1.1 Institute profile

Established in 1983 by the Maharashtra Education Society (MES), IMCC is a distinguished institute recognized by SPPU, dedicated to providing high-quality postgraduate education in Computers and Management. Located in Pune, the institute's motto, "FACTA-NON-VERBA," underscores its commitment to cultivating professionals whose actions speak louder than words. Emphasizing total personality development and fostering a unique 'Teacher-Disciple Relationship,' IMCC aims to produce adept professionals who excel in flexibility, integration, and transformation. The institute's conducive environment nurtures budding managers, channelizing their enthusiasm and rationale to seamlessly integrate business and technological management skills. With a faculty dedicated to synthesizing business acumen and technology insights, IMCC prepares students for tangible success through a dynamic and creative educational approach.

1.2 Abstract

WorkBees s is a web application designed to bridge the gap between freelancers and clients seeking professional services. With a focus on seamless collaboration, WorkBees offers a diverse range of freelance services across various categories, catering to the needs of both clients and freelancers. The project's objectives include developing a user-friendly interface, incorporating diverse service categories, implementing an efficient matching algorithm, integrating a secure payment system, and enabling real-time communication features. The scope of WorkBees encompasses features such as user registration and profiles, service listings, matching and recommendations, a secure payment gateway, messaging tools, and a ratings and reviews system. By providing a centralized platform for hiring and project management, WorkBees aims to streamline the freelance marketplace, making it easier for clients to find the right talent and for freelancers to showcase their skills and expertise.

1.3 Existing system and need for system

Existing system:

Currently, the freelance market is characterized by a multitude of platforms, each with its own set of features and limitations. These platforms often lack cohesion, making it challenging for both clients and freelancers to navigate and find suitable matches efficiently. While some platforms offer basic search functionalities, they often rely on manual browsing and lack robust algorithms for matching clients with freelancers based on specific project requirements. Communication tools on these platforms may be limited, hindering effective collaboration between parties. Additionally, payment systems may vary in terms of security and reliability, raising concerns among users about the safety of financial transactions.

Need for system:

The dynamic nature of the freelance economy demands a centralized platform that addresses the shortcomings of existing systems. WorkBees emerges to fill this gap by offering a comprehensive solution that streamlines the freelance hiring process. Several key factors underscore the need for such a system:

1.3.1. Efficiency: Existing platforms may lack efficient matching algorithms, leading to prolonged search times and mismatches between clients and freelancers. WorkBees aims to enhance

efficiency by implementing a robust matching algorithm that connects clients with freelancers based on their project requirements and skills, thereby streamlining the hiring process.

1.3.2. Coherence: With numerous platforms available, there is a need for cohesion in the freelance market. WorkBees provides a unified platform where clients can easily find a diverse range of freelance services across various categories, simplifying the hiring process and reducing fragmentation in the market.

1.3.3. Security: Concerns about the security of financial transactions are prevalent among users of existing platforms. WorkBees addresses these concerns by integrating a secure payment system, ensuring that transactions between clients and freelancers are conducted safely and reliably.

1.3.4. Communication: Effective communication is essential for successful collaboration, yet existing platforms may offer limited communication tools. WorkBees enhances communication by providing an in-app messaging tool, facilitating real-time interaction between clients and freelancers and fostering productive working relationships.

In summary, the need for WorkBees arises from the inefficiencies and limitations of existing freelance platforms, highlighting the demand for a cohesive, efficient, and secure system that facilitates seamless collaboration between clients and freelancers.

1.4 Scope of system

The "WorkBees" project entails a comprehensive array of functionalities aimed at establishing an intuitive and effective platform for connecting freelancers and clients seeking professional services. These essential features comprise:

1.4.1. User Registration and Profiles:

WorkBees will facilitate the creation of detailed profiles for both freelancers and clients. These profiles will allow users to showcase their skills, expertise, and project history, providing valuable insights for potential collaborators.

1.4.2. Service Listings:

Freelancers will have the capability to list their services on WorkBees, presenting clear and thorough descriptions of their offerings, including rates and delivery times. This feature will empower clients to easily peruse and select services that align with their specific requirements.

1.4.3. Matching and Recommendations:

WorkBees will incorporate an advanced search facility employing sophisticated algorithms to suggest suitable freelancers for given projects. These recommendations will be based on criteria such as skills, experience, and past performance, ensuring optimal matches between clients and freelancers.

1.4.4. Secure Payment Gateway:

To facilitate seamless and secure transactions between clients and freelancers, WorkBees will integrate a robust payment gateway. This feature will instil confidence in users regarding the reliability and security of financial transactions conducted through the platform.

1.4.5. Messaging Tool:

Effective communication is paramount for successful collaboration. Hence, WorkBees will provide an intuitive in-app messaging tool enabling users to communicate in real-time. This feature will streamline communication and enhance project management capabilities.

1.4.6. Ratings and Reviews:

WorkBees will empower clients to rate and review freelancers based on their collaboration experience. These ratings and reviews will furnish valuable feedback for both freelancers and clients, fostering accountability and transparency within the platform.

The "WorkBees" project endeavours to establish a robust and user-friendly ecosystem facilitating seamless collaboration between clients and freelancers.

1.5 Operating environment – Hardware and Software

1.5.1. Hardware specification

Processor: Intel(R) Core(TM) i5-8350U CPU @ 1.70GHz 1.90 GHz

RAM: 16.00 GB

System type: 64-bit operating system, x64-based processor

1.5.2. Software Requirements

Language - JavaScript

Technology used – React JS, Node JS, Express JS, HTML, CSS

IDE: Visual Studio Code

Operating system – windows 11

Database –MongoDB

Browser – Chrome, Firefox, Microsoft Edge.

1.6 Brief description of technology used

1.6.1 Operating system used

The WorkBees project operates on the Windows operating system. Windows is a widely used operating system developed by Microsoft, known for its user-friendly interface and broad compatibility with software and hardware. It provides a familiar environment for developers and users alike, offering a range of tools and applications to support various project requirements. With its widespread adoption and robust support ecosystem, Windows serves as a reliable platform for the WorkBees project, facilitating seamless development and collaboration among team members.

1.6.2 RDBMS/ No SQL used to build database

For the WorkBees project, the database technology used is MongoDB, which is a NoSQL database. MongoDB is a popular choice for projects built on the MERN stack due to its flexibility and scalability, making it well-suited for handling the dynamic and diverse data requirements of web applications. As a NoSQL database, MongoDB allows for the storage of unstructured or semi-structured data, which aligns well with the nature of web applications that often deal with changing data schemas and large volumes of data. Additionally, MongoDB integrates seamlessly with the other technologies in the MERN stack, enabling smooth development and deployment processes for the WorkBees project.

2.1 Study of similar system

Before embarking on the development of the WorkBees project, a comprehensive study of similar systems in the freelance marketplace domain was conducted to gain insights into existing solutions, identify industry trends, and understand user preferences. The study focused on examining various platforms that connect freelancers with clients seeking professional services, analysing their features, functionalities, user experience, and business models. Key areas of investigation included:

2.1.1. Platform Features and Functionalities: A thorough analysis of similar systems was conducted to understand the range of features offered, such as user registration, service listings, project management tools, communication channels, payment gateways, and rating/review systems. This helped in identifying essential features and innovative functionalities that could be incorporated into the WorkBees platform to enhance its competitiveness and user satisfaction.

2.1.2. User Experience (UX) Design: The study delved into the UX design aspects of comparable platforms, examining their interface layout, navigation structure, ease of use, and responsiveness across different devices. Insights gained from this analysis informed the design decisions for WorkBees, ensuring a user friendly and intuitive interface that promotes engagement and satisfaction.

2.1.3. Market Trends and User Preferences: By researching market trends and user preferences in the freelance marketplace industry, valuable insights were obtained regarding the evolving needs and expectations of both freelancers and clients. This informed strategic decision-making regarding feature prioritization, target audience alignment, and market positioning for WorkBees.

