Service-sphere documentation

Developed By: Eng. Menna A. Hassan

Project Type: Web Application.
Tech Stack: React.js | .NET Core | MSSQL | AI Integration.

Abstract

ServiceSphere is a cutting-edge freelancing platform that revolutionizes project management by empowering clients to initiate and manage complex projects requiring collaborative teams, leveraging advanced AI algorithms. Unlike traditional platforms limited to single-task assignments, ServiceSphere automates project milestone generation and optimizes team assembly based on client specifications.

Key features include secure JWT-authenticated login, detailed freelancer profiles, and advanced job filtering for streamlined navigation. AI-driven enhancements refine job descriptions and automate milestone planning, ensuring clarity and efficiency throughout project lifecycles. Real-time notifications via SignalR and seamless communication through WhatsApp integration foster dynamic collaboration, complemented by secure payment processing with Stripe.

ServiceSphere's architecture, combining React for frontend responsiveness and .NET for backend robustness, ensures scalability and performance. By bridging the gap between project complexity and efficient management, ServiceSphere sets a new standard in freelance platforms, enhancing productivity and client satisfaction in the global freelance marketplace.

Table of Contents

Contents	Page#
List of Figures	8
List of Tables	12
List of Abbreviations	13
1. Introduction	15
1.1 Introduction	16
1.2 Problem Statement and Motivation	17
1.3 Project Objectives	18
1.4 Thesis Organization	19
2. Related Work	20
2.1 Background	21
2.2 Types of applications	22
2.3 Proposed Modifications	24
2.4 Comparisons of used development tools	25
2.5 Summary	28
3. Methodologies and Tools	29
3.1 Design Methods	30

3.1.1 User-Centric Design 30

3.1.2 Agile Methodology 30

3.1.3 Design Thinking 31

3.2 Development Tools 31

3.2.1 Web and Mobile-based Tools 30

3.1.1.1 Frontend Development Tools 33

3.1.1.2 Backend Development Tools 34

3.1.1.3 Mobile Development Tools 37

3.2.2 AI and Machine Learning Tools 38

3.3 Project Management 40

3.3.1 Agile and Sprints 40

3.3.2 Version Control 40

4. System analysis & Design 42

4.1 Requirements Gathering 43

4.2 System Scope 45

4.3 User Story Definitions 51

4.4 System Design 55

4.4.1 System Architecture 55

4.4.2 Client-Side Components 55

4.4.3 Backend Components 56

4.5 System UI 58

4.5.1 Mobile UI Flow 60

4.5.2 Web UI Flow 64

4.6 Database ERD 68

4.7 Summary 69

5. System Implementation 70

5.1 Code Implementation 71

5.1.1 Frontend Development 71

5.1.2 Backend Development 81

5.2 AI Integration with GPT-3.5-turbo-instruct 89

5.3 Web Basic Version 93

5.3.1 Frontend Web Development 93

5.3.2 Backend Web Development 99

5.4 Web Improved Features 103

5.4.1 Frontend Advanced features 103

5.4.2 Backend Advanced features 105

5.5 Mobile App Development 107

5.5.1 Code Implementation 113

5.5.2 Mobile Interface 119

6. Testing and Evaluation 124

6.1 Functional Testing 125

6.2 User Experience Testing 131

6.3 How ServiceSphere Meets Evaluation Criteria 135

6.3.1 Functional Requirements Testing Findings 135

6.3.2 Non-Functional Requirements Testing Findings 137

6.4 Limitations and Potential Enhancements 138

7. Conclusions and Future Work 141

7.1 Conclusion 142

7.2 Future Work 144

References 148

List of Figures

Figure 1: Upwork website Interface

FIgure 2: Fiver Website Interface

Figure 3: Freelancer Website Interface

Figure 4: ReactJS Logo

Figure 5: .NET Core Logo

Figure 6: Flutter Logo

Figure 7: SQL Server

Figure 8: Figma Logo

Figure 9: Microsoft Visual Studio Code

Figure 10: SignalR Logo

Figure 11: MonsterASP Logo

Figure 12: Postman Logo

Figure 13: OpenAI Logo

Figure 14:login and register mobile flow (UI)

Figure 15:Messaging Mobile UI

Figure 16:Login Webpage UI

Figure 17:Home WebPage UI

Figure18:Project Webpage UI

Figure19:Client Home Webpage UI

Figure20:Freelancer Home Webpage UI

Figure21:ERD

Figure22: ServiceSphere Backend Directory Structure

Figure23:Freelancer Entity

Figure24:Application Layer

Figure25:Generic Repository

Figure26:API Key Creation

Figure27:Text Refinement code in C#

Figure28:Milestones Code Part 1

Figure29:Milestones Code part 2

Figure30:TeamAssembly Code part2

Figure31:TeamAssembly Code part1

Figure32:Login

Figure33:Register

Figure34:Freelancer Profile

Figure35:Profile step 1

Figure36:Profile step 2

Figure37:Profile step 3

Figure38:Profile step 4

Figure39:Profile step 5

Figure40:Submit Proposal

Figure41:Post

Figure42:All Proposals

Figure43:Contract

Figure44:Register

Figure45:Create Profile

Figure46:Service Post Endpoint

Figure47:project post endpoint

Figure48:proposal submit

Figure49:contract

Figure50:Review Endpoint

Figure51:AI Match

Figure52:Notification

Figure53:Stripe Payment

Figure54:Description Refinement

Figure55:Refinement text endpoint

Figure56:AI Refinement endpoint

Figure57:Notification endpoint

Figure58:Payment endpoint

Figure59:Register Model

Figure60:loginModel

Figure61:Payment

Figure62:Delete Post

Figure63:Delete Post

Figure64:Edit Post

Figure65:Get All Posts

Figure66:Talents

Figure67:Proposals

Figure68:Get All Proposals

Figure69:Submit Proposal

Figure70:Login

Figure71:SignUp

Figure72:Login-Register

Figure73:Filling Info

List of Tables

Table 1: Project Root (ServiceSphere.Apis)

Table 2: Root Directory

Table 3: Controllers Directory

Table 4: wwwroot Directory

Table 5: Migrations Directory

List of Abbreviations

AI: Artificial Intelligence

API: Application Programming Interface

SDLC: System Development Life Cycle

UCD: User-Centered Design

UML: Unified Modeling Language

ORM: Object-Relational Mapper

IDE: Integrated Development Environment

JWT: JSON Web Token

WCAG: Web Content Accessibility Guidelines

CRUD: Create, Read, Update, Delete

HTML: HyperText Markup Language

CSS: Cascading Style Sheets

Sass: Syntactically Awesome Style Sheets

UI: User Interface

UX: User Experience

MSSQL: Microsoft SQL Server

EF Core: Entity Framework Core

WCAG: Web Content Accessibility Guidelines

GDPR: General Data Protection Regulation

NLP: Natural Language Processing

Chapter 1
Introduction

1.Introduction

The freelance market is experiencing unprecedented growth, fueled by the global shift towards remote work and the increasing reliance on specialized skills from a diverse pool of talent worldwide. This expansion highlights the need for robust platforms that can efficiently manage freelance projects, ensuring secure transactions, effective communication channels, and streamlined project workflows. As businesses and individuals increasingly turn to freelancers for specialized tasks and projects, the demand for platforms that can facilitate seamless collaboration and project management has never been more pronounced.

In response to these evolving market demands, ServiceSphere emerges as a pivotal solution. It offers a comprehensive platform designed to address the complexities of freelance project management with advanced features and technologies. ServiceSphere integrates AI-driven project management capabilities to automate tasks such as milestone generation and team assembly, thereby optimizing project efficiency and reducing administrative burdens. Secure payment processing through trusted services like Stripe ensures reliable financial transactions, fostering trust and confidence among users.

Moreover, ServiceSphere prioritizes user experience by implementing intuitive interfaces, robust security measures such as JWT-authenticated login systems, and real-time communication tools like WhatsApp integration. These features not only enhance operational efficiency but also empower freelancers and clients to collaborate seamlessly across projects of varying scales and complexities.

By focusing on these core functionalities and leveraging modern technologies, ServiceSphere aims to establish new benchmarks in freelance project management and collaboration platforms. It seeks to provide freelancers and clients alike with a secure, user-friendly environment that supports innovation, productivity, and growth in the global freelance economy.

1.2 Problem Statement and Motivation

Despite the rapid growth of the freelance market, current platforms face significant challenges that hinder efficient project management and collaboration. Key issues include inadequate security measures in authentication processes, limited communication tools that hamper real-time interaction, and a lack of advanced project management features necessary for handling complex tasks and team-based projects. These shortcomings lead to concerns such as insecure transactions, difficulty in finding suitable projects through inefficient search and filtering tools, and a lack of AI-driven capabilities to streamline workflow processes.

ServiceSphere is motivated by these persistent challenges within the freelance industry. The platform aims to revolutionize freelance project management by providing a robust solution that addresses these critical gaps. By implementing secure JWT-authenticated login systems, integrating real-time communication tools like WhatsApp, and leveraging AI for automated project milestone generation and team assembly, ServiceSphere seeks to enhance security, efficiency, and collaboration across its user base.

Furthermore, ServiceSphere is designed to improve the overall freelance experience by offering a user-friendly interface, sophisticated job filtering mechanisms, and advanced AI-driven enhancements that refine job descriptions and optimize project workflows. By resolving these fundamental issues, ServiceSphere aims to set a new standard in freelance platforms, ensuring a more secure, streamlined, and productive environment for both freelancers and clients worldwide.

1.3 Thesis Objectives:

The primary objective of this thesis is to design and develop ServiceSphere, a pioneering platform that redefines freelance project management and collaboration. ServiceSphere aims to achieve the following specific objectives:

1.Effortless Project Management: ServiceSphere simplifies complex project management by automating milestone generation and team assembly using AI. This enables clients to undertake extensive projects without extensive managerial expertise, reducing costs significantly.

2.Enhanced Security: Implementing robust JWT-based authentication and advanced data protection ensures secure user credentials and a reliable login and registration process.

3.Comprehensive Profiles: ServiceSphere allows freelancers to create detailed profiles and enables clients to post a wide range of job opportunities.A sophisticated filtering system ensures efficient project matching.

4.Seamless Communication: Integrating platforms like WhatsApp and utilizing SignalR for real-time notifications ensures smooth client-freelancer communication throughout projects.

5.AI-Driven Efficiency: Leveraging AI to refine job descriptions, automate milestone planning, and assemble optimal project teams improves project efficiency and resource allocation.

ServiceSphere aims to set new industry standards by addressing current freelance market challenges and introducing innovative features that enhance productivity and collaboration.

1.4 Thesis Organization

The rest of this thesis is organized as follows:

Chapter 2 Related Work:

This chapter includes an Overview of existing freelance platforms, their features, limitations, and the technologies and methodologies informing ServiceSphere's development.

Chapter 3 : Methodologies and Tools

This chapter discusses the methodologies (e.g., Agile) and technologies used in ServiceSphere's development, emphasizing its technological stack and approach to meet market demands.

Chapter 4:System Analysis and Design

This chapter details the system architecture of ServiceSphere, including client-side and backend components, as well as additional features like notification systems, search and filtering mechanisms, and payment gateways.

Chapter 5: Implementation

This chapter describes the practical implementation of ServiceSphere, focusing on the development process, integration of technologies, and the challenges encountered during the development.

Chapter 6: Evaluation and Testing

This chapter covers the evaluation and testing of ServiceSphere, detailing the testing methodologies employed, the criteria for success, and the results of various tests. It also includes user feedback and performance metrics, providing a comprehensive assessment of the system's reliability, usability, and effectiveness.

Chapter 7: Conclusion and Future Work

This chapter concludes the thesis by summarizing the key findings and contributions of the research. It also outlines potential areas for future development and improvement in ServiceSphere.

References lists all sources cited throughout the thesis for further exploration and validation.

Chapter 2
Related Work

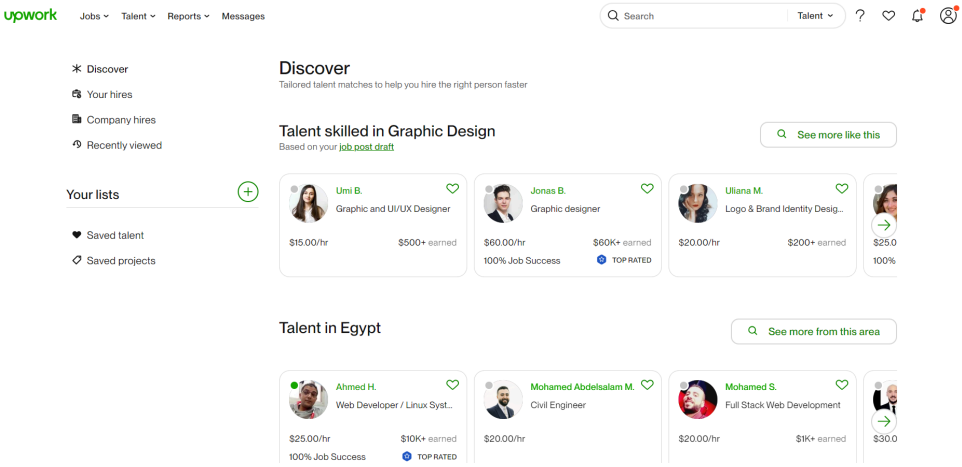
2.1 Background

The freelance economy has seen significant growth in recent years, driven by the rise of remote work and the increasing demand for specialized skills. Freelancers now make up a substantial portion of the global workforce, contributing to various industries such as technology, creative arts, marketing, and more. As a result, numerous platforms have emerged to facilitate freelance work, providing a marketplace where clients can find freelancers and manage projects effectively.

These platforms typically offer functionalities such as job postings, freelancer profiles, secure payment systems, and communication tools. However, many of these platforms still face challenges in ensuring security, providing advanced project management features, and facilitating seamless communication. ServiceSphere aims to address these gaps by integrating modern technologies such as AI, real-time notifications, and robust security measures, thus providing a more comprehensive solution for freelance project management.

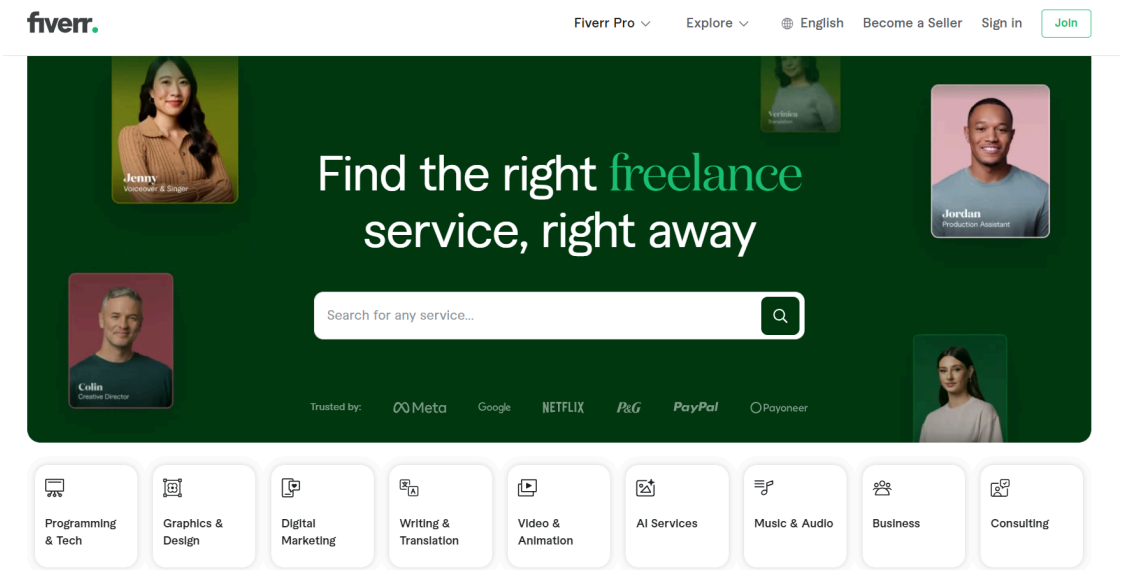
2.2 Types of Applications

1. **Upwork**: Provides job postings, freelancer profiles, and a payment system. Lacks advanced AI functionalities for project management and real-time communication integrations.



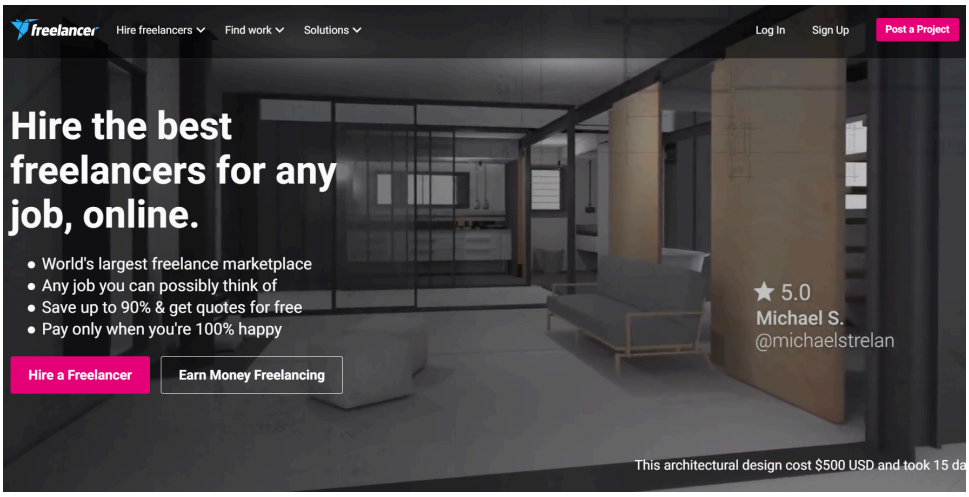
(figure 1:Upword Interface)

2. **Fiverr**: Focuses on quick gigs with a simple job posting and profile management system. Limited in handling larger, complex projects. (figure



2:fiver interface)

3. **Freelancer**: Offers a wide range of services but has a complex user interface and lacks advanced AI features for project milestone generation and team assembly.



(figure 3:Freelancer Interface)

2.3 Proposed Modifications:

ServiceSphere aims to differentiate itself by incorporating several advanced features not commonly found in existing platforms:

- **AI-Driven Project Management**:
    - **Text Refinement**: Use AI to enhance job descriptions for clarity and effectiveness.
    - **Milestone Generation**: Automatically generate project milestones to aid in planning and execution.
    - **Team Assembly**: Employ AI to form project teams based on specific requirements, optimizing resource allocation.

- **Advanced Communication Tools**:
    - Integrate WhatsApp for direct and immediate communication between clients and freelancers, enhancing flexibility and accessibility.
- **Enhanced Notification System**:
    - Implement real-time notifications using SignalR to keep users informed about project activities without needing to refresh their browsers or apps.
- **Secure and Efficient Payment System**:
    - Use Stripe for secure payment processing, ensuring funds are held until project milestones are met.

- **User-Friendly Interface**:
    - Develop a responsive and intuitive UI using React, ensuring ease of use across various devices and user categories.

By integrating these features, ServiceSphere aims to provide a more efficient, secure, and user-friendly platform for freelance collaboration and project

management, addressing gaps in the current market offerings and setting new standards for the industry.

2.4 Comparisons of Used Development Tools

ServiceSphere combines modern tools and technologies for robust functionality, security, and user experience. Here's a comparison with other platforms:

1. **Frontend Development**:
    - **React**: ServiceSphere utilizes React for building its dynamic and responsive user interfaces. React is known for its flexibility, performance, and component-based architecture, which allows for efficient state management and rendering. This makes it ideal for creating a user-friendly and interactive frontend.
    - **Angular**: Some other platforms, like Upwork, use Angular, a framework developed by Google. Angular provides a comprehensive solution for building dynamic web applications with features like two-way data binding and dependency injection.
    - **Vue.js**: Platforms such as Freelancer.com use Vue.js, a progressive JavaScript framework that offers a balance between the complexity of Angular and the simplicity of React. Vue.js is known for its ease of integration and flexibility.

2. **Backend Development**:
    - **.NET**: ServiceSphere employs .NET for its backend operations, leveraging its powerful performance capabilities and robust security features. .NET supports large-scale applications with complex business logic and offers extensive library support, making it a reliable choice for backend development.

- **Node.js**: Other platforms, like Fiverr, use Node.js, a JavaScript runtime built on Chrome's V8 engine. Node.js is known for its event-driven, non-blocking I/O model, which makes it efficient for handling concurrent operations.
- **Django**: Some platforms, like Toptal, use Django, a high-level Python framework that encourages rapid development and clean, pragmatic design. Django includes built-in features for security and scalability.

3. **Database Management**:
   - **SQL Server**: ServiceSphere uses SQL Server for its database management, known for its reliability, security features, and support for complex queries. SQL Server is well-suited for applications requiring robust data management capabilities.
   - **MongoDB**: Platforms like Upwork use MongoDB, a NoSQL database known for its flexibility and scalability. MongoDB uses a document-oriented data model, allowing for dynamic schema design.
   - **PostgreSQL**: Some platforms, like Freelancer.com, use PostgreSQL, an open-source relational database system known for its extensibility and standards compliance.

4.Authentication and Authorization:

4. **JWT (JSON Web Tokens)**: ServiceSphere implements JWT for secure authentication and authorization, providing a compact, URL-safe means of representing claims to be transferred between two parties. JWTs are widely used for securing API calls and managing user sessions.
5. **OAuth**: Other platforms use OAuth, an open standard for access delegation commonly used for token-based authentication. OAuth

provides secure and limited access to user resources without exposing credentials.

6. **Payment Integration**:
    - **Stripe**: ServiceSphere integrates Stripe for secure and efficient payment processing. Stripe is known for its ease of integration, security features, and support for a wide range of payment methods.
    - **PayPal**: Some platforms, like Fiverr, use PayPal, a global payment platform that offers secure payment processing and extensive user base. PayPal's integration is straightforward, making it a popular choice for many online services.

By leveraging these development tools, ServiceSphere aims to build a robust and user-friendly platform that addresses the current challenges in freelance project management. The combination of modern frontend and backend technologies, along with secure payment systems and advanced authentication methods, ensures that ServiceSphere can provide a comprehensive solution for freelancers and clients.

2.5 Summary

The freelance economy has grown significantly due to the rise of remote work and demand for specialized skills, leading to the emergence of numerous platforms for managing freelance projects. These platforms offer job postings, freelancer profiles, secure payments, and communication tools but often face challenges in security, advanced project management, and seamless communication. ServiceSphere aims to address these gaps by integrating modern technologies like AI-driven project management, real-time notifications, robust security measures, and user-friendly interfaces. It incorporates tools such as React, .NET, SQL Server, and Stripe to provide a comprehensive, efficient, and secure platform, setting new standards for freelance project management.

Chapter 3
Methodologies and Tools

3.1 Design Methods

In the development of ServiceSphere, a transformative freelance and project collaboration platform, we employed a range of design methods to ensure the system's robustness, user-friendliness, and scalability. The chosen methods facilitated structured planning, effective problem-solving, and iterative improvement throughout the project's lifecycle.

3.1.1 User-Centered Design (UCD)

- Definition: UCD is an iterative design process focusing on users and their needs in each phase of the design process. In this method, designers use various research and design techniques to create highly usable and accessible products.

- Application in ServiceSphere: We conducted extensive user and existing platforms research to understand the needs of clients, freelancers, and project teams. This included surveys, interviews, and usability testing. The insights gained guided the design of intuitive user interfaces and functionalities such as detailed user profiles, AI-powered matching, and real-time communication tools.

3.1.2 Agile Methodology

- Definition: Agile is an iterative and incremental approach to software development which emphasizes flexibility, continuous improvement, and rapid delivery of functional software.

- Application in ServiceSphere: The project was broken down into small, manageable sprints, each delivering a part of the overall functionality. Regular sprint reviews and retrospectives helped us incorporate feedback quickly, adjust plans, and improve the system iteratively. Tools like Scrum and Kanban were utilized to manage tasks and track progress.

3.1.3 Design Thinking

- Definition: Design Thinking is a solution-based approach to solving problems, focusing on understanding the human needs involved, re-framing the problem in human-centric ways, and iterating on ideas.

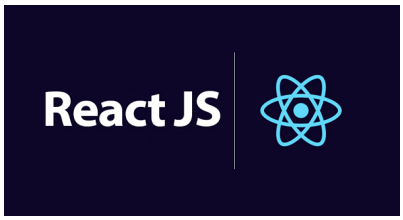
- Application in ServiceSphere: The design thinking process was employed to tackle complex problems like AI-powered team assembly, extensive project management, payment processing, Milestones generation and job postings. This involved empathizing with users, defining the problem, ideating solutions, creating prototypes, exploring different scenarios and testing them rigorously.

3.2 Development Tools

Selecting the right tools was crucial for efficiently developing a robust and scalable platform. Below is an overview of the tools utilized:

3.2.1 Web and Mobile-based Tools

- **React.js**: Used for building a responsive and interactive user interface. React.js allowed for the creation of dynamic and high-performance front-end components that enhanced user experience.



(figure 4:ReactJS logo)

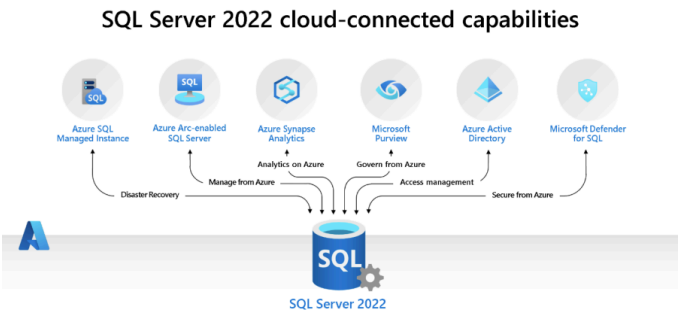
- **.NET**: Utilized for developing robust and secure back-end services. The .NET framework provided a comprehensive programming model and extensive API set, which facilitated the development of ServiceSphere APIs.

(figure 5:.NET Core logo)

- **Flutter**: Utilized for creating cross-platform mobile applications. Flutter's single codebase approach enabled the development of native-like apps for both iOS and Android, ensuring a consistent user experience across different devices.

(figure 6:Flutter logo)

- **MSSQL**: Selected for its reliable and efficient data management capabilities. MSSQL's powerful querying, transaction support, and data integrity features were essential for managing complex relational data and ensuring data consistency across the platform.



(figure 7: SQL Server)

- **Figma**: Used for UI/UX design and prototyping. Figma allowed for collaborative design efforts, enabling the creation of detailed and interactive prototypes that guided the development of the user interface.

(figure 8:Figma logo)

3.2.1.1 Front-End Development Tools

- **HTML**: Used for structuring the content on the web pages. HTML provided the foundational elements of the web interface.

- **CSS**: Employed for styling the HTML elements. CSS was essential for designing the layout, colors, fonts, and overall visual appearance of the platform.

- **Bootstrap**: Utilized for responsive design and pre-built UI components. Bootstrap's grid system and components helped in creating a consistent and mobile-friendly design quickly.

- **jQuery**: Used for simplifying JavaScript operations and enhancing interactivity. jQuery was employed for tasks such as DOM manipulation, event handling, and implementing animations.

- **Sass** (Syntactically Awesome Style Sheets): Used as a CSS preprocessor to add functionality to CSS such as variables, nested rules, and mixins, making the stylesheet more maintainable and efficient.

- **Redux**: Utilized for state management. Redux provided a centralized store for the state of the application, enabling predictable state changes and easier debugging.

- **Axios**: Used for making HTTP requests to interact with backend APIs. Axios provided a promise-based HTTP client, simplifying the process of making asynchronous requests and handling responses.

- **React Router**: Employed for enabling navigation among different views or components in a React application. React Router allowed for the creation of single-page applications with dynamic routing.

3.2.1.2 Backend Development Tools with .NET

Core Tools

- **.NET Core**: The framework used for developing the backend services. .NET Core provides a powerful, cross-platform runtime and framework for building high-performance, scalable web applications and APIs.

- **ASP.NET Core Web API**: Used for building HTTP services that can be consumed by a variety of clients, including browsers and mobile devices. ASP.NET Core Web API provides a robust framework for creating RESTful services.

- C#: The primary programming language used for developing backend services in .NET. C# offers a rich set of features and a powerful development environment for building scalable and maintainable applications.

- Microsoft Visual Studio: The integrated development environment (IDE) used for developing, debugging, and deploying .NET applications. Visual Studio provides a comprehensive set of tools and features that streamline the development process.

(figure 9:Microsoft Visual Studio logo)

- **SignalR:** Implemented for enabling real-time notification functionality. SignalR provides the capability to push updates from the server to the clients instantly, which is essential for the ServiceSphere notification system.

(figure 10:SignalR logo)

- **MonsterASP.NET:** Utilized as an external service/platform for hosting the web API project. MonsterASP.NET provides a robust and scalable environment for deploying and managing web applications. By leveraging its hosting capabilities, the development team ensured reliable uptime, efficient load handling, and secure access to the API services.

(figure 11:MonsterASP logo)

Data Management

- **MSSQL (Microsoft SQL Server)**: Chosen for its reliable and efficient data management capabilities. MSSQL provides powerful querying, transaction support, and data integrity features, essential for managing complex relational data.

- **Entity Framework Core**: An ORM (Object-Relational Mapper) used to interact with the database. Entity Framework Core simplifies data access by

allowing developers to work with data using .NET objects, eliminating the need for most of the data access code typically required.

Authentication and Authorization

- **JWT (JSON Web Tokens)**: Used for implementing stateless authentication. JWT provides a compact and self-contained way to securely transmit information between parties as a JSON object. It is widely used for authenticating users and securing API endpoints in ASP.NET Core applications.

- **ASP.NET Core Identity**: A membership system that adds login functionality to applications. ASP.NET Core Identity supports multi-factor authentication, external authentication providers, and more.

