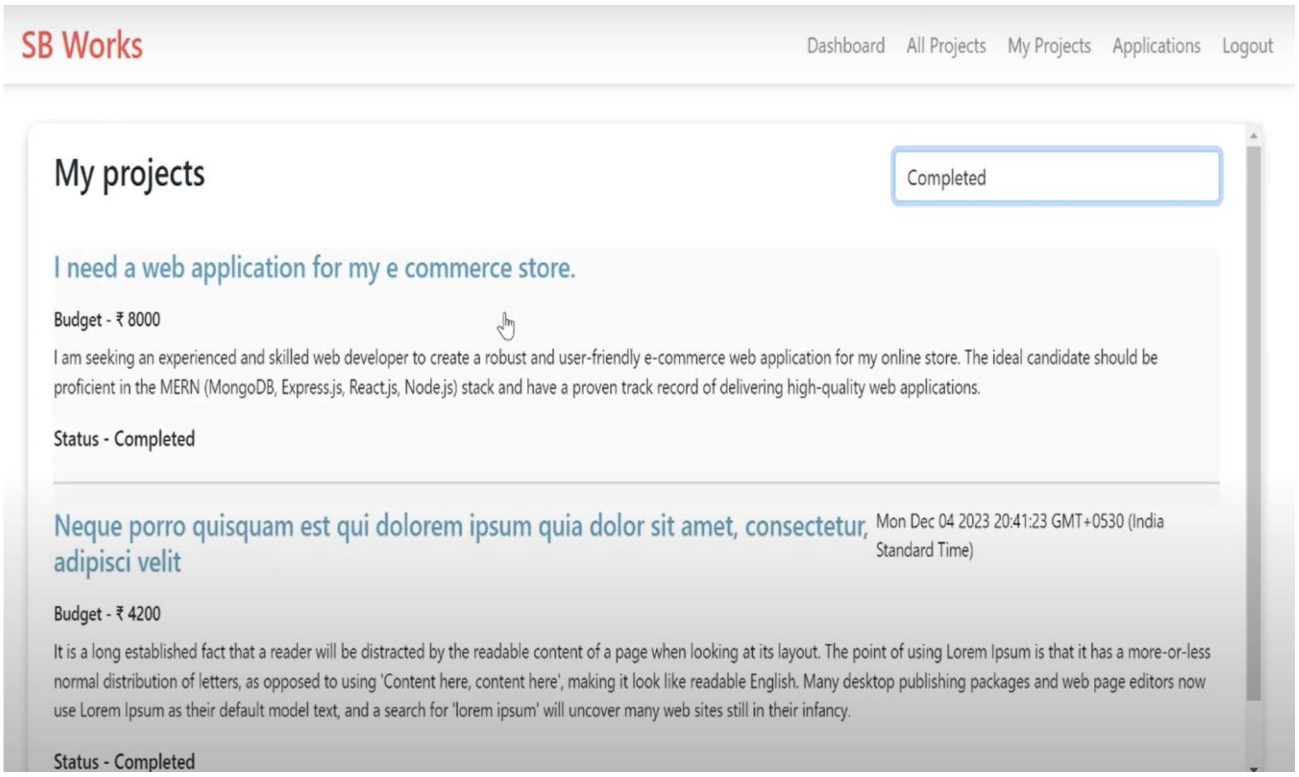
Budget ₹ 6500

Whole body image analysis is a field in medical imaging and artificial intelligence that involves detecting, classifying, or segmenting conditions across entire body scans, such as X-rays, CT, or MRI. Here's a breakdown of what whole-body image analysis typically includes, how it's done, and how you can build a project around it.

DL

0 bids ₹ 0 (avg bid)

Freelance projects:


SB Works

Dashboard All Projects My Projects Applications Logout

My projects

Completed

I need a web application for my e commerce store.

Budget - ₹ 8000

I am seeking an experienced and skilled web developer to create a robust and user-friendly e-commerce web application for my online store. The ideal candidate should be proficient in the MERN (MongoDB, Express.js, React.js, Node.js) stack and have a proven track record of delivering high-quality web applications.

