**FREELANCER WEBSITE USING MERN**

**TEAM MEMBERS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **NAME** | **ROLE** |
| 1 | DHARSHINI J | TESTING |
| 2 | KIRUTHIKA P | DEVELOPER |
| 3 | SWATHI S | DEVELOPER |
| 4 | JEMIMA S | BACKEND |
| 5 | SARAVANAN | DATABASE |

**PROJECT OVERVIEW**

**PURPOSE**

The purpose of this project is to build a modern, full-stack web application that connects freelancers with clients. This platform serves as a comprehensive solution for freelancers to showcase their skills and portfolios, while clients can browse, hire, and collaborate with professionals for their projects. Utilizing the MERN (MongoDB, Express.js, React.js, Node.js) stack, the application ensures a seamless, responsive, and scalable user experience.

**Empower Freelancers:**  
Provide freelancers with tools to create profiles, display their portfolios, and attract potential clients in an efficient and professional manner.

**Enable Client Collaboration:**  
Allow clients to post job opportunities, search for freelancers based on skills and experience, and manage projects within the platform.

**Seamless Interaction:**  
Implement a messaging system to enable real-time communication between freelancers and clients.

**Secure Transactions:**  
Integrate secure payment gateways for clients to make payments and freelancers to receive payments safely.

**Scalable and Reliable Architecture:**  
Leverage the MERN stack to deliver a scalable, maintainable, and high-performance application that handles large user traffic.

**Features and Functionalities of the Freelancing Website**

 **User Authentication and Authorization:**

* Secure sign-up and login for freelancers and clients using JWT (JSON Web Token).
* Social login options (Google, Facebook) for easy access.

 **Freelancer Profiles:**

* Freelancers can create detailed profiles showcasing their skills, experience, certifications, and portfolio.
* Upload profile pictures, portfolio samples, and resumes.

 **Client Dashboards:**

* Clients can create accounts to post projects, track ongoing projects, and manage their freelance hires.
* Search and filter features to find freelancers by skill, experience, or ratings.

 **Job Posting and Bidding:**

* Clients can post project requirements with detailed descriptions and budgets.
* Freelancers can submit proposals or bids on listed projects.

 **Advanced Search and Filtering:**

* Search for freelancers or projects using criteria like skills, location, budget, and rating.
* Dynamic filters for real-time results.

 **Real-Time Messaging System:**

* Integrated chat system to facilitate communication between clients and freelancers.
* Notifications for new messages and updates.

 **Payment Gateway Integration:**

* Secure and reliable payment options (e.g., Stripe, PayPal).
* Clients can pay for projects, and freelancers can withdraw earnings.
* Milestone-based payment options for larger projects.

 **Project Management Tools:**

* Built-in tools to manage project timelines, track progress, and share files.
* Status updates (e.g., "In Progress," "Completed") to keep everyone informed.

 **Reviews and Ratings:**

* Clients can leave feedback and rate freelancers based on project outcomes.
* Freelancers can also rate clients, ensuring accountability.

 **Admin Panel for Platform Management:**

* Admins can manage users, moderate content, and resolve disputes.
* Analytics dashboard to track platform activity and performance.

 **Mobile-Responsive Design:**

* Fully responsive layout ensuring accessibility across devices, including smartphones and tablets.

 **Notifications System:**

* Email and in-app notifications for job postings, new bids, payments, and messages.

**SETUP INSTRUCTIONS**

**PREREQUISITES**

To set up and run the Freelancer Website project using the MERN stack, the following software dependencies and tools are required:

**Software Dependencies:**

**1.Node.js**

Required to run the backend server and manage dependencies.

[Download Node.js](https://nodejs.org)

**2.MongoDB**

Database to store user details, project data, and transactions.

Install a local MongoDB instance or use a cloud solution likeMongoDB Compass.

**3.React.js**

Frontend framework for building the user interface.

Runs in the browser, typically managed with npm.

**4.Express.js**

Web framework for the backend to handle API requests.

**5.npm (Node Package Manager):**

Comes with Node.js and is required to install project dependencies.

**Development Tools:**

**1.Git**

For version control and managing code repositories.

[Download Git](https://git-scm.com)

**2.Code Editor**

Recommended: Visual Studio Code (VS Code).

[Download VS Code](https://code.visualstudio.com)

**3.Postman**

For testing APIs during backend development.

[Download Postman](https://www.postman.com)

**4.Browser**

A modern web browser for testing the frontend, such as Google Chrome or Firefox.

**Installed Node.js Dependencies:**

1. **Backend Dependencies:**
   * express: Web server framework.
   * mongoose: MongoDB object modeling tool.
   * jsonwebtoken: For user authentication using JWT.
   * bcryptjs: For hashing passwords securely.
   * dotenv: For managing environment variables.

Install them using:

bash

Copy code

npm install express mongoose jsonwebtoken bcryptjs dotenv

1. **Frontend Dependencies:**
   * react: Core React library for building UI.
   * react-router-dom: For handling navigation between pages.
   * axios: For making HTTP requests.
   * redux (optional): For state management (if needed).

Install them using:

bash

Copy code

npm install react react-router-dom axios

**Environment Variables:**

Set up a .env file in the root directory to manage configuration:

env

Copy code

MONGO\_URI=<Your MongoDB connection string>

JWT\_SECRET=<Your secret key for JWT>

PORT=5000

**Installation: Step-by-Step Guide for Freelancer Website Project**

**1. Clone the Repository**

To get the project files on your local machine:

bash

Copy code

git clone <repository\_url>

cd <project\_directory>

**2. Install Dependencies**

1. **Backend Dependencies:**
   * Navigate to the backend folder (e.g., server/) if it's structured separately:

bash

Copy code

cd server

npm install

* + This installs the following dependencies:
    - express
    - mongoose
    - jsonwebtoken
    - bcryptjs
    - dotenv

1. **Frontend Dependencies:**
   * Navigate to the frontend folder (e.g., client/) if the frontend is in a separate directory:

bash

Copy code

cd client

npm install

* + This installs the following dependencies:
    - react
    - react-router-dom
    - axios

**3. Set Up Environment Variables**

* Create a .env file in the **backend directory** (or wherever the backend code is located).
* Add the following environment variables:

env

Copy code

MONGO\_URI=<Your MongoDB connection string>

JWT\_SECRET=<Your secret key>

PORT=5000

* + **MONGO\_URI**: Replace with your MongoDB connection string (local or cloud).
  + **JWT\_SECRET**: A strong secret key for generating and verifying JWT tokens.
  + **PORT**: Port number to run your backend server.

**4. Start the Development Server**

1. **Start Backend Server:**
   * Navigate to the backend directory:

bash

Copy code

cd server

npm start

* + The server should start running on the defined PORT (e.g., http://localhost:5000).

1. **Start Frontend Development Server:**
   * Navigate to the frontend directory:

bash

Copy code

cd client

npm start

* + The React development server will typically run at http://localhost:3000.

**5. Test the Application**

* Open your browser and test the frontend at http://localhost:3000.
* Verify the backend by making API requests using tools like Postman or directly through the frontend.

**Software Requirements:** Ensure **Node.js**, **MongoDB**, and **Git** are installed before starting.

**Dependencies:** All mentioned backend (express, mongoose, etc.) and frontend (react, axios, etc.) dependencies are installed during the npm install steps.

**Environment Variables:** Properly setting up the .env file aligns with the prerequisites for MongoDB, JWT, and server configuration.

**Database Setup**

* **Local MongoDB Setup:**
  + If running MongoDB locally, ensure MongoDB is installed and the service is running.
  + Create a new database (e.g., freelancer\_website) using the MongoDB shell or GUI tools like **MongoDB Compass**.
  + Update the MONGO\_URI in the .env file to point to your local database:

env

Copy code

MONGO\_URI=mongodb://localhost:27017/freelancer\_website

**Cloud MongoDB Setup (MongoDB Atlas):**

* + Create a cluster on [MongoDB Atlas](https://www.mongodb.com/cloud/atlas).
  + Whitelist your IP address.
  + Get the connection string and replace it in your .env file.