API Documentation and Testing

- **Swagger**: Utilized for generating interactive API documentation. Swagger provides a UI that allows developers and consumers to explore and test the API endpoints.

- **Postman**: Employed for testing and debugging API endpoints. Postman provides a user-friendly interface for making HTTP requests, inspecting responses, and ensuring that the API endpoints function correctly.

(figure 12:Postman logo)

3.2.1.2 Mobile Development Tools with .NET

In a Flutter project, developers typically use a combination of tools to streamline development, ensure code quality, and manage dependencies. Here are some commonly used development tools for Flutter projects:

1. **Flutter SDK:** The core software development kit provided by Google for building Flutter applications. It includes the Flutter framework, Dart SDK, and command-line tools.
2. **IDEs (Integrated Development Environments):**
    - **Android Studio / IntelliJ IDEA:** Officially recommended IDEs for Flutter development. They offer features like code completion, debugging, and integration with Flutter-specific tools.
    - **Visual Studio Code (VS Code):** Another popular choice with a Flutter extension that provides similar features to Android Studio/IntelliJ IDEA.
3. **Flutter Inspector:** A tool integrated into IDEs (like Android Studio and VS Code) that allows developers to visualize and explore the UI hierarchy of Flutter applications during runtime.
4. **Dart DevTools:** A suite of performance and debugging tools for Dart and Flutter. It includes tools for inspecting widgets, analyzing performance, and debugging network requests.
5. **FlutterFire:** A set of Flutter plugins that enable Flutter apps to use Firebase services. This includes plugins for Firestore, Authentication, Cloud Functions, and more.
6. **Flutter Packages:** The Dart ecosystem includes a package manager (pub) that hosts thousands of packages for various functionalities. Developers often leverage these packages for features like HTTP requests, state management, animations, etc.
7. **Git:** Version control system used for managing source code changes. Platforms like GitHub, GitLab, or Bitbucket are often used for collaborative development and code repository hosting.
8. **Continuous Integration / Continuous Deployment (CI/CD) Tools:** Jenkins, CircleCI, Travis CI, etc., are used to automate testing, building, and deploying Flutter applications.

9. **Third-Party Libraries:** Besides official Flutter packages, developers often use third-party libraries for specific functionalities not covered by the core Flutter framework or official packages.
10. **Emulators / Simulators**: For testing Flutter applications on different device configurations and operating systems. Android Emulator (for Android) and iOS Simulator (for iOS) are commonly used.

These tools collectively support the development lifecycle of Flutter applications, from initial coding to testing and deployment, ensuring efficient and effective development processes.

3.2.3 AI and Machine Learning Tools

AI Integration with GPT-3.5-turbo-instruct

- **OpenAI** is an AI research and deployment company dedicated to ensuring that artificial general intelligence (AGI) benefits all of humanity. OpenAI develops advanced machine learning models, including the GPT-3.5-turbo-instruct model, which is known for its ability to understand and generate human-like text based on the input it receives.

  -GPT-3.5-turbo-instruct: This version of the model enhances natural language processing capabilities, making it adept at various tasks such as text generation, refinement, and understanding complex queries.

(figure 13:OpenAI logo)

Design Patterns in ServiceSphere Backend

Repository Pattern: In ServiceSphere, The repository pattern was used in conjunction with Entity Framework Core to manage database operations. Generic repositories were created to handle CRUD operations for different entities, ensuring that data access logic is encapsulated within repository classes. This approach simplifies the interaction with the data layer and promotes code reuse.

Unit of Work Pattern: In ServiceSphere, This pattern was implemented to ensure data consistency and integrity during complex operations. By coordinating the work of multiple repositories, the Unit of Work ensures that all changes are committed together, or none at all, thereby maintaining a consistent state in the database.

Singleton Pattern: In ServiceSphere, This pattern was used for services that need to maintain a single state, such as configuration managers and logging services. By ensuring these services have only one instance, ServiceSphere avoids redundant resource usage and potential conflicts, providing a consistent and reliable service throughout the application.

Specification Pattern: In ServiceSphere, The Specification Pattern was employed to encapsulate the criteria used for querying the database. This approach makes the code more readable and maintainable by separating the query logic from the rest of the application. Specifications can be combined and reused, providing a flexible way to handle complex querying requirements.

Dependency Injection Pattern: In ServiceSphere, This pattern was widely used throughout the ASP.NET Core application. Services, repositories, and other dependencies were injected into controllers and other classes via constructors.

This approach made the code more modular and easier to test, as dependencies could be easily mocked or substituted in unit tests.

3.3 Project Management

Efficient project management practices were critical in ensuring the timely and successful delivery of ServiceSphere.

3.3.1 Agile and Sprints

- Sprint Planning: Defined clear objectives and deliverables for each sprint.

- Daily Standups: Conducted brief daily meetings to discuss progress, obstacles, and next steps.

- Sprint Reviews: Held at the end of each sprint to demonstrate completed work and gather stakeholder feedback.

- Retrospectives: Used to reflect on what went well and what could be improved for future sprints.

- Trello: Implemented for visual task management and tracking. Trello's board and card system provided an intuitive way to manage sprint tasks, track progress, and ensure that everyone on the team was aware of the status of different tasks and stories.

3.3.2 Version Control

- Git: Employed for version control to manage code changes, track history, and facilitate collaboration among the development team.

- GitHub: Used as a repository hosting service, providing tools for code review, issue tracking, and project management.

Incorporating these methods and tools ensured a user-centered, agile, and efficient development process for ServiceSphere. The strategic selection and application of design methods and development tools significantly contributed to creating a platform that meets user needs, adapts to changing requirements, and maintains high performance and scalability. This structured approach is exemplary for any large-scale software development project.

Chapter 4
System Analysis & Design

4.1 Requirements Gathering

The Requirements Gathering phase is crucial to ensure ServiceSphere meets stakeholder needs. We aimed to create a user-friendly platform for freelancers and clients in Construction & Renovation, Event Planning, Education, and Maintenance.

By analyzing market trends and interviewing clients and freelancers, we identified key challenges:

- Clients: Need for better project management tools.

- Freelancers: Importance of detailed profiles and collaborative tools.

These insights led to:

- Creation of service sub-categories.

- Implementation of an AI-driven team formation feature to enhance collaboration and project execution.

Research and Creation of ServiceSphere

Our research on freelancing in Egypt identified several key insights:

- Youthful Demographics and Digital Literacy: Egypt's young, educated population uses digital platforms for income but lacks visibility and tools to showcase skills.

- High Unemployment Rates: Many seek freelance opportunities due to the formal job market's limitations.

- Diverse Skill Set: Egyptian freelancers possess skills in industries like Engineering, Construction, Event Planning, and Education.

- Need for Comprehensive Platforms: Existing platforms lack efficient project management and collaboration tools, especially for large-scale projects.

Creation of ServiceSphere

ServiceSphere was created to address these needs with features such as detailed profiles, intuitive design, localized payment solutions, and AI-powered project management. This platform aims to enhance freelance collaboration and project management, benefiting the Egyptian freelance community and setting new industry standards.

Brainstorming

Initial Idea Development:

1. Identifying the Market Need:

- Problem: Reliance on Facebook groups for freelance services in Egypt lacks structure, reliability, and efficiency.

- Solution: Create a dedicated freelancing platform for various everyday needs, providing a more organized and user-friendly alternative.

2. Expanding the Concept:

- Large-Scale Projects: Envision a system to handle extensive projects like renovations, forming dedicated teams to coordinate various freelancers seamlessly.

Feature Elicitation and System Design:

1. Feature Identification:

- Key Features: Account management, comprehensive hiring system, notifications, collaborative project management tools, and AI-powered service matching.

2. Imagining System Design:

- System Interaction: Brainstormed user interface, backend architecture, database management, and external service integration to ensure scalability for both small tasks and larger projects.

Outcome:

The brainstorming sessions refined our vision for ServiceSphere, identifying essential features and conceptualizing a robust system design. This laid the foundation for developing a comprehensive platform to connect clients, freelancers, and project teams across diverse industries in Egypt and beyond.

4.2 System Scope

ServiceSphere is an innovative online platform revolutionizing freelance collaboration and project management across industries like Engineering & Architecture, Construction/Renovation, Event Planning, and Education. It connects clients, freelancers, and project teams, enabling efficient project lifecycle management from initiation to closure.

4.2.1 Functional Requirements

Functional requirements specify what the system should do and how it should behave. They describe the features and functionalities that the system should provide to the user and the stakeholders. The functional requirements for this project are:

- **Secure Login and Registration:** ServiceSphere incorporates JWT authentication to ensure a secure login and registration process, safeguarding user credentials and enabling efficient session management across the platform.

- **Freelancer Profile Management:** Enables freelancers to create and update detailed profiles showcasing their skills, experience, and services they provide.

- **Project and service Posting:** Enables clients to post a variety of job opportunities, from individual services to extensive, multi-faceted projects. This flexibility caters to different project scopes and client needs, ensuring that both small-scale tasks and large-scale operations can be efficiently managed and executed within the platform.

- **Filtering:** ServiceSphere incorporates a sophisticated filtering system that allows freelancers to efficiently navigate and select job posts based on specific categories. This feature is essential for streamlining the search process and ensuring that freelancers can quickly find projects that match their expertise and interest areas.

- **Administrative and Supportive Features:**

    - **Contracts Management:** Facilities for managing and viewing contracts between freelancers and clients.

    - **Categories and Sub-Categories:** Detailed listings of services categorized to aid in easier navigation and matching.

- **Communication Tools:** integrating WhatsApp to facilitate direct and immediate communication between clients and freelancers. This integration allows users to move seamlessly from the platform's internal systems to a widely used messaging service, providing a more flexible and accessible means of communication.

- Real-Time Notifications: includes notifications for proposal submissions by freelancers, proposal acceptance by client and contract initiations. The integration of SignalR ensures that users receive notifications without needing to refresh their browser or app. This seamless experience keeps users engaged and informed about relevant activities and updates related to their projects.

- **Payment:** integrates a secure and efficient payment system using Stripe, a leading payment processing platform known for its reliability and robust security measures. This integration ensures that financial transactions are seamlessly managed and that funds are securely held until project milestones are successfully met.

AI and Advanced Functionality

- **Text Refinement:** ServiceSphere employs AI technology to refine job and project descriptions, ensuring they are clear and effective. This enhancement increases the appeal of postings, attracting suitable freelancers and clients more effectively.

- **Project Milestone Generation:** With AI algorithms, ServiceSphere automatically generates project milestones based on requirements and objectives. This feature aids in project planning by breaking tasks into manageable stages, facilitating smoother project management.

- **Team Assembly for Extensive Projects:** ServiceSphere utilizes AI to assemble project teams based on requirements and services needed when a client

posts a project using ai. This intelligent feature optimizes resource allocation and enhances collaboration for successful project outcomes

4.2.2 Non-Functional Requirements

Non-functional requirements specify how well the system should do what it does. They describe the quality attributes and constraints that the system should meet or comply with. They affect the user's satisfaction and the system's performance, reliability, security, and maintainability. The non-functional requirements for this project are:

1. Usability:

- The user interface is crafted using React, known for its responsiveness and capability to build intuitive, dynamic user experiences that are easy to navigate for various user categories.

- Multilingual support is robustly handled by React's ecosystem, which facilitates seamless integration of internationalization libraries to cater to a global user base.

2. Performance:

- .NET is utilized for backend operations due to its powerful performance capabilities, ensuring quick response times and efficient handling of peak loads, especially in search and matching functions where speed is crucial.

- React's efficient rendering and update mechanisms help maintain a high-performance frontend, making the user experience smooth even during high-traffic periods.

3. Scalability:

- The .NET platform supports large and complex projects with a high degree of scalability, allowing ServiceSphere to efficiently manage an increasing number of users and concurrent operations.

- Flutter's ability to run on multiple platforms with a single codebase significantly simplifies scaling mobile applications, supporting a unified and consistent performance across all devices.

4. Security:

- .NET provides a robust framework that includes advanced security measures, such as JWT authentication, which is crucial for protecting sensitive user data and managing secure, token-based user sessions across the platform. This helps in ensuring that the login and registration process is highly secure.

- The integration of JWT with React enhances front-end security, ensuring that user credentials are protected, and session management is efficient. React's compatibility with JWT allows for secure API calls and helps maintain compliance with GDPR and other data protection regulations.

5. Reliability:

- .NET's stable environment and extensive library support ensure that the system operates consistently with minimal downtime, enhancing the overall system reliability.

- Implementations of fail-safe mechanisms are supported by .NET's comprehensive error handling and logging capabilities, which help in quickly diagnosing and recovering from system failures.

6. Maintainability:

- The use of modular code in .NET and React supports maintainability, allowing developers to update and manage the system efficiently without disrupting service.

- ServiceSphere enhances maintainability using .NET design patterns, including Repository and Unit of Work for streamlined data management and transaction consistency, Dependency Injection for reducing component coupling and improving testability, and the Specification Pattern for robust encapsulation of business rules. These practices ensure the platform is modular, scalable, and easy to update, supporting efficient long-term maintenance and upgrades.

7. Accessibility:

- React supports the development of accessible web applications by adhering to WCAG 2.1 guidelines, making the platform usable for people with disabilities.

- Flutter also emphasizes accessibility in mobile development, providing widgets and tools that help create accessible mobile apps, ensuring that accessibility standards are met across all platforms.

By leveraging the strengths of these technologies, ServiceSphere ensures that it meets its non-functional requirements effectively, supporting its goal to provide a secure, reliable, and user-friendly platform for freelance collaboration and project management.

4.3 User Story Definitions

1. User Registration and Authentication

As a user, I want to:

- Create a detailed profile with my skills, expertise, and past experience, so that I can showcase my capabilities to potential clients or collaborators.

- Securely register and log in so that my account is protected.

2. Service and Project Posting

As a client, I want to:

- Post individual services I need, specifying details such as requirements, budget, and timeframe, so that I can find freelancers for specific tasks.

- Create entire projects, outlining project scope, objectives, budget, and priorities, so that I can manage larger projects comprehensively.

3. Service Categories and Sub-Categories

As a user, I want to:

- Navigate an intuitive and user-friendly interface that categorizes services and projects into fields like Engineering & Architecture, Construction/Renovation, Event Planning, and Education. so that I can easily find what I need.

- Refine my search based on categories, so that I can find specific job postings that match my requirements.

4. Client Options

As a client, I want to:

- Hire individual freelancers for specific tasks, so that I can address small-scale needs efficiently.

- Opt for complete project management by forming a dedicated team for end-to-end project execution, so that my larger projects are managed seamlessly.

5. AI-Powered Matching and Team Assembly

As a client, I want to:

- Use AI algorithms to match my requirements with suitable freelancers or teams, so that I can find the best talent for my project.

- Automatically assemble teams based on predefined criteria, so that the right skills are matched to each project aspect.

- Refine my text while posting jobs.

6. Communication and Collaboration Tools

As a user, I want to:

- Use messaging functionalities to enable real-time communication among users, so that I can collaborate effectively.

- Assign tasks and track progress, so that project management is transparent and accountable.

7. Payment

As a client, I want to:

- Use secure payment gateways for service payments and project milestones, so that transactions are safe.

- Have funds held in escrow until services or project milestones are satisfactorily completed, so that trust is ensured between clients and freelancers.

8. Rating and Review System

As a user, I want to:

- Leave ratings and detailed reviews after the completion of services or projects, so that the quality of work is recognized.

9. Notification and Alerts

As a user, I want to:

- Receive real-time in-app notifications about service/project updates, messages, proposals, deadlines, and milestones, so that I stay informed.

10. Help and Support

As a user, I want to:

- Utilize self-help resources like a ChatBot, so that common queries are addressed quickly.

User Scenario Examples

Extensive Project: Apartment Renovation Project

As a client (Sarah), I want to:

- Log into ServiceSphere and initiate a new construction project by providing detailed requirements and goals.

- Use AI refinement to break down the project into individual services.

- Review and approve the AI-generated breakdown, team composition, and service details.

- Communicate and collaborate with the formed teams using live chat, task assignments, and file-sharing functionalities.

- Monitor project progress and milestones using the integrated calendar and task tracker.

- Provide feedback and ratings for freelancers and the overall service upon completion.

Individual Service: Air Conditioner Repair Service

As a client (Sarah), I want to:

- Search for freelancers who specialize in fixing air conditioning units.

- Send a direct message to a selected freelancer (Jake) to request service and provide additional details.

- Receive a quote and availability from the freelancer.

- Agree to the terms and confirm the appointment through the platform.

- Monitor the repair process and receive updates on the progress.

- Complete the payment and leave feedback after the service is satisfactorily completed.

These user stories and scenarios provide a comprehensive overview of how ServiceSphere caters to the needs of clients and freelancers, ensuring efficient project management and collaboration across various industries.

4.4 System Design

4.4.1 System architecture

System architecture refers to the high-level design and organization of software, hardware, and network components within a system. It provides a blueprint for how different components of a system interact and work together to achieve the desired functionality and performance.

ServiceSphere architecture is decomposed into about 3 parts or sides, where there is a client-side which includes the components of the way that the user can communicate with the application and a back-end (C#.net) side which is about how the application work and cooperate to achieve a specific task and the client-side and back-end are collaborating together to achieve the main features of the application, and also there is an additional components side which considered as the authorization part of the application. Thus, the system architecture of ServiceSphere can be described as follows:

4.4.2 Client-Side Components

Front-End Framework:

ASP.NET Razor Pages or MVC: Describe how ASP.NET Razor Pages or MVC templates are used to render server-side HTML views and handle user interactions.

JavaScript Frameworks: If applicable, mention the use of JavaScript frameworks like React, for building dynamic and responsive user interfaces.

User Interface (UI):

Layout and Design: Explain the overall UI design principles, such as responsive design, accessibility considerations, and UI/UX patterns.

Components and Widgets: List specific UI components (e.g., forms, tables, modals) used throughout the application.

Client-Side Scripting:

JavaScript: Detail any custom JavaScript code used for client-side validation, dynamic UI updates, or asynchronous operations.

Styling and CSS:

CSS Frameworks: Mention any CSS frameworks (e.g., Bootstrap) used for styling and layout management.

Custom Styles: Document any custom CSS stylesheets or preprocessors used to enhance the UI aesthetics and maintain consistency.

State Management:

Client-Side State: Describe how client-side state management is handled, whether through local storage, session storage, or state management libraries.

Integration with Back-End:

API Consumption: Explain how the client-side communicates with the back-end APIs (e.g., ASP.NET Web API endpoints) to fetch and update data.

Authentication and Authorization: Detail how authentication tokens (JWT, OAuth) are managed and passed between the client-side and server-side components.

4.4.3 Backend Components

- Application Layering:

  Presentation Layer: ASP.NET MVC (Model-View-Controller) for handling user interface and user input.

  Business Logic Layer: C# classes and libraries for implementing business rules and logic.

  Data Access Layer: Entity Framework for interacting with the database.

- Database Management:

Database Server: SQL Server or another relational database management system.

Database Design: Entity-Relationship (ER) diagram showing tables, relationships, and key attributes.

Data Access: Utilize ORM (Object-Relational Mapping) frameworks like Entity Framework for seamless interaction with the database.

- Authentication and Authorization:

  Implement ASP.NET Identity or another authentication mechanism for user login, registration, and session management.

  Define roles and permissions to control access to different parts of the application.

- API Integration:

  Develop APIs using ASP.NET Web API to facilitate communication with external systems, such as mapping services or social media platforms.

  Implement RESTful API endpoints for data exchange and integration purposes.

Additional Components

- Notification System: Implement a notification system to keep freelancers and clients informed about project updates, messages, and other relevant events.

- Search and Filtering: Develop robust search and filtering capabilities to help users easily find freelancers, projects, and categories relevant to their needs.

- Review and Rating System: Integrate a review and rating system where clients can provide feedback on freelancers' work, helping to build trust and transparency within the platform.

- Messaging and Communication: Implement a messaging system that allows seamless communication between freelancers and clients, including real-time messaging and file sharing capabilities.

- Analytics and Reporting: Include analytics tools to track platform usage, project success rates, and other key metrics. Provide reporting features for clients and freelancers to analyze their activities and performance.

- Payment Gateway: Integrate a secure payment gateway to facilitate transactions between clients and freelancers, ensuring smooth and reliable payment processing.

- Admin Dashboard: Develop an administrative dashboard to manage user accounts, review disputes, monitor platform activities, and perform other administrative tasks efficiently.

- Support System: Implement a support system to handle user inquiries, provide assistance, and resolve issues promptly, ensuring a positive user experience.

4.5 System UI

The System UI of ServiceSphere is designed to provide users with an intuitive and engaging interface that allows them to seamlessly explore statues and monuments using augmented reality. The UI elements are carefully crafted to deliver a visually appealing and user-friendly experience.

In ServiceSphere we tried to keep the ancient style while making it as modern as possible by using the right colors and fonts and came out with a user-friendly application that matches our identity. The used fonts are a mixture of black and white and its gradients, and we used the Dm serif text(main font) and stardos stencil font with its all weights for the whole project (Web - Mobile).

Color Palette

1. Light Gray (#F2F2F2)

- Usage: Background
- Description: A very light gray used for the main background, providing a clean and minimalistic look.

2. Black (#000000)

- Usage: Text and Buttons
- Description: Standard black used for text and interactive elements such as the "Next" and "Back" buttons, ensuring high contrast and readability.

3. Dark Gray (#333333)

- Usage: Icons and Secondary Text
- Description: Dark gray used for the icons and secondary text, maintaining readability while being softer than pure black.

4. White (#FFFFFF)

- Usage: Button Text
- Description: Pure white used for button text, creating a stark contrast with the black background of the buttons.

5. Purple(#7e22ce)

- Usage: Button
- Description: Pure Purple used for button.

For the UI screens we used Figma, as for the logo, icons, and vectors we used Photoshop and Illustrator as a result came up with these powerful and good-looking Application/Website user interface.

In Figma we designed the Sitemap and the User Interface of the Application/Website while maintaining the design concepts of UI designing, as in Illustrator we designed the logo of ServiceSphere and a specific set of icons for the project, finally we used the Photoshop to build up the project identity to be more famous and well-known with the appearance of the Application.

In each flow of the application, we are going to show the sitemap and the flow of every screen in the whole project.

4.5.1 Mobile UI flow

Sign up and Login flow

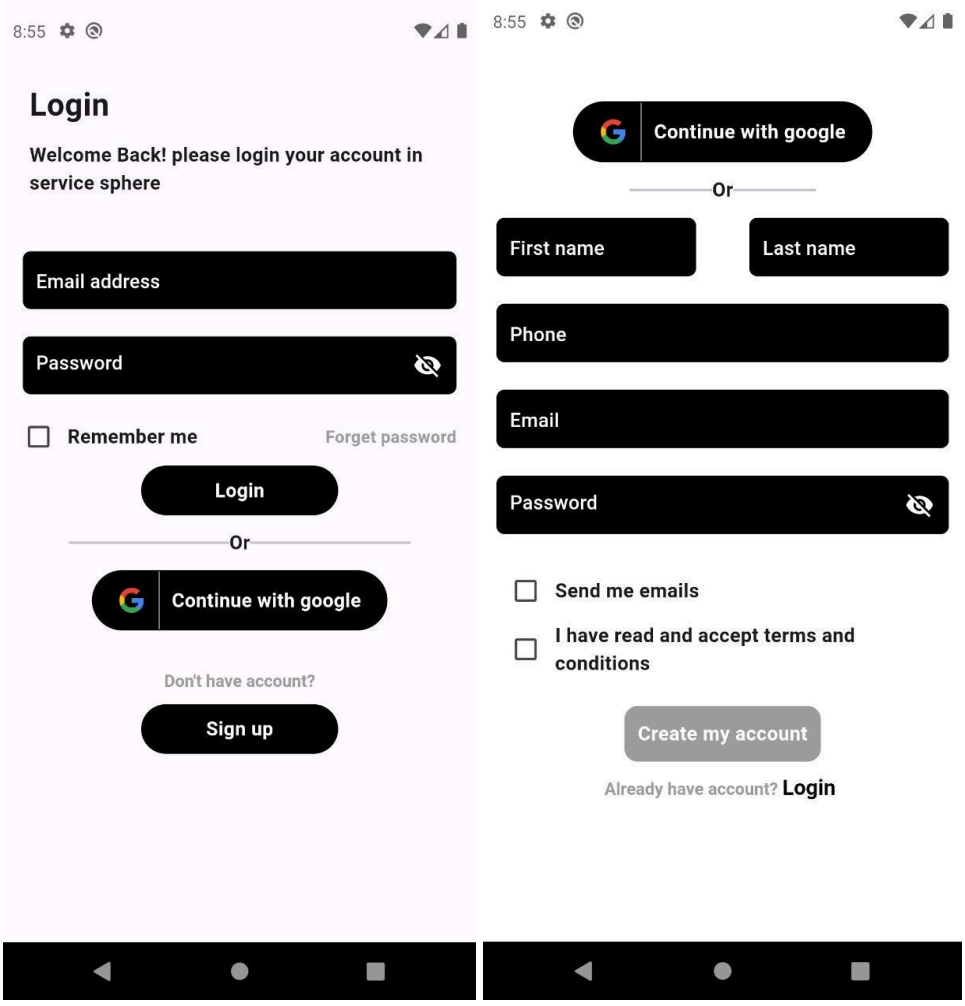
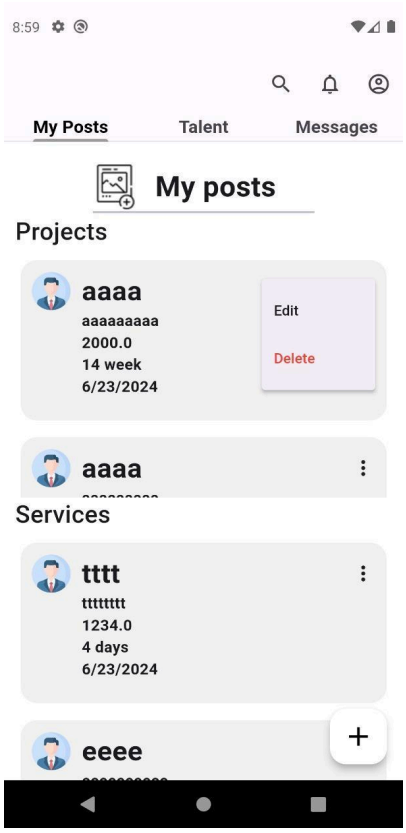
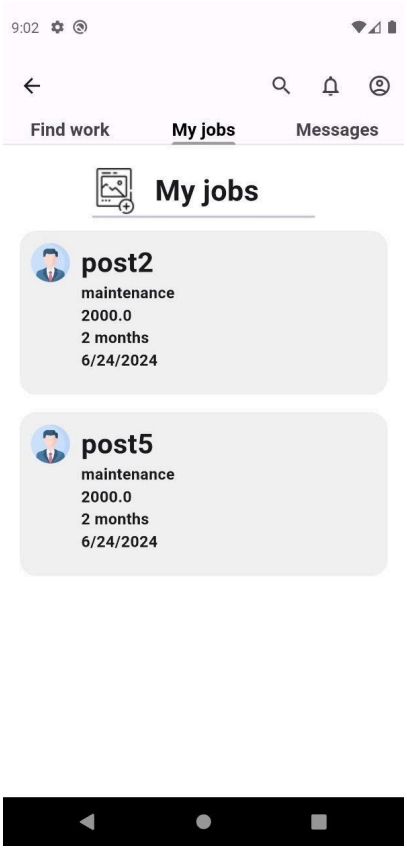


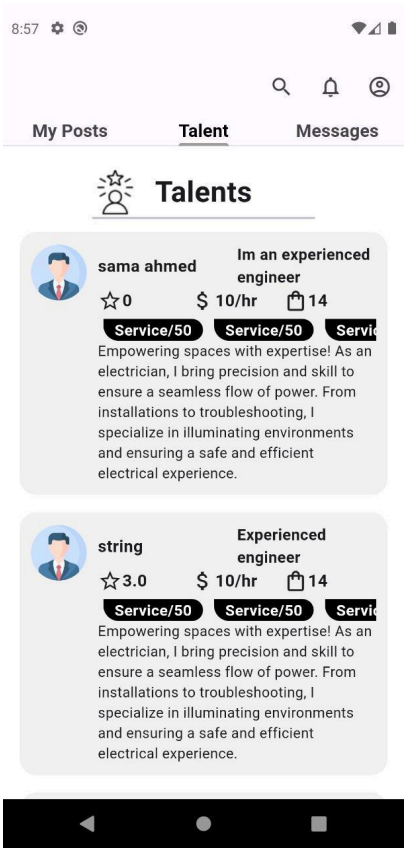
figure14:(login and register flow)

Home screen :

