WEB BASED PROJECT BIDDING SYSTEM FOR AN IT ORGANIZATION

By

A. FAMEEHA ZEENATH

Register No: 820323622019

of

A.V.C. COLLEGE OF ENGINEERING

Mayiladuthurai, Mannampandal – 609305

A PROJECT REPORT

submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

in partial fulfillment of the requirements for the award of the degree

of

MASTER OF COMPUTER APPLICATIONS

JULY 2025

WEB BASED PROJECT BIDDING SYSTEM FOR AN IT ORGANIZATION

By

A. FAMEEHA ZEENATH

Register No: 820323622019

of

A.V.C. COLLEGE OF ENGINEERING

Mayiladuthurai, Mannampandal – 609305

A PROJECT REPORT

submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

in partial fulfillment of the requirements for the award of the degree

of

MASTER OF COMPUTER APPLICATIONS

JULY 2025

CERTIFICATE

A.V.C. COLLEGE OF ENGINEERING

Mannampandal, Mayiladuthurai– 609305

| Approved by AICTE | | Affiliated to Anna University, Chennai |

| Accredited by NBA (CSE, EEE, ECE & MECH) & NAAC with 'A' Grade (3rd Cycle) |

| An ISO 9001:2015 Certified Institution |

DEPARTMENT OF COMPUTER APPLICATIONS

BONAFIDE CERTIFICATE

Certified that this project report titled **"WEB BASED PROJECT BIDDING SYSTEM FOR AN IT ORGANIZATION "** is the bonafide work of **Ms. A. FAMEEHA ZEENATH (Reg. No: 820323622019)** who carried out the project under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred an earlier occasion on this or any other candidate.

SUPERVISOR **HEAD OF THE DEPARTMENT**

Place:

Date:

Submitted for the Anna University Examination held on:----------

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

DECLARATION

I, **A. FAMEEHA ZEENATH (Reg. No: 820323622019),** the student of A.V.C. College of Engineering, Department of Computer Applications, would like to declare that the project work entitled **"WEB BASED PROJECT BIDDING SYSTEM FOR AN IT ORGANIZATION"** is the result of the original work done by me during the course of the study and is submitted on the partial fulfilment for the award of degree of **"MASTER OF COMPUTER APPLICATIONS"** of Anna University, Chennai.

Place: Mannampandal

Signature

[A.FAMEEHA ZEENATH]

Date:

ABSTRACT

ABSTRACT

A freelance software developer is a professional who independently works on software projects for clients. These developers offer specialized expertise in various roles needed when developing software projects. Hiring these developers can aid smaller businesses to effectively hire outside services without spending large amounts for a dedicated team. Freelancers are a great, cost-effective alternative for start-ups and businesses looking to reduce hiring costs. Hiring specialized trained developers allows companies to utilize their expertise without additional training while reducing overhead spending compared to hiring a full-time team. The platform empowers buyers to peruse detailed profiles, assessing the skills, experiences, and past work of registered coders. Through a seamless bidding and project posting system, companies can articulate their needs, allowing freelancers to submit proposals that align with specific project requirements. With a sophisticated skill-matching algorithm, transparent review systems, and user-friendly features, Web Based Project Bidding System for an IT Organization aims to redefine the freelance recruitment experience, making it seamless, secure, and business-friendly.

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

This project work itself is an acknowledgment to the intensity drive and technical competence of many who have contributed to it. I express my thanks to everyone who contributed and guided me much for the successful completion of this simple effort.

I express my sincere thanks to **Dr. P. BALASUBRAMANIAN, M.E., Ph.D., Principal, A.V.C. College of Engineering** who helped me in providing required facility in completing the project.

I express my gratitude thanks to **Dr. S. SELVAMUTHUKUMARAN, M.C.A., Ph.D., Vice-Principal, Professor cum HOD, Department of Computer Applications, A.V.C. College of Engineering** who allowed me to do my final project work independently and effectively.

I am grateful to my internal project guide **Dr. A. SUGANTHI, MCA., M.Phil., Ph.D., Assistant Professor, Department of Computer Applications, A.V.C College of Engineering** for her valuable guidance, idea, advice and encouragement for the successful completion of this project to identify our mistakes and also achieving our goal.

I am thankful to my external project guide **Mr. TAJUDEEN,** Project Developer, Mind IT Solution, Trichy for his valuable guidance for successful completion of this project.

I would like to record our deepest gratitude to our parents for their constant encouragement and support which motivated us to complete our project on time.

My hearty thankful to my friends who were with me and gave me all the support I needed to complete my project successfully.

A. FAMEEHA ZEENATH

TABLE OF CONTENTS

LIST OF TABLES (i)

LIST OF FIGURES

LIST OF ABBREVIATIONS

S.NO.	ABBREVIATION	MEANING
1	PY	Python
2	SQL	Structural Query Language
3	HTML	Hyper Text Markup Language
4	CSS	Cascading Style Sheet
5	GPL	General Public License
6	IDE	Integrated Developing Environment
7	UML	Unified Modelling Language
8	CSRF	Cross Site Request Forgery
9	XSS	Cross Site Scripting

CHAPTER 1

INTRODUCTION

CHAPTER 1
INTRODUCTION

A Freelancer is an individual who earns money on a per-job basis, usually for short-term work as an independent contractor. A freelancer is not an employee of a firm so they're at liberty to complete different jobs concurrently for various individuals or firms unless they're contractually committed to working exclusively until a particular project is completed. Freelancers are considered part of the gig economy.

1.1. Freelancer

Key Takeaways

- A freelancer is an independent contractor who earns wages on a per-job or per-task basis, typically for short-term work.
- Benefits of freelancing include the freedom to work from home or from a non-traditional workspace, a flexible work schedule, and a better work/life balance.
- An independent journalist who reports on stories of their own choosing and then sells the articles to the highest bidder would be an example of a freelancer.
- Freelancers don't usually enjoy benefits from the firms they perform work for, such as health insurance or retirement plans.
- Gig workers may also be considered freelancers.

Understanding Freelancers

Freelancers are typically considered to be independent contractors. They are contracted to perform work for another entity as a non-employee. Instead of being employed in a permanent position by a company, freelancers work on a per project or contract basis, often for numerous different clients or companies. They are self-employed, meaning they are considered to work for themselves. There isn't a set amount of hours you must work to be considered a freelancer.

Some freelancers freelance full-time. Other people, meanwhile, may freelance as a side job to get some extra income. As independent contractors, freelancers typically require signed contracts for the job to be done. They'll agree to a predetermined fee based on the time, skill, and effort required to complete the task. This may be a flat fee, a per-hour fee, a per-day fee, a per-project fee, or some other similar measure.

Web-based freelance marketplace designed to connect skilled developers with employers across the globe. The platform provides a centralized environment where freelancers can showcase their skills, bid on projects, and receive secure payments, while employers can post job requirements, review portfolios, and hire the most suitable candidates.

Developed using Python (Flask) for the backend and HTML, CSS, and JavaScript for the frontend, project bidding system aims to streamline the freelancing process through well-structured modules like Registration, Portfolio, Project Bidding, Payment, Feedback, and Reports.

The system ensures transparency, efficiency, and security in every transaction, making it a reliable platform for both developers and businesses. By automating project management and enabling real-time communication between freelancers and clients, this project plays a key role in bridging the gap between global talent and opportunity.

CHAPTER 2

ORGANIZATION PROFILE

CHAPTER 2

ORGANIZATION PROFILE

GENIAL DIGITECH SOLUTIONS

Genial Digitech based in Chennai, India has been started by a team of well-groomed Professionals with decade of experience in business Process and IT domain. With its qualified professionals team the company is successful in the areas of Software Development, Support, Resourcing and Infrastructure Setup. We fulfil our client's requirements by providing efficient, cost effective, quality solutions on time. We in Genial Digitech have trained and experienced professionals in the areas of IT Development, Implementation, Testing, Production Support and Infrastructure Setup.

Overview

Genial Digitech Solutions is a forward-thinking IT services and digital transformation company committed to delivering high-quality, innovative solutions that drive real business results. Established in 2020, the company has rapidly evolved into a trusted technology partner for startups, SMEs, and enterprise-level clients across the globe.

With a strong focus on digital innovation, Genial Digitech Solutions specializes in providing custom software development, web and mobile application development, cloud services, UI/UX design, data analytics, and IT consulting services. Our mission is to empower businesses through technology and help them stay ahead in a fast-paced digital landscape.

Vision

To be a globally recognized technology partner, offering sustainable and scalable digital solutions that transform businesses and improve lives.

Mission

- To deliver reliable, cost-effective, and cutting-edge IT services.
- To build long-term partnerships through transparency and trust.
- To foster a culture of continuous innovation and excellence.

Core Values

- **Integrity**: We believe in doing the right thing with honesty and accountability.
- **Innovation**: We encourage creative thinking and embrace new technologies.
- **Customer-Centricity**: Our clients are at the heart of everything we do.
- **Quality**: We are committed to delivering nothing less than excellence.
- **Collaboration**: We value teamwork and believe in growing together.

Services Offered

1. **Custom Software Development**
   Tailored software solutions that align with your business goals and workflows.
2. **Mobile App Development**
   Engaging and performance-oriented Android and iOS apps with modern UI.
3. **Web Development**
   Scalable, secure, and SEO-friendly websites and web applications.
4. **Digital Marketing**
   Strategic marketing campaigns including SEO, PPC, SMM, and content marketing.
5. **Cloud Solutions**
   Migration, deployment, and support for AWS, Azure, and Google Cloud.
6. **UI/UX Design**
   User-centered designs to enhance usability, accessibility, and user satisfaction.
7. **IT Consulting**
   Expert advice to improve IT performance and technology alignment with business goals.

Clients and Reach

Genial Digitech Solutions serves clients from various domains such as e-commerce, healthcare, education, finance, and logistics. With a growing client base in India, UAE, the USA, and Southeast Asia, we are proud to be a global company with a local heart.

Team

Our team comprises passionate developers, designers, marketers, and business strategists who work together to deliver results beyond expectations. We nurture talent and ensure continuous learning and development through workshops, certifications, and mentoring programs.

Why Choose Us?

- Proven track record of successful project deliveries
- Agile development methodology
- Strong emphasis on client satisfaction
- Transparent communication and reporting
- 24/7 support and maintenance services

Contact Us

Plot no 531, MERRYDALE, F1, Sivaraman Street,
Ram Nagar North Extension, Madipakam,
Chennai 600 091, Tamilnadu, India.
Website : www.genialdigitech.com
Email : info@genialdigitech.com
Phone : 044 4208 5859

CHAPTER 3

SYSTEM CONFIGURATION

CHAPTER 3

SYSTEM CONFIGURATION

3.1. Hardware Requirements

- Processors: Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAM
- Disk space: 320 GB
- Operating systems: Windows® 10, macOS*, and Linux*

3.2 Software Requirements

- Server Side : Python
- Client Side : HTML, CSS, Bootstrap
- IDE : Dreamweaver
- Back end : MySQL 5.
- Server : Wampserver 2i

CHAPTER 4

SOFTWARE FEATURES

CHAPTER 4
SOFTWARE FEATURE

4.1. FRONT END

4.1.1. PYTHON 3.7.4

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages. Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain.

Python is currently the most widely used multi-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc. The biggest strength of Python is huge collection of standard library which can be used for the following:

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)

- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like OpenCV, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia
- Scientific computing
- Text processing and many more.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of their features support functional programming. Many other paradigms are supported via extensions, including design by contract and logic programming. Python is often referred to as a *'glue language'* because it can seamlessly integrate components written in other languages.

Python's core philosophy is summarized in the Zen of Python (PEP 20), which includes aphorisms such as these:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

However, Python features regularly violate these principles and have received criticism for adding unnecessary language bloat. Responses to these criticisms note that the Zen of Python is a guideline rather than a rule. The addition of some new features had been controversial: Guido van Rossum resigned as Benevolent Dictator for Life after conflict about adding the assignment expression operator in Python 3.8.

4.1.2. FRONT END: DESIGN

BOOTSTRAP 4

Bootstrap is a free and open-source tool collection for creating responsive websites and web applications. It is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.

It solves many problems which we had once, one of which is the cross-browser compatibility issue. Nowadays, the websites are perfect for all the browsers (IE, Firefox, and Chrome) and for all sizes of screens (Desktop, Tablets, Phablets, and Phones). All thanks to Bootstrap developers -Mark Otto and Jacob Thornton of Twitter, though it was later declared to be an open-source project.
Easy to use: Anybody with just basic knowledge of HTML and CSS can start using Bootstrap
Responsive features: Bootstrap's responsive CSS adjusts to phones, tablets, and desktops
Mobile-first approach: In Bootstrap, mobile-first styles are part of the core framework
Browser compatibility: Bootstrap 4 is compatible with all modern browsers (Chrome, Firefox, Internet Explorer 10+, Edge, Safari, and Opera)

Using an IDE

As good as dedicated program editors can be for your programming productivity, their utility pales into insignificance when compared to Integrated Developing Environments (IDEs), which offer many additional features such as in-editor debugging and program testing, as well as function descriptions and much more.

Bootstrap is a powerful, open-source front-end framework developed by Twitter for building responsive and mobile-first websites and web applications. It provides a rich collection of pre-designed HTML, CSS, and Javascript components that significantly speed the web development.

4.1.3. FLASK

Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible. Flask does not have built-in abstraction layer for database handling, nor does it have formed a validation support. Instead, Flask supports the extensions to add such functionality to the application. Although Flask is rather young compared to most Python frameworks, it holds a great promise and has already gained popularity among Python web developers. Let's take a closer look into Flask, so-called "micro" framework for Python. Flask was designed to be easy to use and extend.

The idea behind Flask is to build a solid foundation for web applications of different complexity. From then on you are free to plug in any extensions you think you need. Also you are free to build your own modules. Flask is great for all kinds of projects. It's especially good for prototyping. Flask is part of the categories of the micro-framework. Micro-framework is normally framework with little to no dependencies to external libraries.

This has pros and cons. Pros would be that the framework is light, there are little dependency to update and watch for security bugs, cons is that some time we will have to do more work by ourself or increase ourself the list of dependencies by adding plugins. Additionally, It has built- in support for handling HTTP methods like GET and POST, managing cookies and sessions, and integrating with various databases such as MySQL, SQLite and PostgreSQL.

4.2 BACK END: MYSQL

MySQL tutorial provides basic and advanced concepts of MySQL. Our MySQL tutorial is designed for beginners and professionals. MySQL is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database. MySQL is open-source and free software under the GNU license. It is supported by Oracle Company.

MySQL database that provides for how to manage database and to manipulate data with the help of various SQL queries. These queries are: insert records, update records, delete records, select records, create tables, drop tables, etc. There are also given MySQL interview questions to help you better understand the MySQL database.

MySQL is currently the most popular database management system software used for managing the relational database. It is open-source database software, which is supported by Oracle Company. It is fast, scalable, and easy to use database management system in comparison with Microsoft SQL Server and Oracle Database. It is commonly used in conjunction with PHP scripts for creating powerful and dynamic server-side or web-based enterprise applications.

It is developed, marketed, and supported by MySQL AB, a Swedish company, and written in C programming language and C++ programming language. The official pronunciation of MySQL is not the My Sequel; it is My Ess Que Ell. However, you can pronounce it in your way. Many small and big companies use MySQL. MySQL supports many Operating Systems like Windows, Linux, MacOS, etc. with C, C++, and Java languages.

4.2.1. WAMPSERVER

WampServer is a Windows web development environment. It allows us to create web applications with Apache2, PHP and a MySQL database. Alongside, PhpMyAdmin allows us to manage easily our database.

WAMPServer is a reliable web development software program that lets you create web apps with MYSQL database and PHP Apache2. With an intuitive interface, the application features numerous functionalities and makes it the preferred choice of developers from around the world. The software is free to use and doesn’t require a payment or subscription.

WampServer is a Windows-based web development environment that allows users to create dynamic web applications using Apache2, PHP, and MySQL. It provides an easy-to-use interface for managing server settings, databases, and projects locally. WampServer simplifies the process of setting up a local server environment, making it ideal for developers to test websites before deployment. It also includes tools like phpMyAdmin for database management. With its tray icon, users can quickly start, stop, or restart services and switch between different PHP or MySQL versions. WampServer is widely used for learning, development, and testing purposes in a local environment.

CHAPTER 5

PROJECT DESCRIPTION

CHAPTER 5
PROBLEM DESCRIPTION

5.1 OVERVIEW OF THE PROJECT

In today's fast-paced digital economy, businesses, especially startups and small to medium-sized enterprises (SMEs), often face challenges when it comes to hiring skilled software developers. Traditional recruitment methods, such as hiring full-time employees or engaging with recruitment agencies, are not only time-consuming but also expensive. Moreover, these methods often require companies to invest in training and resources to ensure that the hired employees meet the project-specific requirements. Additionally, businesses may struggle to find developers with niche or highly specialized skills, especially when the local talent pool is limited.

5.2 PROBLEM STATEMENT

This limitation can lead to delays in project timelines, increased costs, and in some cases, the inability to bring innovative products to market quickly. To address these challenges, the proposed project, **Web based project bidding system for an IT organization**, introduces a comprehensive and user-friendly platform designed to simplify and optimize the hiring process for freelance software developers.

Web based project bidding system for an IT organization offers a global marketplace where businesses can connect with a curated pool of talented developers from around the world. This platform enables companies to post projects, receive tailored proposals from freelancers, and choose the best candidates based on detailed profiles, skillsets, and previous work experience.

The introduction of a sophisticated skill-matching algorithm ensures that businesses find the right developer quickly, reducing the time and cost associated with traditional hiring methods. The platform also features a transparent review and rating system, which builds trust and accountability between businesses and freelancers.

5.3 MODULE DESCRIPTIONS

This project contains six modules, They are,

NAME OF THE MODULES

Registration Module

- Projects
- Freelancer
- Employer

Portfolio Module

- Create Portfolio
- Add Experience
- Add Education & Certifications
- Upload Work Samples
- Upload / Delete Portfolio Info

Project Bidding Module

- View Available Projects
- Place a Bid
- Manage Bids
- Notification Alerts

Incentives & payment Module

- View Earning
- Link Payment Methods
- Receive Payment
- Track Incentives

Feedback & Review Module

- Give Feedback to Freelancer
- Receive Employee Feedback
- View Past Review

Report Module

- Project wise- Report
- Freelancer wise- Report
- Employee wise-report

MODULE DESCRIPTIONS

Registration Module

- **Projects –** Register new projects with detailed descriptions, specify project categories, required skills, and timelines. Set budget range and payment terms for each project. This feature allows employers to register new projects by entering comprehensive details such as the project title, description, required skills, category, budget range, timeline, and payment terms. These registered projects become available for freelancers to view and bid on.

- **Freelancer –** Sign up using email or social media, create a personalized profile with skills, Bio, and photo, set hourly rate, availability, and preferred work type. Freelancers can register through email or social media. During registration, they can create a professional profile including their personal details, skills, profile picture, bio, hourly rate, availability, and preferred job types. This profile is used by employers to evaluate and hire suitable freelancers.

- **Employer –** Register to post job listings and view freelancer profiles, manage multiple project postings from a single dashboard, communicate with freelancers and assign tasks**.** Employers can sign up to post project listings, manage ongoing and completed projects,

and search for freelancers. Upon registration, employers are given access to a dashboard to monitor project progress, bids received, and freelancer profiles.

Portfolio Module

- **Create Portfolio –** Build a professional portfolio with personal details, Highlight top skills and areas of expertise, add a professional summary or tagline. Freelancers can create a complete portfolio by entering personal and professional information such as name, designation, skills, and profile summary to represent their identity.

- **View Experience –** Add previous job or freelance experiences with timeline, list achievement or milestones reached in each role. Enables users to enter their previous work history including job title, organization name, work duration, and a detailed description of the responsibilities and achievements. This gives employers a clear view of the freelancer's background.

- **View Qualification –** Enter academic qualifications with institution and year, upload Users can input academic qualifications such as school, college, degrees, and certifications obtained through online platforms or professional bodies. This adds to the freelancer's credibility and shows commitment to professional development.

- **Upload Work Samples –** showcase work using images, documents, or links, categorize samples based on skill or industry. Freelancers can upload images, videos, documents, GitHub links, or portfolio URLs of their previous projects. This provides concrete proof of their work quality and style.

- **Upload / Delete Portfolio Info –** Edit, update, or delete outdated portfolio items, maintain up-to-date information. Allows for real-time updates, letting freelancers keep their profile current by adding new achievements or removing outdated information. Freelancers can tag their portfolio with relevant skills and keywords, improving discoverability by employers through search and filter functions.

Project Bidding Module

- **View Available Projects –** View available projects posted by employers, view project details such as description, skills required, and employer profile. Freelancers can **view available projects** with complete details, including title, description, required skills, budget, and timeline. They can filter projects based on categories, skill match, or budget range, and even bookmark them for future bidding.

- **Place a Bid-** Place bids on suitable projects based on skills and interest, mention estimated cost, completion time, and strategy. Withdraw or edit bids before deadline. The **bid placement feature** allows freelancers to submit competitive offers by entering a proposed price, delivery timeline, and a personalized proposal. They can also attach files or links to relevant work samples and choose between fixed or hourly bids.

- **Manage Bids –** Manage submitted bids and update proposals, view the list of all submitted bids and their current status. Through the **Manage Bids** section, freelancers can monitor all their active, pending, accepted, or rejected bids, edit or withdraw proposals, and analyze past bidding performance to improve future submissions.

- **Notification Alerts –** Receive real time alerts and notifications, receive notifications for bid acceptance or message from employers. Enable or disable email and in- app notifications. The **Notification Alert system** sends real-time updates about bid status changes, new messages from employers, upcoming deadlines, and recommended projects that match the freelancer's skill set.

Payment & Incentives Module

- **View Earning –** View total earning from all completed projects, access detailed transaction history with dates and project names, generate earnings report for custom date ranges. The **View Earnings** feature allows freelancers to see a detailed breakdown of their income from completed projects, including the amount earned, deductions, and pending payments.

- **Link Payment Methods –** Add multiple payment options, set a default method for receiving payments, securely manage and update payment credentials. Through the **Link Payment Methods** functionality, users can securely connect their preferred payment gateways such as bank accounts to enable smooth withdrawals.

- **Receive Payment –** Receive payment automatically after project completion, get status updates for pending or processed payments. The **Receive Payment** option ensures timely transfer of funds once a project is successfully completed and approved by the employer.

- **Track Incentives –** Track bonuses for early delivery or high quality work, view performance- based rewards from employers. Freelancers can also **Track Incentives** like performance-based bonuses, milestone rewards, or loyalty-based perks offered by the platform. This module ensures accurate record-keeping, timely transactions, and a transparent payment experience, helping freelancers manage their finances efficiently and confidently.