2.1.4. Monetization Strategies: The study explored diverse monetization strategies employed by similar platforms, including subscription models, transaction fees, premium features, and advertising. This facilitated the identification of viable revenue streams and informed the development of a sustainable business model for WorkBees.

2.1.5. Success Stories and Challenges: Analysis of success stories and challenges faced by existing platforms provided valuable lessons and best practices for the development and operation of WorkBees. Understanding the factors contributing to the success or failure of similar systems helped in mitigating risks and optimizing strategies for achieving long-term viability and growth.

2.2 Feasibility study

The feasibility study conducted for the WorkBees project assessed its viability and potential for success from various perspectives, including technical, economic, operational, and scheduling aspects. Key components of the feasibility study included:

2.2.1. Technical Feasibility: This aspect evaluated the project's technical requirements, including the capabilities of the MERN stack (MongoDB, Express.js, React, Node.js) to meet the development needs of WorkBees. Considerations such as the availability of skilled developers, compatibility with required third-party integrations (e.g., payment gateways), and scalability of the chosen technologies were assessed to ensure the project's technical feasibility.

2.2.2. Economic Feasibility: An analysis of the project's economic feasibility involved estimating the initial development costs, ongoing operational expenses, and potential revenue streams. This included evaluating factors such as infrastructure costs, development resources, marketing expenses, and projected user acquisition and retention rates. Financial projections and return on investment (ROI) calculations were conducted to determine the project's profitability and sustainability.

2.2.3. Operational Feasibility: The operational feasibility study focused on assessing the practicality of implementing and managing the WorkBees platform within the organization. This involved evaluating factors such as resource availability, organizational readiness, workflow integration, regulatory compliance, and potential challenges in day-to-day operations. Mitigation strategies for identified risks and constraints were developed to ensure smooth project execution.

2.2.4. Scheduling Feasibility: The scheduling feasibility analysis involved creating a detailed project timeline and development roadmap to assess the feasibility of meeting project deadlines and milestones. This included identifying critical path activities, allocating resources effectively, and managing dependencies and constraints. Contingency plans were devised to address potential delays or setbacks and ensure timely project delivery.

Based on the findings of the feasibility study, WorkBees was deemed viable and strategically sound, with favourable assessments across technical, economic, operational, and scheduling dimensions. The study provided valuable insights into the project's potential challenges and opportunities, enabling informed decision-making and effective planning for its successful implementation and launch.

2.3 Objectives of proposed system

Seamless Collaboration: Foster seamless collaboration between freelancers and clients by providing a user-friendly platform that simplifies the process of hiring, project management, and communication.

2.3.1 Diverse Service Offerings: Offer a diverse range of freelance services across various categories to cater to the needs of a broad user base and accommodate various skill sets and expertise.

2.3.2 Efficient Matching Algorithm: Implement an advanced matching algorithm that connects clients with freelancers based on their specific requirements, skills, experience, and preferences, facilitating optimal project outcomes.

2.3.3 User Engagement: Enhance user engagement and satisfaction through intuitive interface design, responsive communication channels, and interactive features that promote active participation and interaction.

2.3.4 Secure Transactions: Ensure the security and integrity of financial transactions by integrating a robust and secure payment gateway that safeguards sensitive information and protects against fraudulent activities.

2.3.5 Real-Time Communication: Enable real-time communication features, including messaging tools and

collaboration platforms, to facilitate seamless communication and collaboration between freelancers and clients throughout the project lifecycle.

2.3.6 Continuous Improvement: Commit to continuous improvement and innovation by soliciting user feedback, monitoring industry trends, and iteratively enhancing the platform's features, functionalities, and user experience to meet evolving needs and expectations.

2.4 Users of system

The WorkBees system is designed to cater to two primary categories of users: freelancers and clients.

Freelancers:

Freelancers are individuals who offer their professional services through the WorkBees platform. They may possess expertise in various fields such as graphic design, web development, writing, marketing, and more. Freelancers utilize the platform to showcase their skills, create profiles, list their services, communicate with potential clients, manage projects, and receive payments for completed work.

Clients:

Clients are individuals or businesses seeking professional services from freelancers. They utilize the WorkBees platform to browse through available freelancers, post job listings, communicate with freelancers, negotiate terms, manage projects, and make payments for completed work.

By catering to the needs of both freelancers and clients, the WorkBees system facilitates efficient collaboration, communication, and project management within the freelance marketplace ecosystem.

3.1 System requirements (Functional and Non-Functional requirements):

3.1.1 Functional Requirements:

User Registration and Authentication: Users should be able to register and create accounts securely. Authentication mechanisms such as email verification or password authentication should be implemented.

Profile Management: Users should have the ability to create, edit, and manage their profiles. This includes adding personal information, skills, portfolio items (for freelancers), and project preferences (for clients).

Service Listings: Freelancers should be able to list their services, including descriptions, pricing, and delivery timeframes. Clients should be able to browse and search for services based on categories, keywords, or filters.

Bidding: Clients should be able to post job listings detailing project requirements, budgets, and deadlines. Freelancers should be able to browse and bid on available jobs, submitting proposals with their pricing and timelines.

Messaging and Communication: Users should be able to communicate securely within the platform. This includes real-time messaging, file sharing, and notifications for new messages or updates.

Payment Processing: The platform should support secure payment processing for transactions between clients and freelancers. This may involve integrating third-party payment gateways and implementing escrow services for milestone-based payments.

Rating and Review System: Both clients and freelancers should be able to rate and review each other upon project completion. This feedback system helps maintain accountability and build trust within the platform.

3.1.2 Non-Functional Requirements:

Security: The system should adhere to industry-standard security practices to protect user data, prevent unauthorized access, and safeguard financial transactions.

Scalability: The platform should be designed to handle increasing numbers of users, services, and transactions without compromising performance or reliability.

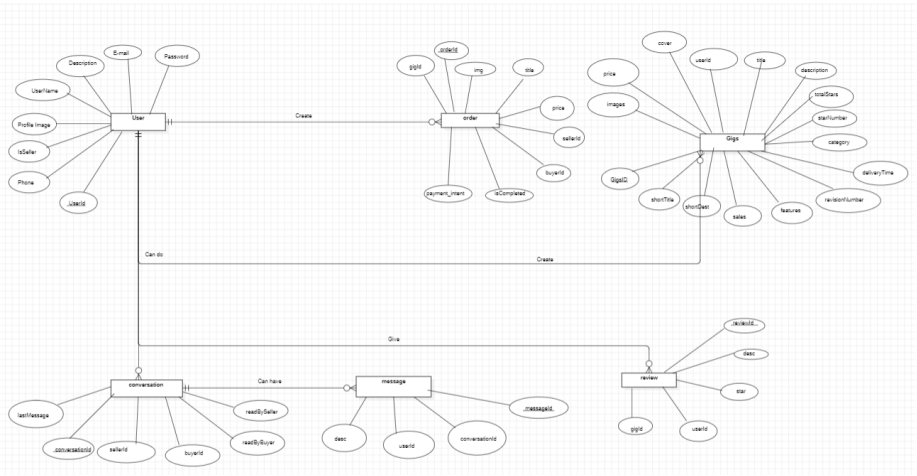
Usability: The user interface should be intuitive, responsive, and accessible across different devices and screen sizes. User interactions should be smooth and efficient.

Performance: The system should have fast response times and minimal downtime. This includes optimizing code, caching frequently accessed data, and scaling resources as needed.

Reliability: The platform should be reliable and available for users at all times. Measures should be in place to mitigate potential failures, such as regular backups and disaster recovery plans.

Support and Maintenance: Adequate support channels should be available to assist users with any issues or inquiries. Regular maintenance and updates should be performed to ensure the system's continued functionality and security.

