NearbyNexus
A location-based service application

Project Report Submitted by

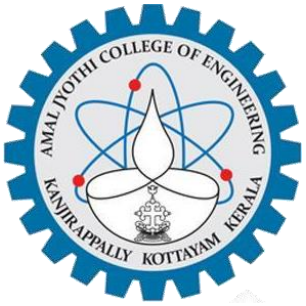
Don Benny

Reg. No.: AJC19MCA-I022

In Partial fulfillment for the Award of the Degree of

INTEGRATED MASTER OF COMPUTER APPLICATIONS (INMCA)

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY



AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC with 'B' grade. Koovappally, Kanjirappally, Kottayam, Kerala – 686518

2023-2024

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report, "**NearbyNexus"** is the bonafide work of **DON BENNY (Regno: AJC19MCA-I022)** in partial fulfillment of the requirements for the award of the Degree of Integrated Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2023-24.

Mr.Binumon Joseph
Internal Guide

Ms. Meera Rose Mathew
Coordinator

Rev. Fr. Dr. Rubin Thottupurathu Jose
Head of the Department

External Examiner

DECLARATION

I hereby declare that the project report **"NearbyNexus"** is a bonafide work done at Amal Jyothi College of Engineering, towards the partial fulfilment of the requirements for the award of the Integrated Master of Computer Applications from APJ Abdul Kalam Technological University, during the academic year 2023-2024.

Date: 03/04/2024 **Don Benny**

KANJIRAPPALLY **Reg: AJC19MCA-I022**

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Manager **Rev. Fr. Dr. Mathew Paikatt** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude towards our Head of the Department **Rev.Fr.Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Ms. Meera Rose Mathew** for her valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Mr. Binumon Joseph** for his inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

DON BENNY

ABSTRACT

In an era defined by connectivity and convenience, the proposed location-based application presents an innovative solution to bridge the gap between users and an extensive array of service providers, ranging from independent entrepreneurs to established corporations. Tailored to cater to the needs of individuals navigating unfamiliar locales, the application prioritizes essential services such as transportation, legal assistance, manual labor, and more. Through seamless integration, the platform incorporates individual service providers, ensuring optimal user convenience and satisfaction.

The application caters to three distinct user categories: Administrators, General Users, and Vendors, each equipped with unique functionalities aimed at enhancing their respective experiences. Administrators oversee platform operations, ensuring secure access through robust authentication processes. Service providers can register their services within the application, empowering users to make informed decisions. Moreover, users can engage in secure, cashless transactions, streamlining the payment process.

Key features include a dynamic search function facilitating easy location of required services, alongside a comprehensive rating and review system enabling user feedback. The application leverage's location data to match users with nearby service providers, facilitating seamless connections. Additionally, users can broadcast service needs, communicate directly with providers, and share their location if necessary.

Furthermore, reporting mechanisms enable users to report issues to administrators, while service providers can provide feedback on users. User referrals facilitate the spread of trusted vendors within user networks, fostering community engagement. Subscription system enables users to subscribe premium features of the application which provides premium services other than normal functions.

Incorporating features such as service request broadcasts, map integration, in-app messaging, location sharing, subscriptions, reporting and many more, the proposed application offers a comprehensive suite of functionalities aimed at delivering a seamless and secure experience for users and service providers alike. Positioned as a transformative force in location-based service applications, the proposed project stands poised to redefine the landscape of user-service provider interactions.

CONTENT

List of Abbreviation

UML	Unified Modeling Language
IDE	Integrated Development Environment
UI	User Interface
SDK	Software Development Kit
BaaS	Backend-as-a-Service
FCM	Firebase Cloud Messaging

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

NearbyNexus revolutionizes user-service provider interactions by seamlessly connecting users with a diverse range of service providers, from independent entrepreneurs to established corporations, catering to essential needs in unfamiliar locales. Administrators, General Users, and Vendors each benefit from unique functionalities ensuring optimal experiences, with administrators overseeing platform operations and service providers registering their offerings. Secure, cashless transactions streamline payments, while dynamic search and robust rating systems enhance user decision-making. Leveraging location data, the app matches users with nearby providers, facilitating seamless connections and enabling service request broadcasts and direct communication. Reporting mechanisms and user referrals foster community engagement and trust. With features like map integration, in-app messaging, location sharing, subscriptions, and reporting, NearbyNexus offers a comprehensive solution, poised to redefine location-based service applications.

1.2 PROJECT SPECIFICATION

NearbyNexus is an innovative platform crafted to streamline access to a diverse array of services, all tailored to the user's specific geographical location. It acts as a vital bridge, connecting individuals in a new town or city with a diverse array of service providers, spanning from individual entrepreneurs to established organizations. This platform revolutionizes the way users seek and interact with services, ensuring a seamless and efficient experience.

- **Admin**

Oversees and regulates the utilization of both users and services within the application.

- View users
- Manage users
- View services
- Add services
- View services created.
- Download user data.
- View revenue matrix
- Manage reports.
- Manage job posts

- **General users**

These are the users who receive the services from the vendors.

- Authentication – Using (Google/Email)
- Search services

- Connect with vendors
- Send service details
- View service providers at current location
- Rate service providers
- Set service emergency level
- Payment
- Post jobs.
- Chat with other users.

- **Vendors**

Offers products and solutions tailored to specific needs and requirements. Maintains a high level of customer service and support throughout the entire process. Continuously improves their offerings through innovation and feedback from customers.

- Create service
- Modify service details
- View local users in the current location
- Set Availability status.
- Go online or offline.
- Apply for jobs.s
- Receive premium benefits.

CHAPTER 2
SYSTEM STUDY

2.1 INTRODUCTION

In an ever-evolving world driven by complex interactions and dependencies, understanding and optimizing systems has become paramount. System study, a multidisciplinary approach, offers a comprehensive framework to dissect, analyze, and improve the functioning of various processes, be it in engineering, business operations, or societal structures. This practice not only unveils hidden inefficiencies but also provides a pathway towards streamlined operations and enhanced outcomes. In this pursuit, effective communication plays a pivotal role. A well-articulated system study report not only encapsulates intricate details but also ensures that insights are accessible and comprehensible to a diverse audience. This is where your penchant for improving writing becomes invaluable. Through precise language, coherent structure, and strategic use of visual aids, you have the power to transform intricate technical details into a narrative that resonates with stakeholders at every level.

In the following sections, we will delve deeper into the principles of system study and explore how your proficiency in writing can be harnessed to unlock its full potential. Together, we will embark on a journey to unravel the intricacies of systems, armed with the powerful tool of effective communication.

2.2 EXISTING SYSTEM

The existing systems predominantly follow organizational service models. However, they often lack direct interaction and may not sufficiently address individualized needs. Additionally, some systems are designed for a singular service and do not support multiple service offerings.

Furthermore, the current systems offer a wide range of services, potentially overwhelming users when it comes to selecting the most suitable provider. The personalized results provided by these systems may not always be easily adaptable to individual user preferences.

2.2.1 NATURAL SYSTEM STUDIED

The current system of manual service search poses challenges of inefficiency and potential risks for users. With the proposed location-based application, users can seamlessly connect with verified service providers, mitigating the risk of encountering unreliable sources. Real-time job information in the user's location optimizes search efforts, reducing unnecessary travel. This transformative solution enhances user convenience and contributes to a safer, more efficient service-seeking experience.

Integrated within the application are features tailored to the needs of Administrators, General Users, and Vendors. Administrators ensure secure access, while service providers register their services, empowering users with informed decisions and facilitating secure transactions. Dynamic search,

rating systems, and location-based matching streamline connections, while communication tools and location sharing enhance user-provider interactions.

Moreover, reporting mechanisms and user referrals foster community engagement and trust. Premium subscriptions offer enhanced features, enriching the user experience. By encompassing a comprehensive suite of functionalities, the proposed application redefines location-based service interactions, positioning itself as a transformative force in the digital landscape.

2.2.2 DESIGNED SYSTEM STUDIED

Google Maps is a widely used mapping service that provides detailed, interactive maps for locations around the world. It offers features like turn-by-turn navigation, real-time traffic updates, and points of interest. Users can find businesses, restaurants, and landmarks, as well as get directions for driving, walking, cycling, and public transportation. Additionally, Google Maps integrates with other Google services, allowing users to explore places in 360-degree Street View imagery. Overall, it's a versatile tool for navigation, exploration, and discovering local businesses and attractions.

2.3 DRAWBACKS OF EXISTING SYSTEM

- Location and Mapping Focus:
- Lack of Individual Service Focus
- Absence of User Job Activity Logs
- No Service History Tracking
- Real-time User Availability Information

2.4 PROPOSED SYSTEM

The proposed system, NearbyNexus, revolutionizes the interaction between users and service providers in unfamiliar locales by seamlessly connecting them through a comprehensive location-based application. It caters to diverse needs, prioritizing essential services like transportation, legal assistance, and manual labor. Administrators ensure secure access, while service providers register their offerings, enabling informed decisions and streamlined transactions.

Users benefit from dynamic search, rating systems, and location-based matching with nearby providers. They can also broadcast service needs, communicate directly, and share locations. Reporting mechanisms and user feedback enhance accountability and trust. Subscriptions offer premium features beyond standard functions, promoting user engagement.

With its inclusive features, NearbyNexus bridges the gap between users and vendors, redefining service access and delivery. Positioned as a transformative solution, it offers a seamless and secure experience, poised to reshape location-based service interactions.

NearbyNexus is an innovative application platform engineered to seamlessly connect users in new towns or cities with an extensive array of service providers, spanning from independent operators to established organizations. Its paramount objective is to streamline the process of accessing services, eliminating the need for users to possess immediate contact information for the providers. To leverage the application's functionalities, users must undergo a straightforward registration process, subsequently granting them access to its diverse features. Similarly, service providers or vendors are required to register and validate their credentials to commence service offerings through the application.

2.5 ADVANTAGES OF PROPOSED SYSTEM

- **Efficient Service History Tracking:**

The system allows for the meticulous tracking and logging of user service activities. This ensures a comprehensive record of interactions, promoting transparency and accountability.

- **Convenient Access to Diverse Services:**

NearbyNexus facilitates easy and convenient access to a wide range of services, catering to individual service providers as well as larger organizations. This eliminates the need for users to have immediate contact information for service providers.

- **Enhanced User and Vendor Verification:**

The registration and verification process for users and vendors ensures a higher level of credibility and authenticity. This contributes to a secure and trustworthy environment for service interactions.

- **Location-Based Service Customization:**

NearbyNexus leverages geographical information to connect users with service providers in their specific location. This tailored approach ensures that users receive services relevant to their current whereabouts, enhancing convenience and relevance.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

This examination plays a crucial role in gauging whether a project is capable of achieving the organization's goals, considering the allocation of resources, workforce, and time. It serves as a guiding compass for developers to explore the project's potential benefits and possibilities in the long term. To ascertain the viability and worthiness of proceeding with a proposed system, a comprehensive feasibility study must be undertaken.

During the feasibility study, the proposed system's impact on the organization is carefully evaluated, along with its ability to meet customer demands and optimize resource utilization. This rigorous evaluation process is a prerequisite before granting approval for the development of a new application. Numerous factors are thoroughly considered in the feasibility study, encompassing technical, financial, and operational aspects. This comprehensive approach ensures that all angles are meticulously assessed and documented in the feasibility study report. The result is an informed decision that charts the path for the project's successful realization while aligning with the organization's strategic objectives.

3.1.1 Economic Feasibility

Conducting an economic feasibility analysis is of utmost importance when assessing the viability of a new project, considering both the cost and time investment involved. This meticulous process involves a comprehensive examination of all relevant factors that can significantly impact the project's success.

By thoroughly evaluating the economic aspects of the project, including potential expenses, returns, and anticipated benefits, the organization gains a clearer understanding of the project's financial feasibility. This empowers decision-makers to move forward with confidence, knowing that the investment aligns with the organization's goals and is likely to yield desirable results.

NearbyNexus can explore various revenue streams, such as charging service providers for premium listings, offering subscription-based services to vendors or users, or earning a commission on transactions made through the application. By diversifying revenue sources, the project can reduce dependency on a single income stream and increase economic resilience.

In the case of the proposed system, NearbyNexus, a rigorous cost-benefit analysis has been conducted. The results of this analysis indicate that the project is not only feasible but also economically viable, remaining within the pre-established budgetary constraints. This positive outcome reaffirms the potential value and profitability of the initiative.

Conducting a thorough market analysis is crucial to determine the economic viability of NearbyNexus. Evaluating the demand for location-based service applications and understanding

user preferences will help gauge the potential user base. Identifying the target market segments, such as tourists, newcomers, or residents seeking local services, will provide valuable insights into revenue generation opportunities. Positive market demand and a sizable target audience increase the project's economic feasibility.

3.1.2 Technical Feasibility

Technical feasibility involves evaluating whether a product or service can be developed and executed using the existing technology and available resources. The analysis assesses the viability of the proposed plan by examining factors such as tools, materials, labour, logistics, and technology. It plays a vital role in identifying and resolving potential project challenges before commencing the work. Creating a flowchart to visualize the product or service's evolution can aid in understanding the system's process during the technical feasibility assessment.

Enabling service providers to create and manage their services within the application demands a well-structured database architecture. An efficient backend system is necessary to store and retrieve service details accurately.

Accurate geolocation and mapping functionalities provide NearbyNexus to identify the user's location and nearby service providers. Integration with mapping APIs, such as Google Maps, is crucial for displaying service providers on a map. Developing an in-app messaging feature requires real-time communication capabilities. In NearbyNexus implementing a messaging system using Web Socket or similar technologies will enable direct communication between users and service providers.

3.1.3 Behavioral Feasibility

The behavioral feasibility of NearbyNexus focuses on the acceptance and adoption of the application by its target users, both service providers and service seekers. Understanding the behavioral aspects of the project is crucial to ensure that the application aligns with user preferences, addresses their needs, and encourages active engagement. Behavioral feasibility necessitates establishing trust and safety measures to ensure that users feel secure while using NearbyNexus. Implementing user verification processes, user reviews, and a reporting system for inappropriate behavior or service quality concerns will build trust and confidence in the application. The behavioral feasibility assessment assures that NearbyNexus is designed with a user-centric approach, meeting the preferences and requirements of its target audience. By prioritizing user experience, safety, engagement, and effective communication, the application is more likely to be embraced and used by users and service providers alike. Ensuring behavioral feasibility will lead

to higher user adoption rates, increased user retention, and long-term success for NearbyNexus in the competitive market.

3.1.4 Feasibility Study Questionnaire

1. Project Overview?

NearbyNexus is a location-based application platform designed to connect users in new towns or cities with a diverse array of service providers. These providers range from individual entrepreneurs to larger organizations, offering a wide range of services. The primary aim is to streamline access to services, eliminating the need for users to have immediate access to provider contact information. To access the application and its functionalities, users must register their accounts through a straightforward registration process. Similarly, service providers or vendors need to register and verify their credentials to begin offering services through the application. The application administrator has access to user details and oversees user activities. Based on user activity and reports, the administrator takes necessary actions with regard to users or vendors. Once logged in successfully, users are presented with a comprehensive list of available services based on their geographical location. Through this application, both users and vendors can efficiently access services and achieve their objectives.

2. To what extent is the system proposed for?

The primary goal of the application is to provide users with easy access to a wide range of local services. By utilizing the user's geographical location, the app connects them with nearby service providers, including individuals and organizations. The app aims to enhance convenience for users by eliminating the need for providers' contact information ahead of time. Users can browse various services, view details, and request services through the app, making it quick and easy to access services. The system is designed to cover diverse service categories, ensuring users can find and use services ranging from transportation (taxis) to professional services (lawyers), household help (plumbers, electricians), and personal services (beauticians, tailors). Additionally, the app allows general users to publish job-related posts to a public feed. These can be seen and bid on by job seekers. The app also enables direct messaging and live location sharing between users. The new features allow users to report, share job posts, etc.

3. Specify the Viewers/Public which is to be involved in System?

- General users(public)
- Service providers (plumbers, electricians, beauticians, tailors, etc.,)

4. List the Modules included in your System?

User management, Vendor management, Service management, Payment integration, Location-based module, Communication module, Service broadcast module, Vendor availability calendar module, Maps, Messaging, etc.

5. Identify the users in your project?

Admin, General users (Public), Service providers.

6. Who owns the system?

Administrator

7. System is related to which firm/industry/organization?

IT – Industry, On-Demand Services Platforms like Uber, Ola

8. Details of person that you have contacted for data collection.

Online resources

Athul Vinayakumar - IT professional

9. Questionnaire to collect details about the project?

a) How does NearbyNexus facilitate communication between users and vendors?

In-app messaging enables direct communication to discuss service details. Users can also share locations with vendors if needed.

b) What benefits does the subscription feature provide?

Subscriptions allow users to avail premium features of the application. Some features are limited to the users. By the premium subscription unlocks the tier.

c) Which payment gateway(s) are integrated into the application for online transactions?

We often use Stripe for integrating payments and transactions throughout the application which enables secure and successful transactions.

d) **What safety/moderation features does NearbyNexus have?**

Reporting features allow inappropriate users to be flagged. Vendors can also give feedback on users.

e) How do you handle and display reviews and ratings on the platform?

The reviews and rating on this platform are designed for both the general users and also for vendor's side. The users can rate the vendor based on his service. The rating is scaled in a scale of 10. Also enables the vendors to rate and provide feedback about the users to whom they provided a service. which describes the attitude and payment responsibility of the user is reviewed.

f) What tasks and responsibilities does the administrator have in managing the platform?

The administrator or moderator of the application have full access control over the application. The admin can manage the users of the application, view and review the services that are created by the vendors.

g) How does the administrator handle user complaints, reports, and vendor verifications?

The administrator has the full access to the user ratings and the admin receives the complaints and reports that are submitted by the users of the application. Based on the report and the severity of the report application the admin reviews the report and take necessary actions

h) What key features does NearbyNexus provide?

Key features include service request broadcasts, map integration, in-app messaging, location sharing, subscriptions, reporting, and referrals.

i) Which geographical locations or regions does the application cover? Is it limited to specific cities or available nationwide/internationally?

Currently the application supports the location that are geographically near to the user which covers a specified radius from the user.

j) What happens when a user broadcasts a service request?

Interested nearby service providers can directly contact the user through in-app messaging to offer their services.

3.2 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor	Snapdragon 700 or related
RAM	4 Gigabytes
Hard disk	30 - 60 Gigabytes

3.2.2 Software Specification

Front End	Flutter, Dart, HTML, React JS
Back End	Firebase
Database	Firebase Database
Client on PC	Windows 7 and above
Technologies used	Dart, Flutter, Rapid API, Firestore & Firebase Storage, Google Maps, Machine Learning, Firebase Messaging, Push Notifications, React JS

3.3 SOFTWARE DESCRIPTION

3.3.1 Flutter

In May 2017, Google unveiled Flutter, a revolutionary free and open-source mobile UI framework. In essence, Flutter empowers developers to craft native mobile applications using a unified code-base. This breakthrough means that you can develop distinct applications for both iOS and Android platforms, all within the same programming language and code-base.