Feedback & Review Module

- **Give Feedback to Freelancer –** Provide feedback to freelancer after project completion. Employers can rate freelancers based on project performance. Through the **Give Feedback to Freelancer** feature, employers can rate and review freelancers based on criteria such as work quality, communication, timeliness, and professionalism. This helps future employers make informed decisions.

- **Receive Employee Feedback –** Freelancers receives ratings from employers after project completion, View employer comments and suggestions for growth. **Receive Employee Feedback** option enables freelancers to view the ratings and comments provided by clients, which can be used to improve future performance.

- **View Past Review –** Access full history of feedback and reviews received and given. Filter reviews by date, project, or rating. Export feedback reports for personal record or profile use. Both parties can **View Past Reviews**, offering a history of feedback received and

given, contributing to the user's overall credibility and reputation on the platform. This module plays a key role in building trust, maintaining quality standards, and encouraging responsible behavior from all users in the marketplace.

Report Module

- **Project wise - report –** Submit reports related to specific project issues, include project ID, timeline, milestones and related documents. Allow status tracking of submitted reports, enable admins to take necessary actions. The **Project-wise Report** feature allows tracking of individual project progress, timelines, completion status, assigned freelancers, and payment details.

- **Freelancer wise -report** – File reports about freelancer behavior or performance, Attach communication logs or task evidence. **Freelancer-wise Report** gives a comprehensive overview of each freelancer's activities, including the number of projects completed, earnings, ratings, and feedback received.

- **Employee wise - report**– Submit complaints or feedback about employers, Freelancers can provide supporting details for better assessments, platform admins handle the report to ensure air resolutions. the **Employee-wise Report** displays the employer's hiring history, posted projects, feedback given, and payment records. These reports support performance evaluation, transparency, and effective decision-making, helping maintain quality and accountability across the platform.

CHAPTER 6

SYSTEM ANALYSIS

CHAPTER 6
SYSTEM ANALYSIS

6.1 EXISTING SYSTEM

The existing systems for freelance and project management platforms involve several key components:

- **Job Listings and Bidding**

Businesses post job opportunities detailing project requirements, deadlines, budgets, and necessary skills. This helps freelancers understand the scope and expectations of each project.

Freelancers browse these listings and submit proposals that include their bid amounts, timelines, and execution strategies. This system enables businesses to compare different offers and choose the most suitable freelancer.

- **Freelancer Profiles**

Freelancers create profiles that showcase their skills, experience, and portfolios. Profiles often include details such as past work, educational background, and professional certifications. Freelancers can tag their profiles with specific skills and expertise to make it easier for businesses to find relevant talent.

- **Communication and Collaboration Tools**

Platforms provide built-in messaging systems for real-time communication between businesses and freelancers. This facilitates discussions regarding project details, requirements, and progress. Users can share files, documents, and other resources necessary for project completion directly through the platform.

- **Payment and Contract Management**

Businesses and freelancers can create and manage contracts that outline project terms, payment schedules, and deliverables. Platforms handle payment transactions securely, often including features for milestone-based payments or payment upon project completion.

- **Review and Rating Systems**

Users can rate and review each other based on their experiences, contributing to a system of feedback and trust-building. Reviews and ratings are publicly displayed on user profiles, providing future collaborators with insights into the reliability and quality of work.

- **Administrative Oversight**

Platform administrators manage user accounts, including handling registration, account issues, and ensuring adherence to platform guidelines. Admins monitor active projects to ensure compliance with platform standards and intervene in case of disputes.

6.1.1 DISADVANTAGES

- **Limited Customization**: Platforms offer generic solutions, making it hard to tailor features to specific user needs.
- **Inadequate Skill Matching**: Skill-matching algorithms may not accurately align freelancers with project requirements.
- **Inefficient Proposal Management**: Evaluating multiple proposals is cumbersome without advanced comparison tools.
- **Security Concerns**: Vulnerabilities in data security and privacy can expose sensitive information.
- **Basic Communication Tools**: Limited communication features may hinder effective project discussions.
- **Inefficient Payment and Contract Management**: Difficulties in automating payments and managing contracts can cause transaction issues.
- **Limited Analytics**: Basic analytics restrict insights into project performance and user engagement.
- **Dispute Resolution Challenges**: Lack of effective mechanisms can leave conflicts unresolved.
- **Notification Overload**: Excessive or poorly managed notifications can lead to missed updates or user fatigue.
- **User Experience Issues**: Outdated interfaces and performance problems can detract from overall user satisfaction.

6.2 PROPOSED SYSTEM

The proposed system is an advanced freelancer management platform designed to address the limitations of existing systems by incorporating sophisticated features for better customization, efficiency, and security. This platform aims to provide a comprehensive and streamlined experience for freelancers and clients, leveraging modern technologies to enhance performance and user satisfaction.

- **Customization Engine**

The platform offers extensive customization options for user profiles, allowing freelancers and clients to tailor their information and project requirements to specific needs. Users can create highly specific project postings with detailed criteria, enhancing the match between projects and freelancers.

- **Intelligent Skill Matching**

Utilizes cutting-edge algorithms to accurately match freelancers' skills with project requirements, improving the efficiency and relevance of job matches. Employs AI to recommend the best freelancers or projects based on historical data and user preferences.

- **Management System**

Features tools that automatically evaluate and rank proposals based on predefined criteria, streamlining the selection process. Provides a user-friendly interface for managing and reviewing multiple proposals, facilitating better decision-making.

- **Enhanced Security Framework**

Implements state-of-the-art encryption to safeguard sensitive user data and transactions. Adds an additional layer of security with multi-factor authentication for user accounts.

- **Robust Communication Tools**

Supports seamless video calls within the platform for more effective communication and collaboration. Offers collaborative tools and chat features to enhance interaction and project management.

- **Automated Payment and Contract Management**

Uses smart contracts to automate the execution and enforcement of agreements, ensuring accurate and timely payments. Integrates with reliable payment systems for secure financial transactions and contract management.

- **Advanced Analytics and Reporting**

Provides in-depth analytics and reporting on project performance, user engagement, and financial metrics. Allows users to create dashboards tailored to their specific needs and interests.

- **Effective Dispute Resolution Mechanisms**

Includes features for resolving disputes and mediating conflicts between freelancers and clients. Tracks the status of disputes and provides feedback on the resolution process.

- **Customizable Notification System**

Users can configure notification preferences to receive relevant updates and avoid overload. Manages and prioritizes notifications to ensure important updates are not missed.

- **Enhanced User Experience**

Features a contemporary and intuitive design that improves navigation and overall user satisfaction. Ensures optimal performance and accessibility across various devices and screen sizes.

The proposed system aims to significantly improve the freelancer management experience by offering advanced features and functionalities, addressing the limitations of current platforms, and enhancing overall user satisfaction and efficiency.

6.2.1 ADVANTAGES OF PROPOSED SYSTEM

- **Customizable User Profiles**: Allows for tailored profiles and project listings to better match specific needs.
- **Advanced Skill Matching**: Utilizes sophisticated algorithms and AI for more accurate freelancer-project alignment.
- **Streamlined Proposal Management**: Automated evaluation and centralized dashboard simplify proposal review and selection.
- **Enhanced Security**: Incorporates advanced encryption and multi-factor authentication to protect user data.
- **Improved Communication Tools**: Integrates video conferencing and shared workspaces for effective collaboration.
- **Automated Payment Handling**: Uses smart contracts for accurate and timely payment processing.

- **Comprehensive Analytics**: Provides detailed reporting and customizable dashboards for better insights into performance.
- **Effective Dispute Resolution**: Features automated mediation tools and resolution tracking to handle conflicts efficiently.
- **Personalized Notifications**: Customizable alert settings reduce notification overload and ensure important updates are received.
- **Modern User Interface**: Offers an intuitive and responsive design for an enhanced user experience across devices.

CHAPTER 7

SYSTEM DESIGN

CHAPTER 7

SYSTEM DESIGN

7.1. UML DIAGRAM

UML, which stands for Unified Modelling Language, is a way to visually represent the architecture, design, and implementation of complex software systems. UML is a standardized modelling language that can be used across different programming languages and development processes, so the majority of software developers will understand it and be able to apply it to their work.

7.1.1. Use case Diagram

Use case diagram are usually referred to as behaviour diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors)

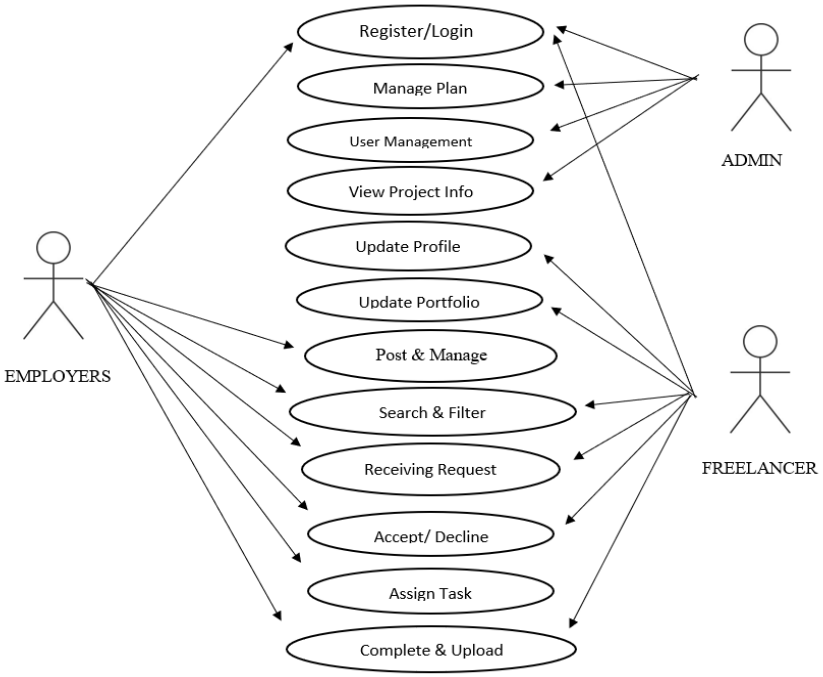


Fig.No 7.1.1 Use case diagram for Web based Project Bidding System for an IT Organization

7.1.2 CLASS DIAGRAM

In Software Engineering, a class diagram in the unified modelling language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

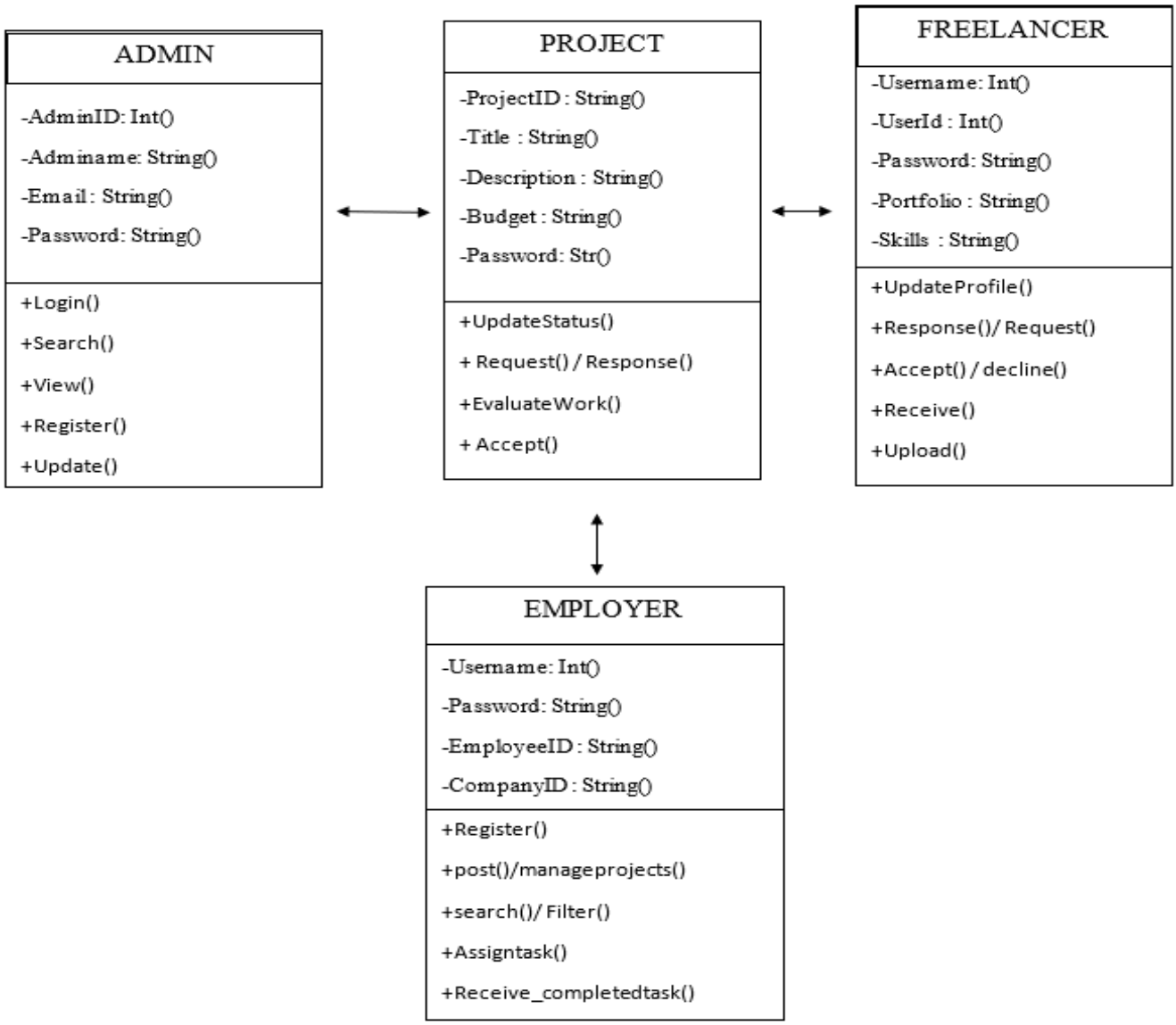


Fig.No. 7.1.2 Class Diagram for Web based Project Bidding System for an IT Organization

7.1.3. ACTIVITY DIAGRAM

We use Activity diagram to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case.

Fig.No.7.1.3 Activity Diagram for Web based Project Bidding System for an IT Organization

7.1.4. SEQUENCE DIAGRAM

A Sequence Diagram is a type of UML (Unified Modelling Language) diagram that illustrates the interaction between objects or components within a system in a sequential manner.

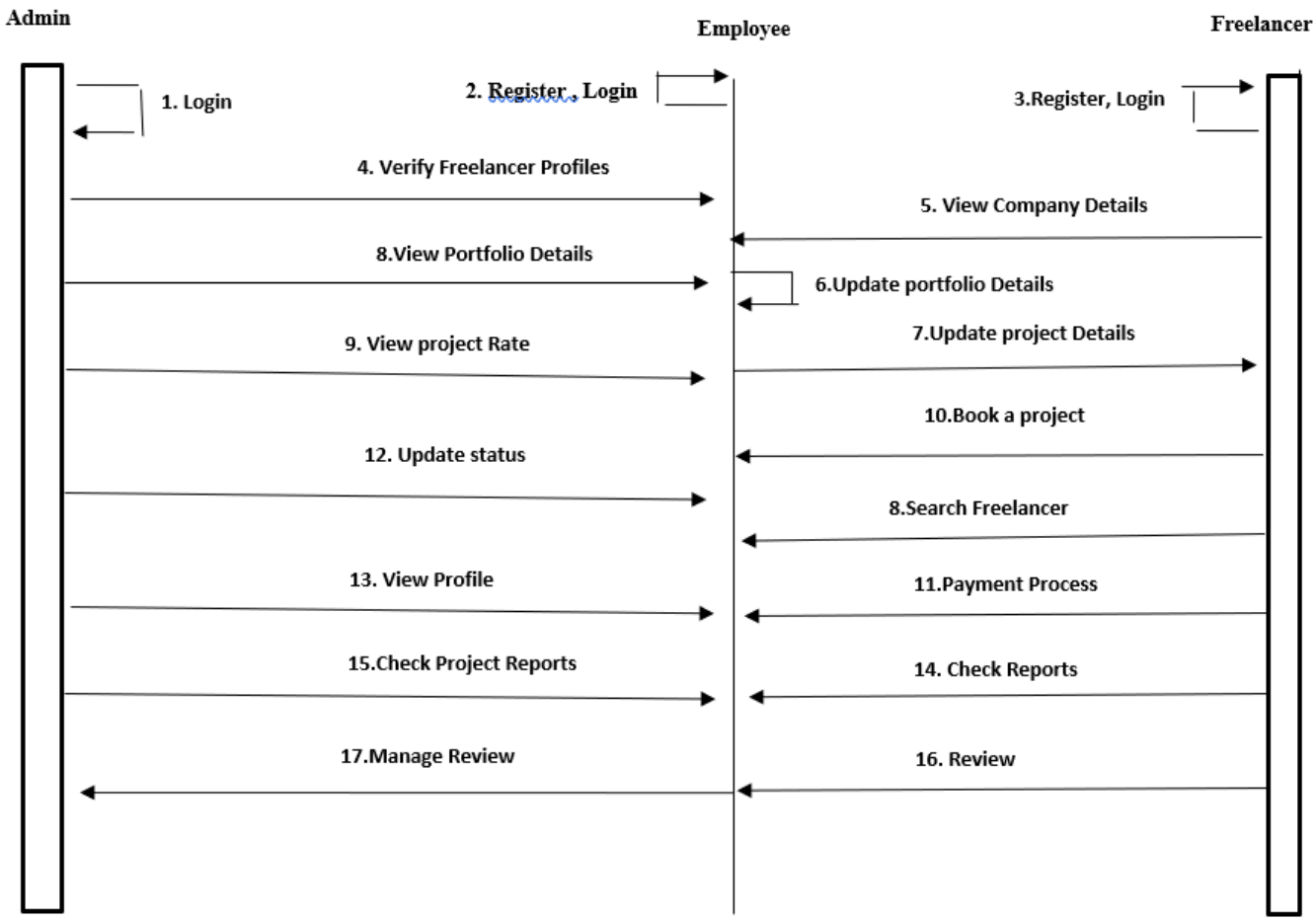


Fig.No.7.1.4 Sequence diagram for Web based Project Bidding System for an IT Organization.

7.1.5. COMPONENET DIAGRAM

A Component Diagram is shown a object integration arranged in time sequence. It depicts the object and classes involved in the scenario and the sequence of messages exchanged between the objects needed the carry out the functionality of the scenario.

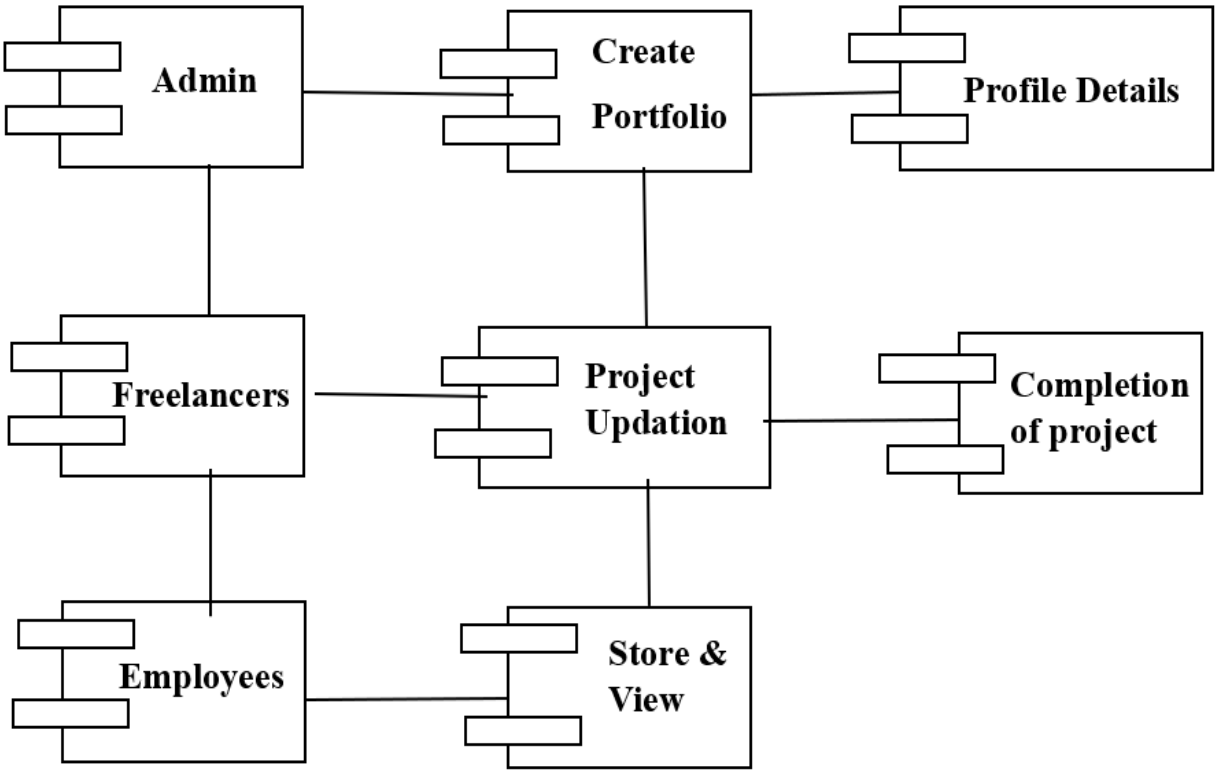


Fig.No.7.1.5 Component Diagram for Web based Project Bidding System for an IT Organization.

7.1.6. DEPLOYMENT DIAGRAM

Deployment diagram show how software is deployed on hardware components in a system. These diagrams are most useful for systems engineers, and they usually show performance, scalability, maintainability, and portability.

