Full Stack Development with MERN

1. Introduction

• **Project Title:** FreelanceFinder: Discovering Opportunities, Unlocking Potential

• **Team Members:**

Team Leader: Jyothsna Devi Mugandi

2. Project Overview

The primary purpose of this Freelancing project is to design and develop FreelanceFinder, a full-stack freelancing platform built using the MERN stack (MongoDB, Express.js, React.js, Node.js). The system aims to create a secure, user-friendly, and efficient online environment that connects clients and freelancers while enabling seamless project management.

The main objective of the project is to design and implement a structured online environment that simplifies freelancing workflows. Clients can publish project requirements, review freelancer proposals, and manage ongoing work, while freelancers can explore opportunities, apply for projects, and communicate directly with clients. By integrating modern web technologies, the platform improves accessibility, productivity, and transparency in project management.

FreelanceFinder incorporates **three primary roles — Client, Freelancer, and Admin** — to ensure organized workflow and effective platform supervision. Clients focus on project creation and management, freelancers handle applications and submissions, and administrators oversee system operations, monitor activity, and maintain platform stability.

• **Features:**

- **Freelancer Features**
  - **Browse Available Projects**
    Freelancers can explore all available projects posted by clients. Projects are displayed with relevant details such as budget, description, and required skills, helping freelancers identify suitable opportunities.
  - **Submit Proposals**
    Freelancers can apply for projects by submitting proposals. This feature allows freelancers to express interest, explain their experience, and negotiate project terms with clients.

- **Chat with Clients Instantly**

  Through the real-time messaging system, freelancers can communicate directly with clients. This helps clarify requirements, discuss project progress, and maintain continuous interaction during the project lifecycle.

- **Receive Ratings and Feedback**

  After completing work, freelancers receive ratings and reviews from clients. These ratings improve credibility and help freelancers build a strong professional profile on the platform.

➢ Client Features

- **Post and Manage Projects**

  Clients can create new projects by providing details such as title, description, budget, and required skills. Once posted, clients can view, edit, or delete their projects through the dashboard. This feature allows clients to manage their work requirements efficiently and track ongoing project progress.

- **Review Freelancer Applications**

  After posting a project, clients receive applications from freelancers. The client dashboard displays all submitted proposals, enabling clients to review freelancer profiles, skills, and messages before selecting the most suitable candidate.

- **Communicate Using Real-Time Chat**

  The system includes a real-time chat feature that allows direct communication between clients and freelancers. Messages are sent instantly without refreshing the page, improving collaboration and reducing communication delays during project execution.

- **Provide Ratings and Reviews**

  Once a project is completed, clients can rate freelancers and provide feedback. This rating system builds trust within the platform and helps future clients evaluate freelancer performance based on previous work.

➢ Admin Features

- **Secure Admin Login**

  The system includes a separate admin authentication mechanism. Admin routes are protected using role-based authorization to ensure only administrators can access management features.

❋ **View All Users and Projects**

Admins can view a centralized list of all registered users and posted projects. This helps monitor platform activity and maintain transparency across the system.

❋ **Monitor Platform Activity**

The admin dashboard provides insights into platform operations, including user interactions, project status, and system behavior. This ensures smooth platform performance and security.