**Install Development Tools**

**Nodemon for Backend Development:**  
Use **Nodemon** to automatically restart the server during development:

### **Initialize Git and Push to a Repository**

1. Initialize a Git repository:

bash

Copy code

git init

git add .

git commit -m "Initial commit"

git branch -M main

git remote add origin <your-repository-url>

git push -u origin main

1. Ensure your backend and frontend are in separate folders (if applicable) and push them both.

### **Configure CORS (Cross-Origin Resource Sharing)**

If the frontend and backend are hosted on different servers or ports, configure CORS to allow them to communicate:

* Install cors in your backend:

bash

Copy code

npm install cors

* Add it to your backend code (e.g., app.js):

javascript

Copy code

const cors = require('cors');

app.use(cors());

### **Testing**

1. **Backend Testing:**
   * Use Postman to test API endpoints and ensure all routes are functioning as expected.
   * Example test: Test user login and registration APIs with valid/invalid inputs.
2. **Frontend Testing:**
   * Test the user interface manually to ensure all pages are responsive and functional.
   * Optionally, use testing libraries like **Jest** or **React Testing Library**:

bash

Copy code

npm install --save-dev jest react-testing-library

### **Linting and Code Formatting**

* Install **ESLint** and **Prettier** to ensure consistent code quality:

bash

Copy code

npm install --save-dev eslint prettier

* Configure .eslintrc.json for your project:

json

Copy code

{

"env": {

"browser": true,

"es2021": true

},

"extends": "eslint:recommended",

"parserOptions": {

"ecmaVersion": 12,

"sourceType": "module"

},

"rules": {}

}

* Add a Prettier configuration (.prettierrc):

json

Copy code

{

"singleQuote": true,

"semi": false

}

### **Deployment**

1. **Deploy Backend:**
   * Host the backend on a platform like [Heroku](https://www.heroku.com/), [Railway](https://railway.app/), or [AWS EC2].
   * Set environment variables (e.g., MONGO\_URI, JWT\_SECRET) in the platform's environment settings.
2. **Deploy Frontend:**
   * Build the React app:

bash

Copy code

npm run build

* + Deploy the build/ directory to a hosting platform like [Netlify](https://www.netlify.com/), [Vercel](https://vercel.com/), or [AWS S3].

1. **Configure Proxy (Optional):**
   * If hosting the frontend and backend separately, configure a proxy in your React app's package.json:

json

Copy code

"proxy": "http://localhost:5000"

### **Monitor and Log Errors**

1. Add error-handling middleware in the backend:

javascript

Copy code

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).send('Something broke!');

});

1. Use tools like **Sentry** for monitoring:

bash

Copy code

npm install @sentry/node

### **Database Backup**

* If using MongoDB, ensure regular backups of your database:
  + Use **MongoDB Atlas backup options** or tools like mongodump.

### **Future Enhancements (Optional)**

* **Add Unit Tests:** Use testing frameworks like Mocha, Chai, or Jest.
* **Optimize SEO:** Improve metadata, sitemaps, and social media previews for better search rankings.
* **Enable SSL:** Use HTTPS for secure communication when deploying.

**FOLDER STRUCTURE**

**CLIENT**

### **1. Folder Structure**

plaintext

Copy code

client/

├── public/

├── src/

│ ├── assets/

│ ├── components/

│ ├── pages/

│ ├── context/

│ ├── utils/

│ ├── hooks/

│ ├── services/

│ ├── App.js

│ ├── index.js

│ ├── styles/

│ ├── constants/

│ ├── routes/

│ ├── api/

└── package.json

### **2. Folder/Module Breakdown**

#### ****Public Folder (****public/****)****

* Contains static files like:
  + index.html: The main HTML file where the React app is rendered.
  + Favicon and other public assets.

#### ****Source Folder (****src/****)****

This folder contains the core functionality and logic of the application.

1. **assets/:**
   * Stores static files like images, fonts, icons, and other assets.
   * Example:

plaintext

Copy code

assets/

├── images/

├── icons/

├── fonts/

1. **components/:**
   * Reusable UI components, such as buttons, headers, footers, forms, and modals.
   * Example components for a freelancing site:
     + Navbar.jsx: Top navigation bar.
     + JobCard.jsx: Component for displaying individual job postings.
     + FreelancerProfile.jsx: Displays freelancer details.
   * Example structure:

plaintext

Copy code

components/

├── Navbar.jsx

├── JobCard.jsx

├── FreelancerProfile.jsx

├── Modal.jsx

└── Button.jsx

1. **pages/:**
   * Contains the main pages of the application.
   * Example pages:
     + Home.jsx: Displays the homepage with job listings and featured freelancers.
     + Login.jsx: Login form for users.
     + Register.jsx: Signup form for freelancers and clients.
     + Dashboard.jsx: Freelancer or client dashboard.
     + JobDetails.jsx: Detailed view of a job posting.
   * Example structure:

plaintext

Copy code

pages/

├── Home.jsx

├── Login.jsx

├── Register.jsx

├── Dashboard.jsx

├── JobDetails.jsx

1. **context/:**
   * Manages global state using **React Context API** (or optionally Redux).
   * Example:
     + AuthContext.jsx: For managing user authentication and login state.
     + JobContext.jsx: For managing job-related data.
2. **utils/:**
   * Utility functions for common operations.
   * Examples:
     + formatDate.js: Format dates for job postings.
     + validation.js: Input validation functions.
3. **hooks/:**
   * Custom React hooks for reusability.
   * Examples:
     + useAuth.js: Hook to get the current user’s authentication status.
     + useFetch.js: Hook to manage API calls.
4. **services/:**
   * API service files to interact with the backend.
   * Examples:
     + authService.js: Handles login and registration API requests.
     + jobService.js: Manages API calls for job postings and applications.
5. **styles/:**
   * Global and component-specific stylesheets.
   * Example:
     + global.css: Application-wide styles.
     + Navbar.css: Specific styles for the Navbar component.
6. **constants/:**
   * Holds constants like API base URLs, route paths, or application-wide settings.
   * Example:

javascript

Copy code

export const API\_BASE\_URL = "http://localhost:5000/api";

1. **routes/:**
   * Defines the routing structure of the app using **React Router**.
   * Example:

javascript

Copy code

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

import Home from "../pages/Home";

import Login from "../pages/Login";

const AppRoutes = () => (

<Router>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/login" element={<Login />} />

</Routes>

</Router>

);

export default AppRoutes;

1. **api/:**
   * Stores the backend API configuration.
   * Example:

javascript

Copy code

import axios from "axios";

const api = axios.create({

baseURL: "http://localhost:5000/api",

headers: { "Content-Type": "application/json" },

});

export default api;

### **3. Key React Files**

1. **App.js:**
   * Root component that integrates all pages, routes, and global components like Navbar and Footer.
   * Example:

javascript

Copy code

import React from "react";

import AppRoutes from "./routes/AppRoutes";

const App = () => (

<>

<AppRoutes />

</>

);

export default App;

1. **index.js:**
   * Main entry point of the application.
   * Example:

javascript

Copy code

import React from "react";

import ReactDOM from "react-dom";

import App from "./App";

import "./styles/global.css";

ReactDOM.render(<App />, document.getElementById("root"));

### **4. Key Features for Frontend (React)**

* **Job Listings:** Display job postings dynamically fetched from the backend.
* **User Authentication:** Login and registration forms for freelancers and clients.
* **Dashboard:** Separate dashboards for freelancers (e.g., applied jobs) and clients (e.g., posted jobs).
* **Responsive Design:** Mobile-friendly layout for users accessing the platform on smaller screens.
* **Real-Time Notifications:** Integrate WebSockets or polling for real-time updates on job applications and messages.
* **Payment Integration:** (Optional) Display payment options or status for completed jobs.

**SERVER**

## **1.Directory Structure**

Organize the backend into a modular and scalable directory structure. Here's an example:

bash

Copy code

backend/

├── config/ # Configuration files (e.g., DB, environment variables)

│ ├── db.js

│ ├── env.js

├── controllers/ # Request handlers

│ ├── authController.js

│ ├── userController.js

│ ├── projectController.js

├── models/ # Database models

│ ├── User.js

│ ├── Project.js

│ ├── Payment.js

├── routes/ # API routes

│ ├── authRoutes.js

│ ├── userRoutes.js

│ ├── projectRoutes.js

├── middlewares/ # Middleware functions

│ ├── authMiddleware.js

│ ├── errorHandler.js

├── services/ # Business logic (e.g., payment processing, notifications)

│ ├── paymentService.js

│ ├── emailService.js

├── utils/ # Utility functions

│ ├── helpers.js

│ ├── validators.js

├── app.js # Express app setup

├── server.js # Server entry point

├── package.json # Dependencies and scripts

└── README.md # Documentation

## **2. Key Features and Implementation**

### **a. Authentication**

* **Use JWT** for stateless authentication.
* Implement **role-based access control** (e.g., admin, freelancer, client).
* Example: authController.js

javascript

Copy code

const jwt = require('jsonwebtoken');

const login = async (req, res) => {

const { email, password } = req.body;

const user = await User.findOne({ email });

if (!user || !user.comparePassword(password)) {

return res.status(401).json({ message: "Invalid credentials" });

