JEPPIAAR
ENGINEERING COLLEGE

FREELANCING APPLICATION MERN

A PROJECT REPORT

SUBMITTED BY

Arockia Varshini J - 310821205013

Lavanya S - 310821205039

Nethra B - 310821205055

Abilash A - 310821205001

Akash Rao J - 310821205008

IN PARTIAL FULFILLMENT FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

JEPPIAAR ENGINEERING COLLEGE

ANNA UNIVERSITY

CHENNAI – 600025

NOVEMBER 2024

ANNA UNIVERSITY:CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"FREELANCING APPLICATION MERN**" is the bonafide work of **"AROCKIA VARSHINI J(310821205013)", "LAVANYA S (310821205039)", "NETHRA B(310821205055)", "ABILASH A(310821205001)" "AKASH RAO J(310821205008)"** who carried out the project under supervision.

SUPERVISOR

BHAGYALAKSHMI T

HEAD OF THE DEPARTMENT

DR.ANITHA JAWAHAR

TABLE OF CONTENTS

1. Introduction

Project Title: Freelancing Application with MERN Stack

This project provides an online platform for freelancers and clients to collaborate on various projects. Freelancers can search for job postings, place bids, and engage in real-time communication with clients. The platform also allows clients to post jobs, review bids, and choose suitable freelancers. The goal of the project is to provide a seamless and efficient experience for both freelancers and clients while ensuring secure transactions, job management, and real-time communication.

Team Members:

- Akash Rao J (Frontend Developer): Responsible for building the user interface using React and ensuring the frontend's responsiveness and user experience.

- Arockia Varshini J (Full Stack Developer): Works on both the frontend and backend to ensure seamless integration and overall functionality of the application.

- Nethra B and Lavanya S (Backend Developers): Handles the backend API development with Node.js, Express.js, and MongoDB.

- Abilash A (UI & UX Design) : Focused on designing the platform's interface and creating intuitive and user-friendly experiences.

The project is developed with the MERN (MongoDB, Express.js, React, Node.js) stack, utilizing Socket.io for real-time communication and Redux for state management on the frontend.

2. Project Overview

Purpose

The Freelancing Platform is designed to bridge the gap between freelancers and clients, offering an intuitive and feature-rich environment where both parties can collaborate effectively. The primary challenge for freelancers is finding reliable clients and projects that match their skills, while clients often struggle to find the right talent for their specific needs. This platform is developed to solve those challenges, providing a one-stop solution for job postings, bidding, real-time communication, and secure payments.

The platform focuses on creating a user-friendly experience that simplifies the workflow for both freelancers and clients. By integrating real-time messaging, secure payment processing, and advanced job management features, the platform ensures an efficient and productive collaboration between users.

The core goals of the platform include:

- Building a Vibrant Community: The platform aims to foster a dynamic community where both freelancers and clients can come together to create and engage in long-term, meaningful professional relationships. It will support various industries and offer a wide range of job opportunities, from short-term gigs to long-term projects.

- Seamless Communication: With the integration of Socket.io for real-time messaging, freelancers and clients can communicate instantly and efficiently. This feature is designed to enhance collaboration, as users can discuss project details, negotiate terms, and resolve issues in real-time, without delays.

- Secure Payment System: Payment handling is a crucial aspect of freelancing platforms, and this project integrates secure payment gateways to handle transactions between clients and freelancers. The system ensures that payments are processed efficiently and securely, offering both parties peace of mind throughout the project’s lifecycle.

- Comprehensive Job Management: The platform enables job postings, bidding systems, and profile management for freelancers and clients alike. Freelancers can browse job listings, place custom bids, and showcase their skills, while clients can manage job posts, review bids, and select the most suitable freelancers for their needs. This ensures that all aspects of job management are centralized in one platform, making it easier to track project progress.

- Enhanced User Experience: Designed with user experience in mind, the platform prioritizes intuitive navigation, an attractive user interface, and seamless integration across the entire system. This makes it easy for both beginners and experienced freelancers and clients to engage with the platform and get the most out of its features.

Features

1. User Authentication:

- The platform uses JSON Web Tokens (JWT) for secure authentication, ensuring that only authorized users can access their accounts and perform actions such as posting jobs or bidding.

- Bcrypt is used to securely hash passwords before storing them in the database, ensuring that user credentials are protected from unauthorized access and potential security breaches. This method guarantees that even in the unlikely event of a data leak, passwords cannot be easily compromised.

2. Profile Management:

- Freelancers can create detailed profiles showcasing their portfolios, work experience, and skills. This allows potential clients to assess their capabilities and make informed decisions. Freelancers can also update their availability, hourly rates, and preferences to ensure their profiles are always current.

- Clients can create accounts to post job listings, manage project progress, and interact with freelancers. They can also track job statuses, review freelancer bids, and view project milestones to stay on top of their hiring process.

3. Job Posting & Bidding System:

- Clients can post job opportunities with detailed descriptions, required skills, and deadlines. This allows freelancers to understand the scope of the project and make informed decisions on whether to bid.

- Freelancers can browse available jobs and place bids with custom proposals. Each proposal can include a price, estimated time to completion, and any additional details the freelancer feels are necessary to win the job. This bidding system encourages competitive pricing and ensures that clients get the best value for their projects.

4. Real-time Messaging:

- The platform incorporates Socket.io to enable real-time messaging between freelancers and clients. Whether it's discussing project details, negotiating terms, or providing updates, this feature ensures that communication remains instantaneous and fluid. This eliminates the need for long email exchanges and provides a faster, more efficient way to communicate.

- Additionally, users will receive notifications for new messages, ensuring they never miss important updates, thus maintaining a smooth flow of work throughout the project.

5. Payment Processing:

- Integration with secure payment gateways ensures that freelancers are paid promptly after completing a project, and clients can make payments securely without needing to leave the platform. The platform will handle invoicing, payment tracking, and dispute resolution to offer a transparent and fair process for both parties.

- The system will also include features like escrow services, where payments are held until the client approves the completed work, ensuring a fair process for both freelancers and clients.

6. Admin Panel:

- An Admin Panel allows platform administrators to manage users, monitor job postings, handle disputes, and oversee platform activity. Admins can view user statistics, handle content moderation, and ensure that the platform runs smoothly and securely.