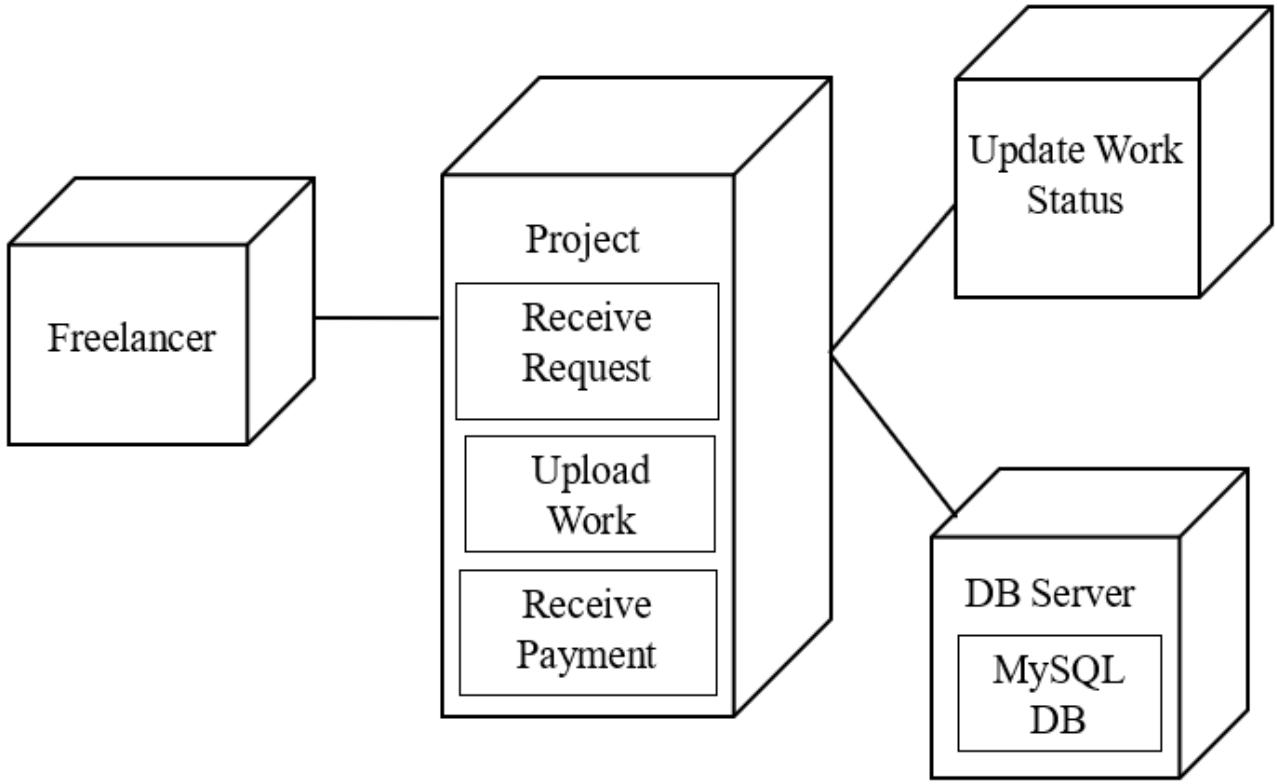


Fig.No.7.1.6 Deployment Diagram for Web based Project Bidding System for an IT Organization

7.2 DATABASE DESIGN

7.2.1. LOGIN TABLE

Table Name: Login

Description: This table stores login credentials for admin users

Field	Data Type	Constraints	Description
User_ID	VARCHAR(20)	FOREIGN KEY references from (freelancer (Freelancer_ID) Employer (Employer_ID), UNIQUE	Unique ID for Freelancer and Employer
Username	VARCHAR(30)	Not Null, UNIQUE	Unique Username
Password	VARCHAR(100)	Not Null	Encrypted Password
Email	VARCHAR(50)	Not Null	User Email
Last_Login	DATETIME	Null	Last Login Timestamp

7.2.2. FREELANCER TABLE

Table Name: Freelancer_details

Description: This table stores details of freelancers

Field	Data Type	Constraints	Description
Freelancer_ID	INT(15)	PRIMARY KEY, UNIQUE	Unique ID for the Freelancer
Full_Name	VARCHAR(50)	Not Null	Full Name of the User
Email	VARCHAR(50)	Not Null, UNIQUE	Unique Email Address
Phone	INT(15)	Not Null	Contact Number
Company_Name	VARCHAR(60)	Not Null	Company Name
Location	VARCHAR(50)	NOT NUL	Current Location
Profile_Image	BLOB	Null	Uploaded Profile Photo
Registration_Date	DATE	Not Null	Date of Account Creation

7.2.3. PORTFOLIO TABLE

Table Name: Portfolio_information

Description: This table stores freelancer portfolio and experience details

Field	Data Type	Constraints	Description
Portfolio_ID	INT(15)	PRIMARY KEY	Unique Portfolio ID
Freelancer_ID	INT(15)	FOREIGN KEY references from Freelancer (Freelancer_ID)	Unique Freelancer ID
Experience	VARCHAR(100)	Not Null	Work Experience Details
Education	VARCHAR(100)	Not Null	Educational Background & Certifications
Work_Samples	BLOB	Null	Uploaded Work Samples
Created_Date	DATE	Not Null	Portfolio Creation Date
Last_Updated	DATE	Null	Last Updated Date

7.2.4. EMPLOYER TABLE

Table Name: Employer_details

Description: This table stores Employer details

Field	Data Type	Constraints	Description
Employer_ID	INT(15)	PRIMARY KEY	Unique employer ID
Company details	VARCHAR(100)	Not Null	Company Background
Location	VARCHAR(50)	Not Null	Location of Employer
Project _Samples	BLOB	Null	Uploaded project Samples
Created_Date	DATE	Not Null	Portfolio Creation Date
Last_Updated	DATE	Null	Last Updated Date

7.2.5. PROJECT TABLE

Table Name: Project_details

Description: Stores project information placed by Employer on posted projects

Field Name	Data Type	Constraints	Description
project_id	INT	PRIMARY KEY	Unique identifier for each project
Employer_id	INT	NOT NULL, FOREIGN KEY References from employer table (Employer_ID)	ID of the Employer who posted the project
Title	VARCHAR(255)	NOT NULL	Title or name of the project
Description	VARCHAR(100)	NULL	Detailed description of the project
Budget	DECIMAL(10,2)	NOT NULL	Budget limit for the project
Deadline	DATE	NOT NULL	Deadline for project completion
Status	ENUM	NOT NULL	Current status of the project
Category	VARCHAR(100)	NOT NULL	Project category or type
created_at	TIMESTAMP	NOT NULL	Timestamp when the project was posted

7.2.6. PROJECT BIDDING TABLE

Table Name: Project_Bidding

Description: Stores bids placed by freelancers on posted projects

Field	Data Type	Constraints	Description
Bid_ID	INT(15)	PRIMARY KEY	Unique Bid ID
Project_ID	INT(15)	FOREIGN KEY(References from Project Table (Project_ID)	ID of Project Being Bid On

Freelancer_ID	INT(15)	FOREIGN KEY (References from freelancer Table (Freelancer_ID)	Bidding Freelancer
Bid_Amount	FLOAT(10,2)	Not Null	Amount Quoted by Freelancer
Bid_Date	DATE	Not Null	Date of Bid Submission
Bid_Status	VARCHAR(20)	Not Null	Status of Bid (e.g., Pending, Accepted)
Proposal_Text	VARCHAR(100)	Null	Description or Proposal Text

7.2.7. INCENTIVES

Table Name: Incentives

Description: Stores incentive and Payment details

Field	Data Type	Constraints	Description
Incentive_ID	INT(15)	PRIMARY KEY	Unique Incentive ID
Freelancer_ID	INT(15)	FOREIGN KEY (references from freelancer Table (Freelancer_ID)	Freelancer ID
Amount	FLOAT(10,2)	Not Null	Paid Amount
Incentive_Method	VARCHAR(20)	Not Null	Linked Incentive Method
Date_Paid	DATE	Not Null	Date of Payment

7.2.8. FEEDBACK REVIEW TABLE

Table Name: Feedback_Review

Description: Stores feedback and reviews between employers and freelancers

Field	Data Type	Constraints	Description
Review_ID	INT(15)	PRIMARY KEY	Unique Review ID
Project_ID	INT(15)	FOREIGN KEY References from project Table (Project_ID)	Related Project ID
Rating	INT(2)	Not Null	Rating Out of 5
Comments	VARCHAR(100)	Null	Feedback Comments
Review_Date	DATE	Not Null	Date of Review

7.2.9. REPORT TABLE

Table Name: Report

Description: Stores report between employer and freelancer

Field Name	Data Type	Constraints	Description
Report_ID	INT	PRIMARY KEY, UNIQUE	Unique identifier for each report
Freelancer_ID	INT	FOREIGN KEY(references from Freelancer Table (Freelancer_ID)	Referencing the Freelancer Id
Employer_ID	INT	FOREIGN KEY(references from employer Table (Empoyer_ID)	Referencing the employer table
Project_ID	INT	FOREIGN KEY(references from project Table (Project_ID)	The project related to the report
Report_type	VARCHAR(100)	NOT NULL	Category/type of the report (e.g., spam, fraud)

Report_message	VARCHAR(100)	NOT NULL	Detailed message describing the issue
Report_status	ENUM	NOT NULL	Current status of the report
Created_at	TIMESTAMP	NOT NULL	Time when the report was created

7.2.10. NOTIFICATION TABLE

Table Name: Notification

Description: Stores system-generated notifications for Employer and Freelancer

Field	Data Type	Constraints	Description
Notification_ID	INT(10)	PRIMARY KEY	Unique notification ID
Project_ID	INT(10)	Not Null	Project ID
Role	VARCHAR(20)	Not Null	'employer' or ‘freelancer'
Message	VARCHAR(100)	Not Null	Notification content
Date_Sent	DATETIME	Not Null	When notification was sent

CHAPTER 8

SYSTEM TESTING

CHAPTER 8

SYSTEM TESTING

8. Types of Testing

- **Unit Testing**

Unit testing focuses on verifying the functionality of individual components and modules within the platform. This phase involves testing core functions such as user registration, project posting, bidding, and payment processing in isolation to ensure each unit performs as expected. Tools like Python's unit test and Flask’s testing utilities will be utilized to execute these tests and identify any issues within the specific components.

- **Integration Testing**

Integration testing evaluates the interactions between different components of the system. This testing ensures that the front-end, back-end, and database work together seamlessly. Key interactions, including user profile updates, project creation, and bid submissions, will be tested to verify smooth data flow and functionality. Tools such as Postman for API testing and Selenium for end-to-end testing will be employed.

- **System Testing**

System testing involves validating the entire platform's compliance with specified requirements. This comprehensive testing ensures that all modules work together as intended. The user journey from registration through project completion will be tested, including communication tools, payment processing, and notification systems. Both automated UI testing using Selenium and manual exploratory testing will be conducted to ensure overall system functionality.

- **Acceptance Testing**

Acceptance testing ensures the system meets the end-users' needs and requirements. This phase includes testing the platform from the user's perspective to confirm that it meets expectations and fulfills real-world use cases. User acceptance testing (UAT) scripts and feedback sessions with potential users will help validate the system's effectiveness and usability.

- **Performance Testing**

Performance testing assesses the system's responsiveness, stability, and scalability under various conditions. The platform will undergo load, stress, and endurance tests to evaluate performance metrics such as response time and system throughput. Tools like Apache JMeter and Locust will be used to simulate various load scenarios and measure system performance.

- **Security Testing**

Security testing identifies vulnerabilities and ensures data protection. This phase includes detecting potential security threats such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). The platform’s authentication and authorization mechanisms will also be thoroughly tested to ensure robust security measures. Tools like OWASP ZAP and Burp Suite will assist in identifying and addressing security issues.

- **Compatibility Testing**

Compatibility testing ensures the platform functions correctly across different browsers and devices. The system will be tested on various web browsers (such as Chrome, Firefox, Safari) and devices (including desktop, tablet, and mobile) to ensure consistent performance and appearance. Tools like BrowserStack and CrossBrowserTesting will facilitate this testing to verify cross-platform compatibility.

- **Usability Testing**

Usability testing assesses the user interface and overall user experience of the platform. Real users will interact with the system to identify any usability issues or areas for improvement. Feedback collected from usability testing sessions and surveys will be used to enhance user satisfaction and interface design.

8.1 TEST PLAN

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product.

Test plan ID	Input	Expected Result	Actual Result	Status
TP-01	Valid email, password, and profile details	The user account is created successfully, and the user is redirected to the profile page with a confirmation message	The account is created successfully, and the user is redirected to the profile page with a confirmation message	Pass
TP-02	Valid username and password	The user logs in successfully and is redirected to the dashboard	The user logs in successfully and is redirected to the dashboard	Pass
TP-03	Project title, description, skills required, deadline, and budget	The project is posted successfully and visible on the project listings page	The project is posted successfully and is visible on the listings page	Pass
TP-04	Bid amount and proposal details	The bid is submitted successfully, and the freelancer receives a confirmation message	The bid is submitted successfully, and the freelancer receives a confirmation message	Pass
TP-05	Project requirements and freelancer skills	The system matches freelancers with relevant skills to the project and provides recommendations	Freelancers are matched based on skills, and recommendations are provided	Pass
TP-06	Multiple proposals for a project	The business can view, sort, and evaluate proposals based on freelancer qualifications	Proposals are viewed, sorted, and evaluated successfully	Pass

TP-07	Payment details and milestones	Payment is processed securely and linked to the appropriate milestones or project completion	Payment is processed securely and linked to the milestones	Pass
TP-08	Contract terms, payment schedule, and deliverables	The contract is created, stored successfully, and visible to both the business and freelancer	The contract is created and visible to both parties	Pass
TP-09	Message sent between freelancer and business	The message is delivered and displayed in real-time to the recipient	The message is delivered and displayed in real-time	Pass
TP-10	Rating and review for a completed project	The review and rating are submitted successfully and visible on the user's profile	The review and rating are submitted successfully and are visible on the profile	Pass

8.2 TEST CASE

A Test case has components that describes input, action and expected response, in order to determine if a feature of an application is working correctly.

LOGIN

TEST CASE ID	TEST PLAN ID	TEST CASE NAME	ACTUAL INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	STATUS
TC01	TP01	Valid login credentials	Correct username & password	User redirected to dashboard	User redirected to dashboard	Pass
TC02	TP01	Invalid login credentials	Incorrect password	Error message shown	Error message shown	Fail

REGISTRATION

TEST CASE ID	TEST PLAN ID	TEST CASE NAME	ACTUAL INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	STATUS
TC03	TP02	Employer registration	Name, email, username, password	Account created	Account created	Pass
TC04	TP02	Freelancer registration	Name, skills, email, password	Profile created	Profile created	Pass

PORTFOLIO

TEST CASE ID	TEST PLAN ID	TEST CASE NAME	ACTUAL INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	STATUS
TC05	TP03	Upload portfolio files	Work samples selected	Files uploaded	Files uploaded	Pass
TC06	TP03	Edit profile summary	New summary entered	Profile updated	Profile updated	Pass

PROJECT BIDDING MODULE

TEST CASE ID	TEST PLAN ID	TEST CASE NAME	ACTUAL INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	STATUS
TC07	TP04	Place a valid bid	Project ID, amount, days	Bid submitted	Bid submitted	Pass
TC08	TP04	Place an invalid bid	Missing cost or time	Error: incomplete bid	Error: incomplete bid	Fail

PAYMENT & INCENTIVES MODULE

TEST CASE ID	TEST PLAN ID	TEST CASE NAME	ACTUAL INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	STATUS
TC09	TP05	Link payment method	Bank details added	Payment method saved	Payment method saved	Pass

TC10	TP05	View earning report	Date range input	Earnings displayed	Earnings displayed	Pass

FEEDBACK & REVIEW

TEST CASE ID	TEST PLAN ID	TEST CASE NAME	ACTUAL INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	STATUS
TC11	TP06	Submit employer feedback	Voice review	Feedback saved	Feedback saved	Pass
TC12	TP06	Submit freelancer rating	Star rating + comments	Rating submitted	Rating submitted	Pass

REPORT MODULE

TEST CASE ID	TEST PLAN ID	TEST CASE NAME	ACTUAL INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	STATUS
TC13	TP07	Submit project-wise report	Project issue & files	Report filed	Report filed	Pass
TC14	TP07	Submit freelancer report	Chat log + comments	Report submitted	Report submitted	Pass

TEST REPORT

Test Title: Freelancer and Project Management Platform Functional Testing

Introduction: The testing was conducted for the Freelancer and Project Management Platform to ensure that all functionalities, from user registration to project management, bidding, and communication, operate correctly and meet the specified requirements. This test report summarizes the results of the conducted tests, highlighting the functionality, performance, and reliability of the platform.

Test Objective: The objective of the testing was to verify that the platform's features work as intended, including user registration and profile management, project posting and bidding, skill

matching, communication tools, payment and contract management, and the admin dashboard. The tests aimed to identify any defects or issues and ensure that the system meets the requirements for usability and functionality.

Test Scope: The testing encompassed various modules of the Freelancer and Project Management Platform, addressing User Registration and Profile Management, Project Posting and Bidding, Skill Matching and Search Engine, Review and Rating System, Communication and Collaboration Tools, Payment and Contract Management, Admin Dashboard, and Notification and Alert System. It does verify that each module operates correctly, delivers accurate results, and provides a cohesive user experience across all functionalities.

Test Environment:

- **Front-end:** Developed using HTML, CSS, and Bootstrap
- **Back-end:** Powered by Python Flask
- **Database:** MySQL
- **Server:** Local development environment with Wampserver
- **Tools:** Browser for UI testing, Python scripts for back-end testing, MySQL Workbench for database verification

Test Conclusion: The platform has been tested thoroughly across all its functionalities, and it meets the specified requirements. The system's core features, including user management, project posting, bidding, communication, and payment processing, operate efficiently without significant issues. The successful execution of tests indicates that the platform is ready for deployment and use by freelancers, businesses, and administrators. All identified issues were resolved during testing, ensuring a reliable and effective system.

8.3 BUG REPORT

A bug report is a documented artifact that we use to communicate a problem or a potential problem with the software. It contains all the information necessary to convey the issue, its seriousness, evidence, steps to reproduce, etc.

BUG ID	TEST PLAN ID	TEST CASE ID	TEST MODULE	BUG DESCRIPTION	BUG STATUS
BR01	TP01	TC01	Login Module	User can log in with an invalid email format	Open
BR02	TP02	TC05	Project Bidding	Bid submission allows empty proposal message	Open
BR03	TP03	TC09	Payment Module	Payment gateway fails to redirect after successful payment	In Progress
BR04	TP04	TC13	Freelancer Profile	Profile image upload fails for JPG files above 2MB	Open
BR05	TP05	TC16	Feedback System	Feedback is not saved when rating is not selected	Fixed
BR06	TP06	TC18	Report Generation	Reports show incorrect total earnings due to currency conversion mismatch	Open
BR07	TP07	TC21	Registration Module	System accepts weak passwords (e.g., “123456”) without validation	In Progress
BR08	TP08	TC25	Admin Dashboard	Admin cannot deactivate a freelancer account after first approval	Open

CHAPTER 9

CONCLUSION

CHAPTER 9
CONCLUSION

In conclusion, Web Based Project Bidding System for an IT Organization effectively addresses the complexities of freelance recruitment by offering a comprehensive and user-friendly platform that connects businesses with skilled developers globally. The platform enhances the hiring process through advanced features such as detailed coder profiles, a sophisticated skill-matching algorithm, and a transparent review system. These features collectively streamline project posting, proposal management, and candidate selection, making it easier for companies to find and hire the right talent. Moreover, Web Based Project Bidding System ensures a secure and efficient transaction environment by incorporating robust communication tools, payment processing systems, and contract management features. This facilitates seamless interactions between businesses and freelancers, ensuring clarity and security throughout the engagement process. The successful implementation and positive test results of Web Based Project Bidding System, which include thorough functionality testing and user acceptance evaluations, affirm its effectiveness and reliability as a freelance recruitment solution. The platform proves to be a valuable asset for businesses seeking specialized development skills while providing freelancers with a global marketplace to showcase their expertise and secure project opportunities. Overall, Web Based Project Bidding System represents a significant advancement in freelance hiring, offering a modern, secure, and efficient approach to connecting talent with opportunity.

CHAPTER 10

FUTURE ENHANCEMENT

CHAPTER 10

FUTURE ENHANCEMENT

Future enhancements for Web Based Project Bidding System aim to further improve its functionality, user experience, and scalability:

1. **Advanced AI-Driven Skill Matching:** Integrating machine learning algorithms to enhance the skill-matching process can provide more accurate and personalized recommendations, improving the relevance of freelancer matches to project requirements.
2. **Blockchain Integration for Enhanced Security:** Implementing blockchain technology for transaction verification and smart contract management can increase security and transparency, ensuring tamper-proof contracts and payment processing.
3. **Augmented Reality (AR) for Portfolio Presentation:** Allowing freelancers to present their portfolios through AR can offer a more interactive and engaging way for businesses to evaluate their work, enhancing the decision-making process.
4. **Real-Time Collaboration Tools:** Developing more sophisticated real-time collaboration features, such as integrated video conferencing and collaborative coding environments, can improve communication and project management between freelancers and businesses.
5. **Expanded Payment Options:** Adding support for a broader range of payment methods, including cryptocurrencies, can cater to a more diverse user base and provide additional flexibility in financial transactions.
6. **Enhanced Analytics and Reporting:** Providing more advanced analytics tools for both freelancers and businesses can offer deeper insights into project performance, user engagement, and financial metrics, facilitating better decision-making.
7. **Mobile Application Development:** Creating a dedicated mobile app for web based project bidding system for an IT organization can enhance accessibility and usability, allowing users to manage projects and communications on the go.
8. **Internationalization and Localization:** Expanding language support and localizing the platform for various regions can make web based project bidding system for an IT organization more accessible to a global audience and accommodate diverse user needs.