}

const token = jwt.sign({ id: user.id, role: user.role }, process.env.JWT\_SECRET);

res.json({ token });

};

### **b. User Management**

* Handle profile creation, updates, and portfolio uploads.
* Separate user roles in the User model.

javascript

Copy code

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

name: String,

email: String,

password: String,

role: { type: String, enum: ['client', 'freelancer', 'admin'] },

profile: Object,

});

module.exports = mongoose.model('User', userSchema);

### **c. Project Management**

* Clients can post projects, freelancers can bid.
* Include features like project status (open, in-progress, completed).
* Example: projectController.js

javascript

Copy code

const createProject = async (req, res) => {

const { title, description, budget } = req.body;

const project = new Project({ title, description, budget, client: req.user.id });

await project.save();

res.status(201).json(project);

};

### **d. Payment Integration**

* Use payment gateways like **Stripe** or **PayPal**.
* Handle **escrow services** for secure payments.
* Example: paymentService.js

javascript

Copy code

const stripe = require('stripe')(process.env.STRIPE\_SECRET);

const createPaymentIntent = async (amount, currency) => {

return await stripe.paymentIntents.create({

amount,

currency,

});

};

### **e. Notifications**

* Use WebSocket (e.g., **Socket.IO**) for real-time notifications.
* Example use case: Notify freelancers of new bids.

## **3. API Structure**

Define clear and RESTful API endpoints. Example:

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/auth/login | POST | Login user |
| /api/auth/register | POST | Register user |
| /api/users/profile | GET | Fetch user profile |
| /api/projects | POST | Create a new project |
| /api/projects/:id | GET | Fetch project details |
| /api/payments | POST | Process payment |

## **4. Middleware**

* **Auth Middleware**: Check JWT and attach user to the request.
* **Error Handling**: Use a centralized error handler.

javascript

Copy code

const errorHandler = (err, req, res, next) => {

res.status(err.status || 500).json({ message: err.message });

};

## **5. Technologies and Tools**

* **Framework**: [Express.js](https://expressjs.com/)
* **Database**: MongoDB (via [Mongoose](https://mongoosejs.com/))
* **Authentication**: JWT, Passport.js
* **Real-time Communication**: Socket.IO
* **Payment Gateway**: Stripe, PayPal
* **Environment Variables**: dotenv

## **6. Deployment**

* Use **Docker** for containerization.

**RUNNING THE APPLICATION**

### **Frontend: Client Directory**

1. Navigate to the frontend directory (commonly named client or frontend):

bash

Copy code

cd client

1. Install dependencies:

bash

Copy code

npm install

1. Start the development server:

bash

Copy code

npm start

* + **Default Port**: The React or Vue.js frontend typically starts at http://localhost:3000.
  + **Environment Variables**: Ensure the API\_URL or REACT\_APP\_API\_URL in your .env file points to the backend (e.g., http://localhost:5000).

### **Backend: Server Directory**

1. Navigate to the backend directory (commonly named server or backend):

bash

Copy code

cd backend

1. Install dependencies:

bash

Copy code

npm install

1. Start the backend server:

bash

Copy code

npm start

* + If using **Nodemon** for development:

bash

Copy code

npm run dev

* + **Default Port**: The Express backend usually starts at http://localhost:5000.
  + **Environment Variables**: Ensure the .env file in the backend contains the correct settings for:
    - Database (e.g., MongoDB connection string).
    - Payment gateways (e.g., Stripe/PayPal keys).
    - JWT secrets.

### **Starting Both Simultaneously**

If you want to start both servers together for your freelancing website, you can use the **concurrently** package.

1. Install concurrently in the root directory:

bash

Copy code

npm install concurrently --save-dev

1. Add a script to the root package.json:

json

Copy code

"scripts": {

"start:all": "concurrently \"cd backend && npm run dev\" \"cd client && npm start\""

}

1. Run the combined command:

bash

Copy code

npm run start:all

* **Frontend Integration**:
  + Set the API base URL in the frontend .env file:

arduino

Copy code

REACT\_APP\_API\_URL=http://localhost:5000

* **Backend Integration**:
  + Use **CORS** middleware to allow API calls from the frontend:

javascript

Copy code

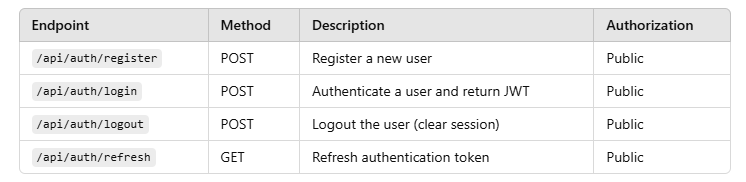
const cors = require('cors');

app.use(cors({ origin: 'http://localhost:3000' }));

* **Database**:
  + Ensure your database (e.g., MongoDB) is running locally or connected to a cloud service like MongoDB Atlas.

**API DOCUMENTATION**

## **1.Authentication and User Management**



### Payload Examples:

* **Register**:

json

Copy code

{

"name": "John Doe",

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

"password": "securePassword",

"role": "freelancer"

}

* **Login**:

json

Copy code

{

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

"password": "securePassword"

}

## **2. User Profile**

### 

### Payload Example (Update Profile):

json

Copy code

{

"name": "John Doe",

"skills": ["JavaScript", "React", "Node.js"],

"bio": "Experienced web developer specializing in full-stack development."

}

## **3. Project Management**

### 

### Payload Example (Create Project):

json

Copy code

{

"title": "Develop a Company Website",

"description": "Looking for a frontend and backend developer to create a responsive website.",

"budget": 2000,

"deadline": "2024-12-01"

}

### Payload Example (Place Bid):

json

Copy code

{

"amount": 1500,

"message": "I have 5 years of experience in web development and can deliver the project on time."

}

## **4. Payment Processing**

### 

### Payload Example (Create Payment Intent):

json

Copy code

{

"projectId": "642be7125a4e57a94f88a62b",

"amount": 2000

}

## **5. Messaging and Notifications**

### 

### Payload Example (Send Message):

json

Copy code

{

"projectId": "642be7125a4e57a94f88a62b",

"recipientId": "642be7125a4e57a94f88a700",

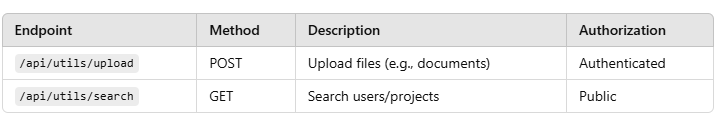
"message": "Hello, I'm interested in your project."

}

## **6. Admin Operations**

## 

## **7. Utilities**



## **Error Handling**

* **Standard Error Response**:

json

Copy code

{

"error": true,

"message": "Resource not found",

"statusCode": 404

}

## **1. Authentication and User Management**

### **1.1 Register**

* **Endpoint**: /api/auth/register
* **Method**: POST
* **Description**: Register a new user.
* **Request Body**:

json

Copy code

{

"name": "John Doe",

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

"password": "password123",

"role": "freelancer"

}

* **Response**:

json

Copy code

{

"message": "User registered successfully",

"user": {

"id": "64b78c9021d9e12f3e0c",

"name": "John Doe",

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

"role": "freelancer"

}

}

### **1.2 Login**

* **Endpoint**: /api/auth/login
* **Method**: POST
* **Description**: Authenticate a user and return a JWT token.
* **Request Body**:

json

Copy code

{

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

"password": "password123"

}

* **Response**:

json

Copy code

{

"message": "Login successful",

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

}

### **1.3 Get User Profile**

* **Endpoint**: /api/users/profile
* **Method**: GET
* **Description**: Fetch the logged-in user's profile.
* **Headers**:

makefile

Copy code

Authorization: Bearer <JWT Token>

* **Response**:

json

Copy code

{

"id": "64b78c9021d9e12f3e0c",

"name": "John Doe",

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

"role": "freelancer",

"profile": {

"skills": ["JavaScript", "Node.js"],

"bio": "Experienced full-stack developer."

