|  |  |  |
| --- | --- | --- |
|  | **MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE**  **Kodambakkam, Chennai-600024**  **(An Autonomous Institution)** |  |

**NM1042 – MERN STACK POWERED BY MONGODB**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**PROJECT TITLE: FREELANCE APPLICATION PLATFORM**

|  |  |
| --- | --- |
| **TEAM ID:** | **NM2024TMID09575** |
| **FACULTY MENTOR:** | **Mrs. M. Sowmiya** |

**Project submitted by,**

| **NAAN MUDHALVAN ID** | **NAME** | **REGISTER NUMBER** |
| --- | --- | --- |
| D4B8AD612A84701B72E47CD18807C724 | *SAI SAKTHI.U* | *311521104046* |
| 4F9548F1BB9894DAA944C7A38E16CDAE | *RITHIKA.G* | *311521104040* |
| 3CC9C8BD25FB307AC3244B01B724D194 | *PREETHAA.U* | *311521104033* |
| 757620247D79C20864D42938DF613EA4 | *IVANA STEEVE* | *311521104020* |

ANNA UNIVERSITY: CHENNAI 600 025

**BONAFIDE CERTIFICATE**

Certified that this project report “**Freelance Application** **Platform**” is the bonafide work of “**Sai Sakthi.U (311520104046), Rithika.G (311521104040), Preethaa.U (311521104033), Ivana Steeve (311521104020)**” Naan Mudhalvan Team ID “**NM2024TMID09575**” who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

Mrs. M. Sowmiya, M.E. Dr.S.Aarthi,M.E., Ph.D.

**ASSISTANT PROFESSOR HEAD OF THE DEPARTMENT**

Computer Science and Engineering

Meenakshi Sundararajan Engineering College

No. 363, Arcot Road, Kodambakkam, Chennai -600024

(An Autonomous Institution)

Computer Science and Engineering

Meenakshi Sundararajan Engineering College

No. 363, Arcot Road, Kodambakkam, Chennai -600024

(An Autonomous Institution)

Submitted for the project viva voce of Bachelor of Engineering in Computer Science and Engineering held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

# INTERNAL EXAMINER EXTERNAL EXAMINER

**ACKNOWLEDGEMENT**

First and foremost, we express our sincere gratitude to our Respected Correspondent **Dr. K. S. Lakshmi**, our beloved Secretary **Mr. N. Sreekanth**, Principal **Dr. S. V. Saravanan** for their constant encouragement, which has been our motivation to strive towards excellence.

Our primary and sincere thanks goes to **Dr. S. Aarthi,** Head of the Department, Department of Computer Science and Engineering, for her profound inspiration, kind cooperation and guidance.

We are grateful to **Mrs. M. Sowmiya**, Internal Guide, Assistant Professor as our project coordinator for their invaluable support in completing our project. We are extremely thankful and indebted for sharing expertise, and sincere and valuable guidance and encouragement extended to us.

Above all, we extend our thanks to God Almighty without whose grace and blessings it would not have been possible.

# ABSTRACT

The Freelancing Application is a full-stack web platform developed using the MERN stack, which comprises MongoDB for data persistence, Express.js for server-side logic, ReactJS for crafting an interactive user interface, and Node.js for backend services. This application is meticulously designed to connect freelancers with clients who require professional services, creating an efficient and user-centric marketplace. It serves as a bridge for individuals and businesses seeking specialized talent for diverse projects and offers freelancers opportunities to showcase and monetize their skills.The platform emphasizes a seamless and highly interactive experience through its range of essential functionalities, which include comprehensive profile management for both freelancers and clients, job posting capabilities, an intuitive bidding system, real-time project tracking dashboards, secure and encrypted communication channels, and integrated payment systems. This holistic approach not only simplifies the hiring and collaboration process but also ensures a high level of transparency and trust between all participants.

By leveraging modern web technologies and best practices in full-stack development, this application aims to deliver an intuitive, efficient, and highly secure experience. It addresses the core needs of the gig economy by offering robust mechanisms for creating long-term professional relationships, fostering productivity, and ensuring that both freelancers and clients can achieve their respective goals with ease and confidence. The scalable architecture and modularity of the MERN stack further allow for future enhancements and integrations, making the platform adaptable to evolving market demands and user expectations.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT** | iv |
|  | **LIST OF ABBREVIATIONS AND**  **EXPANSIONS** | viii |
| **1.** | **INTRODUCTION** | **1** |
|  | 1.1 TEAM DETAILS | 1 |
|  | 1.2 ABOUT THE PROJECT | 1 |
| **2.** | **PROJECT OVERVIEW** | 2 |
|  | 2.1 PURPOSE | 3 |
|  | 2.2 FEATURES | 4 |
| **3.** | **SYSTEM ARCHITECTURE** | 8 |
|  | 3.1 FRONTEND | 8 |
|  | 3.2 BACKEND | 12 |
|  | 3.3 DATABASE | 16 |
| **4.** | **SETUP INSTRUCTIONS** | 22 |
|  | 4.1 PREREQUISITES | 22 |
|  | 4.2 INSTALLATION | 23 |
| **5.** | **FOLDER STRUCTURE** | 26 |
|  | 5.1 CLIENT: REACT FRONTEND STRUCTURE | 26 |
|  | 5.2 SERVER: NODE.JS BACKEND STRUCTURE | 27 |
| **6.** | **RUNNING THE APPLICATION** | 29 |
|  | 6.1 FRONTEND | 29 |
|  | 6.2 BACKEND | 29 |
| **7.** | **API DOCUMENTATION** | 31 |
| **8.** | **AUTHENTICATION** | 39 |
| **9.** | **USER INTERFACE** | 43 |
| **10.** | **TESTING** | 45 |
| **11.** | **SCREENSHOTS OR DEMO** | 46 |
| **12.** | **KNOWN ISSUES** | 53 |
| **13.** | **FUTURE ENHANCEMENTS** | 54 |
| **14** | **CONCLUSION** | 55 |

**LIST OF ABBREVIATIONS AND EXPANSIONS**

| **ABBREVIATIONS** | **NAME OF THE FIGURE** |
| --- | --- |
| API | Application Programming Interface |
| CRUD | Create, Read, Update, and Delete |
| JSON | JavaScript Object Notation |
| JWT | JSON Web Tokens |
| MERN | MongoDB, Express, React, and Node |
| NPM | Node Package Manager |
| REST | REpresentational State Transfer |
| SPA | Single-page Application |
| UI | User Interface |
| URI | Uniform Resource Identifier |
| URL  UX  HTTPS  SSL | Uniform Resource Locator  User Experience  Hypertext Transfer Protocol  Secure Sockets Layer |

# CHAPTER 1

# INTRODUCTION

* 1. **TEAM DETAILS**

**Project Title:** Freelance Application

**Team ID:** NM2024TMID09575

**Team Members:**

Sai Sakthi.U – Backend Developer

Rithika.G – Backend Developer

Preethaa.U –Frontend Developer

Ivana Steeve – Frontend Developer

* 1. **ABOUT THE PROJECT**

The Freelancing Application is a feature-rich web platform designed to connect freelancers with clients, enabling seamless collaboration and project management in the modern gig economy. Developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), the application caters to the needs of both freelancers and clients by providing a robust, intuitive, and secure environment.