Flutter consists of two important parts:

- **SDK (Software Development Kit):** This toolkit equips you with a set of tools to create applications. It also provides the means to translate your code into native machine code, tailored for both iOS and Android platforms.
- **Framework (UI Library with Widgets):** This encompasses a versatile collection of reusable user interface (UI) elements that you can customize to align with your specific

requirements. These elements encompass buttons, text input fields, sliders, and various other interactive objects.

Applications developed with Flutter utilize the Dart programming language. Although Google introduced Dart in October 2011, it has undergone significant advancements since its inception. Dart serves as a front-end coding framework that empowers the creation of programs for both web and mobile platforms.

3.3.2 Firebase

Firebase is a Back-end-as-a-Service (BaaS) platform that equips developers with a diverse set of tools and services to create top-notch applications, expand their user community, and generate revenue. Built on Google's technology platform, Firebase employs a No-SQL database system where data is stored in documents that closely resemble JSON structures.

Key features of Firebase:

- **Cloud Firestore**: This is a flexible, scalable No-SQL cloud database that allows for seamless data storage and synchronization in real-time.
- **Cloud Messaging:** Firebase Cloud Messaging (FCM) enables reliable and efficient delivery of messages to target devices, including iOS, Android, and web.
- **Crashlytics**: Firebase Crashlytics provides detailed crash reports, allowing developers to quickly identify and fix issues in their apps.
- **Performance Monitoring:** This feature helps developers gain insights into the performance of their app, allowing them to optimize it for better user experiences.
- **Remote Configure:** Firebase Remote Configure allows you to customize your app without publishing an app update. It's useful for A/B testing and making dynamic adjustments.
- **Dynamic Links:** These are deep links that can lead users to specific content or pages within your app, improving user engagement.
- **AdMob Integration:** Firebase integrates seamlessly with Google's mobile advertising platform, allowing developers to monetize their apps effectively.
- **App Indexing**: Firebase helps improve app discoverability by allowing Google Search to index the content within your app.
- **Machine Learning Kit**: Firebase includes machine learning capabilities, making it easy to implement features like image labeling and text recognition in your app.
- **App Distribution:** This feature allows for the secure distribution of pre-release versions of your app to trusted testers.

These features collectively make Firebase a comprehensive and powerful platform for app development, offering a wide range of tools to enhance user experience, app performance, and developer productivity.

CHAPTER 4
SYSTEM DESIGN

4.1 INTRODUCTION

System design is a critical phase in the process of application development, playing a pivotal role in shaping the architecture and functionality of software. It encompasses the creation of a structured blueprint that outlines how various components, modules, and services within the application will interact and collaborate to meet specific objectives. This process involves making crucial decisions regarding database architecture, server configuration, and overall system architecture to ensure scalability, efficiency, and robustness. Additionally, system design takes into account factors such as user experience, security, and data management to create a well-rounded and effective application. It requires a deep understanding of the application's requirements, as well as proficiency in various technologies and programming paradigms to construct a robust foundation for the development process.

In essence, system design serves as the architectural backbone of any software project. It involves breaking down the complex functionalities and requirements of the application into manageable components, each with a defined purpose and relationship with other elements. Through meticulous planning and consideration of various technical and user-centric aspects, system design lays the groundwork for developers to implement code efficiently and cohesively. A well-crafted system design not only ensures that the application meets performance and scalability requirements but also provides a framework for future enhancements and maintenance. It is a critical step that bridges the gap between conceptualizing an application and transforming it into a functional, reliable, and user-friendly software solution.

4.2UML DIAGRAM

Unified Modeling Language (UML) stands as a foundational tool in software engineering, renowned for its role in visually representing complex systems and processes. It provides a standardized set of graphical notations that facilitate the clear depiction of various aspects of a system's structure and behavior. Originating from the collaboration of industry experts, UML has gained widespread acceptance and adoption in both academia and industry. It serves as a powerful communication tool, enabling stakeholders, including developers, designers, and clients, to attain a shared understanding of system architecture, design, and functionality. UML diagrams act as a lingua franca, transcending language barriers and ensuring a consistent means of conveying intricate software concepts, ultimately enhancing the efficiency and effectiveness of the software development process.

Types of UML diagrams

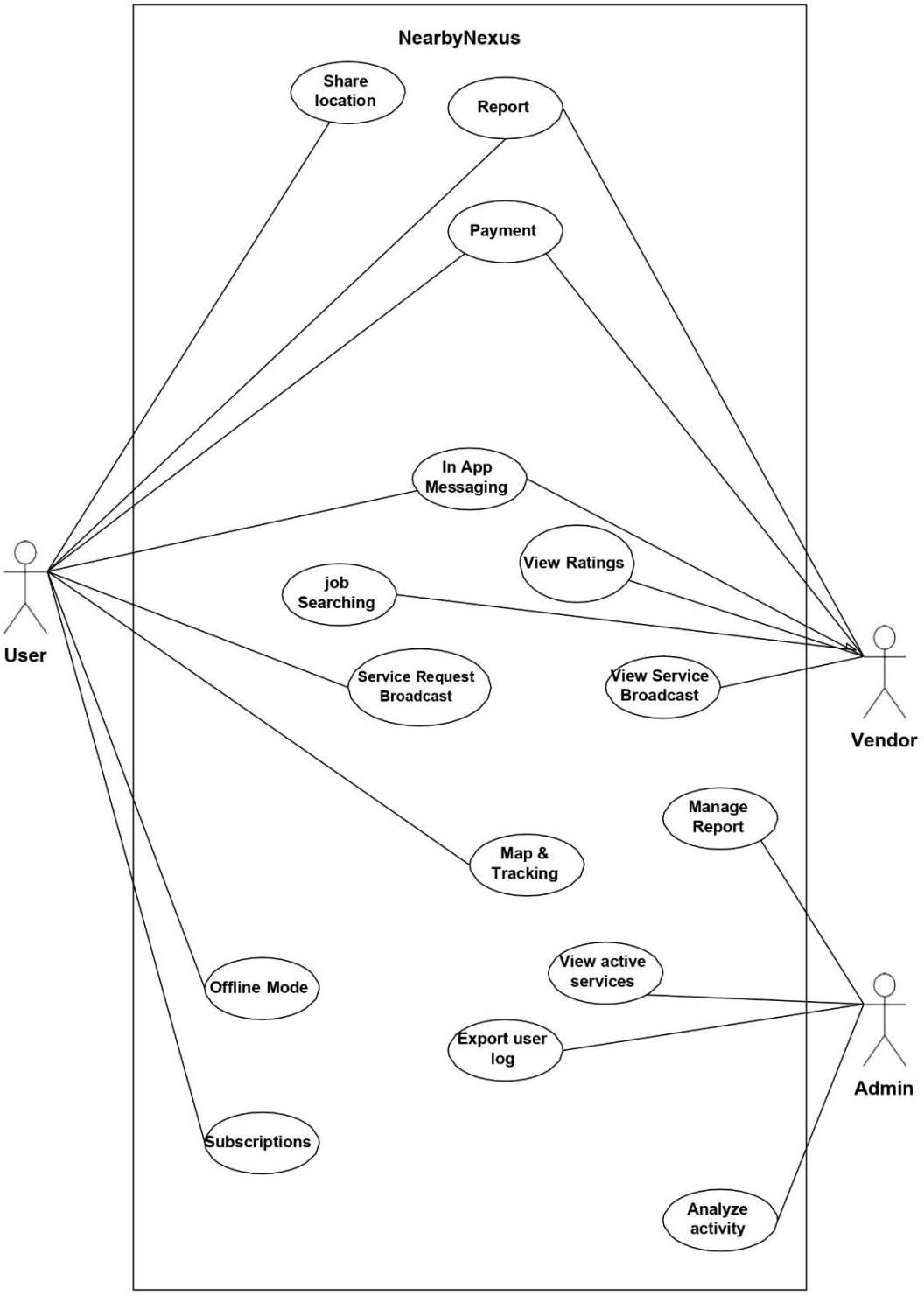
- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Activity diagram
- State chart diagram
- Deployment diagram
- Component diagram

4.2.1 USE CASE DIAGRAM

Use Case Diagrams, a cornerstone in software engineering, serve as a visual representation of the interactions between a system and its external entities. At their core, they provide a structured means of identifying and defining the various functionalities a system offers and how these functionalities are accessed by different actors or entities. Actors, representing users, systems, or external entities, are depicted along with the specific use cases they engage with. Associations between actors and use cases elucidate the nature of these interactions, clarifying the roles and responsibilities of each entity within the system. This detailed visual representation not only enhances communication among stakeholders but also provides a clear blueprint for system functionality, laying the foundation for the subsequent stages of the software development process. Overall, Use Case Diagrams play a pivotal role in aligning development efforts with user expectations, ensuring that the resulting software system fulfills its intended purpose effectively and efficiently.

- **Actor Definition:** Clearly define and label all actors involved in the system. Actors represent external entities interacting with the system.
- **Use Case Naming:** Use descriptive names for use cases to accurately convey the functionality they represent.
- **Association Lines:** Use solid lines to represent associations between actors and use cases. This signifies the interaction between entities.
- **System Boundary:** Draw a box around the system to indicate its scope and boundaries. This defines what is inside the system and what is outside.

- **Include and Extend Relationships:** Use "include" relationships to represent common functionalities shared among multiple use cases. Use "extend" relationships to show optional or extended functionality.

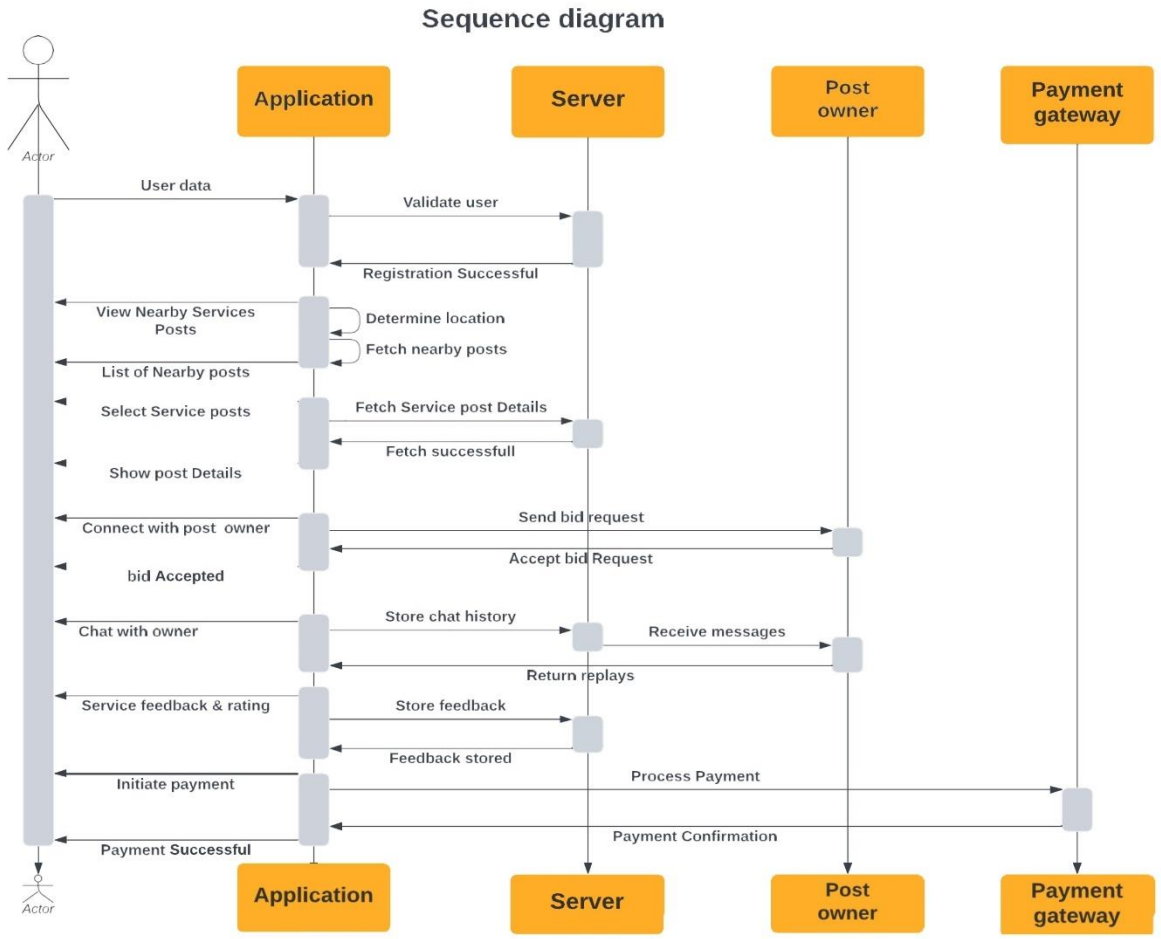


4.2.1.1 Use case diagram

4.2.2 SEQUENCE DIAGRAM

Sequence Diagrams stand as dynamic models in software engineering, portraying the chronological flow of interactions between various objects or components within a system. They spotlight the order in which messages are exchanged, revealing the behavior of the system over time. Actors and objects are represented along a vertical axis, with arrows indicating the sequence of messages and their direction. Lifelines, extending vertically from actors or objects, illustrate their existence over the duration of the interaction. These diagrams serve as a vital tool for visualizing system behavior and understanding the temporal aspects of a software process. Through Sequence Diagrams, stakeholders gain valuable insights into how different elements collaborate to achieve specific functionalities, facilitating more effective communication among development teams and stakeholders alike. This detailed representation not only aids in detecting potential bottlenecks or inefficiencies but also provides a foundation for refining system performance in the later stages of software development.

- **Vertical Ordering:** Represent actors and objects along a vertical axis, indicating the order of interactions from top to bottom.
- **Lifelines:** Extend vertical lines from actors or objects to denote their existence and participation in the interaction.
- **Activation Bars:** Use horizontal bars along lifelines to show the period during which an object is active and processing a message.
- **Messages and Arrows:** Use arrows to indicate the flow of messages between objects, specifying the direction of communication.
- **Self-Invocation:** Use a looped arrow to represent self-invocation, when an object sends a message to itself.
- **Return Messages:** Indicate return messages with a dashed arrow, showing the response from the recipient.
- **Focus on Interaction:** Sequence Diagrams focus on the chronological order of interactions, avoiding implementation details.
- **Concise Notation:** Use clear and concise notation to represent messages and interactions, avoiding unnecessary complexity.
- **Consider System Boundaries:** Clearly define the boundaries of the system to indicate what is included in the interaction.
- **Feedback and Validation**: Seek feedback from stakeholders and team members to ensure the diagram accurately represents the system behavior



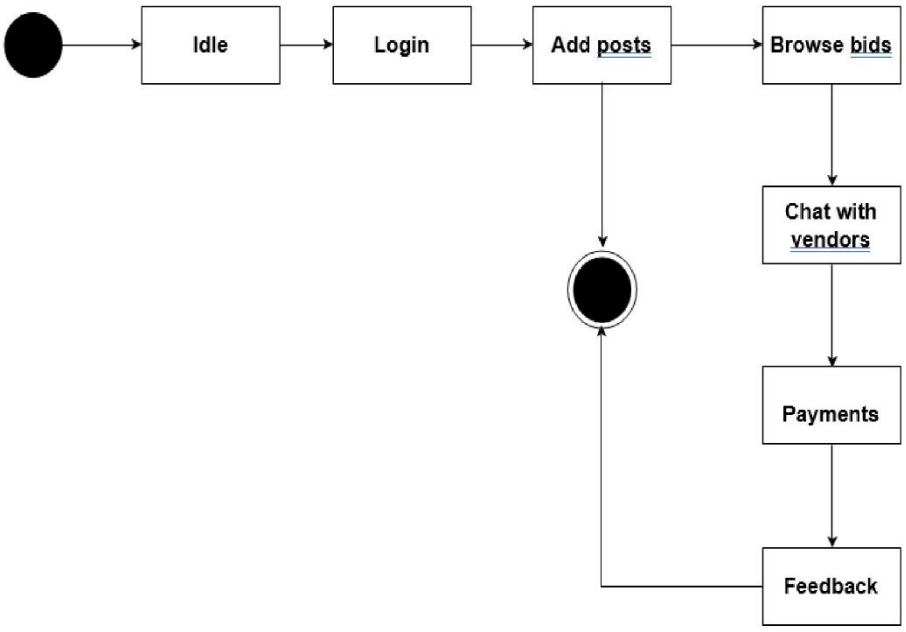
4.2.2.1 Sequence diagram

4.2.3 STATE CHART DIAGRAM

A State Chart Diagram, a fundamental component of UML, provides a visual representation of an object's lifecycle states and the transitions between them. It depicts the dynamic behavior of an entity in response to events, showcasing how it transitions from one state to another. Each state represents a distinct phase in the object's existence, while transitions illustrate the conditions triggering state changes. Initial and final states mark the commencement and termination of the object's lifecycle. Orthogonal regions allow for concurrent states, capturing multiple aspects of the object's behavior simultaneously. Hierarchical states enable the representation of complex behaviors in a structured manner. Entry and exit actions depict activities occurring upon entering or leaving a state. Moreover, guard conditions ensure that transitions occur only under specified circumstances. State Chart Diagrams play a crucial role in understanding and designing the dynamic behavior of systems, aiding in the development of robust and responsive software applications.

Key notations for State Chart Diagrams:

- **Initial State:** Represented by a filled circle, it signifies the starting point of the object's lifecycle.
- **State:** Depicted by rounded rectangles, states represent distinct phases in an object's existence.
- **Transition Arrow:** Arrows denote transitions between states, indicating the conditions triggering a change.
- **Event:** Events, triggers for state changes, are labeled on transition arrows.
- **Guard Condition:** Shown in square brackets, guard conditions specify criteria for a transition to occur.
- **Final State:** Represented by a circle within a larger circle, it indicates the end of the object's lifecycle.
- **Concurrent State:** Represented by parallel lines within a state, it signifies concurrent behaviors.
- **Hierarchy:** States can be nested within other states to represent complex behavior.
- **Entry and Exit Actions**: Actions occurring upon entering or leaving a state are labeled within the state.
- **Transition Labels:** Labels on transition arrows may indicate actions or operations that accompany the transition.



