Full Stack Development with MERN

AJ Works – Freelancing Platform

1. Introduction

- **Project Title:** AJ Works – Freelancing Platform

  AJ Works is a dynamic full-stack freelancing marketplace built using the MERN stack. It connects freelancers with clients across various industries, including creative design, development, content writing, and technical services. The platform focuses on simplifying project posting, bidding, communication, and delivery through an intuitive, user-friendly interface.

  The platform also supports real-time notifications, chat messaging, and a file-submission workflow, making collaboration seamless. AJ Works demonstrates practical implementation of complex multi-role systems, scalable backend design, and enterprise-level UI development.

2. Project Overview

- **Purpose:** The purpose of AJ Works is to create a digital ecosystem where clients and freelancers can collaborate professionally without external tools. It aims to provide a secure, efficient, and transparent freelancing environment inspired by platforms like Upwork and Fiverr.

3. Architecture

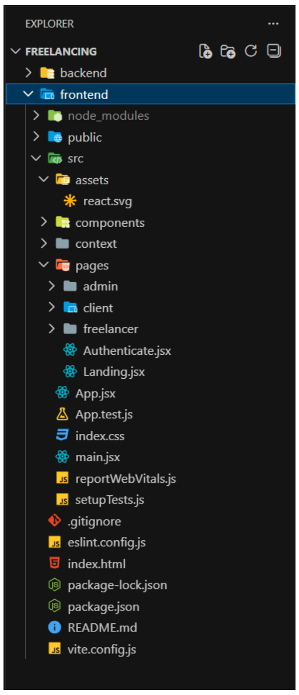
- **Frontend (React.js):** AJ Works uses React.js to build a modular, responsive, and interactive interface. Components include dashboards, project cards, proposal forms, chat UI, and notification panels. State is managed using Redux Toolkit or Context API.
- **Backend (Node.js + Express.js):** The backend follows a microservice-like modular architecture with separate controllers, middleware, and routing. Express.js handles REST API routing for authentication, projects, bids, chats, reviews, and admin operations.

4. Setup Instructions

- **Prerequisites:**
  - Node.js v16+
  - npm / yarn
  - MongoDB Atlas or local MongoDB
  - Postman
  - VS Code
  - Git
- **Installation:**
  - cd frontend
  - npm install
  - cd ../backend
  - npm install

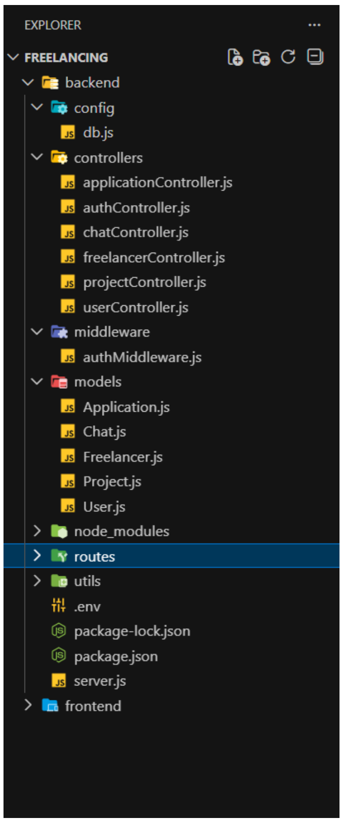
5. Folder Structure

- **Client:** React frontend.



```
EXPLORER ...
FREELANCING
  backend
  frontend
    node_modules
    public
    src
      assets
        react.svg
      components
      context
      pages
        admin
        client
        freelancer
        Authenticate.jsx
        Landing.jsx
      App.jsx
      App.test.js
      index.css
      main.jsx
      reportWebVitals.js
      setupTests.js
    .gitignore
    eslint.config.js
    index.html
    package-lock.json
    package.json
    README.md
    vite.config.js
```

- **Server:** Node.js backend.



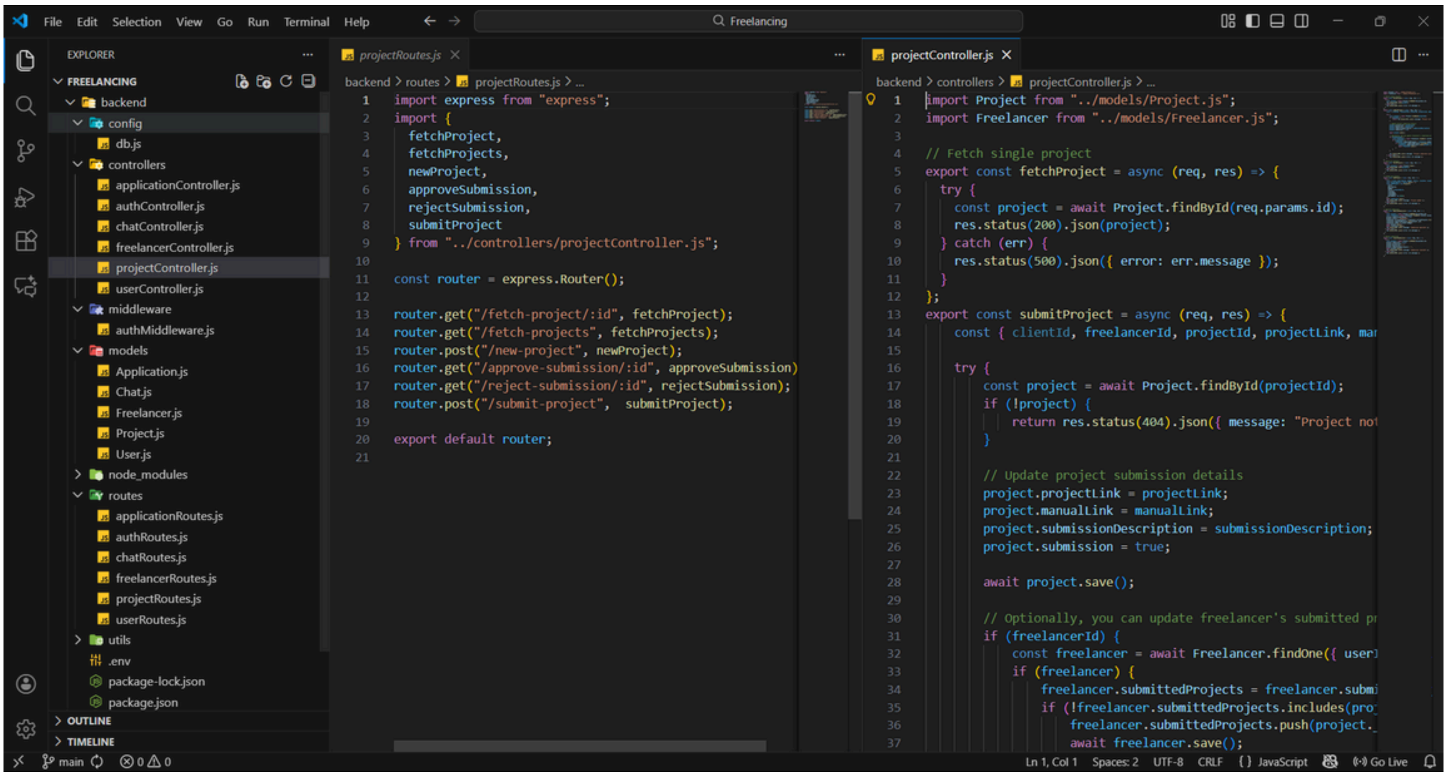
```
EXPLORER ...
FREELANCING
  backend
    config
      db.js
    controllers
      applicationController.js
      authController.js
      chatController.js
      freelancerController.js
      projectController.js
      userController.js
    middleware
      authMiddleware.js
    models
      Application.js
      Chat.js
      Freelancer.js
      Project.js
      User.js
    node_modules
    routes
    utils
    .env
    package-lock.json
    package.json
    server.js
  frontend
```

6. Running the Application

- **Frontend:** `npm start` in the client directory.
  - cd frontend
  - npm start
- **Backend:** `npm start` in the server directory.
  - cd frontend