Freelancers can create detailed profiles to showcase their skills and expertise, bid on projects, and manage their tasks efficiently. Clients, on the other hand, can post job requirements, review freelancer profiles, and hire professionals to fulfill their project needs. The platform ensures smooth interactions through features such as real-time messaging, milestone tracking, and integrated payment gateways, offering an end-to-end solution for freelancing activities.

The backend leverages Node.js and Express.js to create a scalable and secure API that communicates with MongoDB for storing and retrieving data, including user profiles, projects, bids, and transactions. The React.js-powered frontend delivers a dynamic and user-friendly interface, optimized for responsiveness across all devices, ensuring an engaging experience for users. Authentication and authorization are implemented using JWT (JSON Web Tokens) to protect user accounts and data, with role-based access control (RBAC) to regulate permissions.

This project addresses the challenges faced by freelancers and clients in finding, communicating, and collaborating effectively. It offers a scalable, efficient, and user-centric solution for managing freelancing operations, making it an ideal tool for individuals and organizations seeking to thrive in the competitive freelancing marketplace.

# 

# CHAPTER 2

# PROJECT OVERVIEW

The Freelancing Application is a full-stack web platform built using the MERN stack (MongoDB, Express.js, React, and Node.js) to facilitate seamless collaboration between freelancers and clients. The application provides a comprehensive solution for job posting, bidding, project management, and secure payments, making it an essential tool for both freelancers seeking opportunities and clients looking for skilled professionals.

The platform features an intuitive and responsive user interface developed using React, enabling users to create profiles, browse projects, submit proposals, and manage ongoing tasks with ease. State management is handled using Redux, ensuring smooth and consistent data flow, particularly for real-time updates in features like messaging and bidding. The backend, powered by Node.js and Express.js, offers RESTful APIs that handle all core functionalities, including user registration, authentication, job postings, and payment processing. JWT (JSON Web Tokens) ensures secure login and role-based access control (RBAC) for both freelancers and clients, safeguarding sensitive user data and permissions.

MongoDB serves as the database for the application, efficiently managing complex data structures such as user profiles, project details, bids, and transactions. The modular architecture of the system allows for scalability and easy integration of additional features, with potential enhancements including AI-driven recommendations, mobile app development, and multi-language support.

Overall, the **Freelancing Application** demonstrates the MERN stack's capability in building modern, feature-rich, and scalable platforms. It addresses the growing needs of the freelancing ecosystem, offering a secure and efficient digital marketplace for individuals and businesses to collaborate and succeed.

**2.1 PURPOSE**

The purpose of the Freelancing Application project is to create a seamless, efficient, and accessible digital platform that connects freelancers with clients seeking professional services. As the demand for remote work and freelancing continues to grow, this application aims to simplify the process of finding and managing freelance opportunities by enabling users to post job listings, submit bids, communicate effectively, and manage projects all within a single platform.

The project is also designed to enhance the operational efficiency of both freelancers and clients. Freelancers can showcase their skills, apply for suitable projects, and manage their work, while clients can easily find qualified professionals, oversee project progress, and ensure secure payment transactions. By integrating features such as real-time communication, profile management, and milestone tracking, the platform delivers a streamlined experience tailored to the needs of all users.

With a secure authentication system, intuitive user interface, and scalable architecture, the **Freelancing Application** strives to reduce the time, effort, and barriers typically associated with traditional freelancing. Ultimately, the goal is to empower users to collaborate effectively, achieve professional growth, and meet project requirements in a transparent, reliable, and user-friendly environment.

**Goals:**

1. **User-Friendly Interface**: Create an intuitive and visually appealing interface where users can easily navigate through the platform, manage profiles, browse job postings, and track project updates effortlessly.
2. **Efficient Job Management**: Develop robust functionality for clients to post job requirements and freelancers to submit bids, ensuring a smooth process from project initiation to completion.
3. **Secure Authentication System**: Implement secure user authentication using JWT to protect user data and provide role-based access control (RBAC) for freelancers and clients.
4. **Real-Time Communication:** Facilitate seamless communication between freelancers and clients through an integrated messaging system to enhance collaboration and resolve queries promptly.
5. **Responsive Design:** Ensure the platform is fully responsive, delivering an optimal experience on desktops, tablets, and mobile devices for all users.
6. **Integrated Payment System:** Build a secure and reliable payment gateway for handling transactions, including project milestones and final payments, to ensure transparency and trust.
7. **Data Management:** Develop a robust backend that securely manages user data, job postings, bids, and project updates while maintaining scalability for future expansion.
8. **Search and Filtering Options:** Provide advanced search and filtering options to help clients find the right freelancer and freelancers to locate projects that match their expertise quickly.
9. **Scalability and Future Expansion:** Design the application to support a growing user base and new features such as AI-based project recommendations, multilingual support, and mobile app integration.
10. **Milestone Tracking and Reporting:** Develop a robust backend that securely manages user data, job postings, bids, and project updates while maintaining scalability for future expansion.

**2.2 FEATURES**

**1.** **User Account Features:**

* Registration and Login: Users can sign up with an email and password and log in securely using authentication mechanisms such as JWT.
* Profile Management: Freelancers and clients can manage their profiles, update personal information, and upload profile pictures or portfolio details.
* Password Recovery: Offers a secure "Forgot Password" option, enabling users to reset their credentials via email verification.

**2.** **Freelancer and Client Features::**

* Job Posting: Clients can create and post detailed job requirements, specifying budgets, timelines, and skillsets needed.
* Bidding System: Freelancers can bid on available jobs, providing competitive offers and personalized proposals.
* Project Management: Users can view project status updates, milestones, and deadlines on a dedicated dashboard.

**3.** **Real-Time Communication:**

* Messaging System: Enables freelancers and clients to interact in real-time, facilitating clear and effective communication throughout the project lifecycle.
* Notifications: Sends timely alerts for bid approvals, project updates, and messages to keep users informed.

**4.** **Payment and Transactions:**

* Integrated Payment Gateway: Provides secure options for handling project payments, including milestone-based payments.
* Transaction History: Users can view past transactions, invoices, and payment details in their profiles.

**5. Search and Filtering Options:**

* Job Search: Freelancers can search for projects using filters like budget, category, or location.
* Freelancer Search: Clients can use advanced filters to find freelancers based on ratings, skills, and availability.

**6.** **Admin Dashboard Features:**

* User Management: Admins can manage user accounts, handle disputes, and monitor platform activities.
* Job Moderation: Admins have tools to approve, edit, or delete job postings if they violate policies.
* Financial Oversight**:** Provides visibility into platform-wide transactions and revenue reports.

**7.**  **Frontend Features:**

* Responsive Design: Ensures the platform adapts to all screen sizes, offering a seamless experience across mobile, tablet, and desktop devices.
* Dynamic Components: Built using reusable React components for maintainability and scalability.