❋ **Manage System Data and Maintain Moderation**

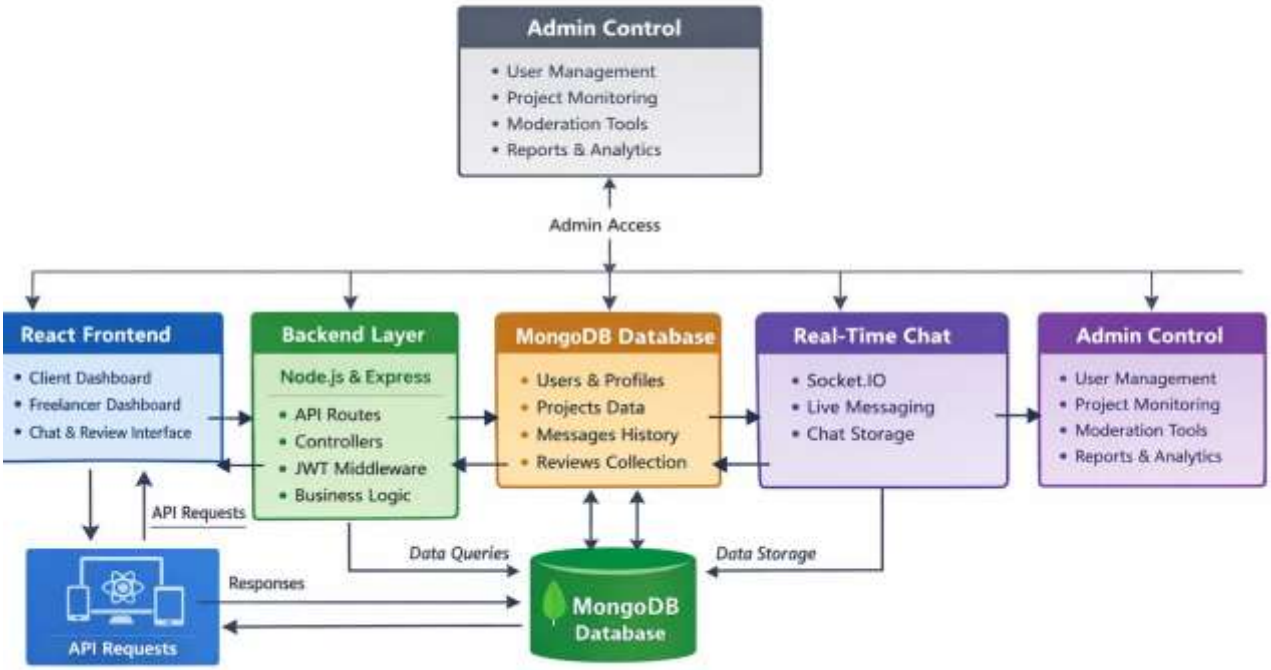
Administrators can oversee content, monitor reviews, and ensure that communication between users follows platform guidelines. This feature helps maintain a safe and professional environment.

- **Goals of the Project**

1. Develop a real-world MERN stack application.
2. Implement secure authentication using JWT tokens.
3. Enable real-time chat communication between users.
4. Integrate rating and review features.
5. Provide an admin control panel for system monitoring.
6. Demonstrate modular backend architecture and RESTful APIs.

3. Architecture

Fig 1 :- System Architecture



I. Frontend Layer (React Web Application):

The frontend represents the presentation layer of the system where users interact with the platform. It is developed using React.js and follows a component-based architecture to ensure scalability and maintainability.

This layer provides separate dashboards for:

- Clients (Project creation and management)
- Freelancers (Project browsing and applications)
- Admin (System monitoring and control)

React Router is used to manage navigation, while role-based route protection ensures secure access to pages.

II. Backend Layer (Node.js & Express.js)

The backend acts as the application logic layer and processes all requests coming from the frontend. It is built using Node.js and Express.js following an MVC architecture.

Main Components:

- Controllers – Handle business logic
- Routes – Define API endpoints
- Middleware – Verify JWT tokens and user roles
- Services – Manage chat, reviews, and project operations

The backend ensures secure communication between users and maintains structured workflows for project creation, freelancer applications, and admin operations.

III..Database Layer (MongoDB)

MongoDB serves as the primary data storage system for the platform. Data is stored using Mongoose schemas to maintain relationships between users, projects, messages, and reviews.

Collections Include:

- Users Collection – Stores client, freelancer, and admin data
- Projects Collection – Stores project details
- Messages Collection – Stores chat history
- Reviews Collection – Stores ratings and feedback

MongoDB allows flexible schema design and supports scalable data management.

IV Real-Time Communication Layer (Chat System)

The platform includes a real-time messaging system that enables instant communication between clients and freelancers.

Architecture Flow:

Frontend Chat Interface → Socket Communication → Backend Server → MongoDB Storage

This layer ensures:

- Instant message delivery
- Live updates without page refresh
- Continuous collaboration between users

V. Admin Control Layer

The Admin Layer superv ises platform activities and maintains system integrity. It is integrated with backend APIs and provides access to platform analytics and user management tools.

Admin Capabilities:

- View all users and projects
- Monitor ratings and reviews
- Maintain system moderation
- Access protected admin routes

Role-based middleware ensures that only authenticated admins can access this layer.