}

}

## **2. Project Management**

### **2.1 Get All Projects**

* **Endpoint**: /api/projects
* **Method**: GET
* **Description**: Fetch all available projects.
* **Response**:

json

Copy code

[

{

"id": "64b8a7f09a9e9b2e6b01",

"title": "Build a website",

"description": "Need a responsive website.",

"budget": 1500,

"deadline": "2024-12-01",

"status": "open"

},

{

"id": "64b8a8e59a9e9b2e6b02",

"title": "Develop a mobile app",

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

"budget": 3000,

"deadline": "2024-11-30",

"status": "open"

}

]

### **2.2 Create a Project**

* **Endpoint**: /api/projects
* **Method**: POST
* **Description**: Create a new project (for clients only).
* **Headers**:

makefile

Copy code

Authorization: Bearer <JWT Token>

* **Request Body**:

json

Copy code

{

"title": "Design a logo",

"description": "Looking for a creative logo designer.",

"budget": 500,

"deadline": "2024-12-15"

}

* **Response**:

json

Copy code

{

"message": "Project created successfully",

"project": {

"id": "64b9c7d79a9e9b2e6c03",

"title": "Design a logo",

"description": "Looking for a creative logo designer.",

"budget": 500,

"deadline": "2024-12-15",

"status": "open"

}

}

### **2.3 Place a Bid**

* **Endpoint**: /api/projects/:id/bid
* **Method**: POST
* **Description**: Place a bid on a project (for freelancers only).
* **Headers**:

makefile

Copy code

Authorization: Bearer <JWT Token>

* **Request Parameters**:
  + id (Path Parameter): The project ID.
* **Request Body**:

json

Copy code

{

"amount": 450,

"message": "I have extensive experience in logo design and can deliver high-quality work."

}

* **Response**:

json

Copy code

{

"message": "Bid placed successfully",

"bid": {

"id": "64ba3f129a9e9b2e6c04",

"projectId": "64b9c7d79a9e9b2e6c03",

"amount": 450,

"freelancerId": "64b78c9021d9e12f3e0c",

"message": "I have extensive experience in logo design and can deliver high-quality work."

}

}

## **3. Payment Processing**

### **3.1 Create Payment Intent**

* **Endpoint**: /api/payments/create
* **Method**: POST
* **Description**: Create a payment intent for a project.
* **Headers**:

makefile

Copy code

Authorization: Bearer <JWT Token>

* **Request Body**:

json

Copy code

{

"projectId": "64b9c7d79a9e9b2e6c03",

"amount": 500

}

* **Response**:

json

Copy code

{

"message": "Payment intent created",

"paymentIntent": {

"id": "pi\_1Lb6e12rK3YE9hDtvEV7",

"amount": 500,

"status": "requires\_payment\_method"

}

}

### **3.2 Release Escrow**

* **Endpoint**: /api/payments/release
* **Method**: POST
* **Description**: Release funds from escrow to the freelancer.
* **Headers**:

makefile

Copy code

Authorization: Bearer <JWT Token>

* **Request Body**:

json

Copy code

{

"projectId": "64b9c7d79a9e9b2e6c03"

}

* **Response**:

json

Copy code

{

"message": "Funds released to freelancer successfully",

"transaction": {

"id": "txn\_1Lb6fA2rK3YE9hDtrEV8",

"amount": 500,

"status": "succeeded"

}

}

## **4. Messaging**

### **4.1 Send a Message**

* **Endpoint**: /api/messages
* **Method**: POST
* **Description**: Send a message to a user regarding a project.
* **Headers**:

makefile

Copy code

Authorization: Bearer <JWT Token>

* **Request Body**:

json

Copy code

{

"projectId": "64b9c7d79a9e9b2e6c03",

"recipientId": "64b78c9021d9e12f3e0c",

"message": "Hello, are you available for this project?"

}

* **Response**:

json

Copy code

{

"message": "Message sent successfully",

"chat": {

"id": "64bacf989a9e9b2e6c05",

"senderId": "64b78c9021d9e12f3e0c",

"recipientId": "64b78c9021d9e12f3e0d",

"content": "Hello, are you available for this project?"

}

}

**AUTHENTICATION**

### **1. Authentication**

Authentication verifies a user's identity before granting access to the system. In this project, authentication is handled using **JWT (JSON Web Tokens)**.

#### ****Steps in Authentication Workflow:****

1. **User Registration**:
   * **Endpoint**: /api/auth/register
   * New users (clients or freelancers) provide their details (name, email, password, etc.).
   * Passwords are hashed using a library like **bcrypt** for security before storing in the database.
2. **User Login**:
   * **Endpoint**: /api/auth/login
   * Users provide their email and password. The backend:
     + Verifies the email exists.
     + Compares the password using the bcrypt library.
   * If valid, a **JWT token** is issued containing:
     + User ID
     + User role (client or freelancer)
     + Expiry time (e.g., 24 hours).
   * The token is signed with a secret key (stored in .env as JWT\_SECRET).
3. **Token Handling**:
   * The JWT token is sent to the client in the response.
   * The client stores the token in:
     + **LocalStorage** (for persistence between sessions) or
     + **HTTP-Only Cookies** (for better security against XSS attacks).
4. **Token Refresh** (Optional):
   * **Endpoint**: /api/auth/refresh
   * If the token expires, a refresh token (stored securely) is used to generate a new JWT.

### **2. Authorization**

Authorization ensures that users can only perform actions they are allowed to based on their role (client or freelancer).

#### ****Implementation Workflow:****

1. **Role-Based Access Control (RBAC)**:
   * Users are assigned roles (e.g., client, freelancer, or admin) during registration.
   * Each role has specific permissions:
     + **Client**: Can post projects, accept bids, release payments.
     + **Freelancer**: Can view projects, place bids, and receive payments.
     + **Admin**: Can manage all user accounts and projects.
2. **Middleware for Authorization**:
   * Requests to protected routes include a JWT in the Authorization header:

makefile

Copy code

Authorization: Bearer <JWT\_TOKEN>

* + A middleware function verifies the token:
    - Decodes the token using the secret key.
    - Extracts the user ID and role.
  + Example middleware:

javascript

Copy code

const jwt = require('jsonwebtoken');

const authenticate = (req, res, next) => {

const token = req.headers.authorization?.split(" ")[1];

if (!token) return res.status(401).json({ error: "Unauthorized" });

jwt.verify(token, process.env.JWT\_SECRET, (err, user) => {

if (err) return res.status(403).json({ error: "Invalid token" });

req.user = user; // Attach user data to the request

next();

});

};

1. **Route Protection**:
   * Routes are protected by adding the authenticate middleware:

javascript

Copy code

app.get('/api/users/profile', authenticate, (req, res) => {

res.json({ profile: req.user });

});

1. **Role-Specific Authorization**:
   * Additional middleware checks the user's role:

javascript

Copy code

const authorize = (role) => (req, res, next) => {

if (req.user.role !== role) {

return res.status(403).json({ error: "Access denied" });

}

next();

};

* + Example usage for a client-only route:

javascript

Copy code

app.post('/api/projects', authenticate, authorize('client'), (req, res) => {

// Create a project

});

### **3. Security Measures**

To enhance the security of authentication and authorization:

1. **Password Hashing**:
   * Use **bcrypt** to hash passwords before storing them in the database.
   * Example:

javascript

Copy code

const bcrypt = require('bcrypt');

const hashedPassword = await bcrypt.hash(password, 10);

1. **Token Expiry**:
   * Set a short expiry time for JWTs (e.g., 15 minutes to 1 hour).
   * Implement refresh tokens for long-lived sessions.
2. **Secure Token Storage**:
   * Store access tokens in **HTTP-only cookies** to prevent XSS attacks.
   * Use **Secure** and **SameSite** cookie flags.
3. **HTTPS**:
   * Enforce HTTPS to encrypt communication between the client and server.
4. **CORS Policy**:
   * Configure the backend to accept requests only from trusted frontend origins:

javascript

Copy code

const cors = require('cors');

app.use(cors({ origin: 'https://your-frontend-domain.com' }));

1. **Input Validation**:
   * Use libraries like **Joi** or **Express Validator** to validate user inputs during login, registration, and API requests.

### **4. Example User Flow**

1. **Client logs in**: Receives a JWT token.
2. **Client posts a project**: Includes the token in the request headers. Middleware verifies the token and allows access if valid.
3. **Freelancer places a bid**: Verifies freelancer's role via middleware before proceeding.
4. **Client accepts a bid**: Role and ownership are checked to ensure only the project owner can perform the action.

### **5. Summary**

* **Authentication**: Handled using JWT for stateless, secure user sessions.
* **Authorization**: Implemented through role-based access control and middleware.
* **Security**: Enhanced through password hashing, token expiry, HTTPS, and CORS policies

## **Tokens, Sessions, and Authentication Details**

### **1. JSON Web Tokens (JWT)**

JWT is the primary method for authentication in this project. It provides a stateless, secure way to authenticate users and manage sessions.