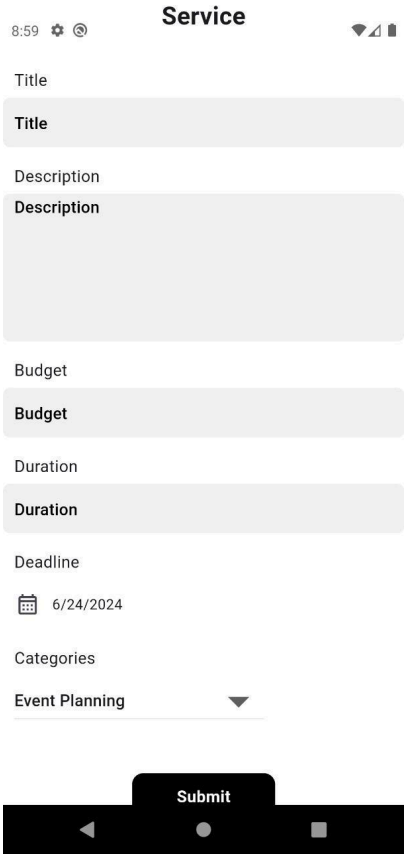




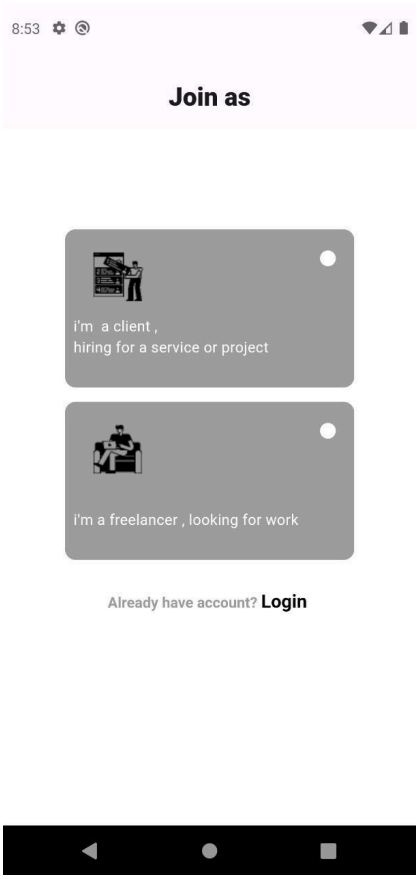
Talents Page

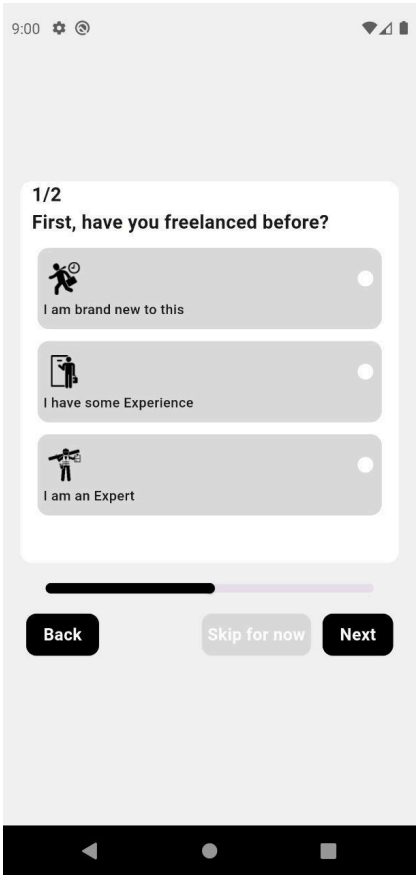


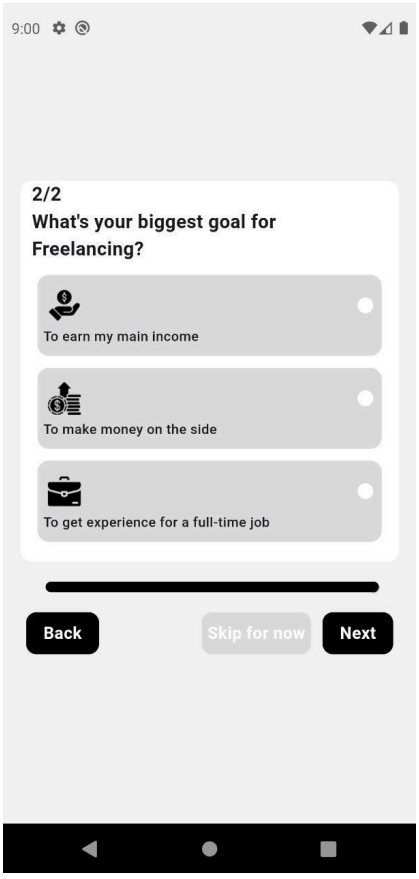
Services Page

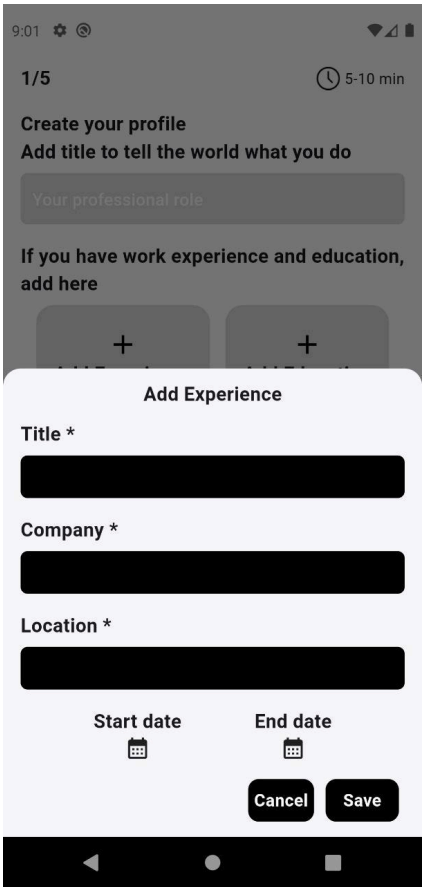


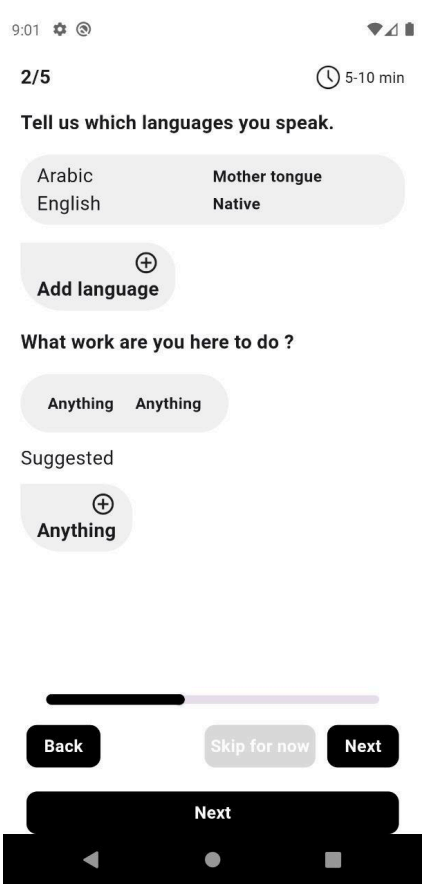
Signup Process

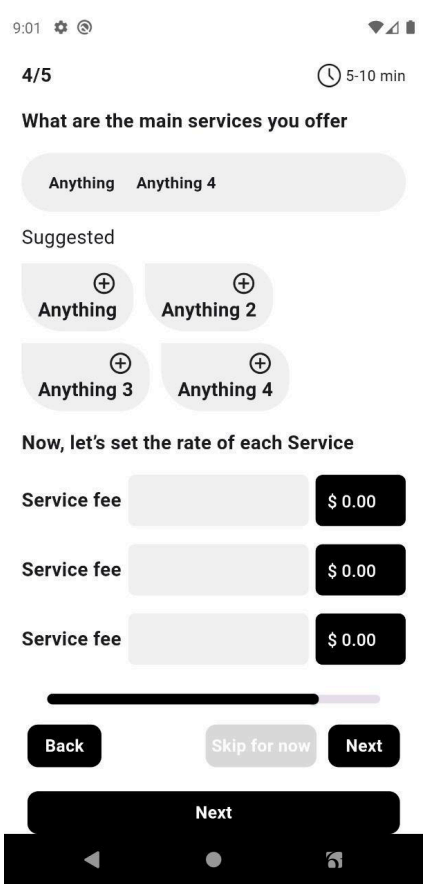












Messages Pages:

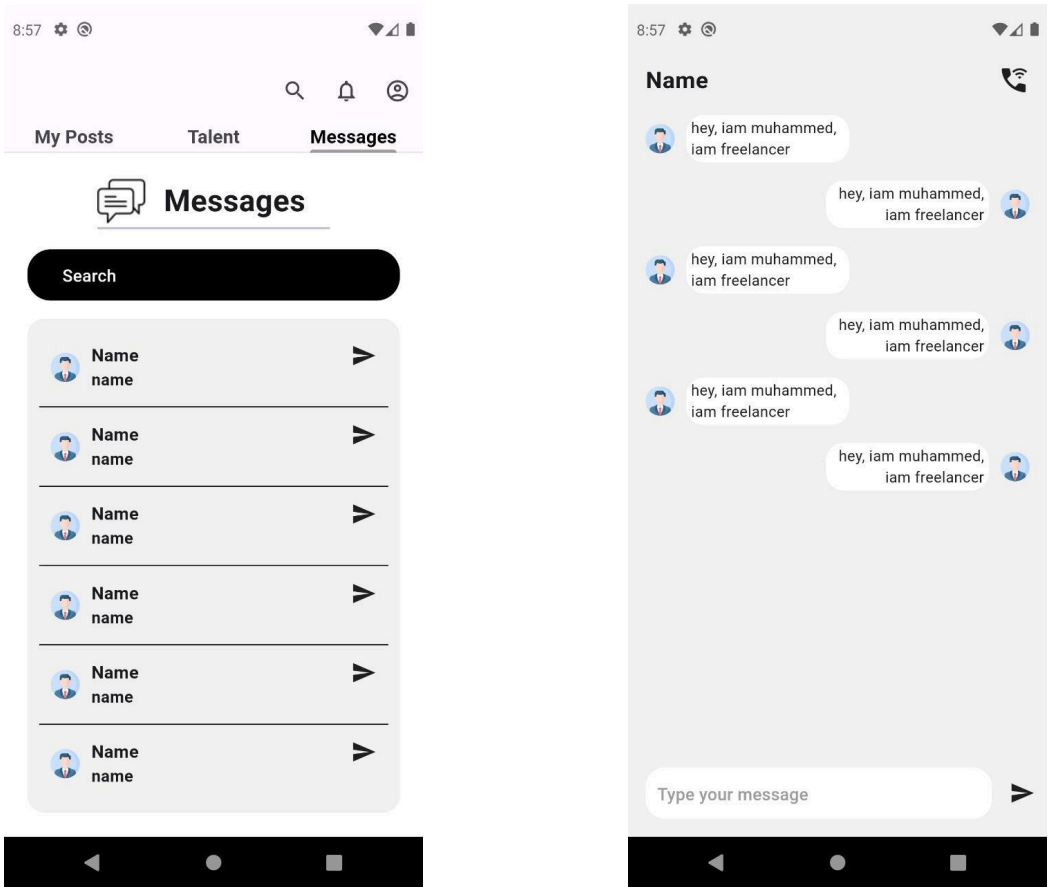


figure 15:(Messaging UI)

4.5.2 Web UI flow

In our project, we have two basic users:

- Freelancer
- ClientAnd each one has a different user interface.

Login

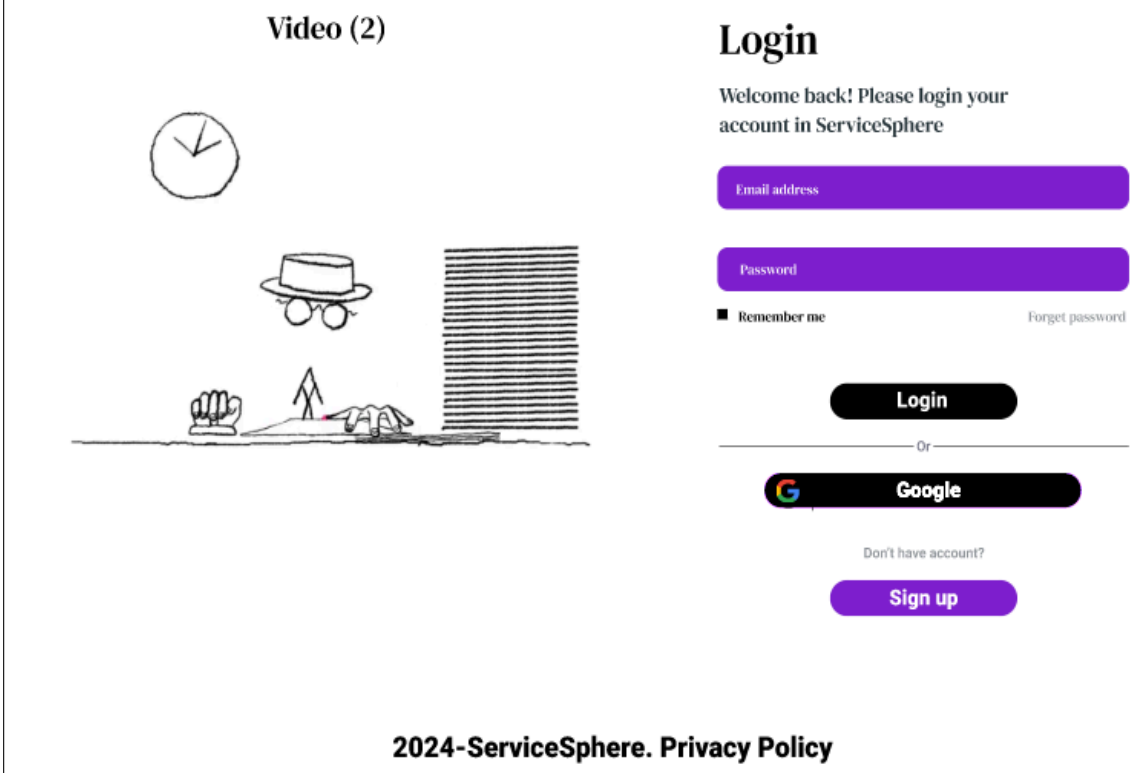


figure 16:(login Webpage ui)

Home page

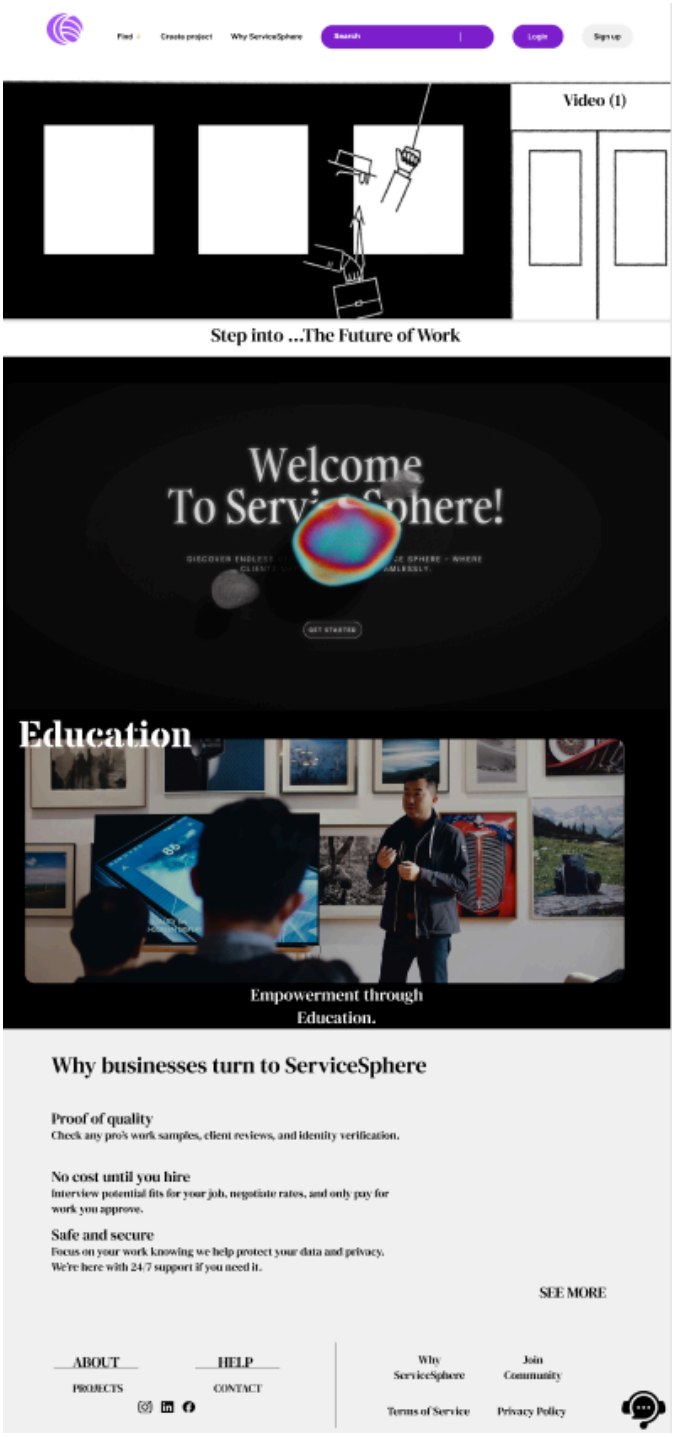


figure 17:(HomeWebpage UI)

Project Introductory Page

figure 18:(Project Webpage UI)

Client UI

Home homepage

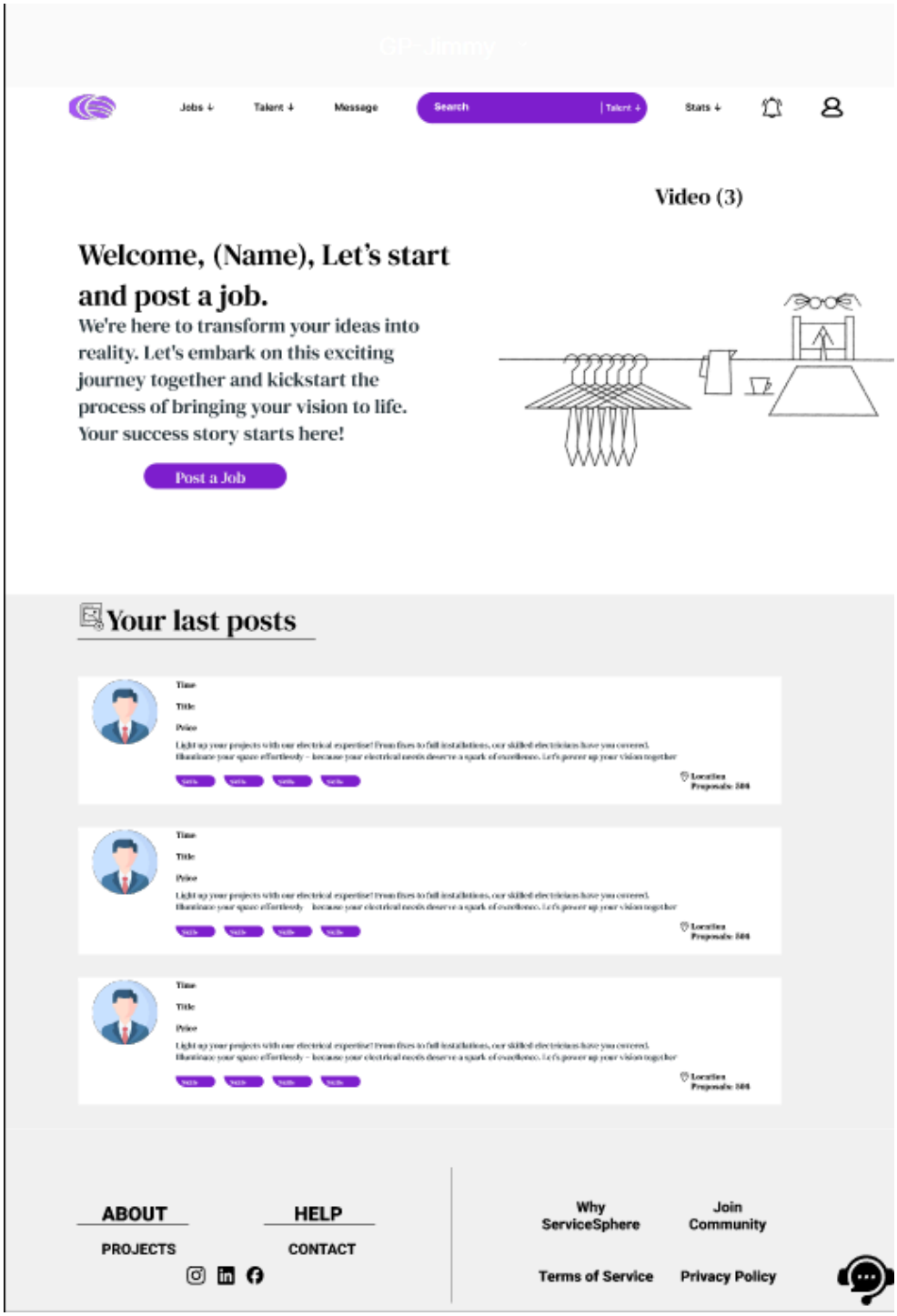
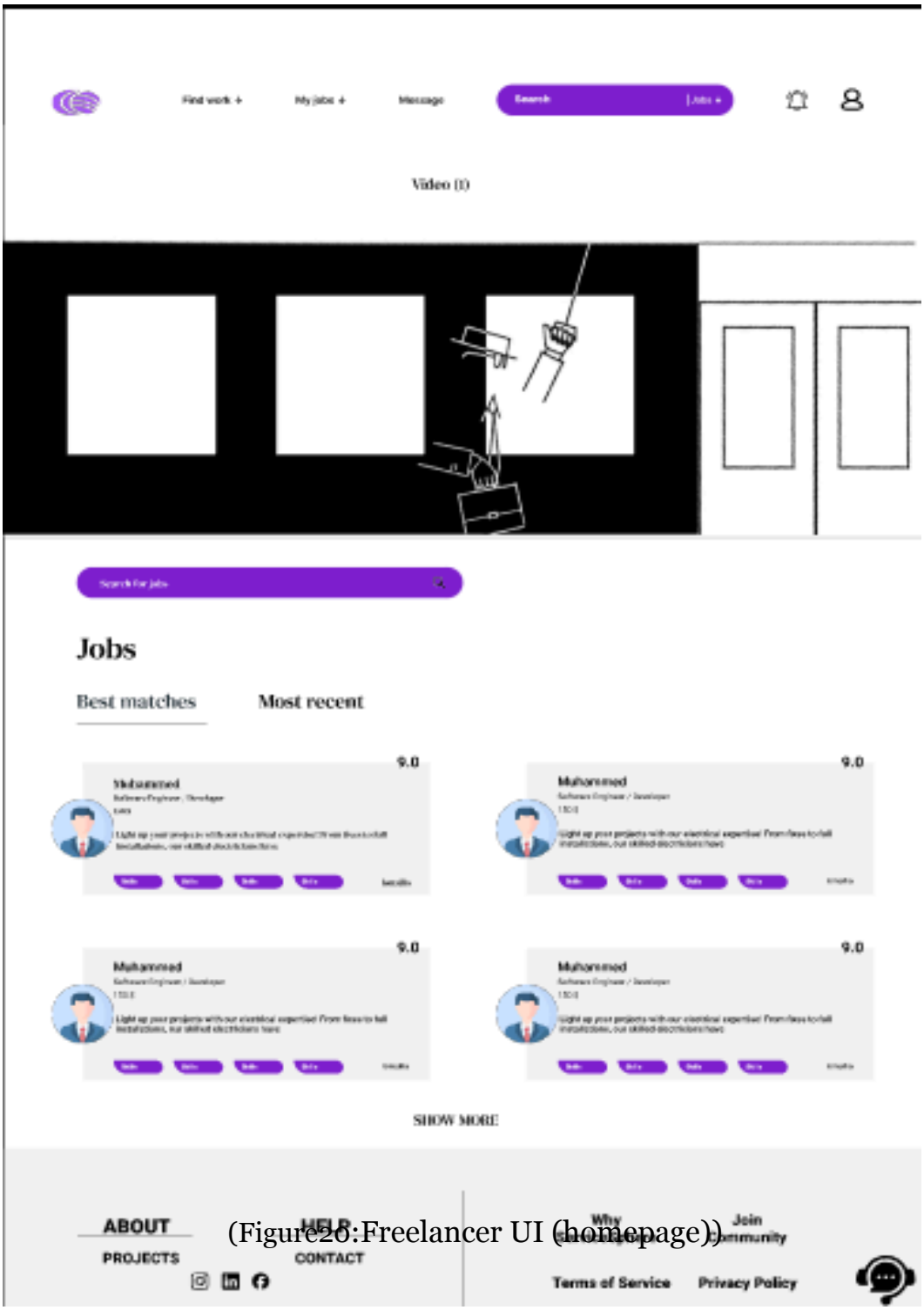


Figure 19:(Client UI (homepage))

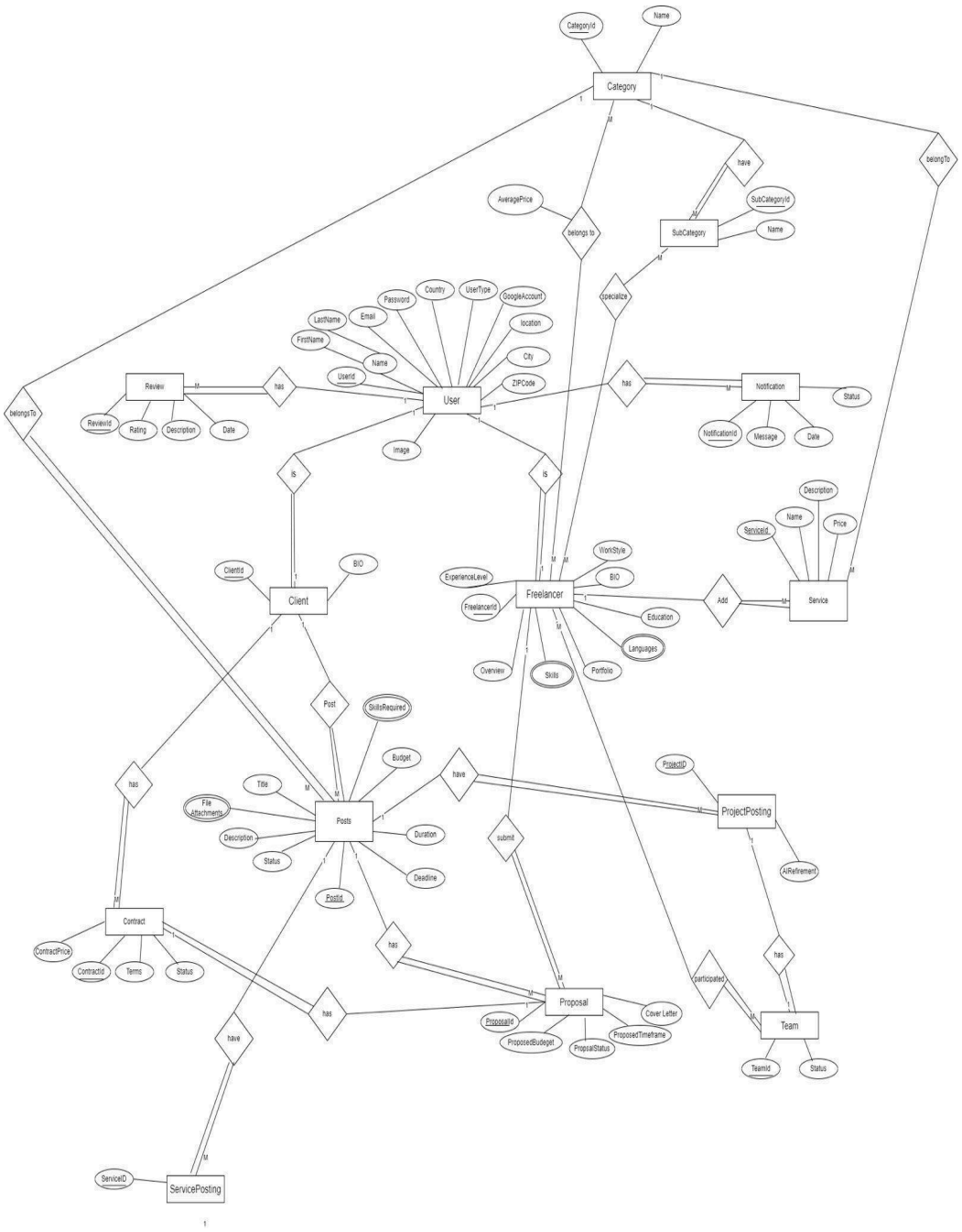
Freelancer UI

Home page



(Figure20:Freelancer UI (homepage))

4.6



Database ERD (figure 21 ERD)

The ERD provides a visual representation of the entities, attributes, and relationships within the ServiceSphere system. It helps in understanding the data structure and the connections between different components, facilitating the development and management of the application's database.

4.7 Summary

The System Design section covers the system architecture, components, and interfaces. It explains the structure and organization of the system, and details the main components and their functionalities, inputs, outputs, and interactions. It also defines the interfaces that the system uses to communicate with other systems or devices.

Chapter 5
System Implementation

5.1 Code Implementation

5.1.1 Frontend Development

React-JS Hooks used

ReactJS hooks are functions that allow you to add state and other features to functional components. They were introduced in ReactJS version 16.8 and have become a popular way to add functionality to functional components without using class components.

Here are ReactJS hooks we used:

- useState:

  useState is a ReactJS hook that allows you to add state to functional components. It is one of the most commonly used hooks in ReactJS and is used to manage stateful data that can change over time.

Here's a breakdown of how useState works:

  - useState takes one argument, which is the initial state value. This value can be any type, such as a number, string, array, or object.

  - useState returns an array with two values: the current state value and a function to update the state value.

- useEffect:

  useEffect is a ReactJS hook that allows you to perform side effects in functional components. Side effects are actions that happen outside of the component, such as fetching data from an API, updating the DOM, or setting up event listeners.

Here's a breakdown of how useEffect works:

- useEffect takes two arguments: a function that performs the side effect, and an optional array of dependencies that determine when the effect should be re-run.
- The side effect function is executed after the component has rendered. You can use it to perform any kind of side effect, such as fetching data from an API, updating the DOM, or setting up event listeners.

- useLocation:

The "useLocation" is a hook provided by React Router, a library that helps with routing in React applications. It allows you to access and interact with the current URL location in your application. By using the "useLocation" hook, you can obtain information about the current URL, such as the pathname, search parameters, and hash.