3.2 Entity relationship diagram (ERD):



3.3 Table structure:

3.3.1 User Table:

Sr. No.	Field Name	Data type	Constraints
1	username	String	Unique, required
2	email	String	Unique, required
3	password	String	Required
4	img	String	-
5	country	String	Required
6	phone	String	-
7	description	String	-
8	isSeller	Boolean	-
9	userId	Object Id	Unique, Auto

3.3.2 Gig Table:

Sr. No.	Field Name	Data type	Constraints
1	gigId	Object Id	Unique, Auto
2	userId	String	Unique
3	title	String	Required
4	desc	String	Required
5	totalStars	Number	-
6	starNumber	Number	-
7	cat	String	-
8	price	Number	Required
9	cover	String	-
10	images	Array of String	-
11	shortTitle	String	Required
12	shortDesc	String	Required
13	deliveryTime	Number	Required
14	revisionTime	Number	Required
15	features	Array of String	-
16	sales	Number	-

3.3.3 Message Table:

Sr. No.	Field Name	Data type	Constraints
1	messageId	Object Id	Unique, Auto
2	conversationId	String	Unique
3	userId	String	Unique
4	desc	String	Required

3.3.4 Order Table:

Sr. No.	Field Name	Data type	Constraints
1	orderId	Object Id	Unique, Auto
2	gigId	String	Unique, Required
3	img	String	-
4	title	String	Required
5	price	Number	Required
6	sellerId	String	Required
7	buyerId	String	Required
8	isCompleted	Boolean	-
9	payment_intent	String	Required

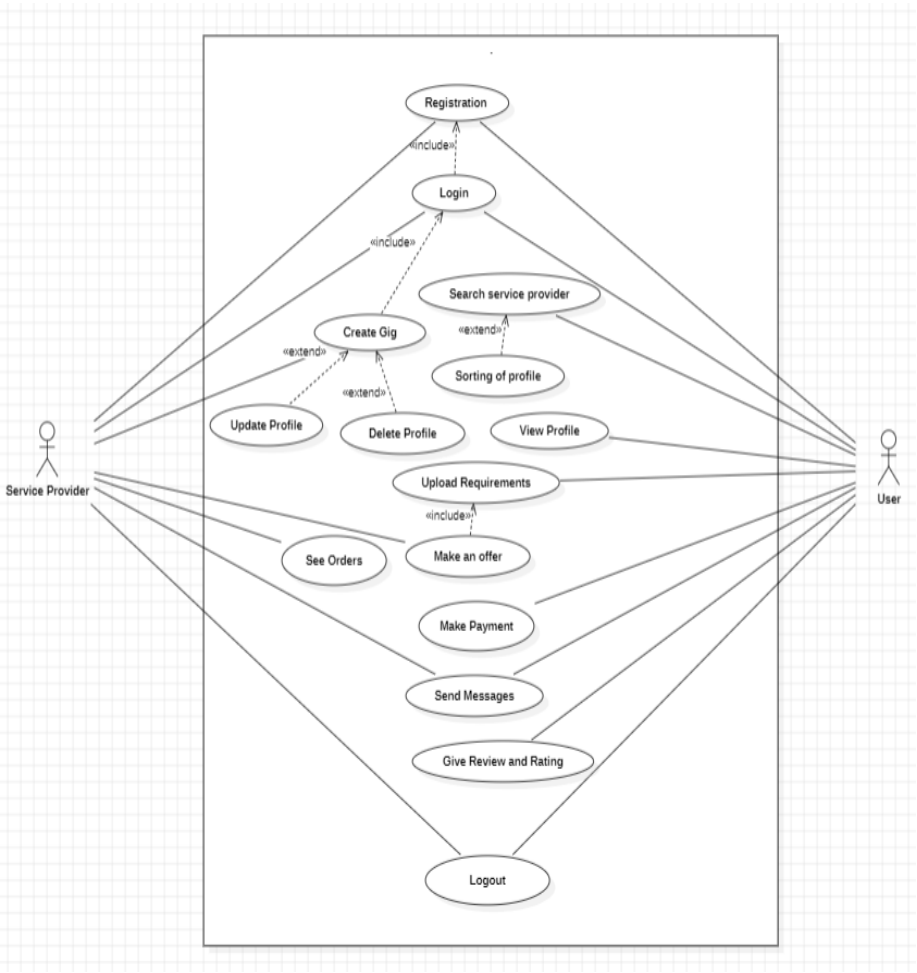
3.3.5 Review Table:

Sr. No.	Field Name	Data type	Constraints
1	reviewId	Object Id	Unique, Auto
2	gigId	String	Unique
3	star	Number	-
4	desc	String	Required

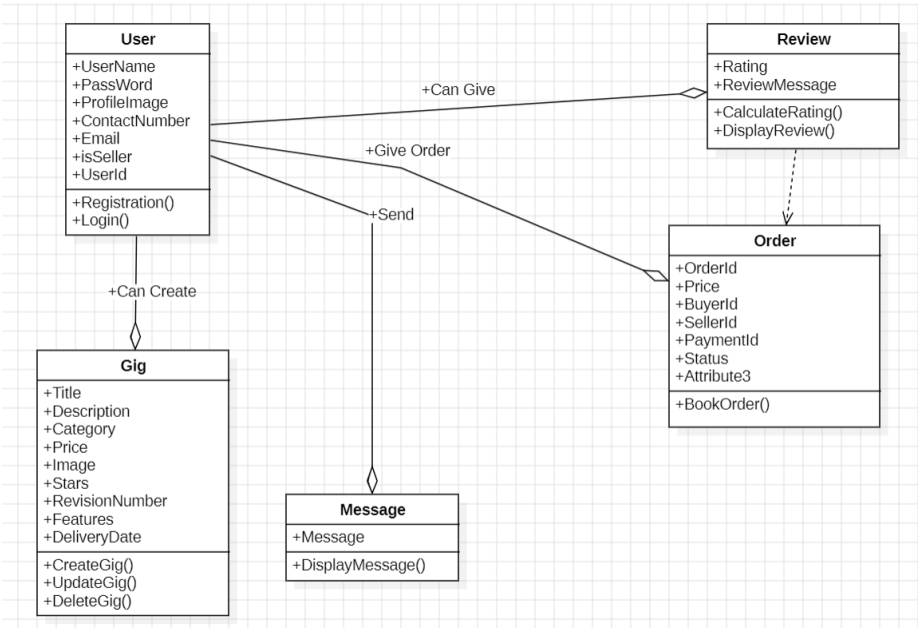
3.3.6 Conversion Table:

Sr. No.	Field Name	Data type	Constraints
1	conversionId	Object Id	Unique, Auto
2	sellerId	String	Required
3	buyerId	String	Required
4	readBySeller	Boolean	-
5	readByBuyer	Boolean	-
6	lastMessage	String	-