#### ****How JWT Works in the Project****:

1. **Token Generation**:
   * After a user successfully logs in, the backend generates a JWT using a library like **jsonwebtoken**.
   * The token contains:
     + **Header**: Specifies the type of token (JWT) and the signing algorithm (HS256).
     + **Payload**: Includes user details such as id, email, and role.

json

Copy code

{

"id": "64b78c9021d9e12f3e0c",

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

"role": "freelancer",

"iat": 1692038476, // Issued at timestamp

"exp": 1692124876 // Expiry timestamp

}

* + - **Signature**: A hash of the header and payload, signed with a secret key (JWT\_SECRET) to ensure token integrity.

1. **Token Structure**:
   * The token is a base64-encoded string consisting of three parts: <header>.<payload>.<signature>.
2. **Token Storage**:
   * The client stores the JWT:
     + In **localStorage** or **sessionStorage** for single-page applications (easier but less secure).
     + In **HTTP-only cookies** for enhanced security against XSS attacks.
3. **Token Expiry**:
   * Tokens are short-lived, typically valid for 15 minutes to 1 hour, and contain an exp (expiry time) claim in the payload.

### **2. Token Usage in API Requests**

* For accessing protected routes, the client includes the JWT in the **Authorization** header:

makefile

Copy code

Authorization: Bearer <JWT\_TOKEN>

* On the backend, middleware verifies the token:
  1. Extracts the token from the Authorization header.
  2. Verifies the token’s signature using JWT\_SECRET.
  3. Decodes the payload and attaches user details to the request object (req.user).

Example verification middleware:

javascript

Copy code

const jwt = require('jsonwebtoken');

const authenticate = (req, res, next) => {

const token = req.headers.authorization?.split(" ")[1];

if (!token) {

return res.status(401).json({ error: "Unauthorized. Token missing." });

}

jwt.verify(token, process.env.JWT\_SECRET, (err, user) => {

if (err) return res.status(403).json({ error: "Invalid or expired token." });

req.user = user; // Attach decoded user info to the request object

next();

});

};

### **3. Refresh Tokens**

Since access tokens are short-lived, refresh tokens are used to maintain user sessions without requiring frequent logins.

#### ****Workflow****:

1. **Refresh Token Generation**:
   * A refresh token (longer-lived, e.g., 7 days) is issued alongside the access token during login.
   * Stored securely in an **HTTP-only cookie**.
2. **Refreshing Tokens**:
   * When the access token expires, the client sends a request to /api/auth/refresh with the refresh token in the cookie.
   * The backend verifies the refresh token and issues a new access token.
3. **Storage**:
   * Refresh tokens are stored in a secure database table and invalidated upon logout or when compromised.

Example refresh route:

javascript

Copy code

app.post('/api/auth/refresh', (req, res) => {

const refreshToken = req.cookies.refreshToken;

if (!refreshToken) return res.status(403).json({ error: "No refresh token provided." });

jwt.verify(refreshToken, process.env.JWT\_SECRET, (err, user) => {

if (err) return res.status(403).json({ error: "Invalid refresh token." });

const newAccessToken = jwt.sign(

{ id: user.id, role: user.role },

process.env.JWT\_SECRET,

{ expiresIn: '1h' }

);

res.json({ accessToken: newAccessToken });

});

});

### **4. Sessions (Optional Approach)**

While the primary method is JWT, sessions can also be implemented for specific use cases, such as user activity tracking or short-term interactions.

#### ****How Sessions Work****:

1. **Session Storage**:
   * Sessions are stored server-side in memory, a database (e.g., Redis), or files.
   * Each session has a unique session ID sent to the client in a cookie.
2. **Session-Based Authentication**:
   * On login, the server creates a session and sends a cookie containing the session ID.
   * On subsequent requests, the server looks up the session ID to verify the user's identity.

Example using Express and express-session:

javascript

Copy code

const session = require('express-session');

app.use(session({

secret: 'your\_secret\_key',

resave: false,

saveUninitialized: true,

cookie: { secure: true } // Only over HTTPS

}));

### **5. Role-Based Authorization**

Authorization ensures that users can access resources or perform actions based on their roles (e.g., client, freelancer, admin).

#### ****Implementation****:

1. **Role in JWT Payload**:
   * Include the user’s role in the token payload during login.

json

Copy code

{ "id": "64b78c9021d9e12f3e0c", "role": "client" }

1. **Middleware for Role Verification**:
   * Add middleware to check the user’s role for protected routes.

javascript

Copy code

const authorize = (roles) => (req, res, next) => {

if (!roles.includes(req.user.role)) {

return res.status(403).json({ error: "Access denied." });

}

next();

};

1. **Usage**:
   * Protect a route to allow only clients:

javascript

Copy code

app.post('/api/projects', authenticate, authorize(['client']), (req, res) => {

// Only clients can create projects

});

### **6. Logout Mechanism**

Logout invalidates tokens to prevent misuse.

#### ****Implementation****:

1. **Token Blacklisting**:
   * Store revoked tokens in a blacklist (e.g., in Redis).
   * Middleware checks if a token exists in the blacklist before granting access.
2. **Delete Refresh Token**:
   * Clear the refresh token from the database and the client’s cookie.

Example logout route:

javascript

Copy code

app.post('/api/auth/logout', authenticate, (req, res) => {

const refreshToken = req.cookies.refreshToken;

// Remove refresh token from DB

res.clearCookie('refreshToken');

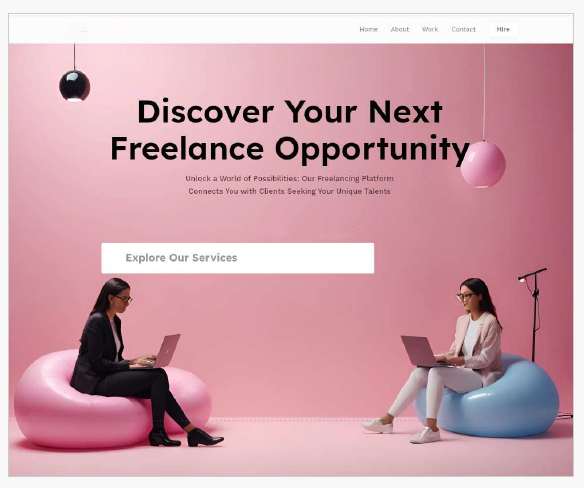
res.json({ message: "Logged out successfully." });

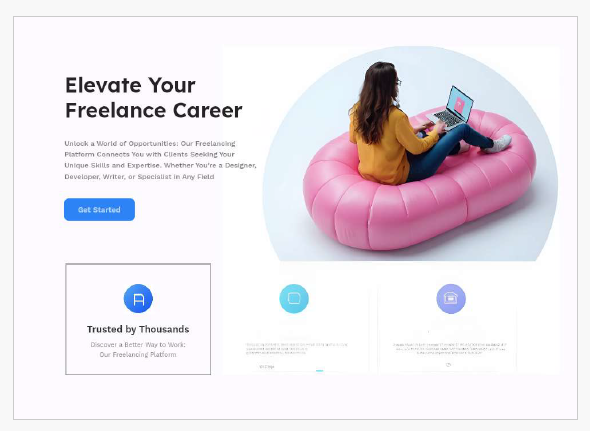
});