Packages Used

- React Router

React Router is a popular routing library for React applications that allows developers to manage the navigation and URL structure of a single-page application (SPA). It provides a set of components that make it easy to implement client-side routing in a React application.

One of the key benefits of using React Router is that it allows developers to create a seamless user experience by updating the content of the application without requiring a page refresh. When a user clicks on a link, React Router updates the URL in the browser's address bar and renders the appropriate component based on the URL. This makes it possible to create dynamic and interactive SPAs that provide a smooth user experience.

React Router also supports nested routes ,Redirects ,Route matching,and route parameters, which allows developers to create more complex navigation patterns.

- Routing in React Router

  Routing in React Router uses the Route component to map URLs to specific components in a web application. React Router also includes Switch and Redirect components to help control the application's flow.

- Nested Routes

  Nested routes make it easy for developers to create complex routing systems. They are defined by placing Route components inside other Route components, allowing the creation of subpages and submenus within the web application

- Redirects

  Redirects allow developers to redirect users to a different URL based on specific conditions. Redirects can be used to create authentication systems, where users are redirected to a login page if they are not logged in.

  React Router provides extensive documentation and support, making it easy for developers to get started and find answers to common questions. It also has a large community of developers who regularly contribute new components and features to the library.

- Formik

  Formik is a popular form management library for React applications that simplifies the process of building and handling forms. It provides a set of components and utilities that make it easy to create forms and manage form data.

  One of the key benefits of using Formik is that it provides a simple and consistent API for working with forms, which can save developers a significant amount of time and effort. It also provides a number of built-in features that help to streamline the form-building process, such as form validation, error messages, and automatic form submission.

  Formik provides several core components that are used to manage forms and form data:

- Formik: wraps a form and provides access to the form's values, errors, and submission handlers.
- Field: represents a form field and provides methods for updating its value and handling changes.
- ErrorMessage: displays an error message for a specific field if it fails validation.

  Formik also provides several utilities that help to simplify form management, such as validate and handleSubmit. The validate function is used to validate the form data and return any errors, while the handleSubmit function is used to handle the submission of the form data.

  One of the main advantages of using Formik is that it provides built-in support for form validation, which can save developers a significant amount of time and effort. Formik supports both

synchronous and asynchronous form validation, which allows developers to easily implement custom validation rules.

Formik also provides extensive documentation and support, making it easy for developers to get started and find answers to common questions. It also has a large community of developers who regularly contribute new components and features to the library.

- FontAwesome

One such popular icon library is FontAwesome, which provides a vast collection of scalable vector icons that can be customized with CSS. FontAwesome offers icons in various styles, including solid, regular, light, and brands, enabling developers to enhance the visual appeal and usability of their websites and applications.

- FontAwesome Icon Collection

FontAwesome comes with an extensive library of icons covering a wide range of use cases, from simple UI elements like buttons and checkboxes to complex graphics representing social media platforms, tools, and various objects. The icons are vector-based, ensuring they scale perfectly at any size without losing quality.

- Customizable Icons

FontAwesome icons are easy to customize. Developers can change their size, color, and style with simple CSS classes. This makes them easy to fit into any design. FontAwesome also offers tools to create custom icons, so developers can design icons to fit their specific needs.

- Easy Integration

FontAwesome is easy to integrate into web projects. Developers can include the library via a CDN, npm package, or by downloading the files directly. The icons can be added to HTML elements using the i or span tags with appropriate CSS classes, making the implementation straightforward and hassle-free.

- Extensive Documentation and Community

FontAwesome provides detailed documentation on using and customizing icons. It includes examples, usage guidelines, and troubleshooting tips, making it easy for developers to get started and solve common problems. Additionally, FontAwesome has a large, active community that offers extra resources, plugins, and support.

FontAwesome is a powerful and flexible icon library that can help developers create visually appealing and user-friendly web applications quickly and efficiently.

- react-icons

react-icons is a lightweight library that allows developers to easily include popular icons in their React applications. It provides a wide range of icons from various icon libraries, such as FontAwesome, Material Icons, and more, all accessible through a consistent API.

Features of react-icons

- Comprehensive Icon Collection: react-icons offers a vast selection of icons from multiple libraries, making it easy to find the perfect icon for any use case.
- Easy Integration: Icons can be imported and used in React components with a simple syntax, ensuring seamless integration into any React project.

- Customizable Icons: Icons can be easily customized in terms of size, color, and style using inline styles or CSS classes, providing flexibility to match any design requirements.
- Lightweight and Performant: react-icons is designed to be lightweight, ensuring that it does not significantly impact the performance of web applications.
- Extensive Documentation: The library comes with comprehensive documentation, making it easy for developers to get started and find answers to common questions.

react-icons simplifies the process of incorporating icons into React applications, enhancing the visual appeal and usability of web projects with minimal effort.

- jQuery

  jQuery is a fast, small, and feature-rich JavaScript library that simplifies the client-side scripting of HTML. It provides an easy-to-use API that works across a multitude of browsers, making it a popular choice for web development.

  Features of jQuery

- DOM Manipulation: jQuery makes it easy to select, manipulate, and traverse HTML elements, simplifying the process of creating dynamic and interactive web pages.
- Event Handling: jQuery provides a powerful mechanism for handling events, allowing developers to capture and respond to user interactions seamlessly.
- Animation and Effects: jQuery offers a wide range of built-in animations and effects, allowing developers to enhance the user experience with minimal effort.
- Extensible: jQuery's plugin architecture allows developers to extend its functionality easily, adding custom features and components as needed.

jQuery remains a powerful and versatile tool for web development, providing a simplified and consistent approach to client-side scripting.

- Microsoft SignalR

  Microsoft SignalR is a library for ASP.NET that allows server-side code to send asynchronous notifications to client-side web applications. It enables real-time web functionality, making it possible to push content updates to connected clients instantly.

  Features of Microsoft SignalR

  Real-time Communication: SignalR enables real-time communication between server and client, allowing for instant data updates and live interaction within web applications.

  Automatic Reconnection: SignalR handles connection management, including automatic reconnection, ensuring a stable communication channel between server and client.

  Scalable: SignalR supports scaling out to multiple servers, making it suitable for high-demand applications requiring real-time updates.

  Multiple Transport Protocols: SignalR supports multiple transport protocols, including WebSockets, Server-Sent Events, and Long Polling, automatically selecting the best transport available based on the client's capabilities.

  Integration with ASP.NET: SignalR integrates seamlessly with ASP.NET, providing a robust framework for building real-time web applications within the Microsoft ecosystem.

  Extensive Documentation and Support: SignalR comes with detailed documentation and a strong community, offering a wealth of resources and support for developers.

Microsoft SignalR is a powerful tool for adding real-time functionality to web applications, enabling instant communication and enhancing the interactivity of web projects.

- Axios

Axios is a popular JavaScript library that can be used to make HTTP requests from a ReactJS application. It provides a simple and easy-to-use API for sending HTTP requests and handling responses. Axios can be used in both browser and server environments, making it a versatile choice for making API calls.

Here are some key features and benefits of using Axios in a ReactJS application:

  - Promise-based API: Axios provides a promise-based API for making HTTP requests, which makes it easy to handle asynchronous operations in a ReactJS application.
  - Interceptors: Axios provides interceptors that can be used to intercept requests and responses, and modify or handle them as needed. This can be useful for adding headers, handling errors, or logging information.
  - Automatic JSON parsing: Axios automatically parses JSON responses, making it easy to work with JSON data in a ReactJS application.
  - Cancel requests: Axios allows you to cancel requests using a cancellation token, which can be useful for handling user interactions such as canceling a long-running request.

Authentication

Authentication in ReactJS typically involves implementing a client-side login system that communicates with a server-side authentication API. There are a few different approaches to implementing authentication in React, but the most common approach involves using JSON Web Tokens (JWTs) to authenticate users.

Here are the basic steps involved in implementing authentication in ReactJS using JWTs:

- Create a login form: Create a login form that allows users to enter their username and password.
- Submit login credentials: When the user submits the login form, send a request to the server-side authentication API to verify the user's credentials.
- Receive a JWT token: If the user's credentials are valid, the server-side API will return a JWT token. This token is a digital signature that can be used to verify the user's identity in subsequent requests.
- Store the JWT token: Store the JWT token in the browser's local storage or in a cookie. This will allow the token to be persistently stored and used in subsequent requests.
- Use the JWT token: Include the JWT token in subsequent requests to the server-side API to authenticate the user. The server-side API can verify the token and use it to identify the user.
- Implement logout function

5.1.2 Backend Development:

ASP.NET Project Directory Structure

When setting up an ASP.NET project with SQL Server as the database, the project directory structure typically follows ASP.NET's conventions, with specific files and directories to manage the project's dependencies, configuration, and code. Here's an overview of the directory structure:

1 Project Root (ServiceSphere.Apis):

File/Folder	Purpose
appsettings.json	This file contains configuration settings for the project, such as database connections, app settings, and logging configurations.
appsettings.Development.json	This file holds configuration settings specific to the development environment.
Program.cs	This file contains the entry point for the application. It sets up the web host and starts the application, configures the services and the app's request pipeline. It is where middleware, routing, and dependency injection are set up
.gitignore	This file specifies files and directories that should be ignored by version control (e.g., Git).
README.md	This file typically contains documentation or information about the project.
Controllers/	This directory contains the controllers, which handle HTTP requests and return responses
Migrations /	This directory contains the database migration files, which track changes to the database schema over time.
wwwroot/	This directory contains static files, such as images, that are served directly to the client.
Properties/	This virtual directory in the project structure shows all the NuGet packages and project references.
.Dependencies/	a virtual environment to isolate project dependencies

2 Root Directory (myproject/):

File/Folder	Purpose
Program.cs	Entry point for the application. Sets up the web host and starts the application.Configures services and the app's request pipeline.

3 Controllers Directory:

File/Folder	Purpose
HomeController.cs	Example of a controller file. Handles HTTP requests and returns responses.

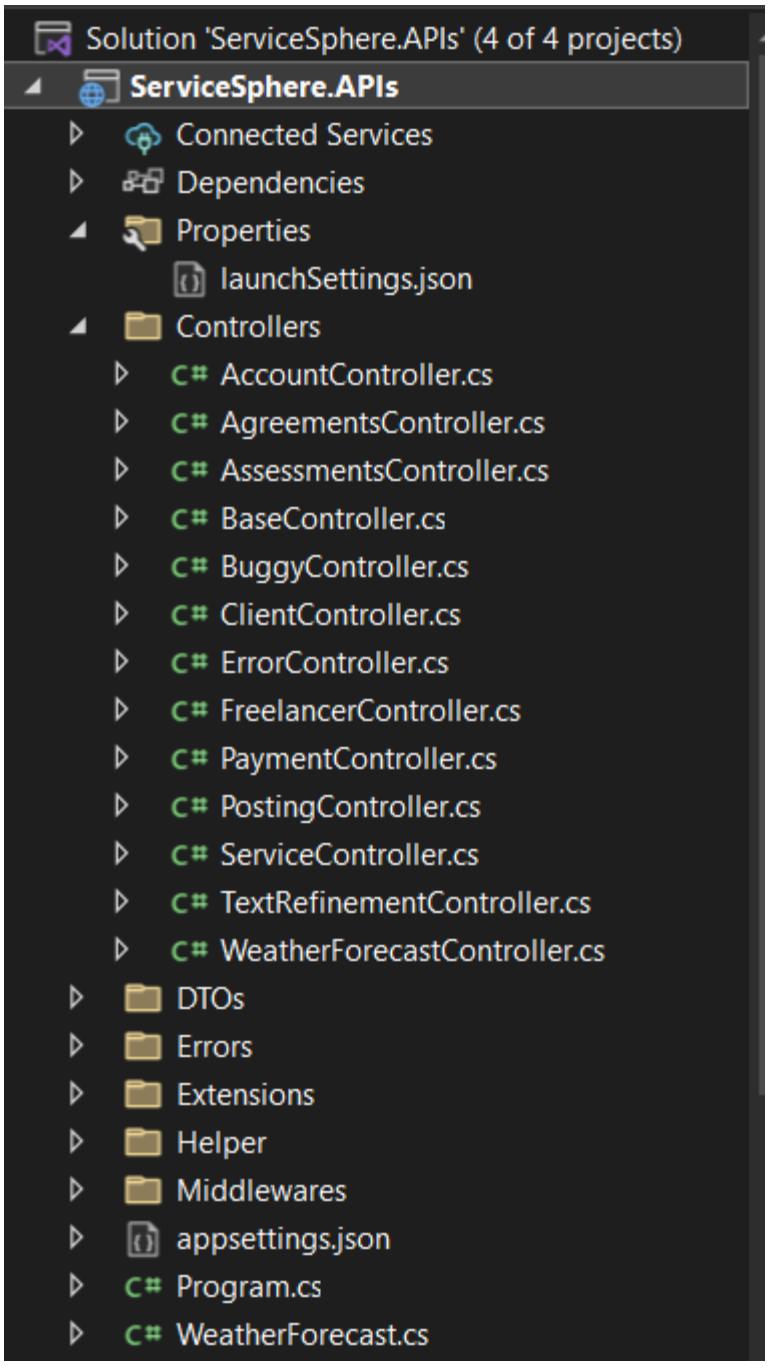
::

4 wwwroot Directory:

File/Folder	Purpose
images/	Contains image files.

5 Migrations Directory:

File/Folder	Purpose
20220101000000_InitialCreate.cs	Example of a migration file. Tracks changes to the database schema.



(Figure 22 : ServiceSphere Backend Directory Structure)

Overall, the directory structure of an ASP.NET project with SQL Server is designed to organize the project's components and dependencies efficiently. By following this structure, developers can maintain a clear and manageable codebase, facilitating the development, deployment, and maintenance of the application. The integration of ASP.NET with SQL Server ensures robust data management and seamless performance for enterprise-level applications.

The Onion Architecture

The Onion Architecture is a software architectural pattern that emphasizes a strong separation of concerns, promoting a domain-centric approach to application design. It is particularly effective for complex and scalable applications, ensuring maintainability and flexibility over time. The benefits of using the Onion Architecture include:

- Separation of concerns for better code organization.
- Code reusability through modular components.
- Flexibility and maintainability for easier updates and modifications.
- Scalability to accommodate evolving requirements.
- Improved collaboration among development teams.
- Enhanced testability for higher code quality and reliability.

Domain Layer

In the Onion Architecture, the Domain Layer is at the core and represents the business logic of the application. This layer is isolated from external concerns and dependencies, ensuring that the core business rules and logic are not influenced by the external layers.

- **Entities:** These are the fundamental building blocks of the domain. They represent the core concepts and business rules.
- **Value Objects:** These are immutable objects that represent a descriptive aspect of the domain with no distinct identity.
- **Aggregates:** These are clusters of domain objects that are treated as a single unit for data changes.
- **Repositories:** These provide an abstract way to access and manipulate aggregates

For example, in a freelancing website, the freelancer entity looks like:

```
19 references
public class Freelancer:User
{

    1 reference
    public string UserName { get; set; }

    0 references
    public string Title { get; set; }

    0 references
    public string WorkExperience { get; set; }
    0 references
    public string? Bio { get; set; }
    2 references
    public ExperienceLevel? experienceLevel { get; set; }
    0 references
    public string? Education { get; set; }
    0 references
    public string? Overview { get; set; }

    0 references
    public string? WorkStyle { get; set; }

    1 reference
    public List<Skill>? Skills { get; set; }


}
```

(Figure 23 : Freelancer Entity)

Application Layer

The Application Layer contains the application-specific business logic and orchestrates the flow of data between the outer layers and the core domain. This layer is responsible for implementing use cases and services.

- Services: These coordinate the application's operations and enforce business rules.

  For example, an application service for managing Notification looks like:

```
2 references
public async Task<IEnumerable<Notification>> GetUnreadNotificationsAsync(string userId)
{
    var spec = new NotificationsSpec(userId);
    return await _notificationsRepo.GetAllWithSpecAsync(spec);
}
2 references
public async Task<IEnumerable<Notification>> GetAllNotificationsAsync(string userId)
{
    var spec = new AllNotificationsSpec(userId);
    return await _notificationsRepo.GetAllWithSpecAsync(spec);
}
3 references
public async Task<IEnumerable<Notification>> SetNotificationReadAsync(string userId)
{
    var spec = new NotificationsSpec(userId);
    var Notifications = await _notificationsRepo.GetAllWithSpecAsync(spec);
    foreach (var notification in Notifications)
    {
        notification.IsRead = true;
        // Assuming you have an update method in your repository to persist changes
        _notificationsRepo.Update(notification);

    }
    await _context.SaveChangesAsync();
    return Notifications;
}
```

(Figure 24:Application Layer)

Infrastructure Layer

The Infrastructure Layer provides implementations for interfaces defined in the core and application layers, and it contains the technical details for accessing external resources like databases, file systems, and web services.

- **Repositories:** Implementations of the repository interfaces for data access.
- **Data Migrations:** Tools and scripts for managing database schema changes.
- **External Services:** Integrations with third-party APIs and services.

For example, a repository implementation for SQL Server might look like:

```
2 references
public class GenericRepositery<T> : IGenericRepositery<T> where T : BaseEntity
{
    private readonly ServiceSphereContext _serviceSphereContext;

    0 references
    public GenericRepositery(ServiceSphereContext serviceSphereContext)
    {
        _serviceSphereContext = serviceSphereContext;
    }

    2 references
    public async Task<IEnumerable<T>> GetAllAsync()
    {
        return await _serviceSphereContext.Set<T>().ToListAsync();
    }

    9 references
    public async Task<IEnumerable<T>> GetAllWithSpecAsync(Ispecification<T> Spec)
    {
        return await ApplySpec(Spec).ToListAsync();
    }

    10 references
    public async Task<T?> GetByIdAsync(int Id)
    {
        return await _serviceSphereContext.Set<T>().FindAsync(Id);
    }

    4 references
    public async Task<T?> GetByIdWithSpecAsync(Ispecification<T> Spec)
    {
        return await ApplySpec(Spec).FirstOrDefaultAsync();
    }
    2 references
    private IQueryable<T> ApplySpec(Ispecification<T> Spec)
    {
        return SpecificationEvaluator<T>.GetQuery(_serviceSphereContext.Set<T>(), Spec);
    }
```

(Figure 25 : Generic Repository)

Entities

In addition to Users, the freelancing website incorporates several other entities that enhance the platform's functionality and user experience. These entities include Contracts, Proposals, Notifications, Reviews, Project Postings, Service

Postings, Categories, SubCategories, Services, Freelancers, Clients, Teams, AppUsers, Skills, Addresses, and PostContracts. These entities facilitate various interactions and transactions between clients and freelancers. Let's explore these entities in more detail:

. **Project Postings** Project postings are listings created by clients describing the work they need to be done. They include details such as project scope, requirements, budget, and deadline. Freelancers can browse these postings and submit proposals.

. **Service Postings** Service postings are listings created by freelancers offering their skills and services. They include descriptions of the services, pricing, and any relevant terms. Clients can browse these postings and hire freelancers directly.

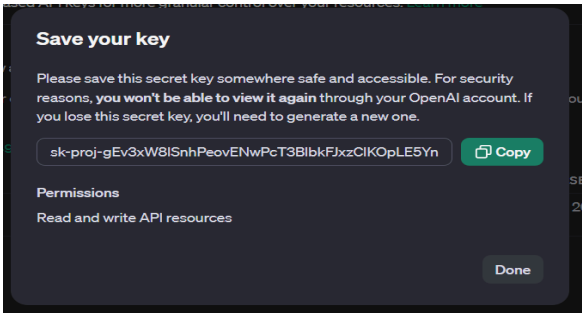
- **Contracts**: Contracts represent the agreements between clients and freelancers for specific projects or services. They include details such as contract terms, milestones, deadlines, payment terms, and other relevant information. Contracts ensure that both parties have a clear understanding of their obligations and expectations, promoting transparency and accountability.
- **Proposals**: Proposals are submissions by freelancers in response to project postings. They outline the freelancer's approach, timeline, and pricing for the project. Clients review these proposals to select the most suitable freelancer for their needs.
- **Notifications**: Notifications keep users informed about important updates and activities on the platform. They can include alerts for new proposals, messages, project updates, contract status changes, and other relevant events.
- **Reviews**: Reviews allow clients and freelancers to provide feedback on their experiences with each other. Reviews can include ratings and comments, helping to build trust and credibility within the community.
- **Categories and SubCategories:** Categories and subcategories help organize project and service postings into specific fields or industries, making it easier for users to find relevant listings.
- **Services:** Services represent the specific offerings provided by freelancers. They can include detailed descriptions, pricing, and terms, allowing clients to understand what is being offered and make informed hiring decisions.
- **Teams:** Teams represent groups of freelancers who collaborate on projects. They can include multiple members with complementary skills, allowing them to take on larger or more complex projects.

- **Skills:** Skills represent the specific abilities or expertise of freelancers. They help categorize freelancers based on their competencies, making it easier for clients to find suitable candidates.
- **Addresses**: Addresses capture the location details of users, which can be useful for certain types of projects or services that require physical presence or shipping.

These entities collectively enhance the freelancing website by providing comprehensive functionality for managing projects, services, users, and their interactions. By incorporating a robust set of features and organizing data efficiently, the platform ensures a smooth and productive experience for both clients and freelancers.

5.2 AI Integration with GPT-3.5-turbo-instruct

OpenAI's GPT-3.5-turbo-instruct model has been integrated into the ServiceSphere platform using this API key :"sk-proj-0FKCjNz4PL1kvNfG2JzaT3BlbkFJHbGsEgkyfzGvwtKG0VQC" to enhance user interactions and streamline project management processes. The model is utilized for various functions, including text refinement, project milestones generation, and team assembly.



(figure 26 API Key creation)

- **Text Refinement**

  The GPT-3.5 model is employed to refine user inputs, ensuring that job postings, project descriptions, and other textual content are clear and professional. This involves sending user input to the model using the APIkey

```
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text.Json;
using System.Threading.Tasks;

namespace ServiceSphere.APIs.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class TextRefinementController : ControllerBase
    {
        private readonly HttpClient _httpClient;
        private readonly IConfiguration _configuration;

        0 references
        public TextRefinementController(HttpClient httpClient, IConfiguration configuration)
        {
            _httpClient = httpClient;
            _configuration = configuration;
        }

        [HttpPost("RefineText")]
        0 references
        public async Task<IActionResult> Post([FromBody] string userInput)
        {
            string openAIKey = _configuration["ApiKey"];
            string prompt = $@"
                Project Title: [Enter Project Title Here]

                Project Description:
                The user has reported the following issues:
                {userInput}

                Please enhance the following text to ensure it is both formal and technically accurate. The desired format is a post

            var requestBody = new
            {
                prompt = prompt,
                model = "gpt-3.5-turbo-instruct",
                max_tokens = 300,
                temperature = 0.5
            };

            string serializedBody = JsonSerializer.Serialize(requestBody);
            _httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", openAIKey);
            _httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

            var content = new StringContent(serializedBody, System.Text.Encoding.UTF8, "application/json");
            var response = await _httpClient.PostAsync("https://api.openai.com/v1/completions", content);

            if (response.IsSuccessStatusCode)
            {
                string responseContent = await response.Content.ReadAsStringAsync();
                return Ok(responseContent);
            }
            else
            {
                return StatusCode((int)response.StatusCode, await response.Content.ReadAsStringAsync());
            }
        }
    }
}
```

- **Project Milestones Generation**

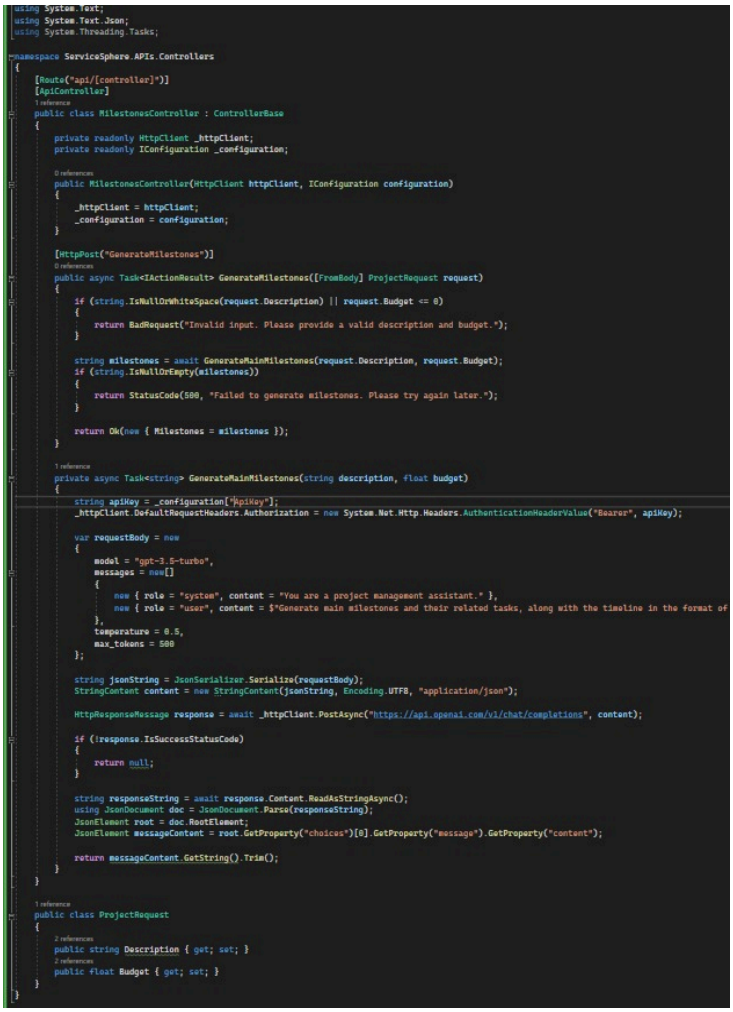
  To generate project milestones, the GPT-3.5 model processes user descriptions of projects and breaks them down into detailed tasks with timelines and budget estimates. This helps users plan and execute projects more efficiently.