Other Tools

- JWT for authentication
- Git for version control

4. Setup Instructions

• **Prerequisites:**

This section lists the software and tools required before running the project.

- Node.js and npm – To run backend server and install packages

- MongoDB – To store application data
- Express.js – For backend framework
- React.js – For frontend user interface
- Mongoose – For database connectivity
- Git – For cloning and version control
- Code Editor (Visual Studio Code recommended)
- Basic knowledge of HTML, CSS, and JavaScript

• **Installation:**

This section provides a step-by-step guide to clone the project, install dependencies, and configure environment variables for running the application locally.

Step 1: Clone the Repository

Open a terminal and run the following command

git clone <repository-url>

Navigate into the project directory:

cd Sbwork-freelancing-platform

Step 2: Backend Setup

1. Navigate to the backend folder:

cd server

2. Install backend dependencies:

npm install

3. Create a .env file in the server directory and configure the following environment variables:

PORT=5000

MONGO_URI=your_mongodb_connection_string

JWT_SECRET=your_jwt_secret_key

4. Start the backend server:

npm start

The backend server will start running on http://localhost:5000 (or the port specified in .env).

Step 3: Frontend Setup

1. Open a new terminal and navigate to the frontend folder:

cd client

2. Install frontend dependencies:

npm install

3. Create a .env file in the client directory (if required) and add the backend API URL:

REACT_APP_API_URL=http://localhost:5000

4. Start the frontend application:

npm start

The frontend application will run on http://localhost:3000 by default.

Step 4: Verify the Installation

- Ensure MongoDB is running (local)
- Backend should connect successfully to the database
- Frontend should load without errors
- Test user registration and login to confirm API connectivity

Step 5: Common Issues & Fixes

- **Port already in use:** Change the port number in the .env file
- **Database connection error:** Verify the MongoDB connection string
- **API not reachable:** Ensure backend server is running before starting frontend

5. Folder Structure

• **Client**: The client folder contains the React.js frontend responsible for the user interface and user interactions.

- public/ – Static assets
- src/components/ – Navbar, ChatBox, Admin UI
- src/pages/ – Login, Dashboard, Admin Panel
- src/Api/ – Axios configuration
- App.js – Route management

Server: The server folder contains the Node.js and Express backend that manages business logic, authentication, and database operations.

Server Folder Structure:

- models/ – User, Project, Message, Review schemas
- routes/ – API endpoints
- controllers/ – Business logic
- middleware/ – JWT authentication
- config/ – Database setup

6. Running the Application

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

o **Frontend:**

1. Open a terminal
2. Navigate to the server directory:

cd server

3. Start the backend server:

npm start

o **Backend:**

1. Open a terminal
2. Navigate to the server directory:

cd server

3. Start the backend server:

npm start

7. API Documentation

This section documents the REST API endpoints exposed by the backend server. Each endpoint includes the HTTP method, purpose, request parameters, and example responses.

13.1 Authentication APIs

Register User

- Endpoint: /api/auth/register
- Method: POST
- Description: Registers a new user (Client or Freelancer)

Request Body:

{

"name": "John Doe",

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

"password": "password123",

"role": "client"

}

Success Response (201):

{

"message": "User registered successfully"

}

Login User

- Endpoint: /api/auth/login
- Method: POST
- Description: Authenticates user and returns JWT token

Request Body:

{

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

"password": "password123"

}

Success Response (200):

{

"token": "jwt_token_here",

"user": {

"id": "12345",

"role": "client"

}

}

13.2 Project APIs

Create Project

- Endpoint: /api/projects/create
- Method: POST
- Authorization: Required (Client)
- Description: Allows a client to create a new project

Request Body:

{

"title": "Website Development",

"description": "Build a MERN stack website",

"budget": 5000

}

Success Response (201):

{

"message": "Project created successfully"

}

Get All Projects

- Endpoint: /api/projects
- Method: GET
- Authorization: Required
- Description: Retrieves all available projects

**Success Response (200):

[

{

"id": "p1",

"title": "Website Development",

"budget": 5000

}

]

Get Client Projects

- Endpoint: /api/projects/client
- Method: GET
- Authorization: Required (Client)
- Description: Retrieves projects created by the logged-in client