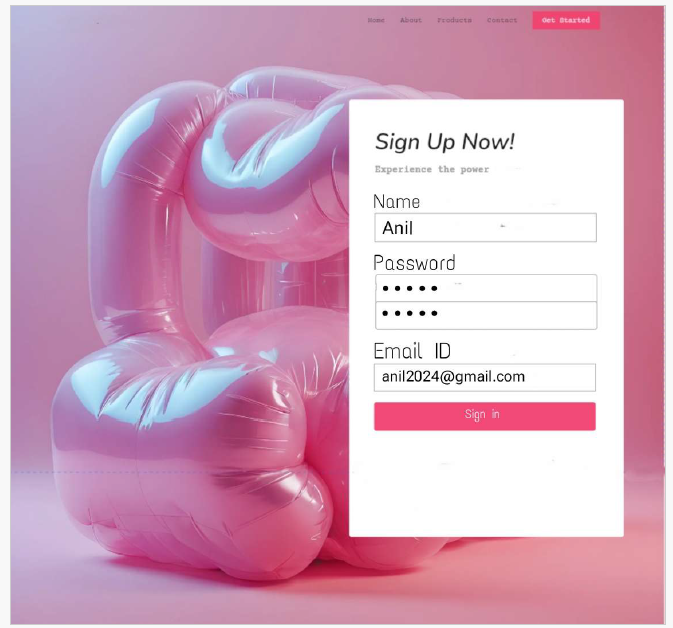
### **7. Summary**

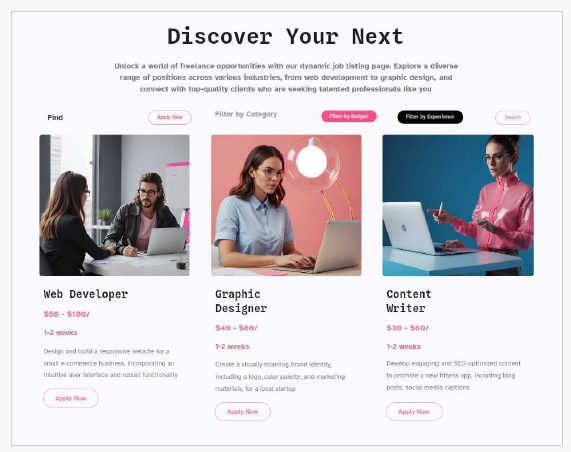
| **Method** | **Description** |
| --- | --- |
| **JWT** | Stateless tokens for authentication; stored in localStorage or HTTP-only cookies. |
| **Refresh Tokens** | Long-lived tokens for renewing expired JWTs. Stored in secure cookies. |
| **Sessions** | Optional server-side approach for specific use cases. |
| **Role-Based Access** | Middleware ensures users have appropriate permissions. |
| **Security Enhancements** | HTTPS, cookie flags, token blacklisting, and short token lifespans |

**USER INTERFACE**

****

****





**TESTING**

### **1. Functional Testing**

**Objective:** Ensure all features work as intended.

* **Focus Areas:**
  + User registration/login for freelancers and clients.
  + Profile creation and editing.
  + Job posting and bidding processes.
  + Chat/messaging system functionality.
  + Payment gateway integration.
  + Notifications (email, SMS, push).
* **Tools Used:**
  + Selenium (for automated functional testing).
  + Postman (for API testing).
  + TestRail (for test case management).

### **2. Performance Testing**

**Objective:** Verify the website’s responsiveness and stability under different conditions.

* **Focus Areas:**
  + Page load time, especially for dashboards and job listings.
  + Handling multiple concurrent users.
  + Stress and load testing for peak times.
* **Tools Used:**
  + JMeter (for load and stress testing).
  + Lighthouse (for performance auditing).
  + New Relic (for real-time performance monitoring).

### **3. Usability Testing**

**Objective:** Ensure the website is intuitive and user-friendly.

* **Focus Areas:**
  + Navigation ease across all devices (desktop, tablet, mobile).
  + Clear CTAs (e.g., "Apply Now," "Post a Job").
  + Accessibility (WCAG compliance).
* **Tools Used:**
  + Crazy Egg or Hotjar (for heatmap analysis).
  + BrowserStack (to test responsiveness across devices).
  + Axe (for accessibility testing).

### **4. Security Testing**

**Objective:** Protect user data and ensure secure transactions.

* **Focus Areas:**
  + Payment data encryption.
  + Prevention of SQL injection and cross-site scripting (XSS).
  + User authentication and authorization.
* **Tools Used:**
  + OWASP ZAP (for identifying security vulnerabilities).
  + Burp Suite (for penetration testing).
  + Nessus (for vulnerability scanning).

### **5. Compatibility Testing**

**Objective:** Ensure the website functions seamlessly across different browsers, devices, and operating systems.

* **Focus Areas:**
  + Browsers: Chrome, Firefox, Safari, Edge, etc.
  + Devices: Desktop, mobile, tablets.
  + Operating Systems: Windows, macOS, Android, iOS.
* **Tools Used:**
  + BrowserStack or Sauce Labs (for cross-browser testing).
  + LambdaTest (for cross-device compatibility).

### **6. Regression Testing**

**Objective:** Ensure new updates or features don’t break existing functionality.

* **Focus Areas:**
  + Retesting critical workflows after every update.
  + Verifying bug fixes.
* **Tools Used:**
  + Selenium with TestNG or JUnit (for automated regression testing).
  + Katalon Studio (for comprehensive regression tests).

### **7. Performance Monitoring and Ongoing Testing**

**Objective:** Maintain optimal performance and functionality post-launch.

* **Focus Areas:**
  + Monitoring server uptime.
  + Tracking user behavior for anomalies.
* **Tools Used:**
  + Google Analytics (for real-time tracking).
  + Dynatrace or Pingdom (for server monitoring).

### **Testing Workflow**

1. Define test scenarios based on features.
2. Write detailed test cases.
3. Execute manual or automated tests in sprints.
4. Log bugs using tools like JIRA or Bugzilla.
5. Conduct re-tests and regression tests after fixes.
6. Use CI/CD pipelines for automated testing during deployments

### 1. **Development Needs**

**Key Tools & Commands:**

* **Version Control:**
  + **Git:** For version control and collaboration.
    - Commands: git clone, git pull, git push, git branch, git merge
  + **Integration with GitHub/GitLab/Bitbucket.**
* **Package Management:**
  + **npm/yarn:** For managing front-end dependencies.
    - Commands: npm install, npm run start, yarn add, yarn build
  + **Composer:** For back-end dependencies (PHP frameworks like Laravel).
    - Commands: composer install, composer update
* **Build and Compilation:**
  + **Webpack/Parcel/Vite:** For asset bundling and compilation.
    - Commands: npm run build, npx webpack
* **Framework CLI Tools:**
  + For frameworks like Angular, React, or Vue.js:
    - Example: npx create-react-app, ng new, vue create
* **Database Management:**
  + MySQL, PostgreSQL, or MongoDB terminal utilities for local database management.
    - Commands: mysql -u root -p, psql, mongo

### **2. Testing Needs**

**Key Tools & Commands:**

* **Automation Testing:**
  + Selenium, Playwright, or Cypress:
    - Commands: npx cypress open, npx playwright test
* **Performance Testing:**
  + Apache JMeter:
    - Commands: jmeter -n -t test-plan.jmx
* **Unit Testing:**
  + **Jest/Mocha/Chai:** For running automated unit tests.
    - Commands: npm test, jest
* **API Testing:**
  + Postman CLI or Newman for automated API tests:
    - Commands: newman run collection.json

### **3. Deployment Needs**

**Key Tools & Commands:**

* **Server Access:**
  + **SSH:** Secure access to remote servers.
    - Commands: ssh user@server, scp file user@server:/path
* **Docker:** For containerized application deployments.
  + Commands: docker build, docker run, docker-compose up
* **CI/CD Integration:**
  + Jenkins/GitHub Actions/Travis CI:
    - Commands: jenkins-cli, .github/workflows
* **Cloud Deployment:**
  + AWS CLI, Azure CLI, or GCP CLI for managing cloud resources.
    - Example: aws s3 cp, az webapp deploy

### **4. Maintenance Needs**

**Key Tools & Commands:**

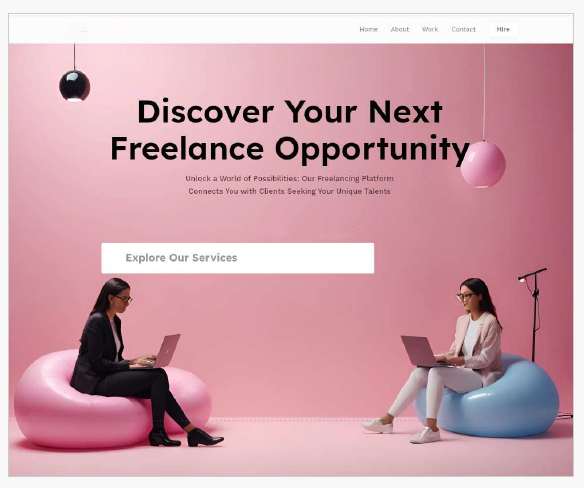
* **Log Monitoring:**
  + Tail and grep for server log analysis.
    - Commands: tail -f logs/error.log, grep -i "error" logs.log
* **Database Backups:**
  + MySQL Dump or pg\_dump:
    - Commands: mysqldump -u root -p dbname > backup.sql, pg\_dump dbname > backup.sql
* **Package Updates:**
  + Commands: npm update, composer update
* **System Health Monitoring:**
  + Tools like **htop**, **top**, or **df -h**.
    - Commands: htop, df -h, free -m