(figure 28 :Milestones Code Part1)

(Figure 29 :Milestones Code part2)

- **Team Assembly**

  The model simplifies project descriptions to identify necessary skills and fetches relevant data from the database to assemble an optimal project team. It selects professionals based on their skills, availability, and budget constraints, ensuring the right fit for each project.



```
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;

namespace ServiceSphere.APIs.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class MilestonesController : ControllerBase
    {
        private readonly HttpClient _httpClient;
        private readonly IConfiguration _configuration;

        0 references
        public MilestonesController(HttpClient httpClient, IConfiguration configuration)
        {
            _httpClient = httpClient;
            _configuration = configuration;
        }

        [HttpPost("GenerateMilestones")]
        0 references
        public async Task<IActionResult> GenerateMilestones([FromBody] ProjectRequest request)
        {
            if (string.IsNullOrWhiteSpace(request.Description) || request.Budget <= 0)
            {
                return BadRequest("Invalid input. Please provide a valid description and budget.");
            }

            string milestones = await GenerateMainMilestones(request.Description, request.Budget);
            if (string.IsNullOrEmpty(milestones))
            {
                return StatusCode(500, "Failed to generate milestones. Please try again later.");
            }

            return Ok(new { Milestones = milestones });
        }

        1 reference
        private async Task<string> GenerateMainMilestones(string description, float budget)
        {
            string apiKey = _configuration["ApiKey"];
            _httpClient.DefaultRequestHeaders.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", apiKey);

            var requestBody = new
            {
                model = "gpt-3.5-turbo",
                messages = new[]
                {
                    new { role = "system", content = "You are a project management assistant." },
                    new { role = "user", content = $"Generate main milestones and their related tasks, along with the timeline in the format of
                },
                temperature = 0.5,
                max_tokens = 500
            };

            string jsonString = JsonSerializer.Serialize(requestBody);
            StringContent content = new StringContent(jsonString, Encoding.UTF8, "application/json");

            HttpResponseMessage response = await _httpClient.PostAsync("https://api.openai.com/v1/chat/completions", content);

            if (!response.IsSuccessStatusCode)
            {
                return null;
            }

            string responseString = await response.Content.ReadAsStringAsync();
            using JsonDocument doc = JsonDocument.Parse(responseString);
            JsonElement root = doc.RootElement;
            JsonElement messageContent = root.GetProperty("choices")[0].GetProperty("message").GetProperty("content");

            return messageContent.GetString().Trim();
        }
    }

    1 reference
    public class ProjectRequest
    {
        2 references
        public string Description { get; set; }
        2 references
        public float Budget { get; set; }
    }
}
```

(Figure 30 :Team Assembly Code in Part 2)

```
1 reference
private (List<Professional>, List<string>) AssembleTeam(List<string> jobTitlesNeeded, double budget, List<Professional> professionals)
{
    var team = new List<Professional>();
    var unfilledRoles = new List<string>();

    var projectManagers = professionals.Where(p => p.Title.Contains("Project Manager") && p.CostPerHour <= budget).ToList();
    if (projectManagers.Any())
    {
        var selectedManager = projectManagers.OrderByDescending(p => p.Rating).ThenBy(p => p.CostPerHour).First();
        team.Add(selectedManager);
        budget -= selectedManager.CostPerHour;
    }
    else
    {
        return (new List<Professional>(), new List<string> { "Project Manager" });
    }

    foreach (var jobTitle in jobTitlesNeeded)
    {
        var eligibleMembers = professionals.Where(p => p.Title.Equals(jobTitle, StringComparison.OrdinalIgnoreCase) && p.CostPerHour <= budget).ToList();
        if (eligibleMembers.Any())
        {
            var selectedMember = eligibleMembers.OrderByDescending(p => p.Rating).ThenBy(p => p.CostPerHour).First();
            team.Add(selectedMember);
            budget -= selectedMember.CostPerHour;
        }
        else
        {
            unfilledRoles.Add(jobTitle);
        }
    }

    return (team, unfilledRoles);
}

[HttpPost("assemble-team")]
0 references
public async Task<IActionResult> AssembleTeam([FromBody] TeamRequest request)
{
    if (string.IsNullOrWhiteSpace(request.Issue))
    {
        return BadRequest("Issue description cannot be empty.");
    }

    var jobTitles = await GenerateJobTitlesAsync(request.Issue);

    if (string.IsNullOrEmpty(jobTitles) || !jobTitles.Contains(","))
    {
        return BadRequest("The issue description provided is not understandable. Please provide a clear and concise description of the issue.");
    }

    var jobTitlesNeeded = jobTitles.Split(',').Select(title => title.Trim()).ToList();
    var filePath = Path.Combine(Directory.GetCurrentDirectory(), "..", "ServiceSphere.Repository", "Data", "DataSeeding", "Professionals.json");
    var professionals = LoadProfessionals(filePath);

    var (team, unfilledRoles) = AssembleTeam(jobTitlesNeeded, request.Budget, professionals);

    var result = new
    {
        Team = team,
        UnfilledRoles = unfilledRoles
    };

    return Ok(result);
}
```

(Figure 31 :TeamAssembly Code part1)

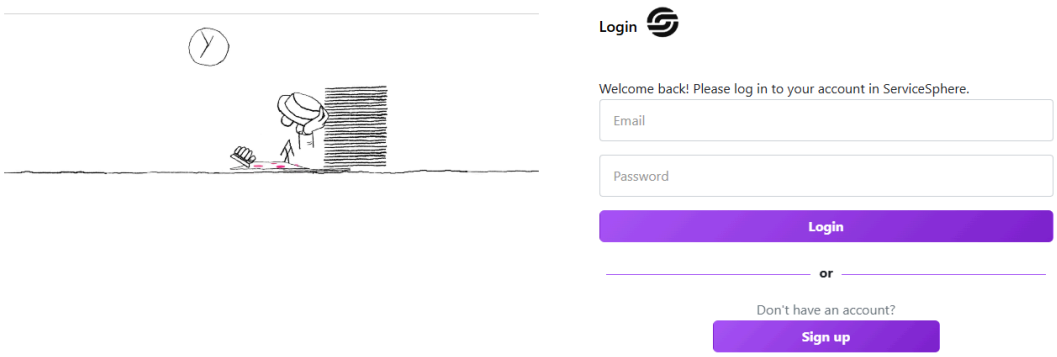
5.3 Web Basic Version

The basic version of the freelancing platform includes core functionalities that allow users to perform essential tasks

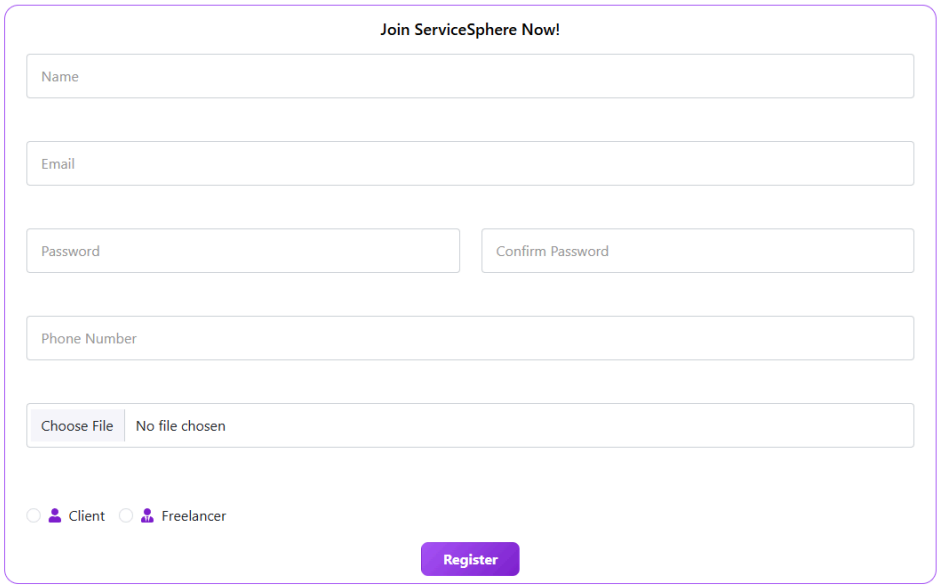
5.3.1 Frontend web development:

- **User Registration and Login:**

  Users can create accounts and log in to access the platform.



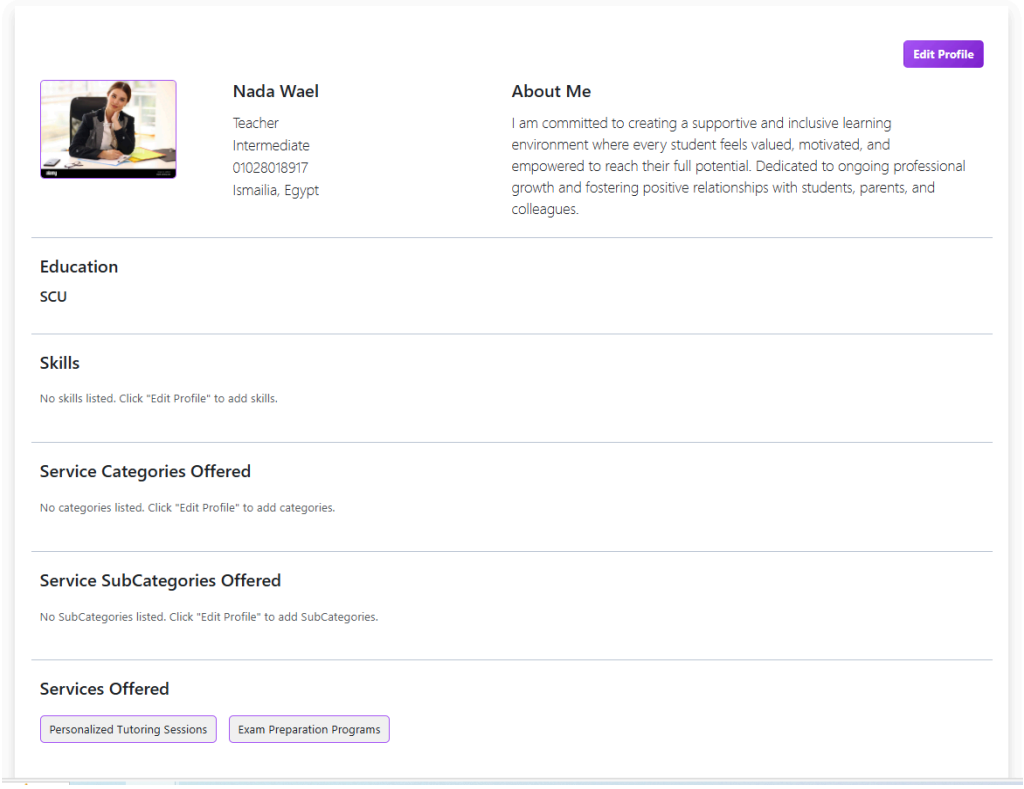
(Figure 32 : login)



(Figure 33 : Register)

Freelancer Side :

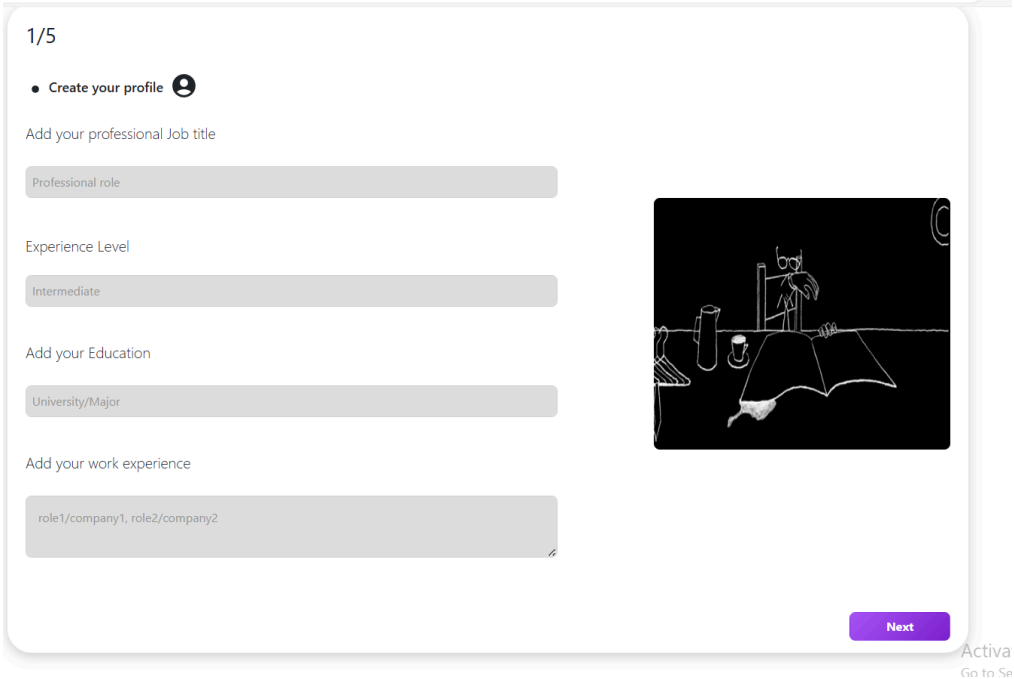
- **Profile Creation:** Freelancers can create and update their profiles with skills, experience, and portfolio.

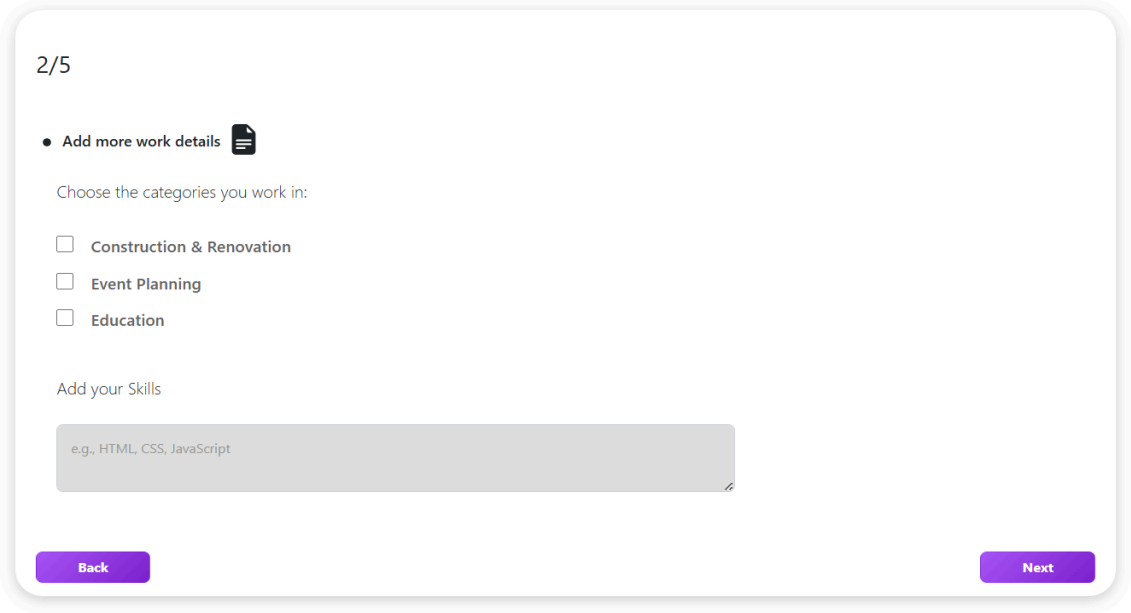


(Figure 34: Freelancer Profile)

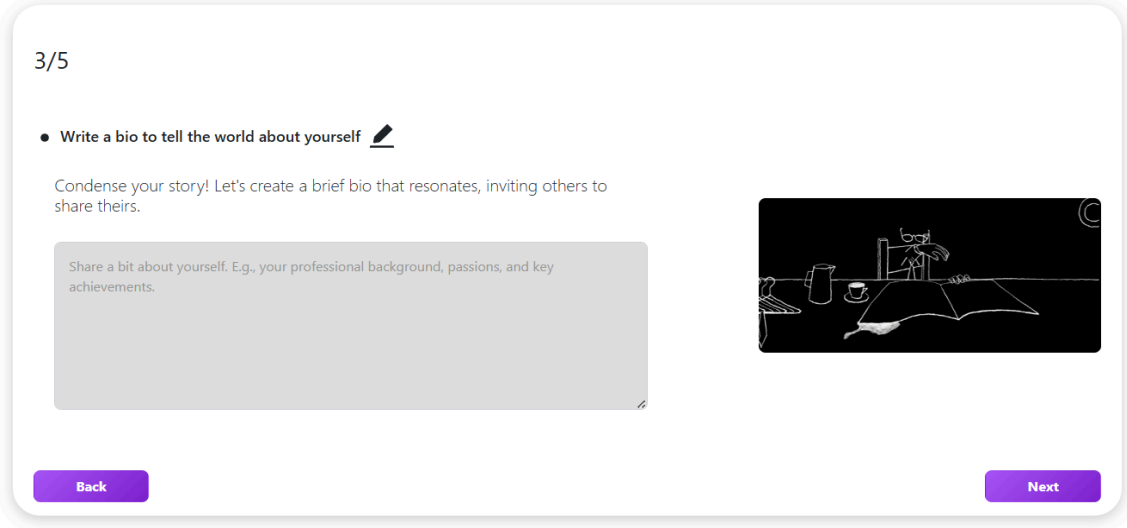
Freelancer Profile Steps: Freelancers can Fill the profile information when registering ServiceSphere

(Figure 35 : Profile step 1)

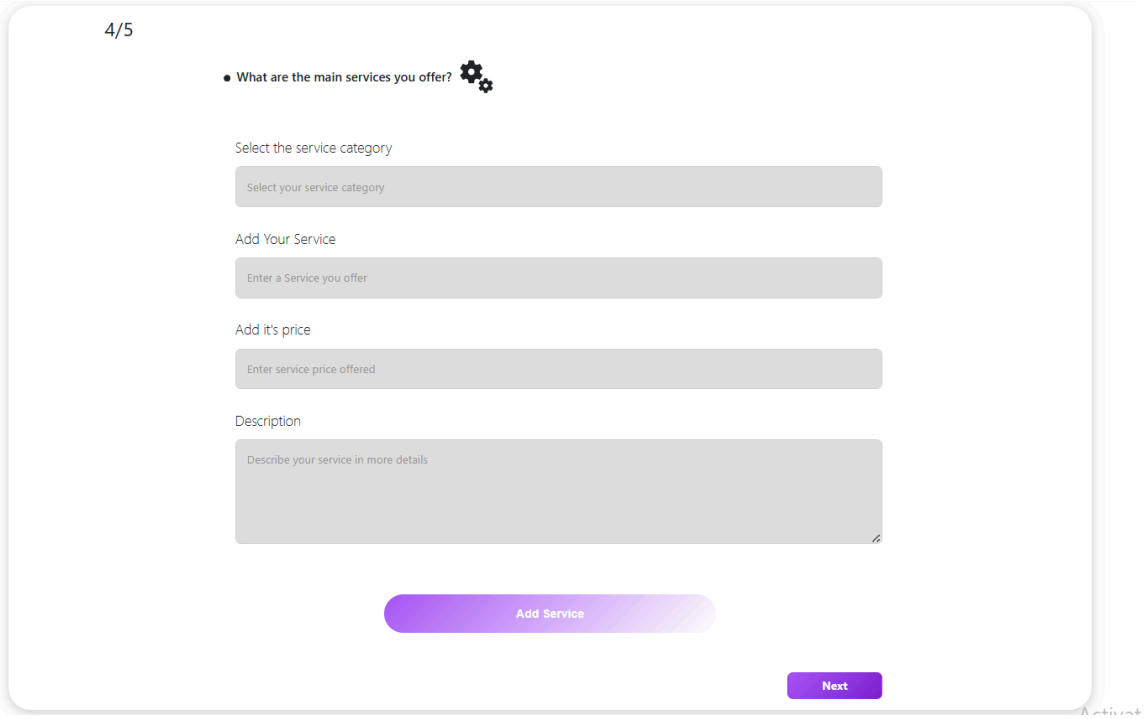




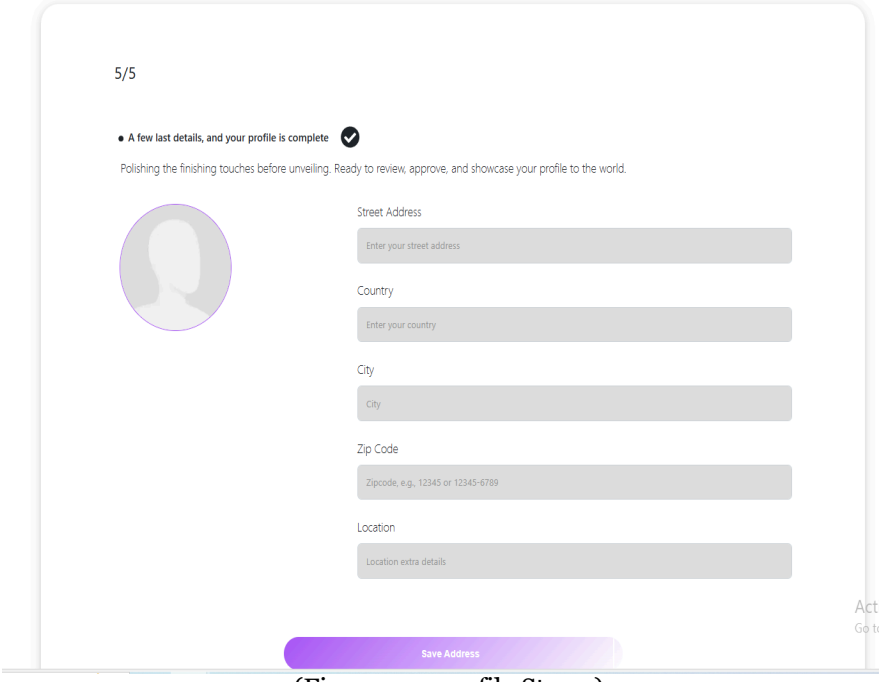
(Figure 36 :profile step 2)



(Figure 37 : profile step 3)

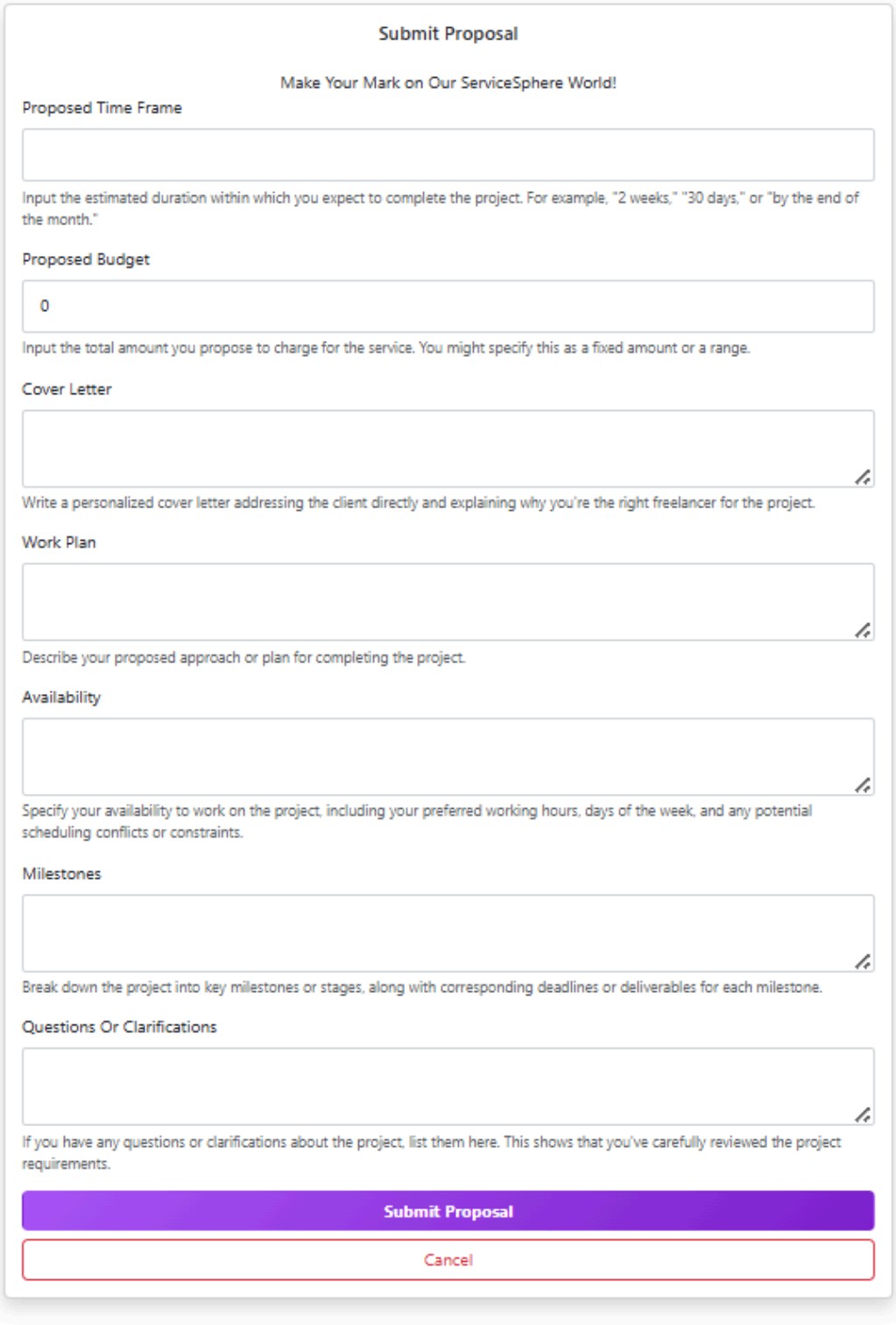


(Figure 38 : profile Step 4)



(Figure 39 : profile Step 5)

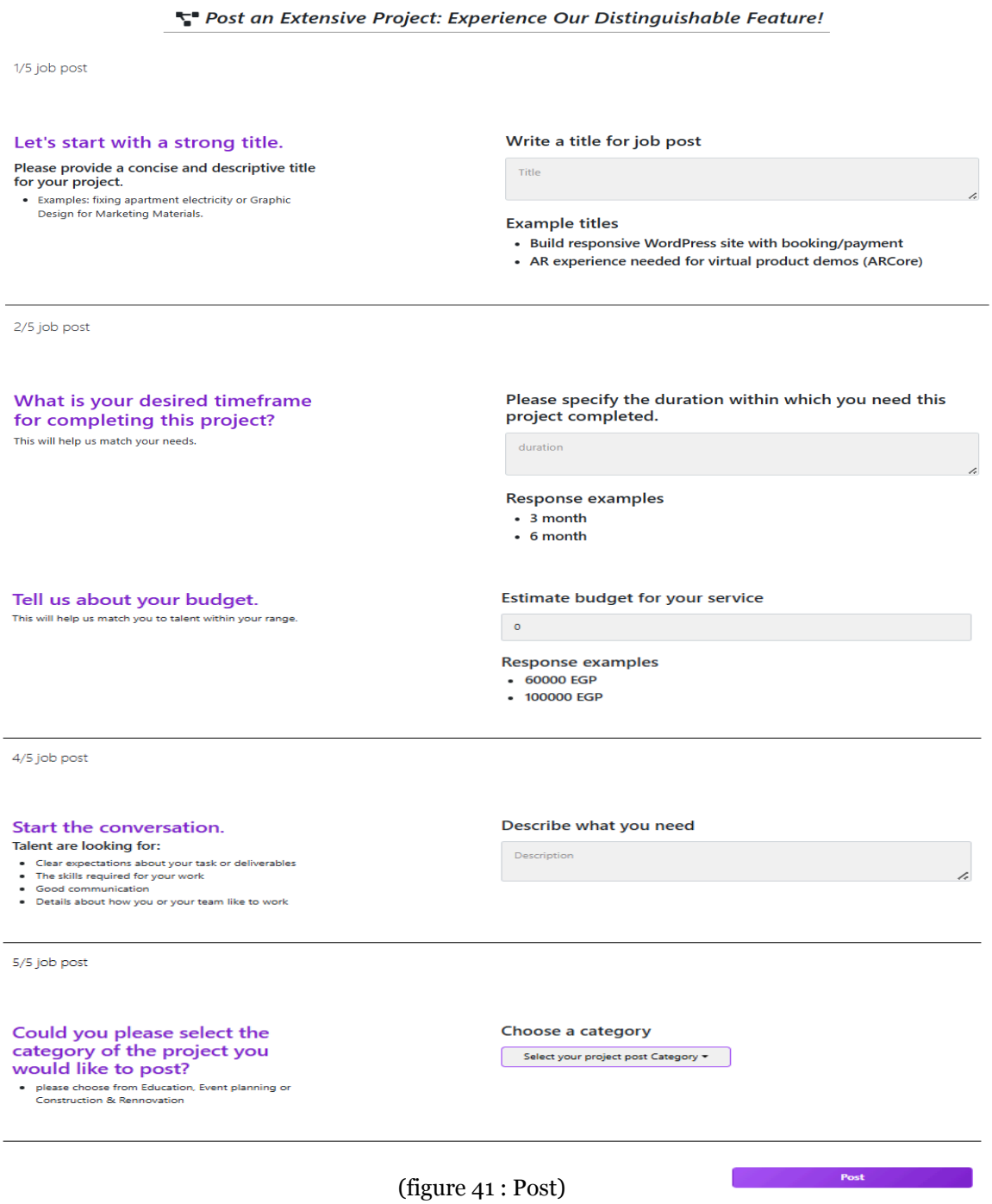
Proposal Submission: Freelancers can view projects and submit proposals.



(Figure 40 : Submit Proposal)

Client Side:

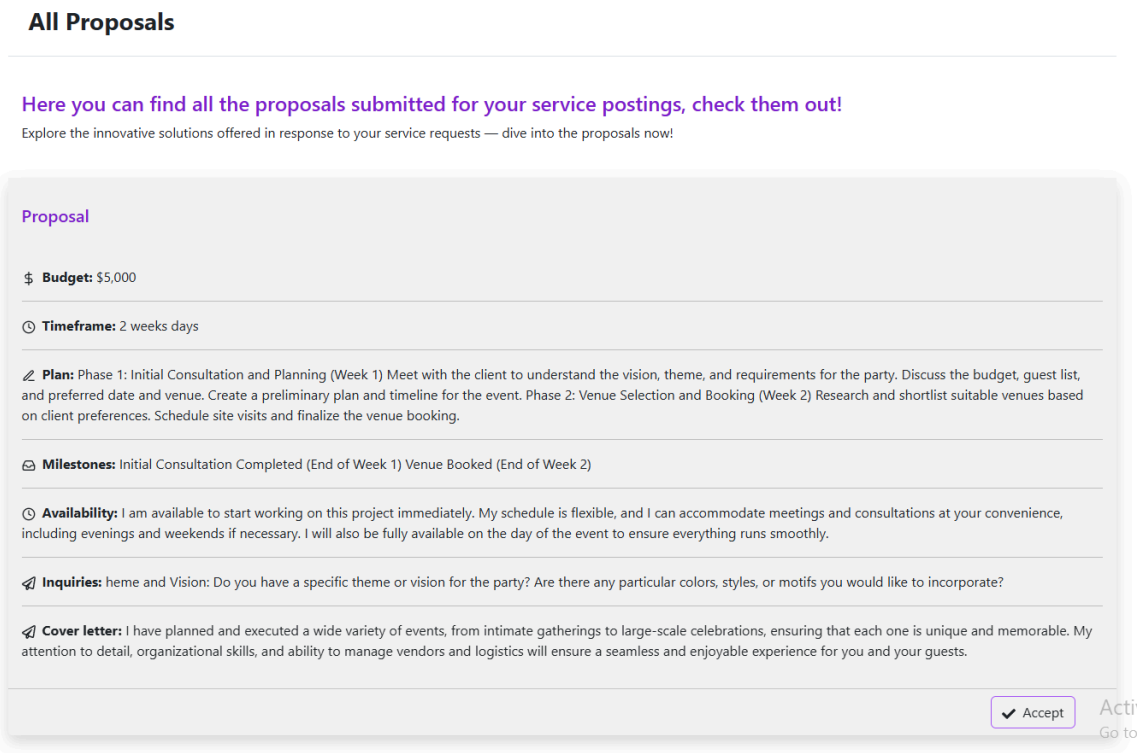
- **Project Posting:** Clients can post projects with detailed descriptions, budgets, and deadlines.



(figure 41 : Post)

-

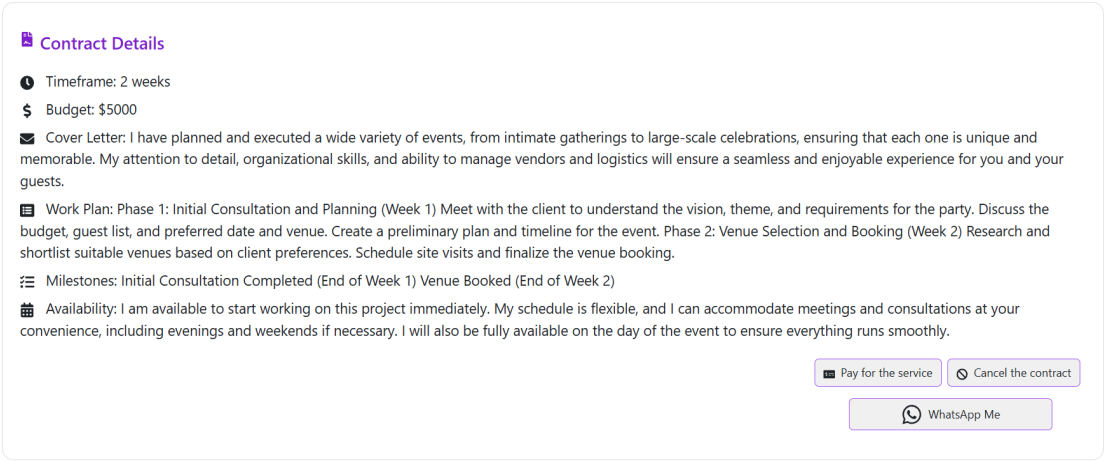
- **Proposal Handling:** clients can accept or reject proposals projects.