**8.** **Backend Functionalities:**

* Role-Based Access Control (RBAC): Differentiates permissions for freelancers, clients, and admins.
* API Endpoints: Securely handles data requests and responses for job postings, user authentication, and transactions.
* Error Handling: Comprehensive error logging and feedback mechanisms improve system reliability.

**9. Security Features:**

* Token-Based Authentication: Ensures secure login sessions for users with JWT.
* Data Encryption: Protects sensitive user data, including passwords and payment details, during transmission and storage.
* Protected Routes: Implements access restrictions for specific routes based on user roles and authentication status.

**10. Scalability and Future Enhancements:**

* Modular Architecture: Facilitates future expansions and integration of new features like AI-driven job recommendations.
* Third-Party Integrations: Ready for integrating external APIs such as cloud storage, analytics tools, or advanced payment gateways.
* Performance Optimization: Implements caching and efficient database queries to handle high traffic effectively.

# 

# CHAPTER 3

# SYSTEM ARCHITECTURE

**3.1 FRONTEND**

**Frontend:**

### 1. **Modern Routing in a Freelancer Application**

* **Dynamic Routing**:
  + Use [id].js in Next.js for pages like job details (/jobs/[id]) or profile details (/profiles/[id]).
  + Example: Dynamic routes can be utilized for freelancer portfolios, enabling each portfolio to be accessed by a unique URL.
* **Server-Side Rendering (SSR)**:
  + Leverage SSR for public pages like freelancer profiles, job listings, and landing pages to improve SEO and enhance first-page load speed.
* **Protected Pages**:
  + Middleware can dynamically check the user's role (admin, freelancer, client) to allow or restrict access to pages like /admin/dashboard or /freelancer/jobs.

### 2. **State Management Alternatives**

* **Zustand or Recoil**:
  + Simplify global state management for features like user sessions or global filters for job searches.
* **React Query**:
  + Ideal for managing server state, such as real-time job bids, notifications, or chat messages.

### 3. **Backend Communication with Modern APIs**

* **Fetch API + SWR**:
  + Lightweight, efficient data fetching and caching for pages like job listings or freelancer search.
* **GraphQL**:
  + Query only the necessary fields for high-relationship data, such as job details, freelancer reviews, and transaction histories.

### 4. **UI/UX Enhancements**

* **Headless UI**:
  + Build accessible modals for submitting bids or dropdowns for filtering job types.
* **CSS-in-JS (Emotion)**:
  + Style individual freelancer cards dynamically based on availability or rating.
* **Figma Integration**:
  + Directly import Figma designs to ensure the implemented UI matches the designer’s wireframes.

### 5. **Custom Hooks & Utilities**

* **Examples**:
  + useJobFetch: Fetch job details with error handling.
  + useBidManagement: Handle adding, updating, and deleting bids dynamically.
  + useUserRole: Dynamically manage access permissions based on roles.

### 6. **Authentication & Authorization**

* **NextAuth.js**:
  + Seamless OAuth integration (e.g., Google or GitHub) for freelancers and clients.
  + Secure session handling for logged-in users.
* **RBAC**:
  + Use libraries like casl to allow only freelancers to bid on projects and only clients to post jobs.

### 7. **Error Handling and Feedback**

* **Error Monitoring**:
  + Track and resolve application errors with Sentry.
* **User Notifications**:
  + Use React Hot Toast for smooth in-app alerts (e.g., "Bid submitted successfully" or "Profile updated").

### 8. **Performance Optimizations**

* **Image Optimization**:
  + Optimize freelancer profile pictures and job-related images for better load times.
* **PWA**:
  + Enable offline mode for job browsing using service workers.
* **Edge Caching**:
  + Deploy on platforms like Vercel to improve global performance.

### 9. **Testing and Quality Assurance**

* **Modern Testing Tools**:
  + Use **Vitest** for unit tests of critical components like job cards or bid submission forms.
  + Replace Cypress with Playwright for robust E2E testing of workflows like job posting or bidding.
* **Code Quality**:
  + Use ESLint and Prettier to enforce consistent code formatting.
  + Integrate Husky hooks for pre-commit checks to maintain standards.

**3.2 BACKEND**

**Backend**:

**1. Server and Application Setup**

* **Express Server**: Express is an excellent choice due to its simplicity and extensive middleware ecosystem. It will power REST APIs that freelancers and clients interact with.
* **Environment Configuration**: .env securely manages sensitive information such as API keys, database URIs, and JWT secrets, ensuring portability across environments.
* **Middleware**:
  + **Body Parser**: Parses JSON payloads for APIs handling user profiles, job postings, or messages.
  + **CORS**: Ensures that the frontend hosted on a different domain can interact with the backend.
  + **Logging**: Logs key details like request paths, response times, and errors for debugging.

### **2. Directory Structure**

* **controllers/**: Houses logic for freelancers (registration, portfolio management), clients (job postings), and admin functionalities.
* **models/**: Includes schemas for freelancers, clients, projects, payments, and reviews.
* **routes/**: Separates APIs for users, projects, and admin roles.
* **middlewares/**: Authentication (JWT-based), authorization (role-based access), and error handling.
* **config/**: Database connection and API configurations.
* **validations/**: Joi schemas validate payloads for creating jobs, bids, or reviews.

### **3. Database Layer**

* **MongoDB with Mongoose**:
  + **Freelancer Model**: Stores profile details (skills, portfolio, hourly rate, availability).
  + **Client Model**: Contains company details, posted jobs, and hiring history.
  + **Project Model**: Tracks the job lifecycle—description, bids, milestones, deadlines, and completion status.

### **4. Authentication and Authorization**

* **JWT**:
  + Frees the need for server-side session management. Ideal for mobile and web clients.
* **Role-Based Access Control**:
  + **Admin**: Manages users, disputes, and payment escalations.
  + **Freelancers**: Can update profiles, submit proposals, and withdraw earnings.
  + **Clients**: Post jobs, review proposals, and release payments.

### **5. Controllers**

* **Freelancer Controller**: Manages registration, portfolio updates, and bid submissions.
* **Client Controller**: Handles job postings, proposal reviews, and hiring actions.
* **Project Controller**: Tracks project milestones, completion, and disputes.

### **6. Routes**

* **User Routes**:
  + **POST /register**: User signup.
  + **POST /login**: User authentication.
  + **GET /profile**: Fetches profile details (protected route).
* **Job Routes**:
  + **POST /jobs**: Allows clients to post jobs (Admin only).
  + **GET /jobs**: Lists available jobs for freelancers.
* **Project Routes**:
  + **GET /projects**: Freelancer views assigned projects.
  + **PUT /projects/:id**: Updates project status (e.g., completed).
* **Payment Routes**:
  + **POST /payments**: Records payments against milestones.

### **7. Middleware**

* **Authentication**: Protects endpoints with JWT.
* **Authorization**: Grants role-based permissions for sensitive actions.
* **Error Handling**: Logs and returns user-friendly error messages.

### **8. Data Validation**

* + Validates registration details, job postings, and bid inputs.

### **9. Utility Functions**

* **Helper Functions**: Handle recurring calculations, like freelancer earnings and project timelines.
* **Error Utilities**: Define custom exceptions for unauthorized access or validation failures.