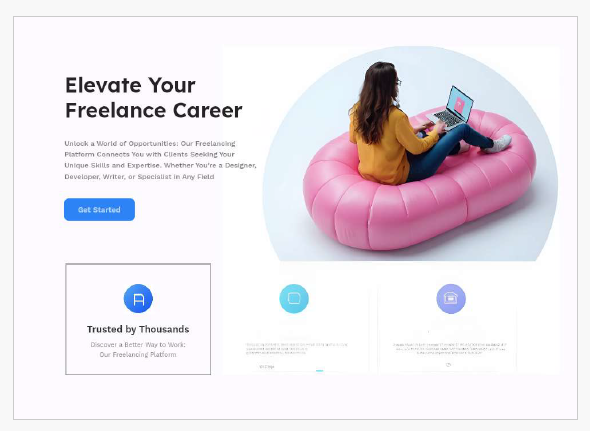
### **5. Debugging Needs**

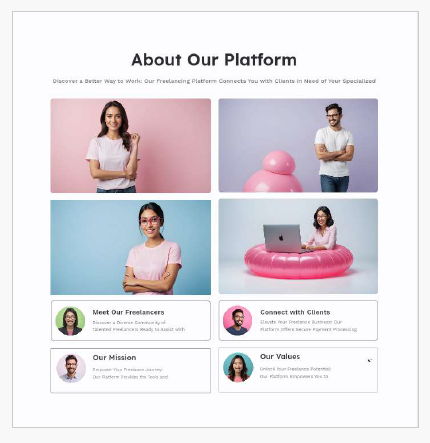
**Key Tools & Commands:**

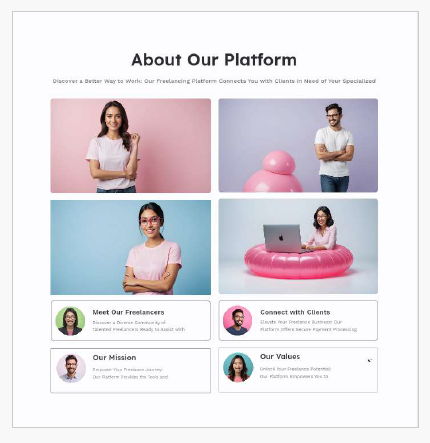
* **Error Tracking:**
  + Integration with tools like Sentry or Rollbar for real-time error logging.
    - Commands: Integrated via the application, minimal terminal setup needed.
* **Code Debugging:**
  + Debuggers for the chosen programming language:
    - Python: pdb
    - Node.js: node inspect
    - PHP: xdebug
* **Browser Debugging:**
  + Start applications with debug flags for inspection:
    - Example: npm run start --inspect

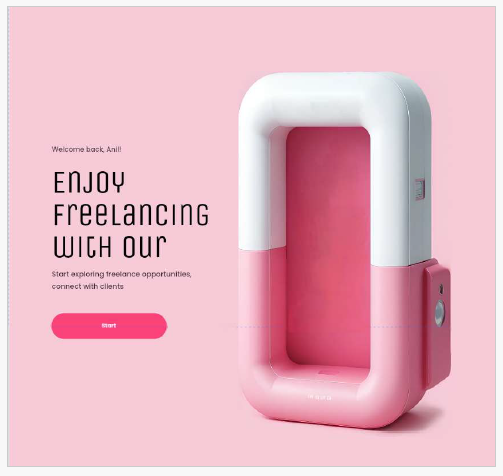
**SCREENSHOTS**

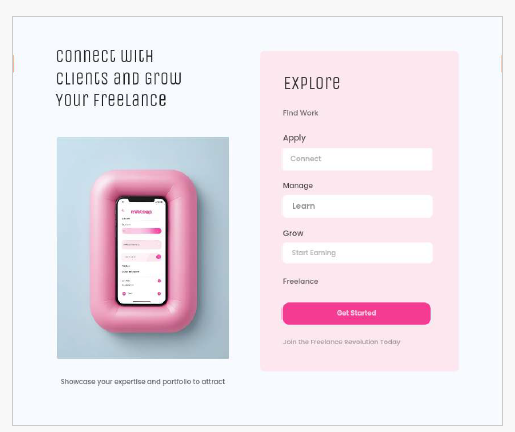
****

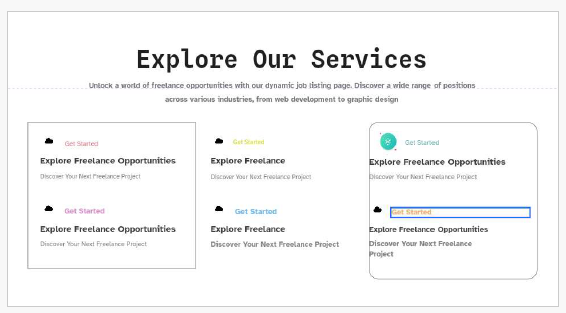
****

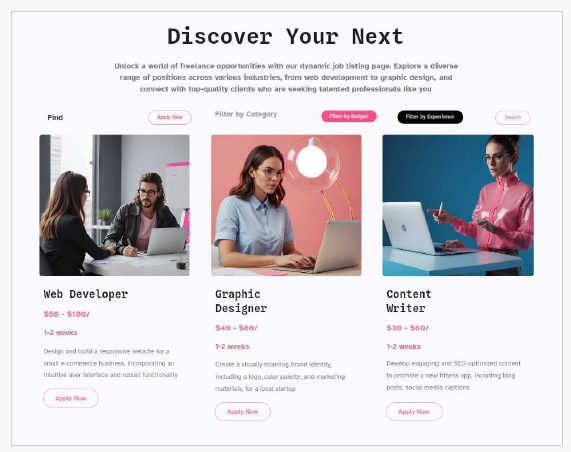


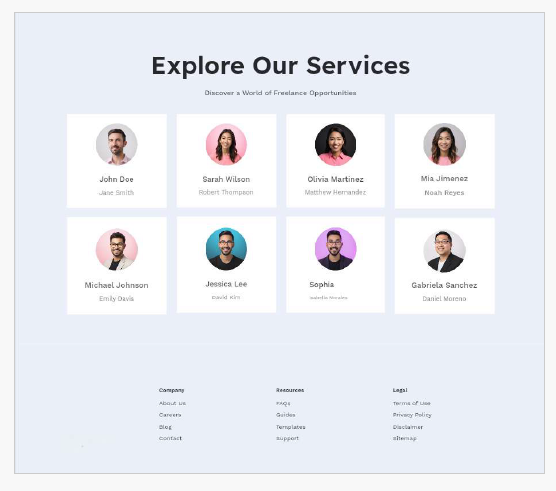












**KNOWN ISSUES**

### **1. User Registration and Login**

* **Issue:** Password Reset Email Delay
  + **Description:** Users sometimes experience a delay in receiving password reset emails.
  + **Workaround:** Suggest users check their spam folders or retry after 10 minutes.
  + **Fix Status:** Under investigation.
* **Issue:** Social Login Errors
  + **Description:** Login via Google or Facebook occasionally fails due to expired API tokens.
  + **Workaround:** Use manual login with email and password.
  + **Fix Status:** Scheduled for next release.

### **2. Profile Management**

* **Issue:** Profile Image Upload Fails
  + **Description:** Certain image formats (e.g., WebP) are not supported.
  + **Workaround:** Use standard formats like PNG or JPEG.
  + **Fix Status:** Planned update to add support for WebP.
* **Issue:** Incomplete Profile Percentage Bug
  + **Description:** The profile completion percentage does not update after adding a portfolio item.
  + **Workaround:** Refresh the page to see the correct percentage.
  + **Fix Status:** Fix being tested.

### **3. Job Posting and Bidding**

* **Issue:** Job Posting Confirmation Delay
  + **Description:** Newly posted jobs sometimes take up to 5 minutes to appear due to server-side caching.
  + **Workaround:** Wait and refresh the page.
  + **Fix Status:** Optimizing caching system in progress.
* **Issue:** Proposal Submission Timeout
  + **Description:** Submitting proposals with large file attachments occasionally times out.
  + **Workaround:** Reduce file size or compress attachments before submission.
  + **Fix Status:** Under active resolution.

### **4. Payment and Billing**

* **Issue:** Currency Conversion Errors
  + **Description:** Incorrect conversion rates displayed in some cases for non-USD transactions.
  + **Workaround:** Cross-verify rates with external tools before confirming payments.
  + **Fix Status:** High priority for the next patch.
* **Issue:** Payment Gateway Downtime
  + **Description:** Intermittent outages with specific gateways (e.g., PayPal).
  + **Workaround:** Retry the transaction later or use an alternative gateway.
  + **Fix Status:** Monitoring and coordination with gateway providers.