- npm start

7. API Documentation

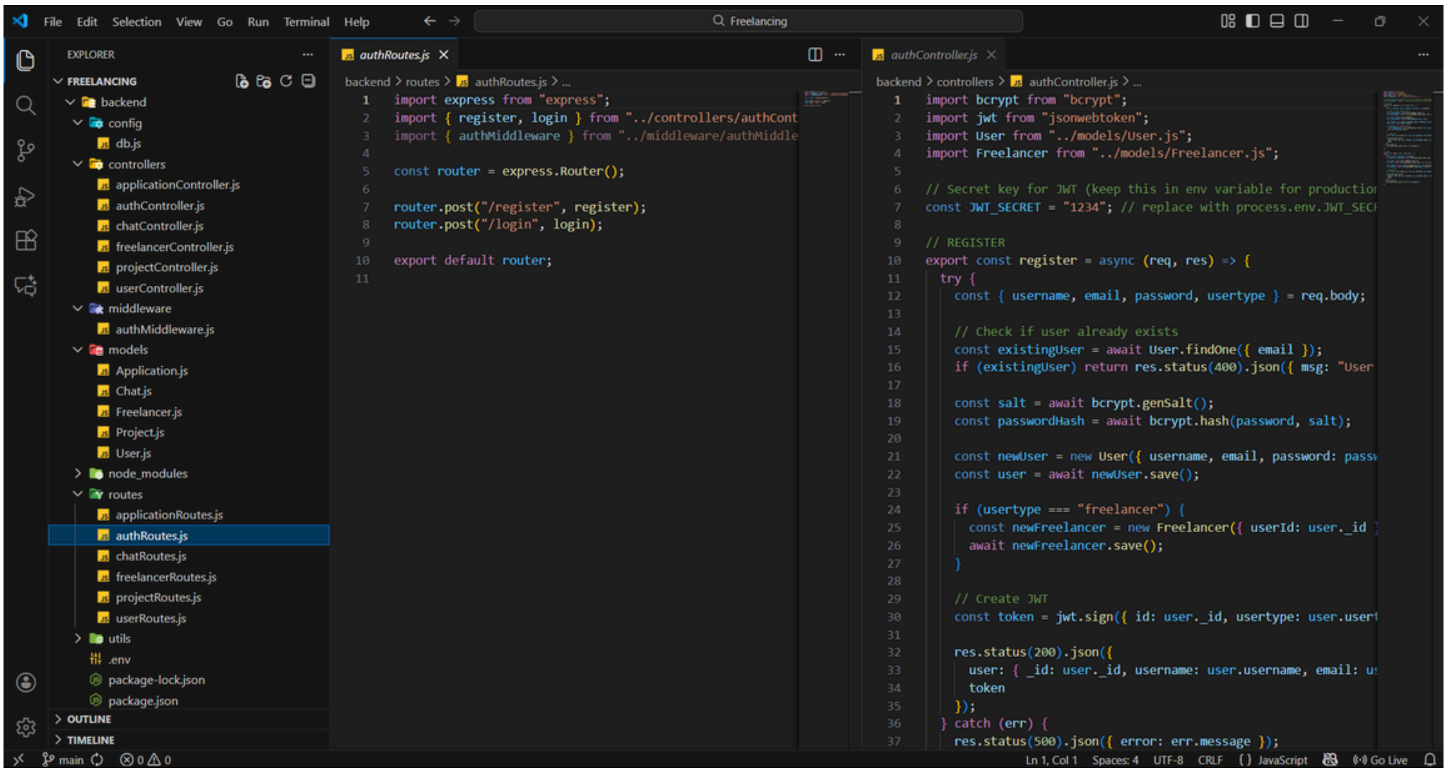


backend > routes > projectRoutes.js > ...

```javascript
import express from "express";
import {
  fetchProject,
  fetchProjects,
  newProject,
  approveSubmission,
  rejectSubmission,
  submitProject
} from "../controllers/projectController.js";

const router = express.Router();

router.get("/fetch-project/:id", fetchProject);
router.get("/fetch-projects", fetchProjects);
router.post("/new-project", newProject);
router.get("/approve-submission/:id", approveSubmission)
router.get("/reject-submission/:id", rejectSubmission);
router.post("/submit-project",  submitProject);

export default router;
```

backend > controllers > projectController.js > ...

```javascript
import Project from "../models/Project.js";
import Freelancer from "../models/Freelancer.js";

// Fetch single project
export const fetchProject = async (req, res) => {
  try {
    const project = await Project.findById(req.params.id);
    res.status(200).json(project);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
export const submitProject = async (req, res) => {
    const { clientId, freelancerId, projectId, projectLink, mar

    try {
        const project = await Project.findById(projectId);
        if (!project) {
            return res.status(404).json({ message: "Project not
        }

        // Update project submission details
        project.projectLink = projectLink;
        project.manualLink = manualLink;
        project.submissionDescription = submissionDescription;
        project.submission = true;

        await project.save();

        // Optionally, you can update freelancer's submitted pr
        if (freelancerId) {
            const freelancer = await Freelancer.findOne({ userI
            if (freelancer) {
                freelancer.submittedProjects = freelancer.submi
                if (!freelancer.submittedProjects.includes(pro
                    freelancer.submittedProjects.push(project._
                    await freelancer.save();
```

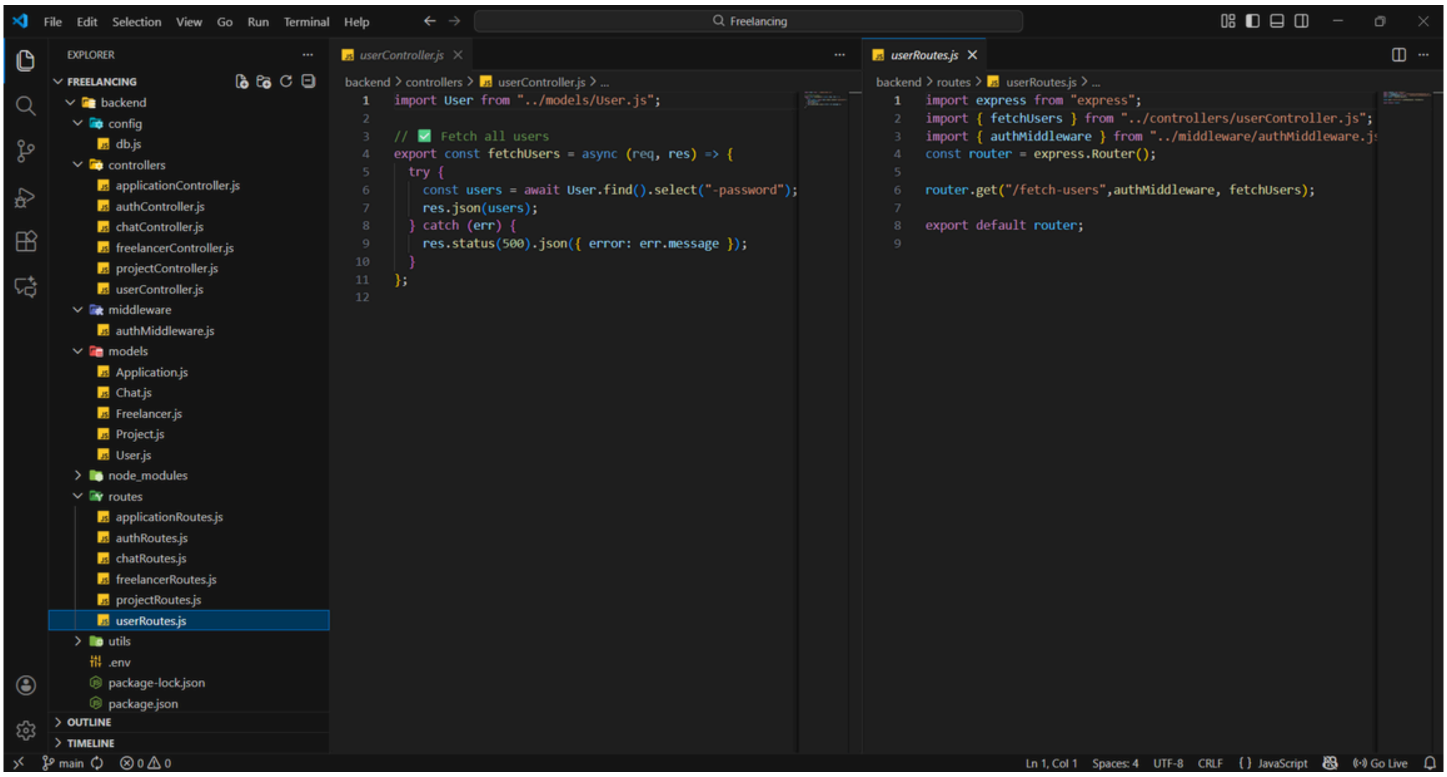


backend > routes > authRoutes.js > ...

```javascript
import express from "express";
import { register, login } from "../controllers/authCont
import { authMiddleware } from "../middleware/authMiddle

const router = express.Router();

router.post("/register", register);
router.post("/login", login);

export default router;
```

backend > controllers > authController.js > ...

```javascript
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import User from "../models/User.js";
import Freelancer from "../models/Freelancer.js";

// Secret key for JWT (keep this in env variable for productior
const JWT_SECRET = "1234"; // replace with process.env.JWT_SECR

// REGISTER
export const register = async (req, res) => {
  try {
    const { username, email, password, usertype } = req.body;

    // Check if user already exists
    const existingUser = await User.findOne({ email });
    if (existingUser) return res.status(400).json({ msg: "User

    const salt = await bcrypt.genSalt();
    const passwordHash = await bcrypt.hash(password, salt);

    const newUser = new User({ username, email, password: passw
    const user = await newUser.save();

    if (usertype === "freelancer") {
      const newFreelancer = new Freelancer({ userId: user._id
      await newFreelancer.save();
    }

    // Create JWT
    const token = jwt.sign({ id: user._id, usertype: user.usert

    res.status(200).json({
      user: { _id: user._id, username: user.username, email: us
      token
    });
  } catch (err) {
    res.status(500).json({ error: err.message });
```