- This panel also includes tools to manage reported issues, approve or reject job postings, and maintain a healthy ecosystem on the platform. This helps ensure that the platform remains free from misuse or fraudulent activities.

7. Search and Filters:

- The platform will offer advanced search functionality, allowing users to search for jobs based on specific criteria such as skills, budget, and job type. Freelancers can filter job listings to find projects that best match their expertise and preferences, while clients can easily locate freelancers who fit their requirements.

- Clients can also filter bids by price, timeline, and freelancer experience, ensuring that they find the best freelancer for their project quickly and efficiently.

3. Architecture

Frontend Architecture

The frontend of the Freelancing Platform is built with React.js, utilizing a component-based structure. This approach allows us to create modular and reusable components, ensuring ease of maintenance and scalability. Key aspects of the frontend architecture include:

- Component-Based Design:

Every major feature of the platform (such as job listings, profiles, and bidding forms) is developed as a separate React component. These components are independent and reusable, which makes the code easier to manage and extend as the platform grows.

- State Management:

The frontend uses Redux Toolkit for global state management. Redux provides a centralized store for the application's state, making it easier to manage and synchronize data across different components. It ensures that updates to key features, such as job postings, bids, and messages, are reflected in real-time across the platform.

- Routing:

React Router is used to navigate between different pages, such as the Home, Profile, Jobs, and Messages pages. This enables users to smoothly transition between sections of the platform without needing to reload the page. React Router Hash Link is also used to provide smooth scrolling to anchor links within the same page.

- UI/UX Design:

The platform is designed with CSS/SCSS, focusing on a clean, responsive layout. A mobile-first approach ensures that the platform works well on mobile devices before enhancing it for larger screens, offering an intuitive and accessible experience across all devices.

Backend Architecture

The backend is developed using Node.js and Express.js, which together create a RESTful API. The backend handles the core operations of the platform, such as user authentication, job management, bidding, and real-time messaging. Key features of the backend architecture include:

- API Routes:

The backend exposes various API routes for interacting with the platform's data. Using Express.js, these routes handle CRUD operations (Create, Read, Update, Delete) for users, jobs, bids, and messages. Clients can post jobs, freelancers can place bids, and users can update profiles.

- Real-Time Communication:

Socket.io is integrated into the backend to enable real-time communication between freelancers and clients. This feature powers the platform's live chat functionality, allowing users to communicate instantly without refreshing the page. Socket.io ensures seamless, two-way communication between users, making collaboration more efficient.

- Security:

Bcrypt is used for hashing passwords before storing them in the database, ensuring that user credentials remain secure. JWT (JSON Web Tokens) are used for authentication, ensuring that only authorized users can access sensitive routes and data, such as user profiles and job listings.

Database Architecture

The platform uses MongoDB, a NoSQL database, to store data. MongoDB is flexible and scalable, making it suitable for the dynamic nature of this platform.

The database is organized into collections, each serving a specific purpose. The key collections include:

- Users:

This collection stores data about freelancers and clients, such as authentication details, contact information, and profiles. For freelancers, it includes skills and portfolio details, while clients can manage their job postings and reviews of freelancers.

- Jobs:

The Jobs collection holds all job postings made by clients. Each document contains details such as the job description, budget, required skills, and deadlines. Freelancers can browse these listings, apply for jobs, and submit bids based on their expertise.

- Bids:

Freelancers can place bids on jobs, and these bids are stored in the Bids collection. Each bid includes information like the bid amount, freelancer's message to the client, and references to the job they're bidding for. This system allows clients to compare bids and choose the best freelancer for the job.

- Messages:

Messages are stored in a dedicated collection, allowing freelancers and clients to chat in real-time. This collection ensures that all communication is saved, so users can revisit their discussions whenever needed. Socket.io powers the live messaging functionality, while MongoDB stores the messages for persistent access.

Here's the updated version with the correct ports:

4. Setup Instructions

Prerequisites

Before setting up the project locally, make sure you have the following software installed on your machine:

- Node.js: Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It is required to run both the frontend and backend of this project. You can download Node.js from the official website [here](https://nodejs.org/).

- MongoDB: MongoDB is a NoSQL database used to store user data, job postings, bids, and messages. You can either install MongoDB locally or use MongoDB Atlas, a cloud-based solution, for easier setup and management.

- npm (Node Package Manager): npm is used to install the necessary dependencies for both the frontend and backend. npm comes pre-installed with Node.js, so you will have it once you install Node.js.

Installation

1. Clone the Repository

Begin by cloning the project repository to your local machine using the following command:

```bash
git clone <repository-url>
```

Replace `<repository-url>` with the actual URL of your project repository.

2. Install Dependencies

After cloning the repository, navigate to the following directories to install the necessary npm packages:

- For the Frontend (Client):

Navigate to the `client` directory, which contains all the frontend code built with React. Run the following command to install the required dependencies:

```bash
cd client/
npm install
```

This will install all the packages listed in the `package.json` for the frontend.

- For the Backend (API):

Similarly, navigate to the api directory, which contains the backend code built with Node.js and Express. Run the following command:

```bash
cd api/
npm install
```

This will install all the necessary backend dependencies such as Express, MongoDB, and other packages required for API functionality.

- For the Socket Server:

The socket directory is responsible for real-time communication between the frontend and backend. Navigate to the socket directory and run the following command to install the required packages:

```bash
cd socket/
npm install
```

This will install the necessary dependencies for handling WebSocket communication using Socket.io.

3. Setup Environment Variables

To ensure the frontend, backend, and socket server communicate securely, you need to set up the environment variables:

- Create a `.env` file in the root directory for both the frontend (client), backend (api), and socket server (socket).

- In the backend `.env` file (`api/.env`), include the following:

```env
MONGODB_URI=<your-mongodb-uri>
JWT_SECRET=<your-secret-key>
```

- MONGODB_URI: Connection string for MongoDB. If using MongoDB Atlas, you'll find this in your dashboard. For local setups, it could look like `mongodb://localhost:27017/your-database-name`.

- JWT_SECRET: A secret key used for generating JWT tokens for authentication. You can generate a random string for this key.

- In the socket `.env` file (`socket/.env`), you may want to include configurations for WebSocket authentication, if applicable.

- Frontend Environment Variables: While the frontend does not typically need environment variables for authentication, you may want to set up some for API URLs, e.g.,

```env
REACT_APP_API_URL=http://localhost:3001
```

This variable can be used to interact with the backend API from the frontend.