### **5. Communication and Messaging**

* **Issue:** Delayed Notifications
  + **Description:** Real-time notifications for messages or job updates are occasionally delayed.
  + **Workaround:** Refresh the dashboard to view updates manually.
  + **Fix Status:** Fix scheduled for next deployment.
* **Issue:** Missing Message Threads
  + **Description:** Rare cases of message threads not displaying after an app update.
  + **Workaround:** Access messages via email copies.
  + **Fix Status:** Investigating database sync issue.

### **6. Mobile Responsiveness**

* **Issue:** Dropdown Menus Misaligned
  + **Description:** Dropdown menus on mobile devices sometimes appear off-screen.
  + **Workaround:** Use landscape mode or desktop view on mobile browsers.
  + **Fix Status:** UI redesign planned.
* **Issue:** Page Freezes on Older Devices
  + **Description:** Pages with high-resolution images or charts occasionally freeze on older mobile devices.
  + **Workaround:** Switch to a desktop or reduce browser load (e.g., close other tabs).
  + **Fix Status:** Optimizing asset loading.

### **7. General Performance**

* **Issue:** Slow Page Load During Peak Hours
  + **Description:** Increased traffic causes slower response times for certain pages (e.g., job listings).
  + **Workaround:** Retry access during off-peak hours.
  + **Fix Status:** Server scaling strategy under development.
* **Issue:** Search Filters Reset Unexpectedly
  + **Description:** Applying multiple filters resets previously selected criteria.
  + **Workaround:** Apply filters one at a time and search immediately.
  + **Fix Status:** Patch in progress.

### **8. Security**

* **Issue:** Session Timeout Not Clear
  + **Description:** Users are logged out without warning due to inactivity but are unaware of session expiration.
  + **Workaround:** Refresh the page or log in again.
  + **Fix Status:** Adding session expiration warnings.
* **Issue:** Password Autofill Compatibility
  + **Description:** Some browsers do not autofill login credentials due to conflicts with form attributes.
  + **Workaround:** Manually enter login details.
  + **Fix Status:** Identified; awaiting browser-side updates

### **9.Notification System**

* **Issue:** Duplicate Notifications
  + **Description:** Users receive multiple identical notifications for the same event, such as a job invitation or proposal acceptance.
  + **Workaround:** Ignore duplicates and check the event directly on the dashboard.
  + **Fix Status:** Investigating backend event triggers.
* **Issue:** Notification Preference Reset
  + **Description:** Notification preferences sometimes reset to default after logging out and back in.
  + **Workaround:** Reconfigure preferences manually.
  + **Fix Status:** Scheduled for next sprint.

### **10.Search and Filtering**

* **Issue:** Irrelevant Search Results
  + **Description:** Job or freelancer search sometimes shows irrelevant results due to poorly indexed keywords.
  + **Workaround:** Refine search terms and use advanced filters.
  + **Fix Status:** Ongoing improvement of the search algorithm.
* **Issue:** Pagination Issues
  + **Description:** Navigating between pages in search results occasionally skips entries or loops back to the first page.
  + **Workaround:** Manually adjust the URL query string to access the desired page.
  + **Fix Status:** High-priority fix in progress.

### **11.Job Management**

* **Issue:** Job Edit Lockout
  + **Description:** Clients cannot edit job postings after receiving a proposal, even for minor changes like correcting typos.
  + **Workaround:** Add clarifications in the project description via comments.
  + **Fix Status:** Discussion ongoing for introducing an "Edit with History" feature.
* **Issue:** Expired Jobs Still Visible
  + **Description:** Expired or filled jobs sometimes continue to appear in search results.
  + **Workaround:** Ignore jobs marked as "Expired" or "Closed."
  + **Fix Status:** Optimizing job expiration logic.

**FUTURE ENHANCEMENTS**

### **1.Advanced Search and Matching Algorithms**

* **Feature:** AI-Powered Job Matching
  + Use machine learning to recommend jobs or freelancers based on past activity, skills, and preferences.
  + Benefit: Saves time for both freelancers and clients by providing highly relevant matches.
* **Feature:** Enhanced Filter Options
  + Add filters like client budget history, freelancer completion rates, or project duration.
  + Benefit: Helps users find exactly what they’re looking for.

### **2. Integrated Learning and Skill Development**

* **Feature:** E-Learning Integration
  + Provide online courses, certifications, or workshops directly on the platform.
  + Benefit: Empowers freelancers to upskill and stay competitive.
* **Feature:** Skill Assessment Tests
  + Allow freelancers to take tests to verify skills and display badges on their profiles.
  + Benefit: Builds client confidence in freelancer expertise.

### **3. Improved Communication Tools**

* **Feature:** Video and Voice Chat Integration
  + Enable in-platform video/voice calls for real-time communication.
  + Benefit: Eliminates the need for external tools, enhancing privacy and security.
* **Feature:** Scheduled Messaging
  + Allow users to schedule messages for specific times (e.g., proposals or project updates).
  + Benefit: Improves communication flexibility across different time zones.

### **4. Payment System Enhancements**

* **Feature:** Milestone-Based Payments
  + Enable clients to set milestones for incremental payments upon task completion.
  + Benefit: Increases transparency and reduces disputes.
* **Feature:** Multi-Currency Support
  + Allow users to view and transact in their preferred currency with live conversion rates.
  + Benefit: Simplifies international transactions.
* **Feature:** Cryptocurrency Payments
  + Introduce secure cryptocurrency payment options (e.g., Bitcoin, Ethereum).
  + Benefit: Broadens the platform’s appeal to tech-savvy users.

### **5. Mobile App Optimization**

* **Feature:** Offline Mode
  + Allow users to draft proposals, update profiles, or view projects offline and sync when online.
  + Benefit: Enhances usability for users in areas with limited connectivity.
* **Feature:** Push Notifications
  + Provide real-time updates for job invitations, message alerts, or proposal responses.
  + Benefit: Keeps users engaged and informed.

### **6. Enhanced Analytics and Insights**

* **Feature:** Advanced Analytics for Freelancers
  + Provide insights into profile views, proposal success rates, and most-requested skills.
  + Benefit: Helps freelancers optimize their profiles and strategies.
* **Feature:** Job Performance Analytics for Clients
  + Offer analytics on job postings, such as number of views, proposal quality, and budget benchmarks.
  + Benefit: Helps clients optimize job postings for better results.

### **7. Community and Networking Features**

* **Feature:** Freelancer Community Forums
  + Create a space for freelancers to share tips, ask questions, and connect with others.
  + Benefit: Builds a supportive user community.
* **Feature:** Client-Freelancer Match Events
  + Host virtual networking events or speed pitches to connect clients and freelancers.
  + Benefit: Provides a unique way to build relationships and find work.

### **8. Enhanced Security and Verification**

* **Feature:** Biometric Login
  + Enable login using fingerprints or facial recognition on supported devices.
  + Benefit: Enhances security and ease of access.
* **Feature:** Advanced Verification Badges
  + Introduce additional verification options like video verification or government ID upload.
  + Benefit: Improves trustworthiness of profiles.

### **9. Customizable User Experience**

* **Feature:** Dashboard Customization
  + Allow users to customize their dashboards with widgets like quick access to jobs, earnings, or messages.
  + Benefit: Provides a personalized experience.
* **Feature:** Dark Mode
  + Offer a dark mode option for reduced eye strain and modern aesthetics.
  + Benefit: Enhances usability for users who work late hours.

### **10. Dispute Resolution Enhancements**

* **Feature:** AI-Based Dispute Assistance
  + Use AI to review and suggest resolutions for disputes between clients and freelancers.
  + Benefit: Speeds up the resolution process and ensures fairness.
* **Feature:** Escrow Enhancements
  + Improve escrow transparency, allowing both parties to track funds throughout the project.
  + Benefit: Reduces financial disputes.

### **11. Marketing and Promotion Features**

* **Feature:** Profile Boosting
  + Allow freelancers to pay for temporary profile boosting in search results.
  + Benefit: Offers a monetization opportunity for the platform and increased visibility for users.
* **Feature:** Job Ad Sponsorships
  + Enable clients to sponsor job listings for better visibility among freelancers.
  + Benefit: Helps clients attract top talent quickly.

### **12. Localization and Accessibility**

* **Feature:** Multi-Language Support
  + Offer full platform translations into various languages.
  + Benefit: Expands the platform's reach globally.
* **Feature:** Accessibility Features
  + Add tools for visually or hearing-impaired users, such as screen reader compatibility or closed captioning.
  + Benefit: Ensures inclusivity.