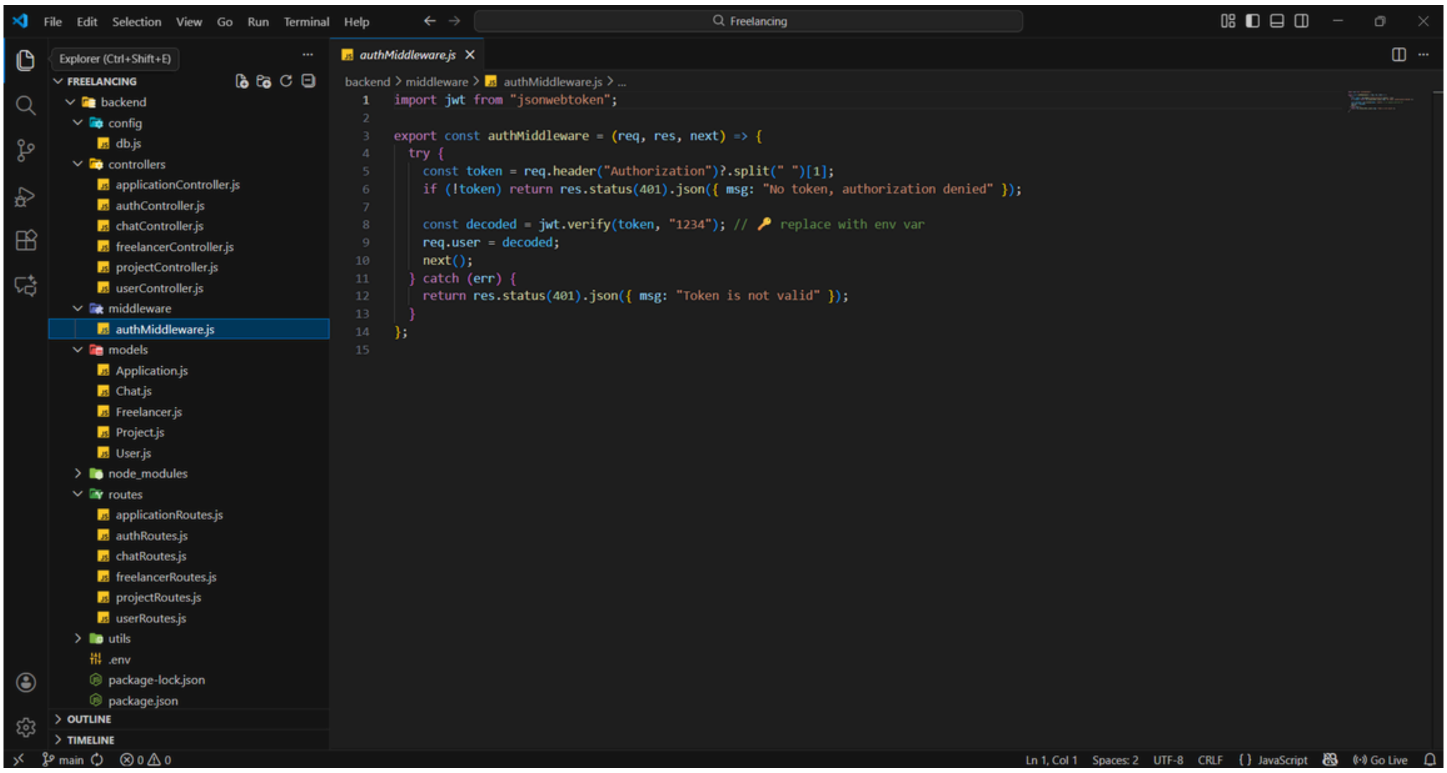

userController.js
backend > controllers > userController.js > ...
userRoutes.js
backend > routes > userRoutes.js > ...
FREELANCING

```javascript
import User from "../models/User.js";

// ✅ Fetch all users
export const fetchUsers = async (req, res) => {
  try {
    const users = await User.find().select("-password");
    res.json(users);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

```javascript
import express from "express";
import { fetchUsers } from "../controllers/userController.js";
import { authMiddleware } from "../middleware/authMiddleware.j
const router = express.Router();

router.get("/fetch-users",authMiddleware, fetchUsers);

export default router;
```

8. Authentication

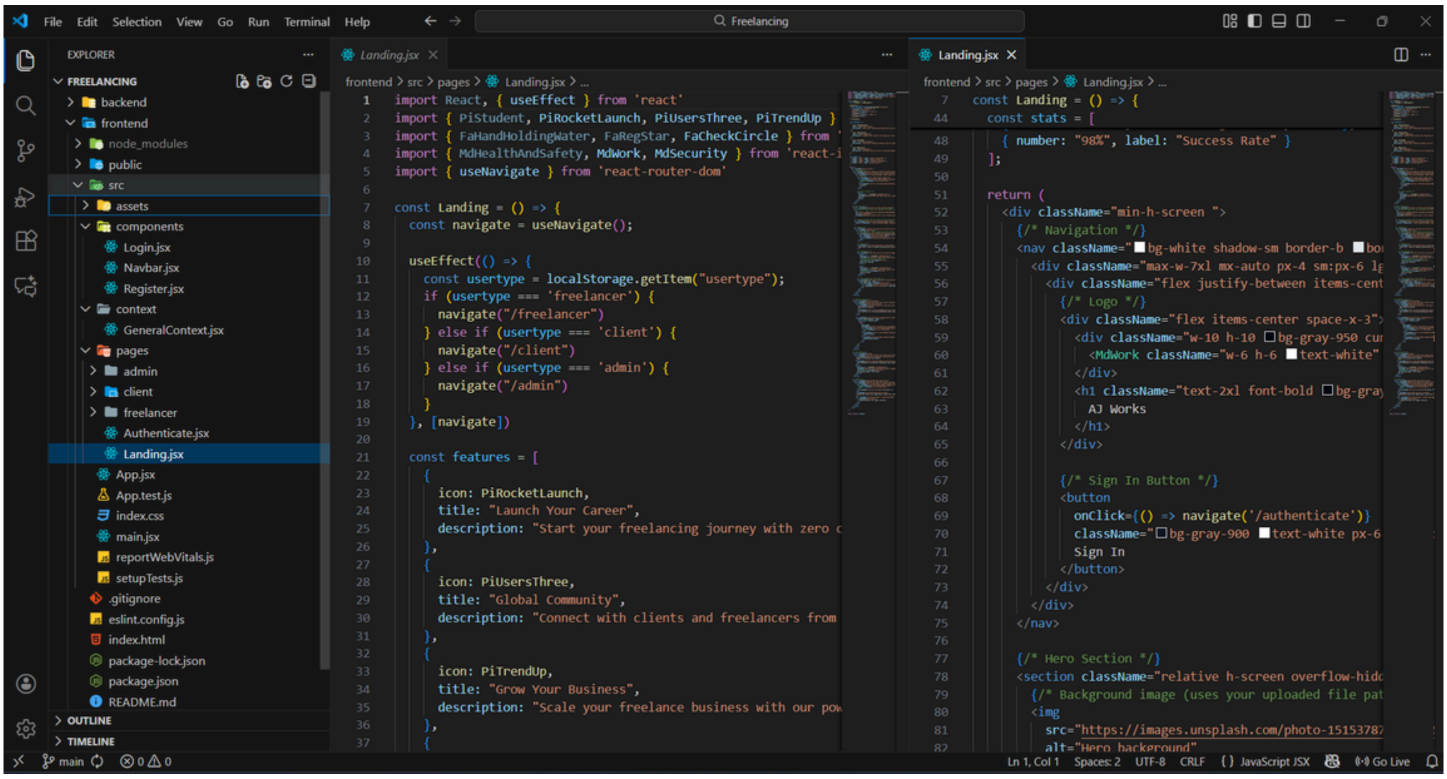
Authentication uses JWT tokens stored in secure cookies. Passwords are hashed using bcrypt. Middleware ensures each route allows access only to the correct role (Client / Freelancer / Admin). Admin routes are strictly protected, and suspicious activities are logged. Email verification can be optionally added.



```javascript
import jwt from "jsonwebtoken";

export const authMiddleware = (req, res, next) => {
  try {
    const token = req.header("Authorization")?.split(" ")[1];
    if (!token) return res.status(401).json({ msg: "No token, authorization denied" });

    const decoded = jwt.verify(token, "1234"); // 🔑 replace with env var
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ msg: "Token is not valid" });
  }
};
```

9. User Interface

The UI includes a clean dashboard for clients and freelancers, project detail pages, bidding UI, chat windows, and file upload sections. React components are styled using Bootstrap or Material UI. UI consistency is maintained using reusable components and global styles. Mobile responsiveness ensures access on any device.

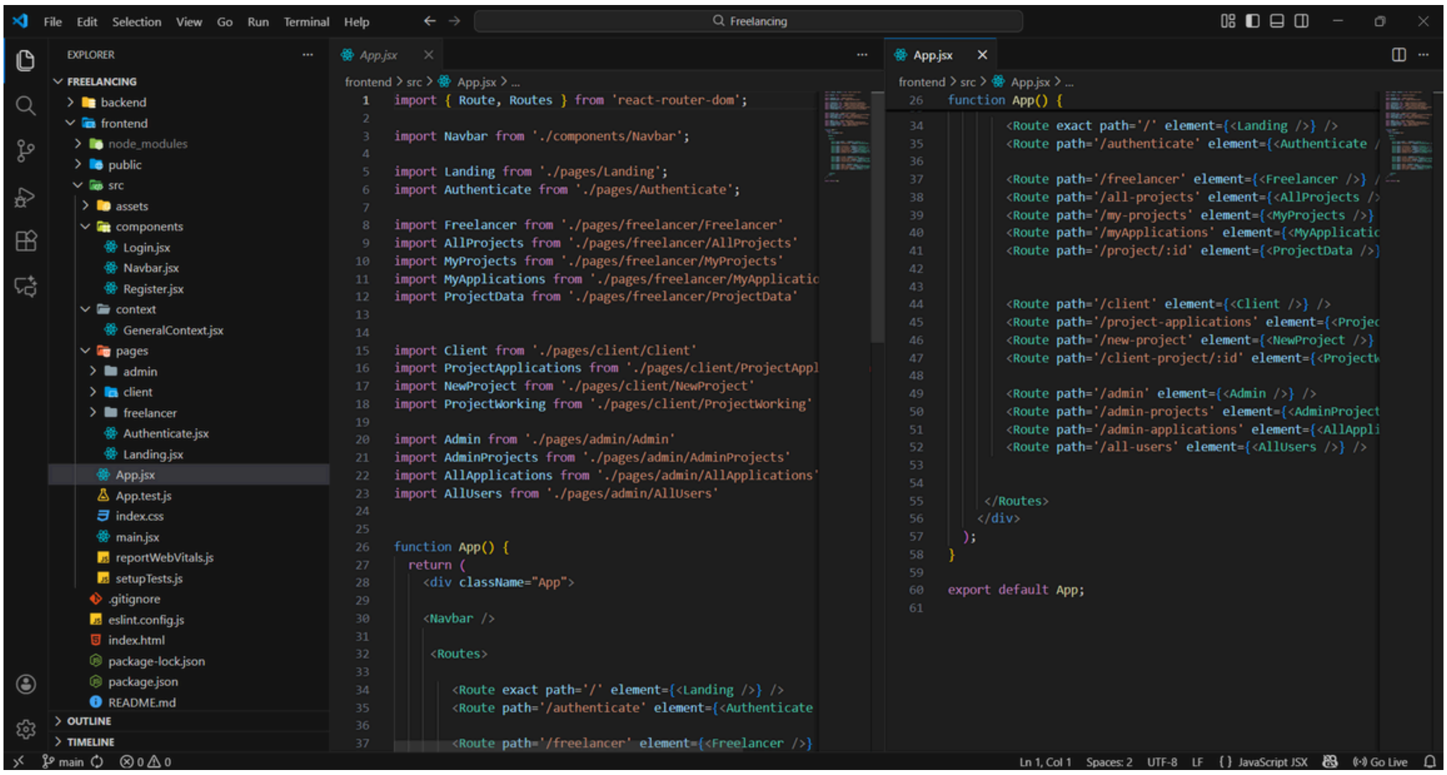


frontend > src > pages > Landing.jsx > ...