Success Response (200):

[

{

"title": "Website Development",

"status": "open"

}

Delete Project

- Endpoint: /api/projects/:id
- Method: DELETE
- Authorization: Required (Client)
- Description: Deletes a project by ID

Success Response (200):

{

"message": "Project deleted successfully"

}

13.3 User APIs

Get User Profile

- Endpoint: /api/users/profile
- Method: GET
- Authorization: Required
- Description: Retrieves logged-in user profile

Success Response (200):

{

"name": "John Doe",

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

"role": "client"

}

13.4 Admin APIs

Get All Users

- Endpoint: /api/admin/users
- Method: GET
- Authorization: Required (Admin)
- Description: Retrieves all registered users

Success Response (200):

[

{

"name": "John Doe",

"role": "client"

}

]

13.5 Error Response Format

Error Response Example (401 / 400):

{

"error": "Unauthorized access"

}

13.6 API Security

- JWT-based authentication
- Protected routes using middleware
- Role-based access control

13.7 Review APIs

Add Review

• Endpoint: /api/reviews
• Method: POST
• Authorization: Required

Request:

{

"projectId": "123",

"rating": 5,

"comment": "Excellent work!"

}

Response:

{

"message": "Review submitted successfully"

}

13.8 Chat APIs

Send Message

• Endpoint: /api/messages
• Method: POST

Request:

{
"chatId": "abc123",
"message": "Hello!"
}

8. Authentication

Authentication in the project is implemented using **JSON Web Tokens (JWT)** to ensure secure access to the application.

1. **User Registration**
    - Users (Client or Freelancer) register by providing details such as name, email, password, and role.
    - The password is encrypted before being stored in the database.
    - User details are saved securely in MongoDB.
2. **User Login**
    - When a user logs in, the system verifies the email and password.
    - If the credentials are valid, the backend generates a **JWT token**.
    - This token is sent to the frontend as part of the response.
3. **Token Storage & Usage**
    - The frontend stores the JWT token (usually in local storage).

- For every protected API request, the token is sent in the request header:

Authorization: Bearer <token>

- This allows the backend to verify the identity of the user for each request.

The project implements **token-based authentication** using **JSON Web Tokens (JWT)** along with **middleware-based authorization**. No traditional server-side sessions are used, making the system scalable and stateless.

1. Token-Based Authentication (JWT)

Token Generation

- When a user successfully logs in, the backend generates a **JWT (JSON Web Token)**.
- The token contains:
  - User ID
  - User role (Admin / Client / Freelancer)
  - Token expiration time
- The token is signed using a secret key stored in environment variables (JWT_SECRET).

Token Example (Conceptual)

Header.Payload.Signature

Token Delivery

- The generated token is sent to the frontend in the login response.
- Example response:

```
{
  "token": "jwt_token_here"
}
```

2. Token Storage and Usage

Frontend Storage

- The frontend stores the JWT token in **localStorage** (or browser memory).
- The token persists across page refreshes.

Token in Requests

- For every protected API request, the frontend includes the token in the HTTP header:

Authorization: Bearer <JWT_TOKEN>

3. Token Verification (Backend)

- A custom **authentication middleware** intercepts incoming requests.
- The middleware:
    1. Extracts the token from the Authorization header
    2. Verifies the token using the secret key
    3. Decodes user information (ID and role)
    4. Attaches user data to the request object
- If the token is:
    - **Missing** → Request is rejected
    - **Invalid or expired** → Request is rejected with 401 Unauthorized

4. Authorization (Role-Based Access Control)

Role Validation

- After authentication, an **authorization middleware** checks the user's role.
- Access is granted or denied based on predefined role permissions.