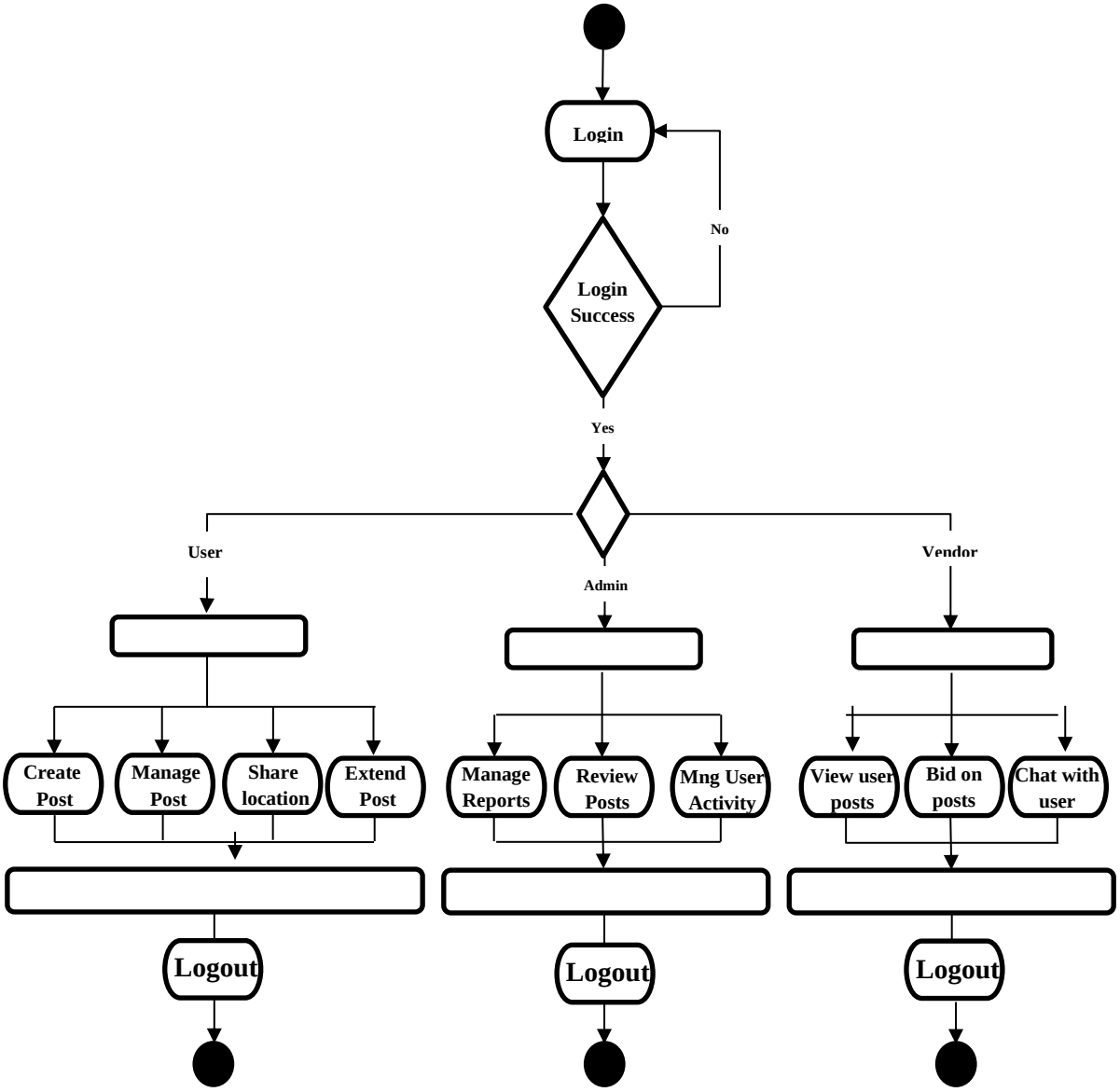
4.2.3.1 State chart diagram

4.2.4 ACTIVITY DIAGRAM

An Activity Diagram is a visual representation within UML that illustrates the flow of activities and actions in a system or process. It employs various symbols to depict tasks, decision points, concurrency, and control flows. Rectangles signify activities or tasks, while diamonds represent decision points, allowing for conditional branching. Arrows indicate the flow of control from one activity to another. Forks and joins denote concurrency, where multiple activities can occur simultaneously or in parallel. Swimlane segregate activities based on the responsible entity, facilitating clarity in complex processes. Initial and final nodes mark the commencement and completion points of the activity. Decision nodes use guards to determine the path taken based on conditions. Synchronization bars enable the coordination of parallel activities. Control flows direct the sequence of actions, while object flows depict the flow of objects between activities. Activity Diagrams serve as invaluable tools for understanding, modeling, and analyzing complex workflows in systems and processes. They offer a structured visual representation that aids in effective communication and system development.

Key notations for Activity Diagrams:

- **Initial Node:** Represented by a solid circle, it signifies the starting point of the activity.
- **Activity:** Shown as a rounded rectangle, it represents a task or action within the process.
- **Decision Node:** Depicted as a diamond shape, it indicates a point where the process flow can diverge based on a condition.
- **Merge Node:** Represented by a hollow diamond, it signifies a point where multiple flows converge.
- **Fork Node:** Shown as a horizontal bar, it denotes the start of concurrent activities.
- **Join Node:** Depicted as a vertical bar, it marks the point where parallel flows rejoin.
- **Final Node**: Represented by a solid circle with a border, it indicates the end of the activity.
- **Control Flow:** Arrows connecting activities, showing the sequence of actions.
- **Object Flow:** Lines with arrows representing the flow of objects between activities.
- **Swimlane:** A visual container that groups activities based on the responsible entity or system component.
- **Partition**: A horizontal or vertical area within a swimlane, further organizing activities.



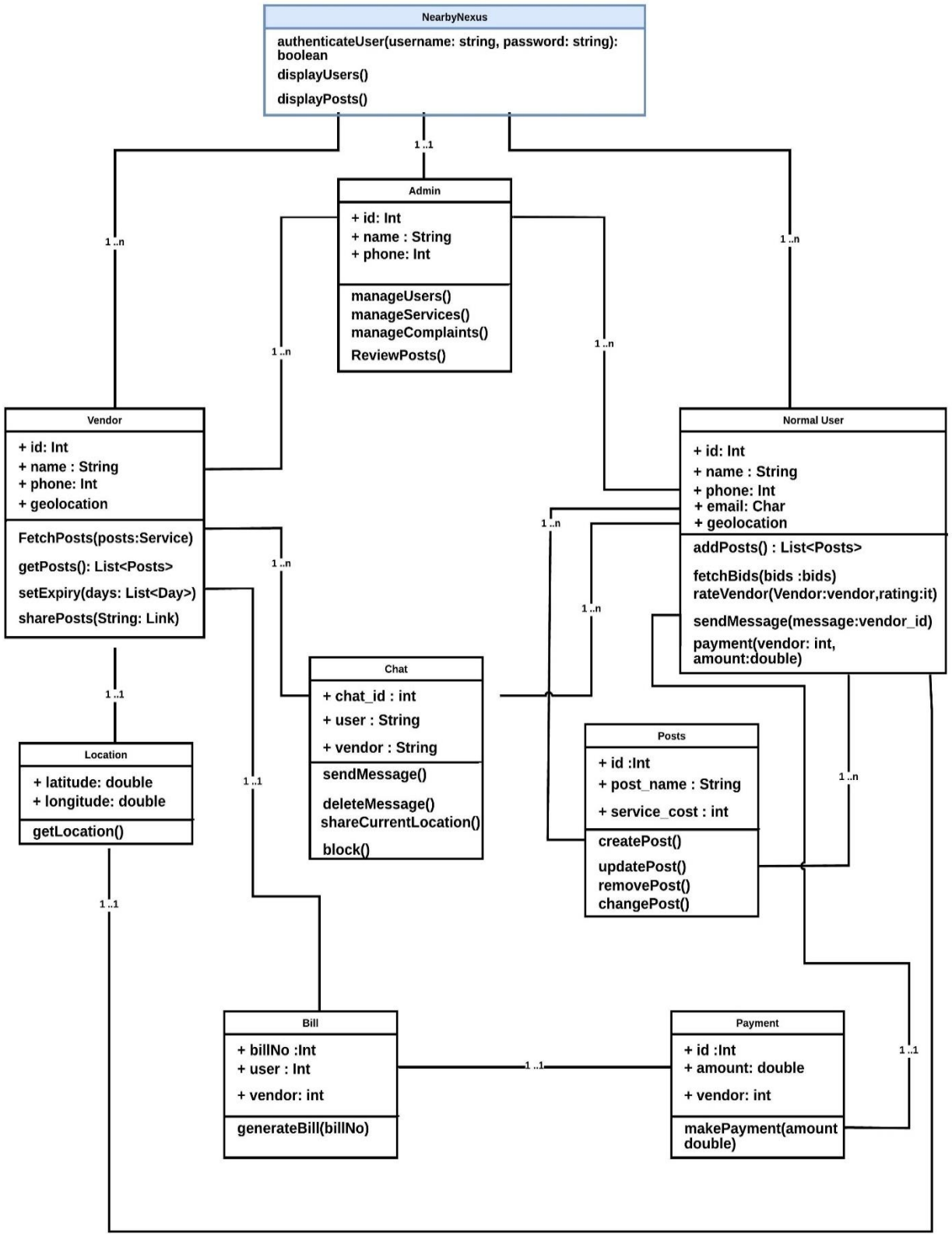
4.2.4.1 Activity diagram

4.2.5 CLASS DIAGRAM

A Class Diagram, a fundamental tool in UML, visually represents the structure of a system by illustrating classes, their attributes, methods, and relationships. Classes, depicted as rectangles, encapsulate data and behavior within a system. Associations between classes indicate relationships, showcasing how they interact. Multiplicity notations specify the cardinality of associations. Inheritance is denoted by an arrow indicating the subclass inheriting from a super-class. Aggregation and composition illustrate whole-part relationships between classes. Interfaces, depicted as a circle, outline the contract of behavior a class must implement. Stereotypes provide additional information about a class's role or purpose. Dependencies highlight the reliance of one class on another. Association classes facilitate additional information about associations. Packages group related classes together, aiding in system organization. Class Diagrams play a pivotal role in system design, aiding in conceptualizing and planning software architectures. They serve as a blueprint for the development process, ensuring a clear and structured approach to building robust software systems.

Key notations for Class Diagrams:

- **Class:** Represented as a rectangle, it contains the class name, attributes, and methods.
- **Attributes:** Displayed as a list within the class, they describe the properties or characteristics of the class.
- **Methods:** Also listed within the class, they define the behaviors or operations of the class
- **Associations:** Lines connecting classes, indicating relationships and connections between them.
- **Multiplicity Notation:** Indicates the number of instances one class relates to another.
- **Inheritance:** Shown as an arrow, it signifies that one class inherits properties and behaviors from another.
- **Interfaces:** Represented by a dashed circle, they define a contract of behavior that implementing classes must follow.
- **Stereotypes:** Additional labels or annotations applied to classes to provide more information about their role or purpose.
- **Dependencies:** Shown as a dashed line with an arrow, they indicate that one class relies on another in some way.
- **Association Classes:** Represented as a class connected to an association, they provide additional information about the relationship.

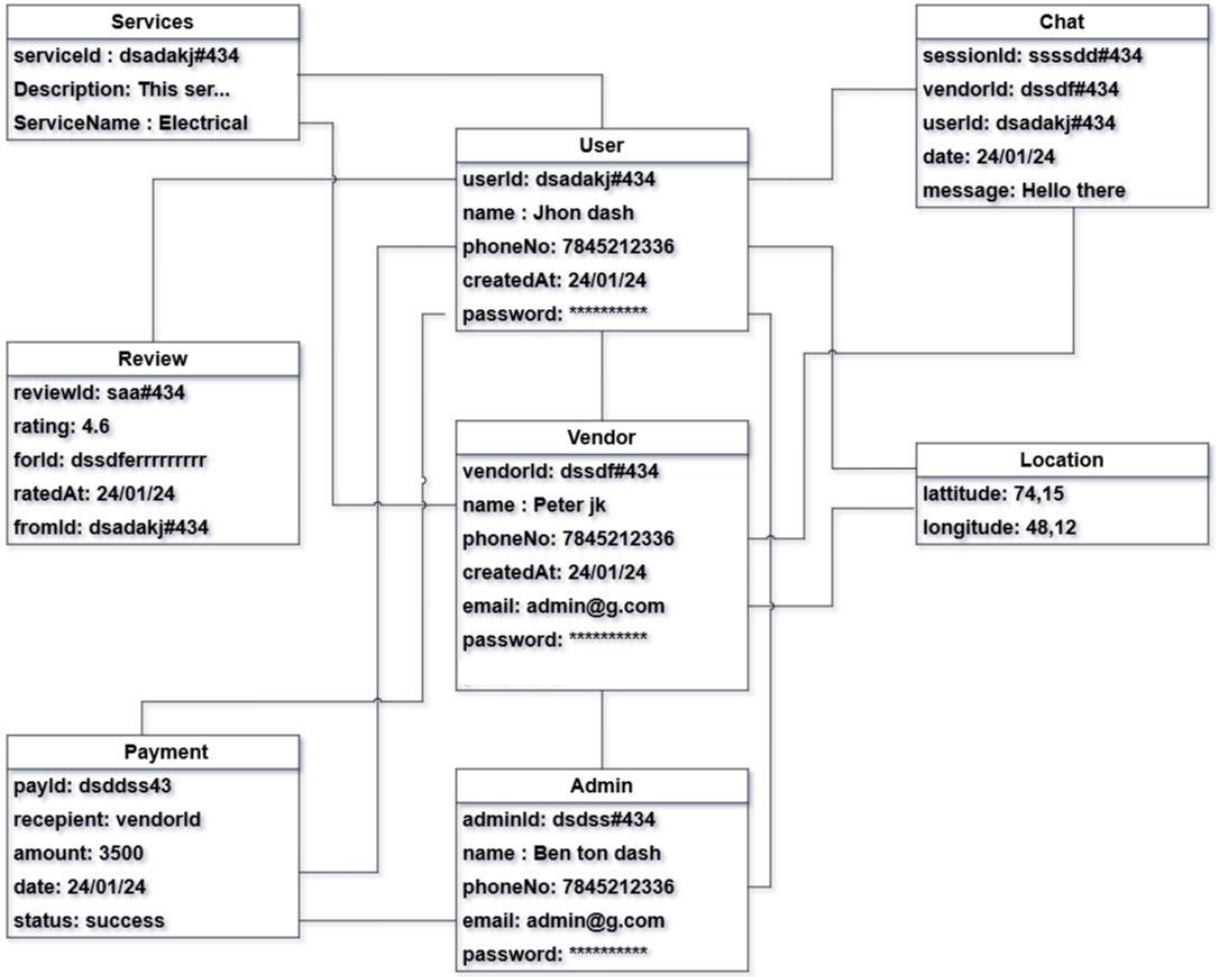


4.2.5.1 Class diagram

4.2.6 OBJECT DIAGRAM

An Object Diagram in UML provides a snapshot of a system at a specific point in time, displaying the instances of classes and their relationships. Objects, represented as rectangles, showcase the state and behavior of specific instances. Links between objects depict associations, highlighting how they interact. Multiplicity notations indicate the number of instances involved in associations. The object's state is displayed through attributes and their corresponding values. Object Diagrams offer a detailed view of runtime interactions, aiding in system understanding and testing. They focus on real-world instances, providing a tangible representation of class relationships. While similar to Class Diagrams, Object Diagrams emphasize concrete instances rather than class definitions. They serve as valuable tools for validating system design and verifying that classes and associations work as intended in practice. Object Diagrams play a crucial role in system validation, ensuring that the system's components and their interactions align with the intended design and requirements.
Key notations for Object Diagrams:

- **Object**: Represented as a rectangle, it contains the object's name and attributes with their values.
- **Links:** Lines connecting objects, indicating associations or relationships between them.
- **Multiplicity Notation:** Indicates the number of instances involved in associations.
- **Attributes with Values:** Displayed within the object, they represent the state of the object at a specific point in time.
- **Role Names:** Labels applied to associations, providing additional information about the nature of the relationship.
- **Object Name:** Represents the name of the specific instance.
- **Association End:** Indicates the end of an association, often with a role name and multiplicity.
- **Dependency Arrow:** Indicates a dependency relationship, where one object relies on another.
- **Composition Diamond:** Represents a stronger form of ownership, where one object encapsulates another.
- **Aggregation Diamond:** Signifies a whole-part relationship between objects.



4.2.6.1 Object diagram

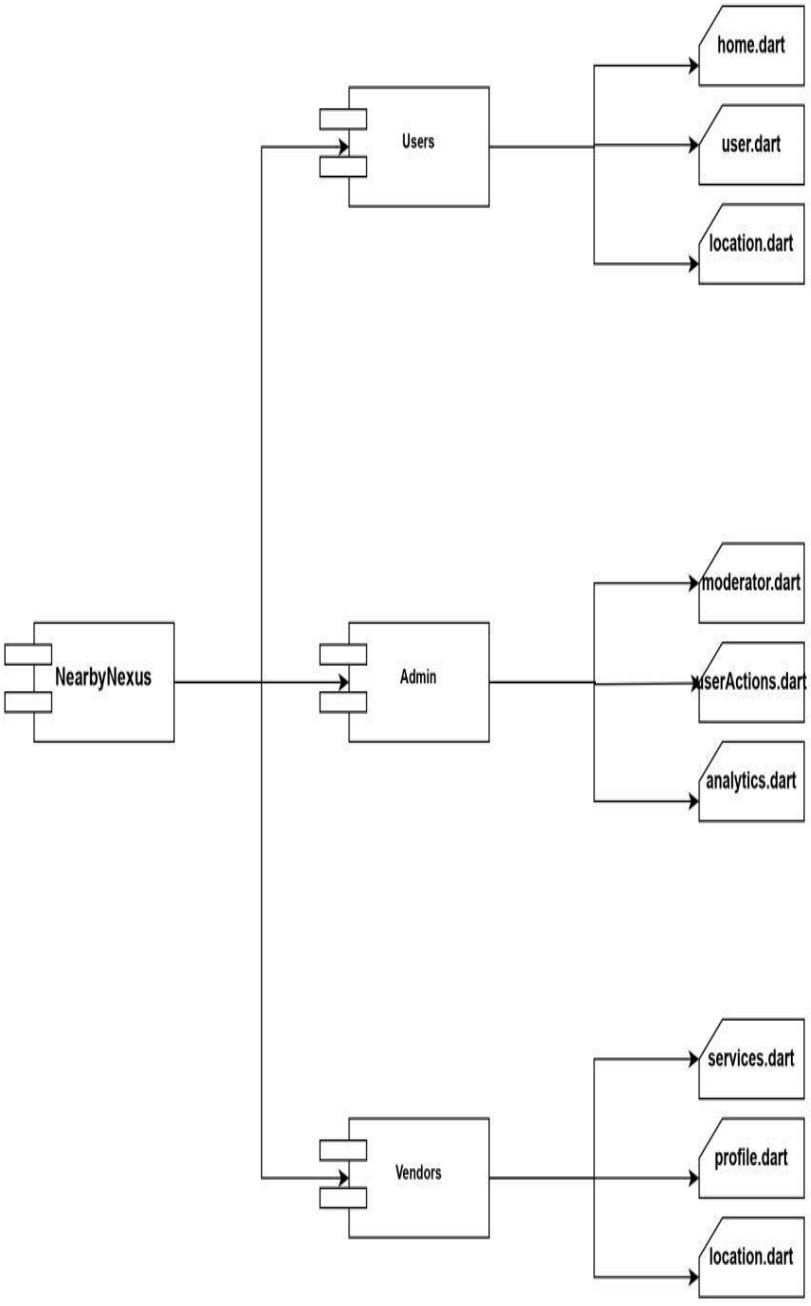
4.2.7 COMPONENT DIAGRAM

A Component Diagram, a vital aspect of UML, offers a visual representation of a system's architecture by showcasing the high-level components and their connections. Components, depicted as rectangles, encapsulate modules, classes, or even entire systems. Dependencies between components are displayed through arrows, signifying the reliance of one component on another. Interfaces, represented by a small circle, outline the services a component offers or requires. Connectors link interfaces to denote the required or provided services. Ports, depicted as small squares, serve as connection points between a component and its interfaces. Stereotypes provide additional information about the role or purpose of a component. Deployment nodes indicate the physical location or environment in which components are deployed. Component Diagrams are

instrumental in system design, aiding in the organization and visualization of system architecture. They emphasize the modular structure, facilitating ease of development, maintenance, and scalability of complex software systems. Overall, Component Diagrams play a pivotal role in planning and orchestrating the architecture of sophisticated software applications.