APPENDIX I

SAMPLE SOURCE CODE

APPENDIX I

SAMPLE SOURCE CODE

```
from flask import Flask, render_template, request, Response, session, flash, redirect,
url_for
from werkzeug.utils import secure_filename
import os
import win32com.client
import pythoncom
import cv2
import string
import random
import numpy as np
import time
import datetime
import mysql.connector
import math
from flask_socketio import SocketIO, emit, join_room, leave_room
from engineio.payload import Payload
Payload.max_decode_packets = 200
from werkzeug.utils import secure_filename
from flask import request as flask_request


app = Flask(__name__)
app.secret_key = "abcdef"

app.config['UPLOAD_FOLDER'] = 'static/resume'  # Folder to store songs

socketio = SocketIO(app, async_mode='eventlet')  # Explicitly set async_mode


_users_in_room = {} # stores room wise user list
_room_of_sid = {} # stores room joined by an used
_name_of_sid = {} # stores display name of users


app.config['UPLOAD_FOLDER1'] = 'static/profile'

app.config['UPLOAD_FOLDER3'] = 'static/poster'# Folder to store songs
```

```
def allowed_file(filename):
    ALLOWED_EXTENSIONS = {'pdf', 'docx', 'jpg', 'png'}
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS


mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    charset="utf8",
    database="coderbox_bid"
)

@app.route('/rate_seeker')
def rate_seeker():
    request_id = request.args.get('request_id')
    seeker_username = request.args.get('seeker_username')
    return render_template('rate_seeker.html', request_id=request_id,
seeker_username=seeker_username)


@app.route('/profile_view')
def profile_view():
    username = session.get('username')

    cursor = mydb.cursor()
    cursor.execute("SELECT * FROM profile WHERE username=%s", (username,))
    data = cursor.fetchone()

    # Get average rating
    cursor.execute("SELECT AVG(rating) FROM ratings WHERE seeker_username =
%s", (username,))
    avg_rating = cursor.fetchone()[0]

    # Get all reviews
    cursor.execute("SELECT recruiter_username, rating, review, date_submitted FROM
ratings WHERE seeker_username = %s", (username,))
    reviews = cursor.fetchall()

    cursor.close()
```

```
    return render_template('profile_view.html', data=data, avg_rating=avg_rating,
reviews=reviews)


@app.route('/submit_rating', methods=['POST'])
def submit_rating():
    recruiter_username = session.get('username')
    request_id = request.args.get('request_id')
    seeker_username = request.args.get('seeker_username')
    rating = int(request.form['rating'])
    review = request.form['review']
    date_submitted = datetime.datetime.now().date()

    cursor = mydb.cursor()
    sql = "INSERT INTO ratings (request_id, recruiter_username, seeker_username,
rating, review, date_submitted) VALUES (%s, %s, %s, %s, %s, %s)"
    val = (request_id, recruiter_username, seeker_username, rating, review,
date_submitted)
    cursor.execute(sql, val)
    mydb.commit()
    cursor.close()

    flash("Rating submitted successfully.", "success")
    return redirect(url_for('request_list'))



@app.route('/', methods=['POST', 'GET'])
def index():

    return render_template('index.html')

@app.route('/admin', methods=['GET', 'POST'])
def admin():
    msg = ""  # default empty message

    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        cursor = mydb.cursor()
```

```
        cursor.execute('SELECT * FROM admin WHERE username = %s AND password =
%s', (username, password))
        account = cursor.fetchone()

        if account:
            session['username'] = username
            session['user_type'] = 'admin'
            msg="success"
         # Assuming 'pro' is your dashboard or home
        else:
            msg = "fail"  # pass failure message

    return render_template('admin.html', msg=msg)

@app.route('/login',methods=['POST','GET'])
def login():


    msg=""
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        cursor = mydb.cursor()
        cursor.execute('SELECT * FROM seeker WHERE username = %s AND password
= %s', (username, password))
        account = cursor.fetchone()

        if account:
            session['username'] = username
            session['user_type'] = 'user'
            msg="success"

        else:
            msg="fail"



    return render_template('login.html',msg=msg)


@app.route('/login1',methods=['POST','GET'])
def login1():
```

```
    msg=""
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        cursor = mydb.cursor()
        cursor.execute('SELECT * FROM recruiter WHERE username = %s AND
password = %s AND status=1', (username, password))
        account = cursor.fetchone()

        if account:
            session['username'] = username
            session['user_type'] = 'user'
            msg="success"

        else:
            msg="fail"



    return render_template('login1.html',msg=msg)


@app.route('/register',methods=['POST','GET'])
def register():

    msg=""
    if request.method=='POST':

        name=request.form['name']
        email=request.form['email']
        mobile=request.form['mobile']
        address=request.form['address']
        username=request.form['username']
        password=request.form['password']


        now = datetime.datetime.now()
        date_join=now.strftime("%Y-%m-%d")
        mycursor = mydb.cursor()

        mycursor.execute("SELECT count(*) FROM seeker where
username=%s",(username, ))
```

```
        cnt = mycursor.fetchone()[0]
        if cnt==0:
            mycursor.execute("SELECT max(id)+1 FROM seeker")
            maxid = mycursor.fetchone()[0]
            if maxid is None:
                maxid=1
            sql = "INSERT INTO seeker(id, name, email, address, mobile, username,
password, date_join) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
            val = (maxid, name, email, address, mobile, username, password, date_join)
            mycursor.execute(sql, val)
            mydb.commit()

            msg="success"

            mycursor.close()
        else:
            msg="fail"

    return render_template('register.html', msg=msg)


@app.route('/submit_payment', methods=['POST'])
def submit_payment():
    request_id = request.form['request_id']
    payment_type = request.form['payment_type']
    amount = request.form['amount']
    card_number = request.form['card']  # Masking recommended in production

    try:
        cursor = mydb.cursor()
        cursor.execute("SELECT max(id)+1 FROM payments")
        maxid = cursor.fetchone()[0]
        if maxid is None:
            maxid=1
        sql = """
            INSERT INTO payments (id, request_id, payment_type, amount, card_number)
            VALUES (%s, %s, %s, %s, %s)
        """
        cursor.execute(sql, (maxid, request_id, payment_type, amount, card_number))
        mydb.commit()
        cursor.close()
        flash("Payment submitted successfully!", "success")
    except Exception as e:
```

```
        flash(f"Payment failed: {str(e)}", "danger")

    return redirect(url_for('request_list'))



@app.route('/register1',methods=['POST','GET'])
def register1():


    msg=""
    st = ""
    mess = ""
    mobile = ""
    username=""
    password=""
    name=""
    if request.method=='POST':

        name=request.form['name']
        company=request.form['company']
        company_type=request.form['company_type']
        email=request.form['email']
        mobile=request.form['mobile']
        location=request.form['location']
        username=request.form['username']
        password=request.form['password']


        now = datetime.datetime.now()
        date_join=now.strftime("%Y-%m-%d")
        mycursor = mydb.cursor()

        mycursor.execute("SELECT count(*) FROM recruiter where
username=%s",(username, ))
        cnt = mycursor.fetchone()[0]
        if cnt==0:
            mycursor.execute("SELECT max(id)+1 FROM recruiter")
            maxid = mycursor.fetchone()[0]
            if maxid is None:
                maxid=1
```

```
            sql = "INSERT INTO recruiter(id, name, company, company_type, email,
location, mobile, username, password, date_join) VALUES (%s, %s, %s, %s, %s, %s,
%s, %s, %s, %s)"
            val = (maxid, name, company, company_type, email, location, mobile, username,
password, date_join)
            mycursor.execute(sql, val)
            mydb.commit()

            msg="success"
            mycursor.close()

        else:
            msg="fail"

    return render_template('register1.html', msg=msg, st=st, mess=mess)



@app.route('/post', methods=['POST', 'GET'])
def post():
    msg = ""
    username = session.get('username')
    cursor = mydb.cursor()
    cursor.execute("Select * from recruiter where username=%s", (username,))
    dat = cursor.fetchone()
    cursor.close()

    name = dat[1]
    company = dat[2]
    company_type = dat[3]
    email = dat[4]
    mobile = dat[6]
    location = dat[5]

    if request.method == 'POST':
        # Even if called job_title etc., they represent project details
        job_title = request.form['job_title']
        job_type = request.form['job_type']
        salary = request.form['salary']
        description = request.form['description']
        file = request.files['job_poster']

        if file and allowed_file(file.filename):
```

```
        random_filename = ''.join(random.choices(string.ascii_letters + string.digits,
k=10)) + ".jpg"
        filepath = os.path.join(app.config['UPLOAD_FOLDER3'], random_filename)
        file.save(filepath)

        date_join = datetime.datetime.now().strftime("%d-%m-%Y")

        mycursor = mydb.cursor()
        mycursor.execute("SELECT max(id)+1 FROM posts")
        maxid = mycursor.fetchone()[0] or 1

        sql = """INSERT INTO posts
           (id, name, company, company_type, email, location, mobile,
           job_title, job_type, salary, descrip, filename, date_join, username)
           VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""
        val = (maxid, name, company, company_type, email, location, mobile,
              job_title, job_type, salary, description, random_filename, date_join,
username)
        mycursor.execute(sql, val)
        mydb.commit()
        mycursor.close()

        return redirect(url_for('my_posts'))
      else:
        msg = "fail"

   return render_template('post.html', msg=msg)


@app.route('/profile', methods=['POST', 'GET'])
def profile():
   msg = ""
   username = session.get('username')
   cursor = mydb.cursor()
   cursor.execute("SELECT * FROM seeker WHERE username=%s", (username,))
   dat = cursor.fetchone()
   cursor.close()

   name = dat[1]
   email = dat[3]
   mobile = dat[2]
   location = dat[4]
```

```
    if request.method == 'POST':
        job_category = request.form['job_category']
        skills = request.form['skills']
        experience = request.form['experience']

        file = request.files['resume']
        if file and allowed_file(file.filename):
            # Generate a random filename
            random_filename = ''.join(random.choices(string.ascii_letters + string.digits,
k=10)) + ".pdf"
            filepath = os.path.join(app.config['UPLOAD_FOLDER'], random_filename)
            file.save(filepath)

            file1 = request.files['profile_picture']
            if file1 and allowed_file(file1.filename):
                random_filename1 = ''.join(random.choices(string.ascii_letters + string.digits,
k=10)) + ".jpg"
                filepath1 = os.path.join(app.config['UPLOAD_FOLDER1'],
random_filename1)
                file1.save(filepath1)

                now = datetime.datetime.now()
                date_join = now.strftime("%d-%m-%Y")

                mycursor = mydb.cursor()
                mycursor.execute("SELECT max(id)+1 FROM profile")
                maxid = mycursor.fetchone()[0]
                if maxid is None:
                    maxid=1


                sql = """INSERT INTO profile(id, name, email, location, mobile, job_category,
skills, experience, resume, profile, date_join, username)
                        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""
                val = (maxid, name, email, location, mobile, job_category, skills, experience,
random_filename, random_filename1, date_join, username)


                mycursor.execute(sql, val)
                mydb.commit()
                msg = "success"
                mycursor.close()
                return redirect(url_for('profile_view'))
```

```
            else:
                msg = "Profile picture file type not allowed."

    return render_template('profile.html', msg=msg)



@app.route('/my_posts', methods=['POST', 'GET'])
def my_posts():

    username=session.get('username')

    cursor=mydb.cursor()
    cursor.execute("Select * from posts where username=%s", (username, ))
    data=cursor.fetchall()
    cursor.close()

    return render_template('my_posts.html', data=data)


##########################################################################
#############################################################



import pytesseract
from PIL import Image
import cv2
import os
import re

pytesseract.pytesseract.tesseract_cmd = 'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'

# Predefined job titles to compare
JOB_TITLES = ["UI/UX Designer", "Video Editor", "Digital Marketing", "Graphic
Designer", "Content Creator"]

def extract_full_text(image_path):
    """Extracts full text from an image using OCR."""
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Improve OCR accuracy
```

```
    processed = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)[1]

    # Extract text
    text = pytesseract.image_to_string(processed, config='--psm 6 --oem 3')
    return text.strip()

def match_job_titles(extracted_text):
    """Matches extracted text with predefined job titles."""
    matched_titles = []
    for job_title in JOB_TITLES:
        if job_title.lower() in extracted_text.lower():  # Case-insensitive partial match
            matched_titles.append(job_title)

    return matched_titles

@app.route('/view_profiles', methods=['GET'])
def view_profiles():
    try:
        poster_path = "D:/coderbox/static/poster/PbXu7dEQDq.jpg"

        # Extract full text from image
        extracted_text = extract_full_text(poster_path)
        print(f"Extracted Text: {extracted_text}")  # Debugging Output
        print(f"Extracted Text:\n{extracted_text}")

        # Match job titles
        matched_job_titles = match_job_titles(extracted_text)
        print(f"Matched Job Titles: {matched_job_titles}")  # Debugging Output

        profiles = []
        if matched_job_titles:
            cursor = mydb.cursor()
            query = "SELECT * FROM profile WHERE skills LIKE %s"
            cursor.execute(query, ('%' + matched_job_titles[0] + '%',))
            profiles = cursor.fetchall()
            cursor.close()

        return render_template('view_profiles.html', job_titles=matched_job_titles,
profiles=profiles)

    except Exception as e:
        return f"Error processing post: {e}"
```

```
##########################################################################
##########################################################

from flask import jsonify



@app.route('/search_profiles', methods=['GET'])
def search_profiles():
    post_id = request.args.get('post_id')

    cursor = mydb.cursor(dictionary=True)
    cursor.execute("""
        SELECT bids.*, profile.name, profile.skills, profile.email, profile.mobile,
               profile.experience, profile.location, profile.profile, profile.resume
        FROM bids
        JOIN profile ON bids.seeker_username = profile.username
        WHERE bids.project_id = %s
    """, (post_id,))

    bids = cursor.fetchall()
    cursor.close()

    return render_template('search_profiles.html', bids=bids)
@app.route('/request1', methods=['POST', 'GET'])
def request1():
    msg = ""
    username = session.get('username')
    postID = request.args.get('post_id')
    pro_id = request.args.get('pro_id')  # from profile
    bid_id = request.args.get('bid_id')  # from bid

    # Get post/project details
    cursor = mydb.cursor()
    cursor.execute("SELECT * FROM posts WHERE id=%s", (postID,))
    dat = cursor.fetchone()
    cursor.close()

    postt_id = dat[0]
    post_name = dat[1]
```

```
    company = dat[2]
    company_type = dat[3]
    post_mobile = dat[4]
    post_email = dat[5]
    post_location = dat[6]
    job_title = dat[7]
    job_type = dat[8]
    salary = dat[9]
    descrip = dat[10]
    filename = dat[11]
    date_join = dat[12]
    post_username = dat[13]

    # Initialize these for both scenarios
    profile_id = name = email = mobile = location = job_category = skills = experience =
resume = profile = pro_username = ""

    # Scenario 1: From Profile
    if pro_id:
        cursor = mydb.cursor()
        cursor.execute("SELECT * FROM profile WHERE id=%s", (pro_id,))
        data = cursor.fetchone()
        cursor.close()

        profile_id = data[0]
        name = data[1]
        email = data[2]
        mobile = data[3]
        location = data[4]
        job_category = data[5]
        skills = data[6]
        experience = data[7]
        resume = data[8]
        profile = data[9]
        pro_username = data[11]

    # Scenario 2: From Bid
    elif bid_id:
        cursor = mydb.cursor(dictionary=True)
        cursor.execute("""
            SELECT b.*, p.id as profile_id, p.name, p.email, p.mobile, p.location,
                   p.job_category, p.skills, p.experience, p.resume, p.profile, p.username as
pro_username
```

```
            FROM bids b
            JOIN profile p ON b.seeker_username = p.username
            WHERE b.id = %s
        """, (bid_id,))
        bid = cursor.fetchone()
        cursor.close()

        profile_id = bid['profile_id']
        name = bid['name']
        email = bid['email']
        mobile = bid['mobile']
        location = bid['location']
        job_category = bid['job_category']
        skills = bid['skills']
        experience = bid['experience']
        resume = bid['resume']
        profile = bid['profile']
        pro_username = bid['pro_username']

    else:
        return "Invalid request. Missing profile or bid identifier.", 400

    now = datetime.datetime.now()
    date_join1 = now.strftime("%d-%m-%Y")

    mycursor = mydb.cursor()
    mycursor.execute("SELECT max(id)+1 FROM request")
    maxid = mycursor.fetchone()[0] or 1

    sql = """
        INSERT INTO request(id, postt_id, post_name, company, company_type,
post_email,
        post_location, post_mobile, job_title, job_type, salary, descrip, filename,
        date_join, post_username, profile_id, name, email, mobile, location,
        job_category, skills, experience, resume, profile, date_join1, pro_username, action)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """

    val = (
        maxid, postt_id, post_name, company, company_type, post_email, post_location,
        post_mobile, job_title, job_type, salary, descrip, filename, date_join,
        post_username, profile_id, name, email, mobile, location, job_category,
```

```
        skills, experience, resume, profile, date_join1, pro_username, '1'
    )

    mycursor.execute(sql, val)
    mydb.commit()
    mycursor.close()

    flash("Request sent successfully!", "success")
    return redirect(url_for('request_list'))



@app.route('/request_list', methods=['POST', 'GET'])
def request_list():
    username = session.get('username')

    cursor = mydb.cursor(dictionary=True)
    cursor.execute("SELECT * FROM request WHERE post_username=%s",
(username,))
    data1 = cursor.fetchall()

    # Fetch all payments
    cursor.execute("SELECT * FROM payments")
    payments_data = cursor.fetchall()
    cursor.close()

    # Group payments by request_id
    payments_by_request = {}
    for payment in payments_data:
        rid = payment['request_id']
        if rid not in payments_by_request:
            payments_by_request[rid] = []
        payments_by_request[rid].append(payment)

    # ðŸ”¥ Fix: make sure payments_by_request is passed into the template
    return render_template('request_list.html', data1=data1,
payments_by_request=payments_by_request)

@app.route('/update_request_status')
def update_request_status():
    request_id = request.args.get("aid")
    status_code = request.args.get("status")
```

```
    # Define allowed status codes
    valid_statuses = {
        '1': 'Pending',
        '2': 'Accepted',
        '3': 'Rejected',
        '4': 'In Process',
        '5': 'Completed'
    }

    if request_id is None or status_code not in valid_statuses:
        flash("Invalid request ID or status.", "danger")
        return redirect(url_for('request_list'))

    try:
        cursor = mydb.cursor()
        cursor.execute("UPDATE request SET action=%s WHERE id=%s", (status_code,
request_id))
        mydb.commit()
        cursor.close()
        flash(f"Request #{request_id} marked as {valid_statuses[status_code]}.", "success")
    except Exception as e:
        flash(f"Error updating request: {str(e)}", "danger")

    return redirect(url_for('user_request'))



@app.route('/pro', methods=['POST', 'GET'])
def pro():

    cursor = mydb.cursor()
    cursor.execute("SELECT * FROM recruiter")
    dat = cursor.fetchall()
    cursor.close()


    act=request.args.get("act")

    if act=="ok":
        rid=request.args.get("rid")
        cursor=mydb.cursor()
        cursor.execute(" UPDATE recruiter SET status=1 where id=%s", (rid, ))
        print("Data Updated Successfully")
```

```
        mydb.commit()
        return redirect(url_for('pro'))

    if act=="no":
        rid=request.args.get("rid")
        cursor=mydb.cursor()
        cursor.execute(" UPDATE recruiter SET status=2 where id=%s", (rid, ))
        print("Data Updated Successfully")
        mydb.commit()
        return redirect(url_for('pro'))


    return render_template('pro.html', data1=dat)



@app.route('/pro1', methods=['POST', 'GET'])
def pro1():

    cursor = mydb.cursor()
    cursor.execute("SELECT * FROM seeker")
    dat4 = cursor.fetchall()
    cursor.close()


    return render_template('pro1.html', data1=dat4)

from flask import Flask, request, jsonify



@app.route('/update_profile', methods=['POST'])
def update_profile():

    username=session.get('username')
    data = request.json
    # Process and update the database with new values
    # Assuming you have a database connection and `users` table
    try:

        cursor = mydb.cursor()
        sql = """UPDATE profile SET name=%s, email=%s, mobile=%s, location=%s,
                skills=%s, experience=%s WHERE username=%s"""
```

```
        cursor.execute(sql, (data['name'], data['email'], data['mobile'],
                             data['location'], data['skills'], data['experience'], username))
        mydb.commit()
        return jsonify({"status": "success"})
    except Exception as e:
        return jsonify({"status": "error", "message": str(e)})


@app.route('/delete_profile', methods=['POST'])
def delete_profile():
    if 'username' in session:
        username = session['username']
        try:
            cursor = mydb.cursor()
            cursor.execute("DELETE FROM profile WHERE username = %s", (username,))
            mydb.commit()
            session.clear()  # Log the user out
            return jsonify({"status": "success"})
        except Exception as e:
            return jsonify({"status": "error", "message": str(e)})
    return jsonify({"status": "error", "message": "Not logged in"})




@app.route('/update_request', methods=['GET'])
def update_request():
    action = request.args.get('action')
    request_id = request.args.get('request_id')

    # Define allowed actions
    valid_actions = {
        'accept': 2,
        'reject': 3,
        'process': 4,
        'complete': 5
    }

    if action not in valid_actions:
        return "Invalid action", 400
```

```
    # Update the request status
    cursor = mydb.cursor()
    cursor.execute("UPDATE request SET action=%s WHERE id=%s",
(valid_actions[action], request_id))
    mydb.commit()

    return redirect('/user_request')

@app.route('/pro2', methods=['GET'])
def pro2():
    job_type = request.args.get('job_type', '').strip()
    if job_type:
        cur = mydb.cursor(dictionary=True)
        cur.execute("SELECT * FROM request WHERE job_type LIKE %s", ('%' +
job_type + '%',))
    else:
        cur = mydb.cursor(dictionary=True)
        cur.execute("SELECT * FROM request")
    requests = cur.fetchall()
    return render_template('pro2.html', requests=requests)



@app.route('/user_request', methods=['GET'])
def user_request():
    username = session.get('username')
    if not username:
        return redirect('/login')

    # Fetch user requests as dictionary for easier access in template
    cursor = mydb.cursor(dictionary=True)
    cursor.execute("SELECT * FROM request WHERE pro_username=%s", (username,))
    data1 = cursor.fetchall()
    cursor.close()

    # Fetch all payments as dictionary
    cursor = mydb.cursor(dictionary=True)
    cursor.execute("SELECT * FROM payments")
    payments_data = cursor.fetchall()
    cursor.close()

    # Group payments by request_id
    payments_by_request = {}
```

```
    for payment in payments_data:
        rid = payment['request_id']
        if rid not in payments_by_request:
            payments_by_request[rid] = []
        payments_by_request[rid].append(payment)

    return render_template('user_request.html', data1=data1,
payments_by_request=payments_by_request)


@app.route("/call", methods=["GET", "POST"])
def call():

    aid=request.args.get("aid")
    if request.method == "POST":
        room_id = request.form['room_id']
        cursor = mydb.cursor()
        cursor.execute("update request set link=%s where id=%s",(room_id, aid))
        mydb.commit()

        return redirect(url_for("entry_checkpoint", room_id=room_id, aid=aid))

    return render_template("call.html")

@app.route("/room/<string:room_id>/")
def enter_room(room_id):
    act=request.args.get("act")


    if room_id not in session:
        return redirect(url_for("entry_checkpoint", room_id=room_id))

    return render_template("chatroom.html", room_id=room_id,
display_name=session[room_id]["name"], mute_audio=session[room_id]["mute_audio"],
mute_video=session[room_id]["mute_video"])

@app.route("/room/<string:room_id>/checkpoint/", methods=["GET", "POST"])
def entry_checkpoint(room_id):


    username=""

    if request.method == "POST":
```

```
        mute_audio = request.form['mute_audio']
        mute_video = request.form['mute_video']
        session[room_id] = {"name": username, "mute_audio":mute_audio,
"mute_video":mute_video}
        return redirect(url_for("enter_room", room_id=room_id))

    return render_template("chatroom_checkpoint.html", room_id=room_id)

@socketio.on("connect")
def on_connect():
    sid = request.sid
    print("New socket connected ", sid)


@socketio.on("join-room")
def on_join_room(data):
    sid = request.sid
    room_id = data["room_id"]
    display_name = session[room_id]["name"]

    # register sid to the room
    join_room(room_id)
    _room_of_sid[sid] = room_id
    _name_of_sid[sid] = display_name

    # broadcast to others in the room
    print("[{}] New member joined: {}<{}>".format(room_id, display_name, sid))
    emit("user-connect", {"sid": sid, "name": display_name}, broadcast=True,
include_self=False, room=room_id)

    # add to user list maintained on server
    if room_id not in _users_in_room:
        _users_in_room[room_id] = [sid]
        emit("user-list", {"my_id": sid}) # send own id only
    else:
        usrlist = {u_id:_name_of_sid[u_id] for u_id in _users_in_room[room_id]}
        emit("user-list", {"list": usrlist, "my_id": sid}) # send list of existing users to the
new member
        _users_in_room[room_id].append(sid) # add new member to user list maintained on
server

    print("\nusers: ", _users_in_room, "\n")
```

```
@socketio.on("disconnect")
def on_disconnect():
    sid = request.sid
    room_id = _room_of_sid[sid]
    display_name = _name_of_sid[sid]

    print("[{}] Member left: {}<{}>".format(room_id, display_name, sid))
    emit("user-disconnect", {"sid": sid}, broadcast=True, include_self=False,
room=room_id)

    _users_in_room[room_id].remove(sid)
    if len(_users_in_room[room_id]) == 0:
        _users_in_room.pop(room_id)

    _room_of_sid.pop(sid)
    _name_of_sid.pop(sid)

    print("\nusers: ", _users_in_room, "\n")


@socketio.on("data")
def on_data(data):
    sender_sid = data['sender_id']
    target_sid = data['target_id']
    if sender_sid != request.sid:
        print("[Not supposed to happen!] request.sid and sender_id don't match!!!")

    if data["type"] != "new-ice-candidate":
        print('{} message from {} to {}'.format(data["type"], sender_sid, target_sid))
    socketio.emit('data', data, room=target_sid)

##########################################################################
##################

@app.route('/all_projects', methods=['GET'])
def all_projects():
    cursor = mydb.cursor()
    cursor.execute("SELECT * FROM posts")
    projects = cursor.fetchall()
    cursor.close()
    return render_template('all_projects.html', projects=projects)
```