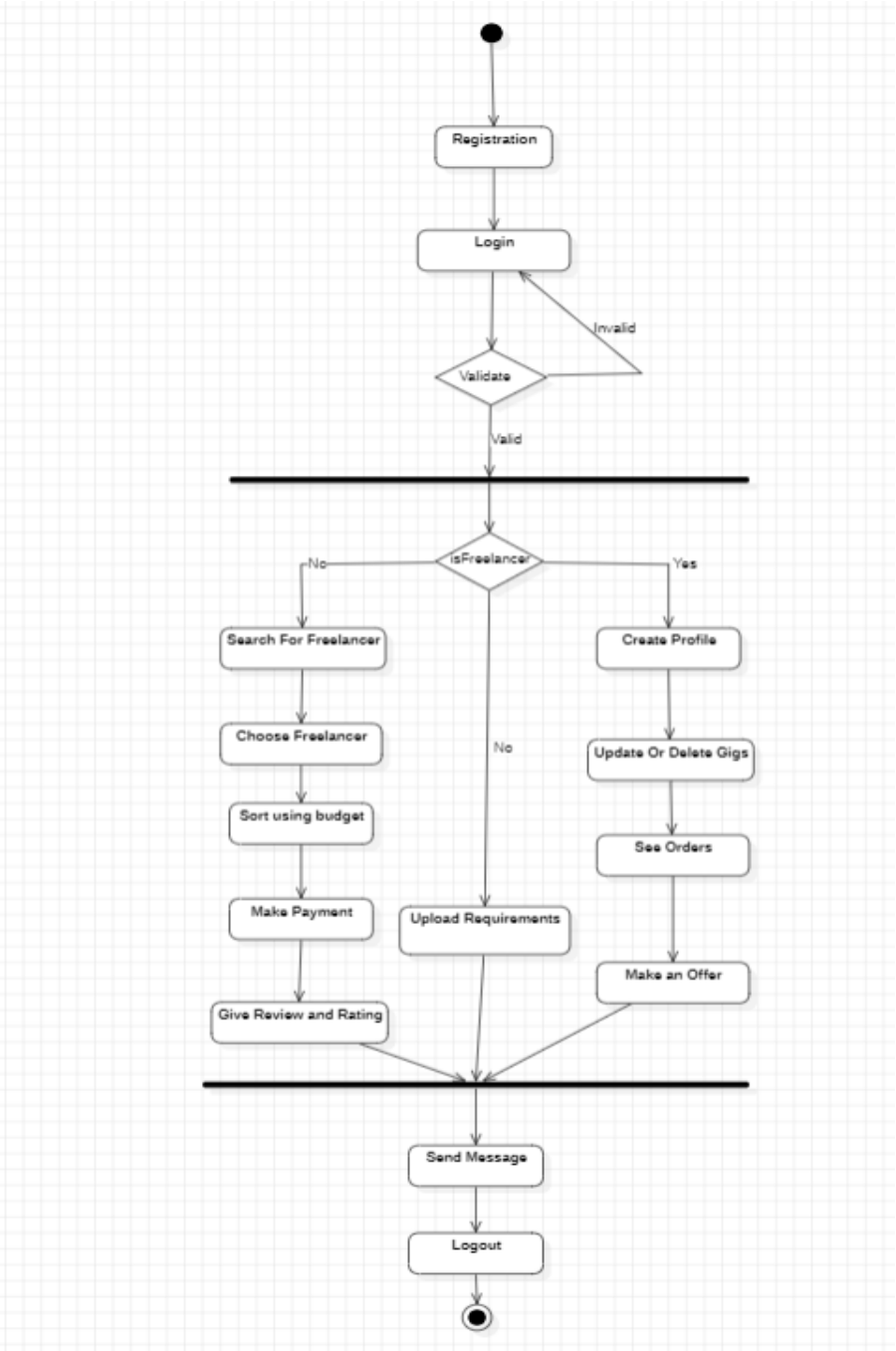
3.4 Use case diagram:



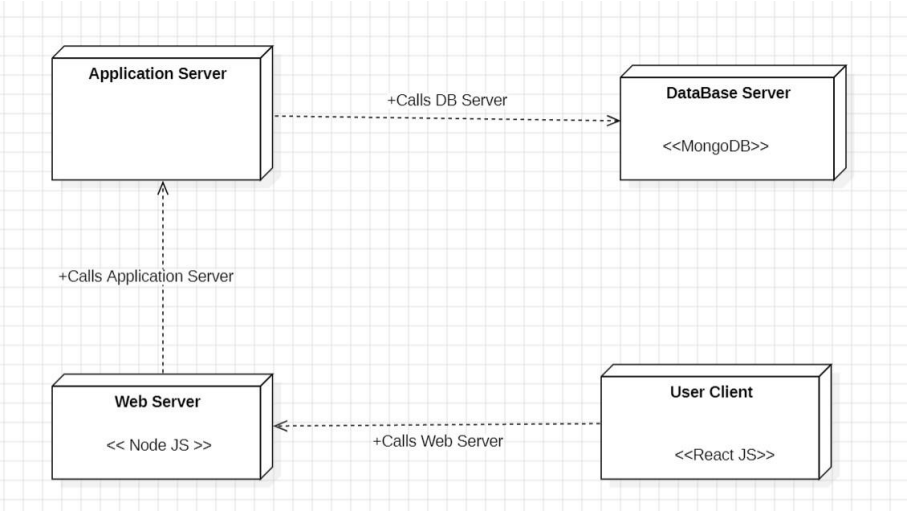
3.5 Class diagram:



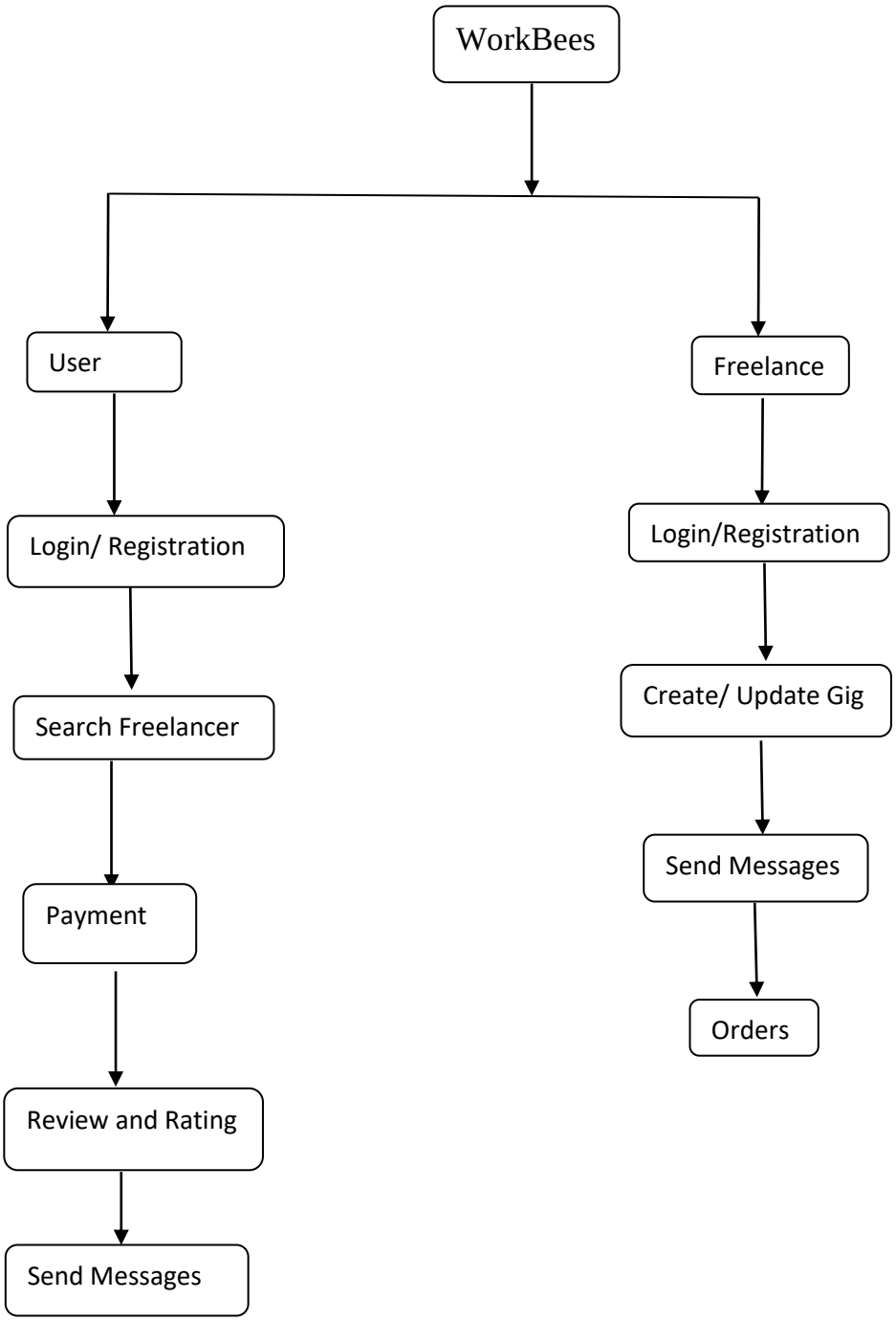
3.6 Activity diagram:



3.7 Deployment diagram:

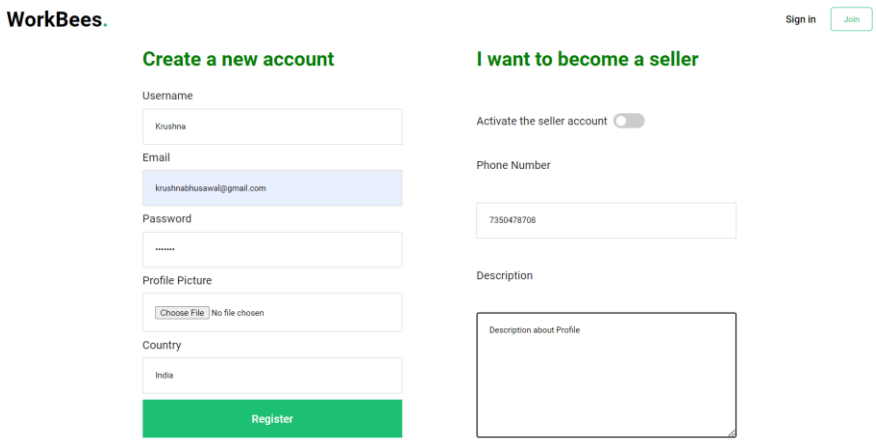


3.8 Module hierarchy diagram:

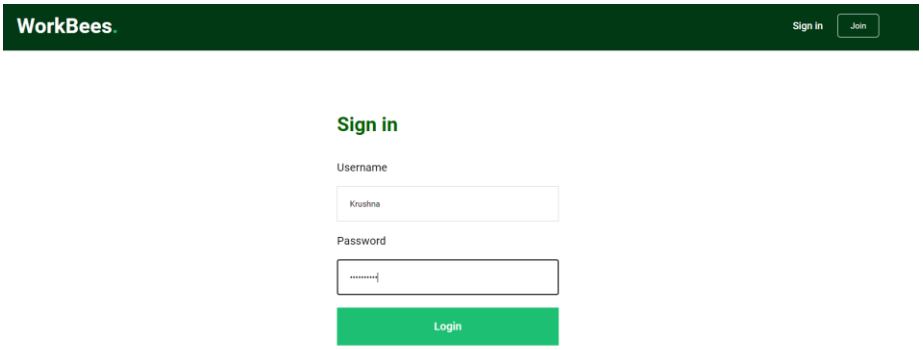


3.9 Sample input and output screens:

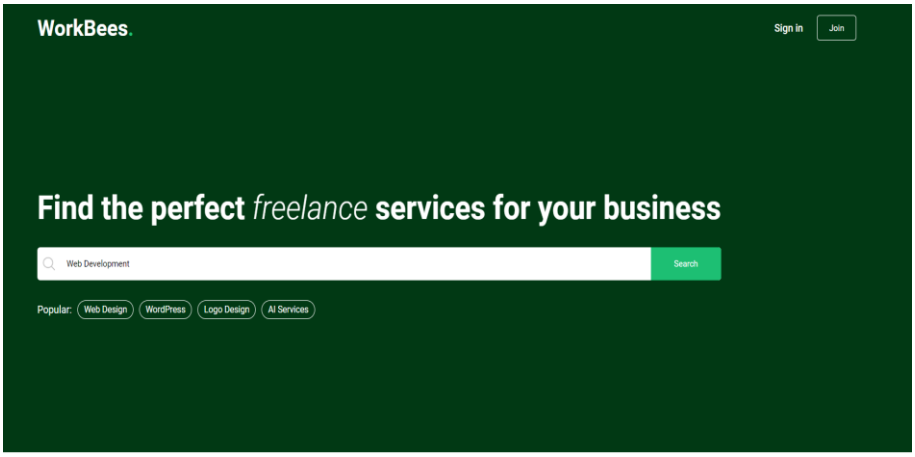
3.9.1 Registration Screen:



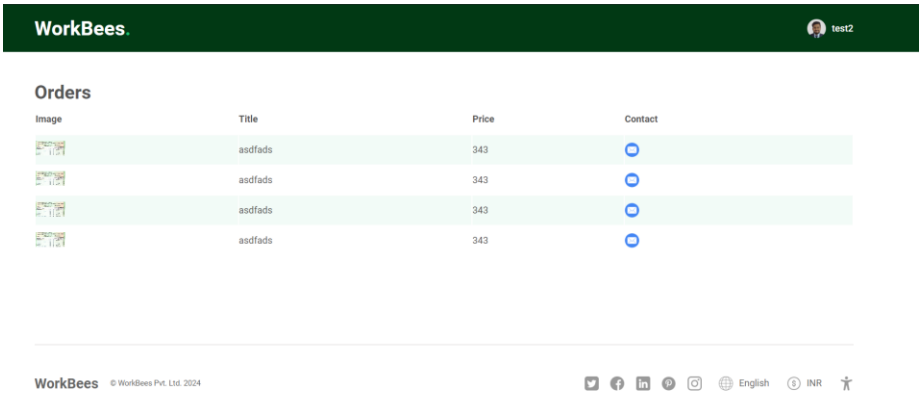
3.9.2 Login Screen:



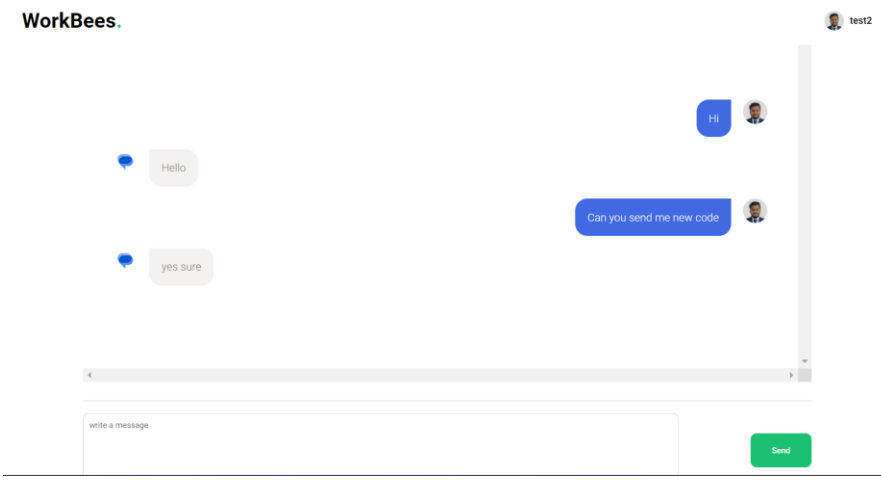
3.9.3 Index Screen:



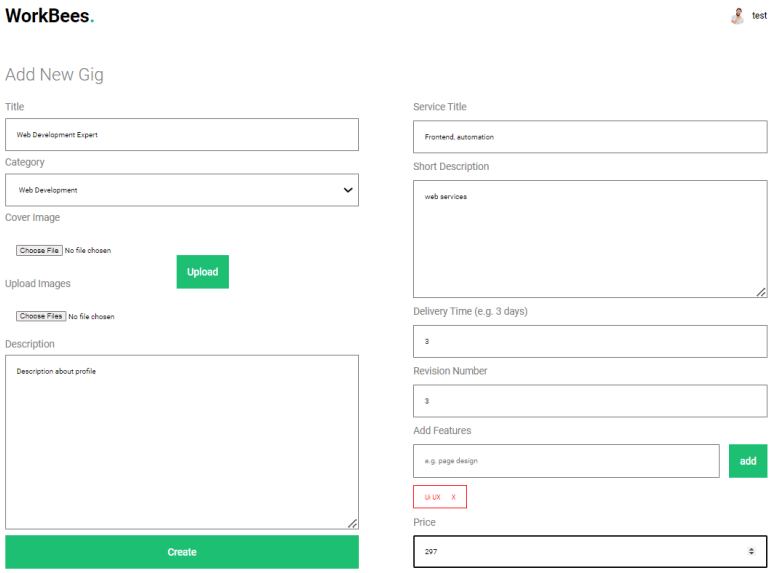
3.9.4 Orders Screen:



3.9.5 Message Screen:



3.9.6 Add New Gig Screen:



4.1 Algorithms:

4.1.1 Messaging system:

a. When a user sends a message.

b. Verify the sender and recipient's identities and permissions.

c. Store the message in the database, including metadata such as timestamps and message status.

d. Update the conversation thread with the new message.

e. Notify the recipient about the new message if they are online.