Key notations for Component Diagrams:

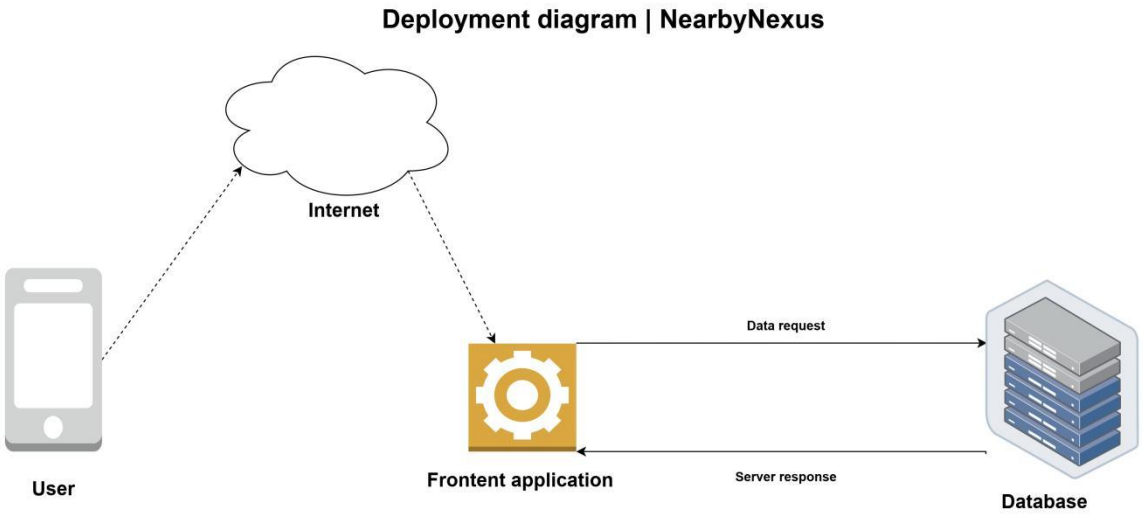
- **Component:** Represented as a rectangle, it encapsulates a module, class, or system.
- **Dependency Arrow:** Indicates that one component relies on or uses another.
- **Interface:** Depicted as a small circle, it outlines the services a component offers or requires.
- **Provided and Required Interfaces:** Connectors link provided interfaces to required interfaces.
- **Port:** Shown as a small square, it serves as a connection point between a component and its interfaces.
- **Stereotypes:** Additional labels or annotations applied to components to provide more information about their role or purpose.
- **Assembly Connector:** Represents the physical connection between two components.
- **Artifact:** A physical piece of information that is used or produced by a software development process.
- **Deployment Node:** Indicates the physical location or environment in which components are deployed.
- **Manifestation Arrow:** Indicates the implementation of an interface by a component.



4.2.7.1 Component diagram

4.2.8 DEPLOYMENT DIAGRAM

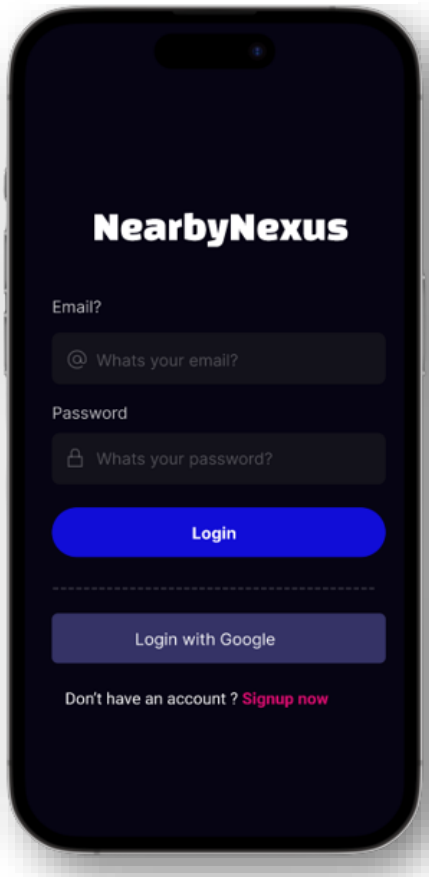
A Deployment Diagram, a crucial facet of UML, provides a visual representation of the physical architecture of a system, showcasing the hardware nodes and software components. Nodes, representing hardware entities like servers or devices, are depicted as rectangles. Artifacts, denoted by rectangles with a folded corner, represent software components or files deployed on nodes. Associations between nodes and artifacts indicate the deployment of software on specific hardware. Dependencies illustrate the reliance of one node on another. Communication paths, shown as dashed lines, represent network connections between nodes. Stereotypes provide additional information about the role or purpose of nodes and artifacts. Deployment Diagrams are instrumental in system planning, aiding in the visualization and organization of hardware and software components. They emphasize the allocation of software modules to specific hardware nodes, ensuring efficient utilization of resources. Overall, Deployment Diagrams play a pivotal role in orchestrating the physical infrastructure of complex software applications.



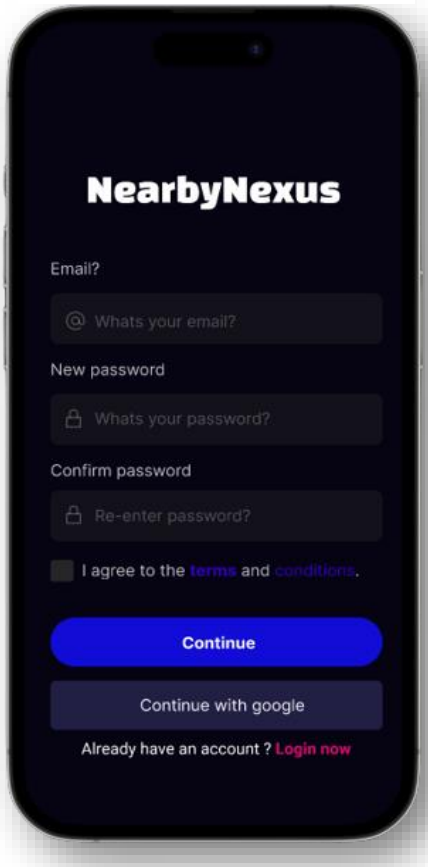
4.2.8.1 Deployment diagram

4.3 USER INTERFACE DESIGN USING FIGMA

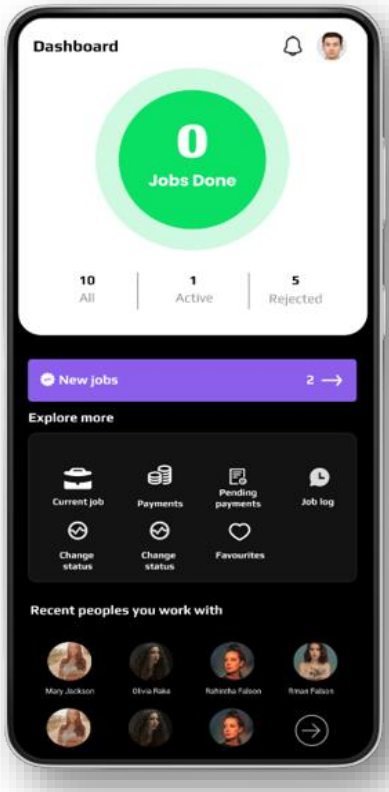
Login screen



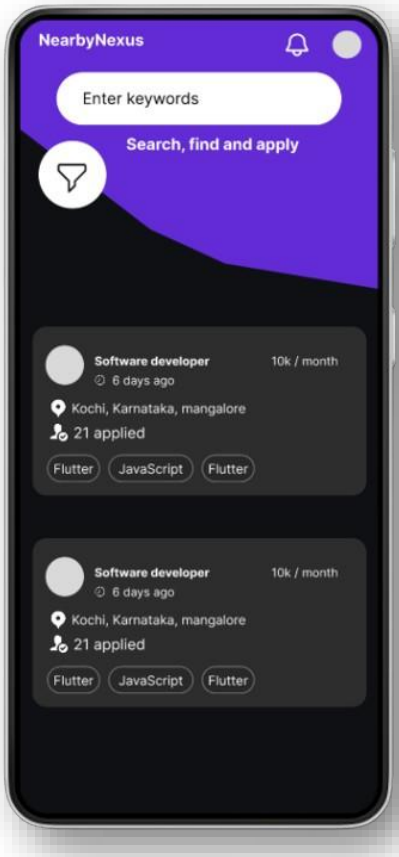
Registration screen



Vendor dashboard

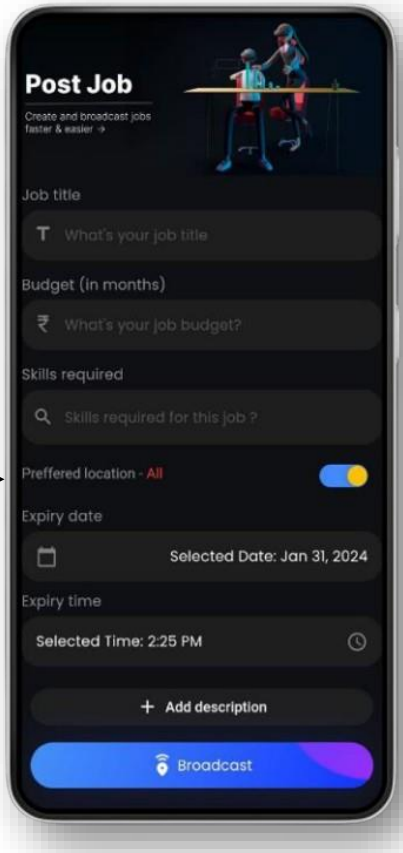


Vendor portfolio

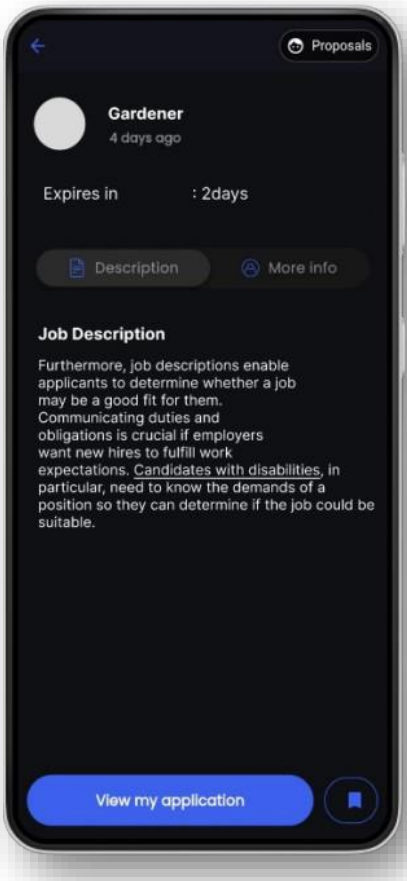


Job Posts

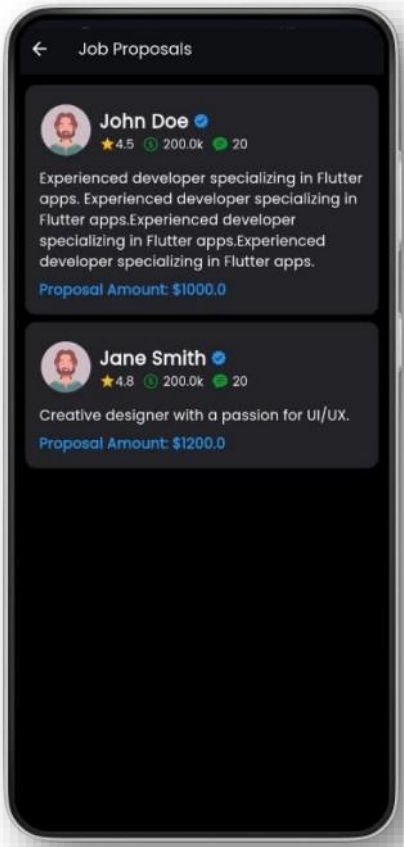
Create Job Posts



Job post view



Job proposals



4.4 DATABASE DESIGN

Database Design is a critical component in the realm of information management and software development. It involves the thoughtful and systematic organization of data to ensure efficient storage, retrieval, and manipulation. A well-designed database serves as the backbone of applications, enabling them to handle large volumes of information with speed and accuracy. This process encompasses defining the structure, relationships, and constraints of data entities, optimizing for performance and scalability. Effective database design is pivotal in minimizing redundancy, ensuring data integrity, and providing a foundation for robust data analytics. It involves a deep understanding of business requirements and user needs, translating them into a coherent and logical data model. The goal of a sound database design is to create a reliable, scalable, and maintainable system that supports the organization's objectives and facilitates seamless information flow.

4.4.1 NoSQL Databases

NoSQL databases, or "Not Only SQL," represent a transformative approach to database management. They diverge from traditional relational databases, excelling in handling large volumes of unstructured or semi-structured data. NoSQL databases are highly scalable, allowing for horizontal scaling across multiple servers, making them ideal for rapidly growing data volumes in web applications and Big Data environments.

These databases come in various types, such as document-oriented, key-value stores, wide-column stores, and graph databases. Each type caters to specific use cases. Document-oriented databases, like MongoDB, are adept at storing JSON-like documents, making them popular for content management systems and real-time analytics.

One key advantage of NoSQL databases is their schema flexibility. They allow for dynamic or semi-structured schemas, enabling data to be added or modified on the fly. This characteristic is invaluable in projects where data structures are likely to evolve over time.

NoSQL databases have found wide adoption in domains like social media, IoT applications, gaming, and real-time analytics. However, it's crucial to choose between NoSQL and traditional relational databases based on the unique requirements of the application. In essence, NoSQL databases offer a potent alternative, providing scalability, flexibility, and high performance in scenarios demanding the handling of large volumes of diverse data. Their diverse types make them indispensable tools in modern data management.

4.4.2 Indexing

Indexing in NoSQL databases is a crucial technique to enhance query performance and retrieval speed. Unlike relational databases, NoSQL databases utilize a variety of indexing methods tailored to different data models. In document-oriented databases like MongoDB, B-tree indexes are commonly used to accelerate search operations based on keys or fields within documents. Hash indexes, on the other hand, are prevalent in key-value stores like Redis, enabling swift retrieval of values associated with specific keys. Wide-column stores like Cassandra utilize techniques like row-level indexing to swiftly locate specific columns within a wide row. Graph databases like Neo4j employ specialized index structures optimized for graph traversal operations, allowing for rapid traversal of connected nodes.While indexing significantly improves read performance, it's important to weigh the trade-offs, as indexes can increase storage requirements and potentially slow down write operations. In summary, indexing plays a vital role in optimizing query performance and is a key aspect of designing efficient data retrieval systems in NoSQL databases.

4.5 TABLE DESIGN

Collection name**: 4.5.1 users**

No	Field name	Datatype	Description
1	name	String	Stores the name of the user.
2	about	String	Stores the about description of the user.
3	address	String	Stores the address of the user.
4	geoLocation	String	Stores the geo-location of the user.
5	image	String	Contains the profile image of the user.
6	status	String	Stores the account status of the user whether he/she is disabled or banned from the application etc..
7	userType	String	Specifies the user's type.
8	activityStatus	String	Specifies if the user is available or not.
9	actualRating	Number	Contains the rating score of the user out of 5
10	totalRating	Number	Contains the sum of the all ratings
11	emailId	Map	Stores the email id of the user and the verification status of the email id.

12	phone	Map	Stores the phone number of the user and the verification status of the phone number
13	kycDetails	Map	Stores the necessary details related users KYC
14	docId	Document	This id is auto generated by Firebase & and act as individual user id.
15	iamRated	Array(references)	The list contains the users who have been rated by the user.
16	paymentLogs	Array(references)	Stores the reference id to the payments collection.
17	userFavourites	Array(references)	The system stores the reference IDs of users who have been added to the user's favorite list.
18	allRatings	Array(references)	This refers to the 'ratings' collection, which contains details pertaining to user ratings.
19	languages	Array	Lists user's spoken languages.
	services	Array	Lists the services provided by the user.
	workingDays	Array	Lists days the user available.

4.5.1 Table (Collection)

Collection name **: 4.5.2 service_actions**

No	Field name	Datatype	Description
1	docId	Document	Unique id to identify the document.
2	clientStatus	String	Status field which is updated by the client based on the ongoing activity(services).
3	description	String	Description of the service that the client needs.
4	location	String	Location of the job to be serviced.
5	service_level	String	Level emergency of the service needed.
6	service_name	String	Name of the service.
7	status	String	Status which is updated by the provider based on the flow of the job.
8	wage	String	The amount agreed by both the users.
9	paymentStatus	String	Provides the status of the payment after the service is done.

10	dateRequested	Timestamp	Snapshot of the date and time when the user requested the job.
11	day	Timestamp	Specifies the day the job need to be done.
12	paymentLog	References	Stores the reference of the payment collection.
13	referencePath	References	Stores the reference id of the service provider.
14	userReference	References	Stores the reference id of the user(client).
15	jobLogs	Array	Stores the activity log of the job.

4.5.2 Table (Collection)

Collection name**: 4.5.3 ratings**

No	Field name	Datatype	Description
1	docId	Document	Unique id to identify the document.
2	feedback	String	Stores the users feedback.
3	jobReference	Reference	This refers to the 'service_actions' collection, which provides information about the job for which the rating was given.
4	ratedBy	Reference	This refers to the collection ‘users’.
5	ratedTo	Reference	This refers to the collection ‘users’.
6	rating	Number	Contains the rating value.
7	timeRated	Timestamp	Supplies the date and time when the rating was submitted.