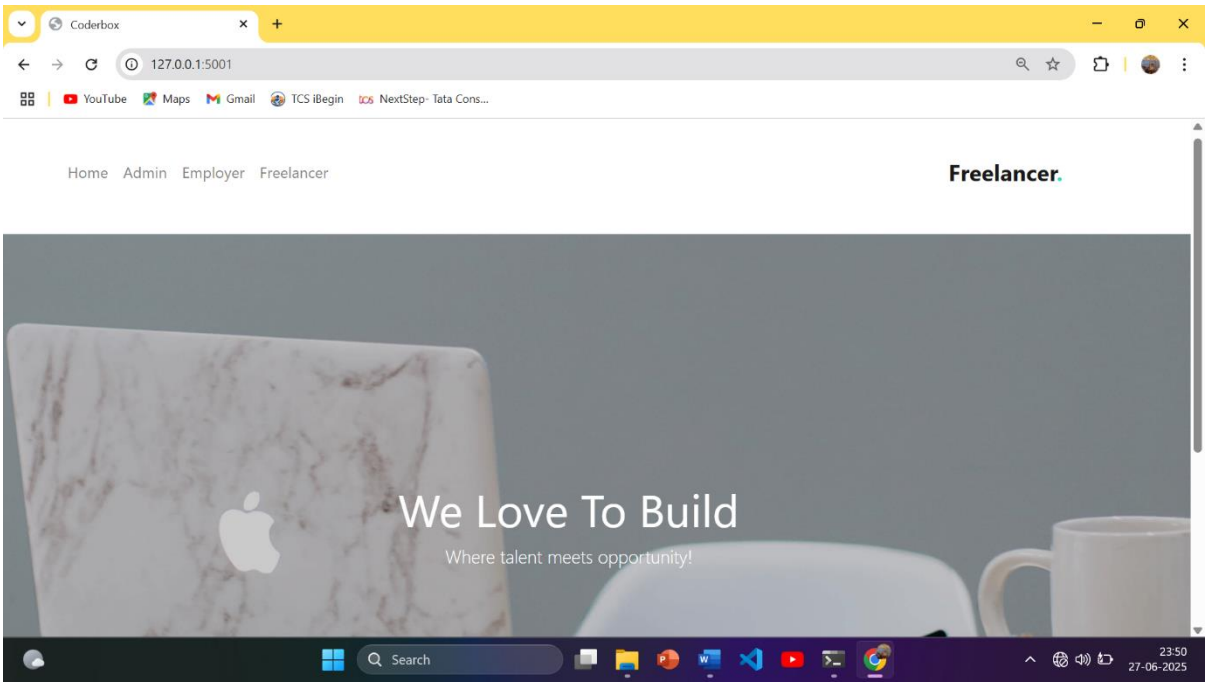
```
@app.route('/submit_bid', methods=['POST'])
def submit_bid():
    if 'username' not in session:
        return redirect(url_for('login'))

    seeker_username = session['username']
    project_id = request.form['project_id']
    bid_amount = request.form['bid_amount']
    timeline = request.form['timeline']
    message = request.form['message']
    date_submitted = datetime.datetime.now().date()

    cursor = mydb.cursor()

    cursor.execute("SELECT max(id)+1 FROM bids")
    maxid = cursor.fetchone()[0] or 1
    cursor.execute("""
        INSERT INTO bids (id, project_id, seeker_username, bid_amount, timeline,
message, date_submitted)
        VALUES (%s, %s, %s, %s, %s, %s, %s)
    """, (maxid, project_id, seeker_username, bid_amount, timeline, message,
date_submitted))
    mydb.commit()
    cursor.close()

    flash("Bid submitted successfully!", "success")
    return redirect(url_for('all_projects'))


@app.route('/logout')
def logout():

    session.clear()
    print("Logged out successfully", 'success')
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(port=5001)
```