4. Run the Server, Client, and Socket Server

After installing all dependencies and setting up your environment variables, you are ready to run the project. Open three separate terminal windows (or tabs) to run the servers simultaneously:

- For the Backend (API):

In the terminal, navigate to the api directory and run the following command to start the backend server:

```bash
cd api/
npm start
```

The backend will now be running at `http://localhost:3001`.

- For the Frontend (Client):

In another terminal window, navigate to the client directory and run:

```bash
cd client/
npm start
```

The frontend will be available at `http://localhost:3000`.

- For the Socket Server:

  In the third terminal window, navigate to the socket directory and run:

  ```bash
  cd socket/
  npm start
  ```

  The socket server will be running, providing real-time communication capabilities between the frontend and backend.

Access the Application

Once all the servers are running, you can access the application in your web browser:

- Frontend: Open your browser and go to http://localhost:3000 to interact with the user interface of the application.
- Backend: The backend API can be accessed at http://localhost:3001, though it is primarily used by the frontend.
- Socket Server: The real-time communication can be tested using the frontend, which connects to the socket server to facilitate messaging between users.

—

5. Folder Structure

Client Folder

The client folder contains all the frontend-related files, built using React.js.

```
client/
│
├── node_modules/            # Contains the dependencies installed for the frontend (via npm install).
├── public/                  # Contains the public files like the `index.html` and favicon.
├── src/                     # Contains all the source code of the React frontend.
│   ├── assets/              # Static files like images, fonts, and icons.
│   ├── components/          # Reusable React components such as JobCard, ProfileForm, BidForm.
│   ├── ClientComponents/    # Components specific to clients (e.g., job postings, bid management).
│   ├── FreelancerComponents/  # Components specific to freelancers (e.g., bidding, profile setup).
│   ├── redux/               # Redux-related code for state management (slices, actions, reducers).
│   └── App.js               # The root component that contains the routing logic for the app.
└── node_modules/            # Dependencies for the frontend are stored here after running `npm install`.
```

- assets/: Contains images, fonts, icons, or other static assets used throughout the frontend application.

- components/: Contains reusable React components that can be used across different pages, such as JobCard, ProfileForm, and BidForm.

- ClientComponents/: Components that are primarily used by the clients (e.g., components for managing job postings and job details).

- FreelancerComponents/: Components tailored for freelancers, including components for bidding on jobs, viewing project listings, and profile editing.

- redux/: Contains Redux slices, actions, and reducers that manage global state such as user data, job listings, and messaging updates.

Server Folder

The server folder contains the backend-related files, which include the API routes, controllers, models, and other necessary server-side logic.

```
api/
│
├── config/                 # Contains configuration files for database connection and environment variables.
├── controllers/            # Contains business logic for creating jobs, placing bids, and managing user messages.
│   ├── jobController.js    # Logic for creating and fetching job posts.
│   ├── userController.js   # Logic for handling user registration, authentication, and management.
│   └── bidController.js    # Logic for placing bids and fetching bid information.
├── middleware/             # Middleware for tasks like authentication checks, error handling, and logging.
```

```
│   ├── authMiddleware.js        # Middleware for verifying JWT tokens and securing routes.
│   └── errorMiddleware.js       # Middleware for handling errors across the application.
├── models/                      # Mongoose schemas for MongoDB collections (users, jobs, bids).
│   ├── userModel.js             # Defines user data schema for freelancers and clients.
│   ├── jobModel.js              # Defines job posting schema with attributes like description, budget, etc.
│   └── bidModel.js              # Defines bid schema with information like bid amount and freelancer details.
├── node_modules/                # Dependencies installed for the backend are stored here.
├── routes/                      # Defines API routes for user authentication, job postings, bidding, and messaging.
│   ├── userRoutes.js            # Contains routes for user management (sign up, login, etc.).
│   ├── jobRoutes.js             # Contains routes for job-related actions (posting, updating, viewing).
│   └── bidRoutes.js             # Contains routes for bid-related actions (placing and fetching bids).
├── uploads/                     # Folder to store uploaded files, such as job-related documents and freelancer portfolios.
└── server.js                    # Entry point for the server, sets up Express.js and connects to the database.
```

- config/: This folder contains the configuration files like the database connection (MongoDB) and the environment variables used by the server.

- controllers/: Handles the business logic for each major action (e.g., user registration, job posting, placing bids). Each controller is responsible for specific actions related to its resource (jobs, bids, users).

- middleware/: Includes reusable middleware functions that are applied to routes, such as authentication verification and error handling.

- models/: Defines Mongoose schemas for the collections in the MongoDB database. Each model corresponds to a collection (users, jobs, bids).

- routes/: Contains Express.js route files that define the API endpoints for jobs, users, bids, etc.

- uploads/: Stores any uploaded files like images or documents related to jobs or user profiles.

- app.js: The entry point to the backend server. It sets up the Express app, connects to the database, and starts the server.

Socket Folder

The socket folder is where real-time communication using Socket.io is managed.

```
socket/
│
├── node_modules/          # Contains dependencies for the socket server.
├── index.js               # Main file where the Socket.io server is initialized, enabling real-time communication.
└── .env                   # Environment variables for configuring the socket server (if needed).
```

- server.js: The main entry point for the Socket.io server. It sets up the WebSocket connections and facilitates real-time messaging between the client and freelancer.

- .env: The configuration file for any environment-specific variables related to the socket server, such as authentication settings for socket connections.

6. Running the Application

To run the Freelancing Application locally, follow the steps below:

Frontend:

1. Navigate to the client directory:

```bash
cd client/
```

2. Install the required dependencies:

```bash
npm install
```

3. Start the frontend server:

```bash
npm start
```

4. The frontend will be available at http://localhost:3000.

Backend:

1. Navigate to the server directory:
```bash
cd server/
```

2. Install the required dependencies:
```bash
npm install
```

3. Start the backend server:
```bash
npm start
```

4. The backend will be available at http://localhost:3001.

Socket Server:

1. Navigate to the socket directory:

```bash
cd socket/
```

2. Install the required dependencies:

```bash
npm install
```

3. Start the socket server:

```bash
npm start
```

4. The socket server will run on the backend, facilitating real-time communication between clients and freelancers.

By following these steps, you'll have the application running locally, with the frontend accessible on port `3000`, the backend on port `3001`, and the socket server enabling real-time communication.