```jsx
import React, { useEffect } from 'react'
import { PiStudent, PiRocketLaunch, PiUsersThree, PiTrendUp }
import { FaHandHoldingWater, FaRegStar, FaCheckCircle } from '
import { MdHealthAndSafety, MdWork, MdSecurity } from 'react-i
import { useNavigate } from 'react-router-dom'

const Landing = () => {
  const navigate = useNavigate();

  useEffect(() => {
    const usertype = localStorage.getItem("usertype");
    if (usertype === 'freelancer') {
      navigate("/freelancer")
    } else if (usertype === 'client') {
      navigate("/client")
    } else if (usertype === 'admin') {
      navigate("/admin")
    }
  }, [navigate])

  const features = [
    {
      icon: PiRocketLaunch,
      title: "Launch Your Career",
      description: "Start your freelancing journey with zero c
    },
    {
      icon: PiUsersThree,
      title: "Global Community",
      description: "Connect with clients and freelancers from
    },
    {
      icon: PiTrendUp,
      title: "Grow Your Business",
      description: "Scale your freelance business with our pow
    },
    {
```

frontend > src > pages > Landing.jsx > ...

```jsx
const Landing = () => {
  const stats = [
      { number: "98%", label: "Success Rate" }
  ];

  return (
    <div className="min-h-screen ">
      {/* Navigation */}
      <nav className="bg-white shadow-sm border-b bor
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg
          <div className="flex justify-between items-cent
            {/* Logo */}
            <div className="flex items-center space-x-3">
              <div className="w-10 h-10 bg-gray-950 cur
                <MdWork className="w-6 h-6 text-white"
              </div>
              <h1 className="text-2xl font-bold bg-gray
                AJ Works
              </h1>
            </div>

            {/* Sign In Button */}
            <button
              onClick={() => navigate('/authenticate')}
              className="bg-gray-900 text-white px-6
              Sign In
            </button>
          </div>
        </div>
      </nav>

      {/* Hero Section */}
      <section className="relative h-screen overflow-hidc
        {/* Background image (uses your uploaded file pat
        <img
          src="https://images.unsplash.com/photo-15153787
          alt="Hero background"
```



frontend > src > App.jsx > ...

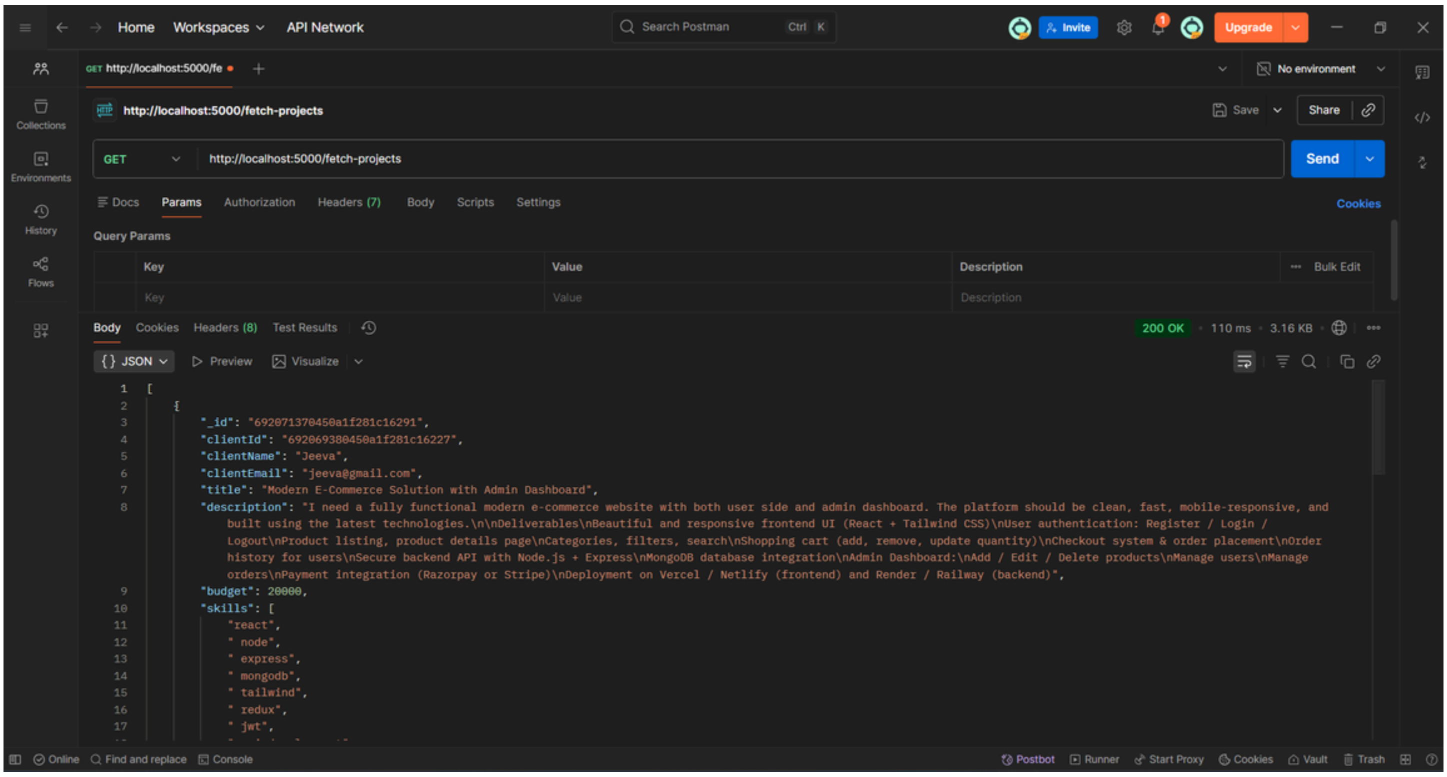
```jsx
import { Route, Routes } from 'react-router-dom';

import Navbar from './components/Navbar';

import Landing from './pages/Landing';
import Authenticate from './pages/Authenticate';

import Freelancer from './pages/freelancer/Freelancer'
import AllProjects from './pages/freelancer/AllProjects'
import MyProjects from './pages/freelancer/MyProjects'
import MyApplications from './pages/freelancer/MyApplicatic
import ProjectData from './pages/freelancer/ProjectData'


import Client from './pages/client/Client'
import ProjectApplications from './pages/client/ProjectAppl
import NewProject from './pages/client/NewProject'
import ProjectWorking from './pages/client/ProjectWorking'

import Admin from './pages/admin/Admin'
import AdminProjects from './pages/admin/AdminProjects'
import AllApplications from './pages/admin/AllApplications'
import AllUsers from './pages/admin/AllUsers'


function App() {
  return (
    <div className="App">

    <Navbar />

     <Routes>

        <Route exact path='/' element={<Landing />} />
        <Route path='/authenticate' element={<Authenticate

        <Route path='/freelancer' element={<Freelancer />}
```

frontend > src > App.jsx > ...