Role Permissions

- **Client**
    - Create projects
    - View and delete own projects
- **Freelancer**
    - View available projects
    - Manage personal profile
- **Admin**
    - View all users
    - Manage platform data

5. Sessions Handling

- The project **does not use server-side sessions**.
- It follows a **stateless authentication model**:
    - No session data is stored on the server
    - Every request is independently authenticated using the JWT
- This improves:
    - Scalability
    - Performance
    - Cloud deployment compatibility

6. Token Expiration and Security

- JWT tokens have an **expiration time** to reduce security risks.
- Once expired:
    - User must log in again to receive a new token
- Additional security measures:
    - Passwords are encrypted before storage
    - Protected routes use middleware
    - Sensitive keys are stored in .env files

7. Error Handling

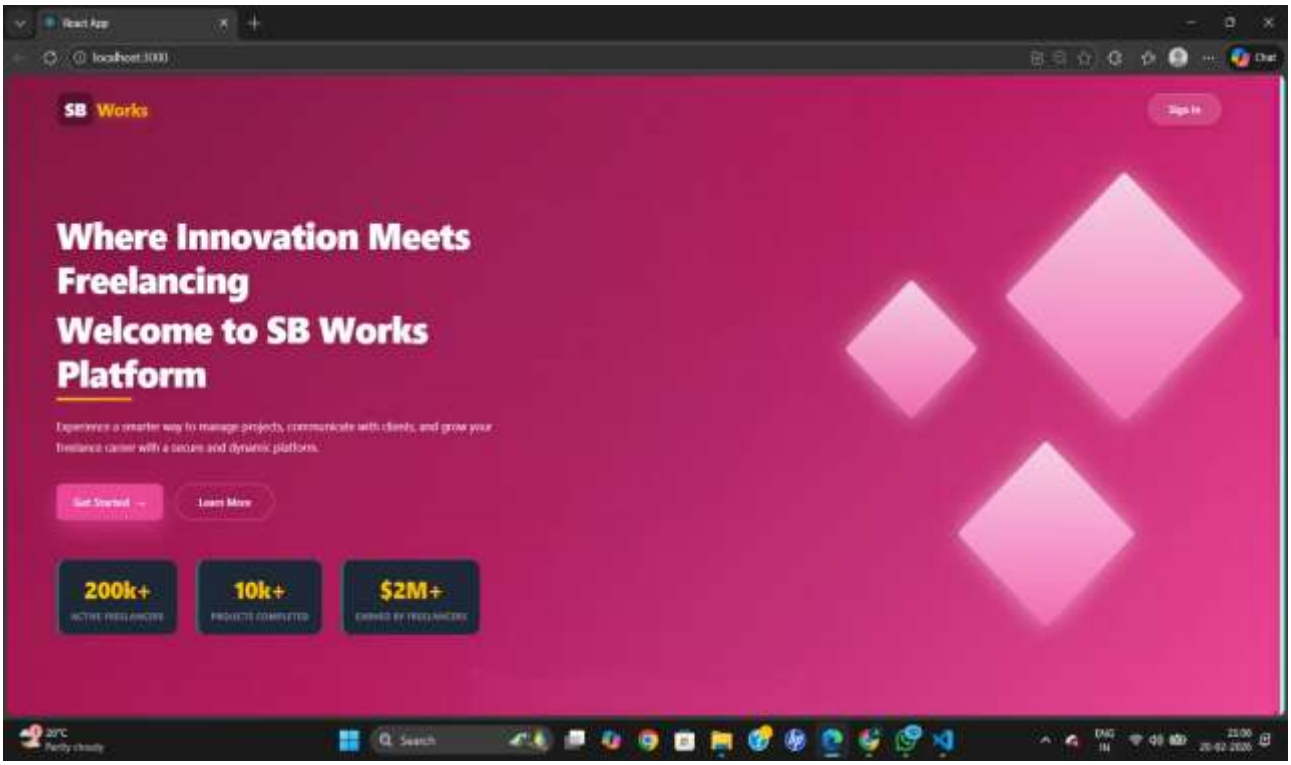
Common Authentication Errors

- **401 Unauthorized** – Invalid or missing token
- **403 Forbidden** – Valid token but insufficient permissions

Example error response:

```
{
  "error": "Unauthorized access"
}
```

9. User Interface



10. Testing

Testing Strategy

The project follows a **manual and functional testing approach** to ensure correct behavior of core features.

- **Unit Testing (Manual):**
  Individual backend APIs and controller functions were tested using sample inputs to verify correct responses.
- **Integration Testing:**
  Tested interaction between the React frontend and Node.js backend to ensure APIs return correct data and actions reflect in the database.
- **Functional Testing:**
  Verified main functionalities such as user registration, login, project creation, project listing, and deletion.
- **Role-Based Testing:**
  Tested the application using different roles (Admin, Client, Freelancer) to ensure proper authorization and restricted access.