4.5.3 Table (Collection)

Collection name :4.5.4 payments

No	Field name	Datatype	Description
1	docId	Document	Unique id to identify the document.
2	amountPaid	String	Provides the amount paid.
3	jobId	Reference	This refers to the 'service_actions' collection, which provides information about the job for which the payment was given.
4	payedBy	Reference	This refers to the collection ‘users’.
5	payedTo	Reference	This refers to the collection ‘users’.
6	paymentTime	Timestamp	Supplies the date and time when the payment was done.
7	payedFor	String	Describes the type of payment.
8	applicationRevenue	String	Describes the revenue generated by the application from each transaction.

4.5.4 Table (Collection)

Collection name :4.5.5 applications

No	Field name	Datatype	Description
1	Applicant_id	Document	Unique id to identify the document.
2	applicationPostedTime	Timestamp	Time at which the bid happened.
3	bid_amount	String	Amount that the bid fixed for by the vendor.
4	jobId	Reference	This refers to the collection ‘job_posts’.
5	payedTo	Reference	This refers to the collection ‘users’.
6	proposal_description	String	Description of the proposal.
7	status	String	Status of the proposal whether accepted or rejected.

4.5.5 Table (Collection)

Collection name **:4.5.6 job_posts**

No	Field name	Datatype	Description
1	applicants	List (Documents)	List of documents that refer to the users who bid on the current job.
2	budget	String	Amount set by the job poster.
3	expiryDate	Timestamp	Time at which the job expires.
4	isWithDrawn	Boolean	Determines whether the post removed or not.
5	jobDescription	String	Description of the current job.
6	JobPostDate	Timestamp	Date & time at which the job is published.
7	jobPostBy	Reference	Reference to the user who posted the job.
8	jobTitle	String	Title of the job.
9	skills	List(String)	List of the skills needed for the job.
10	Status	Map	Determines whether the job is active and also reason if it's not active.

4.5.6 Table (Collection)

CHAPTER 5
SYSTEM TESTING

5.1 INTRODUCTION

System Testing is a crucial phase in the software development life cycle, where the entire system is evaluated against specified requirements and functionalities. It is a comprehensive and structured approach to validate that the software meets its intended objectives. This phase involves testing the integrated system as a whole to ensure that all components work together seamlessly. System Testing verifies the system's compliance with both functional and non-functional requirements, including performance, security, and usability. It is conducted in an environment that closely simulates the production environment, providing a real-world scenario for testing. The primary goal of System Testing is to identify and rectify any discrepancies or defects before the software is deployed to end-users. Through rigorous testing processes and thorough documentation, System Testing helps in delivering a reliable and high-quality software product.

Testing is the systematic process of running a program to uncover potential errors or flaws. An effective test case possesses a high likelihood of revealing previously unnoticed issues. A test is considered successful when it reveals a previously unidentified error. If a test functions as intended and aligns with its objectives, it can detect flaws in the software. The test demonstrates that the computer program is operating in accordance with its intended functionality and performing optimally. There are three primary approaches to assessing a computer program: evaluating for accuracy, assessing implementation efficiency, and analyzing computational complexity.

5.2 TEST PLAN

A test plan is a thorough document that delineates the strategy, scope, objectives, resources, schedule, and expected outcomes for a specific testing endeavor. It functions as a guiding framework for carrying out testing activities, guaranteeing that every facet of the testing process is methodically organized and executed. Additionally, the test plan establishes the roles and responsibilities of team members, outlines the required testing environment, and sets forth the criteria for the successful completion of testing activities. This document plays a pivotal role in ensuring that the testing phase is conducted in a structured and effective manner, ultimately contributing to the overall success of the project. The levels of testing include:

- Integration Testing
- Unit testing
- Validation Testing or System Testing
- Output Testing or User Acceptance Testing
- Automation Testing

- Widget Testing

5.2.1 Integration Testing

Integration Testing stands as a pivotal phase in the software testing process, dedicated to scrutinizing the interactions and interfaces among diverse modules or components within a software system. Its primary objective is to ascertain that individual units of code seamlessly converge to create a unified and functional system. In stark contrast to unit testing, which assesses individual units in isolation, integration testing delves into the interplay between these units, with a keen eye for any disparities, communication glitches, or integration hurdles. By subjecting the integrated components to rigorous testing, development teams aim to affirm that these elements function cohesively, addressing any potential issues before deployment. This systematic evaluation is instrumental in ensuring that the software operates as an integrated whole, free from any unforeseen conflicts or errors that may arise from the convergence of individual modules.

5.2.2 Unit Testing

Unit Testing is not only a meticulous examination of discrete units or components within a software system but also an indispensable quality assurance measure. This phase serves as a crucial foundation for the entire software testing process, where the focus lies on isolating and scrutinizing individual units of code. The objective remains unwavering: to verify that each unit performs its designated function accurately, yielding precise outputs for predefined inputs.

Moreover, Unit Testing operates independently, detached from other components, and any external dependencies are either emulated or replaced by "mock" objects, ensuring controlled evaluation. This meticulous process establishes a robust foundation for the software, confirming that each unit functions reliably and adheres meticulously to its predefined behavior.

The significance of Unit Testing cannot be overstated, as it acts as a vanguard against potential discrepancies or errors early in the development cycle. This proactive approach not only fortifies the integrity and reliability of the software but also lays the groundwork for subsequent testing phases, thereby fostering a robust and dependable software solution.

This meticulous process ensures that each unit functions reliably and adheres precisely to its defined behavior. By subjecting individual code units to rigorous scrutiny, any discrepancies or errors are identified and rectified early in the development cycle, bolstering the overall integrity and reliability of the software.

5.2.3 Validation Testing or System Testing

Validation Testing places the end-users at the forefront of evaluation, ensuring that the software aligns precisely with their anticipated needs and expectations. This phase stands distinct from other

testing methodologies, as its primary objective is to authenticate that the software, in its final form, serves its intended purpose seamlessly within the real-world scenarios it was designed for.

As a culmination of the testing process, Validation Testing carries the responsibility of confirming that the software not only meets the defined technical specifications but also delivers genuine value to its users. It does so by scrutinizing the software against the backdrop of actual usage, thereby fortifying its readiness for deployment.

Moreover, in Validation Testing, user stories and acceptance criteria form the cornerstone of assessment. Stakeholders' expectations are meticulously validated, ensuring that every specified requirement is met. Additionally, beta testing, a common practice in this phase, involves a select group of end-users testing the software in a live environment, providing invaluable feedback that can inform potential refinements.

5.2.4 Output Testing or User Acceptance Testing.

Output Testing, also known as Results Validation, is a critical phase in the software testing process. Its primary focus is to verify the correctness and accuracy of the output generated by a software application. The goal is to ensure that the system produces the expected results for a given set of inputs and conditions.

Key aspects of Output Testing include:

- **Comparison with Expected Results:** This phase involves comparing the actual output of the software with the expected or predefined results.
- **Test Case Design:** Test cases are designed to cover various scenarios and conditions to thoroughly evaluate the accuracy of the output.
- **Validation Criteria**: The criteria for validating the output are typically defined during the requirements and design phase of the software development process.
- **Regression Testing:** Output Testing often includes regression testing to ensure that changes or updates to the software do not affect the correctness of the output.
- **Data Integrity:** It verifies that data is processed and displayed correctly, without any corruption or loss.
- **Precision and Completeness:** Output Testing assesses not only the precision of the results but also their completeness in addressing the requirements.
- **Error Handling:** It evaluates how the system handles errors or exceptions and ensures that appropriate error messages are displayed.

5.2.5 Automation Testing

Automation Testing stands as a cornerstone in the software testing process, harnessing the power of automated tools and scripts to meticulously execute test cases. In stark contrast to manual testing, which hinges on human intervention, automation testing brings forth a streamlined approach, employing software to conduct repetitive, intricate, and time-consuming tests. This methodology not only heightens operational efficiency but also significantly diminishes the likelihood of human error, ensuring precise and reliable results. Moreover, it empowers thorough testing across a diverse array of scenarios and configurations, from browser compatibility to load and performance assessments.

By automating the testing process, organizations can realize a myriad of benefits. It enables the seamless execution of regression tests, providing confidence that existing functionalities remain intact after each round of enhancements or modifications. Furthermore, automation facilitates the concurrent execution of multiple tests, thereby expediting the overall testing cycle. This approach is particularly invaluable in environments characterized by rapid development and frequent software updates, such as Agile and DevOps setups.

5.2.6 Widget Testing

Widget Testing in Flutter is a critical step towards building robust and reliable user interfaces. By isolating and scrutinizing individual widgets, developers gain confidence in the functionality and appearance of each component. This level of granularity allows for precise testing, ensuring that widgets respond correctly to various user interactions and scenarios.

One of the key advantages of Widget Testing is the ability to mock dependencies. This means that external services or resources that the widget relies on can be simulated, allowing for controlled and predictable testing environments. Additionally, developers can set expectations and employ assertions to validate the widget's behavior under different conditions.

Furthermore, Widget Testing is seamlessly integrated with popular testing frameworks like `flutter_test`. This ensures that tests can be organized efficiently and run as part of the automated testing process, providing rapid feedback on the status of widgets.

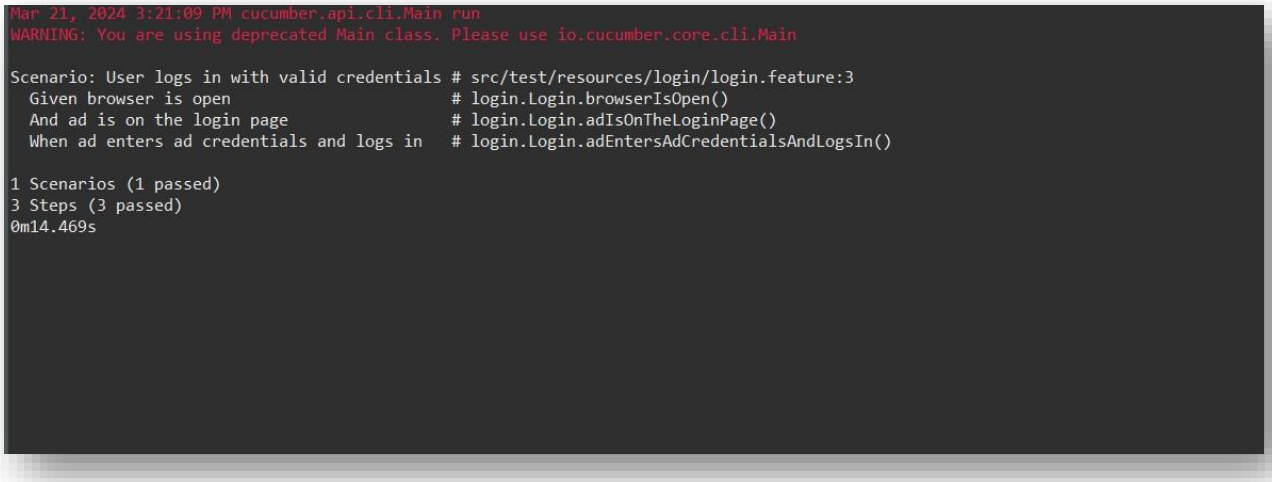
Another important facet of Widget Testing is the inclusion of Golden Tests. These tests capture screenshots of the widget's visual appearance and compare them against reference images. This helps maintain visual consistency, making sure that UI elements render consistently across different devices and screen sizes.

Test Case 1

Code

```
package login;
import io.cucumber.java.en.Given;
```

```
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.en.And;
public class Login {
   WebDriver driver = null;
   JavascriptExecutor js = null;
   @Given("browser is open")
   public void browserIsOpen() {
      System.setProperty("webdriver.gecko.marionette", "C:\\Users\\donbe\\eclipse-
workspace\\login\\src\\test\\resources\\drivers\\geckodriver.exe");
      driver = new FirefoxDriver();
      driver.manage().window().maximize();
   }
   @And("ad is on the login page")
   public void adIsOnTheLoginPage() throws Exception {
      driver.navigate().to("http://localhost:3000/");
      Thread.sleep(3000);
   }
   @When("ad enters ad credentials and logs in")
   public void adEntersAdCredentialsAndLogsIn() throws Throwable {
      try {
         driver.findElement(By.id("email_feild")).sendKeys("hexated100@gmail.com");
         driver.findElement(By.id("password_field")).sendKeys("2588520@Don");
         driver.findElement(By.id("submit_btn")).click();
         Thread.sleep(3000);
      } catch (Exception e) {
         e.printStackTrace();
      } finally {
         if (driver != null) {
            driver.quit();
         }
      }
   }
}
```



Test case 1 - Screenshot

Test Report

Test Case 1

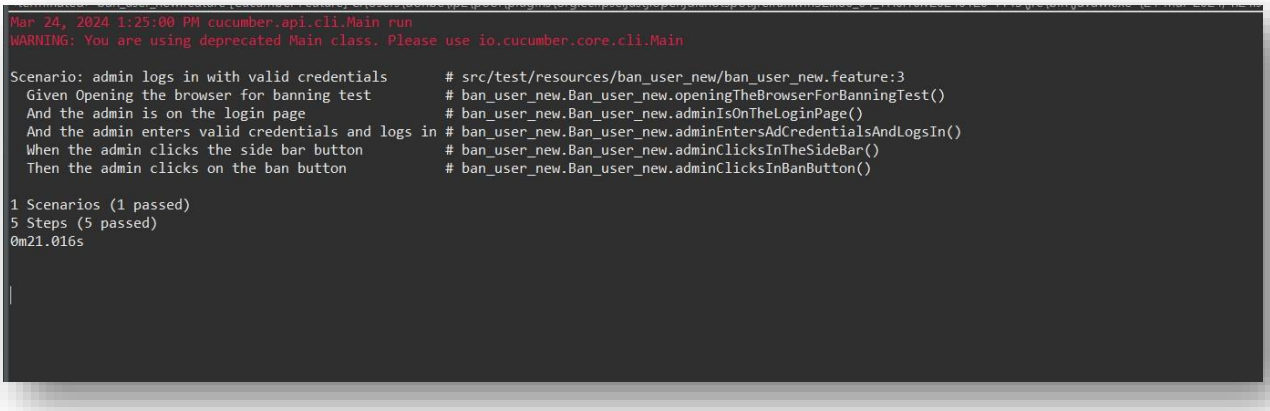
Project Name: NearbyNexus					
Login Test Case					
Test Case ID: Test_1			**Test Designed By: Don Benny**		
Test Priority (Low/Medium/High):High			**Test Designed Date: 18 March 2024**		
Module Name: Login Screen			**Test Executed By: Mr. Binumon Joseph**		
Test Title: Login Scenario			**Test Execution Date:25 March 2024**		
Description: Test the login page					
Pre-Condition: Admin has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fai l)**
1	Navigate to login screen		Navigate to login screen	Navigated to login screen	Pass
2	Trying to login with valid credentials	hexated100@gmail.com 258552@Dm	Admin should be navigated to the dashboard	Dashboard displayed	Pass
Post-Condition: After successful validation with the database, the admin is able to log into their account.					

Test Case 2:

Code

```
package ban_user_new;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.After;
import io.cucumber.java.Before;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
public class Ban_user_new {
   WebDriver driver = null;
   @Before // This method will run before each scenario
   public void setUp() {
            System.setProperty("webdriver.gecko.marionette "C:\\Users\\donbe\\eclipse-
workspace\\ban_user\\src\\test\\resources\\drivers\\geckodriver.exe");
      driver = new FirefoxDriver();
      driver.manage().window().maximize();
   }
   @After // This method will run after each scenario
   public void tearDown() {
      if (driver != null) {
         driver.quit();
      }
   }
   @Given("Opening the browser for banning test")
   public void openingTheBrowserForBanningTest() {
      // No need to initialize the driver here anymore
   }
   @And("the admin is on the login page")
   public void adminIsOnTheLoginPage() throws InterruptedException {
      driver.navigate().to("http://localhost:3000/");
      Thread.sleep(3000);
   }
   @And("the admin enters valid credentials and logs in")
   public void adminEntersAdCredentialsAndLogsIn() throws InterruptedException {
      driver.findElement(By.id("email_feild")).sendKeys("hexated100@gmail.com");
      driver.findElement(By.id("password_field")).sendKeys("2588520@Don");
      driver.findElement(By.id("submit_btn")).click();
      Thread.sleep(3000);
   }
   @When("the admin clicks the side bar button")
   public void adminClicksInTheSideBar() throws InterruptedException {
      driver.navigate().to("http://localhost:3000/users");
```

```
        Thread.sleep(3000);
    }
    @Then("the admin clicks on the ban button")
    public void adminClicksInBanButton() throws InterruptedException {
        driver.findElement(By.id("user_banButton_1")).click();
        Thread.sleep(3000);
    }
}
```



Test case 2 – Screenshot

Test report.

Test Case 2					
Project Name: NearbyNexus					
Banning a user					
Test Case ID: Test_1			**Test Designed By: Don Benny**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 18 March 2024**		
Module Name: Ban user			**Test Executed By: Mr. Binumon Joseph**		
Test Title: Admin function test			**Test Execution Date: 25 March 2024**		
Description: Testing functionalities of the admin for banning					
Pre-Condition: User has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/ Fai l)**
1	Navigate to login screen		Navigate to login screen	Navigated to login screen	Pass

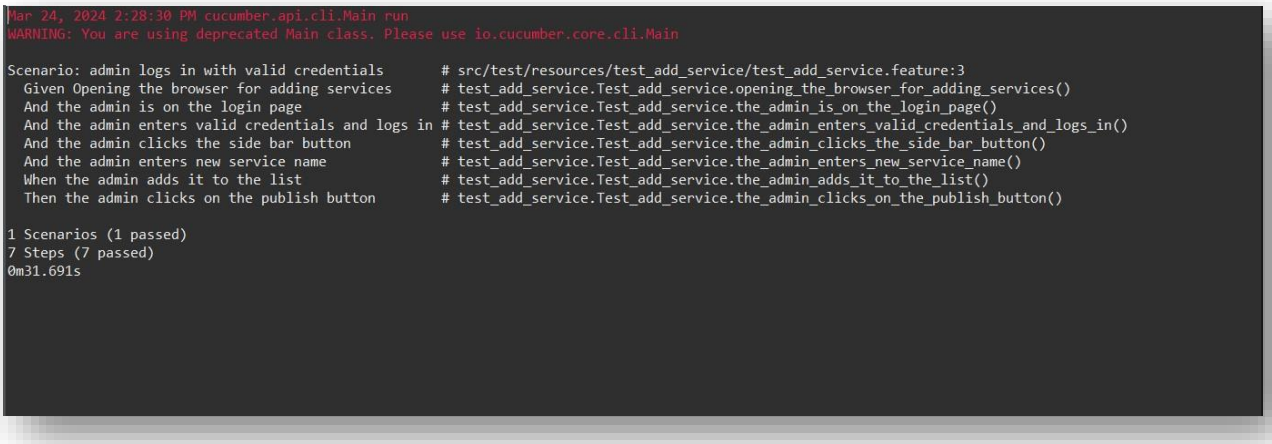
2	Trying to login with valid credentials	hexated100@gmail.com 258552@Dm	Admin should be navigated to the dashboard	Dashboard displayed	Pass
3	The admin clicks the side bar button	Tap on side bar button	Admin should Navigate to vendor user data screen	Navigated to user data screen	Pass
4	The admin clicks on the ban button	Tap on ban button	The user should be banned from the application	Successfully banned the user.	Pass
Post-Condition: After successful validation with the database, the user is able to log into their account. admin can be able to ban user from the application.					