```jsx
function App() {
        <Route exact path='/' element={<Landing />} />
        <Route path='/authenticate' element={<Authenticate /

        <Route path='/freelancer' element={<Freelancer />} /
        <Route path='/all-projects' element={<AllProjects />
        <Route path='/my-projects' element={<MyProjects />}
        <Route path='/myApplications' element={<MyApplicatic
        <Route path='/project/:id' element={<ProjectData />}


        <Route path='/client' element={<Client />} />
        <Route path='/project-applications' element={<Projec
        <Route path='/new-project' element={<NewProject />}
        <Route path='/client-project/:id' element={<ProjectW

        <Route path='/admin' element={<Admin />} />
        <Route path='/admin-projects' element={<AdminProject
        <Route path='/admin-applications' element={<AllAppli
        <Route path='/all-users' element={<AllUsers />} />


      </Routes>
    </div>
  );
}

export default App;
```

10. Testing

Testing includes manual testing with Postman for backend API endpoints like login, project creation, and bid submission. Frontend testing includes form validation, user flows, and dashboard navigation. Chat system testing ensures message syncing and real-time updates.


GET http://localhost:5000/fetch-projects

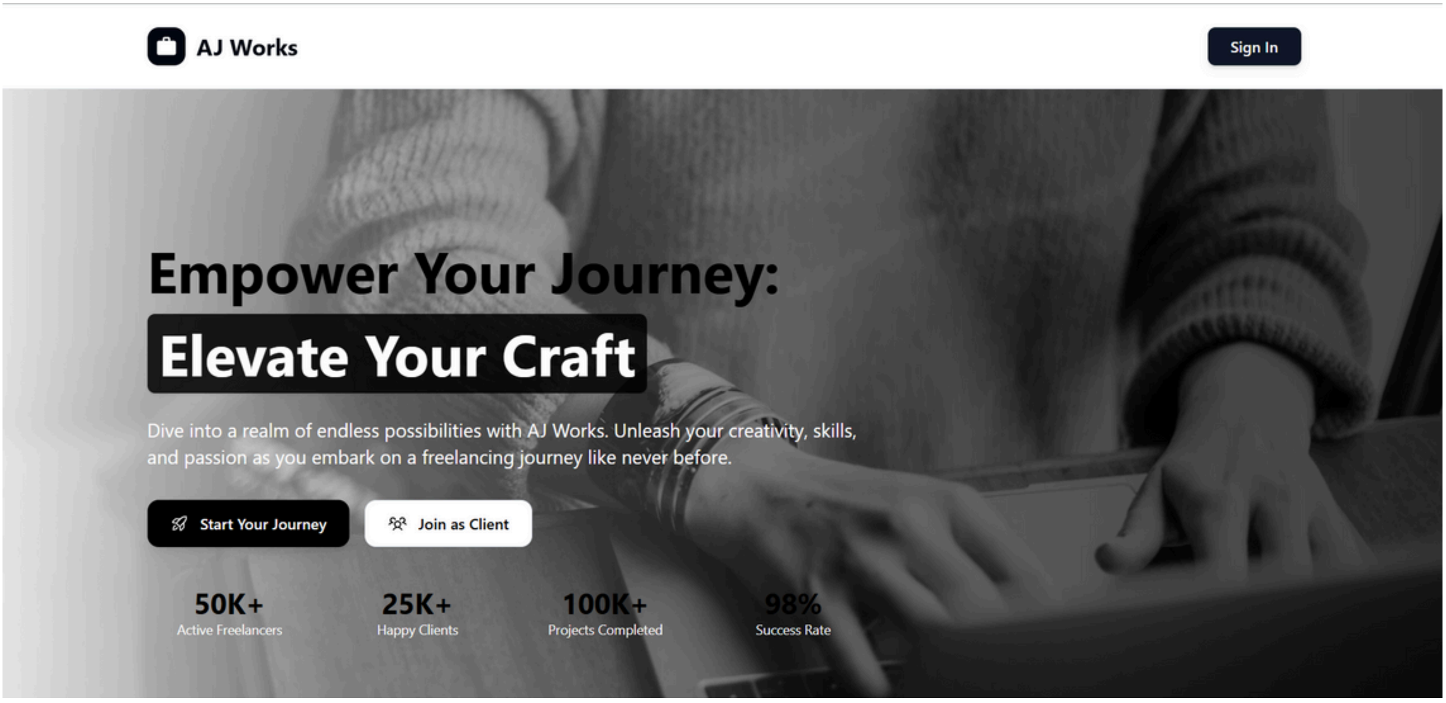
Query Params

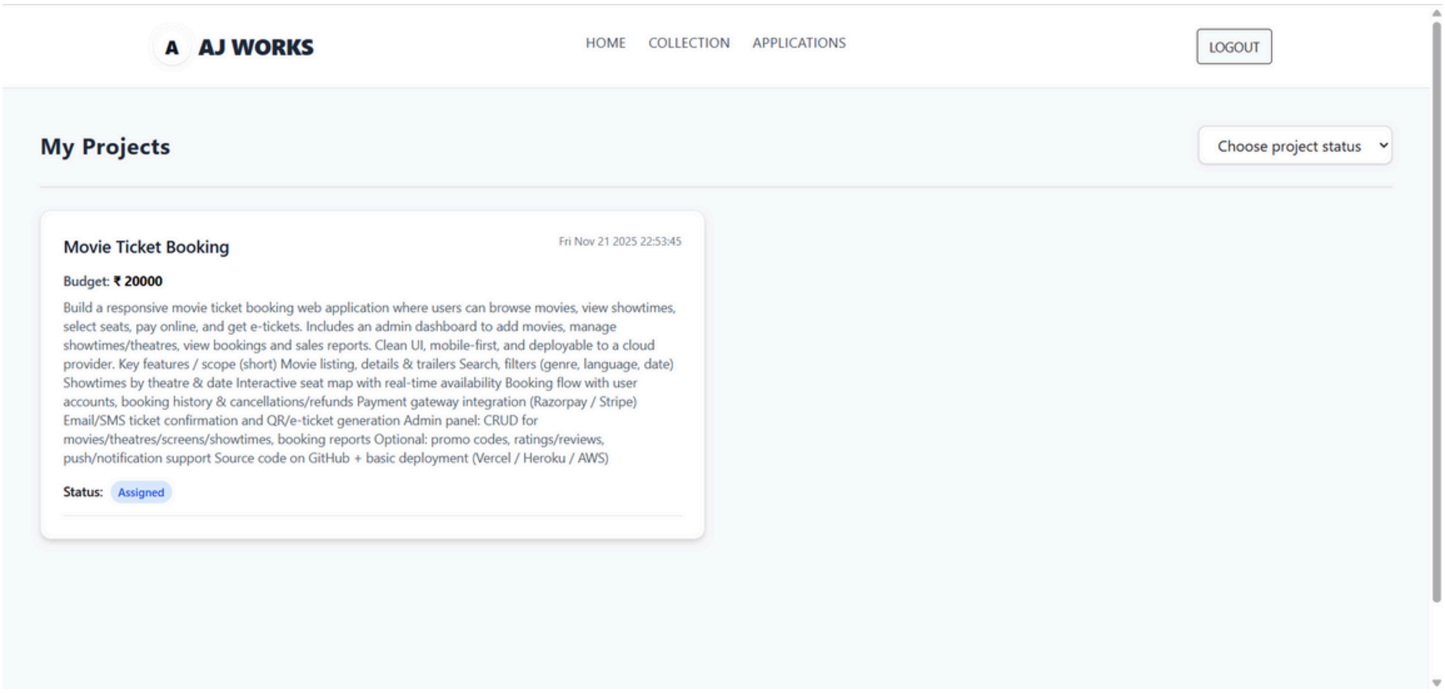
Key	Value	Description
Key	Value	Description

Body · 200 OK · 110 ms · 3.16 KB