### **10. Security Considerations**

* **Password Hashing**: Uses bcrypt for secure password storage.
* **Rate Limiting**: Prevents brute-force attacks on login routes.
* **Data Sanitization**: Mitigates injection attacks by sanitizing inputs.

### **11. Deployment and Scalability**

* **Environment-Specific Configurations**: Eases the transition between development, staging, and production environments.
* **Containerization**:
  + Docker ensures consistency in deployment.
  + Load balancers like NGINX can manage traffic spikes.

### **12. Load Balancing and Scaling**

* Cloud services (AWS, Azure, Google Cloud) allow for auto-scaling and load balancing to accommodate high traffic, ensuring smooth operation.

**3.3 DATABASE**

#### ****1. Users Collection****

* **Purpose**: To store freelancer and client profiles, including roles and authentication details.
* **Fields**:
  + \_id (ObjectId): Unique identifier for each user.
  + name (String): Full name of the user.
  + email (String): Unique identifier for login.
  + password (String): Hashed password.
  + role (String): Defines if a user is a "freelancer" or "client".
  + profile (Object): Contains additional details:
    - bio (String): Short bio for freelancers or clients.
    - skills (Array of Strings): Key skills for freelancers.
    - portfolio (Array): Links to portfolio samples (freelancers only).
  + createdAt (Date): Account creation timestamp.
  + updatedAt (Date): Timestamp for profile updates.
* **Interactions**:
  + Registration and login workflows for both freelancers and clients.
  + Role-based access control for freelancers (project bidding) and clients (project posting).

#### ****2. Projects Collection****

* **Purpose**: To manage project postings by clients and bids by freelancers.
* **Fields**:
  + \_id (ObjectId): Unique identifier for each project.
  + title (String): Name of the project.
  + description (String): Detailed project requirements.
  + clientId (ObjectId): References the users collection (client who posted the project).
  + budget (Number): Client's estimated budget.
  + category (String): Project category (e.g., "Web Development", "Design").
  + status (String): Current status (e.g., "open", "in progress", "completed").
  + createdAt (Date): Timestamp for project creation.
  + bids (Array of objects): Contains freelancer bids:
    - freelancerId (ObjectId): References the users collection.
    - bidAmount (Number): Amount proposed by the freelancer.
    - bidMessage (String): Proposal details or message from the freelancer.
* **Interactions**:
  + Clients create and update project postings.
  + Freelancers submit bids on open projects.
  + Clients award projects to specific freelancers.

#### ****3. Contracts Collection****

* **Purpose**: To manage the relationship between freelancers and clients for awarded projects.
* **Fields**:
  + \_id (ObjectId): Unique identifier for the contract.
  + projectId (ObjectId): References the projects collection.
  + clientId (ObjectId): References the users collection.
  + freelancerId (ObjectId): References the users collection.
  + startDate (Date): Date when the project work begins.
  + endDate (Date): Agreed deadline for completion.
  + status (String): Current status of the contract (e.g., "active", "completed").
  + paymentStatus (String): Payment state (e.g., "pending", "paid").
  + amount (Number): Agreed project payment amount.
* **Interactions**:
  + Tracks awarded projects.
  + Allows clients to release payments upon project completion.
  + Facilitates contract updates, such as extending deadlines.

#### ****4. Transactions Collection****

* **Purpose**: To manage payments between freelancers and clients.
* **Fields**:
  + \_id (ObjectId): Unique transaction ID.
  + contractId (ObjectId): References the contracts collection.
  + clientId (ObjectId): References the users collection.
  + freelancerId (ObjectId): References the users collection.
  + amount (Number): Payment amount.
  + status (String): Payment status (e.g., "successful", "failed").
  + createdAt (Date): Transaction timestamp.
* **Interactions**:
  + Logs payments for awarded contracts.
  + Tracks payment status for both freelancers and clients.

#### ****5. Reviews Collection****

* **Purpose**: To collect feedback for freelancers and clients post-project.
* **Fields**:
  + \_id (ObjectId): Unique identifier for each review.
  + contractId (ObjectId): References the contracts collection.
  + reviewerId (ObjectId): References the users collection (client or freelancer).
  + reviewedId (ObjectId): References the users collection (freelancer or client).
  + rating (Number): Rating score (e.g., 1 to 5).
  + comments (String): Feedback text.
  + createdAt (Date): Timestamp for review submission.
* **Interactions**:
  + Clients review freelancers upon project completion and vice versa.
  + Builds a reputation system for freelancers and clients.

### **Common MongoDB Operations**

1. **Project Search and Filters**:
   * Freelancers search for projects by category or budget:

db.projects.find({ category: "Web Development", status: "open" }).sort({ budget: -1 });

1. **Aggregated Insights for Admins**:
   * Calculate total earnings per freelancer:

db.transactions.aggregate([

{ $match: { status: "successful" } },

{ $group: { \_id: "$freelancerId", totalEarnings: { $sum: "$amount" } } }

]);

1. **Role-Based Access**:
   * Restrict actions based on user role:

const user = await db.users.findOne({ email: "example@example.com" });

if (user.role === "freelancer") {

// Allow bidding

} else if (user.role === "client") {

// Allow posting projects

}

1. **Joining Collections with** 
   * Fetch project details with client and freelancer info:

db.projects.aggregate([

{ $lookup: { from: "users", localField: "clientId", foreignField: "\_id", as: "clientDetails" } },

{ $lookup: { from: "users", localField: "bids.freelancerId", foreignField: "\_id", as: "freelancerDetails" } }

]);

1. **Data Validation**:
   * Use Mongoose for schema validation:

const ProjectSchema = new mongoose.Schema({

title: { type: String, required: true },

budget: { type: Number, min: 0 },

});

# CHAPTER 4

# SETUP INSTRUCTIONS

**4.1 INSTALL PREREQUISITES**

**MongoDB:** A NoSQL database for storing data.

**Express.js:** A web application framework for Node.js.

**React:** A JavaScript library for building user interfaces.

**Node.js:** A JavaScript runtime for server-side development.

**JWT:** JSON Web Tokens for user authentication.

**bcrypt:** A library for hashing user passwords.

Before starting, ensure the following software dependencies are installed on your machine:

* **Node.js**:
* Download and install Node.js
* Verify the installation by running:  
  **node -v**
* **NPM (Node Package Manager)**:
* NPM is usually installed alongside Node.js.
* Verify with:  
  **npm -v**
* **MongoDB**:
* Install MongoDB
* Ensure MongoDB is running on your local machine or connect it to a cloud-based MongoDB service (e.g., MongoDB Atlas).
* For local installations, start MongoDB with:  
   **mongod.**
* Make sure MongoDB is running on its default port (27017), or update the connection string in the project configuration if it's hosted elsewhere.
* **Git**:
* To clone the repository, ensure Git is installed.
* Verify with:  
   **git --version.**

**4.2 STEPS**

**1. Clone the Repository**

1. Open a terminal window.
2. Clone the repository to your local machine using Git

[**https://github.com/15saisakthi/freelancer\_app**](https://github.com/15saisakthi/freelancer_app)

     3. Navigate into the project directory:  
             
     **cd freelancer**

**2. Install Dependencies**

1. Make sure you have Node.js and npm installed. You can verify their installation with: **node -v**

**npm -v**

1. Once inside the project directory, install all required dependencies using:  
    **npm install**

 This command will install all necessary packages specified in the package.json file.