(Figure 42 :All Proposals)

- **Contract**

  Contracts formalize the agreement between clients and freelancers, ensuring clarity and protecting the interests of both parties. The platform includes features to facilitate contract creation, acceptance, and management.



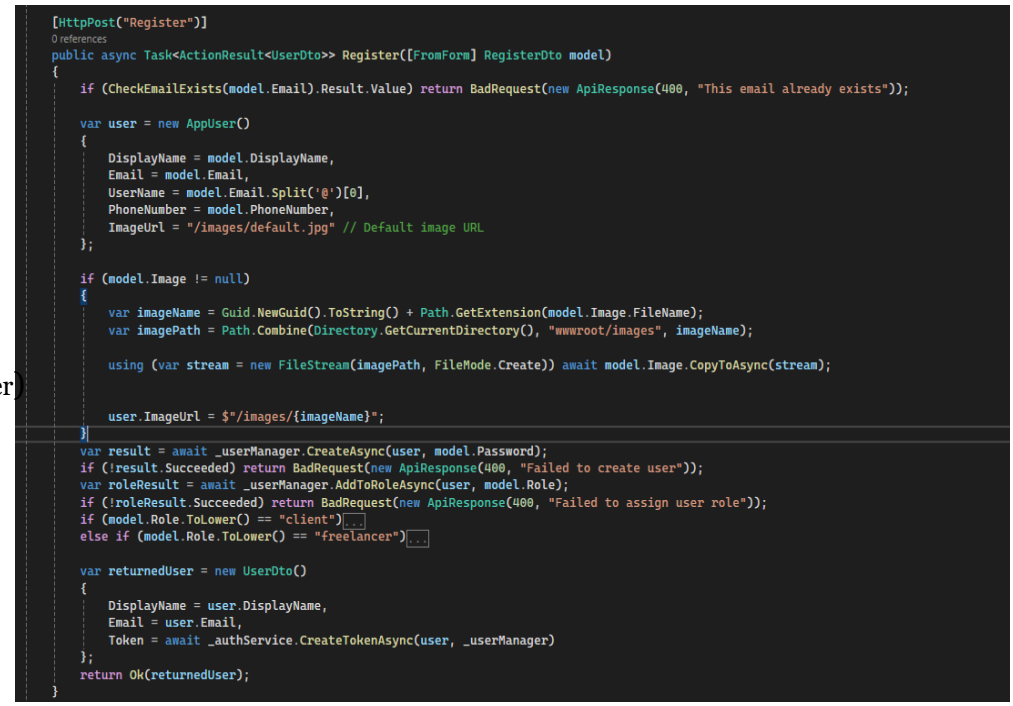
(Figure 43: contract)

5.3.2 Backend Web Development:

The basic version of the freelancing platform includes core functionalities that allow users to perform essential tasks:

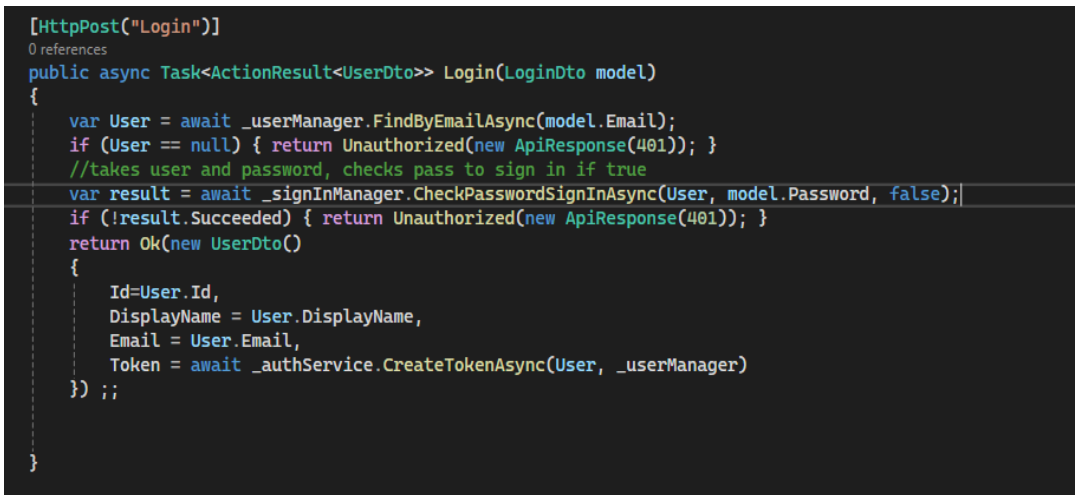
User Registration and Login: Users can create accounts and log in to access the platform.

(figure 44 register)



```
[HttpPost("Register")]
0 references
public async Task<ActionResult<UserDto>> Register([FromForm] RegisterDto model)
{
    if (CheckEmailExists(model.Email).Result.Value) return BadRequest(new ApiResponse(400, "This email already exists"));

    var user = new AppUser()
    {
        DisplayName = model.DisplayName,
        Email = model.Email,
        UserName = model.Email.Split('@')[0],
        PhoneNumber = model.PhoneNumber,
        ImageUrl = "/images/default.jpg" // Default image URL
    };

    if (model.Image != null)
    {
        var imageName = Guid.NewGuid().ToString() + Path.GetExtension(model.Image.FileName);
        var imagePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot/images", imageName);

        using (var stream = new FileStream(imagePath, FileMode.Create)) await model.Image.CopyToAsync(stream);


        user.ImageUrl = $"/images/{imageName}";
    }
    var result = await _userManager.CreateAsync(user, model.Password);
    if (!result.Succeeded) return BadRequest(new ApiResponse(400, "Failed to create user"));
    var roleResult = await _userManager.AddToRoleAsync(user, model.Role);
    if (!roleResult.Succeeded) return BadRequest(new ApiResponse(400, "Failed to assign user role"));
    if (model.Role.ToLower() == "client")...
    else if (model.Role.ToLower() == "freelancer")...

    var returnedUser = new UserDto()
    {
        DisplayName = user.DisplayName,
        Email = user.Email,
        Token = await _authService.CreateTokenAsync(user, _userManager)
    };
    return Ok(returnedUser);
}
```

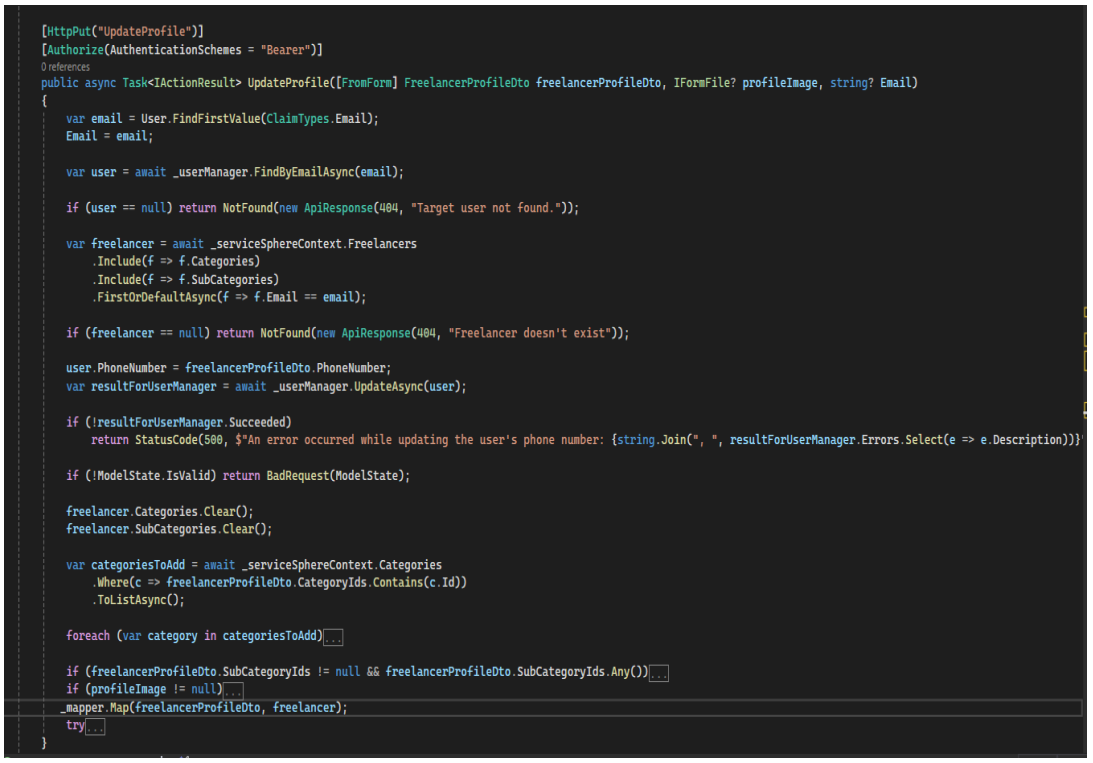
(figure 45 login)



```
[HttpPost("Login")]
0 references
public async Task<ActionResult<UserDto>> Login(LoginDto model)
{
    var User = await _userManager.FindByEmailAsync(model.Email);
    if (User == null) { return Unauthorized(new ApiResponse(401)); }
    //takes user and password, checks pass to sign in if true
    var result = await _signInManager.CheckPasswordSignInAsync(User, model.Password, false);
    if (!result.Succeeded) { return Unauthorized(new ApiResponse(401)); }
    return Ok(new UserDto()
    {
        Id=User.Id,
        DisplayName = User.DisplayName,
        Email = User.Email,
        Token = await _authService.CreateTokenAsync(User, _userManager)
    }) ;;


}
```

Profile Creation: Freelancers can create and update their profiles with skills, experience, and portfolio.

(figure 45 create profile)

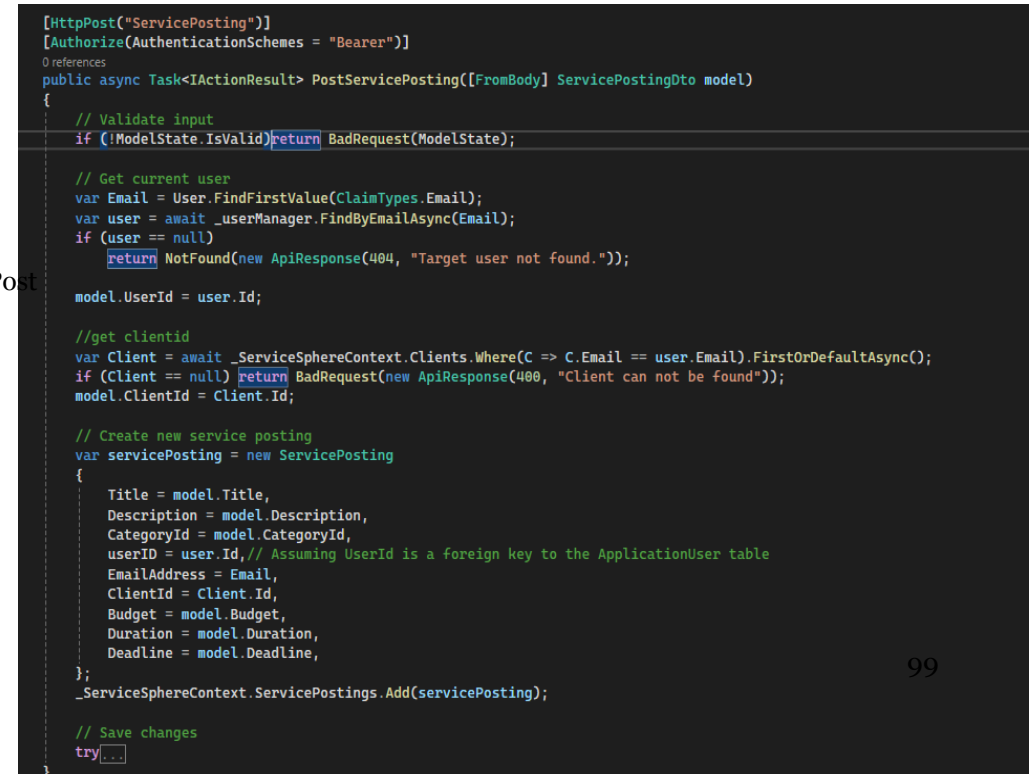


```
[HttpPut("UpdateProfile")]
[Authorize(AuthenticationSchemes = "Bearer")]
0 references
public async Task<IActionResult> UpdateProfile([FromForm] FreelancerProfileDto freelancerProfileDto, IFormFile? profileImage, string? Email)
{
    var email = User.FindFirstValue(ClaimTypes.Email);
    Email = email;

    var user = await _userManager.FindByEmailAsync(email);

    if (user == null) return NotFound(new ApiResponse(404, "Target user not found."));

    var freelancer = await _serviceSphereContext.Freelancers
        .Include(f => f.Categories)
        .Include(f => f.SubCategories)
        .FirstOrDefaultAsync(f => f.Email == email);

    if (freelancer == null) return NotFound(new ApiResponse(404, "Freelancer doesn't exist"));

    user.PhoneNumber = freelancerProfileDto.PhoneNumber;
    var resultForUserManager = await _userManager.UpdateAsync(user);

    if (!resultForUserManager.Succeeded)
        return StatusCode(500, $"An error occurred while updating the user's phone number: {string.Join(", ", resultForUserManager.Errors.Select(e => e.Description))}

    if (!ModelState.IsValid) return BadRequest(ModelState);

    freelancer.Categories.Clear();
    freelancer.SubCategories.Clear();

    var categoriesToAdd = await _serviceSphereContext.Categories
        .Where(c => freelancerProfileDto.CategoryIds.Contains(c.Id))
        .ToListAsync();

    foreach (var category in categoriesToAdd)...

    if (freelancerProfileDto.SubCategoryIds != null && freelancerProfileDto.SubCategoryIds.Any())...
    if (profileImage != null)...
    _mapper.Map(freelancerProfileDto, freelancer);
    try...
}
```

Project Posting: Clients can post projects with detailed descriptions, budgets, and deadlines.

Service Post endpoint:

(figure 46 servicePost endpoint)



```
[HttpPost("ServicePosting")]
[Authorize(AuthenticationSchemes = "Bearer")]
0 references
public async Task<IActionResult> PostServicePosting([FromBody] ServicePostingDto model)
{
    // Validate input
    if (!ModelState.IsValid)return BadRequest(ModelState);

    // Get current user
    var Email = User.FindFirstValue(ClaimTypes.Email);
    var user = await _userManager.FindByEmailAsync(Email);
    if (user == null)
        return NotFound(new ApiResponse(404, "Target user not found."));

    model.UserId = user.Id;

    //get clientid
    var Client = await _ServiceSphereContext.Clients.Where(C => C.Email == user.Email).FirstOrDefaultAsync();
    if (Client == null) return BadRequest(new ApiResponse(400, "Client can not be found"));
    model.ClientId = Client.Id;

    // Create new service posting
    var servicePosting = new ServicePosting
    {
        Title = model.Title,
        Description = model.Description,
        CategoryId = model.CategoryId,
        userID = user.Id,// Assuming UserId is a foreign key to the ApplicationUser table
        EmailAddress = Email,
        ClientId = Client.Id,
        Budget = model.Budget,
        Duration = model.Duration,
        Deadline = model.Deadline,
    };
    _ServiceSphereContext.ServicePostings.Add(servicePosting);

    // Save changes
    try...
}
```

(figure 47 project post endpoint)

```
[HttpPost("ProjectPosting")]
[Authorize(AuthenticationSchemes = "Bearer")]
0 references
public async Task<ActionResult<ProjectPostingDto>> PostProject([FromBody] ProjectPostingDto model)
{
    // Validate input
    if (!ModelState.IsValid)return BadRequest(ModelState);
    // Get current user
    var email = User.FindFirstValue(ClaimTypes.Email);
    var user = await _userManager.FindByEmailAsync(email);
    if (user == null) return NotFound(new ApiResponse(404, "Target user not found."));
    model.UserId = user.Id;
    //get clientid
    var Client = await _ServiceSphereContext.Clients.Where(C => C.Email == user.Email).FirstOrDefaultAsync();
    if (Client == null) return BadRequest(new ApiResponse(400, "Client can not be found"));
    model.ClientId = Client.Id;
    // map project posting
    var projectPosting = _mapper.Map<ProjectPosting>(model);

    // Assuming model includes something like List<SubCategoryRequirement> SelectedSubCategories
    // where SubCategoryRequirement might be a DTO with SubCategoryId and TeamMembersRequired properties
    var projectSubCategories = new List<ProjectSubCategory>();
    foreach (var sc in model.ProjectSubCategories) // Adjust according to your actual structure
    {
        var projectSubCategory = new ProjectSubCategory
        {
            SubCategoryId = sc.SubCategoryId,
            TeamMembersRequired = sc.TeamMembersRequired
        };
        projectSubCategories.Add(projectSubCategory);
    }

    projectPosting.ProjectSubCategories = projectSubCategories;
    // Add the project posting to the context
    _ServiceSphereContext.ProjectPostings.Add(projectPosting);

    // Save changes
    try...
}
```

Proposal Submission: Freelancers can view projects and submit proposals.

(figure 48 : proposal submit)

```
[HttpPost("SubmitProposalForServicePosting")]
[Authorize(AuthenticationSchemes = "Bearer")]
0 references
public async Task<IActionResult> SubmitProposalForServicePosting([FromBody] ProposalDto proposalDto)
{
    var Email = User.FindFirstValue(ClaimTypes.Email);
    var user = await _userManager.FindByEmailAsync(Email);
    if (user == null)
    {
        return NotFound(new ApiResponse(404, "Target user not found."));
    }
    //if (proposalDto.userId != user.Id)
    //{
    //    return Unauthorized(new ApiResponse(404, "You aren't authorized submit a proposal"));
    //}
    proposalDto.userId = user.Id;
    //set freelancer id
    var freelancer = await _serviceSphereContext.Freelancers.Where(F => F.Email == Email).FirstOrDefaultAsync();
    if (freelancer == null) return BadRequest(new ApiResponse(404, "No freelancer found"));
    proposalDto.FreelancerId = freelancer.Id;
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    // Check if the ProjectPosting exists
    var ServicePosting = await _serviceSphereContext.ServicePostings.FindAsync(proposalDto.ServicePostingId);
    if (ServicePosting == null)
    {
        return NotFound($"ServicePosting with ID {proposalDto.ServicePostingId} not found.");
    }

    try...
}
```

Contract: Contracts formalize the agreement between clients and freelancers, ensuring clarity and protecting the interests of both parties. The platform includes features to facilitate contract creation, acceptance, and management.

(figure 49: contract)

```
[Authorize(AuthenticationSchemes = "Bearer")]
[HttpPost("MakePostContract")]
0 references
public async Task<IActionResult> MakeContractForPost(int ProposalId)
{
    var proposal=await _serviceSphereContext.Proposals.Where(P=>P.Id==ProposalId).FirstOrDefaultAsync();
    if (proposal == null) return NotFound(new ApiResponse(404, "Not found Proposal"));
    PostContract postContract = new PostContract();
    postContract.CoverLetter = proposal.CoverLetter;
    postContract.WorkPlan= proposal.WorkPlan;
    postContract.Budget=proposal.ProposedBudget;
    postContract.Availability=proposal.Availability;
    postContract.Milestones=proposal.Milestones;
    postContract.ProposalId = proposal.Id;
    postContract.FreelancerId = proposal.FreelancerId;
    postContract.Timeframe = proposal.ProposedTimeframe;

    //get client id
    var Email = User.FindFirstValue(ClaimTypes.Email);
    var client = await _serviceSphereContext.Clients.Where(F => F.Email == Email).FirstOrDefaultAsync();
    if (client == null) return BadRequest(new ApiResponse(404, "No client found"));
    postContract.ClientId = client.Id;

    //get post contract
    await _postContractRepo.AddAsync(postContract);
   await _serviceSphereContext.SaveChangesAsync();

    var contract = _mapper.Map<PostContractDto>(postContract);

    return Ok(contract);
}
```

Review and Rating System: Clients can leave reviews and ratings for freelancers after project complet

(figure 50

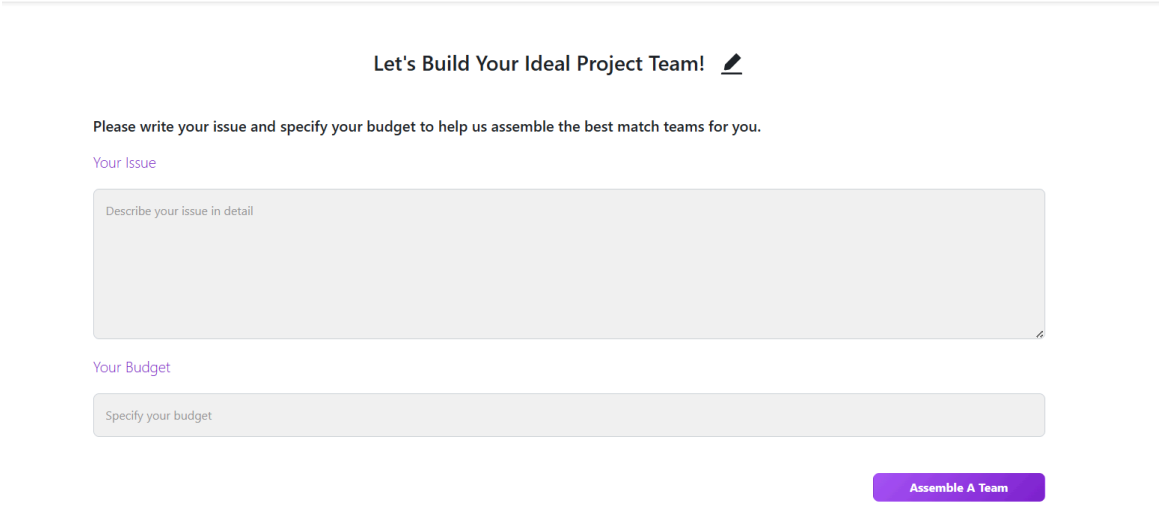
Review endpoint)

```
[Authorize(AuthenticationSchemes = "Bearer")]
[HttpPost("PostReview")]
0 references
public async Task<IActionResult> PostReview([FromBody] ReviewDto reviewDto, string targetUserId)
{
    //var userId = User.FindFirstValue(ClaimTypes.NameIdentifier); // Get the current user's ID
    var Email = User.FindFirstValue(ClaimTypes.Email);
    var user = await _userManager.FindByEmailAsync(Email);


    var targetUser = targetUserId;
    if (targetUser == null)
    {
        return NotFound(new ApiResponse(404,"Target user not found."));
        //return NotFound("Target user not found.");

    }
    var review = new Review
    {
        Description = reviewDto.Description,
        TargetUserId = targetUser,
        Rating = reviewDto.Rating,
        ReviewerId = user.Id, // The current user is the reviewer
        ReviewerName=user.DisplayName,
        Date = DateTime.UtcNow
    };

    _context.Reviews.Add(review);
    await _context.SaveChangesAsync();

    return Ok(new { message = "Review submitted successfully" });
}
```

5.4 Web Improved Features

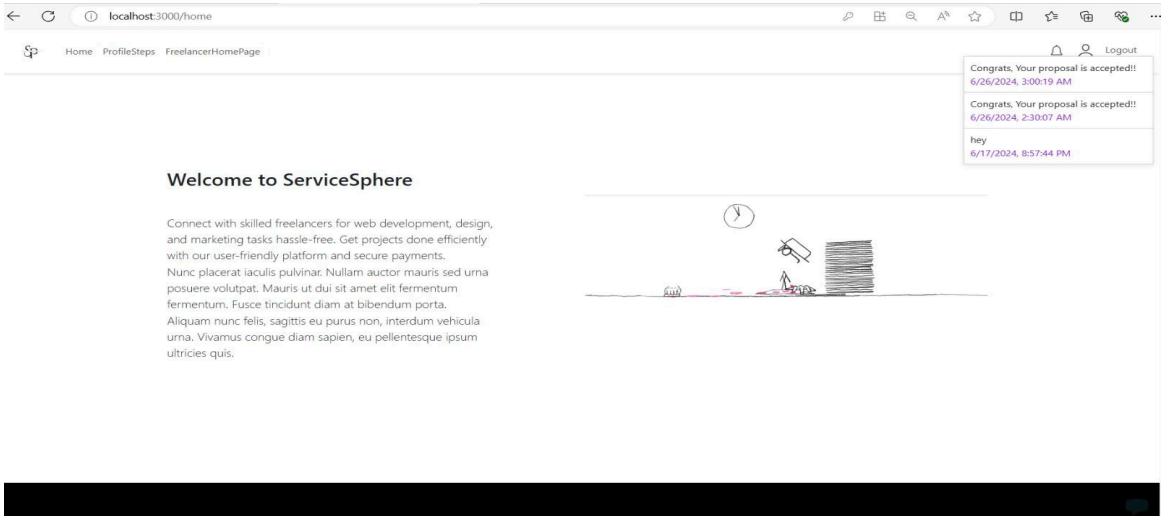
5.4.1 Frontend Advanced Features:

- AI-Based Matching: An AI feature that refines project posts and suggests suitable freelancers based on project requirements.



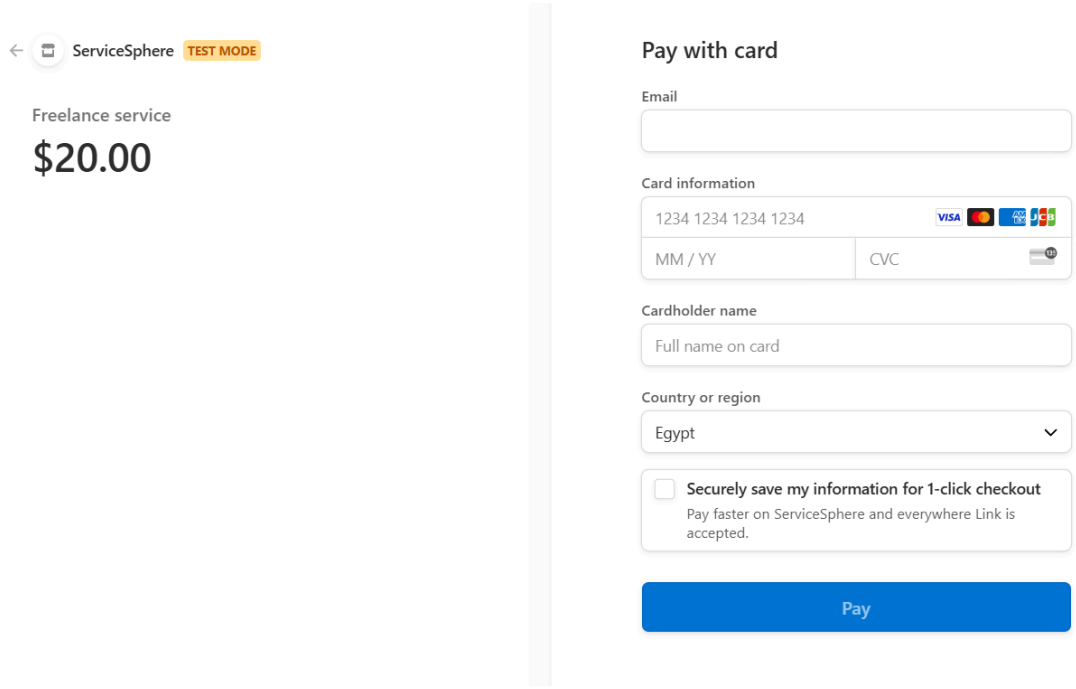
(Figure 51 :AI Match)

- **Notification System:** Real-time notifications for proposals, messages, project updates, and reviews to keep users informed.



(Figure 52 : Notification)

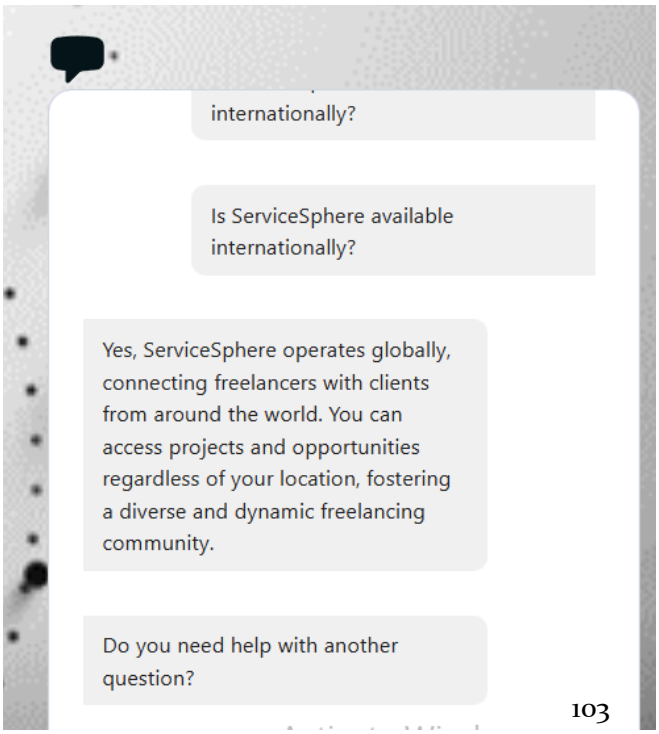
- **Payment Integration:** A secure payment system that holds funds in escrow until project completion, ensuring secure transactions.