```
[
    {
        "_id": "692071370450a1f281c16291",
        "clientId": "692069380450a1f281c16227",
        "clientName": "Jeeva",
        "clientEmail": "jeeva@gmail.com",
        "title": "Modern E-Commerce Solution with Admin Dashboard",
        "description": "I need a fully functional modern e-commerce website with both user side and admin dashboard. The platform should be clean, fast, mobile-responsive, and
            built using the latest technologies.\n\nDeliverables\nBeautiful and responsive frontend UI (React + Tailwind CSS)\nUser authentication: Register / Login /
            Logout\nProduct listing, product details page\nCategories, filters, search\nShopping cart (add, remove, update quantity)\nCheckout system & order placement\nOrder
            history for users\nSecure backend API with Node.js + Express\nMongoDB database integration\nAdmin Dashboard:\nAdd / Edit / Delete products\nManage users\nManage
            orders\nPayment integration (Razorpay or Stripe)\nDeployment on Vercel / Netlify (frontend) and Render / Railway (backend)",
        "budget": 20000,
        "skills": [
            "react",
            " node",
            " express",
            " mongodb",
            " tailwind",
            " redux",
            " jwt",
```

11. Screenshots or Demo





My Projects

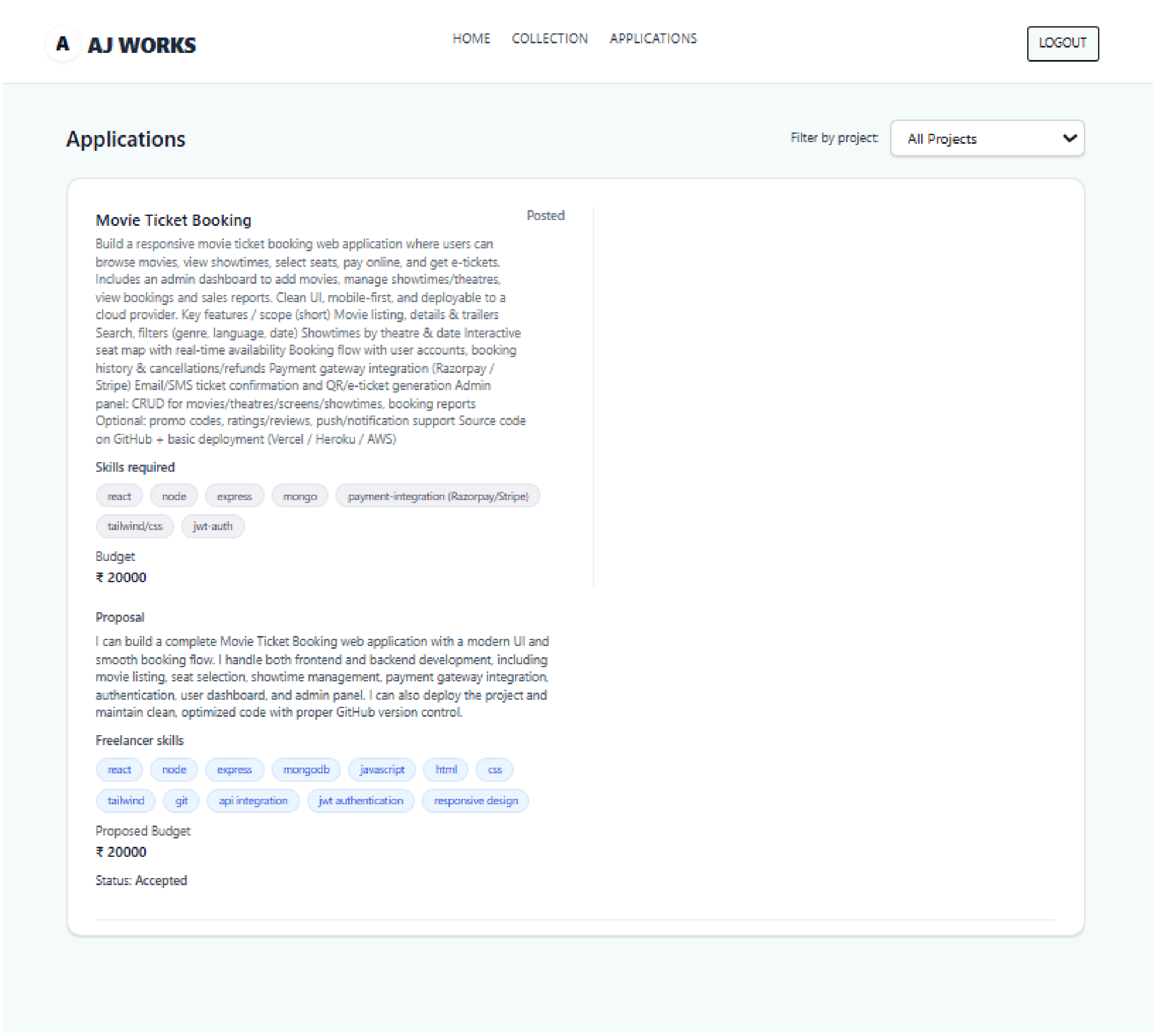
Movie Ticket Booking

Fri Nov 21 2025 22:53:45

Budget: ₹ 20000

Build a responsive movie ticket booking web application where users can browse movies, view showtimes, select seats, pay online, and get e-tickets. Includes an admin dashboard to add movies, manage showtimes/theatres, view bookings and sales reports. Clean UI, mobile-first, and deployable to a cloud provider. Key features / scope (short) Movie listing, details & trailers Search, filters (genre, language, date) Showtimes by theatre & date Interactive seat map with real-time availability Booking flow with user accounts, booking history & cancellations/refunds Payment gateway integration (Razorpay / Stripe) Email/SMS ticket confirmation and QR/e-ticket generation Admin panel: CRUD for movies/theatres/screens/showtimes, booking reports Optional: promo codes, ratings/reviews, push/notification support Source code on GitHub + basic deployment (Vercel / Heroku / AWS)

Status: Assigned



Applications

Movie Ticket Booking

Posted

Build a responsive movie ticket booking web application where users can browse movies, view showtimes, select seats, pay online, and get e-tickets. Includes an admin dashboard to add movies, manage showtimes/theatres, view bookings and sales reports. Clean UI, mobile-first, and deployable to a cloud provider. Key features / scope (short) Movie listing, details & trailers Search, filters (genre, language, date) Showtimes by theatre & date Interactive seat map with real-time availability Booking flow with user accounts, booking history & cancellations/refunds Payment gateway integration (Razorpay / Stripe) Email/SMS ticket confirmation and QR/e-ticket generation Admin panel: CRUD for movies/theatres/screens/showtimes, booking reports Optional: promo codes, ratings/reviews, push/notification support Source code on GitHub + basic deployment (Vercel / Heroku / AWS)

Skills required

react node express mongo payment-integration (Razorpay/Stripe) tailwind/css jwt-auth

Budget
₹ 20000

Proposal

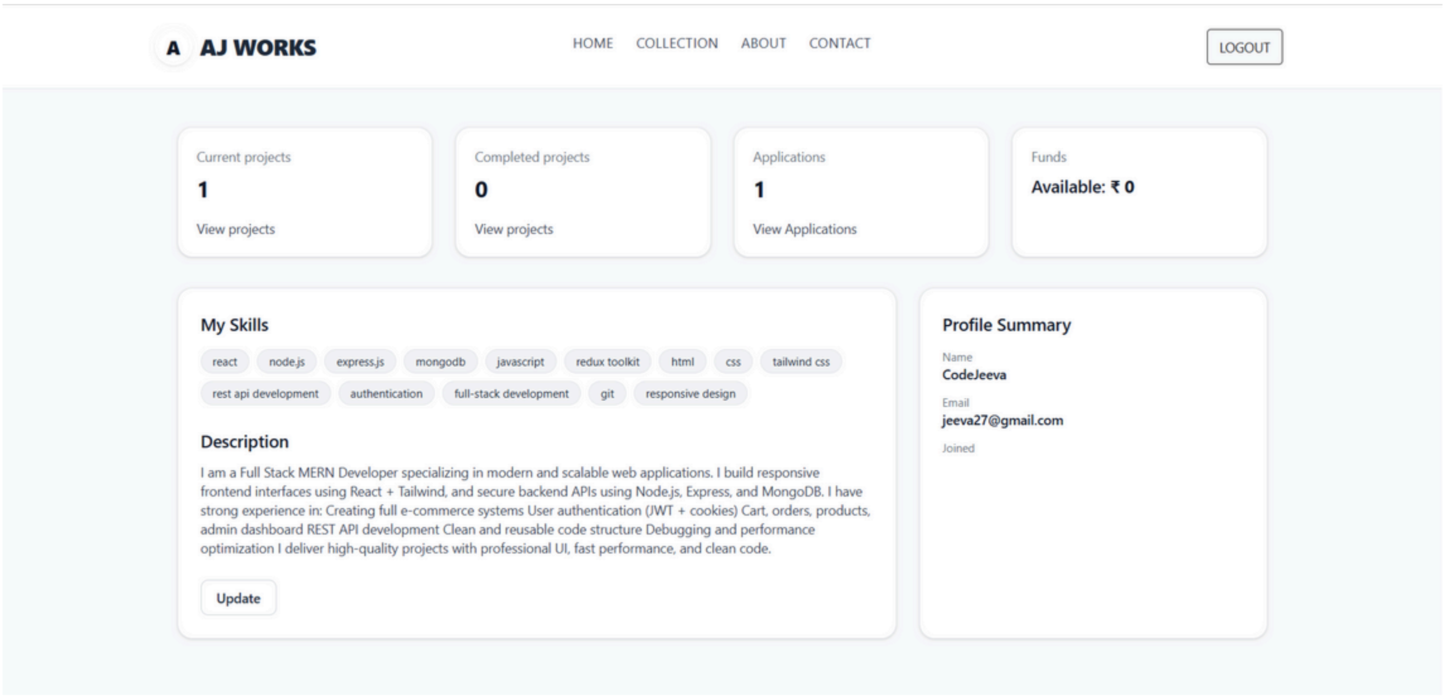
I can build a complete Movie Ticket Booking web application with a modern UI and smooth booking flow. I handle both frontend and backend development, including movie listing, seat selection, showtime management, payment gateway integration, authentication, user dashboard, and admin panel. I can also deploy the project and maintain clean, optimized code with proper GitHub version control.