**3. Set Up Environment Variables** 1**.** Create a .env file in the root directory with the following environment variables:

**URI="mongodb+srv://rithicselab:rithika@cluster1.7q2g6.mongodb.net/?retryWrites=true&w=majority&appName=Cluster1"**

**PASSWORD="rithika"**

**PORT=6001**

2.Save the .env file.

**4. Verify MongoDB Connection**

1.To ensure that your MongoDB is connected properly:

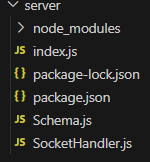
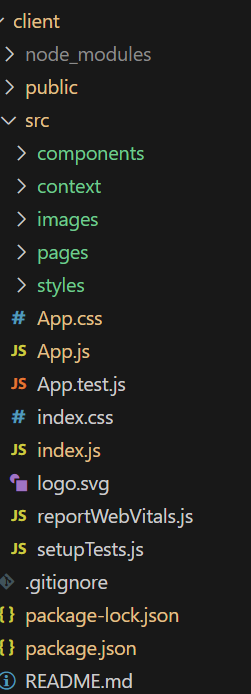
* Check if MongoDB is running (mongod command if using a local setup).
* Alternatively, connect the database with **MongoDB Compass** (GUI) or **Mongo Shell** to confirm the connection and view collections as they are created.

# 

# CHAPTER 5

# FOLDER STRUCTURE

The Grocery WebApp follows a well-organized folder structure for both the **Frontend** (React) and **Backend** (Node.js) to ensure clarity, scalability, and maintainability of the project.

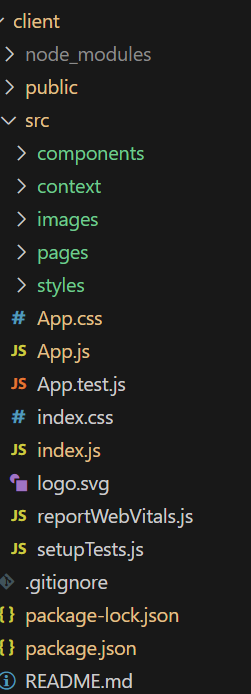
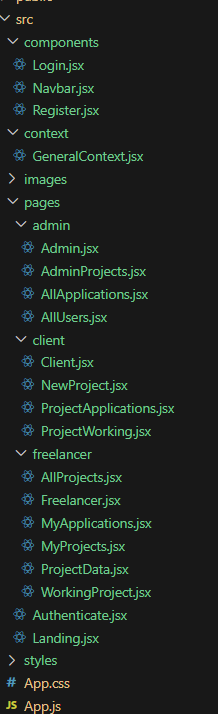
**** ****

*Figure 5.1.Overall Folder Structure*

**5.1 CLIENT: REACT FRONTEND STRUCTURE**

The **React frontend** of the **freelancer application** project has a typical structure:

* **src**: Contains all React components, styles, and images.
* **Components**: Includes reusable UI elements like buttons, forms, and product listings.
* **Pages**: Contains specific views such as home, login, register, and cart.
* **Services**: Includes API functions for interacting with the backend.
* **App.js**: The root component responsible for routing and overall structure.

**** ****

*Figure 5.1.1. Client Structure*

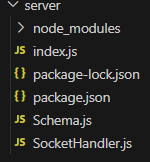
**5.2 SERVER: NODE.JS BACKEND STRUCTURE**

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

The **Node.js backend** of the **freelance application project** is structured into several key folders:

* **controllers:** Contains logic for handling requests (e.g., user authentication).
* **models:** Defines the MongoDB schema and interacts with the database.
* **routes:** Defines endpoint routes and connects them to controllers.
* **middleware:** Handles tasks like authentication and authorization.
* **utils:** Includes utility functions (e.g., for encryption or email sending).

The project uses Express.js for routing and JWT for authentication.

****

*Figure 5.1.2. Servers Structure*

# 

# 

# 

# 

# CHAPTER 6

# RUNNING THE APPLICATION

To run the Freelance application locally, the following steps are done for starting both the frontend and backend servers:

**6.1 FRONTEND:**

1. **Start the Frontend Server:**

After installing the dependencies, use the following command to start the React development server:

**cd frontend**

**npm start**

This will compile the React application and start a local development server.

1. **Access the Application:**

The above command will start the application, and we should see a message indicating that the server is running (e.g., "Server started on port 3000"). Open the web browser and go to http://localhost:3000 (or the port you specified in the .env file) to access the application.

**6.2 BACKEND:**

1. **Start the Backend Server:**

Once the dependencies are installed, use the following command to start the backend server:

**npm start**

This will initialize the backend server and connect it to the database.

1. **Access the Backend:**

The above command will start the application, and we should see a message indicating that the server is running (e.g., "Server started on port 3000"). Open the web browser and go to http://localhost:3000 (or the port you specified in the .env file) to access the application.

# 

# 

# CHAPTER 7

# API DOCUMENTATION

**1. User Routes**

**POST /api/users/register**

* **Description**: Registers a new user (either a freelancer or client).
* **Request Body**:

json

Copy code

{

"name": "John Doe",

"email": "john.doe@example.com",

"password": "password123",

"role": "freelancer" // "client" or "freelancer"

}

* **Response**:
  + **200 OK**: User successfully registered.

json

Copy code

{

"message": "User registered successfully.",

"user": {

"id": "5f8d0d55b54764421b7156b6",

"name": "John Doe",

"email": "john.doe@example.com",

"role": "freelancer"

}

}

* + **400 Bad Request**: Missing or invalid data.

json

Copy code

{

"message": "Invalid input data."

}

**POST /api/users/login**

* **Description**: Authenticates a user and returns a JWT token.
* **Request Body**:

json

Copy code

{

"email": "john.doe@example.com",

"password": "password123"

}

* **Response**:
  + **200 OK**: Successfully logged in and token issued.

json

Copy code

{

"message": "Login successful.",

"token": "JWT\_Token\_Here"

}

* + **401 Unauthorized**: Incorrect credentials.

json

Copy code

{

"message": "Invalid credentials."

}

**GET /api/users/me**

* **Description**: Retrieves the logged-in user's profile.
* **Headers**: Authorization: Bearer <JWT\_Token>
* **Response**:
  + **200 OK**: User profile retrieved successfully.

json

Copy code

{

"id": "5f8d0d55b54764421b7156b6",

"name": "John Doe",

"email": "john.doe@example.com",

"role": "freelancer"

}

* + **401 Unauthorized**: Missing or invalid token.

json

Copy code

{

"message": "Authorization failed."