(figure 53 : Stripe Payment)

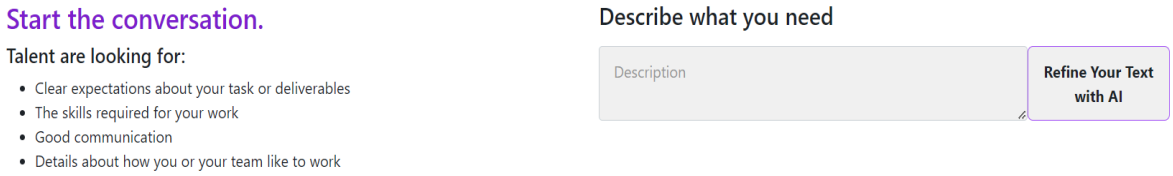
- **Customer Support:** Integrated help center, FAQs, and support channels to assist users with any issues they encounter.

(figure 54 :FAQ chat)



- **Text Refinements**

  Text refinement tools enhance the quality and clarity of project descriptions making interactions more professional and effective.



(figure 54: Description Refinement)

5.4.2 Backend Advanced Features:

Based on user feedback and evolving requirements, several improved features were added to enhance the platform:

- **AI Text Refinements**

  Text refinement tools enhance the quality and clarity of project descriptions, proposals, and communications, making interactions more professional and effective.

(figure 55: Refinement text endpoint)

```
[HttpPost("RefineText")]
0 references
public async Task<IActionResult> Post([FromBody] string userInput)
{
    string openAIKey = "sk-proj-0FKCjNz4PL1kvNfG2JzaT3BlbkFJHbGsEgkyfzGvwtKG0VQC";
    string prompt = $@"
        Project Title: [Enter Project Title Here]

        Project Description:
        The user has reported the following issues:
        {userInput}
        Please enhance the following text to ensure it is both formal and technically accurate.
    The desired format is a post with the problem title positioned at the beginning, succeeded immediately by the project description.
    The text should initiate with the problem statement preceded by 'I' and conclude with a statement akin to 'I'm seeking a team,'
    maintaining simplicity and avoiding the term 'client' at the outset.";

    var requestBody = new
    {
        prompt = prompt,
        model = "gpt-3.5-turbo-instruct",
        max_tokens = 300,
        temperature = 0.5
    };

    string serializedBody = JsonSerializer.Serialize(requestBody);
    _httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", openAIKey);
    _httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

    var content = new StringContent(serializedBody, System.Text.Encoding.UTF8, "application/json");
    var response = await _httpClient.PostAsync("https://api.openai.com/v1/completions", content);

    if (response.IsSuccessStatusCode)
    {
        string responseContent = await response.Content.ReadAsStringAsync();
        return Ok(responseContent);
    }
    else
    {
        return StatusCode((int)response.StatusCode, await response.Content.ReadAsStringAsync());
    }
}
```

- **AI-Based Matching:**

  An AI feature that refines project posts and suggests suitable freelancers based on project requirements.

```
1 reference
private (List<Professional>, List<string>) AssembleTeam(List<string> jobTitlesNeeded, double budget, List<Professional> professionals)
{
    var team = new List<Professional>();
    var unfilledRoles = new List<string>();

    var projectManagers = professionals.Where(p => p.Title.Contains("Project Manager") && p.CostPerHour <= budget).ToList();
    if (projectManagers.Any())
    {
        var selectedManager = projectManagers.OrderByDescending(p => p.Rating).ThenBy(p => p.CostPerHour).First();
        team.Add(selectedManager);
        budget -= selectedManager.CostPerHour;
    }
    else
    {
        return (new List<Professional>(), new List<string> { "Project Manager" });
    }

    foreach (var jobTitle in jobTitlesNeeded)
    {
        var eligibleMembers = professionals.Where(p => p.Title.Equals(jobTitle, StringComparison.OrdinalIgnoreCase) && p.CostPerHour <= budget).ToList();
        if (eligibleMembers.Any())
        {
            var selectedMember = eligibleMembers.OrderByDescending(p => p.Rating).ThenBy(p => p.CostPerHour).First();
            team.Add(selectedMember);
            budget -= selectedMember.CostPerHour;
        }
        else
        {
            unfilledRoles.Add(jobTitle);
        }
    }

    return (team, unfilledRoles);
}
```

(figure 56:AI Refinement endpoint)

- **Notification System:** Real-time notifications for proposals, messages, project updates, and reviews to keep users informed.

```
[HttpGet ("GetUnreadNotifications")]
[Authorize(AuthenticationSchemes = "Bearer")]
0 references
public async Task<ActionResult<IEnumerable<Notification>>> GetUnreadNotifications()
{
    var Email = User.FindFirstValue(ClaimTypes.Email);
    var user = await _userManager.FindByEmailAsync(Email);
    var notifications = await _notificationService.GetUnreadNotificationsAsync(user.Id);
    await _notificationService.SetNotificationReadAsync(user.Id);
    return Ok(notifications);
}

[HttpGet("GetAllNotifications")]
[Authorize(AuthenticationSchemes = "Bearer")]
0 references
public async Task<ActionResult<IEnumerable<Notification>>> GetAllNotifications()
{
    var Email = User.FindFirstValue(ClaimTypes.Email);
    var user = await _userManager.FindByEmailAsync(Email);
    var notifications = await _notificationService.GetAllNotificationsAsync(user.Id);
    await _notificationService.SetNotificationReadAsync(user.Id);
    return Ok(notifications);
}
```

(figure 57 Notification endpoint)

- **Payment Integration:**

  A secure payment system that holds funds in escrow until project completion, ensuring secure transactions.

```
[HttpPost("create-checkout-sessionF")]
0 references
public ActionResult CreateCheckoutSessionF(string freelancerStripeAccountId)
{
    var options = new SessionCreateOptions
    {
        PaymentMethodTypes = new List<string> { "card" },
        LineItems = new List<SessionLineItemOptions>
{
    new SessionLineItemOptions
    {
        PriceData = new SessionLineItemPriceDataOptions
        {
            UnitAmount = 2000, // Price in cents
            Currency = "usd",
            ProductData = new SessionLineItemPriceDataProductDataOptions
            {
                Name = "Freelance service",
            },
        },
        Quantity = 1,
    },
},
        Mode = "payment",
        SuccessUrl = "https://yourdomain.com/success?session_id={CHECKOUT_SESSION_ID}",
        CancelUrl = "https://yourdomain.com/cancel",
        PaymentIntentData = new SessionPaymentIntentDataOptions
        {
            // Optionally specify that the charge collected should be transferred to the connected account
            TransferData = new SessionPaymentIntentDataTransferDataOptions
            {
                Destination = freelancerStripeAccountId,
            },
        },
    };
    var service = new SessionService();
    Session session = service.Create(options);
    return Ok(new { Url = session.Url });
}
```

(figure 58 :Payment endpoint)

5.5 Mobile Application development

Mobile app development involves creating software for mobile devices using network connections for remote computing. The process includes bundling software, backend API implementation, and device testing for optimal performance.

Benefits of mobile apps include:

1. Enhanced Accessibility: Apps extend system accessibility to smartphones and tablets globally.

2. Seamless User Experience: Optimizes user interaction with touch gestures, push notifications, and offline access.

3. Ubiquitous Connectivity: Ensures continuous service access despite internet limitations.

4. Device Integration: Utilizes device capabilities like cameras and GPS.

5. Targeting Mobile-First Users: Meets preferences of mobile-centric users.

6. Branding and Engagement: Strengthens brand presence via app icons and direct user engagement.

Platforms like iOS and Android dominate mobile app development, each with distinct SDKs and toolchains.

Development approaches:

- Native Mobile Apps: Platform-specific development for superior performance and native feature access.

- Cross-Platform Native Apps: Use frameworks like Flutter for single-codebase apps across platforms, compromising slightly on performance.

Native apps are specifically designed for a particular platform like iOS or Android, providing superior performance and full access to device hardware:

- Performance and User Engagement: Native apps offer better performance compared to web-based or hybrid apps due to direct access to device APIs and hardware.
- Platform-Specific Development: Developers use languages like Swift or Objective-C for iOS and Java or Kotlin for Android, optimizing apps for each platform's ecosystem.

- Advantages: Native apps provide excellent UI/UX, utilize platform-specific features like biometric authentication, and offer better security and reliability.
- Disadvantages: Development is platform-specific, requiring separate codebases and maintenance efforts. They also undergo strict app store review processes.

Cross-Platform Mobile Applications

Cross-platform frameworks like Flutter and React Native enable developing apps for multiple platforms using a single codebase:

- Code Reusability: Developers write code once and deploy it across iOS and Android platforms, reducing development time and costs.
- Development Efficiency: Frameworks like React Native (JavaScript) and Flutter (Dart) support hot reload, allowing instant code updates and faster iteration.
- Performance Considerations: While performance may not match native apps, cross-platform frameworks aim to minimize performance gaps through optimizations.
- Adoption and Community Support: These frameworks are popular due to their ease of use, extensive community support, and the ability to build complex UIs.

API Integration in Mobile Applications

APIs play a crucial role in mobile apps by enabling communication with external services, data retrieval, and enhancing app functionalities:

- API Basics: APIs (Application Programming Interfaces) facilitate interaction between different software systems, allowing apps to access data and services.
- RESTful APIs: Commonly used for mobile apps due to their lightweight and scalable nature. They use HTTP requests to perform CRUD (Create, Read, Update, Delete) operations on resources.
- API Security: Important considerations include authentication, encryption, and handling sensitive data securely.
- Integration Benefits: APIs enable integration with third-party services, data synchronization, and real-time updates, enhancing app functionality and user experience.

Flutter Framework for Mobile App Development

Flutter is a popular cross-platform framework developed by Google, known for its performance and UI flexibility:

- Dart Programming Language: Used for Flutter app development, Dart is easy to learn and offers features like asynchronous programming and a strong type system.
- Widget-Based UI Development: Flutter uses widgets as the building blocks for UI components, providing flexibility and customization options.
- Hot Reload: Key feature allowing developers to see changes instantly during development, speeding up the iteration process.
- Platform-Specific APIs: Flutter provides plugins for accessing platform-specific APIs and functionalities, ensuring native-like performance and behavior.

Future Trends and Considerations

Mobile app development continues to evolve with advancements in AI, AR/VR, and IoT integration. Developers should consider:

- Emerging Technologies: Integration of AI for personalized experiences, AR/VR for immersive interactions, and IoT for connected devices.
- Security and Privacy: Focus on data protection regulations and user privacy concerns, ensuring apps comply with global standards.
- Cloud Integration: Leveraging cloud services for scalability, data storage, and backend management to support growing user bases.

By focusing on these areas, developers can create robust, secure, and engaging mobile applications that meet the evolving demands of users and businesses alike.

Flutter, by Google, enables cross-platform app development with fast iteration and native performance benefits.

APIs are vital for mobile app functionality, facilitating interactions with external services and data integration, enhancing user experience and app capabilities.

Security and Privacy Considerations

1. Data Encryption and Secure Storage:
    - Apps must encrypt sensitive data both in transit (using HTTPS/SSL) and at rest (using encryption algorithms) to prevent unauthorized access.
    - Secure storage practices ensure that user credentials, payment information, and personal data are protected against breaches.
2. User Authentication and Authorization:
    - Implementing robust authentication mechanisms (e.g., OAuth, biometric authentication) ensures that only authorized users can access sensitive app features and data.
    - Multi-factor authentication (MFA) adds an extra layer of security by requiring multiple credentials for user verification.
3. Compliance with Data Protection Regulations:
    - Apps must adhere to global regulations like GDPR, CCPA, and HIPAA, which govern data collection, processing, and user consent.
    - Privacy policies and transparent data practices build user trust and compliance with legal requirements.

Cloud Integration and Backend Services

1. Scalability and Performance:
    - Cloud services (e.g., AWS, Google Cloud, Azure) provide scalable infrastructure for app backend, storage, and computing needs.
    - Serverless architectures (e.g., AWS Lambda, Google Cloud Functions) optimize resource allocation and cost efficiency.
2. Backend as a Service (BaaS):
    - BaaS platforms (e.g., Firebase, Parse) offer pre-built backend functionalities (e.g., databases, user authentication, analytics) that accelerate app development.
    - API integration with BaaS facilitates real-time updates, offline data synchronization, and seamless user experiences across devices.
3. Microservices and Containerization:
    - Adopting microservices architecture and containerization (e.g., Docker, Kubernetes) enables modular development, deployment flexibility, and easier scalability.

- Container orchestration platforms manage app lifecycle, resource allocation, and auto-scaling to optimize performance and reliability.

User Experience (UX) and Interface Design

1. Responsive Design and Accessibility:
    - Designing apps with responsive layouts ensures usability across various screen sizes and orientations (mobile, tablet, desktop).
    - Accessibility features (e.g., screen readers, text resizing, color contrast) accommodate users with disabilities and enhance inclusivity.
2. Gesture-Based Interactions and UI Patterns:
    - Intuitive UI/UX design incorporates gesture-based controls, swipe actions, and interactive animations to enhance user engagement.
    - Consistent UI patterns (e.g., Material Design for Android, Human Interface Guidelines for iOS) provide familiar navigation and usability standards.
3. Usability Testing and Iterative Design:
    - Conducting usability tests and gathering user feedback iteratively improves app usability, identifies pain points, and validates design decisions.
    - A/B testing, heatmaps, and analytics tools measure user behavior, interactions, and conversion rates to optimize app performance and user satisfaction.

Conclusion

Mobile application development continues to evolve with advancements in technology, user expectations, and industry standards. By embracing emerging technologies, prioritizing security and privacy, leveraging cloud services, and focusing on user-centric design principles, developers can create innovative, scalable, and secure mobile apps that deliver exceptional user experiences across platforms.

5.5.1 code implementation for mobile

5.5.1.1 Common Features (client/freelancer)

- Signup**:**

User can Create an Account by passing:

-First Name

-Last Name

-Phone Number

-Email

-Password

-Role (Client / Freelancer)

```
late RegisterModel registerModel;
Future<void> register(String userName, String email, String password,
    String phone, String role) async {
  emit(Registering());
  final NetworkService response = await _authenticationRepository.register(
      userName, email, password, phone, role);
  response.when(succeed: (loadedData) {
    registerModel = loadedData;
    emit(RegisteringSucceed());
  }, failure: (NetworkExceptions error) {
    emit(RegisteringFailed(networkExceptions: error));      You, last month •
  });
}
```

(figure 59 Register Model)

Login

User Can Login into his Account by Passing

-Email

-Password

```
LoginModel? loginModel;
Future<void> login(
    String email, String password, BuildContext context) async {
  emit(LoggingIn());
  final NetworkService response =
      await _authenticationRepository.login(email, password);
  response.when(succeed: (loadedData) {
    loginModel = loadedData;
    getIt<SharedPreferences>().setString('token', loginModel!.token!);
    getIt<SharedPreferences>().setString('id', loginModel!.id!);
    getIt<SharedPreferences>().setString('email', loginModel!.email!);
    getIt<SharedPreferences>()
        .setString('userType', loginModel!.userType!.toString());
    emit(LoggingInSucceed());
  }, failure: (NetworkExceptions error) {
    emit(LoggingInFailed(networkExceptions: error));
  });
}
```

(figure 60 loginModel)

Sending/viewing Notifications

```
List<NotificationsModel> proposalsList = [];
Future getUserNotifications() async {
  emit(LoadingNotifications());
  final NetworkService data =
      await _notificationsRepository.getNotifications();
  data.when(succeed: (loadedData) {
    proposalsList = loadedData;
    emit((NotificationsLoadedSuccessfully()));
  }, failure: (failure) {
    emit((LoadingNotificationsFailed()));
  });
}

Future sendNotification(String message) async {
  emit(LoadingNotifications());
  final NetworkService data =
      await _notificationsRepository.sendNotification(message);      You, 5
  data.when(succeed: (loadedData) {
    // proposalsList = loadedData;
    emit((NotificationsLoadedSuccessfully()));
  }, failure: (failure) {
    emit((LoadingNotificationsFailed()));
  });
}
}
```

(figure 60 Notification)

Payment

```
Future pay() async {
  final NetworkService data = await _paymentRepository.pay();
  data.when(succeed: (succeed) {
    emit(state);
  }, failure: (failure) {
    emit(state);
  });
 }
}
```

(figure 61 payment)

5.5.1.2 Client Features

- **Post (Project/Service)**

User is allowed to post a new job whether it's Project or Service

```
Future postService(Map<String, dynamic> body) async {
  emit(Posting());
  NetworkService data = await _postingsRepository.postService(body);
  data.when(succeed: (loadedData) {
    emit(PostedSuccessfully());
  }, failure: (failure) {
    emit(PostingFailure());
  });
}
    You, last week • very good steps
Future postProject(Map<String, dynamic> body) async {
  emit(Posting());
  NetworkService data = await _postingsRepository.postProject(body);
  data.when(succeed: (loadedData) {
    emit(PostedSuccessfully());
  }, failure: (failure) {
    emit(PostingFailure());
  });
}
```

(figure 62 Post)

- **Viewing/Editing/Deleting Posts**

The Client is allowed to View or edit, delete his Job Posting

(Delete)

```
Future deletePost(PostType postType, int id) async {
  emit(DeletingPost());
  NetworkService data = await _postingsRepository.deletePost(postType, id);
  data.when(succeed: (loadedData) {
    emit(PostDeletedSuccessfully());
    getAllPostings();
  }, failure: (failure) {
    emit(DeletingPostFailed());
  });
}
```

(figure 63 Delete Post)

(Edit)

```
Future editPost(PostType postType, int id, Map<String, dynamic> body) async {
  emit(EditingPost());
  NetworkService data =
      await _postingsRepository.editPost(postType, id, body);
  data.when(succeed: (loadedData) {
    emit(PostEditedSuccessfully());
  }, failure: (failure) {
    emit(EditingPostPostFailed());      You, 1 second ago • Uncommitted chang
  });
}
}
```

(figure 64 edit post)

(View)

```
Future getAllPostings({int? categoryId}) async {
  emit(LoadingPostings());
  // Future.delayed(const Duration(seconds: 2));
  try {
    Future.wait([
      getProjectsPostings(categoryId: categoryId),
      getServicePostings(categoryId: categoryId)
    ]);
    if (projectsPostsModel.isNotEmpty && servicesPostsModel.isNotEmpty) {
      emit(PostingsLoadedSuccessfully());

      allPosts = [...projectsPostsModel, ...servicesPostsModel];      You, 4
    }
  } catch (e) {
```

- (figure 65 Get All Posts)

Talents

The Client is allowed to look for special talents

```
List<TalentsModel>? talentsModel;
final TalentsRepository _talentsRepository;
TalentsCubit(this._talentsRepository) : super(TalentsStateInitial());
static TalentsCubit get(BuildContext context) =>
    BlocProvider.of<TalentsCubit>(context);
Future getFreelancers() async {
  emit(LoadingTalents());
  NetworkService data = await _talentsRepository.getFreelancers();
  data.when(succeed: (loadedData) {
    talentsModel = loadedData;
    emit(TalentsLoadedSuccessfully());
  }, failure: (failure) {
    emit(TalentsLoadingFailed());
  });
}
```

(figure 66 Talents)

- **Proposal**

The Client is allowed to view the sent Proposals and accept or reject it

```
List<ProposalModel> proposalsList = [];      You, 4 hours ago • 11:24 PM
Future getUserProposals() async {
    emit(LoadingProposals());
    final NetworkService data = await _proposalRepository.getUserProposals();
    data.when(succeed: (loadedData) {
      proposalsList = loadedData;
      emit(ProposalsLoadedSuccessfully());
    }, failure: (failure) {
      emit(ProposalsLoadingFailure());
    });
  }
```

(figure 67 Proposals)

(Accept)

```
Future acceptProposal(int proposalId) async {
  emit(LoadingProposals());
  final NetworkService data =
      await _proposalRepository.acceptProposal(proposalId);
  data.when(succeed: (loadedData) {
    emit(ProposalsLoadedSuccessfully());
  }, failure: (failure) {
    emit(ProposalsLoadingFailure());
  });
 }
}
```

(figure 67 Accept Proposal)

5.5.1.3 Freelancer Features

- **Find Work**

The Freelancer can find work based on his category

```
Future getAllPostings({int? categoryId}) async {
  emit(LoadingPostings());
  // Future.delayed(const Duration(seconds: 2));
  try {
    Future.wait([
      getProjectsPostings(categoryId: categoryId),
      getServicePostings(categoryId: categoryId)
    ]);
    if (projectsPostsModel.isNotEmpty && servicesPostsModel.isNotEmpty) {
      emit(PostingsLoadedSuccessfully());

      allPosts = [...projectsPostsModel, ...servicesPostsModel];      You, 4
    }
  } catch (e) {
    emit(PostingsLoadingFailed());
  }
 }
```

(figure 68 get all proposal)

- **Submitting Proposal**

The Freelancer Submits a proposal for a work

```
ProposalModel? proposalModel;
Future submitProposal(PostType postType, Map<String, dynamic> body) async {
  emit(SubmittingProposal());
  final NetworkService data =
      await _proposalRepository.submitProposal(postType, body);
  data.when(succeed: (loadedData) {
    proposalModel = loadedData;
    emit(SubmittingSucceeded());
  }, failure: (failure) {
    emit(SubmittingFailure());
  });
}
```

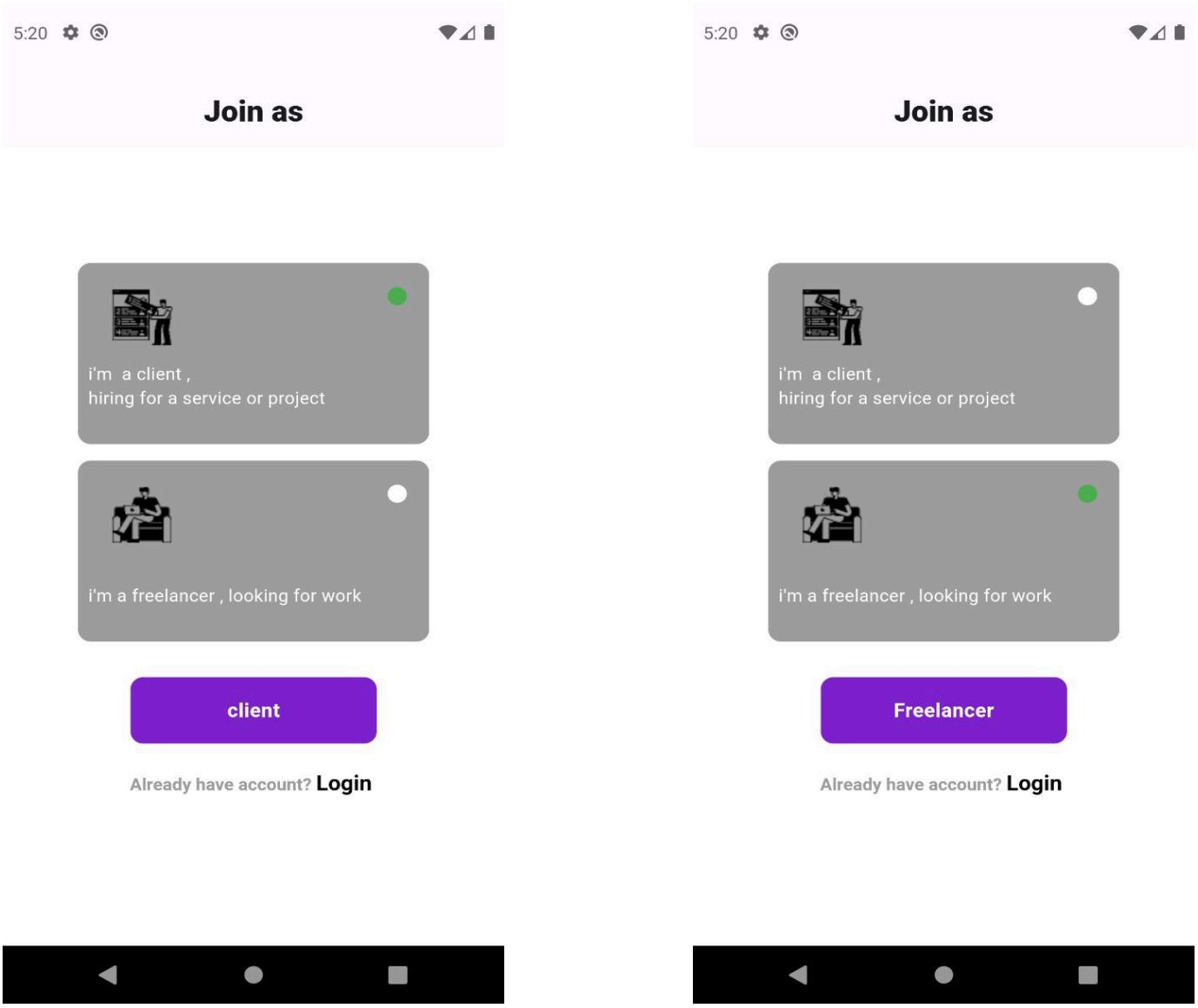
(figure 69 submit proposal)

5.5.2 Mobile Interface

Select User Role :

you can Signup as a (client / freelancer)

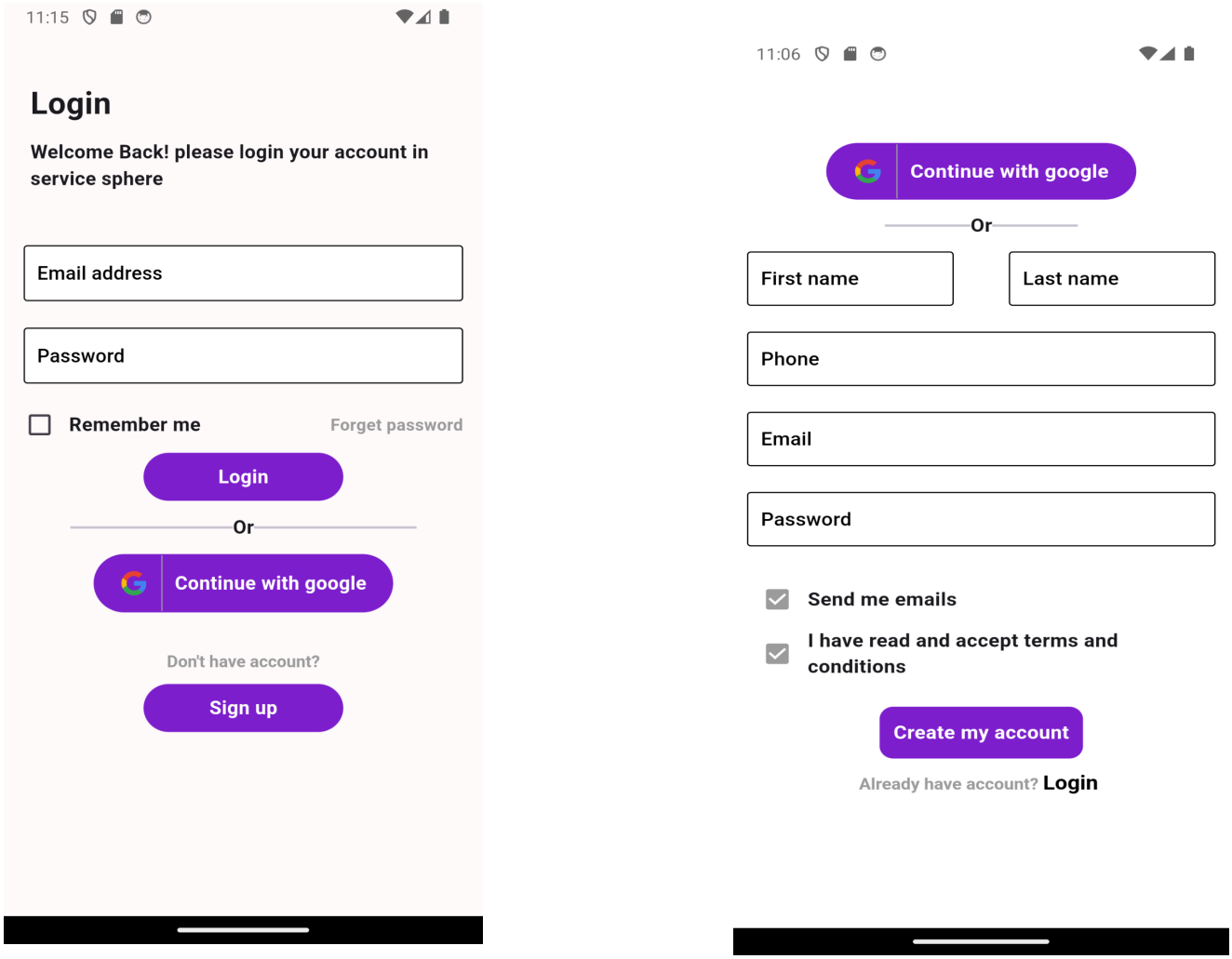
(figure 70 login)



(figure 71 Signup)