Freelancer skills

react node express mongodb javascript html css tailwind git api integration jwt authentication responsive design

Proposed Budget
₹ 20000

Status: Accepted


A AJ WORKS

HOME COLLECTION ABOUT CONTACT

LOGOUT

Current projects
1
View projects

Completed projects
0
View projects

Applications
1
View Applications

Funds
Available: ₹ 0

My Skills

react · node.js · express.js · mongodb · javascript · redux toolkit · html · css · tailwind css · rest api development · authentication · full-stack development · git · responsive design

Description

I am a Full Stack MERN Developer specializing in modern and scalable web applications. I build responsive frontend interfaces using React + Tailwind, and secure backend APIs using Node.js, Express, and MongoDB. I have strong experience in: Creating full e-commerce systems User authentication (JWT + cookies) Cart, orders, products, admin dashboard REST API development Clean and reusable code structure Debugging and performance optimization I deliver high-quality projects with professional UI, fast performance, and clean code.

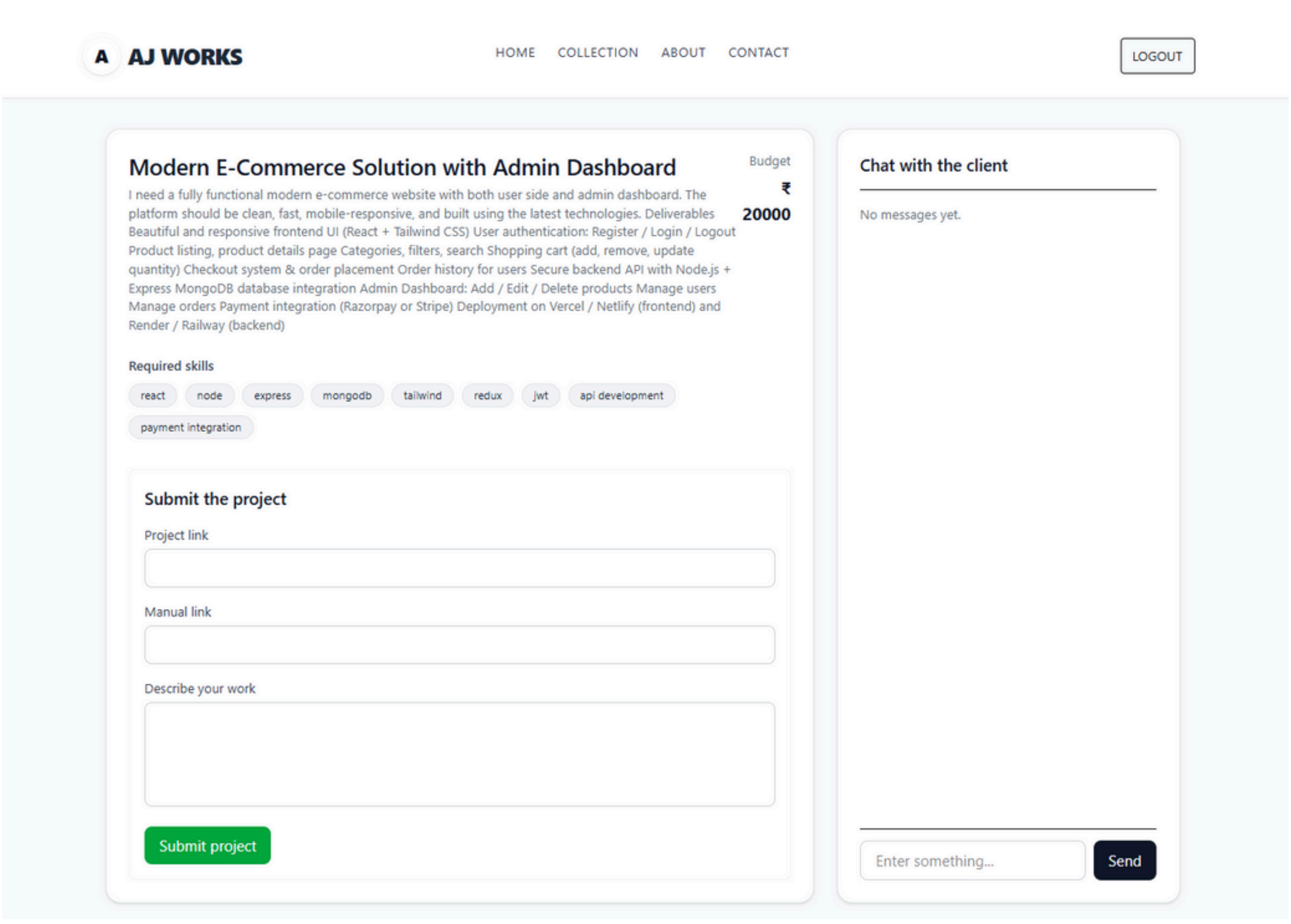
Update

Profile Summary

Name
CodeJeeva

Email
jeeva27@gmail.com

Joined


A AJ WORKS

HOME COLLECTION ABOUT CONTACT

LOGOUT

Modern E-Commerce Solution with Admin Dashboard

Budget
₹
20000

I need a fully functional modern e-commerce website with both user side and admin dashboard. The platform should be clean, fast, mobile-responsive, and built using the latest technologies. Deliverables Beautiful and responsive frontend UI (React + Tailwind CSS) User authentication: Register / Login / Logout Product listing, product details page Categories, filters, search Shopping cart (add, remove, update quantity) Checkout system & order placement Order history for users Secure backend API with Node.js + Express MongoDB database integration Admin Dashboard: Add / Edit / Delete products Manage users Manage orders Payment integration (Razorpay or Stripe) Deployment on Vercel / Netlify (frontend) and Render / Railway (backend)

Required skills

react · node · express · mongodb · tailwind · redux · jwt · api development · payment integration

Submit the project

Project link

Manual link

Describe your work

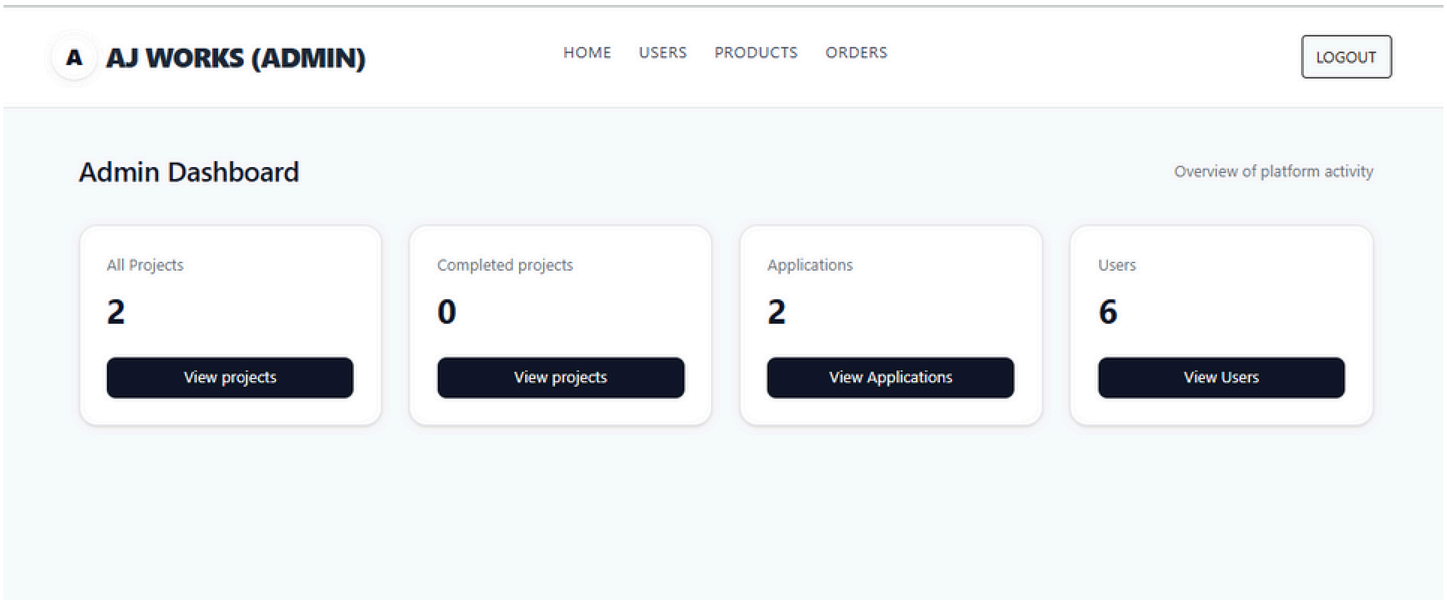
Submit project

Chat with the client

No messages yet.

Enter something...

Send



A **AJ WORKS (ADMIN)**

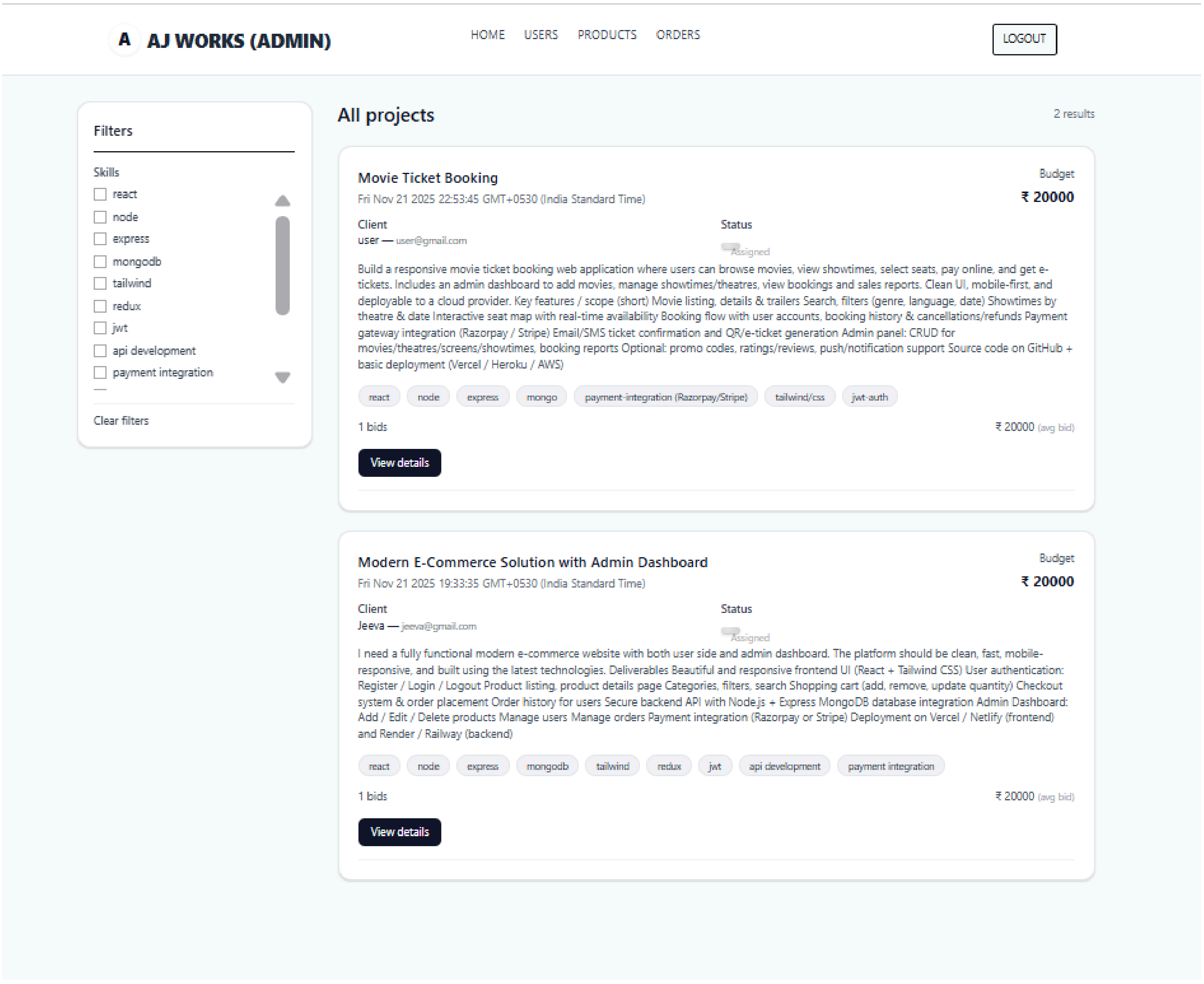
HOME USERS PRODUCTS ORDERS

LOGOUT

Admin Dashboard

Overview of platform activity

All Projects	Completed projects	Applications	Users
2	0	2	6
View projects	View projects	View Applications	View Users



A **AJ WORKS (ADMIN)**

HOME USERS PRODUCTS ORDERS

LOGOUT

Filters

Skills

- [ ] react
- [ ] node
- [ ] express
- [ ] mongodb
- [ ] tailwind
- [ ] redux
- [ ] jwt
- [ ] api development
- [ ] payment integration

Clear filters

All projects

2 results

Movie Ticket Booking

Budget

₹ 20000

Fri Nov 21 2025 22:53:45 GMT+0530 (India Standard Time)

Client
user — user@gmail.com

Status
Assigned

Build a responsive movie ticket booking web application where users can browse movies, view showtimes, select seats, pay online, and get e-tickets. Includes an admin dashboard to add movies, manage showtimes/theatres, view bookings and sales reports. Clean UI, mobile-first, and deployable to a cloud provider. Key features / scope (short) Movie listing, details & trailers Search, filters (genre, language, date) Showtimes by theatre & date Interactive seat map with real-time availability Booking flow with user accounts, booking history & cancellations/refunds Payment gateway integration (Razorpay / Stripe) Email/SMS ticket confirmation and QR/e-ticket generation Admin panel: CRUD for movies/theatres/screens/showtimes, booking reports Optional: promo codes, ratings/reviews, push/notification support Source code on GitHub + basic deployment (Vercel / Heroku / AWS)