}

**2. Job Routes**

**POST /api/jobs**

* **Description**: Creates a new job posting (for clients).
* **Request Body**:

json

Copy code

{

"title": "Web Development Project",

"description": "Develop a full-stack web application.",

"budget": 5000,

"deadline": "2024-12-31",

"clientId": "5f8d0d55b54764421b7156b6"

}

* **Response**:
  + **201 Created**: Job created successfully.

json

Copy code

{

"message": "Job posted successfully.",

"job": {

"id": "5f8d0d55b54764421b7156b7",

"title": "Web Development Project",

"description": "Develop a full-stack web application.",

"budget": 5000,

"deadline": "2024-12-31"

}

}

**GET /api/jobs**

* **Description**: Retrieves all job postings.
* **Response**:
  + **200 OK**: Successfully retrieved job listings.

json

Copy code

[

{

"id": "5f8d0d55b54764421b7156b7",

"title": "Web Development Project",

"description": "Develop a full-stack web application.",

"budget": 5000,

"deadline": "2024-12-31",

"clientId": "5f8d0d55b54764421b7156b6"

},

{

"id": "5f8d0d55b54764421b7156b8",

"title": "Mobile App Development",

"description": "Build a cross-platform mobile app.",

"budget": 4000,

"deadline": "2025-01-15",

"clientId": "5f8d0d55b54764421b7156b9"

}

]

**GET /api/jobs/**

* **Description**: Retrieves a specific job posting by its ID.
* **Parameters**: jobId (the ID of the job)
* **Response**:
  + **200 OK**: Job details retrieved.

json

Copy code

{

"id": "5f8d0d55b54764421b7156b7",

"title": "Web Development Project",

"description": "Develop a full-stack web application.",

"budget": 5000,

"deadline": "2024-12-31",

"clientId": "5f8d0d55b54764421b7156b6"

}

**3. Bid Routes**

**POST /api/bids**

* **Description**: Submits a bid on a job (by freelancers).
* **Request Body**:

json

Copy code

{

"jobId": "5f8d0d55b54764421b7156b7",

"freelancerId": "5f8d0d55b54764421b7156b6",

"bidAmount": 4500,

"proposal": "I can complete this project within 3 weeks."

}

* **Response**:
  + **201 Created**: Bid successfully submitted.

json

Copy code

{

"message": "Bid submitted successfully.",

"bid": {

"id": "5f8d0d55b54764421b7156b7",

"jobId": "5f8d0d55b54764421b7156b7",

"freelancerId": "5f8d0d55b54764421b7156b6",

"bidAmount": 4500,

"proposal": "I can complete this project within 3 weeks."

}

}

**GET /api/bids/job/**

* **Description**: Retrieves all bids for a specific job posting.
* **Parameters**: jobId (the ID of the job)
* **Response**:
  + **200 OK**: List of bids for the job.

json

Copy code

[

{

"id": "5f8d0d55b54764421b7156b7",

"freelancerId": "5f8d0d55b54764421b7156b6",

"bidAmount": 4500,

"proposal": "I can complete this project within 3 weeks."

},

{

"id": "5f8d0d55b54764421b7156b8",

"freelancerId": "5f8d0d55b54764421b7156b7",

"bidAmount": 4000,

"proposal": "I can finish the project in 2 weeks."

}

]

**4. Transaction Routes**

**POST /api/transactions**

* **Description**: Creates a new transaction (for payment processing).
* **Request Body**:

json

Copy code

{

"userId": "5f8d0d55b54764421b7156b6",

"amount": 4500,

"transactionType": "payment", // "payment" or "refund"

"jobId": "5f8d0d55b54764421b7156b7",

"status": "completed"

}

* **Response**:
  + **201 Created**: Transaction successfully created.

json

Copy code

{

"message": "Transaction processed successfully.",

"transaction": {

"id": "5f8d0d55b54764421b7156b7",

"userId": "5f8d0d55b54764421b7156b6",

"amount": 4500,

"transactionType": "payment",

"status": "completed"

}

}

**Error Responses**

* **400 Bad Request**: Invalid request or missing parameters.

json

Copy code

{

"message": "Bad request. Missing required fields."

}

* **404 Not Found**: Resource not found (e.g., job, user, bid).

json

Copy code

{

"message": "Resource not found."

}

* **500 Internal Server Error**: Server encountered an error while processing the request.

json

Copy code

{

"message": "Something went wrong. Please try again later."

}

# CHAPTER 8

# AUTHENTICATION

#### Backend Overview (Node.js & Express)

1. **User Schema**:
   * Stores details for freelancers and clients, including securely hashed passwords and role-based information (e.g., "freelancer" or "client").
   * Additional fields include freelancer portfolios, skills, and client project details.
2. **Login and Registration**:
   * Registration endpoint collects user details and stores them securely.
   * Login endpoint verifies credentials and provides a token to authenticated users.
3. **Token Management**:
   * A JSON Web Token (JWT) is issued upon successful login, containing user-specific details like ID and role.
   * Tokens are signed using a secure secret key and have an expiration time to enhance security.
4. **Role-Based Access Control (RBAC)**:
   * Middleware ensures specific actions are allowed only for certain roles. For example:
     + Only "clients" can post projects.
     + Only "freelancers" can submit bids.
5. **Middleware Validation**:
   * Verifies incoming tokens to determine if the user is authenticated.
   * Prevents unauthorized access to protected backend resources like project management.

#### Frontend Overview (React)

1. **Token Storage**:
   * Tokens are stored securely in localStorage or as httpOnly cookies, depending on the application's sensitivity to security threats like XSS.
2. **Authentication State**:
   * The frontend checks for the presence of a token to confirm whether the user is logged in.
   * Invalid or expired tokens redirect users to the login page.
3. **Protected Components**:
   * Components like freelancer profiles or client dashboards are only accessible when a valid token is detected.
4. **Conditional UI Rendering**:
   * Based on the user’s role, features like "Post Project" (client) or "Bid Now" (freelancer) are displayed selectively.

#### Workflow of Authentication and Authorization

1. **User Login**:
   * The user provides credentials. If valid, the backend issues a JWT token.
   * The frontend stores the token for future API requests.
2. **Accessing Resources**:
   * For every protected resource request, the token is sent in the request headers.
   * The backend validates the token and checks the user's permissions.
3. **Session Management**:
   * JWT tokens allow stateless sessions. When a token expires, the user is prompted to log in again.
4. **Logout and Security**:
   * On logout, tokens are cleared from storage.
   * Sensitive applications may use token blacklisting or rotation to prevent misuse.

#### Features Specific to Freelancer Applications

1. **Freelancer Dashboard**:
   * Lists projects, submitted bids, and earnings.
   * Displays metrics like ratings and completed jobs.
2. **Client Dashboard**:
   * Allows project creation, bid management, and freelancer selection.
3. **Access Control**:
   * Freelancers can only view and bid on projects, not post them.
   * Clients cannot place bids but can view freelancer profiles and manage their projects.
4. **Role-Specific Enhancements**:
   * Notifications for new project postings (freelancers) or bid submissions (clients).
   * Detailed profile views for freelancers, accessible only to logged-in clients.