- **Error Handling Testing:**
  Checked invalid inputs, incorrect login credentials, and unauthorized access scenarios.

Testing Tools Used

- **Postman:** Used to test backend REST APIs (GET, POST, DELETE).
- **Browser Developer Tools:** Used to monitor API requests, responses, and console errors.
- **MongoDB Compass:** Used to verify database entries and data consistency.
- **Manual UI Testing:** Performed by interacting with the application through the browser.

11. Screenshots or Demo

Fig 1.1 Register Page

\

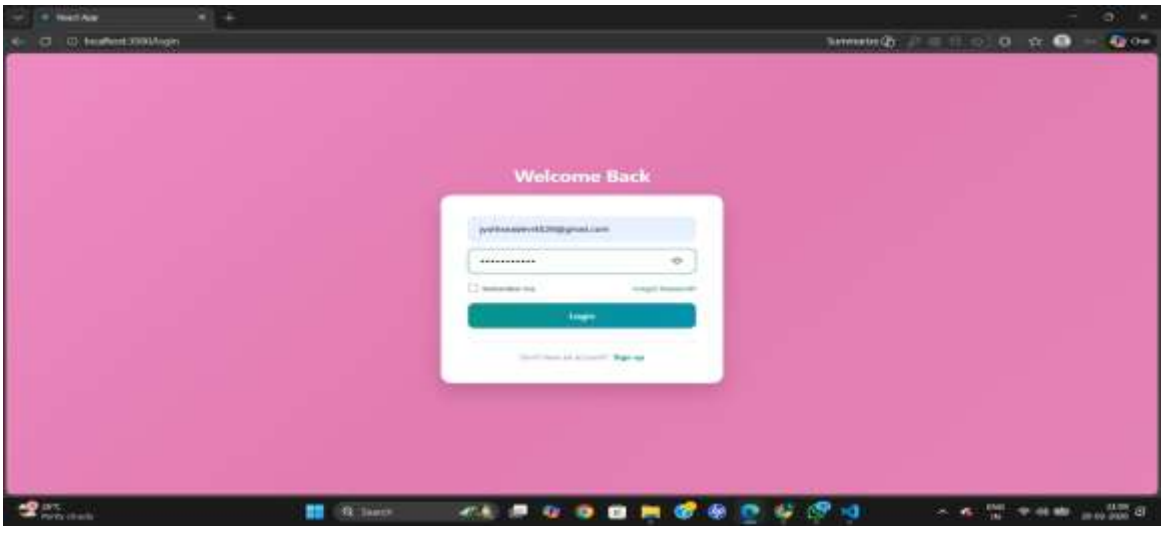


Fig 1.2 Login Page

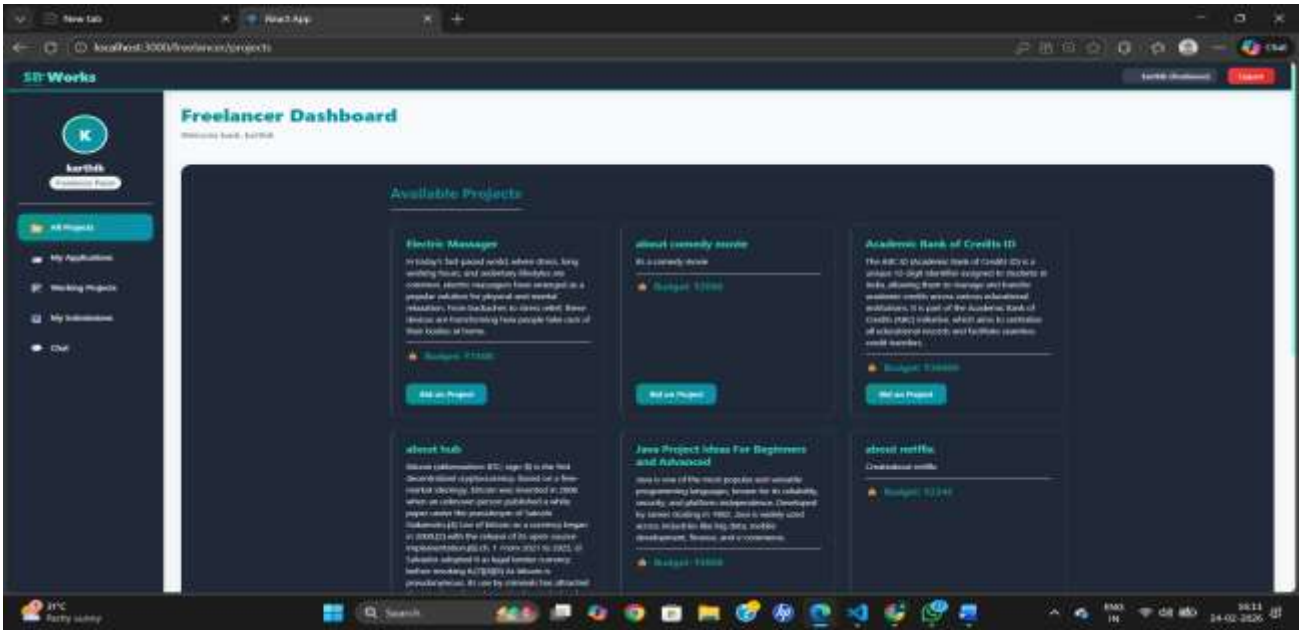


Fig 1.3 Freelancer Dashboard

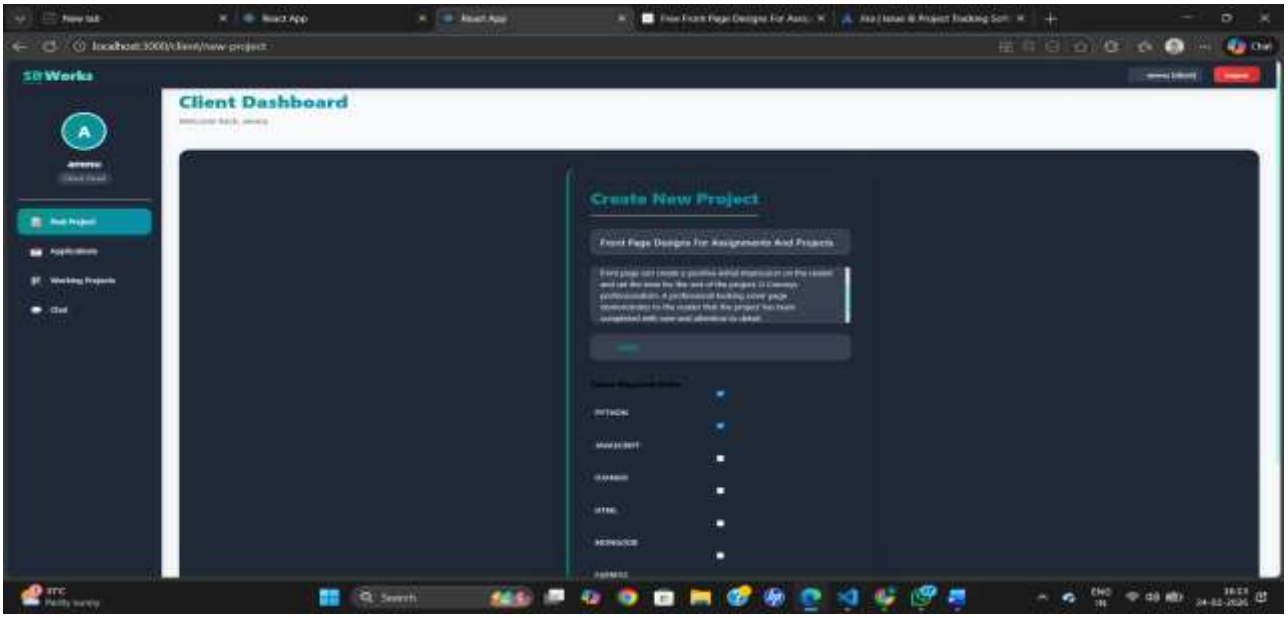


Fig 1.4 Client Dashboard

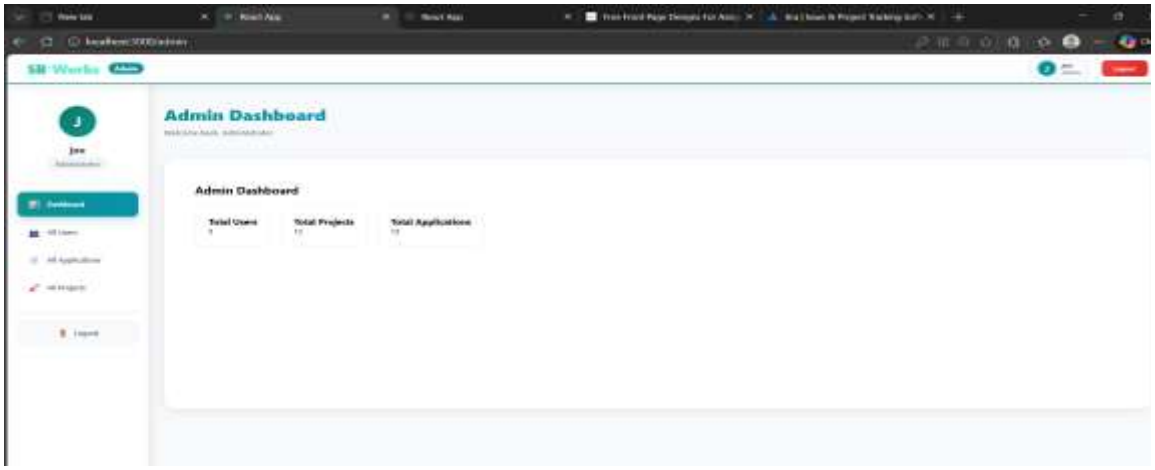


Fig 2.5 Admin Dashboard

12. Known Issues

Known Bugs and Issues

This section describes the known bugs, limitations, and issues that users and developers should be aware of while using or maintaining the application.

1. Authentication Issues

- JWT tokens expire after a fixed duration; users may be logged out automatically after inactivity.
- If the token is deleted from browser storage, protected pages may redirect to the login screen unexpectedly.