Test Case 3:

Code

```
package test_add_service;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.After;
import io.cucumber.java.Before;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
public class Test_add_service {
   WebDriver driver = null;
   @Given("Opening the browser for adding services")
   public void opening_the_browser_for_adding_services() {
      System.setProperty("webdriver.gecko.marionette", "C:\\Users\\donbe\\eclipse-
workspace\\test_add_service\\src\\test\\resources\\drivers\\geckodriver.exe");
      driver = new FirefoxDriver();
      driver.manage().window().maximize();
   }
   @Given("the admin is on the login page")
   public void the_admin_is_on_the_login_page() throws Exception {
      driver.navigate().to("http://localhost:3000/");
      Thread.sleep(3000);
   }
   @Given("the admin enters valid credentials and logs in")
   public void the_admin_enters_valid_credentials_and_logs_in() throws Exception {
      driver.findElement(By.id("email_feild")).sendKeys("hexated100@gmail.com");
      driver.findElement(By.id("password_field")).sendKeys("2588520@Don");
      driver.findElement(By.id("submit_btn")).click();
      Thread.sleep(3000);
   }
```

```
    @Given("the admin clicks the side bar button")
    public void the_admin_clicks_the_side_bar_button() throws Exception {
        driver.navigate().to("http://localhost:3000/services");
        Thread.sleep(5000);
    }
    @Given("the admin enters new service name")
    public void the_admin_enters_new_service_name() throws InterruptedException {
        driver.findElement(By.id("serviceNameInput")).sendKeys("Oil service");
        Thread.sleep(3000);
    }
    @When("the admin adds it to the list")
    public void the_admin_adds_it_to_the_list() throws InterruptedException {
        driver.findElement(By.id("add_service_btn")).click();
        Thread.sleep(3000);
    }
    @Then("the admin clicks on the publish button")
    public void the_admin_clicks_on_the_publish_button() throws Exception {
        driver.findElement(By.id("publish_btn")).click();
        Thread.sleep(3000);
    }
    @After
    public void tearDown() {
        if (driver != null) {
            driver.quit();
        }
    }
}
```



Test case 3 - Screenshot

Test report.

Test Case 3

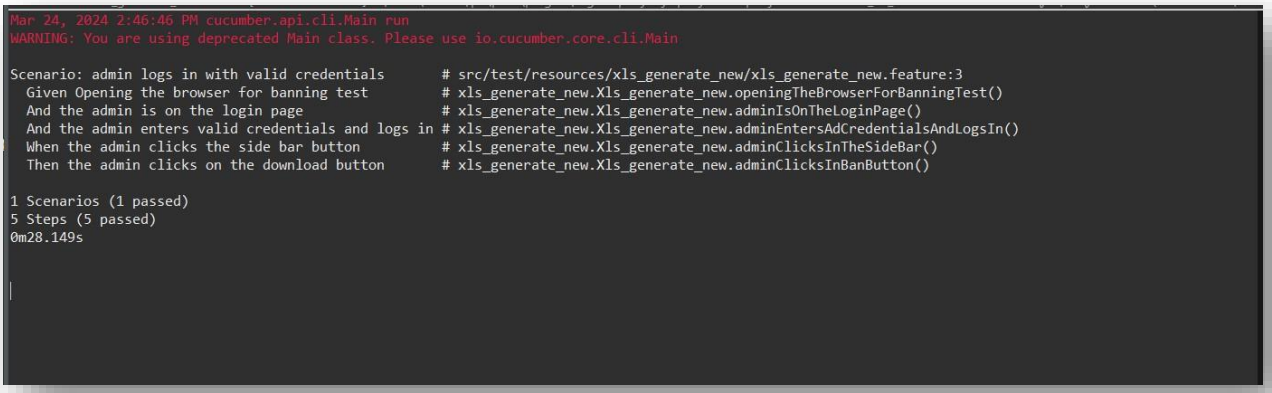
Project Name: NearbyNexus					
Admin adds services					
Test Case ID: Test_1			**Test Designed By: Don Benny**		
Test Priority (Low/Medium/High):High			**Test Designed Date: 18 March 2024**		
Module Name: Service add screen			**Test Executed By: Mr. Binumon Joseph**		
Test Title: service adding			**Test Execution Date: 25 March 2024**		
Description: Testing functionalities of the service adding					
Pre-Condition : Vendor have to be negotiated the amount					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fai l)**
1	Navigate to login screen		Navigate to login screen	Navigated to login screen	Pass
2	Trying to login with valid credentials	hexated100@gmail.com 258552@Dm	Admin should be navigated to the dashboard	Dashboard displayed	Pass
3	The admin clicks the side bar button	Tap on side bar button	Admin should Navigate to vendor add service screen	Navigated to add service screen	Pass
4	The admin enters new service name	Test service	Data should add into the text box	Service name added to the textbox	Pass
5	The admin adds it to the list	Taps on the plus button	Service name should add to the list	Service name successfully added to the list	Pass
6	The admin clicks on the publish button	Tap on the publish button	List data should be added to the database.	List data successfully added to the database.	Pass
Post-Condition: After successful validation with the database, the admin is able to log into their account. The service names should be added to the database.					

Test Case 4:

Code

```
package xls_generate_new;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.After;
import io.cucumber.java.Before;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
public class Xls_generate_new {
        WebDriver driver = null;
   @Before // This method will run before each scenario
   public void setUp() {
         System.setProperty("webdriver.gecko.marionette", "C:\\Users\\donbe\\eclipse-
workspace\\ban_user\\src\\test\\resources\\drivers\\geckodriver.exe");
      driver = new FirefoxDriver();
      driver.manage().window().maximize();
   }
   @After // This method will run after each scenario
   public void tearDown() {
      if (driver != null) {
         driver.quit();
      }
   }
   @Given("Opening the browser for banning test")
   public void openingTheBrowserForBanningTest() {
      // No need to initialize the driver here anymore
   }
   @And("the admin is on the login page")
   public void adminIsOnTheLoginPage() throws InterruptedException {
      driver.navigate().to("http://localhost:3000/");
      Thread.sleep(3000);
   }
   @And("the admin enters valid credentials and logs in")
   public void adminEntersAdCredentialsAndLogsIn() throws InterruptedException {
      driver.findElement(By.id("email_feild")).sendKeys("hexated100@gmail.com");
      driver.findElement(By.id("password_field")).sendKeys("2588520@Don");
      driver.findElement(By.id("submit_btn")).click();
      Thread.sleep(3000);
   }
   @When("the admin clicks the side bar button")
   public void adminClicksInTheSideBar() throws InterruptedException {
```

```
        driver.navigate().to("http://localhost:3000/users");
        Thread.sleep(3000);
    }
    @Then("the admin clicks on the download button")
    public void adminClicksInBanButton() throws InterruptedException {
        driver.findElement(By.id("download_as_xl")).click();
        Thread.sleep(3000);
    }
}
```



Test case 4 - Screenshot

Test report.

Test Case 4

Project Name: NearbyNexus					
User data generate & download					
Test Case ID: Test_1			**Test Designed By: Don Benny**		
Test Priority(Low/Medium/High):High			**Test Designed Date: 18 March 2024**		
Module Name: user report generate			**Test Executed By: Mr. Binumon Joseph**		
Test Title : User report test			**Test Execution Date: 25 March 2024**		
Description: Testing functionalities of the admin					
Pre-Condition: admin have to logged in					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fai l)**
1	Navigate to login screen		Navigate to login screen	Navigated to login screen	Pass
2	Trying to login with valid credentials	hexated100@gmail.com 258552@Dm	Admin should be navigated to the dashboard	Dashboard displayed	Pass

3	The admin clicks the side bar button	Tap on side bar button	Admin should Navigate to vendor user data screen	Navigated to user data screen	Pass
4	The admin clicks on the download button	Tap on download button	The .csv file should be generated with users' data	The .csv file generated with users data	Pass
Post-Condition: After successful validation with the database, the user is able to log into their account. .csv file generated.					

CHAPTER 6
IMPLEMENTATION

6.1 INTRODUCTION

Project implementation is the phase where plans and strategies transform into tangible actions and outcomes. It marks the transition from theoretical concepts to practical application. This pivotal stage requires meticulous planning, resource allocation, and a dedicated team to execute tasks according to the established timeline and objectives. In this phase, the project team translates the project's blueprints into real-world activities, ensuring that each step aligns with the overarching goals. Effective project implementation demands clear communication, robust leadership, and a keen eye for detail. This introduction sets the stage for a comprehensive understanding of the project implementation process, emphasizing its significance in achieving the envisioned goals. As we delve deeper, we will explore key components, strategies, and best practices that contribute to successful project implementation.

The crux of successful project implementation lies not just in technical proficiency, but also in the art of effective communication. Clear channels of dialogue serve as the lifeblood that sustains the project's momentum, fostering synergy among team members and stakeholders alike. A bedrock of robust leadership provides the necessary guidance and inspiration, steering the ship through uncharted waters with confidence and purpose. Additionally, an unyielding commitment to detail acts as the linchpin that secures the integrity of each executed task.

This introduction sets the stage for a profound comprehension of the project implementation process, underlining its indomitable significance in realizing the envisioned goals. As we embark on this journey of exploration, we will unfurl the tapestry of key components, unveil strategies, and illuminate best practices that form the crucible of triumphant project implementation.

The implementation state involves the following tasks:

- Careful planning.
- Investigation of system and constraints.
- Design methods to achieve the changeover.

6.2 IMPLEMENTATION PROCEDURES

6.2.1 User Training

User training is a critical component of ensuring the effective utilization of any application software. It involves imparting the necessary knowledge and skills to end-users, enabling them to navigate and utilize the software efficiently. This training equips users with a comprehensive understanding of the software's features, functions, and capabilities. Through hands-on sessions and guided tutorials, users learn how to perform tasks, customize settings, and troubleshoot

common issues. Moreover, user training fosters confidence and proficiency, empowering individuals to maximize their productivity while using the application. Regular updates and refresher sessions further enhance user competence, ensuring they stay abreast of new features and functionalities.

6.2.2 Training on the Application Software

Training on the application software is a structured program designed to familiarize individuals with the intricacies and functionalities of a specific software application. It encompasses a range of topics, from basic navigation to advanced features, tailored to meet the diverse needs of users. This training often includes interactive demonstrations, hands-on exercises, and Q&A sessions to facilitate effective learning. Trainers may also provide supplemental resources such as user manuals or online guides for reference. By the end of the training, participants are equipped skills and knowledge required to proficiently utilize the application software in their respective contexts.

6.2.3 System Maintenance

System maintenance is a crucial aspect of ensuring the seamless operation and longevity of any software application. It encompasses a series of tasks aimed at monitoring, optimizing, and troubleshooting the underlying infrastructure on which the application runs. This includes activities such as regular software updates, performance monitoring, and data backups. Additionally, system maintenance involves identifying and rectifying any potential vulnerabilities or inefficiencies that may impede the software's performance. Proactive maintenance measures contribute to a stable and secure environment, minimizing the risk of unexpected downtime or data loss.

6.2.4 Installing

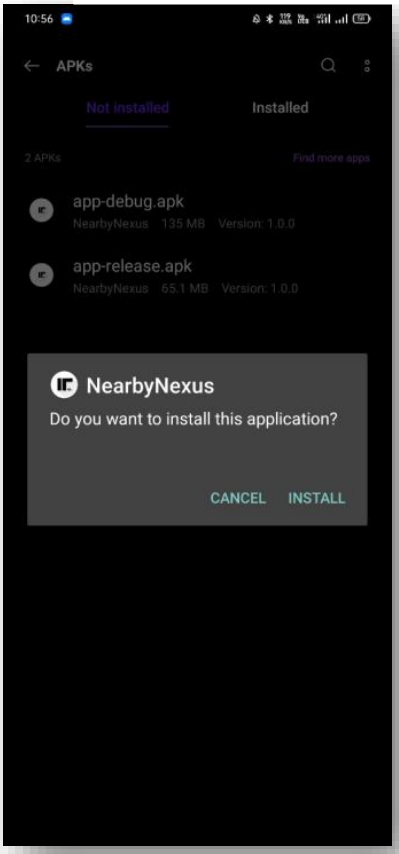
Installing this application is a straightforward process that grants accessibility on mobile devices supporting API versions from 29 to 30. Here's a concise guide:

- **Compatibility Verification:** Ensure your mobile device runs on API version 29 to 30 for seamless installation and operation.
- **Download the APK:** Obtain the APK file from a trusted source, ensuring it matches the application's official version.
- **Adjust Security Settings:** If needed, enable 'Unknown Sources' in your device's settings to allow APK installations.
- **Locate and Run the APK:** Access the downloaded APK file, typically found in the 'Downloads' folder, and tap to initiate installation.
- **Permissions and Installation:** Grant necessary permissions for the application to function optimally and proceed with the installation process.

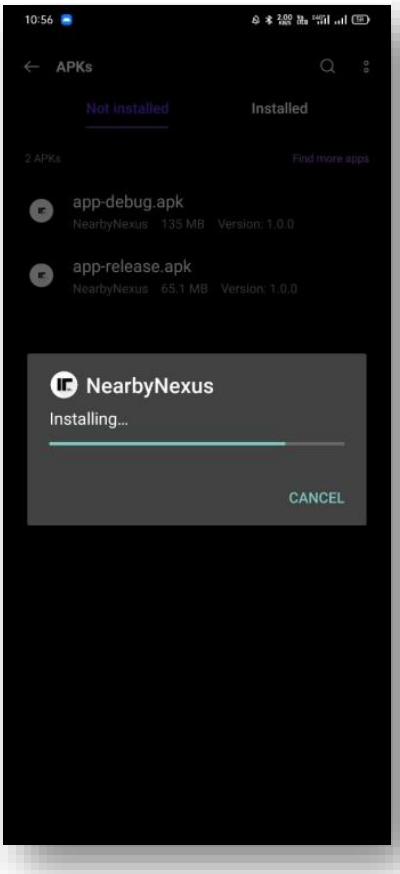
- **Quick Installation:** APKs often install swiftly, bypassing the need for extensive downloads from official app stores.
- **Independence from App Stores:** APKs offer the advantage of being downloadable from various sources, granting users flexibility in obtaining apps.
- **Beta and Unreleased Versions:** Users can access beta or unreleased versions of applications via APKs, providing a preview of upcoming features.
- **Offline Installation:** APKs can be shared and installed without an active internet connection, enhancing accessibility.

Screenshots

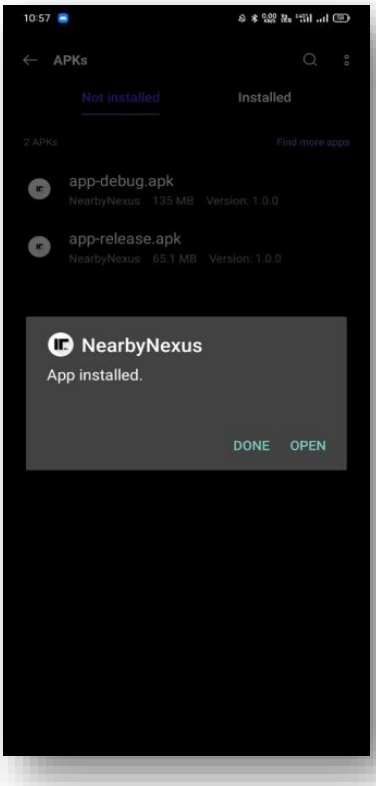
6.2.5.1 Installing



6.2.5.2 Installing



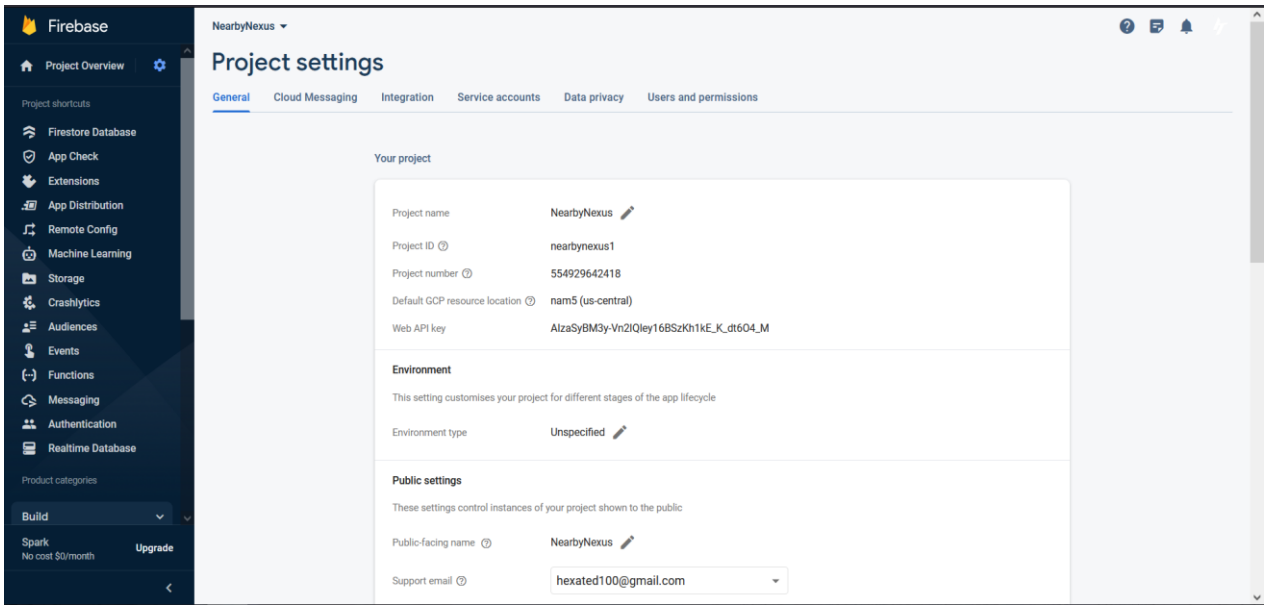
6.2.5.2 Installation complete



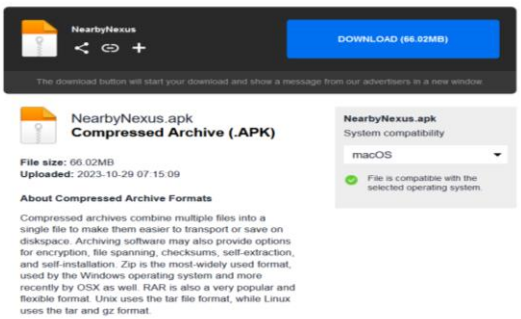
6.2.5 Hosting (Application)

Hosting within Firebase is the process of providing a platform or infrastructure for an application or website to be accessible over the internet. It involves allocating server space, configuring network settings, and ensuring optimal performance and reliability within the Firebase hosting environment. Hosting services in Firebase come with Firebase-specific features like Firebase Hosting deployment, security measures, scalability options, and technical support tailored to meet the diverse needs of applications and businesses within Firebase. Choosing the right hosting solution within Firebase is crucial for ensuring a seamless and accessible online presence integrated with Firebase services.