7. API Documentation

The backend exposes several endpoints for managing users, jobs, bids, and messages. Below is the detailed API documentation for each endpoint:

User Authentication

POST /api/auth/register

- Description: Register a new user (freelancer or client).
- Request Body:

```json
{
  "name": "John Doe",
  "email": "johndoe@example.com",
  "password": "password123",
  "role": "freelancer"
}
```

```
- Response:
```json
{
  "message": "User registered successfully"
}
```

POST /api/auth/login

- Description: Login a user and return a JWT token.
- Request Body:
```json
{
  "email": "johndoe@example.com",
  "password": "password123"
}
```
- Response:
```json
{
  "token": "JWT_Token_Here"
}
```
```

Job Management

POST /api/jobs

- Description: Create a new job listing (client only).
- Request Body:
  ```json
  {
    "title": "Web Developer Needed",
    "description": "Looking for an experienced web developer.",
    "budget": 500,
    "skills": ["React", "Node.js"]
  }
  ```
- Response:
  ```json
  {
    "message": "Job posted successfully"
  }
  ```

GET /api/jobs

- Description: Retrieve all job listings.
- Response:
  ```json
  [
    {
      "title": "Web Developer Needed",
      "description": "Looking for an experienced web developer.",
      "budget": 500,
      "skills": ["React", "Node.js"]
    },
    {
      "title": "UI Designer for Mobile App",
      "description": "Seeking a UI designer for an upcoming mobile app project.",
      "budget": 300,
      "skills": ["Figma", "UI/UX"]
    }
  ]
  ```

Bidding

POST /api/bids

- Description: Place a bid on a job (freelancer only).
- Request Body:

```json
{
  "jobId": "job_id_here",
  "amount": 400,
  "message": "I am an experienced developer and can complete this job quickly."
}
```

- Response:

```json
{
  "message": "Bid placed successfully"
}
```

Messages

POST /api/messages

- Description: Send a message to a client or freelancer.
- Request Body:

```json
```

```
{
  "recipientId": "user_id_here",
  "message": "Can you provide more details about the job?"
}
```

- Response:

```json
{
  "message": "Message sent successfully"
}
```

8. Authentication

Authentication Strategy

The authentication system for the freelancing platform is based on JWT (JSON Web Tokens). The platform employs a token-based authentication system to manage user sessions.

- User Registration:
  - When a user registers (either a freelancer or client), the system hashes the password using Bcrypt.
  - The user details are saved in the MongoDB database, and a JWT token is returned for authentication.

- Login:

  - Upon login, the system checks the credentials against the database.

  - If the credentials are valid, the user receives a JWT token, which must be included in the Authorization header of subsequent requests.

- Authorization:

  - The JWT token is used to authenticate and authorize users for protected routes.

  - The backend verifies the token using a secret key stored in the environment variables (JWT_SECRET).

- Session Management:

  - The frontend stores the JWT token in the browser's localStorage or sessionStorage, which is sent with every request to the backend to secure access to protected routes.

By utilizing this token-based authentication strategy, the platform ensures that only authorized users can access certain features and manage their account securely.

9. User Interface

UI Features:

The freelancing platform is designed with a clean, modern interface focused on enhancing usability for both clients and freelancers. Below are some of the key components of the user interface:

- Homepage:

  - The homepage offers a clear overview of the platform, featuring options for both registering and logging in. It provides a seamless entry point for new users and returning users.

- Dashboard:

  - The dashboard provides a personalized view for each user (client or freelancer), allowing them to manage their jobs, bids, and messages efficiently. Key sections include a summary of active jobs, placed bids, and real-time communication.

- Job Listings:

  - This section displays a list of all available jobs, with useful filters to help users easily navigate through different job categories, budgets, and skills. Freelancers can browse job opportunities and decide where to place their bids.

- Profile Management:

  - Users can update their profile with essential details, including skills, experience, and portfolio. This section helps freelancers showcase their capabilities, while clients can maintain their company and project details.

Screenshots:

1. Homepage of the freelancing platform:

   - The homepage gives a first impression with clear call-to-action buttons like "Register" and "Login."

2. Job Listings Page:

- The job listings page displays a dynamic list of available projects with detailed information, such as job title, description, budget, and required skills.

10. Testing

Testing Strategy:

Testing is essential to ensure the platform performs as expected, providing a smooth user experience and maintaining security and reliability. Various testing strategies and tools were employed to guarantee the platform's functionality.

Unit Testing:

- React Testing Library and Jest were used for testing frontend components. Key UI components like JobCard, BidForm, and ProfileForm were thoroughly tested to ensure they render correctly and behave as expected during user interactions.

- Unit tests focused on checking the basic functionality of each component, such as ensuring that form inputs update correctly, job details display as expected, and bids can be submitted without errors.

API Testing:

- Mocha and Chai were employed for backend testing to ensure API endpoints work correctly.

- Tests were created for key routes, including:
  - User registration: Ensuring valid input creates a user and invalid input triggers an error.
  - Job posting: Verifying that jobs are correctly added to the database.
  - Bid submission: Ensuring that freelancers can place bids and that the system responds with appropriate success or error messages.

Integration Testing:

- Full-Flow Testing: We tested the integration of frontend and backend to ensure that the user experience flows smoothly from job creation to bid submission and messaging.

- Mock Data: To simulate real user interaction, mock data was used to test the entire process, including job creation, placing bids, and sending messages. This ensured that each part of the application works together as expected.

Performance Testing:

- Tools like Lighthouse were utilized to assess the platform's performance. These tests focused on page load times, responsiveness, and overall efficiency.