APPENDIX II

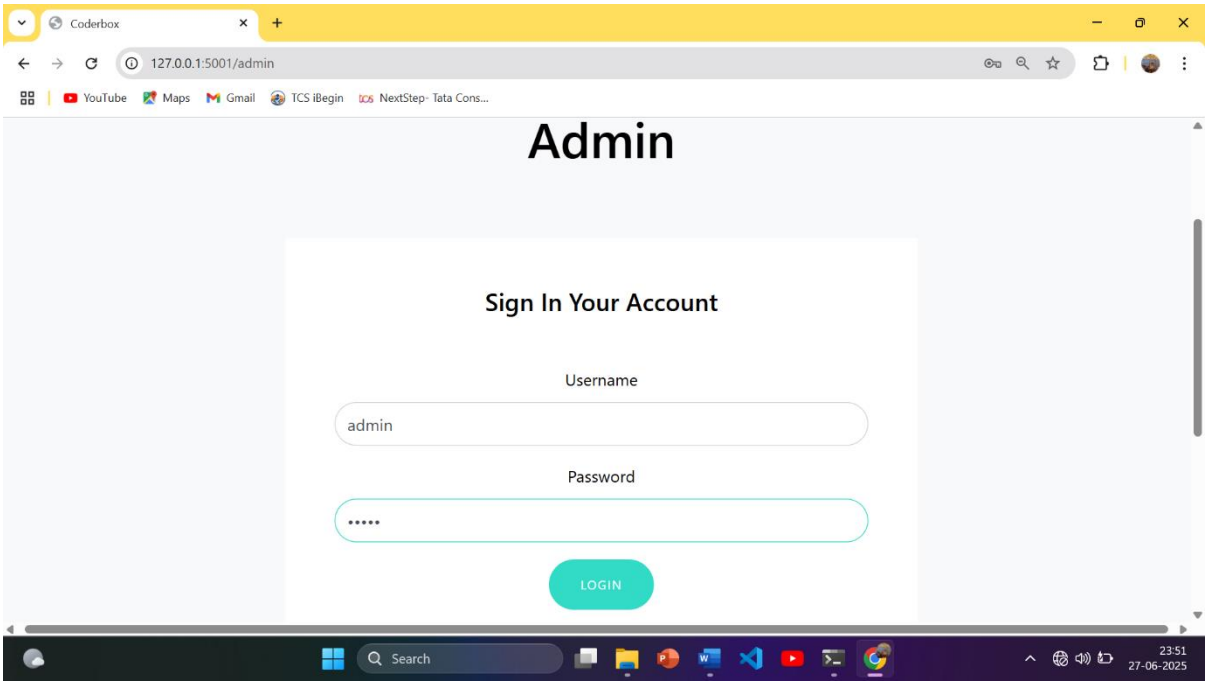
SAMPLE SCREENSHOT

APPENDIX II
SAMPLE SCREENSHOT

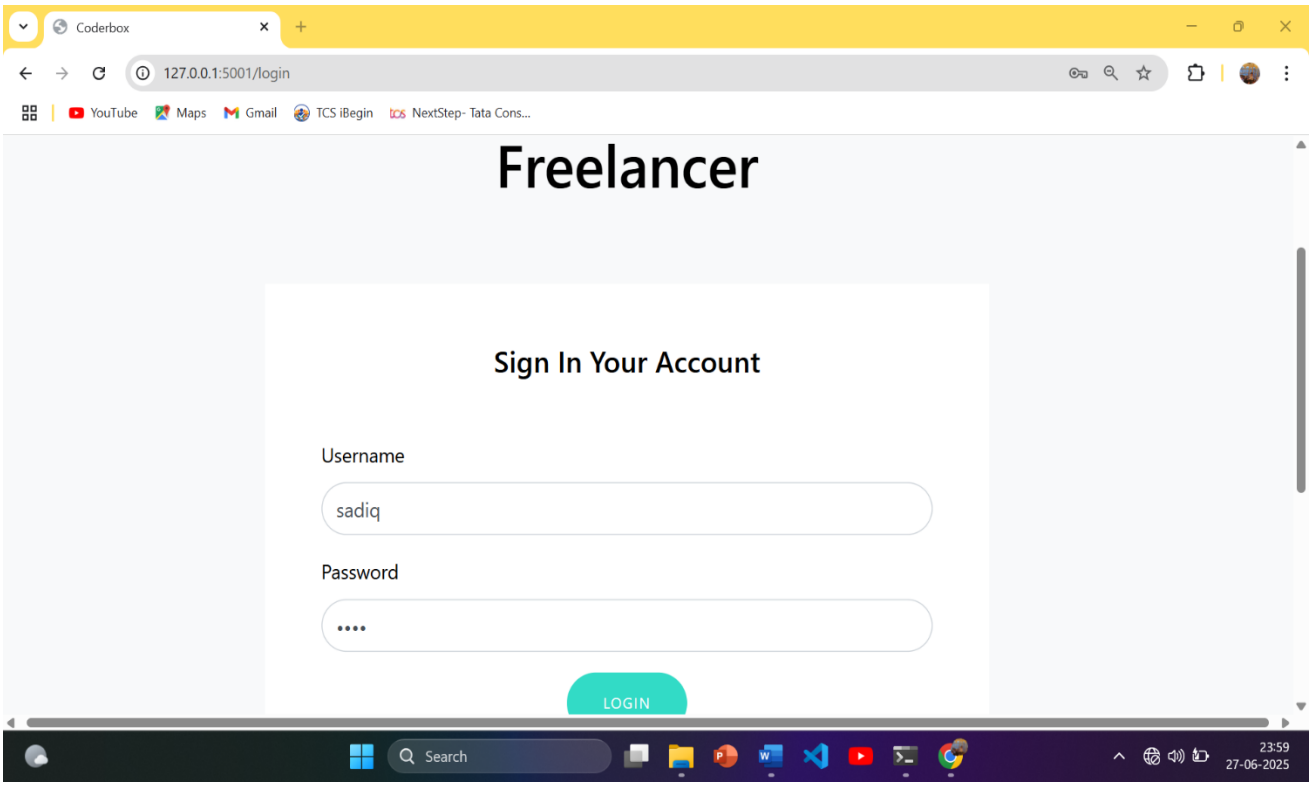
HOME PAGE



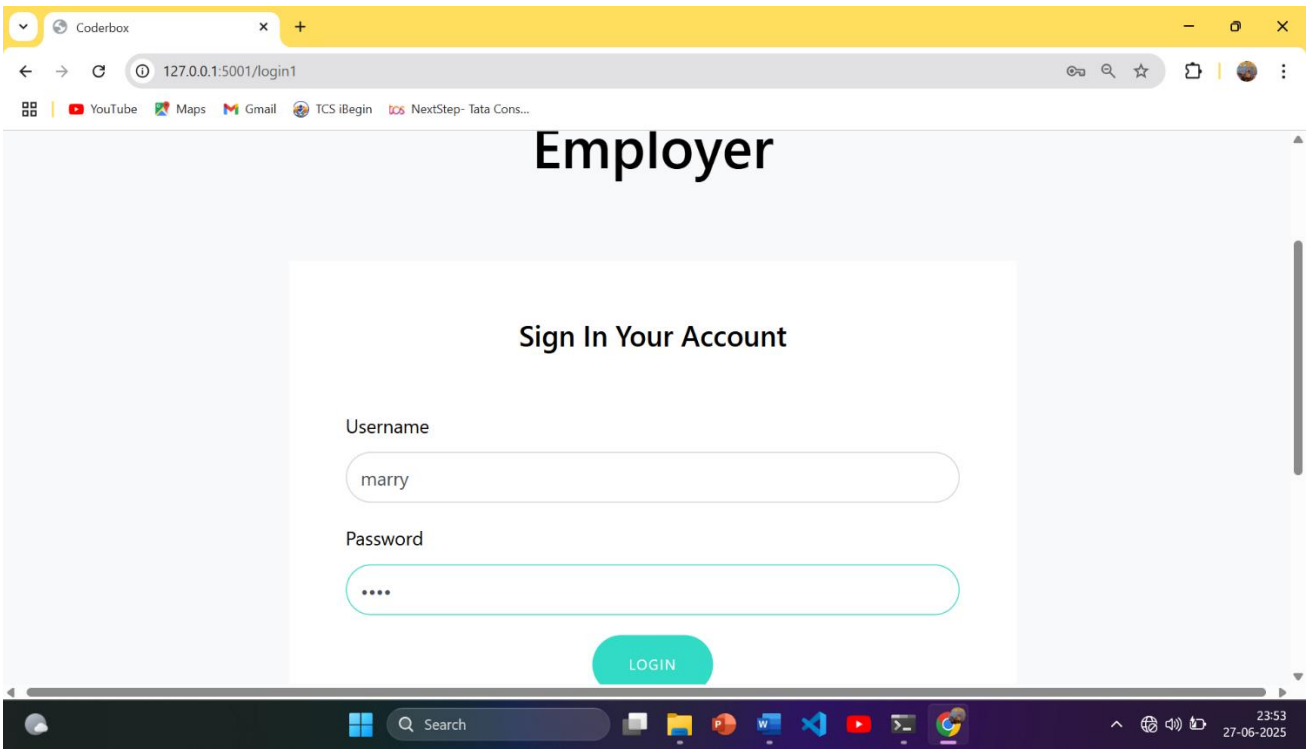
ADMIN LOGIN



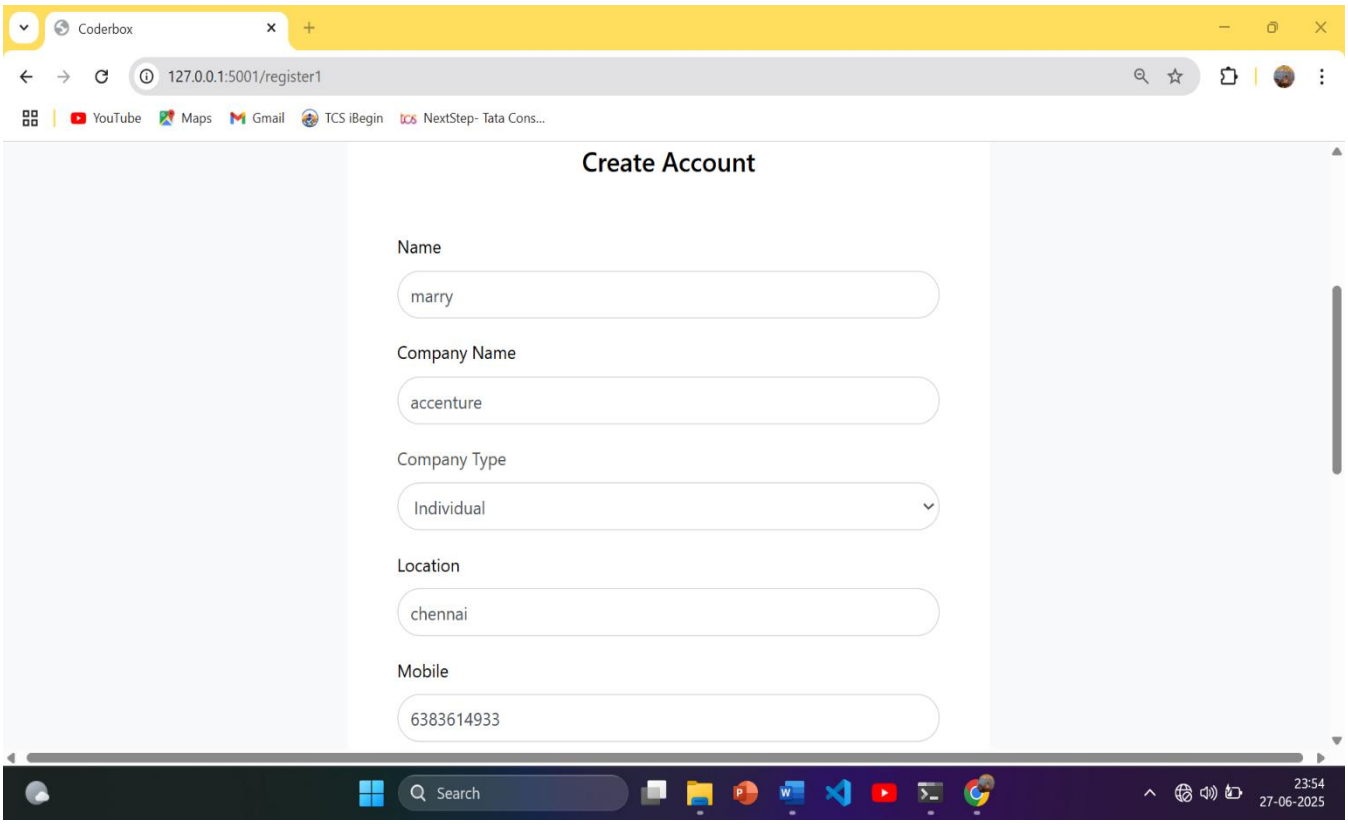
FREELANCER LOGIN



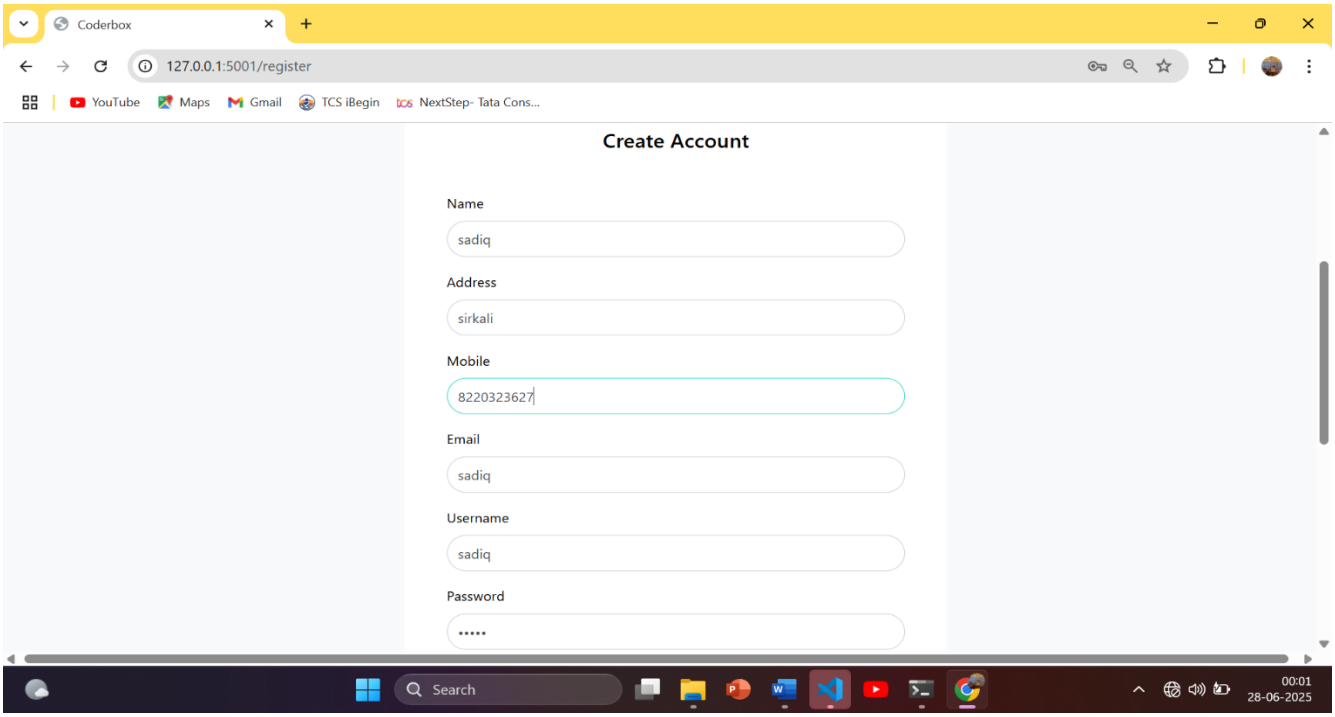
EMPLOYER LOGIN



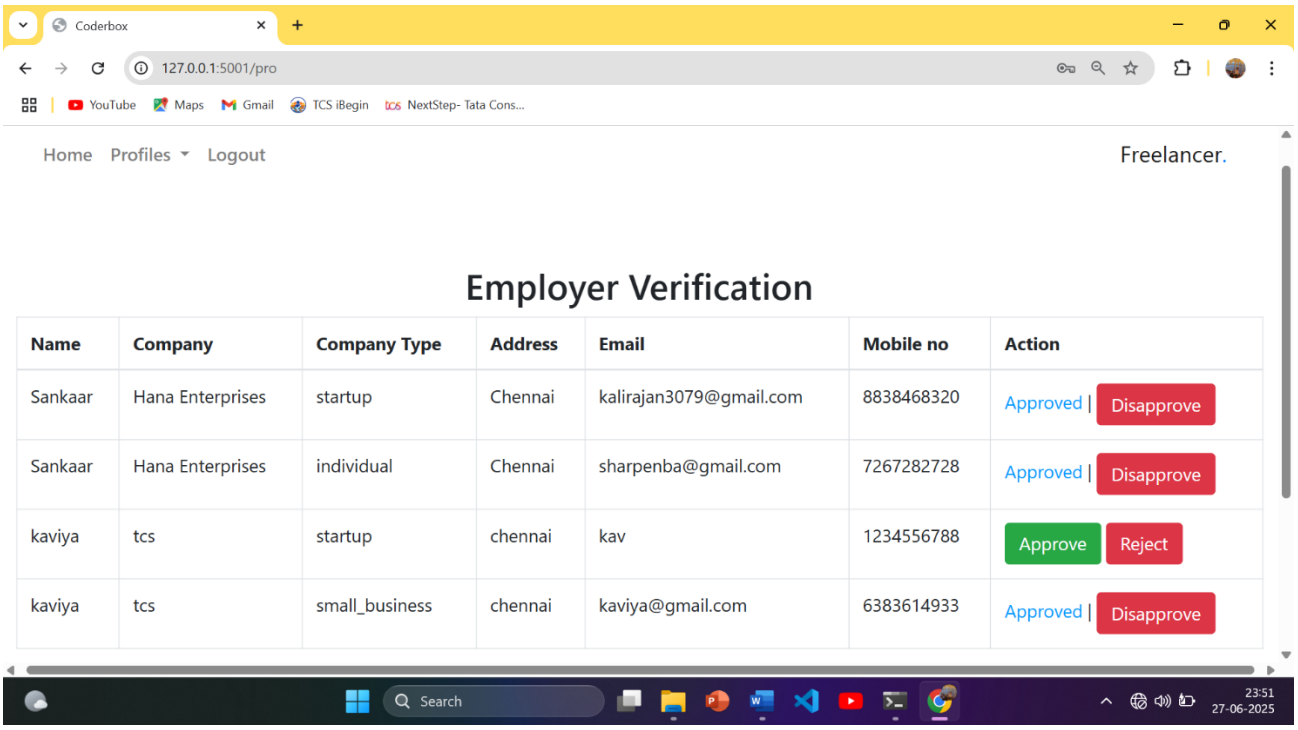
EMPLOYER SIGNUP



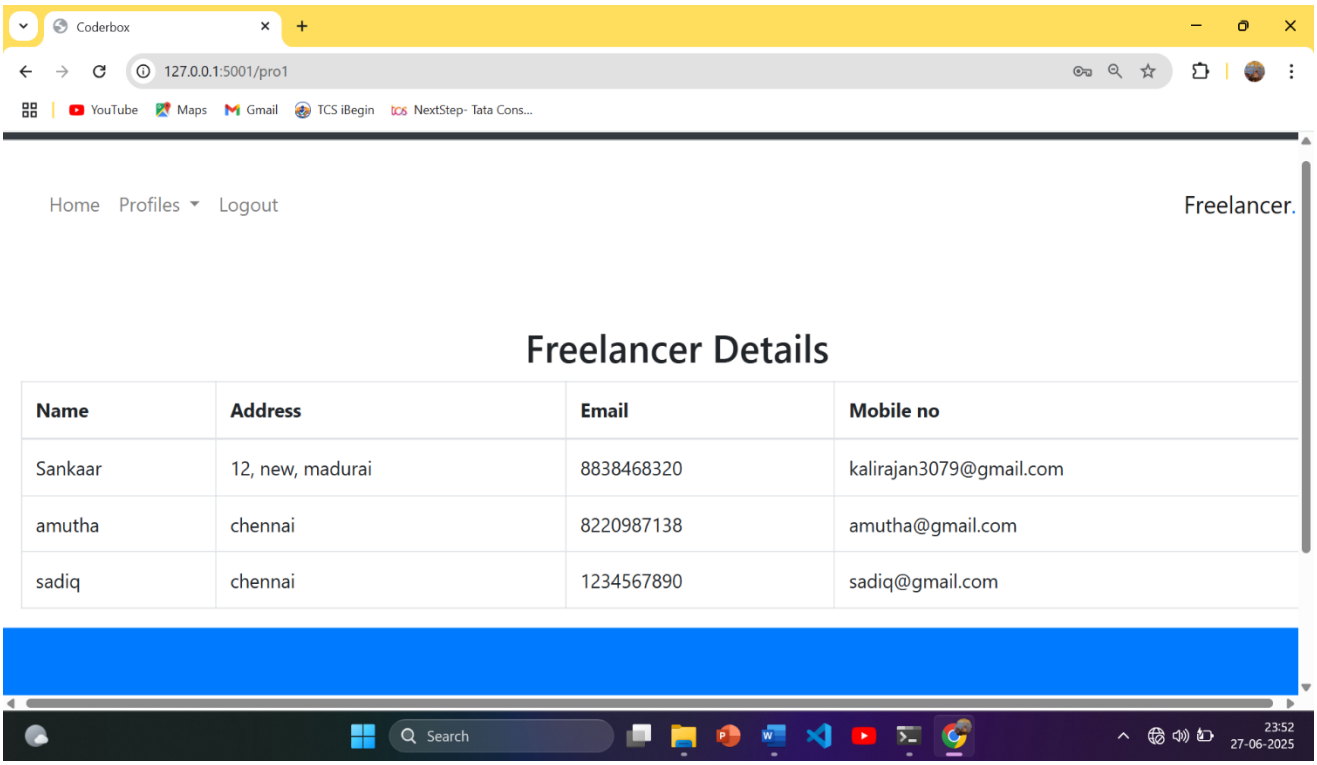
FREELANCER SIGNUP



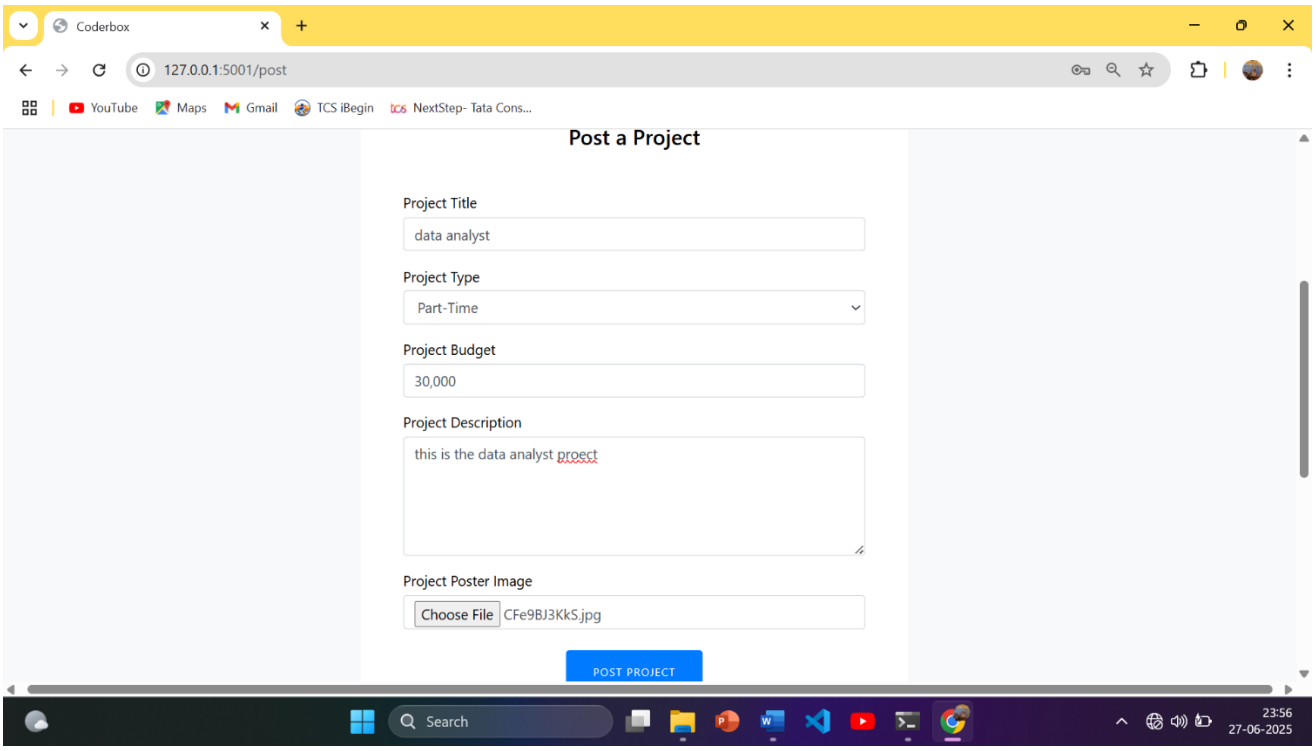
EMPLOYER VERIFICATION



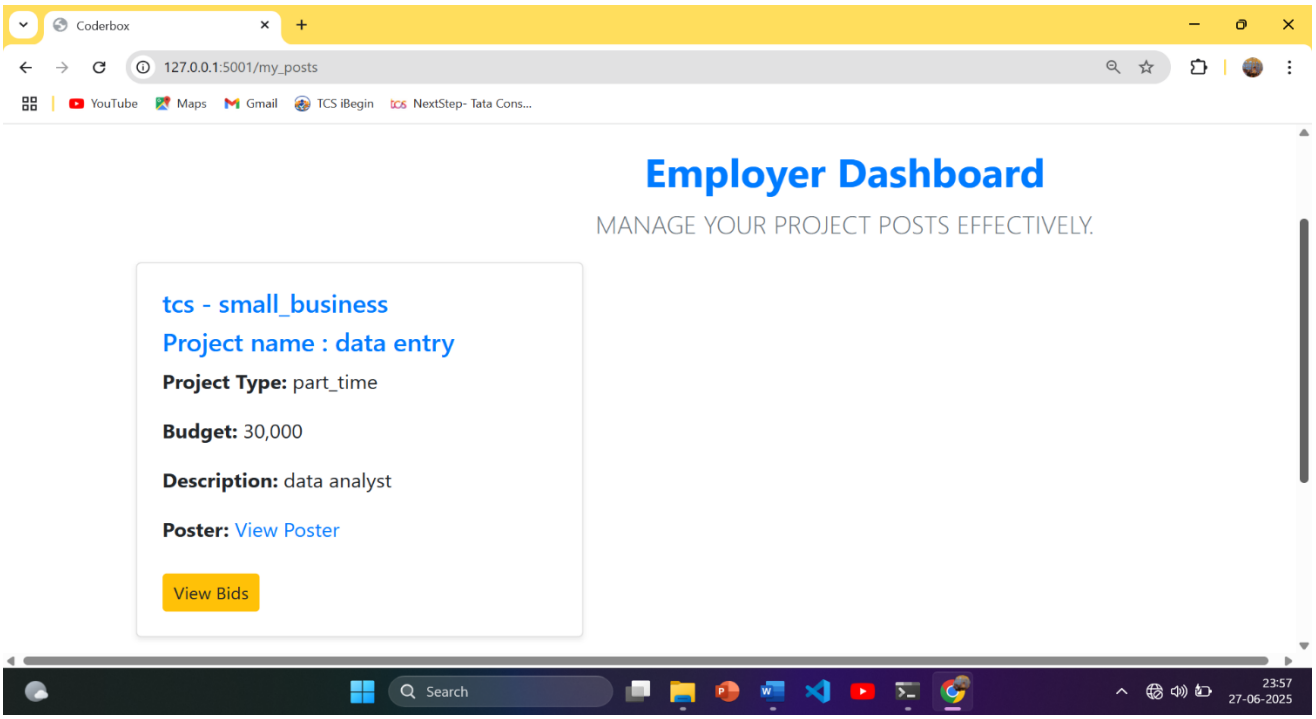
FREELANCER DETAILS



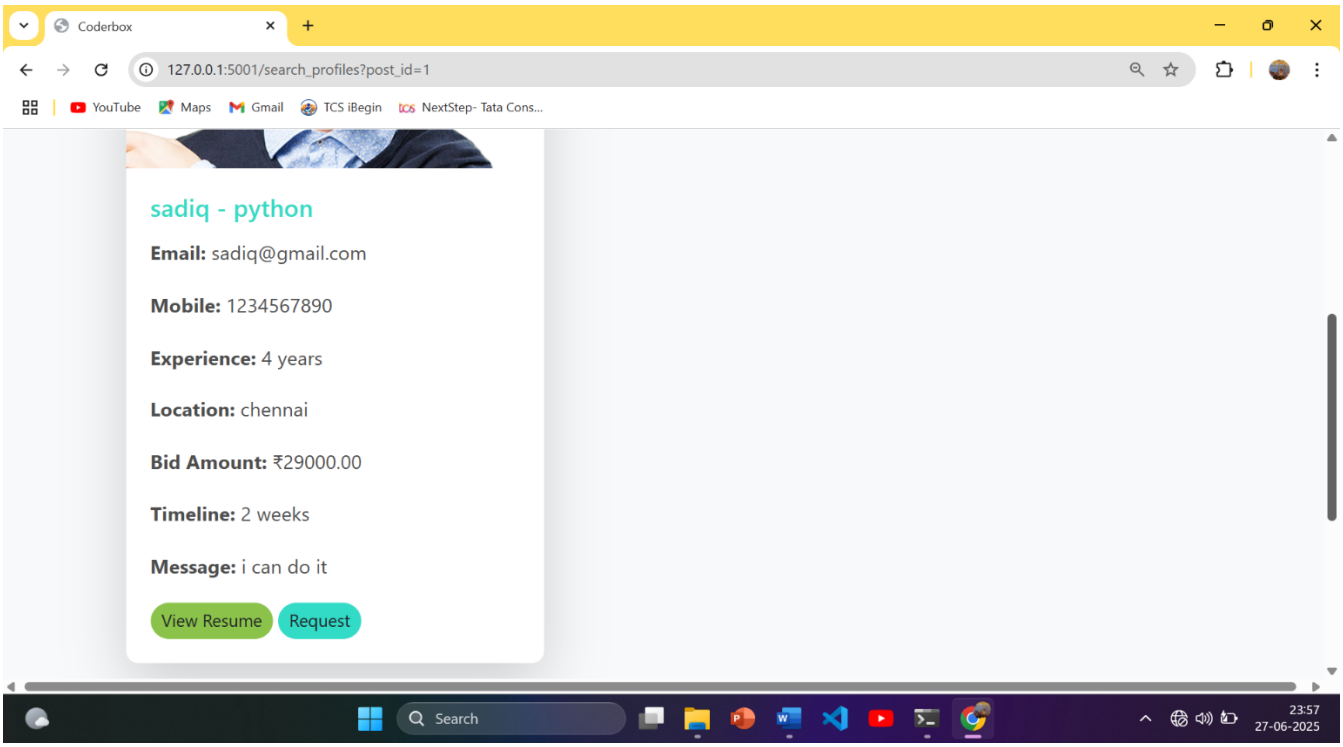
PROJECT DETAILS PAGE



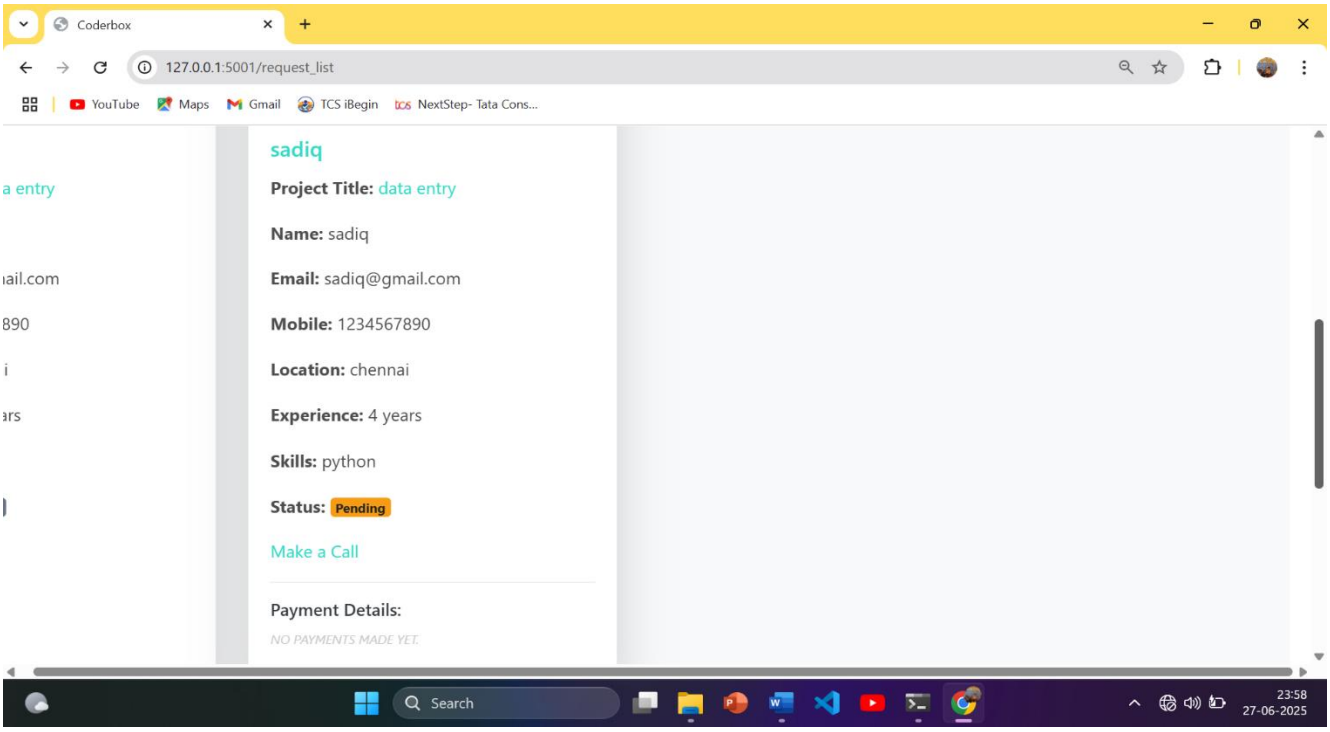
EMPLOYER DASHBOARD PAGE



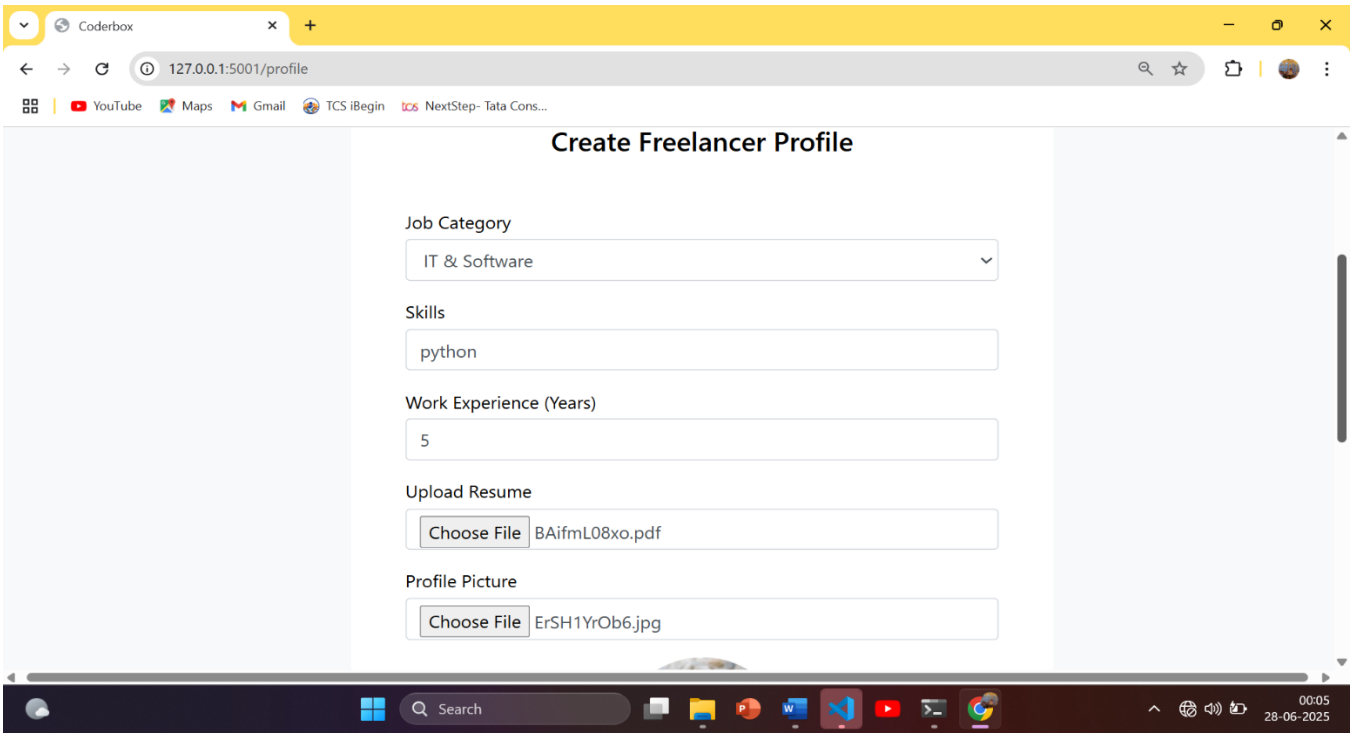
PROJECT BIDDING PAGE



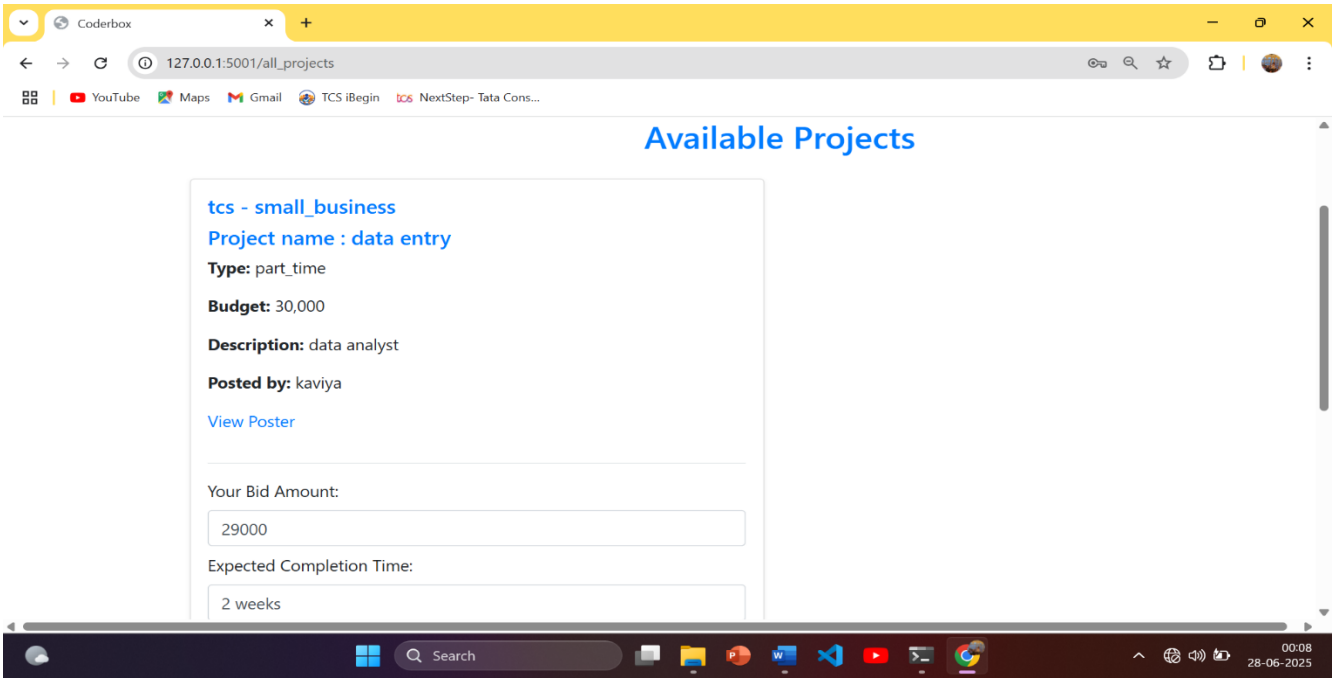
PROJECT REQUEST PAGE



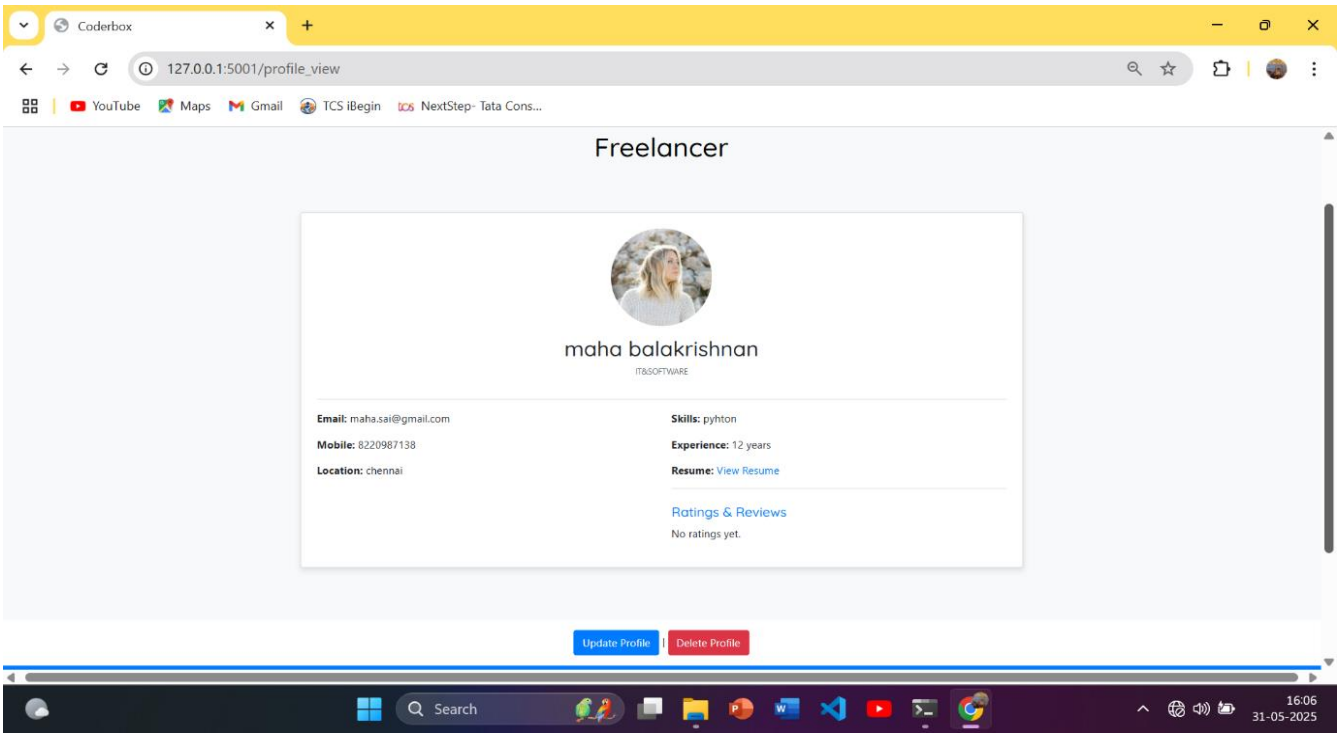
FREELANCER PORTFOLIO PAGE



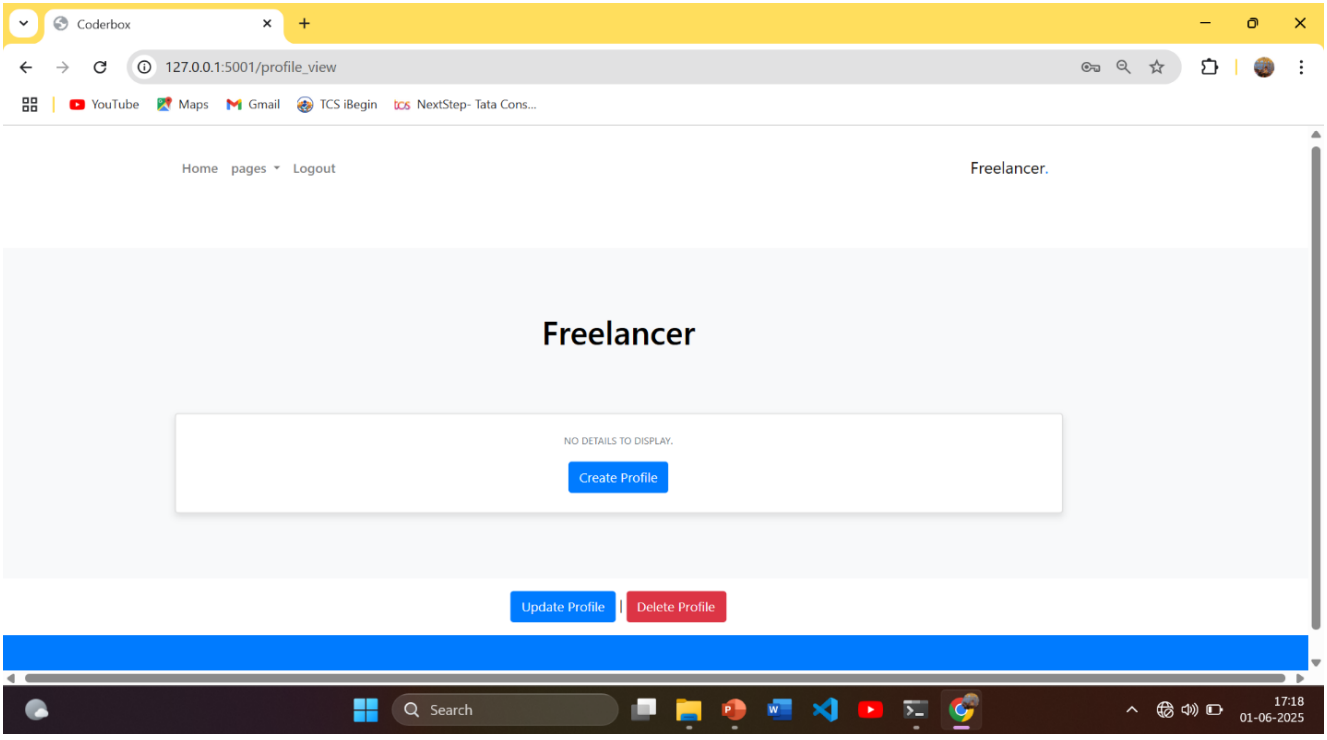
PROJECT AVAILABLE PAGE



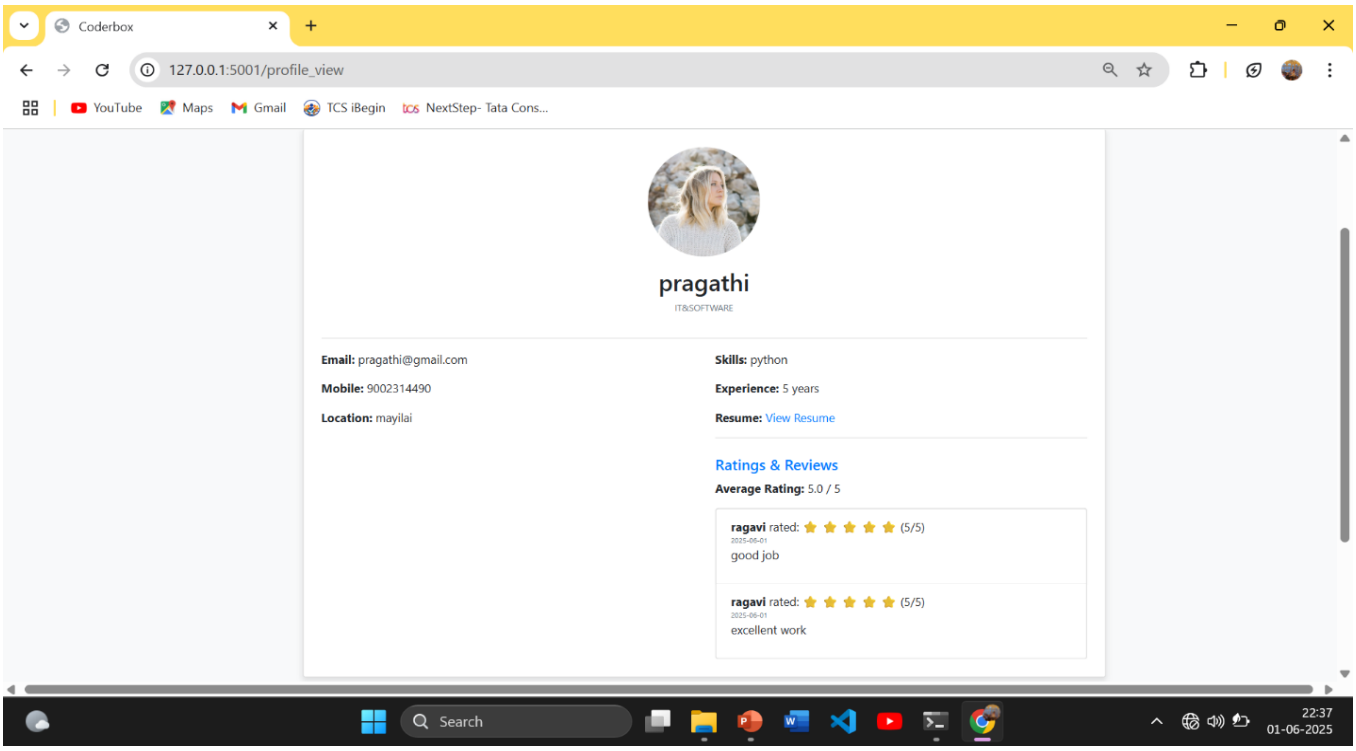
VIEW PEOFILE PAGE



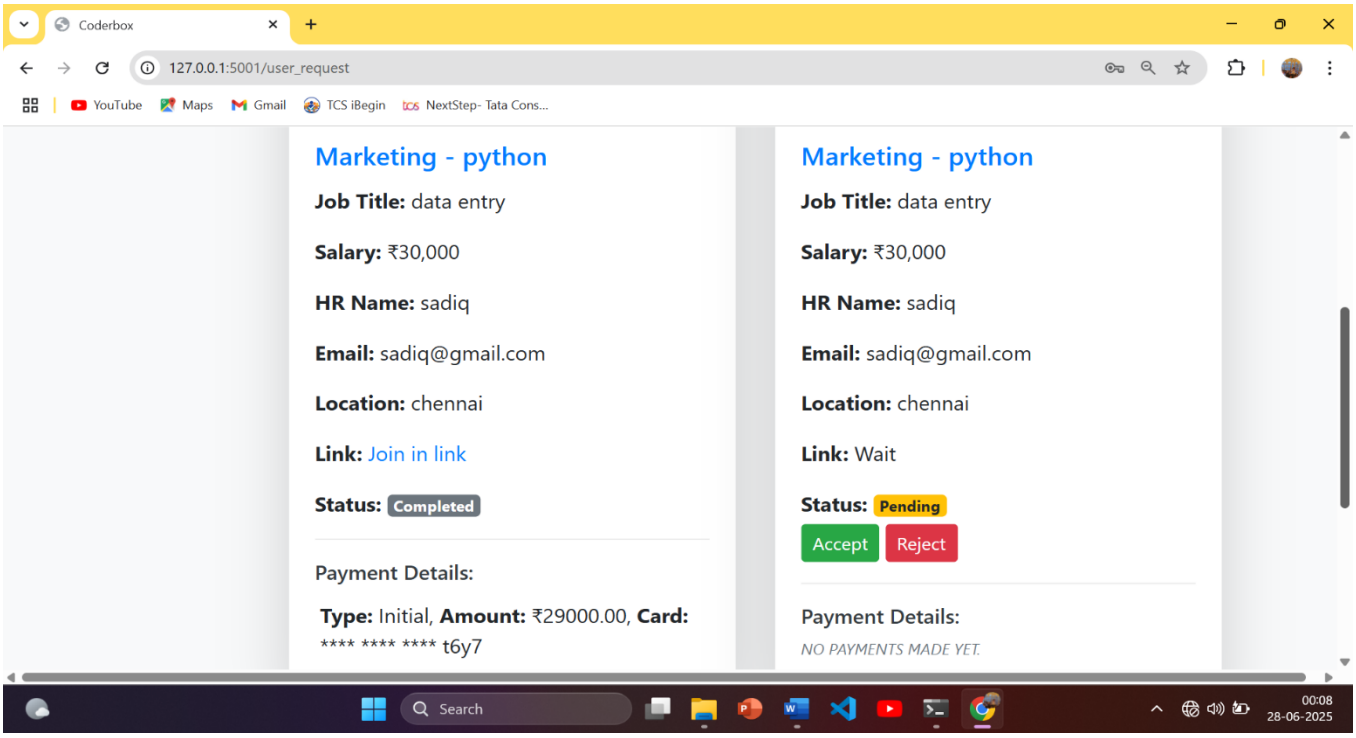
PROFILE DELETION PAGE



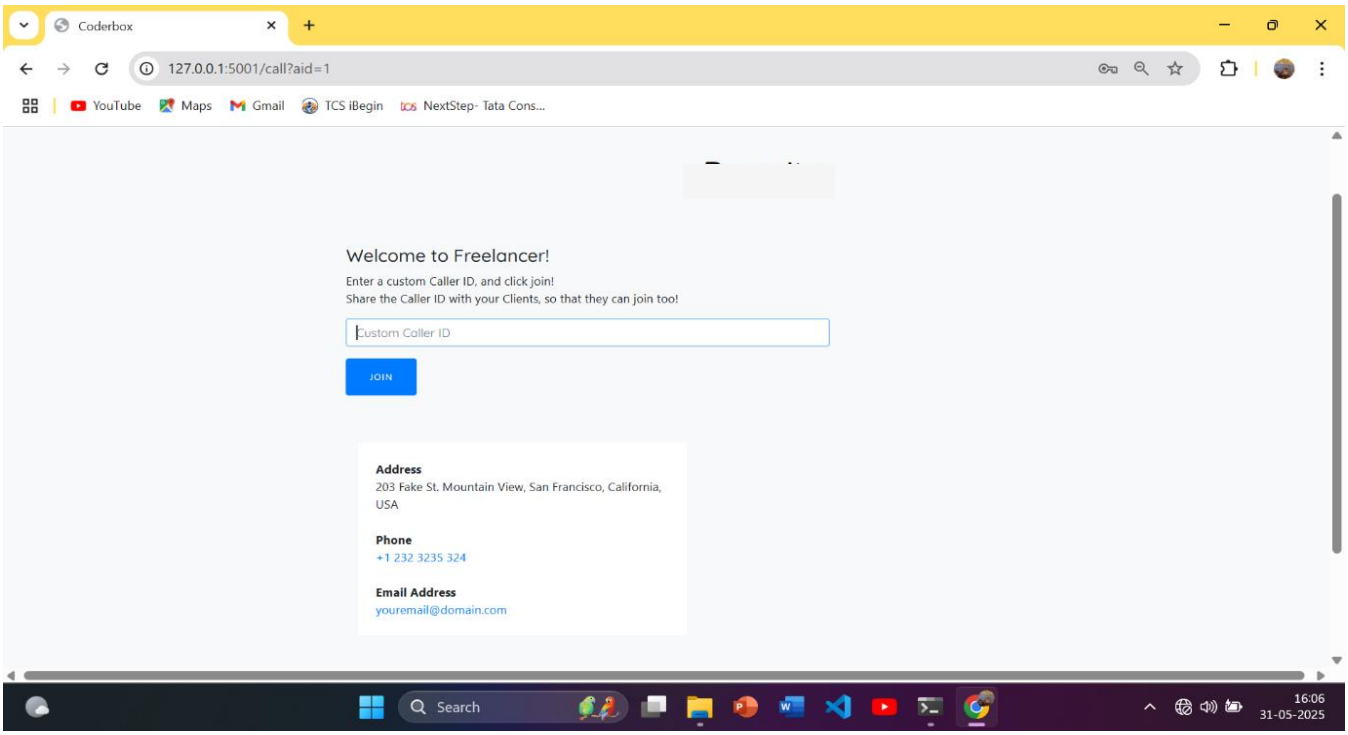
PROFILE UPDATION PAGE



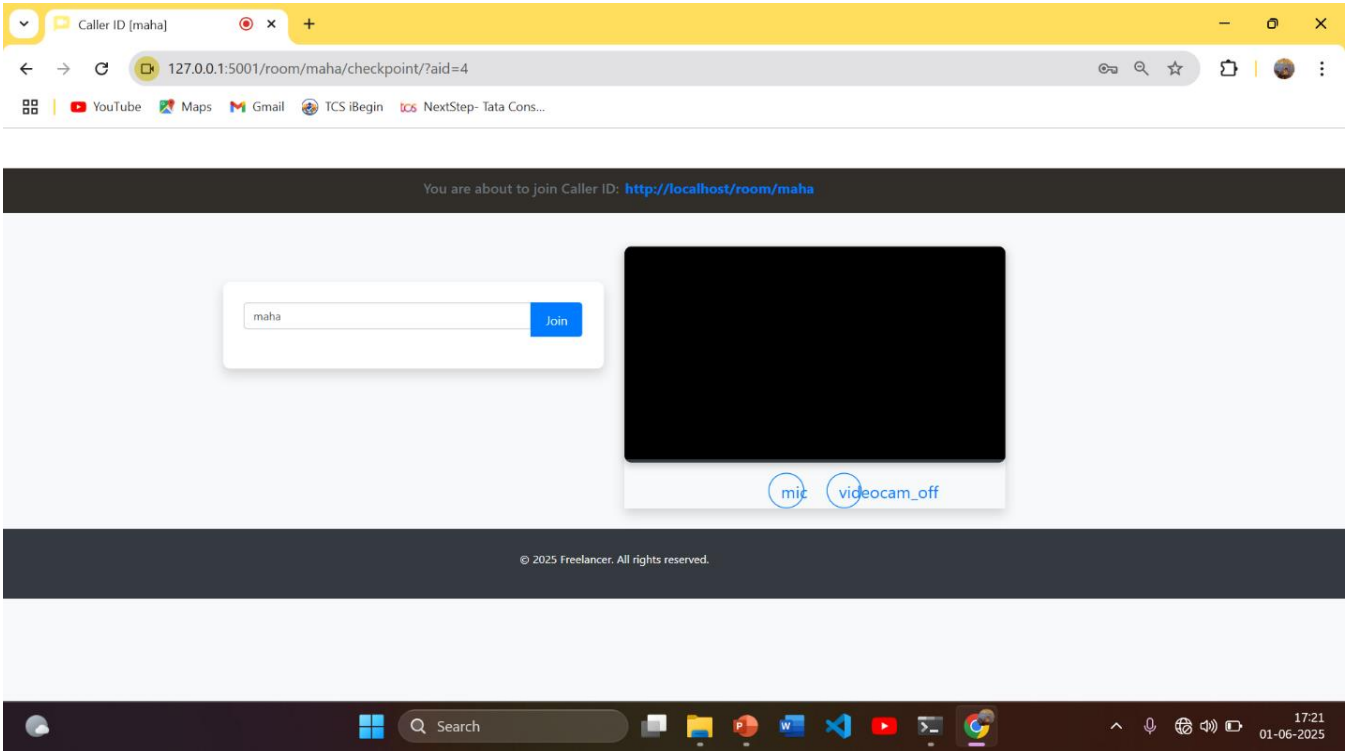
FREELANCER PROJECT PAGE



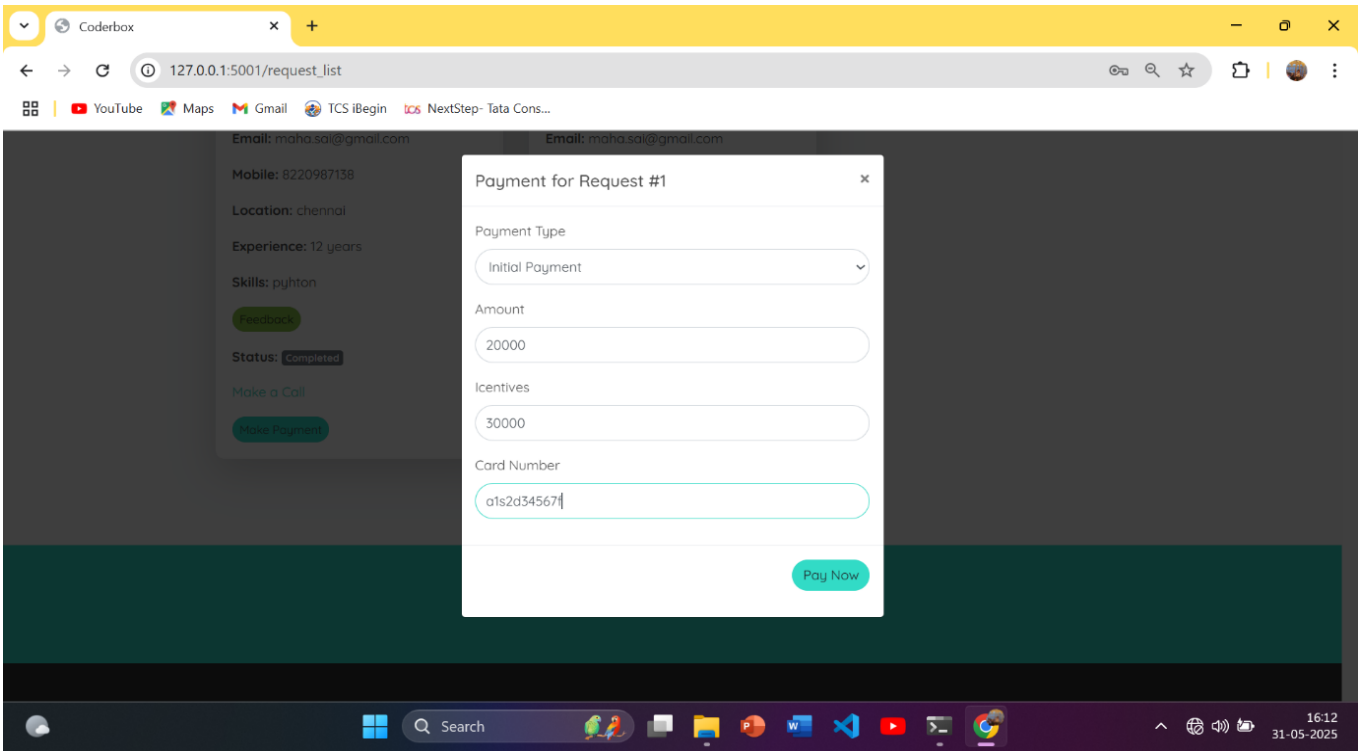
CONFERENCING PAGE



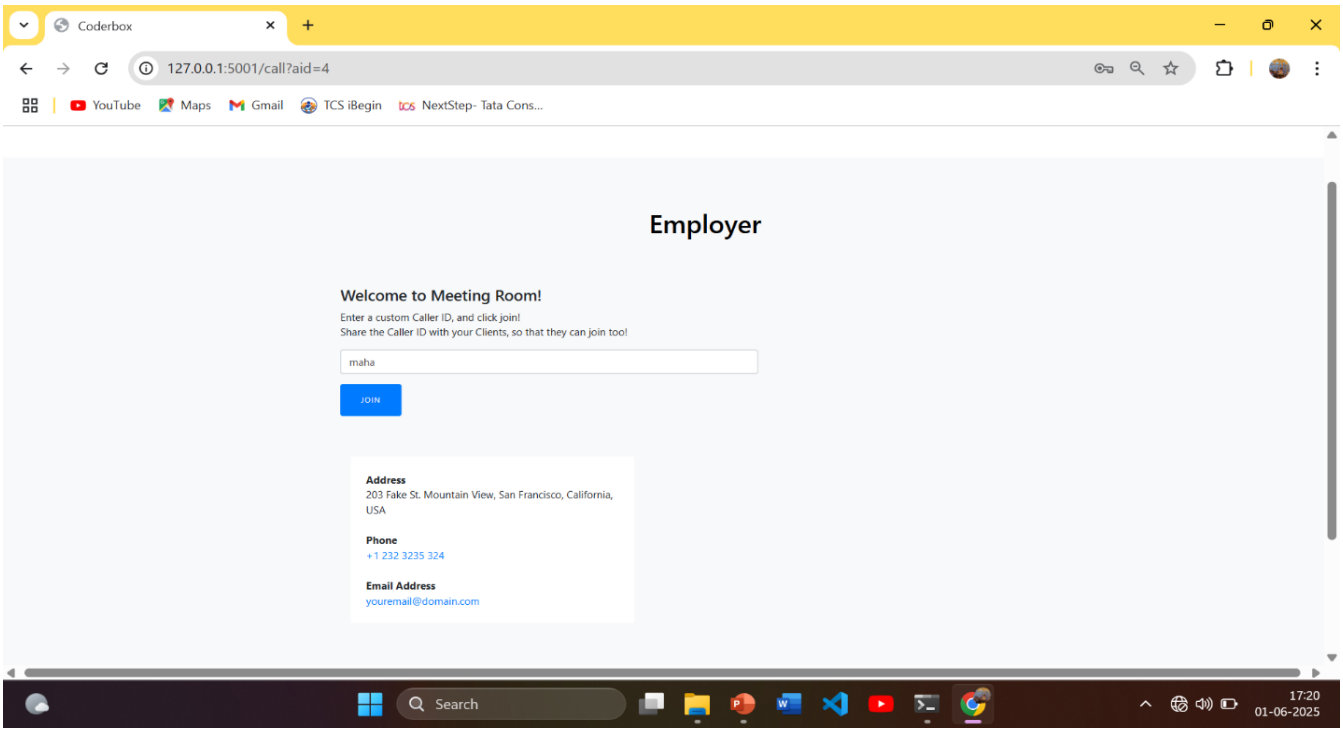
MEETING ROOM PAGE



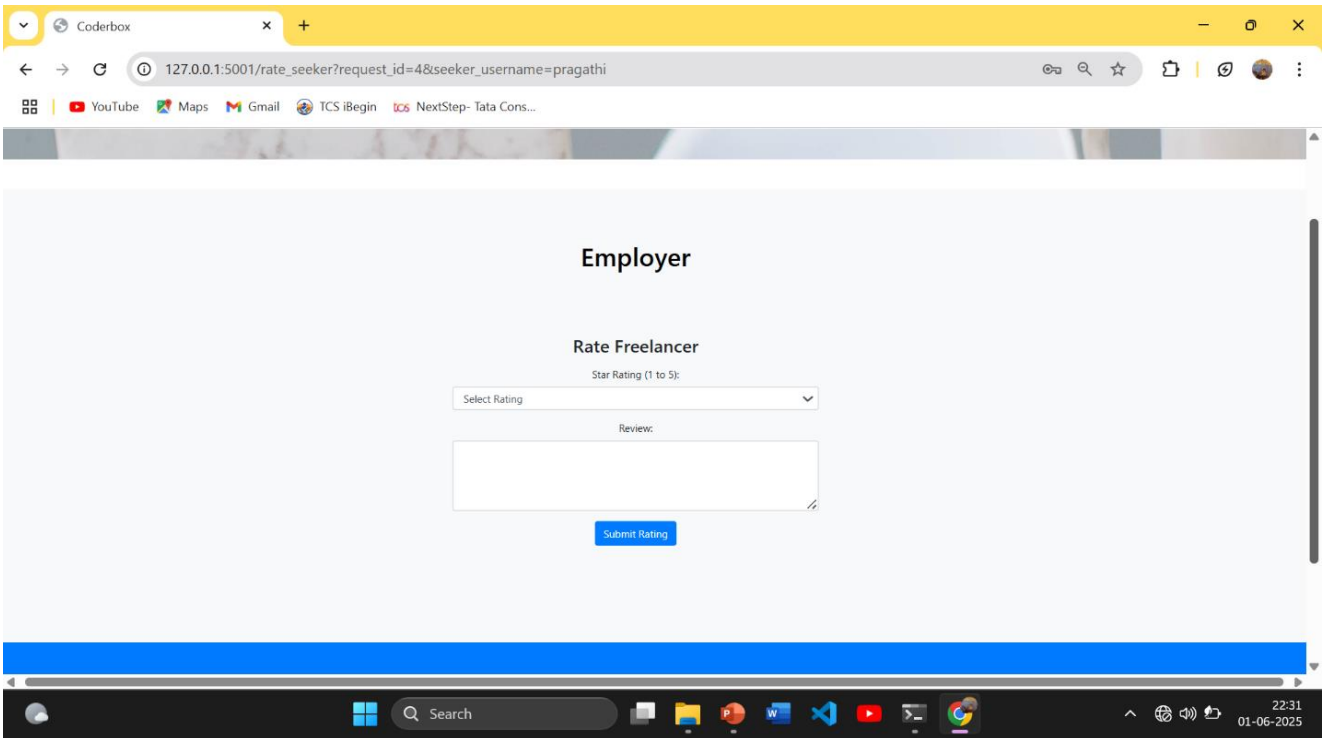
INCENTIVES PAGE



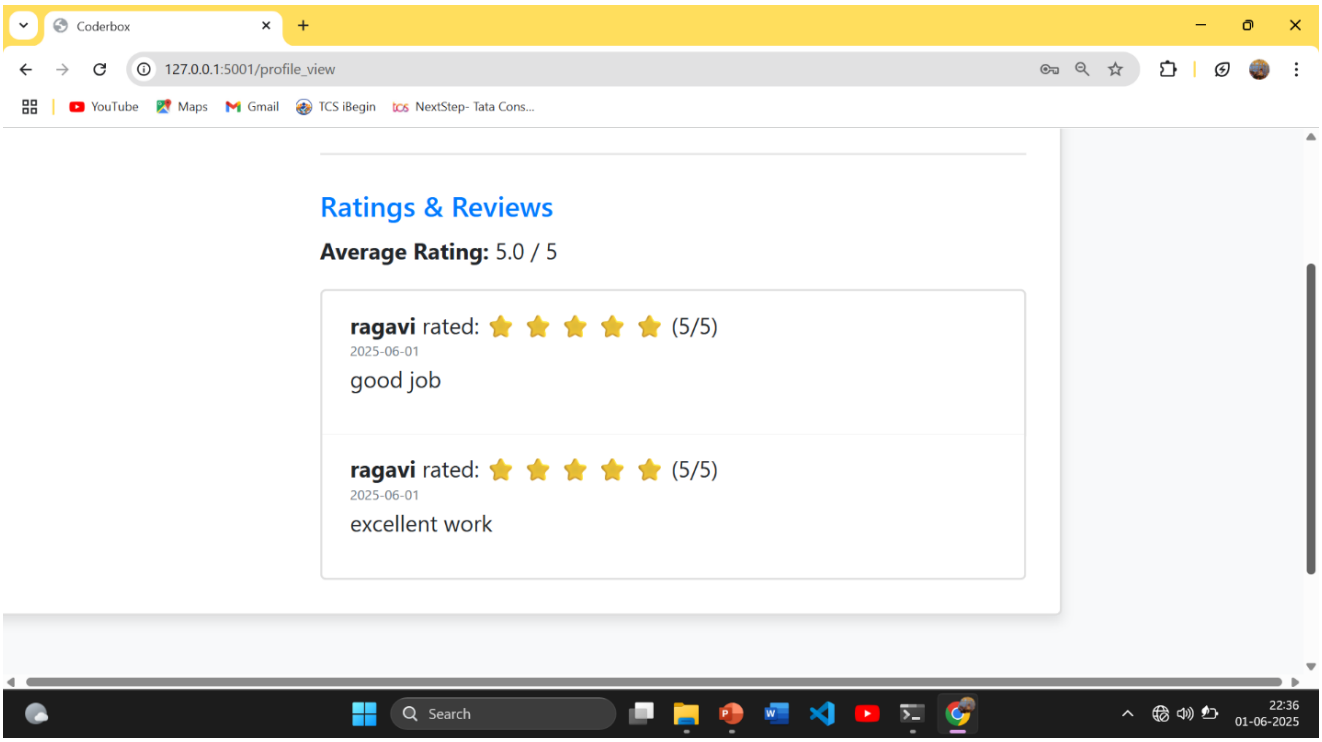
EMPLOYER MEETING PAGE



FEEDBACK PAGE