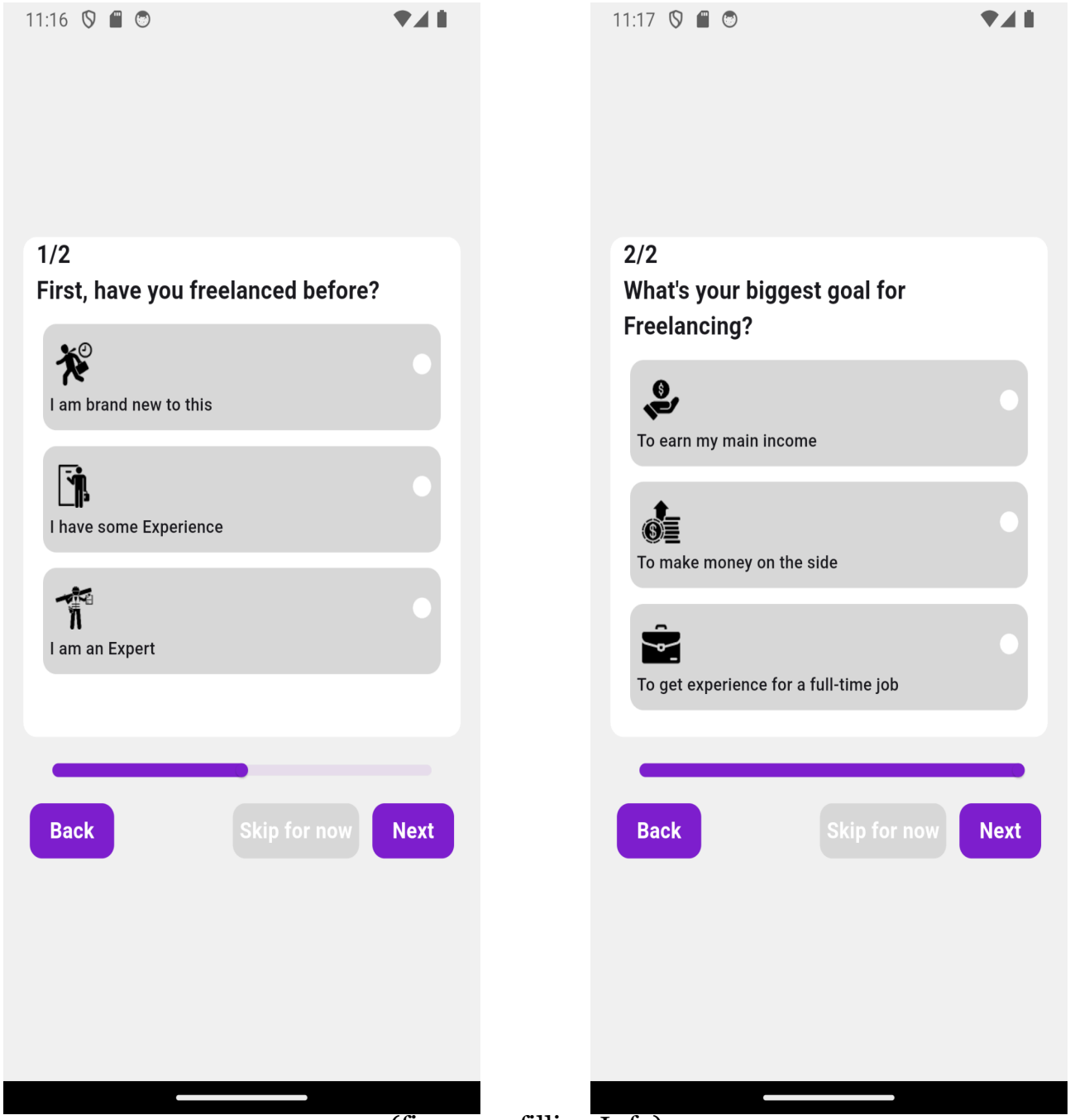
Login :

signup:

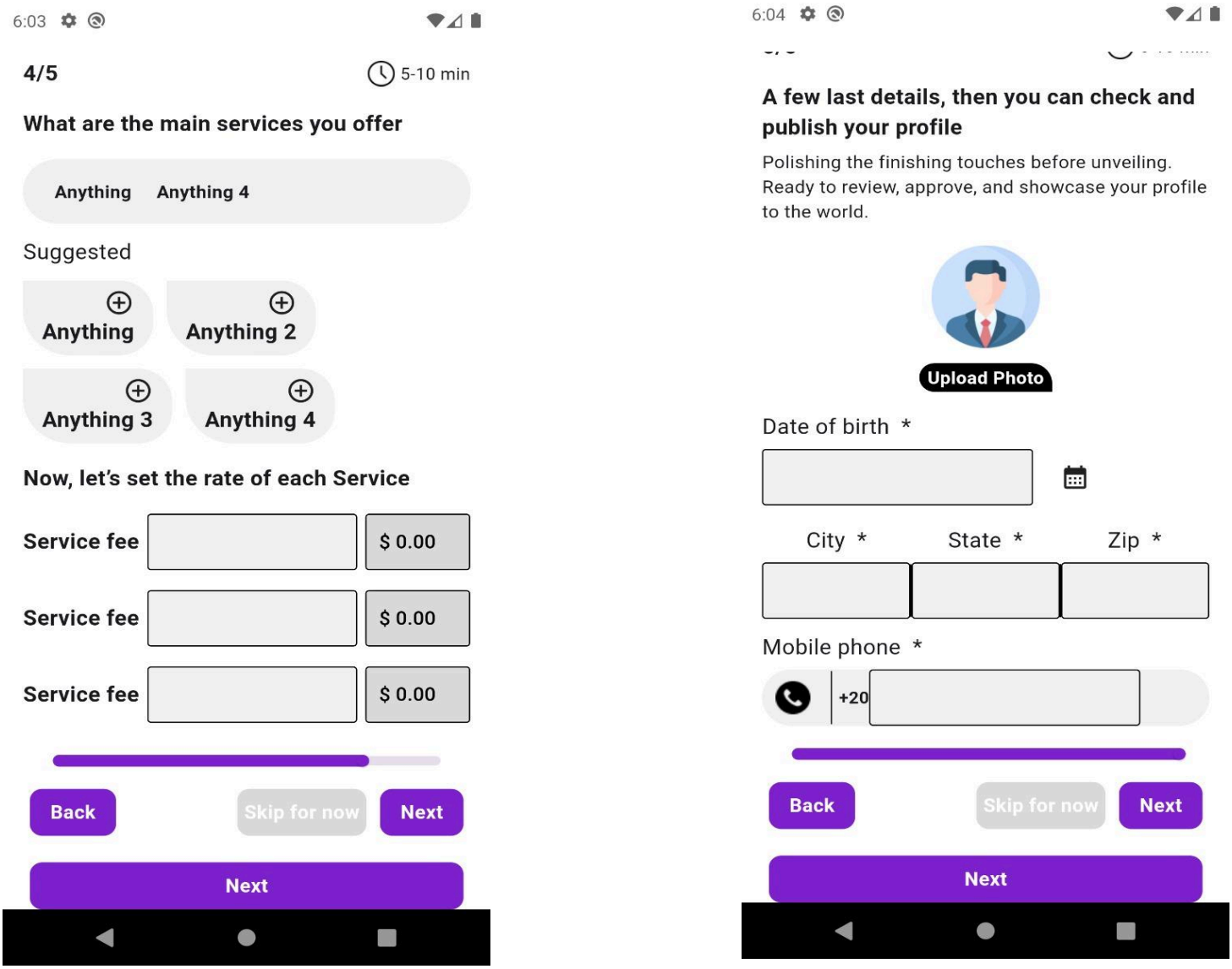


(figure 72 login-register)

freelancer information :



(figure 73 filling Info)



(figure 73 filling info 2)

Chapter 6
Testing and Evaluation

6.1 Functional Testing

Functional testing ensures that each feature of the freelancing platform works according to the requirements. This type of testing verifies that the system performs its intended functions correctly. Below is a detailed explanation of the functional testing for the various features of the freelancing platform.

1. Client Posting a Service post or a Project post

Objective: Verify that clients can successfully create and post a service post or project post.

Tests:

- Create Post: Ensure the client can navigate to the post creation page, fill in necessary details (title, description, category, budget, deadline), and successfully submit the post.
- View Post: Verify that the post appears on the platform with all the entered details.
- Edit Post: Test the client's ability to edit the post details and ensure changes are updated.
- Delete Post: Confirm that the client can delete the post and that it no longer appears on the platform.

2. Freelancer Proposals

Objective: Ensure freelancers can view service or project posts and submit proposals.

Tests:

- View Posts: Verify that freelancers can see all available posts and their details.
- Submit Proposal: Check that freelancers can submit proposals with required information (proposal text, budget, timeframe).
- Edit Proposal: Ensure freelancers can edit their proposals before the client reviews them.
- Withdraw Proposal: Confirm that freelancers can withdraw their proposals if needed.

3. Client Review and Decision on Proposals

Objective: Ensure clients can review, accept, or reject proposals submitted by freelancers.

Tests:

- **View Proposals:** Verify that clients can see all proposals for their post.
- **Accept Proposal:** Check that clients can accept a proposal, triggering necessary notifications and agreements.

4. Reviews

Objective: Ensure that clients can leave reviews for freelancers after a project is completed.

Tests:

- **Submit Review:** Verify that clients can submit a review and rating for a freelancer.
- **View Reviews:** Ensure reviews are visible on the freelancer's profile.

5. Freelancer Profiles

Objective: Verify that freelancers can create and manage their profiles. **Tests:**

- **Create Profile:** Ensure freelancers can create a profile with personal information, skills, services offered, and a profile picture.
- **Edit Profile:** Check that freelancers can update their profile information.
- **View Profile:** Confirm that clients can view freelancer profiles, including services offered and reviews.

6. Client Exploration of Freelancers

Objective: Ensure clients can search for and explore freelancers.

Tests:

- **Search Function:** Verify the functionality of filtering and sorting options.
- **Profile View:** Ensure clients can click on freelancer profiles and view detailed information.
- **Service Request:** Confirm that clients can request a service directly from the freelancer's profile.

7. AI-based Text Refinement and Freelancer Matching

Objective: Ensure the AI functionality assists clients in refining their post text and matches them with suitable freelancers.

Tests:

- **Text Refinement:** Verify that the AI can refine and enhance the client's post text.
- **Freelancer Matching:** Ensure the AI suggests relevant freelancers based on the post's details.
- **Accuracy:** Test the accuracy and relevance of AI suggestions.
- **Usability:** Confirm that clients find the AI refinement and matching process user-friendly and efficient.

8. Notification System

Objective: Verify that both clients and freelancers receive timely and accurate notifications regarding proposals, messages, project updates, and reviews.

Tests:

- **Proposal Notifications:** Ensure clients receive notifications when new proposals are submitted and freelancers receive notifications on proposal acceptance.
- **Project Update Notifications:** Confirm that notifications are sent for project status updates.

9. Payment System

Objective: Ensure that the payment gateway is secure and functional, allowing clients to make payments and freelancers to receive payments without issues.

Tests:

- **Payment Processing:** Verify that clients can successfully make payments for services.
- **Security:** Test the security of the stripe gateway, including encryption and fraud prevention measures.

10. Messaging and Communication

Objective: Ensure clear and effective communication between clients and freelancers through Whatsapp integration.

Tests:

- **Send and Receive Messages:** Verify that users are redirected to a whatsapp chat with the provided phone number.

11. User Authentication and Security

Objective: Verify the security measures in place for user authentication, including password management and data encryption.

Tests:

- **Account Creation:** Ensure users can create accounts securely.
- **Login/Logout:** Verify that users can log in and out securely.
- **Data Encryption:** Confirm that user data is encrypted both in transit and at rest.

13. Mobile Compatibility

Objective: Ensure the platform's functionality on various mobile devices and browsers for a seamless experience across different platforms.

Tests:

- **Responsive Design:** Verify that the platform's layout adapts to different screen sizes.
- **Functionality:** Test all platform features on various mobile devices and browsers.
- **Performance:** Check the performance and loading times on mobile devices.
- **User Experience:** Ensure a consistent and user-friendly experience on mobile.

14. Customer Support

Objective: Verify the availability and effectiveness of customer support features FAQs through ChatBot.

Tests:

- **FAQs:** Verify that the ChatBot FAQ section covers common questions and issues.

Conclusion

Functional testing is crucial for ensuring that each feature of the freelancing platform works as intended. By systematically testing each function, we can identify and address any issues, ensuring a smooth and reliable user experience for both clients and freelancers.

6.2 User Experience Testing for the Freelancing Platform

User Experience (UX) Testing focuses on evaluating the overall experience of users as they interact with the freelancing platform. This involves assessing usability, accessibility, design consistency, and overall satisfaction to ensure that the platform is user-friendly and meets the needs of both clients and freelancers. Below is a detailed explanation of the various aspects of UX testing.

1. Usability Testing

Objective: Assess how easily users can navigate and use the platform to achieve their goals.

Tests:

- **Navigation:** Ensure users can easily find their way around the platform, from the homepage to specific features like posting a project or viewing proposals.
- **Task Completion:** Test how quickly and efficiently users can complete key tasks, such as creating a profile, submitting a proposal, or making a payment.
- **Error Handling:** Verify that the platform provides clear and helpful error messages when users make mistakes (e.g., entering incorrect information).
- **Learnability:** Evaluate how quickly new users can learn to use the platform effectively.

2. Accessibility Testing

Objective: Ensure that the platform is accessible to users with disabilities, complying with accessibility standards such as WCAG (Web Content Accessibility Guidelines).

Tests:

- **Keyboard Navigation:** Confirm that all functionalities can be accessed using a keyboard alone, without the need for a mouse.
- **Screen Reader Compatibility:** Ensure that the platform is compatible with screen readers, providing meaningful and clear information for visually impaired users.
- **Color Contrast:** Verify that there is sufficient contrast between text and background colors to be readable by users with visual impairments.
- **Alternative Text:** Check that all images and multimedia have descriptive alternative text for users who rely on screen readers.

3. Design Consistency

Objective: Ensure a consistent design across the platform to provide a cohesive user experience.

Tests:

- **Visual Consistency:** Verify that fonts, colors, buttons, and other visual elements are consistent across different pages and sections.
- **Functional Consistency:** Ensure that similar actions (e.g., submitting forms, navigating menus) work in the same way throughout the platform.
- **Brand Alignment:** Confirm that the platform's design aligns with the overall branding guidelines and provides a professional appearance.

4. Performance Testing

Objective: Evaluate the platform's performance to ensure a smooth and responsive user experience.

Tests:

- **Load Time:** Measure the time it takes for pages to load, ensuring they load quickly even with heavy traffic.
- **Responsiveness:** Verify that the platform responds quickly to user inputs, such as clicks and form submissions.
- **Scalability:** Test the platform's ability to handle a large number of simultaneous users without performance degradation.

5. Mobile Experience

Objective: Ensure that the platform provides an excellent user experience on mobile devices.

Tests:

- **Responsive Design:** Verify that the platform's layout adapts to different screen sizes and orientations.
- **Mobile Navigation:** Ensure that navigation is intuitive and easy to use on mobile devices, with accessible menus and buttons.
- **Touch Interactions:** Test touch interactions, such as tapping, swiping, and pinching, to ensure they work smoothly on mobile devices.
- **Performance on Mobile:** Check the loading times and responsiveness on various mobile devices and browsers.

6. User Feedback and Satisfaction

Objective: Gather and analyze user feedback to understand their satisfaction and identify areas for improvement.

Tests:

- **Surveys and Questionnaires:** Conduct surveys to gather users' opinions on various aspects of the platform, such as ease of use, design, and overall satisfaction.
- **User Interviews:** Perform in-depth interviews with users to gain insights into their experiences, challenges, and suggestions.
- **Usability Testing Sessions:** Observe users as they interact with the platform to identify pain points and areas for enhancement.
- **Feedback Collection:** Implement mechanisms for users to provide feedback directly on the platform, such as feedback forms or rating systems.

7. Task Efficiency

Objective: Measure how efficiently users can complete tasks on the platform.

Tests:

- **Time on Task:** Measure the time it takes for users to complete specific tasks, such as creating a profile or posting a project.
- **Number of Steps:** Count the number of steps required to complete a task, aiming to minimize unnecessary steps.
- **Error Rate:** Track the number of errors users encounter while performing tasks and the time taken to recover from these errors.

8. Onboarding Experience

Objective: Ensure that new users can quickly and easily get started on the platform.

Tests:

- **Sign-Up Process:** Verify that the sign-up process is straightforward and quick.
- **Welcome Tour:** Ensure that any welcome tours or introductory guides are helpful and not overwhelming.
- **First-Time Use:** Observe new users as they navigate the platform for the first time to identify any initial confusion or difficulties.

Conclusion

User Experience Testing is essential to ensure that the freelancing platform provides a positive and efficient experience for both clients and freelancers. By focusing on usability, accessibility, design consistency, performance, mobile experience, user feedback, task efficiency, and onboarding, we can create a platform that meets the needs of its users and provides a seamless and satisfying experience.

6.3 How ServiceSphere Meets Evaluation Criterion

In this section, we will explore how ServiceSphere, a hypothetical or real software/system, satisfies both functional and non-functional evaluation criteria. This examination will encompass various findings that highlight the system's capabilities, performance, and overall quality.

6.3.1 Functional Requirements Testing Findings

- **User Registration and Authentication:** ServiceSphere successfully handles user registration and authentication with the use of Formik in React and JWT for secure authentication. Formik ensures real-time error handling and validation to prevent the entry of dummy data, while JWT provides a secure method for verifying user identity and maintaining secure sessions.

- **Service and Project Posting:** Allows clients to post services and projects with detailed descriptions, budgets, and timeframes. The system accurately captures and displays these details.

- **AI-Powered Matching and Team Assembly:** Effectively matches clients with suitable freelancers and assembles project teams based on predefined criteria.

- **Communication and Collaboration Tools:** Offers seamless integration with WhatsApp, enabling users to directly redirect to a chat with the freelancer using the entered phone number.

- **Rating and Review System:** Facilitates reliable feedback by allowing clients to review freelancers.

- **Profile Creation:** The profile creation feature for freelancers was thoroughly tested and found to be efficient and user-friendly, allowing freelancers to create and manage their profiles with ease.

- **Real Time Notifications:** Testing revealed that ServiceSphere's real-time notifications are highly effective, providing instant alerts for proposal submissions by freelancers, proposal acceptance by clients, and contract initiations. The integration of SignalR ensures that users receive these notifications without needing to refresh their browser or app, creating a seamless experience that keeps users engaged and informed about relevant activities and updates related to their projects.

- **Text Refinement:** Testing showed that this enhancement significantly increases the appeal of postings, attracting suitable freelancers and clients more effectively.

- **Project Milestones Generation:** The AI-driven project milestone generation feature was thoroughly tested and found to be highly beneficial. By automatically generating project milestones based on requirements and objectives, this feature

aids in project planning by breaking tasks into manageable stages, facilitating smoother project management.

6.3.2 Non-Functional Requirements Testing Findings

- Performance: Testing confirmed that .NET's powerful backend performance capabilities ensure quick response times and efficient handling of peak loads, particularly in search and matching functions. React's efficient rendering and update mechanisms maintain a high-performance frontend, providing a smooth user experience even during high-traffic periods.

- **Scalability:** Tests showed that the .NET platform supports scalability, allowing ServiceSphere to manage an increasing number of users and concurrent operations effectively. Flutter's ability to run on multiple platforms with a single codebase simplifies scaling mobile applications, ensuring consistent performance across devices.

- **Security:** Testing validated that .NET provides advanced security measures, including JWT authentication, crucial for protecting sensitive user data and managing secure user sessions. The integration of JWT with React enhances front-end security, ensuring secure API calls and compliance with data protection regulations.

- **Reliability:** ServiceSphere's system reliability was confirmed through .NET's stable environment and extensive library support, which ensures minimal downtime. Fail-safe mechanisms supported by .NET's error handling and logging capabilities help in quickly diagnosing and recovering from system failures.

- **Maintainability:** The use of modular code in .NET and React supports maintainability, allowing efficient system updates without service disruption. Testing showed that .NET design patterns like Repository, Unit of Work, Dependency Injection, and the Specification Pattern contribute to a modular, scalable, and easy-to-maintain platform.

- **Accessibility:** React and Flutter's adherence to accessibility standards were validated, ensuring the platform is usable for people with disabilities. React supports WCAG 2.1 guidelines for web applications, while Flutter provides tools to create accessible mobile apps, meeting accessibility standards across all platforms.

6.4 Limitations and Potential Enhancements for ServiceSphere

While ServiceSphere excels in many areas, there are inherent limitations that could be addressed through targeted enhancements. These improvements could significantly elevate the platform's effectiveness and user satisfaction. Here's an overview of key limitations along with potential advancements:

1. Integration and Customization Flexibility:

- Potential Enhancement: Expanding customization options and settings could enable users to tailor the platform more closely to their specific needs. Advanced configuration tools for interface and workflow customization would enhance user satisfaction and adaptability.

2. AI-Driven Features:

- Limitation: Current AI capabilities, while advanced, may sometimes lack precision in freelancer matching and text refinement, potentially leading to suboptimal suggestions.

- Potential Enhancement: Implementing more sophisticated AI algorithms that learn from user interactions and feedback could improve the accuracy and relevance of matching and text refinement features. Continuous AI training with updated data sets will also help in adapting to evolving user needs and preferences.

3. Security and Data Protection:

- Potential Enhancement: Strengthening security measures through advanced encryption methods, regular security audits, and the integration of multi-factor authentication could significantly reduce vulnerabilities. A layered security approach would protect against evolving threats and ensure robust data integrity.

4. Scalability and Performance:

- Potential Enhancement: Enhancing the infrastructure to better support scalability can help accommodate growing user numbers and data loads without compromising performance. Implementing dynamic resource allocation and more efficient data handling mechanisms can ensure the platform remains responsive and stable under varied loads.

Conclusion

Addressing these limitations through innovative enhancements will not only improve ServiceSphere's functionality but also enhance user engagement and trust. By continually updating the platform with the latest technological advancements and user feedback, ServiceSphere can maintain its competitive edge in the dynamic freelancing marketplace.

Overall Evaluation

ServiceSphere successfully meets each evaluation criterion, providing a functional, user-friendly, performant, secure, and accurate platform for freelance and project collaboration. Continuous improvements and user feedback will ensure it remains responsive to user needs and maintains its high standards of performance and security.

Future Enhancements

Ongoing monitoring and user feedback will guide future enhancements. Potential improvements include further refinement of AI algorithms, additional security measures, and expanded features to continually enhance user satisfaction and platform functionality.

Chapter 7
Conclusions and Future Work

7.1 Conclusion

The freelancing platform project documentation outlines the comprehensive process of designing and implementing a robust and user-friendly system tailored to the needs of clients and freelancers. By following a structured approach, the project successfully addresses the core requirements and provides an efficient, secure, and scalable solution for freelancing activities.

Key Highlights:

1. **Design and Implementation:**
    - User Stories: Detailed user stories for clients, freelancers, and platform administrators guided the development process, ensuring all user needs were addressed.
    - Wireframe Design: Wireframes provided a clear blueprint for the user interface, facilitating the creation of an intuitive and visually appealing platform.
    - Basic Version and Improved Features: The iterative development approach allowed for the initial release of a functional platform, followed by enhancements based on user feedback and evolving requirements.

2. **Core Functionalities:**
    - User Registration and Profile Creation: Simplified processes for users to join the platform and showcase their skills or post projects.
    - Project Posting and Proposal Submission: Streamlined features for clients to post projects and freelancers to submit proposals, ensuring a seamless matching process.
    - Messaging System: An integrated messaging system enabling clear and efficient communication between clients and freelancers.
    - Review and Rating System: A transparent review system to maintain quality and build trust within the community.

3. **Enhanced Features:**
    - AI-Based Matching: Advanced AI tools for refining project posts and suggesting suitable freelancers, increasing the likelihood of successful collaborations.
    - Notification and Payment Systems: Real-time notifications and secure payment systems ensuring timely updates and transactions.
    - Mobile Compatibility and Analytics: Responsive design for various devices and detailed analytics for better project and performance management.
    - Security Measures: Implementation of two-factor authentication and other security features to protect user data and transactions.

4. **Supporting Features:**
    - Contracts: Formalized agreements between clients and freelancers, ensuring clarity and protection for both parties.
    - Text Refinements: AI-based tools for improving the quality of project descriptions and proposals, enhancing professionalism.
    - Customer Support: Comprehensive support features, including help centers and direct support channels, to assist users in resolving issues.

Final Thoughts:

- The freelancing platform stands out as a versatile and powerful tool, addressing the complexities of the freelancing ecosystem. By prioritizing user experience, security, and continuous improvement, the platform not only facilitates successful project collaborations but also fosters a thriving community of professionals.
- The detailed documentation serves as a testament to the thorough planning, design, and implementation efforts that went into creating the platform. It provides a clear roadmap for future enhancements and potential scalability, ensuring the platform remains relevant and valuable in the ever-evolving freelancing landscape.

- Through this project, we have demonstrated the importance of a user-centric approach, the integration of advanced technologies, and the commitment to quality and security, all of which contribute to the platform's success and sustainability.

7.2 Future Work

The freelancing platform, while comprehensive and robust in its current state, has significant potential for further enhancements and expansions. Future work will focus on leveraging emerging technologies, expanding functionalities, and improving user experience to maintain the platform's competitiveness and relevance in the dynamic freelancing market.

1. Enhanced AI Capabilities

AI-Driven Insights:

- **Predictive Analytics:** Implement predictive analytics to forecast project success, freelancer performance, and client satisfaction.
- **AI Mentorship:** Develop AI-based mentorship tools to guide freelancers in skill development and career growth based on their performance and market trends.

Natural Language Processing (NLP):

- **Advanced Text Refinements:** Enhance text refinement tools using advanced NLP to offer more nuanced and context-aware suggestions.
- **Chatbots:** Introduce intelligent chatbots for instant support, capable of understanding and responding to user queries in a conversational manner.

2. Advanced Collaboration Tools

Real-Time Collaboration:

- **Integrated Tools:** Develop integrated project management tools for real-time collaboration, including document sharing, version control, and task management.
- **Video Conferencing:** Add in-platform video conferencing capabilities to facilitate more effective communication and collaboration between clients and freelancers.

Virtual Workspaces:

- **Collaborative Environments:** Create virtual workspaces where teams can collaborate on projects, share updates, and manage tasks collectively.

3. Expanded Payment Options

Cryptocurrency Payments:

- **Blockchain Integration:** Explore the integration of blockchain technology to offer cryptocurrency payment options, ensuring faster and more secure transactions.
- **Smart Contracts:** Implement smart contracts for automatic and transparent contract execution and payment release.

4. Enhanced Security Measures

Data Protection:

- **Advanced Encryption:** Upgrade encryption protocols to protect sensitive user data and ensure compliance with the latest security standards.
- **Anomaly Detection:** Introduce AI-driven anomaly detection systems to identify and prevent fraudulent activities and security breaches.

5. Global Expansion

Localized Platforms:

- **Language Support:** Expand language support to cater to a global user base, including multilingual interfaces and localized customer support.
- **Regional Compliance:** Ensure compliance with regional regulations and standards to facilitate smooth operations in different countries.

Market-Specific Features:

- **Cultural Adaptation:** Adapt features to meet the cultural and market-specific needs of different regions, enhancing user engagement and satisfaction.

6. User Experience Enhancements

Personalization:

- **Customized Dashboards:** Develop customizable dashboards that allow users to personalize their interface based on their preferences and needs.
- **Recommendation Engines:** Enhance recommendation engines to provide more personalized project and freelancer suggestions based on user behavior and preferences.

Gamification:

- **Achievement Badges:** Introduce gamification elements such as achievement badges and leaderboards to motivate users and increase engagement.
- **Skill Challenges:** Implement skill challenges and contests to encourage freelancers to showcase their abilities and gain recognition.

7. Community Building

Networking Features:

- **Professional Networks:** Create features that enable users to build professional networks, share insights, and collaborate on industry-specific projects.
- **Community Forums:** Develop community forums and discussion boards where users can share experiences, seek advice, and provide feedback.

8. Continuous Improvement and Feedback

User Feedback Loop:

- **Regular Surveys:** Conduct regular user surveys to gather feedback on platform features, usability, and overall satisfaction.
- **Beta Testing:** Involve users in beta testing of new features to gather insights and make necessary adjustments before full deployment.

Iterative Development:

- **Agile Methodology:** Continue to employ agile development practices, allowing for iterative improvements and quick adaptation to changing user needs and market conditions.

References

[1] Norman, D. A., & Draper, S. W. (Eds.). (1986). User centered system design: New perspectives on human-computer interaction. CRC Press.

[2] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Thomas, D. (2001). Manifesto for agile software development. Agile Alliance. Retrieved from https://agilemanifesto.org/

[3] Banks, A., & Porcello, E. (2017). Learning React: Functional Web Development with React and Redux. O'Reilly Media, Inc.

[4] Galloway, J., Wilson, P., Allen, K., & Matson, D. (2014). Professional ASP.NET MVC 5. John Wiley & Sons.

[5] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

[6] Rubin, K. S. (2012). Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley.

[7] Pressman, R. S., & Maxim, B. R. (2020). "Software Engineering: A Practitioner's Approach." McGraw-Hill Education.

[8] Wiegers, K. E., & Beatty, J. (2013). "Software Requirements." Pearson Education.

[9] Martin, R. C. (2008). "Clean Code: A Handbook of Agile Software Craftsmanship." Prentice Hall.

[10] Fowler, M. (2002). "Patterns of Enterprise Application Architecture." Addison-Wesley Professional.

[11] Davis, A. M. (1993). "Software Requirements: Objects, Functions, and States." Prentice Hall.

[12] Upwork. (2021). Project Catalog. Retrieved from https://www.upwork.com/project-catalog

[13] Upwork. (2021). *How to Post a Job*. Retrieved from https://www.upwork.com/hiring/start/how-to-post-a-job-on-upwork/

[14] Freelancer. (2021). *Find Freelancers*. Retrieved from https://www.freelancer.com/find-freelancers

[15] Freelancer. (2021). *Advanced Search Options*. Retrieved from https://www.freelancer.com/search

[16] Fiverr. (2021). *Gig Matching Algorithm*. Retrieved from https://www.fiverr.com/

[17] Fiverr. (2021). *Fiverr Business Tools*. Retrieved from https://business.fiverr.com/

[18] LinkedIn. (2021). *Creating a Profile*. Retrieved from https://www.linkedin.com/help/linkedin/answer/1121/creating-your-profile?lang=en

[19] LinkedIn. (2021). *Review and Endorsement Features*. Retrieved from https://www.linkedin.com/help/linkedin/answer/35359?lang=en