Status - Completed

Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit

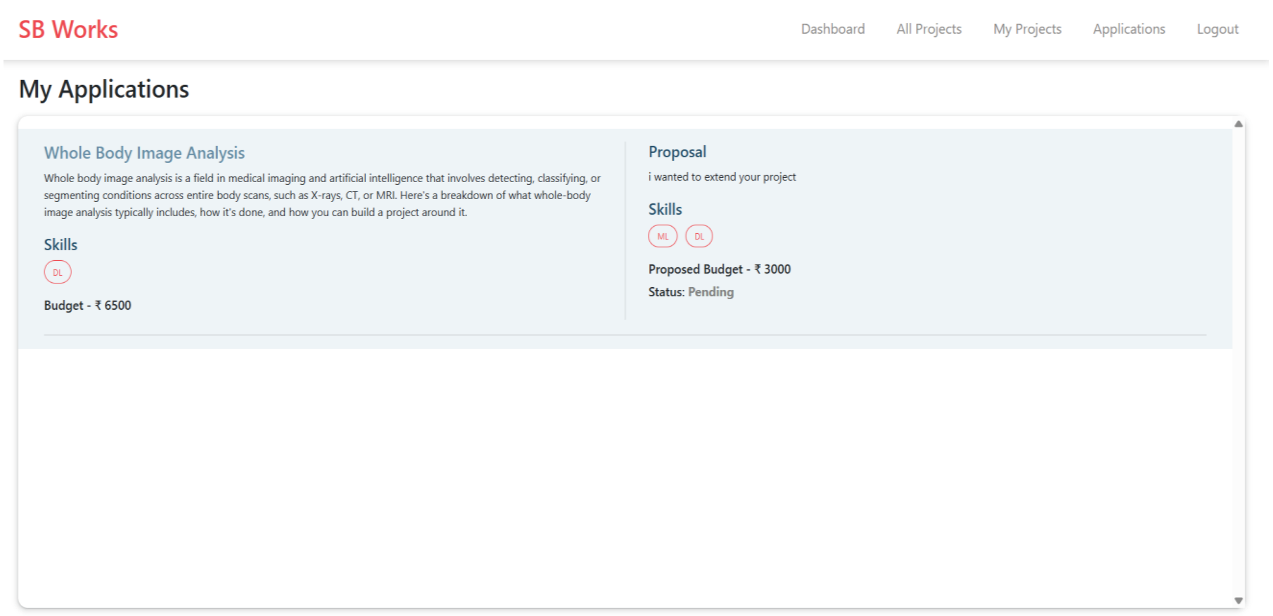
Mon Dec 04 2023 20:41:23 GMT+0530 (India Standard Time)

Budget - ₹ 4200

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy.

Status - Completed

Applications:


SB Works

Dashboard All Projects My Projects Applications Logout

My Applications

Whole Body Image Analysis

Whole body image analysis is a field in medical imaging and artificial intelligence that involves detecting, classifying, or segmenting conditions across entire body scans, such as X-rays, CT, or MRI. Here's a breakdown of what whole-body image analysis typically includes, how it's done, and how you can build a project around it.