4.1.2 Payment processing:

a. When a client awards a job to a freelancer:

b. Set up a stripe account for the project funds.

c. Deduct the agreed-upon project fees from the client's account and hold them in escrow.

d. Release funds for the completed milestone to the freelancer.

e. Deduct any platform fees or taxes from the released funds.

f. Transfer the funds to the freelancer's account.

4.1.3 User registration algorithm:

a. Collect user-provided registration data such as username, email, password, and role (freelancer or client).

b. Validate the provided data for completeness and format adherence.

c. Check if the provided email is unique and not already registered in the system.

d. If validation passes and the email is unique, create a new user account with the provided information.

e. Generate a unique verification token and save that token for future authentication.

4.1.4 Review system algorithm:

a. Upon completion of project by freelancer user can give review and rating.

b. Collect review data such as ratings (e.g., stars) and written feedback from user.

c. Calculate an overall rating for each freelancer based on the received reviews.

d. Update the freelancer profiles with the received ratings and feedback.

4.2 Code snippets:

4.2.1 Routing code:

```
import "./app.scss";

import { createBrowserRouter, Outlet, RouterProvider } from "react-router-dom";

import React from "react";

import Navbar from "./components/navbar/Navbar";

import Footer from "./components/footer/Footer";

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

import Gigs from "./pages/gigs/Gigs";

import Gig from "./pages/gig/Gig";

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

import Register from "./pages/register/Register";

import Add from "./pages/add/Add";

import Orders from "./pages/orders/Orders";

import Messages from "./pages/messages/Messages";

import Message from "./pages/message/Message";

import MyGigs from "./pages/myGigs/MyGigs";

import {
```

```
  QueryClient,
  QueryClientProvider,
} from "@tanstack/react-query";
import Pay from "./pages/pay/Pay";
import Success from "./pages/success/Success";
function App() {
  const queryClient = new QueryClient();

  const Layout = () => {
    return (
      <div className="app">
        <QueryClientProvider client={queryClient}>
          <Navbar />
          <Outlet />
          <Footer />
        </QueryClientProvider>
      </div>
    );
```

```
};


const router = createBrowserRouter([
  {
    path: "/",
    element: <Layout />,
    children: [
      {
        path: "/",
        element: <Home />,
      },
      {
        path: "/gigs",
        element: <Gigs />,
      },
      {
        path: "/myGigs",
        element: <MyGigs />,
```

```
},
{
  path: "/orders",
  element: <Orders />,
},
{
  path: "/messages",
  element: <Messages />,
},
{
  path: "/message/:id",
  element: <Message />,
},
{
  path: "/add",
  element: <Add />,
},
{
```

```
  path: "/gig/:id",
  element: <Gig />,
},
{
  path: "/register",
  element: <Register />,
},
{
  path: "/login",
  element: <Login />,
},
{
  path: "/pay/:id",
  element: <Pay />,
},
{
  path: "/success",
  element: <Success />,
```

```
      },
    ],
  },
]);

 return <RouterProvider router={router} />;
}

export default App;
```

4.2.2 Gig creation code (Frontend):

```
import React from "react";
import { Link } from "react-router-dom";
import "./MyGigs.scss";
import getCurrentUser from "../../utils/getCurrentUser";
import { useMutation, useQuery, useQueryClient } from "@tanstack/react-query";
import newRequest from "../../utils/newRequest";
```

```
function MyGigs() {
  const currentUser = getCurrentUser();


  const queryClient = useQueryClient();
  console.log(currentUser);
  const { isLoading, error, data } = useQuery({
    queryKey: ["myGigs"],
    queryFn: () =>

newRequest.get(`/gigs?userId=${currentUser._id}`).then((res) =>
{
        return res.data;
      }),
  });
  console.log(data)
  const mutation = useMutation({
    mutationFn: (id) => {
      return newRequest.delete(`/gigs/${id}`);
    },
```

```
  onSuccess: () => {
    queryClient.invalidateQueries(["myGigs"]);
  },
});

const handleDelete = (id) => {
  mutation.mutate(id);
};

return (
  <div className="myGigs">
    {isLoading ? (
      "loading"
    ) : error ? (
      "error"
    ) : (
      <div className="container">
        <div className="title">
```

```
  <h1>Gigs</h1>
  {currentUser.isSeller && (
    <Link to="/add">
      <button>Add New Gig</button>
    </Link>
  )}
</div>
<table>
  <tr>
    <th>Image</th>
    <th>Title</th>
    <th>Price</th>
    <th>Sales</th>
    <th>Action</th>
  </tr>
  {data.map((gig) => (
    <tr key={gig._id}>
      <td>
```

```
            <img className="image" src={gig.cover} alt="" />
          </td>
          <td>{gig.title}</td>
          <td>{gig.price}</td>
          <td>{gig.sales}</td>
          <td>
            <img
              className="delete"
              src="./img/delete.png"
              alt=""
              onClick={() => handleDelete(gig._id)}
            />
          </td>
        </tr>
      ))}
    </table>
  </div>
)}
```

```
    </div>
  );
}

export default MyGigs;
```

4.2.3 Routing code (Backend):

```
import express from "express";
import mongoose from "mongoose";
import dotenv from "dotenv";
import userRoute from "./routes/user.route.js";
import gigRoute from "./routes/gig.route.js";
import orderRoute from "./routes/order.route.js";
import conversationRoute from "./routes/conversation.route.js";
import messageRoute from "./routes/message.route.js";
import reviewRoute from "./routes/review.route.js";
import authRoute from "./routes/auth.route.js";
```

```
import cookieParser from "cookie-parser";
import cors from "cors";


const app = express();
dotenv.config();
mongoose.set("strictQuery", true);
const connect = async () => {
  try {
    await
mongoose.connect("mongodb+srv://krushna:12345@cluster0.wg
efire.mongodb.net/");
    console.log("Connected to mongoDB!");
  } catch (error) {
    console.log(error);
  }
};
app.use(cors({ origin: "http://localhost:5173", credentials: true
}));
app.use(express.json());
```

```
app.use(cookieParser());
app.use("/api/auth", authRoute);
app.use("/api/users", userRoute);
app.use("/api/gigs", gigRoute);
app.use("/api/orders", orderRoute);
app.use("/api/conversations", conversationRoute);
app.use("/api/messages", messageRoute);
app.use("/api/reviews", reviewRoute);
app.use((err, req, res, next) => {
  const errorStatus = err.status || 500;
  const errorMessage = err.message || "Something went wrong!";
  return res.status(errorStatus).send(errorMessage);
});

app.listen(8800, () => {
  connect();
  console.log("Backend server is running!");
});
```

5.1 Test strategy:

5.1.1 Scope and Objectives:

- Ensure comprehensive test coverage for all key functionalities including user registration, profile management, service listings, messaging, payment processing, and review system.

- Verify platform usability, performance, security, and scalability under various scenarios.

- Validate compatibility across different devices, browsers, and screen sizes.

5.1.2 Test Environment Setup:

- Establish test environments mirroring production settings, including databases, servers, and third-party integrations.

- Utilize a combination of real and simulated user data to mimic diverse usage patterns and scenarios.

- Implement continuous integration and deployment (CI/CD) pipelines for automated testing and deployment.

5.1.3 Testing Types:

a. Functional Testing:

- Perform end-to-end testing of all user flows including registration, profile setup, service listing, messaging, payment processing, and review submission.