- No automatic token refresh mechanism is implemented.

2. Backend Issues

- Incorrect MongoDB connection string or missing environment variables can prevent the server from starting.
- Backend server must be restarted manually after code changes if nodemon is not used.
- Error messages returned by some APIs are generic and may not clearly indicate the root cause.

3. Frontend Issues

- Page refresh may cause logout if token persistence is not handled properly.
- Limited user-friendly error messages for failed API requests.
- Loading indicators are not available for all API calls, which may affect user experience.

4. Role-Based Access Limitations

- Incorrect role assignment during user registration can restrict access to certain features.
- UI-level restrictions depend on correct role validation from the backend.

5. Performance and Scalability

- Application performance may degrade with a large number of users or projects.
- No caching or pagination implemented for large datasets.

6. Browser Compatibility

- The application is optimized for modern browsers.
- Minor UI inconsistencies may occur in older browser versions.

7. Security Limitations

- Tokens are stored on the client side, which may be vulnerable if not handled securely.
- No advanced security features such as rate limiting or request throttling are implemented.

13. Future Enhancements

Although the FreelanceFinder platform successfully implements core freelancing functionalities, several improvements can enhance scalability, security, and user experience in future versions.

- Payment gateway integration for secure online transactions.
- Real-time notification system for messages, project updates, and reviews.
- Advanced analytics dashboard for admin monitoring and platform insights.
- AI-based freelancer recommendation based on skills and ratings.
- Video call feature between client and freelancer for live discussions.
- Secure token storage using HTTP-only cookies instead of localStorage.
- Performance optimization through caching and pagination.
- Mobile application version using React Native or Flutter.

14. APPENDIX

- My Project Source code Files are available at :
  https://drive.google.com/drive/folders/15gVc2thCpXiq06EWhkKiOYjwuIvqBA5z?usp=drive_link
- My project Demo Video link is available at :
  https://drive.google.com/drive/folders/1fdxppY3AmzaJhRJSIxaNBAxCdstc5Uqe?usp=drive_link
- Github Resopitory Link :
  https://github.com/jyothsna2617/Freelancing_Sbworks_Platform_Design