Skills

DL

Budget - ₹ 6500

Proposal

i wanted to extend your project

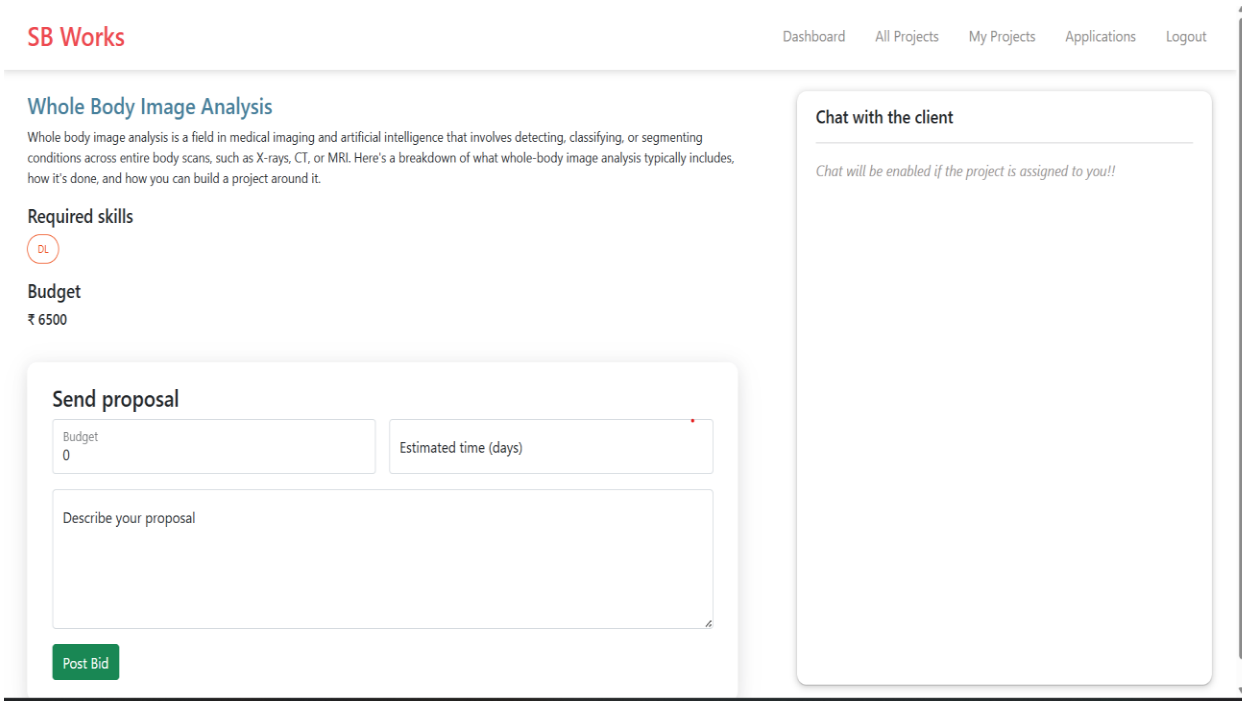
Skills

ML DL

Proposed Budget - ₹ 3000

Status: Pending

Project page:


SB Works

Dashboard
All Projects
My Projects
Applications
Logout

Whole Body Image Analysis

Whole body image analysis is a field in medical imaging and artificial intelligence that involves detecting, classifying, or segmenting conditions across entire body scans, such as X-rays, CT, or MRI. Here's a breakdown of what whole-body image analysis typically includes, how it's done, and how you can build a project around it.

Required skills

DL

Budget

₹ 6500

Send proposal

Budget
0

Estimated time (days)