- Conduct regression testing to ensure new features and enhancements do not adversely impact existing functionality.

b. Usability Testing:

- Enlist real users or usability testers to evaluate the platform's ease of use, navigation, and overall user experience.

- Gather feedback to identify pain points, usability issues, and areas for improvement.

c. Performance Testing:

- Conduct load testing to assess system performance under expected and peak loads.

- Measure response times, throughput, and resource utilization to identify bottlenecks and optimize system performance.

d. Security Testing:

- Perform penetration testing to identify vulnerabilities and security flaws.

- Verify compliance with security standards and protocols to safeguard user data and financial transactions.

e. Compatibility Testing:

- Validate platform functionality across different browsers (e.g., Chrome, Firefox, Safari) and devices (e.g., desktop, mobile, tablet).

- Ensure responsive design and layout adaptability to various screen sizes and resolutions.

5.1.4 Test Cases and Scenarios:

- Develop test cases covering positive and negative scenarios for each functionality.

- Include edge cases, boundary conditions, and error handling scenarios to validate robustness and resilience.

- Test various user roles (freelancer, client, admin) and permissions to ensure proper access control and security enforcement.

5.1.5 Test Execution and Reporting:

- Execute test cases systematically, documenting test results, observations, and any identified defects.

- Prioritize and triage defects based on severity and impact on system functionality.

- Provide regular status updates and test reports to stakeholders, highlighting progress, findings, and recommendations for improvement.

5.1.6 User Acceptance Testing (UAT):

- Coordinate with stakeholders to conduct UAT sessions, allowing them to validate system functionality and provide feedback.

- Address any reported issues and incorporate necessary changes based on user feedback before final deployment.

5.1.7 Regression Testing and Maintenance:

- Establish regression test suites to be executed after each deployment or system update.

- Continuously monitor and maintain test cases, updating them to reflect changes in requirements or system behaviour.

- Incorporate feedback from production usage to iteratively improve the test strategy and ensure ongoing quality assurance.

By following this comprehensive test strategy, the WorkBees platform can undergo thorough testing to validate its functionality, performance, security, and usability, ensuring a high-quality user experience and system reliability.

5.2 Unit test plan:

5.2.1. Objective:

The primary objective of unit testing is to validate the functionality of individual components or units of code in isolation to ensure they meet the specified requirements and behave as expected.

5.2.2. Scope:

- User registration and authentication

- Profile management

- Messaging system

- Payment processing

- Review system

5.2.3. Testing Tools:

- Utilize testing frameworks compatible with the technology stack of the WorkBees platform, such as Jest for JavaScript/Node.js.

5.2.4. Test Cases:

- Develop unit test cases for each function or method within the targeted components, covering various scenarios including:

- Valid input data

- Invalid input data

- Edge cases and boundary conditions

- Error handling and exception paths

5.2.5. Test Environment:

- Set up a dedicated test environment isolated from the production environment to run unit tests.

- Configure mock dependencies or stubs to simulate external interactions (e.g., database calls, API requests).

5.2.6. Test Execution:

- Execute unit tests using automated testing frameworks to validate the behavior of individual code units.

- Monitor test results for pass/fail status and capture any encountered errors or failures.

5.2.7. Test Coverage:

- Aim for high test coverage by ensuring that each function or method within the targeted components has corresponding unit test coverage.

- Strive to cover all execution paths, branches, and logic branches within the code.

5.2.8. Documentation:

- Document unit test cases, including descriptions, inputs, expected outputs, and preconditions.

- Maintain documentation on test coverage metrics, including the percentage of code covered by unit tests.

5.3 Acceptance test plan:

5.3.1. Objective:

- The objective of the acceptance testing is to verify that the WorkBees platform meets the specified requirements and satisfies the needs of stakeholders.

5.3.2. Scope:

- Acceptance testing will cover end-to-end scenarios and user workflows within the WorkBees platform, including:

  - User registration and authentication

  - Profile setup and management

  - Service listing and job posting

  - Messaging and communication

  - Payment processing

  - Review submission and rating

5.3.3. Test Cases:

- Develop acceptance test cases based on user stories and requirements documentation, covering key functionalities and user interactions.

- Include positive and negative scenarios, edge cases, and boundary conditions to ensure thorough validation.

5.3.4. Test Environment:

- Utilize a staging or pre-production environment that closely resembles the production environment to conduct acceptance testing.

- Ensure availability of realistic test data and configurations to simulate real-world usage scenarios.

5.3.5. Test Execution:

- Execute acceptance tests manually or using automated testing tools, depending on the complexity and nature of the test cases.

- Follow predefined test scripts and scenarios to guide the testing process and ensure consistency.

5.3.6. Functional Coverage:

- Ensure that acceptance tests cover all critical functionalities and user journeys within the WorkBees platform, including both primary and secondary use cases.

5.3.7. User Experience Testing:

- Evaluate the user experience and interface design of the platform, focusing on usability, intuitiveness, and accessibility.

- Solicit feedback from representative users or stakeholders to identify any usability issues or areas for improvement.

5.3.8. Integration with Stakeholders:

- Collaborate closely with stakeholders, including end users, product owners, and business analysts, to validate requirements and expectations.

- Conduct walkthroughs or demonstrations of acceptance tests to ensure alignment with stakeholder needs.

5.3.9. Regression Testing:

- Perform regression testing to verify that new features or changes do not adversely impact existing functionality.

- Re-run acceptance tests on updated builds or versions of the

platform to validate system stability and reliability.

5.3.10. Defect Management:

- Document and track any defects or issues identified during acceptance testing using a dedicated defect tracking system.

- Prioritize and address defects based on severity and impact on system functionality and user experience.

5.3.11. Sign-off and Approval:

- Obtain sign-off and approval from relevant stakeholders once acceptance testing is complete and all requirements have been verified satisfactorily.

- Ensure that stakeholders are satisfied with the overall quality and performance of the WorkBees platform before proceeding to production deployment.

5.4 Test Case / Test Script:

5.4.1 Landing Page Test Case:

Testcase ID	Description	Expected Result	Actual Result	Status
TC 01	Verify that the landing page loads without errors.	The landing page should loads successfully and all elements should display correctly.	The landing page is loading successfully and all elements are displaying correctly.	Pass
TC 02	Verify functionality of navigation links on the landing page.	All links should navigate to the intended pages without any issues.	All links are navigating to the intended pages without any issues.	Pass
TC 03	Verify “WorkBees” logo.	Logo should be displayed on top left corner.	Logo is displaying on top left corner.	Pass
TC 04	Verify “WorkBees”	User should be redirected	User is redirected to	Pass

	logo.	to home page upon clicking on logo.	home page upon clicking on logo.	
TC 05	Verify Join button.	User should be redirected to registration page upon clicking on Join button.	User is redirected to registration page upon clicking on Join button.	Pass
TC 06	Verify footer section.	Footer section should be displayed at bottom of page.	Footer section is displaying at bottom of page.	Pass
TC 07	Verify category carousel.	Multiple gig categories should displayed on category carousel cards.	Multiple gig categories are displaying on category carousel cards.	Pass
TC 08	Verify category card.	User should be redirected to particular category gigs page upon	User is redirected to particular category gigs page upon	Pass

		clicking on category card.	clicking on category card.	
TC 09	Verify Sign in button.	User should be redirected to Sign in page.	User is redirected to Sign in page.	Pass

5.4.2 Registration/ Login Page Test Case:

TestCase ID	Description	Expected Result	Actual Result	Status
TC 10	Verify user registration process.	Users should able to register successfully with valid details.	Users is successfully registered with valid details.	Pass
TC 11	Validate the user login process.	Users should able to log in with their registered credentials.	Users is able to log in with their registered credentials.	Pass