Firebase Cloud server



6.2.5.1 Server details



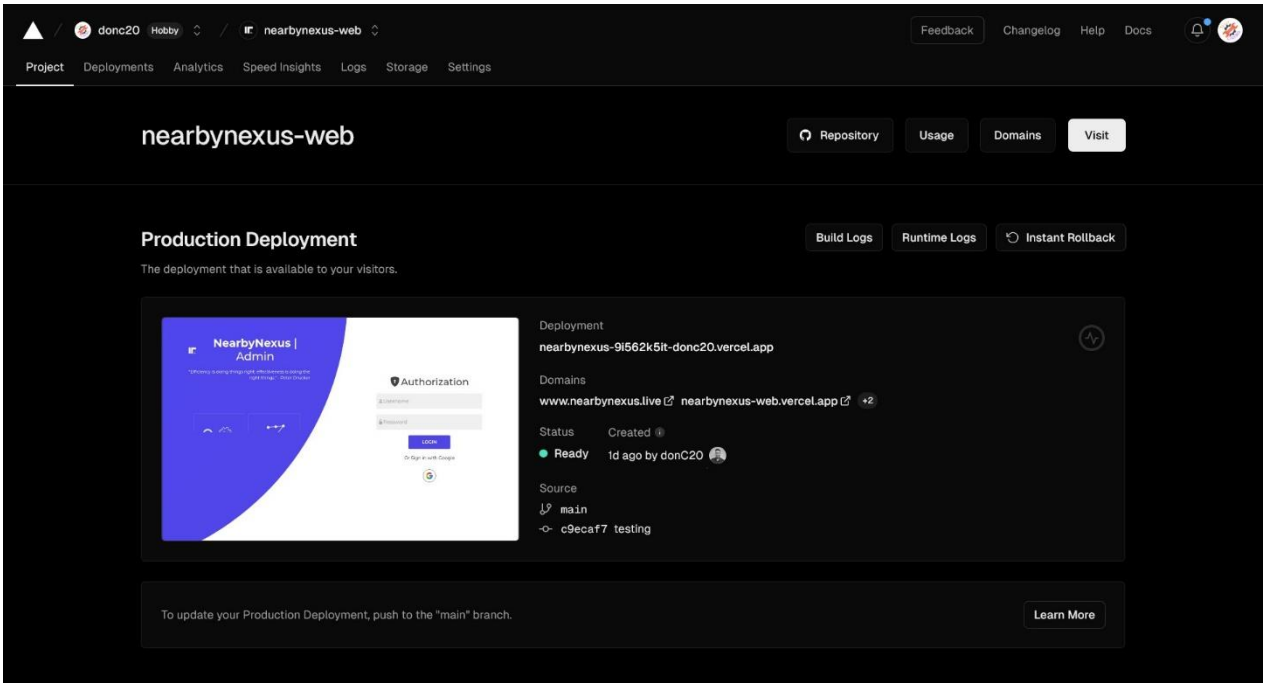
6.2.5.2 Share & Download application

Downloadable link https://www.mediafire.com/file/ohc2fogrhu8sxby/NearbyNexus.apk/file

6.2.6 Hosting (Web)

Hosting within Firebase is the process of providing a platform or infrastructure for a React-based website with a Firebase backend to be accessible over the internet. It involves allocating server space, configuring network settings, and ensuring optimal performance and reliability within the Firebase hosting environment. Hosting services in Firebase come with Firebase-specific features like Firebase Hosting deployment, security measures, scalability options, and technical support tailored to meet the diverse needs of applications and businesses using Firebase as the backend. However, in the case of website development using React with Firebase as the backend hosted on Vercel and following CI/CD practices, the hosting environment would be provided by Vercel. This involves deploying the React frontend alongside the Firebase backend, ensuring seamless integration, and utilizing Vercel's features for deployment, scaling, and continuous integration/continuous deployment (CI/CD) pipelines. Choosing the right hosting solution within Vercel is crucial for ensuring a seamless and accessible online presence integrated with Firebase services and leveraging Vercel's capabilities for front-end deployment and management.

Vercel deployment server



6.2.6.1 Deployment

Scannable QR code for the website

OR

https://www.nearbynexus.live/

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

In conclusion, this groundbreaking project marks a pivotal shift in the realm of location-based services, forging seamless connections between users and an expansive array of service providers. Its core mission lies in prioritizing essential services, bridging critical gaps for individuals navigating unfamiliar territories. The three-tiered user framework—comprising of Administrators, General Users, and Vendors establishes a vibrant, interdependent ecosystem.

Administrators wield the power of robust verification protocols, fortifying the platform's security infrastructure. Simultaneously, service providers empower users by furnishing them with a wealth of information to make informed choices. A state-of-the-art, secure, and cashless transaction system is in place, simplifying the payment process and ensuring hassle-free transactions.

The application's intuitively designed search and rating system augments the user experience, enabling swift and precise service discovery, while also facilitating invaluable feedback loops. Real-time location tracking stands as a cornerstone feature, fine-tuning the connection process by swiftly linking users with proximate service providers.

Furthermore, the project is poised to redefine the landscape of location-based services, offering a paradigm-shifting, user-centric, secure, and convenient solution for accessing a myriad of essential services in any given location. The integration of emerging technologies and a commitment to user satisfaction underscore its potential to not only meet but exceed expectations.

In essence, this project embodies a transformative force within the domain of location-based services, poised to reshape the way users interact with and access a diverse range of essential offerings. Its far-reaching implications extend beyond mere convenience, promising a revolutionized approach to service provision on a global scale.

7.2 FUTURE SCOPE

Future enhancements entail the features we may incorporate into our software product down the line, enhancing the overall user experience. The project holds significant potential for expansion in the future. Its inherent flexibility allows seamless integration of forthcoming modifications and expansions, facilitating timely updates to meet evolving demands.

The proposed location-based service application exhibits significant potential for future expansion and enhancement. Here are some potential avenues for future development:

Integration of Artificial Intelligence (AI) and Machine Learning (ML):

- Implementing AI and ML algorithms can enhance user experience by providing personalized service recommendations based on user preferences and behavior patterns

Expansion of Service Categories:

- Introducing additional service categories based on user feedback and evolving market trends can further diversify the offerings and meet a wider range of user needs.

Multi-Lingual Support:

- Including multi-lingual support will make the application accessible to a global audience, ensuring inclusivity and expanding its user base

Social Media Integration:

- Integrating with popular social media platforms can enable users to share their experiences and recommendations, fostering a sense of community within the application.

Voice and Chatbot Integration:

- Enabling voice assistants or chatbots can streamline user interactions.

CHAPTER 8

BIBLIOGRAPHY

REFERENCES:

- Gary B. Shelly, Harry J. Rosenblatt, “System Analysis and Design”, 2009.
- Roger S Pressman, “Software Engineering”, 1994.
- PankajJalote, “Software engineering: a precise approach”, 2006.
- James lee and Brent ware Addison, “Open source web development with LAMP”, 2003
- IEEE Std 1016 Recommended Practice for Software Design Descriptions.

WEBSITES:

- https://pub.dev/
- https://console.firebase.google.com
- https://docs.flutter.dev/development/ui/widgets
- https://fluttergems.dev/
- https://medium.com/
- https://www.npmjs.com/

CHAPTER 9

APPENDIX

9.1 Sample Code

9.1.1 dashboard.jsx

```
import 'dart:convert';
import React, { useEffect, useState } from 'react';
import style from '../css/dashboard.module.css';
import { getAllDataOnCondition } from '../components/Apifunction';
import ServicesUsage from '../components/dasboardComponents/servicesUsage';
import RevenueMatrix from '../components/dasboardComponents/revenueMetrix';
const Dashboard = () => {
   const [userCount, setUserCount] = useState(0);
   const [userOnlineCount, setUserOnlineCount] = useState(0);
   const [generalUserCount, setgeneralUserCount] = useState(0);
   useEffect(() => {
      // get user count
      getAllDataOnCondition('users', [
         { field: 'userType', operator: '!=', value: 'admin' },
         { field: 'status', operator: '==', value: 'active' }
      ], (data) => {
         setUserCount(data.length);
      });

      // get online count
      getAllDataOnCondition('users', [
         { field: 'userType', operator: '!=', value: 'admin' },
         { field: 'status', operator: '==', value: 'active' },
         { field: 'online', operator: '==', value: true }
      ], (data) => {
         setUserOnlineCount(data.length);
      });

      // get general user  count
      getAllDataOnCondition('users', [
         { field: 'userType', operator: '==', value: 'general_user' },
      ], (data) => {
         setgeneralUserCount(data.length);
      });
   }, []);

   return (
      <div className={style.dashboard}>
         <div className="container p-4">
            <div className="d-flex flex-column justify-content-start">
               <h2 className={style.topHeading}>Dashboard</h2>
            </div>
            <br />
            {/* cards */}
```

```
        <div className={`row gap-4 ${style.staticsContainer}`}>
          {/* card 1 */}
          <div className={`col-lg-3 col-md-6 col-sm-12 col-12 shadow ${style.cookieCard} ${style.cardOne}`}>
            <div>
              <div className="d-flex align-items-center">
                <i className="bi bi-people-fill text-light fs-3 me-2"></i>
                <span className='fw-bold fs-4 text-light'>{userCount}</span>
              </div>
              <p className={style.cookieHeading}>Users</p>
            </div>
            <p className={style.cookieDescription}>
              <i className="bi bi-info-circle me-1 text-light"></i>
              Displays the total user count.
            </p>
          </div>
          {/* card 2 */}
          <div className={`col-lg-3 col-md-6 col-sm-12 shadow ${style.cookieCard} ${style.cardTwo}`}>
            <div>
              <div className="d-flex align-items-center">
                <i className="bi bi-broadcast-pin text-light fs-3 me-2"></i>
                <span className='fw-bold fs-4 text-light'>{userOnlineCount}</span>
              </div>
              <p className={style.cookieHeading}>Online Users</p>
            </div>
            <p className={style.cookieDescription}>
              <i className="bi bi-info-circle me-1 text-light"></i>
              Displays the total online user count.
            </p>
          </div>
          {/* card 3 */}
          <div className={`col-lg-3 col-md-6 col-sm-12 shadow ${style.cookieCard} ${style.cardThree}`}>
            <div>
              <div className="d-flex align-items-center">
                <i className="bi bi-backpack2-fill text-light fs-3 me-2"></i>
                <span className='fw-bold fs-4 text-light'>{generalUserCount}</span>
              </div>
              <p className={style.cookieHeading}>General Users</p>
            </div>
            <p className={style.cookieDescription}>
              <i className="bi bi-info-circle me-1 text-light"></i>
              Displays the total count of users who are general to the application.
            </p>
          </div>
          {/* card 4 */}
          <div className={`col-lg-3 col-md-6 col-sm-12 shadow ${style.cookieCard} ${style.card}`}>
```

```
                <div>
                  <div className="d-flex align-items-center">
                    <i className="bi bi-headset text-light fs-3 me-2"></i>
                    <span className='fw-bold fs-4 text-light'>{userCount - generalUserCount}</span>
                  </div>
                  <p className={style.cookieHeading}>Service providers</p>
                </div>
                <p className={style.cookieDescription}>
                  <i className="bi bi-info-circle me-1 text-light"></i>
                  Displays the total count of users who are provides services to the users.
                </p>
              </div>
            </div>
            {/* cards ends */}
            <div className="row gap-3 mt-4">
              {/* services cards */}
              <RevenueMatrix />
              <ServicesUsage />
            </div>
          </div>
        </div>
    );
}
export default Dashboard;
```

9.1.2 revenueMatrix.jsx

```
import React, { useEffect, useState } from 'react';
import { AreaChart, Area, XAxis, YAxis, CartesianGrid, Tooltip, ResponsiveContainer } from 'recharts';
import style from '../../css/dashboard.module.css';
import { formatDate, getAllDataOnCondition } from '../Apifunction';
import CommonLoader from '../commonLoader';
import { utils, writeFile } from 'xlsx';

const RevenueMatrix = () => {
  const [selectedMonth, setSelectedMonth] = useState(new Date().getMonth() + 1);
  const [selectedYear, setSelectedYear] = useState(new Date().getFullYear());
  const [totalRevenue, setTotalRevenue] = useState(0);
  const [convertedData, setConvertedData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [printableData, setprintableData] = useState([]);

  useEffect(() => {
    const fetchData = async () => {
      const firstDayOfMonth = new Date(selectedYear, selectedMonth - 1, 1);
      const lastDayOfMonth = new Date(selectedYear, selectedMonth, 0);
      setLoading(true);
```

```
        try {
          getAllDataOnCondition('payments', [
            { field: 'paymentTime', operator: '>=', value: firstDayOfMonth },
            { field: 'paymentTime', operator: '<=', value: lastDayOfMonth }
          ], (data) => {
            // Convert Firebase Timestamps to Date objects and format data
            setprintableData(data)
            const newData = data.map(item => ({
              name: formatDate(item.paymentTime), // Convert Firebase Timestamp to formatted date string
              revenue: parseInt(item.applicationRevenue) // Convert commission to integer
            }));
            var totalAmount = 0;

            data.forEach((value, index) => {
              // Check if the value of 'commission' is a valid number
              if (!isNaN(value['applicationRevenue'])) {
                // If it's a valid number, parse and add it to the total
                totalAmount = totalAmount + parseFloat(value['applicationRevenue']);
              }
            });

            setTotalRevenue(totalAmount);
            setConvertedData(newData);
            setLoading(false);
          });

        } catch (error) {
          console.error('Error fetching data:', error);
          setConvertedData([]); // Reset data if there's an error
        }
      };

      fetchData();
    }, [selectedMonth, selectedYear]);

    const printData = async () => {
      console.log(printableData);
      const wb = utils.book_new();
      var dataToPrint = [];
      printableData.forEach((item) => {
        // Push each user object into the dataToPrint array
        dataToPrint.push({
          PaymentId: item.id,
          JobId: item.jobId.id ?? '',
          PayedFor: item.payedFor,
          Revenue: item.applicationRevenue,
          Payed: item.amountPaid,
```

```
            PayedTo: item.payedTo.id,
            PayedBy: item.payedBy.id,
            PayedOn: formatDate(item.paymentTime)
            // Add other fields as needed
          });
        });

        // Convert the dataToPrint array to a worksheet
        const ws = utils.json_to_sheet(dataToPrint);

        // Add the worksheet to the workbook
        utils.book_append_sheet(wb, ws, 'revenueData');

        // Write the workbook to a file
        // This example writes to a file named 'users.xlsx' in the current directory
        writeFile(wb, 'revenueData.xlsx');
    }

    return (
        <div className="col p-4 rounded bg-white shadow">
            <div className={`d-flex justify-content-between align-items-center ${style.amountValue}`}>
                <div className="title d-flex flex-column">
                    <h5>Revenue</h5>
                    <div className='d-flex gap-2'>
                        <select name="month" id={style.monthSelector} value={selectedMonth} onChange={(e) => setSelectedMonth(parseInt(e.target.value))}>
                            {Array.from({ length: 12 }, (_, index) => index + 1).map((month) => (
                                <option key={month} value={month}>{new Date(selectedYear, month - 1).toLocaleString('default', { month: 'long' })}</option>
                            ))}
                        </select>
                        <select name="year" id={style.yearSelector} value={selectedYear} onChange={(e) => setSelectedYear(parseInt(e.target.value))}>
                            {Array.from({ length: new Date().getFullYear() - 2021 }, (_, index) => 2022 + index).map((year) => (
                                <option key={year} value={year}>{year}</option>
                            ))}
                        </select>
                        {convertedData.length > 0 ?
                            <button className='p-1 rounded btn btn-transparent border' onClick={printData}><small className='d-flex gap-2 '><i className="bi bi-printer-fill"></i>Export</small></button>
                            : <></>}
                    </div>
                </div>
                <div>
                    <span><i className="bi bi-currency-rupee"></i>{totalRevenue}</span>
                </div>
            </div>
```

```
        <br />
        {loading ? <div className='d-flex justify-content-center align-items-center h-25'>
          <CommonLoader />
        </div> :
          convertedData.length === 0 ?
            <div className='d-flex justify-content-center align-items-center h-100'>
              <b>No revenue generated so far!</b>
            </div> :
            <ResponsiveContainer width="100%" className={style.revenueMetrixChart}>
              <AreaChart data={convertedData} margin={{ top: 10, right: 30, left: 0, bottom: 0 }}>
                <CartesianGrid strokeDasharray="3 3" />
                <XAxis dataKey="name" />
                <YAxis />
                <Tooltip />
                <Area type="monotone" dataKey="revenue" strokeWidth={3} stroke="#5046e5"
fill="#EDECFC" />
              </AreaChart>
            </ResponsiveContainer>
        }
      </div>
    );
};

export default RevenueMatrix;
```