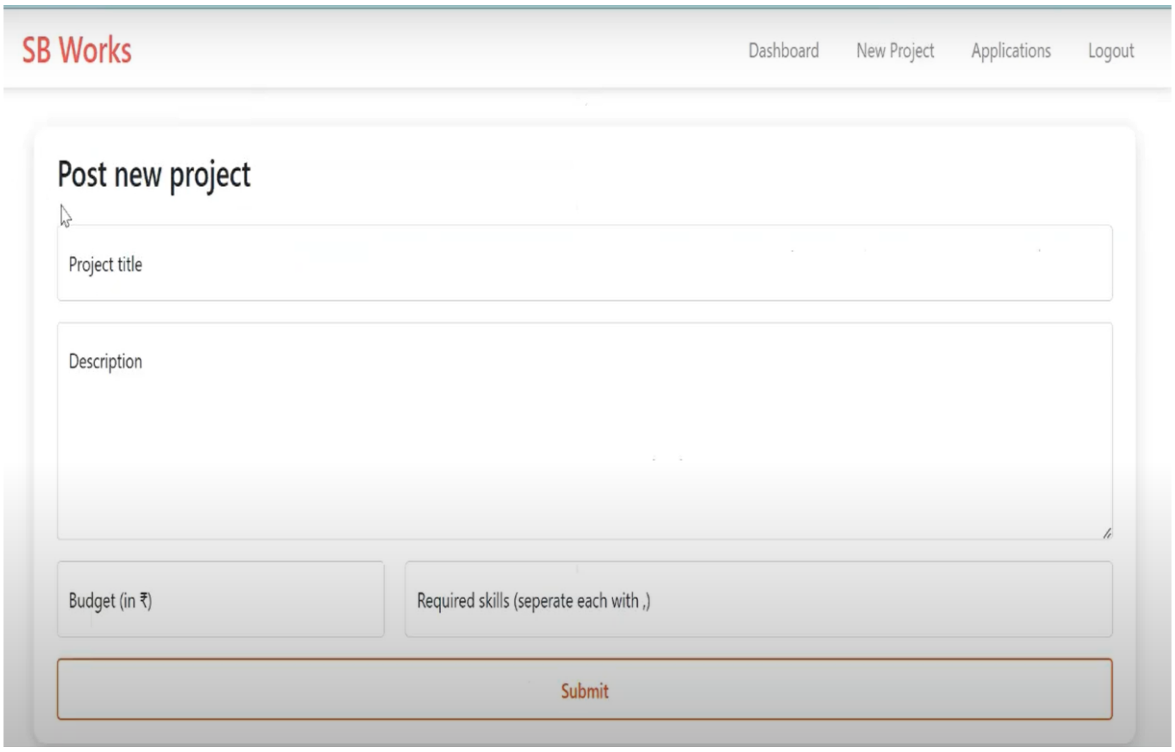
Describe your proposal

Post Bid

Chat with the client

Chat will be enabled if the project is assigned to you!!

New project:



Post new project

Project title

Description

Budget (in ₹)

Required skills (seperate each with ,)

12. Known Issues

Despite thorough development and testing, a few known issues and limitations still exist in the current version of FreelanceFinder:

◈ 1. Real-Time Chat Delay

Occasionally, messages in the chat module experience slight delays in rendering when many users are active simultaneously. This is due to limitations in the current Socket.io event handling and will be optimized in future iterations**.**

◈ **2. Project Filtering and Search**

The freelancer dashboard currently lacks advanced filtering options such as sorting by skills, budget, or project status, making project discovery less efficient.

◈ 3. Admin Panel Limitations

The admin role has limited visibility and control; there's no UI for managing user status (e.g., block, delete) or project moderation.

◈ **4. No Rate Limiting or Throttling**

Backend APIs do not yet implement request throttling or rate limiting, leaving the system vulnerable to abuse (e.g., spamming endpoints).

13. Future Enhancements

The current version of FreelanceFinder serves as a functional minimum viable product (MVP). Several enhancements are planned to elevate functionality, performance, and user experience:

◈ **1. Payment Gateway Integration**

Implementing a secure and reliable payment system (e.g., Razorpay, Stripe) to facilitate transactions between clients and freelancers.

◈ **2. Rating and Review System**

Allowing clients and freelancers to rate and review each other after project completion to build trust and improve accountability.

◈ **3. Advanced Project Filtering and Search**
Adding filters by skills, budget range, deadline, project status, and a full-text search bar for better discoverability of projects.

◈ **4. Mobile App Version**
Developing a cross-platform mobile application using React Native or Flutter to enhance accessibility and user convenience.

◈ **5. Email and OTP Verification**
Adding email verification and optional two-factor authentication (2FA) to improve account security.

◈ **6. Admin Dashboard Expansion**
Creating a more powerful admin panel with tools to moderate content, manage users, view analytics, and take administrative actions.

◈ **7. Notification System**
Implementing real-time notifications for bid approvals, chat messages, project submissions, and other important actions.

◈ **8. AI-based Job Recommendations**
Using machine learning to suggest relevant projects or freelancers based on skills, history, and user behavior.