REVIEWS PAGE



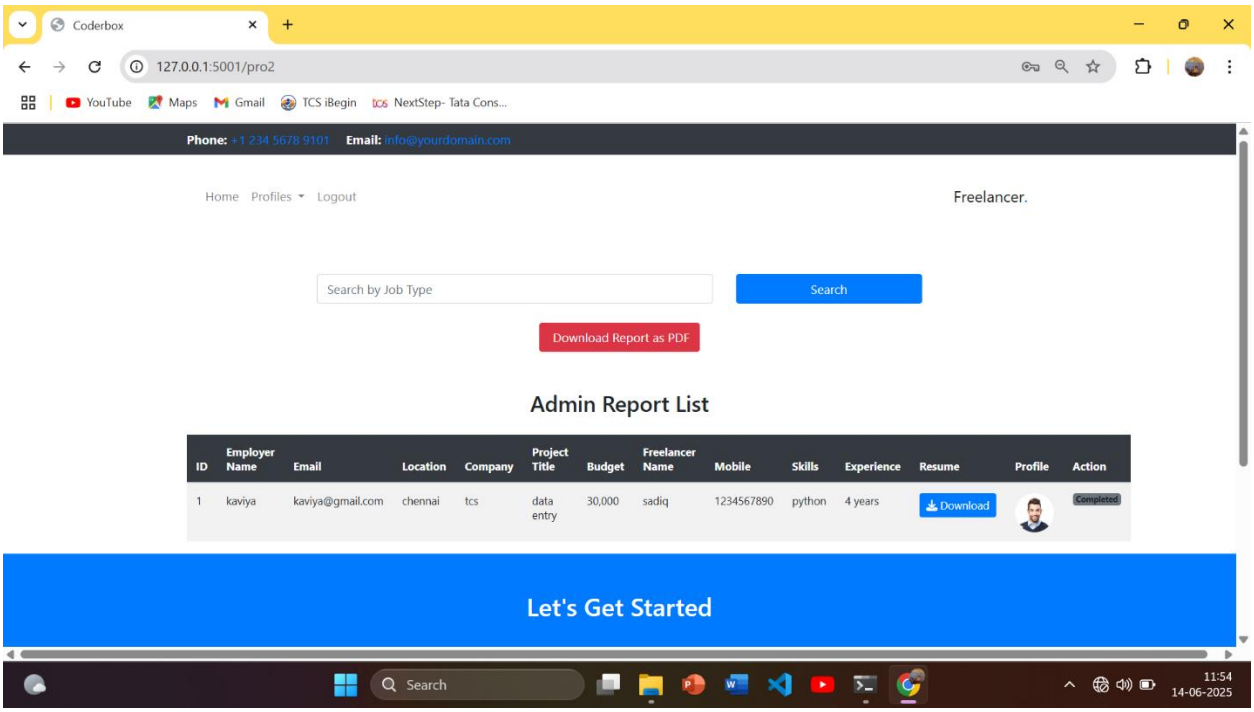
APPENDIX III

SAMPLE REPORT

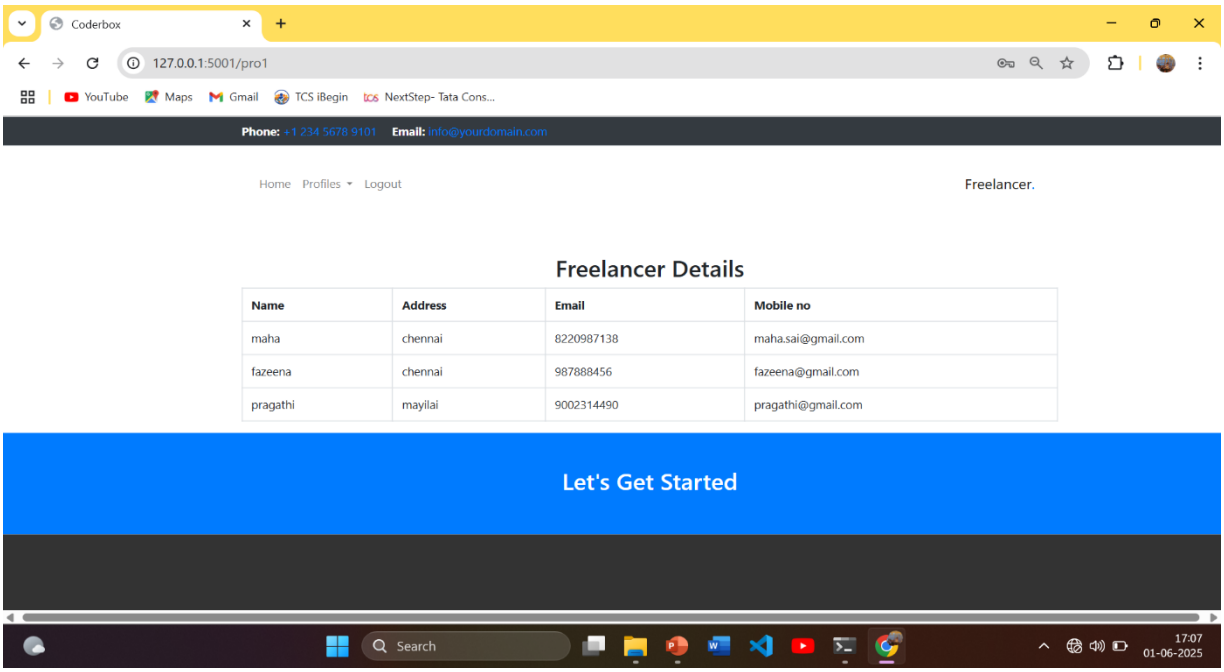
APPENDIX III

SAMPLE REPORT

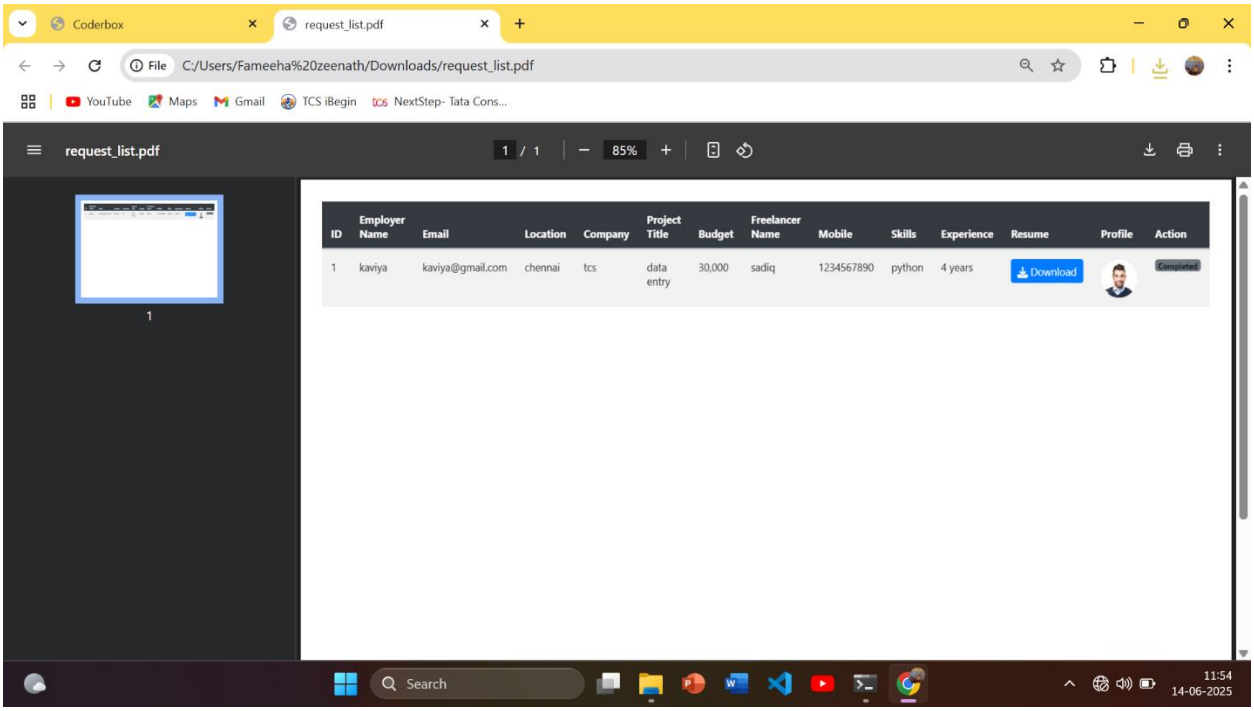
EMPLOYER WISE REPORT LIST



FREELANCER WISE REPORT



ADMIN WISE REPORT LIST



CHAPTER 11

REFERENCES

CHAPTER 11
REFERENCES

JOURNAL REFERENCES

1. A. A. Osintseva, "ICT and Mental Health of Professionals Working from Home," IEEE, 2023. https://doi.org/10.1109/EDM58354.2023.10225054
2. H. R. I. E. Ranasinghe & K. S. Ranasinghe, "Ensemble Learning Approach for Predicting Job Satisfaction on Freelancing Jobs in Sri Lanka," IEEE, 2023. https://doi.org/10.1109/CENTCON56610.2022.10051528
3. I. Rauf & M. Petre, "Security Thinking in Online Freelance Software Development," IEEE, 2023. https://doi.org/10.1109/ICSE-SEIS58686.2023.00008
4. B. Pallam & M. M. Gore, "Boomerang: Blockchain-based Freelance Paradigm on Hyperledger," IEEE, 2019. https://doi.org/10.1109/ICCCNT45670.2019.8944572
5. C. B. A. Khan & A. Ahsan, "Recommended Configuration Management Practices for Freelance Software Developers," IEEE, 2014. https://doi.org/10.1109/ICSESS.2014.6933524

BOOK REFERENCES

1. Python: "Python Crash Course" offers a comprehensive introduction to Python programming.
2. Flask: "Flask Web Development" provides detailed guidance on building web applications with Flask.
3. MySQL: "Learning MySQL" is a practical guide to mastering MySQL database management.