TC 12	Verify user registration process.(Invalid credentials)	Users should not able to register successfully with invalid details.	Users is not able registered with invalid details.	Pass
TC 13	Verify login functionality with wrong username.	User should not able to login with wrong username.	User is not able to login with wrong username.	Pass
TC 14	Verify login functionality with wrong password.	User should not able to login with wrong password.	User is not able to login with wrong password.	Pass

5.4.3 Profile Section Test Case:

TestCase ID	Description	Expected Result	Actual Result	Status
TC 15	Test the functionality to upload and manage profile image.	User should able to upload profile image successfully.	User successfully uploaded profile image.	Pass
TC 16	Verify that freelancers can create gigs with accurate descriptions, pricing, and delivery timeframes.	Freelancer should able to create gig with accurate information.	Freelancer is able create gig.	Pass

5.4.4 Payment Processing Page Test Case:

TestCase ID	Description	Expected Result	Actual Result	Status
TC 17	Verify that user can make payments securely using Stripe payment gateway.	User should able to make payment.	User is able to make payment.	Pass
TC 18	Verify user can make payment within time limit.	User should able make payment within time limit.	User is able to make payment within time limit.	Pass

5.4.5 Review and Rating:

TestCase ID	Description	Expected Result	Actual Result	Status
TC 19	Validate that users can submit reviews and ratings upon project completion.	User should able to submit review and rating.	User is able to submit review and rating.	Pass
TC 20	Ensure that reviews and ratings are displayed accurately on user profiles and project listings.	Review and rating should be displayed correctly.	Review and rating displayed correctly.	Pass

5.4.6 My Gigs page:

TestCase ID	Description	Expected Result	Actual Result	Status
TC 21	Validate MyGigs page.	Freelancer can view their uploaded gigs.	Freelancer is able to view their uploaded gigs.	Pass
TC 22	Validate “Add New Gig” tab.	Freelancer should be redirected to add new gig page.	Freelancer is redirected to add new gig page.	Pass
TC 23	Validate image upload functionality.	Image should be uploaded and display in image section.	Image is uploaded and displaying in image section.	Pass

5.4.7 Category page:

TestCase ID	Description	Expected Result	Actual Result	Status
TC 24	Validate Budget sorting functionality.	User should able to sort gigs as per budget.	User is able to sort gigs as per budget.	Pass
TC 25	Verify sorting functionality.	User should able to sort gigs as per Best selling.	User is able to sort gigs as per Best selling.	Pass

5.5 Defect Report/ Test Log:

Defect ID: 01

Module / Feature: Payment and Ratings

Description: Users are unable to modify their submitted ratings for completed tasks.

Steps to Reproduce:

1. Complete a task assigned by another user.
2. Navigate to the payment section for the completed task.
3. Make the payment successfully.
4. Submit a rating for the completed task.
5. Attempt to modify the submitted rating.

Expected Behaviour: After submitting a rating, users should have the ability to modify their rating for the completed task if they change their opinion or find the initially submitted rating inaccurate.

Actual Behaviour: Users do not have the option to modify their submitted ratings once they have been entered into the system.

Status: Open

Defect ID: 02

Module / Feature: Overall System Performance

Description: Users experience significant latency when navigating between pages and performing actions within the Workbees platform.

Steps to Reproduce:

1. Log in to the Workbees platform.

2. Navigate between different sections/modules, such as tasks, messaging, user profile, etc.

3. Observe the time it takes for each page to load and for actions to be processed.

Expected Behaviour: Users should experience smooth and responsive navigation and actions within the platform, with minimal delay in page loading and action execution.

Actual Behaviour: There is noticeable latency when navigating between pages and performing actions, resulting in a sluggish user experience.

Status: Open

6.1. Dependency on Internet Connectivity: WorkBees relies heavily on internet connectivity. Users may face limitations or interruptions in accessing the platform's services if they have poor or no internet connection.

6.2. Platform Stability: The stability of the platform may be affected by factors such as server downtime, maintenance, or technical issues. Users may experience disruptions in service availability during such periods.

6.3. Security Concerns: Despite implementing security measures, the platform may still be vulnerable to cyber threats such as hacking, data breaches, or unauthorized access. Ensuring robust security protocols and continuous monitoring is essential to mitigate these risks.

6.4. Limited Feature Set: Initially, the platform may lack certain advanced features or functionalities compared to established freelance marketplaces. Users may find the feature set limiting, affecting their overall experience and satisfaction with the platform.

6.5. Scalability Challenges: As the user base grows, scalability challenges may arise in terms of handling increased traffic, user interactions, and data volume. Ensuring scalability and performance optimization becomes crucial to accommodate growth without compromising user experience.

6.6. User Adoption and Engagement: Encouraging user adoption and engagement may pose a challenge, particularly in competitive markets. Convincing freelancers and clients to use the platform over established alternatives requires effective marketing and incentive strategies.

6.7. Customer Support: Providing effective customer support to address user inquiries, issues, and complaints in a timely manner may be challenging, especially during peak periods or unforeseen circumstances.

6.8. Technological Evolution: Rapid advancements in technology and changes in market dynamics may render certain platform features obsolete or less competitive over time. Continuous innovation and adaptation are necessary to stay relevant in the dynamic freelance marketplace.

7.1. Enhanced User Experience (UX): Improve the platform's usability and intuitiveness through a user-centric design approach. Implement intuitive navigation, streamlined workflows, and personalized user experiences to enhance user satisfaction and engagement.

7.2. Mobile Application Development: Develop dedicated mobile applications for iOS and Android platforms to provide users with seamless access to WorkBees services on mobile devices. Optimize the mobile experience with responsive design, push notifications, and native functionalities to enhance user engagement and accessibility.

7.3. Real-time Collaboration Tools: Integrate real-time collaboration tools such as video conferencing, screen sharing, and virtual whiteboards to facilitate seamless communication and collaboration between clients and freelancers. Enhance project management capabilities and productivity by enabling real-time interaction and feedback exchange.

7.4. AI-Powered Recommendations: Implement AI-driven recommendation engines to provide personalized service recommendations and freelancer matches based on user preferences, past interactions, and project requirements. Leverage machine learning algorithms to continuously refine and improve r

7.5. Analytics and Insights Dashboard: Develop comprehensive analytics and insights dashboards to provide users with actionable data and insights into their projects, performance metrics, and market trends. Empower users to make informed decisions and optimize their freelance activities through data-driven insights and recommendations.

7.6. Community Engagement Features: Foster a sense of community and collaboration among users by introducing community engagement features such as forums, discussion boards, and knowledge-sharing platforms. Encourage users to connect, network, and share expertise within the WorkBees community.

7.7. Continuous Performance Optimization: Invest in continuous performance optimization efforts to enhance platform scalability, reliability, and responsiveness. Conduct regular performance audits, optimize codebase, and leverage caching and load balancing techniques to ensure optimal platform performance under varying workloads.

By implementing these proposed enhancements, the WorkBees platform can elevate its functionality, user experience, and competitive advantage in the freelance marketplace, attracting more users and fostering long-term growth and success.

WorkBees platform represents a promising solution aimed at bridging the gap between freelancers and clients seeking professional services. With its focus on seamless collaboration, diverse service offerings, and user-friendly interface, WorkBees has the potential to revolutionize the freelance marketplace.

Throughout the project, thorough analysis, planning, and implementation have been undertaken to ensure the platform meets the needs and expectations of its users. From the selection of the MERN stack for robust development to the integration of stripe payment gateway, every aspect has been carefully considered to deliver a comprehensive and innovative solution.

However, it's important to acknowledge that the success of WorkBees hinges not only on its technological prowess but also on factors such as user adoption, market dynamics, and continuous improvement. Therefore, ongoing efforts in marketing, user engagement, and enhancement of platform features will be essential to drive adoption and maintain competitiveness in the ever-evolving freelance landscape.

Overall, WorkBees holds significant promise to simplify the process of hiring freelancers and managing projects while providing a platform for freelancers to showcase their skills and connect with potential clients. With dedication, innovation, and a commitment to delivering value to its users, WorkBees is poised to make a meaningful impact in the world of freelance work.