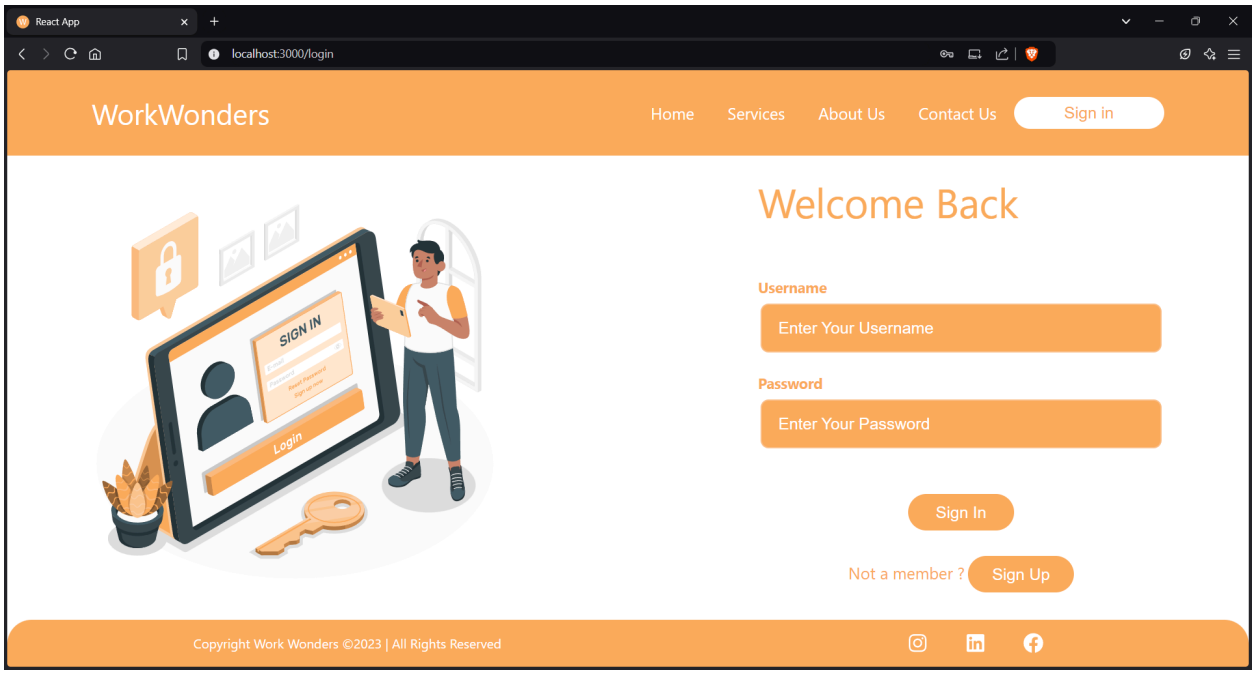
- Performance testing helped ensure that the platform is optimized for speed, even with large numbers of users and interactions, and that the UI remains responsive on both desktop and mobile devices.

By utilizing a comprehensive testing strategy, the platform ensures a smooth, error-free user experience and robust backend functionality.

11. Screenshots

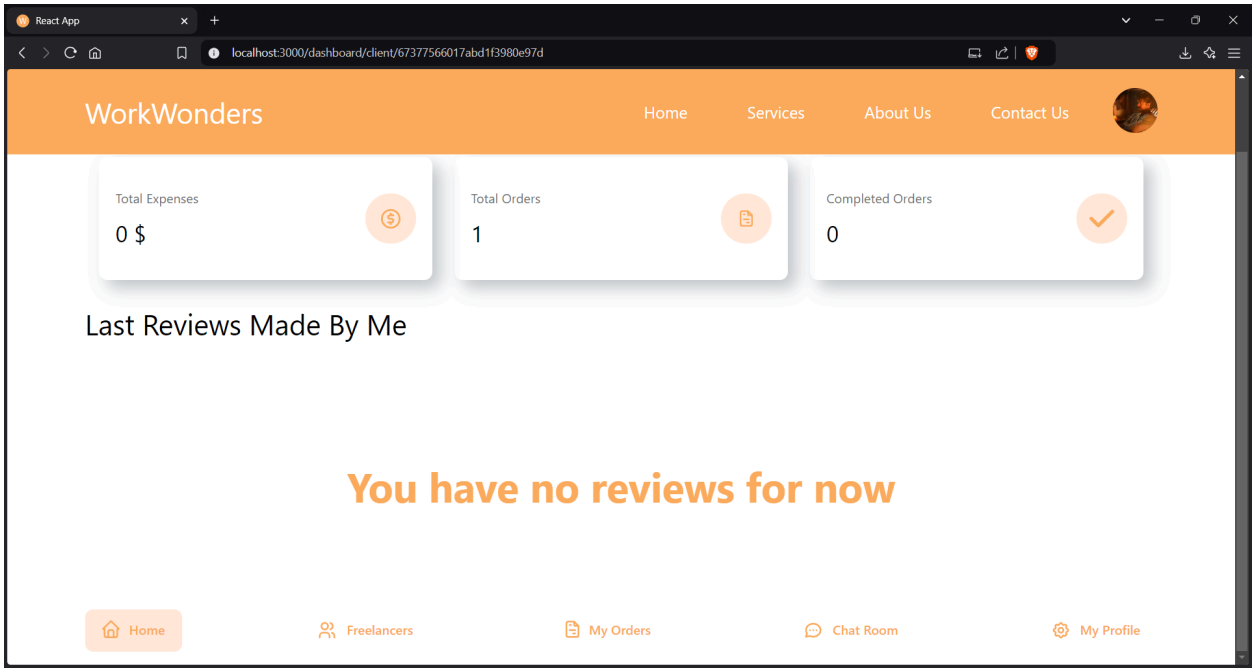
Screenshots:

The screenshots below highlight key features of the freelancing platform:



1. Login Page:

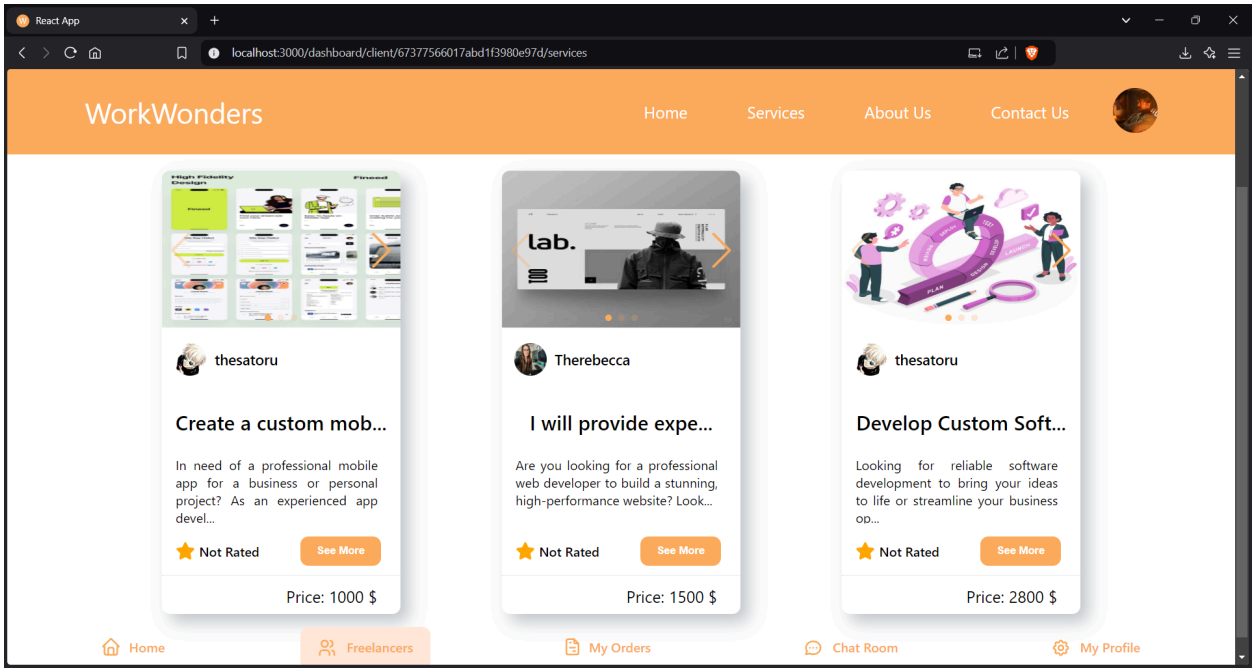
- The login page provides users with a straightforward interface for signing into their accounts with their credentials.



2. User Dashboard:

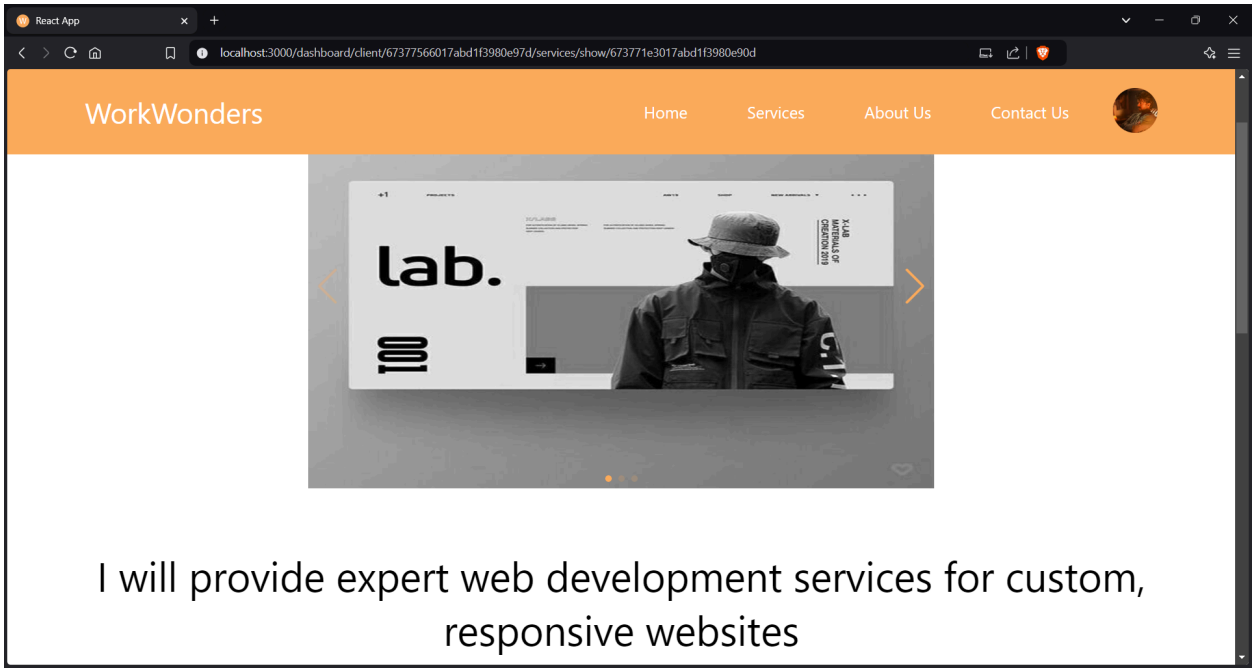
- The user dashboard offers a personalized experience, where users can manage their active jobs, placed bids, and ongoing communications.

These screenshots provide a visual overview of the platform's layout and functionality.



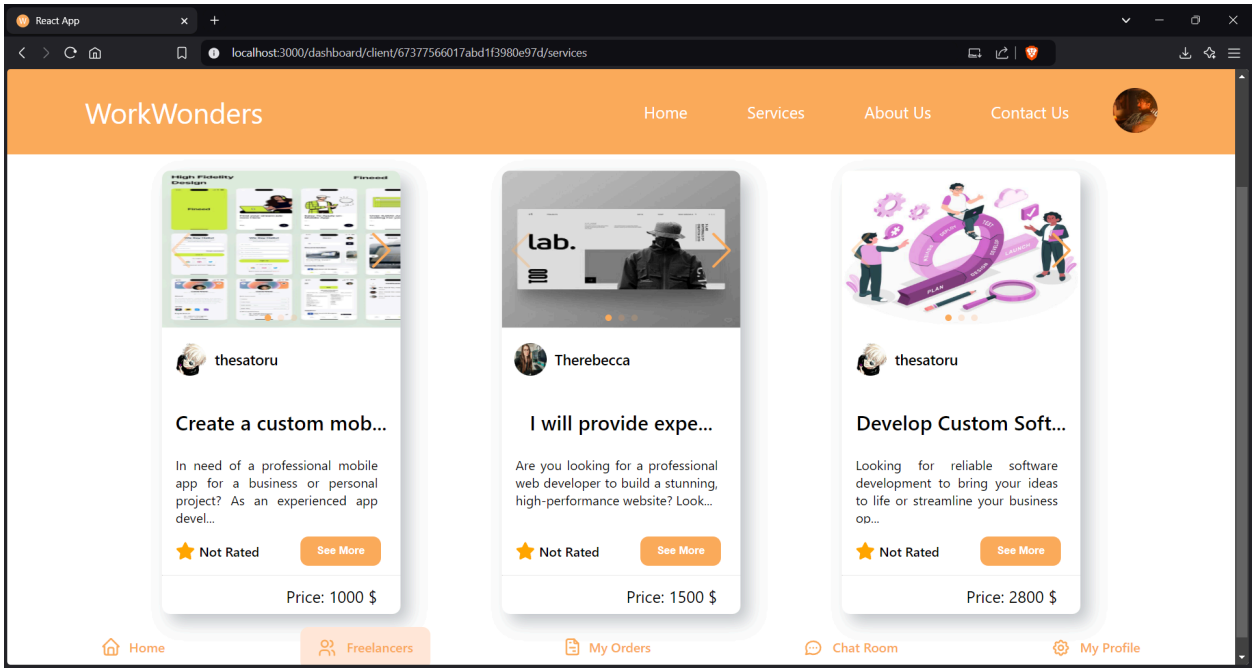
3. Job Listings Page:

- Displays all the available jobs posted by clients. Freelancers can filter jobs based on various criteria like skills, budget, and category.



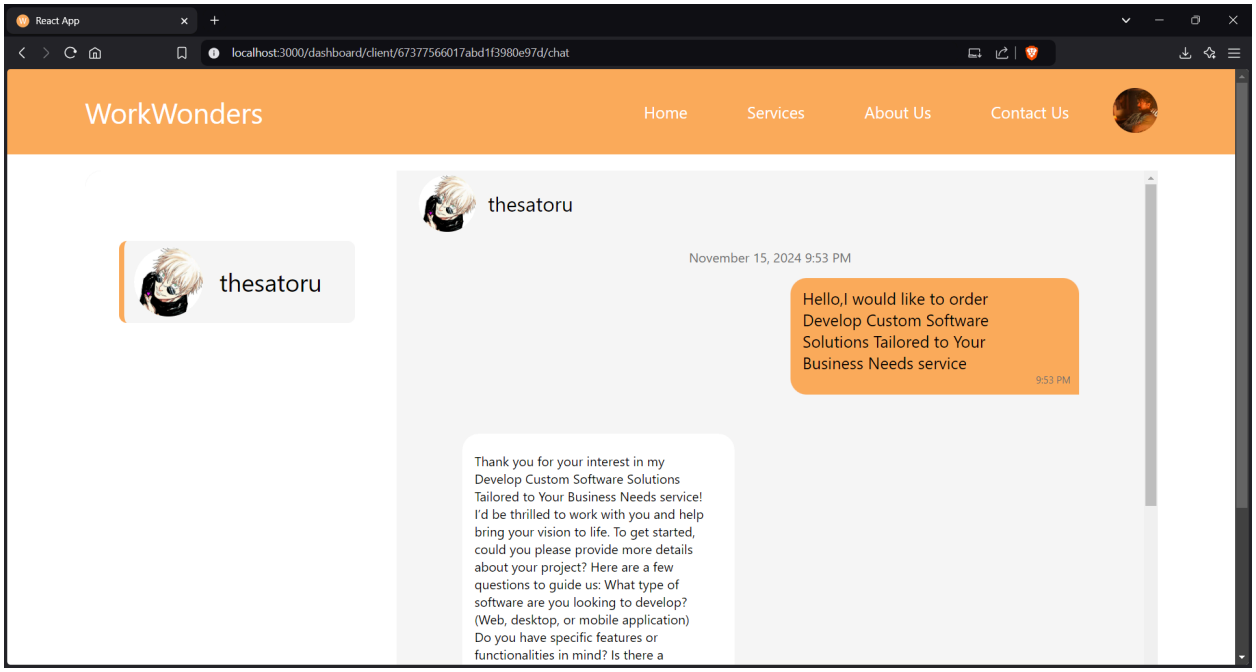
4. Job Detail Page:

- Provides detailed information about a specific job listing, including job description, budget, required skills, and client information. Freelancers can view the job details and place their bids.



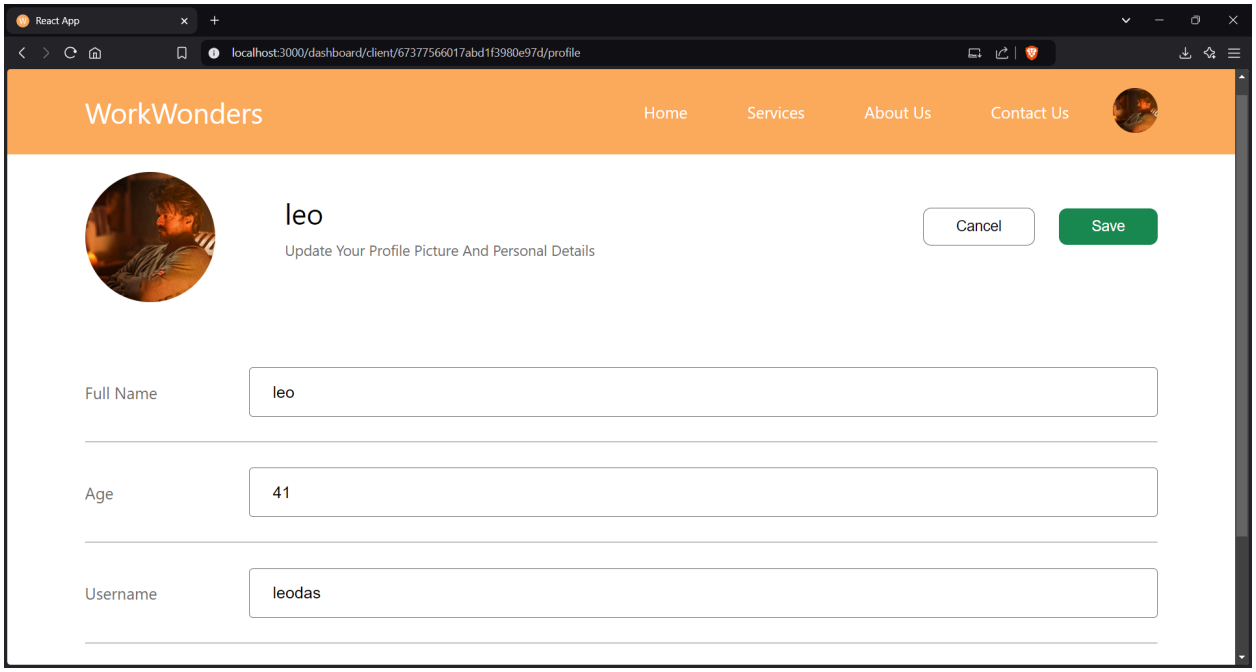
5.Bidding Page:

- Displays job details and allows freelancers to place bids.