react node express mongo payment-integration (Razorpay/Stripe) tailwind/css jwt-auth

1 bids

₹ 20000 (avg bid)

View details

Modern E-Commerce Solution with Admin Dashboard

Budget

₹ 20000

Fri Nov 21 2025 19:33:35 GMT+0530 (India Standard Time)

Client
Jeeva — jeeva@gmail.com

Status
Assigned

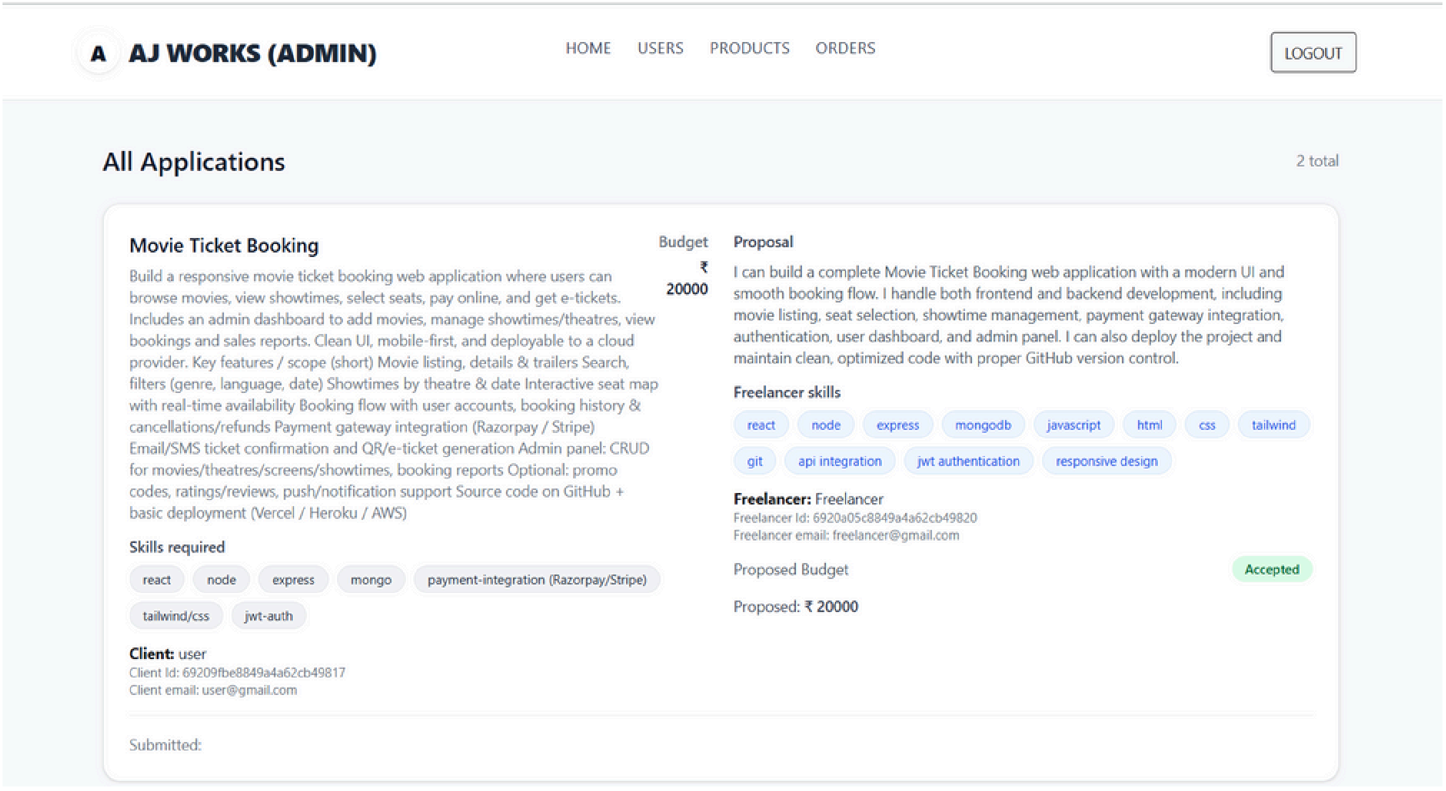
I need a fully functional modern e-commerce website with both user side and admin dashboard. The platform should be clean, fast, mobile-responsive, and built using the latest technologies. Deliverables Beautiful and responsive frontend UI (React + Tailwind CSS) User authentication: Register / Login / Logout Product listing, product details page Categories, filters, search Shopping cart (add, remove, update quantity) Checkout system & order placement Order history for users Secure backend API with Node.js + Express MongoDB database integration Admin Dashboard: Add / Edit / Delete products Manage users Manage orders Payment integration (Razorpay or Stripe) Deployment on Vercel / Netlify (frontend) and Render / Railway (backend)

react node express mongodb tailwind redux jwt api development payment integration

1 bids

₹ 20000 (avg bid)

View details



All Applications

2 total

Movie Ticket Booking

Budget ₹ 20000

Build a responsive movie ticket booking web application where users can browse movies, view showtimes, select seats, pay online, and get e-tickets. Includes an admin dashboard to add movies, manage showtimes/theatres, view bookings and sales reports. Clean UI, mobile-first, and deployable to a cloud provider. Key features / scope (short) Movie listing, details & trailers Search, filters (genre, language, date) Showtimes by theatre & date Interactive seat map with real-time availability Booking flow with user accounts, booking history & cancellations/refunds Payment gateway integration (Razorpay / Stripe) Email/SMS ticket confirmation and QR/e-ticket generation Admin panel: CRUD for movies/theatres/screens/showtimes, booking reports Optional: promo codes, ratings/reviews, push/notification support Source code on GitHub + basic deployment (Vercel / Heroku / AWS)

Skills required

react · node · express · mongo · payment-integration (Razorpay/Stripe) · tailwind/css · jwt-auth

Client: user
Client Id: 69209fbe8849a4a62cb49817
Client email: user@gmail.com

Proposal

I can build a complete Movie Ticket Booking web application with a modern UI and smooth booking flow. I handle both frontend and backend development, including movie listing, seat selection, showtime management, payment gateway integration, authentication, user dashboard, and admin panel. I can also deploy the project and maintain clean, optimized code with proper GitHub version control.

Freelancer skills

react · node · express · mongodb · javascript · html · css · tailwind · git · api integration · jwt authentication · responsive design

Freelancer: Freelancer
Freelancer Id: 6920a05c8849a4a62cb49820
Freelancer email: freelancer@gmail.com

Proposed Budget

Accepted

Proposed: ₹ 20000

Submitted:

12. Known Issues

- Payment gateway not yet integrated
- Real-time chat could be further optimized
- Email notifications not implemented
- Dark mode not added yet
- Admin analytics dashboard in progress

13. Future Enhancements

- Integrate Stripe/Razorpay for payments
- Add video call option for client-freelancer meetings
- Improve search with AI suggestions
- Add skill tests and certifications
- Build mobile app version
- Add project milestone and invoice system