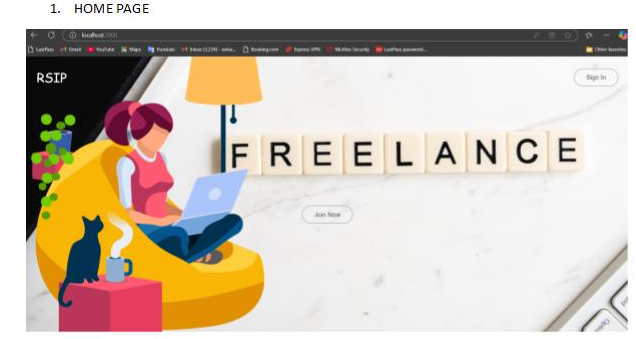
#### Security Considerations

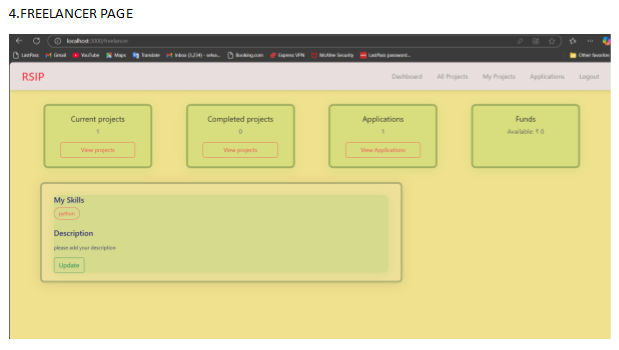
1. **Token Expiry and Refresh**:
   * Tokens expire after a set period, requiring reauthentication.
   * Refresh tokens can be implemented for seamless session renewal.
2. **Storage Best Practices**:
   * Use httpOnly cookies for enhanced security in sensitive applications.
   * Avoid storing tokens in places vulnerable to cross-site scripting (XSS).
3. **Secure Communication**:
   * All token exchanges happen over HTTPS to prevent interception.

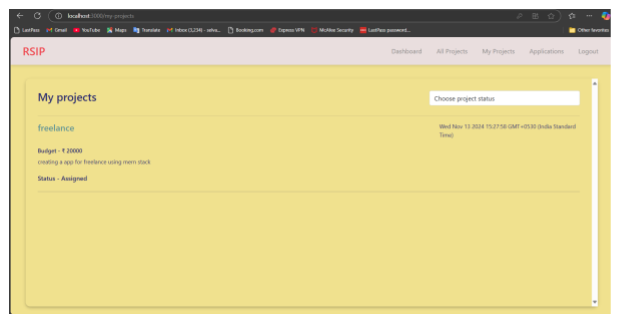
# CHAPTER 9

# USER INTERFACE

The freelance application’s UI, built with React, is designed to offer an intuitive and responsive experience for clients, freelancers, and administrators. It simplifies workflows through tailored dashboards, enabling clients to post projects and manage hires, freelancers to showcase skills and track jobs, and admins to oversee platform activities and resolve disputes. With a focus on seamless navigation, accessibility, and real-time updates, the UI ensures efficient and engaging interactions for all users.

****

****

****

*Figure 9.1. User Interface*

# CHAPTER 10

# TESTING

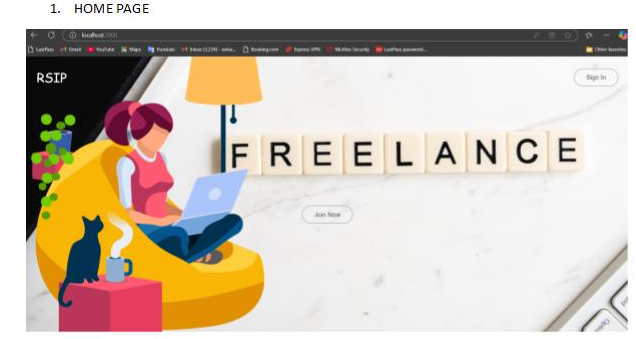
The grocery app was tested to ensure functionality, performance, and user experience consistency. The testing strategy focused on:

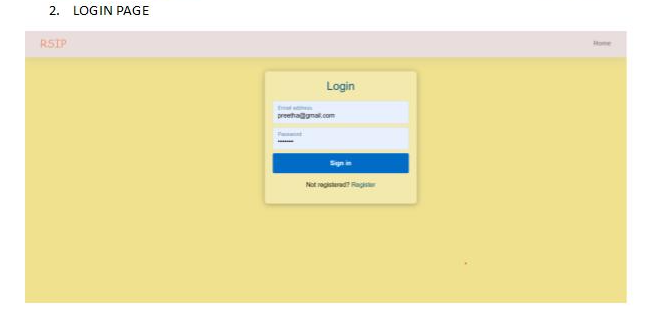
1. **Unit Testing:** Key components and backend API endpoints were tested using Jest and Mocha, ensuring each function worked independently as expected.
2. **Integration Testing:** Integrated workflows like adding items to the cart and checkout were tested to verify interactions between components.
3. **End-to-End Testing:** Tools like Cypress simulated real user flows to identify any issues in the full user journey.
4. **Manual Testing:** Additional manual testing was conducted to validate responsiveness and user interface behavior on different devices.

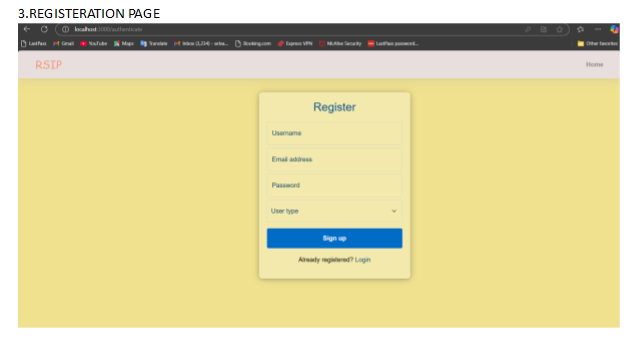
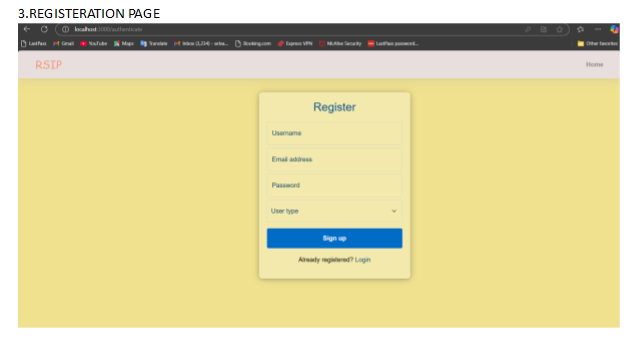
# CHAPTER 11