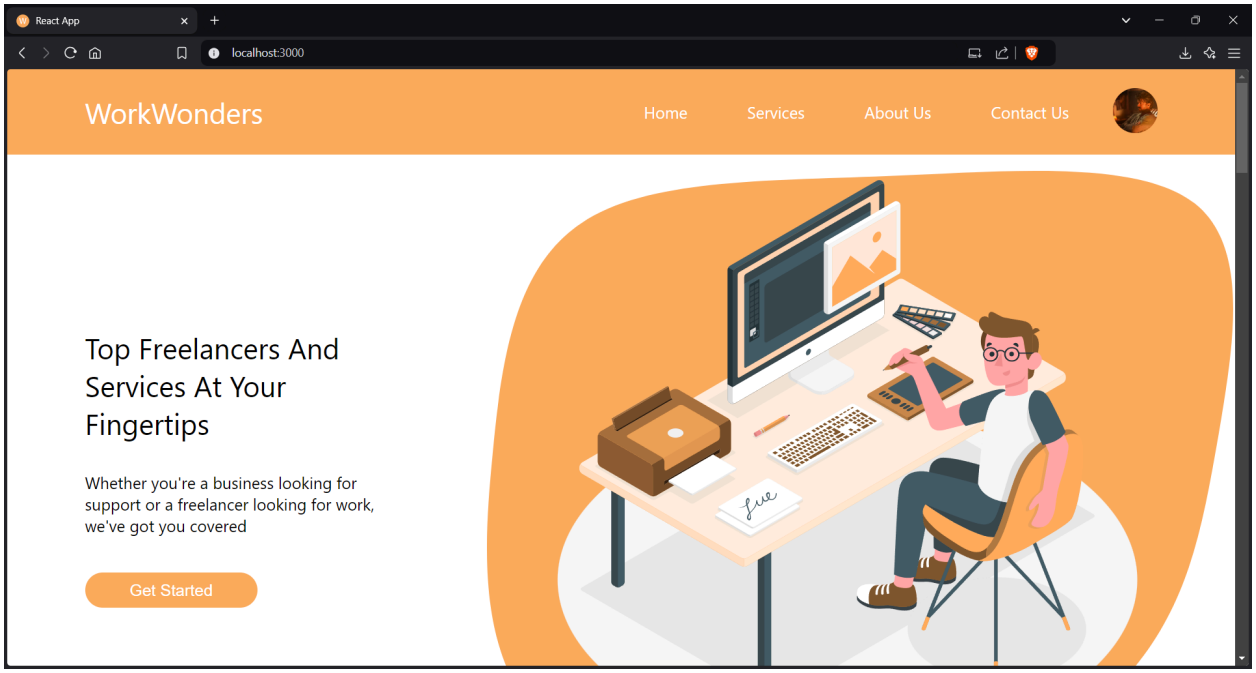
6. Chatroom Page:

- Allows real-time communication between clients and freelancers, enabling them to discuss job details and updates.



7. User Profile Page:

- Displays user information, including skills, experience, portfolio, and personal details, allowing users to edit and update their profiles.



8. Homepage:

- Displays an overview of the platform with options to register or log in, highlighting key features like job listings and available freelancer services.

12. Known Issues

1. Messaging Delay:

- Occasionally, users may experience a slight delay in message delivery during real-time chat. This issue is related to network latency or server-side optimizations and is actively being worked on to improve responsiveness and reduce delay.

2. Job Filter:

- There have been some reports regarding the accuracy of job results when applying multiple filters (e.g., budget, skills, and job types). Currently, the filter system is being optimized to improve its accuracy and functionality.

3. Mobile Responsiveness:

- While the platform follows a mobile-first design principle, some UI components may display misalignment or overflow on extremely small devices or unconventional screen sizes. This issue is being addressed in the upcoming update to enhance mobile responsiveness across all device sizes.

13. Future Enhancements

1. Payment Integration:

- We plan to integrate a secure payment gateway, such as Stripe, to facilitate transactions between freelancers and clients. This will ensure that payments for completed projects are processed smoothly and securely.

2. Advanced Filtering:

- The job search functionality will be enhanced with additional filtering options, such as:
  - Job Type (Full-time, Part-time, Contract, etc.)
  - Location-based Search
  - Skill-based Search
  - Experience Level Filtering
- These improvements will provide users with a more refined and personalized job search experience.

3. Rating and Reviews:

- A rating and review system will be implemented, enabling freelancers and clients to provide feedback after job completion. This will help maintain quality standards, improve trust within the community, and assist users in choosing reliable collaborators.

4. Notification System:

- Push notifications will be added to notify users about:
  - New job postings
  - Bid status updates (accepted, rejected, etc.)
  - Incoming messages
  - Job status updates (such as project completion)
- This will improve user engagement and ensure timely updates for all actions on the platform.

5. Admin Dashboard:

- An enhanced Admin Panel will be developed, providing administrators with the following capabilities:

  - User Management: Ability to view, suspend, or ban users if they violate platform rules.

  - Activity Monitoring: A real-time activity tracker that monitors platform traffic, active jobs, and user interactions.

  - Reporting and Analytics: A comprehensive dashboard for generating reports and analyzing platform performance and usage.

- This update will provide more control over platform operations and enhance overall user safety and experience.

Conclusion:

In conclusion, the development of this freelancing platform using the MERN stack has been a successful demonstration of the power and flexibility of modern web technologies. By leveraging React, Node.js, Express.js, and MongoDB, the platform provides a responsive, scalable, and secure environment for both freelancers and clients. The integration of key features such as job posting, bidding, messaging, and real-time communication has created a seamless user experience, enabling efficient collaboration between freelancers and clients.

Result:

The app has been successfully developed using the required software and technologies.