4. Wampserver: "WampServer: Installation, Configuration, Administration, and Maintenance" covers all aspects of setting up and managing a Wampserver environment.
5. Pandas: "Python for Data Analysis" is a go-to resource for learning Pandas for data manipulation and analysis.
6. Scikit Learn: "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" offers practical examples for using Scikit Learn for machine learning tasks.
7. Matplotlib: "Python Data Science Handbook" provides concise tutorials on data visualization with Matplotlib.
8. NumPy: "Python for Data Analysis" is a valuable resource for understanding and using NumPy for numerical computing in Python.
9. Seaborn: "Python Data Science Handbook" includes tutorials on creating attractive statistical graphics with Seaborn.
10. JSON for Blockchain: "Mastering Blockchain" offers insights into using JSON for blockchain development and integration.
11. Bootstrap: "Bootstrap 4 Quick Start" provides quick and easy tutorials for getting started with Bootstrap for responsive web design.

WEB REFERENCES

1. Python: https://www.python.org/doc/
2. Flask: https://flask.palletsprojects.com/
3. MySQL: https://dev.mysql.com/doc/
4. Wampserver: http://www.wampserver.com/
5. Pandas: https://pandas.pydata.org/docs/
6. Scikit Learn: https://scikit-learn.org/stable/documentation.html
7. Matplotlib: https://matplotlib.org/stable/contents.html
8. NumPy: https://numpy.org/doc/
9. Seaborn: https://seaborn.pydata.org/docs/
10. JSON for Blockchain: https://www.json.org/json-en.html
11. Bootstrap: https://getbootstrap.com/docs/