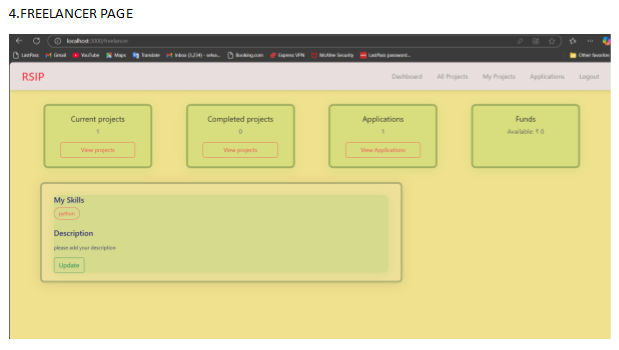
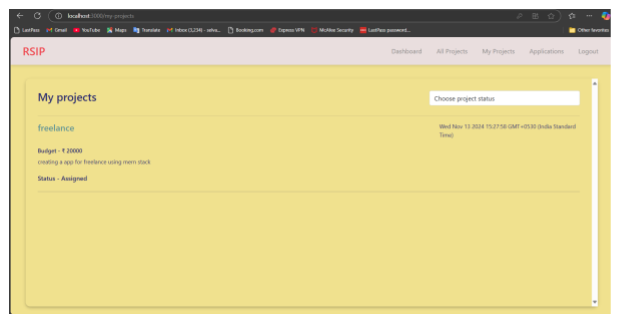
# SCREENSHOTS OR DEMO

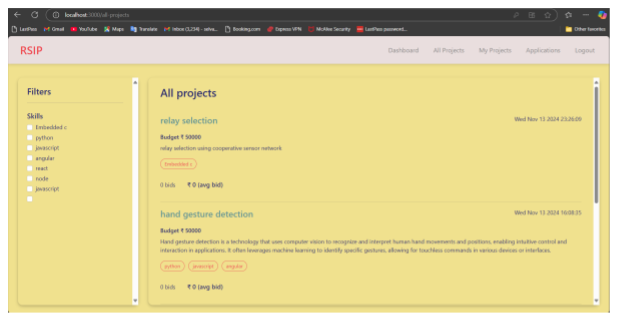
**11.1 SCREENSHOTS:**

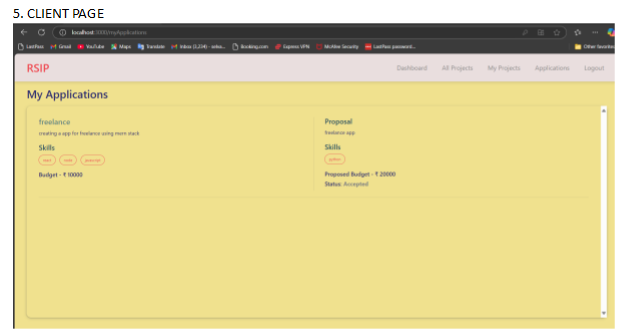
****

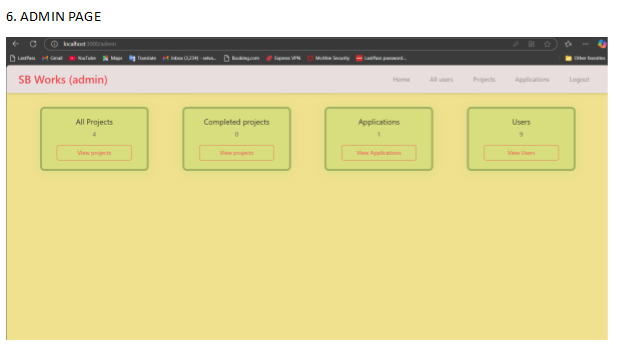
****

****

****

****

****

****

**Demo Video Link : https://drive.google.com/file/d/1fh3Yr3BWVD8wCJZGoThdxrDf5r71yxqu/view?usp=sharing**

# 

# CHAPTER 12

# KNOWN ISSUES

1. **Socket.io Connection Errors**:
   * Misconfiguration in the backend or incorrect URL for the WebSocket server might cause chat or notification features to fail.
2. **Frontend-Backend Communication**:
   * Cross-Origin Resource Sharing (CORS) issues if the API endpoints are not configured to handle requests from the client correctly.
3. **State Management**:
   * UI inconsistencies due to inadequate or improper state management in React.
4. **Database Connectivity**:
   * MongoDB-related issues such as schema validation errors or connection timeouts.

**Specific Indicators:**

* **Missing Dependencies**:
  + Projects using outdated or mismatched package versions can face issues, especially if react-scripts or other libraries are not compatible with the current Node.js version.
* **Responsive Design**:
  + Pages may not render correctly on different devices if CSS or UI elements are not fully responsive.

# 

# CHAPTER 13

# FUTURE ENHANCEMENTS

* 1. **Enhanced User Profiles**
* Skills Assessment: Integrate tests for freelancers to validate skills (e.g., coding challenges, writing tests).
* Portfolio Section: Allow freelancers to showcase their past projects with images, videos, or downloadable files.
* Client Verification: Add verification badges for trustworthy clients (e.g., ID or payment verification).

2. **Project Management Tools**

* Task Management: Include features for task assignment, tracking, and deadline management.
* Collaborative Workspace: Provide shared documents and tools for client-freelancer collaboration.
* Time Tracking: Allow freelancers to track hours worked directly within the app.

3. **Improved UI/UX**

* Dark Mode: Add a dark theme for better usability at night.
* Mobile App: Develop a dedicated mobile application for Android and iOS.
* Accessibility Features: Ensure the platform is inclusive with support for screen readers and keyboard navigation.

4.**Enhanced Notifications System**

* Custom Alerts: Allow users to customize notifications for bids, messages, and project updates.
* Email Integration: Send email summaries for project updates or new opportunities.

5. **Review and Rating Improvements**

* Detailed Feedback: Allow clients and freelancers to leave structured feedback (e.g., punctuality, communication, quality of work).
* Dispute Resolution: Add a feature to manage conflicts between freelancers and clients.

# 

# CHAPTER 14

# CONCLUSION

The Freelancing Application built using the MERN stack (MongoDB, Express.js, React, and Node.js) serves as a comprehensive and scalable platform aimed at bridging the gap between freelancers and clients seeking professional services. Throughout this project, we have focused on creating an intuitive, feature-rich, and secure environment that facilitates seamless collaboration, efficient project management, and secure transactions.

The architecture of the application leverages modern web development best practices, with a React-based frontend offering a responsive and engaging user interface and a robust backend powered by Node.js and Express.js. The use of MongoDB for the database ensures flexibility, scalability, and high performance for managing complex user data, projects, and transactions.

Our commitment to implementing advanced features such as real-time communication, bidding and profile management, and a robust authentication system underscores our dedication to building a platform that addresses the diverse needs of both freelancers and clients. Additionally, the outlined potential enhancements—including machine learning recommendations, mobile application development, and multi-language support—demonstrate our vision for continuous improvement and adaptability to evolving user demands.

While challenges and known issues remain, they present opportunities for refinement and growth, guiding the application's future development trajectory. The **Freelancing Application** not only showcases the power and versatility of full-stack development with MERN but also emphasizes the importance of creating solutions that prioritize user experience, security, and scalability.

We are confident that this project can play a pivotal role in transforming the way freelancers and clients connect, collaborate, and achieve their professional goals, fostering a more efficient, transparent, and accessible digital marketplace.

**GITHUB LINK: https://github.com/15saisakthi/freelancer\_app**