9.1.3 chat_screen.dart

```
import 'dart:io';
import 'package:NearbyNexus/components/message_card.dart';
import 'package:NearbyNexus/components/my_date_util.dart';
import 'package:NearbyNexus/functions/api_functions.dart';
import 'package:NearbyNexus/misc/colors.dart';
import 'package:NearbyNexus/models/message.dart';
import 'package:NearbyNexus/screens/vendor/functions/vendor_common_functions.dart';
import 'package:NearbyNexus/screens/vendor/screens/subscription_screen.dart';
import 'package:cached_network_image/cached_network_image.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:emoji_picker_flutter/emoji_picker_flutter.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_svg/svg.dart';
import 'package:getwidget/getwidget.dart';
import 'package:image_picker/image_picker.dart';
import 'package:logger/logger.dart';
class ChatScreen extends StatefulWidget {
  final String userId;
  const ChatScreen({Key? key, required this.userId}) : super(key: key);
```

```
 @override
 _ChatScreenState createState() => _ChatScreenState();
}
class _ChatScreenState extends State<ChatScreen> {
 late Size mq;
 var logger = Logger();
 List<Message> _list = [];
 final _textController = TextEditingController();
 bool _showEmoji = false, _isUploading = false, _isSending = false;
 Future<void> deleteCollection(String path) async {
  var collectionRef = FirebaseFirestore.instance.collection(path);
  var batchSize = 10;
  await collectionRef.get().then((querySnapshot) async {
   if (querySnapshot.size == 0) return;
   var batch = FirebaseFirestore.instance.batch();
   querySnapshot.docs.forEach((document) {
    batch.delete(document.reference);
   });
   await batch.commit();
   // Recursively call deleteCollection until the collection is empty
   await deleteCollection(path);
  });
 }
 Future<bool> _onBackPressed() async {
  // back pressed button event
  return (await showDialog(
      context: context,
      builder: (_) => AlertDialog(
       content: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
         Column(
          children: [
           SvgPicture.asset(
            'assets/icons/svg/crown-svgrepo-com.svg',
            height: 50,
            width: 50,
           ),
           SizedBox(height: 10),
           Text(
            'Upgrade to continue',
            style: TextStyle(
             fontSize: 18,
             fontWeight: FontWeight.bold,
            ),
           ),
           SizedBox(height: 15),
           Text(
```

```
                'Your messages will be cleared after viewing, Upgrade to keep the message history.',
                textAlign: TextAlign.justify,
              ),
              SizedBox(height: 15),
              GFButton(
                onPressed: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => SubscriptionScreen(),
                    ),
                  );
                },
                text: "Upgrade",
                textColor: Colors.black,
                color: Colors.amberAccent,
                size: GFSize.LARGE,
                shape: GFButtonShape.pills,
                fullWidthButton: true,
              ),
              SizedBox(
                height: 10,
              ),
              GFButton(
                onPressed: () async {
                  await deleteCollection(
                          'chats/${ApiFunctions.getConversationID(widget.userId)}/messages')
                      .then((value) => Navigator.of(context).pop(true));
//                  FirebaseFirestore.instance
//                      .collection('chats')
//                      .doc()
//                      .delete();
//                  DocumentReference documentReference = FirebaseFirestore
//                      .instance
//                      .collection('chats')
//                      .doc(ApiFunctions.getConversationID(widget.userId));

// // Call the delete method to remove the document
//                  documentReference.delete().then((_) {
//                    Navigator.of(context).pop(true);

//                    print('Document successfully deleted');
//                  }).catchError((error) {
//                    print('Error deleting document: $error');
//                  });
                },
                text: "Clear",
                textColor: Colors.white,
```

```
                color: Colors.red,
                size: GFSize.LARGE,
                shape: GFButtonShape.pills,
                fullWidthButton: true,
              )
            ],
          ),
        ],
      ),
    ),
  )) ??
  false;
}
@override
Widget build(BuildContext context) {
 return WillPopScope(
  onWillPop: _onBackPressed,
  child: Scaffold(
   backgroundColor: Theme.of(context).colorScheme.background,
   appBar: AppBar(
    elevation: 1,
    automaticallyImplyLeading: false,
    flexibleSpace: SafeArea(child: _appBar()),
   ),
   body: Builder(
    builder: (BuildContext context) {
     mq = MediaQuery.of(context).size;
     return Column(
      children: [
       Expanded(
        child: StreamBuilder(
         stream: ApiFunctions.getAllMessages(widget.userId),
         builder: (context, snapshot) {
          switch (snapshot.connectionState) {
           case ConnectionState.waiting:
            return SizedBox();
           case ConnectionState.none:
            return Center(
             child: CircularProgressIndicator(),
            );
           case ConnectionState.active:
           case ConnectionState.done:
            final data = snapshot.data?.docs;
            logger.e(data);
            _list = data
                  ?.map((e) => Message.fromJson(e.data()))
                  .toList() ??
              [];
```

```
            if (_list.isNotEmpty) {
              return ListView.builder(
                reverse: true,
                itemCount: _list.length,
                padding: EdgeInsets.only(top: 10),
                physics: const BouncingScrollPhysics(),
                itemBuilder: (context, index) {
                  return MessageCard(
                    message: _list[index],
                    isSent: _isSending,
                  );
                },
              );
            } else {
              return const Center(
                child: Text(
                  'Say Hii! 👋',
                  style: TextStyle(
                    color: Colors.black,
                    fontSize: 20,
                  ),
                ),
              );
            }
          }
        },
      ),
    ),
    if (_isUploading)
      const Align(
        alignment: Alignment.centerRight,
        child: Padding(
          padding: EdgeInsets.symmetric(
            vertical: 8,
            horizontal: 20,
          ),
          child: CircularProgressIndicator(strokeWidth: 2),
        ),
      ),
    _chatInput(),
    if (_showEmoji)
      SizedBox(
        height: mq.height * .35,
        child: EmojiPicker(
          textEditingController: _textController,
          config: Config(
            height: 256,
            checkPlatformCompatibility: true,
```

```
                emojiViewConfig: EmojiViewConfig(
                  emojiSizeMax: 32 * (Platform.isIOS ? 1.30 : 1.0),
                ),
              ),
            ),
          ),
        ),
      ],
    );
  },
 ),
),
);
}

Widget _appBar() {
 return StreamBuilder<Map<String, dynamic>>(
  stream: VendorCommonFn().streamUserData(
   uidParam:
     FirebaseFirestore.instance.collection('users').doc(widget.userId),
  ),
  builder: (context, snapshot) {
   if (snapshot.connectionState == ConnectionState.waiting) {
    return SizedBox();
   } else if (snapshot.hasError) {
    return Center(
     child: Text('Error: ${snapshot.error}'),
    );
   } else {
    Map<String, dynamic>? userData = snapshot.data;
    String userName = userData?['name'] ?? 'Unknown';
    String img =
      userData?['image'] ?? "https://via.placeholder.com/350x150";
    bool isOnline = userData?['online'] ?? false;
    final last_seen = userData?['last_seen'];

    return InkWell(
     onTap: () {},
     child: Row(
      children: [
       // IconButton(
       //   onPressed: () => Navigator.pop(context),
       //   icon: Icon(
       //    CupertinoIcons.back,
       //   ),
       // ),
       SizedBox(
        width: 20,
       ),
```

```
            ClipRRect(
              borderRadius: BorderRadius.circular(100),
              child: CachedNetworkImage(
                width: 40,
                height: 40,
                fit: BoxFit.cover,
                imageUrl: img,
                errorWidget: (context, url, error) =>
                    CircleAvatar(child: Icon(CupertinoIcons.person)),
              ),
            ),
            const SizedBox(width: 10),
            Column(
              mainAxisAlignment: MainAxisAlignment.center,
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  userName,
                  style: TextStyle(
                    fontSize: 16,
                    color: Theme.of(context).colorScheme.onSecondary,
                    fontWeight: FontWeight.w500,
                  ),
                ),
                Text(
                  isOnline
                      ? 'Online'
                      : MyDateUtil.getLastActiveTime(
                          context: context, lastActive: last_seen),
                  style: TextStyle(
                    fontSize: 13,
                    color: isOnline
                        ? Colors.green
                        : Color.fromRGBO(220, 220, 220, 1),
                  ),
                ),
              ],
            )
          ],
        ),
      );
    }
  },
 );
}

// bottom chat input field
 Widget _chatInput() {
```

```
return Padding(
  padding: EdgeInsets.symmetric(vertical: 10, horizontal: 10),
  child: Row(
    children: [
      //input field & buttons
      Expanded(
        child: Card(
          color: KColors.backgroundDark,
          shape: RoundedRectangleBorder(
              side: BorderSide(
                  color: const Color.fromARGB(46, 255, 255, 255)),
              borderRadius: BorderRadius.circular(100)),
          child: Row(
            children: [
              //emoji button
              IconButton(
                  onPressed: () {
                    FocusScope.of(context).unfocus();
                    setState(() => _showEmoji = !_showEmoji);
                  },
                  icon: const Icon(Icons.emoji_emotions,
                      color: Color.fromARGB(255, 255, 255, 255), size: 25)),

              Expanded(
                  child: TextField(
                controller: _textController,
                keyboardType: TextInputType.multiline,
                style: TextStyle(
                    color: Color.fromARGB(255, 205, 205, 205),
                    fontSize: 14),
                maxLines: null,
                onTap: () {
                  if (_showEmoji) setState(() => _showEmoji = !_showEmoji);
                },
                decoration: const InputDecoration(
                    hintText: 'Type Something...',
                    hintStyle: TextStyle(
                        color: Color.fromARGB(255, 205, 205, 205),
                        fontSize: 14),
                    border: InputBorder.none),
              )),

              //pick image from gallery button
              IconButton(
                  onPressed: () async {
                    final ImagePicker picker = ImagePicker();

                    // Picking multiple images
```

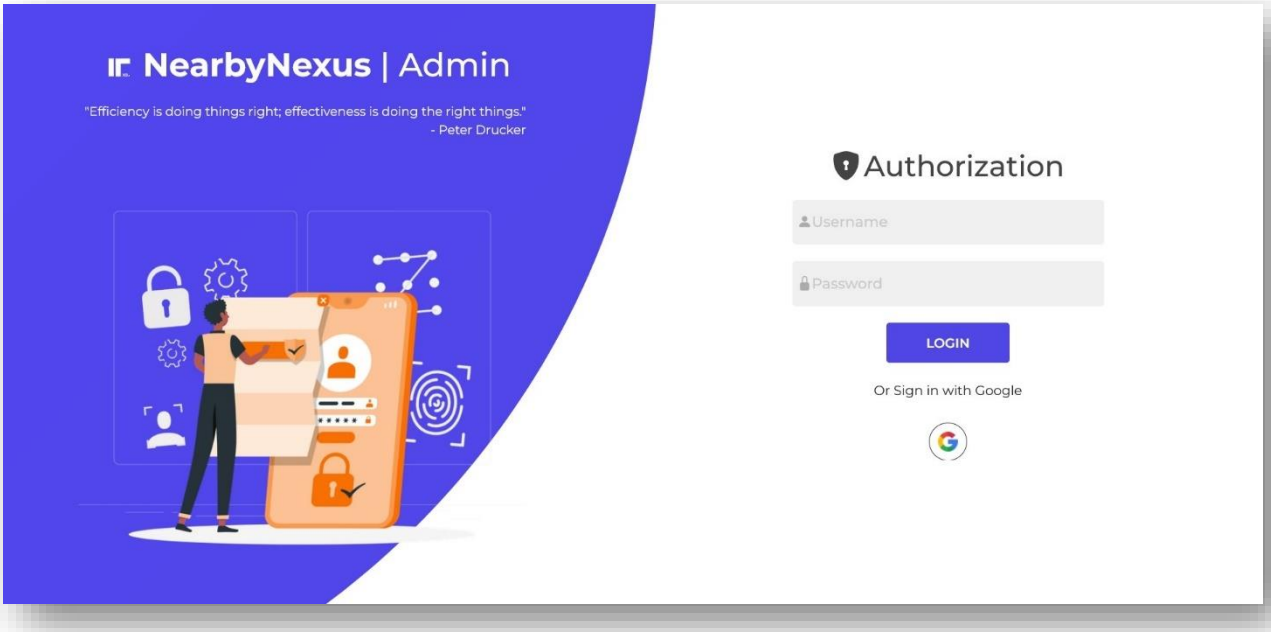
```
            final List<XFile> images =
              await picker.pickMultiImage(imageQuality: 70);

            // uploading & sending image one by one
            for (var i in images) {
              // log('Image Path: ${i.path}');
              setState(() => _isUploading = true);
              Map<String, dynamic>? recepientData =
                  await VendorCommonFn().fetchParticularDocument(
                      'users', widget.userId);
              await ApiFunctions.sendChatImage(
                  recepientData, widget.userId, File(i.path));
              setState(() => _isUploading = false);
            }
          },
          icon: const Icon(Icons.image,
              color: Color.fromARGB(255, 255, 255, 255), size: 26)),
      //take image from camera button
      IconButton(
          onPressed: () async {
            final ImagePicker picker = ImagePicker();

            // Pick an image
            final XFile? image = await picker.pickImage(
                source: ImageSource.camera, imageQuality: 70);
            if (image != null) {
              // log('Image Path: ${image.path}');
              setState(() => _isUploading = true);
              // recipent data
              Map<String, dynamic>? recepientData =
                  await VendorCommonFn().fetchParticularDocument(
                      'users', widget.userId);
              await ApiFunctions.sendChatImage(
                  recepientData, widget.userId, File(image.path));
              setState(() => _isUploading = false);
            }
          },
          icon: const Icon(Icons.camera_alt_rounded,
              color: Color.fromARGB(255, 255, 255, 255), size: 26)),

      //adding some space
      SizedBox(width: 10),
    ],
   ),
  ),
 ),
//send message button
MaterialButton(
```

```
          onPressed: () async {
            if (_textController.text.isNotEmpty) {
              setState(() => _isSending = true);
              Map<String, dynamic>? recepientData = await VendorCommonFn()
                  .fetchParticularDocument('users', widget.userId);
              ApiFunctions.sendMessage(recepientData, widget.userId,
                  _textController.text, Type.text);
              _textController.text = '';
              setState(() => _isSending = false);
            }
          },
          minWidth: 0,
          padding: const EdgeInsets.all(10),
          shape: const CircleBorder(),
          color: KColors.primary,
          child: SvgPicture.asset(
            'assets/images/vector/send_icon.svg',
            color: Colors.white,
            height: 25,
            width: 25,
          ),
        )
      ],
    ),
  );
 }
 // end of classs
}
```

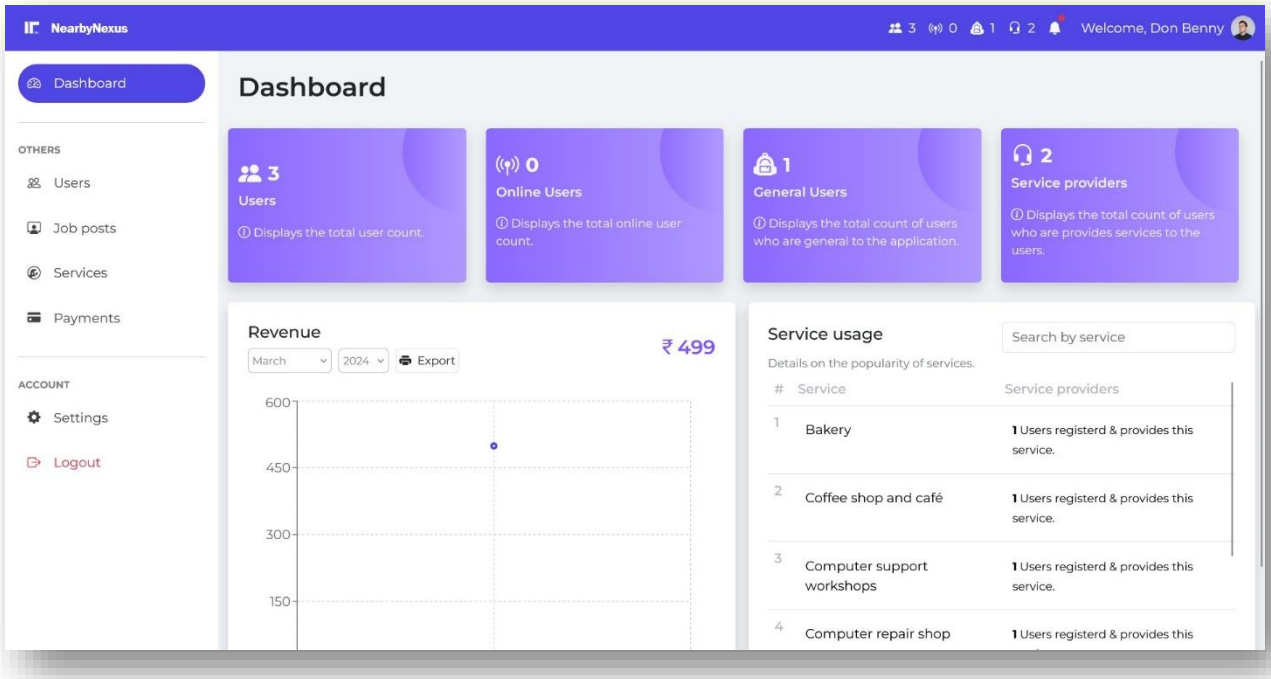
9.2 Screen Shots

Admin panel - web

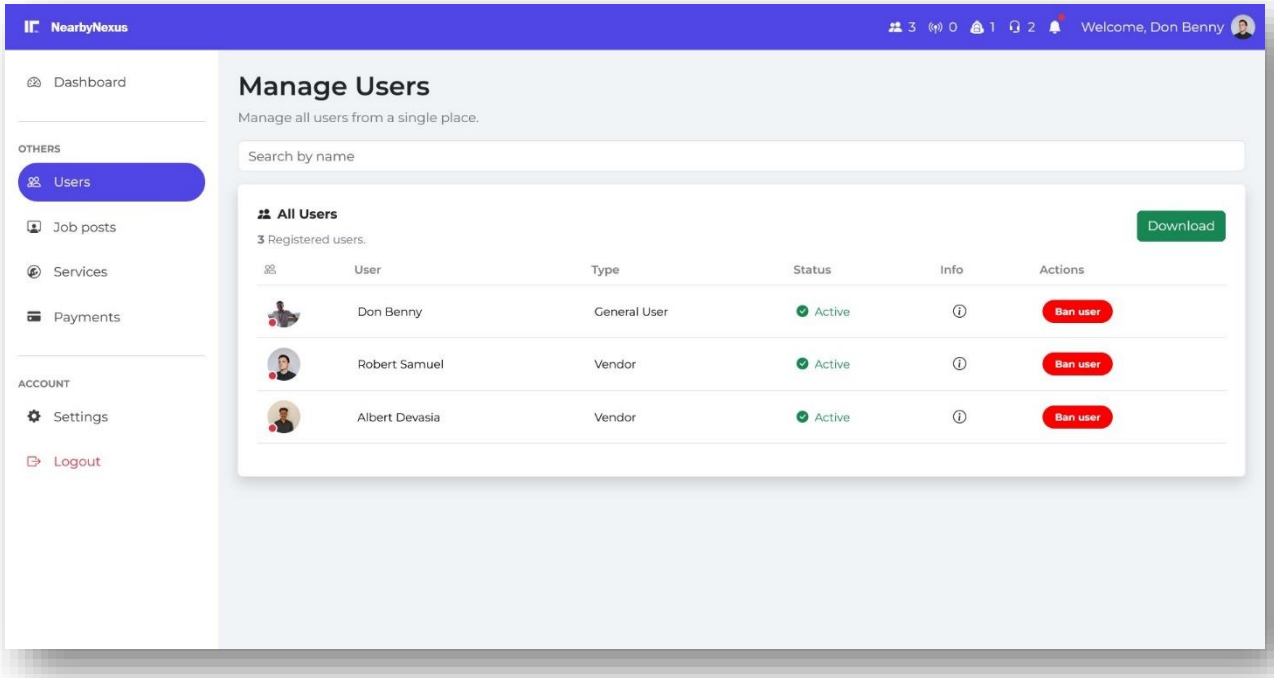
9.2.1 Login page

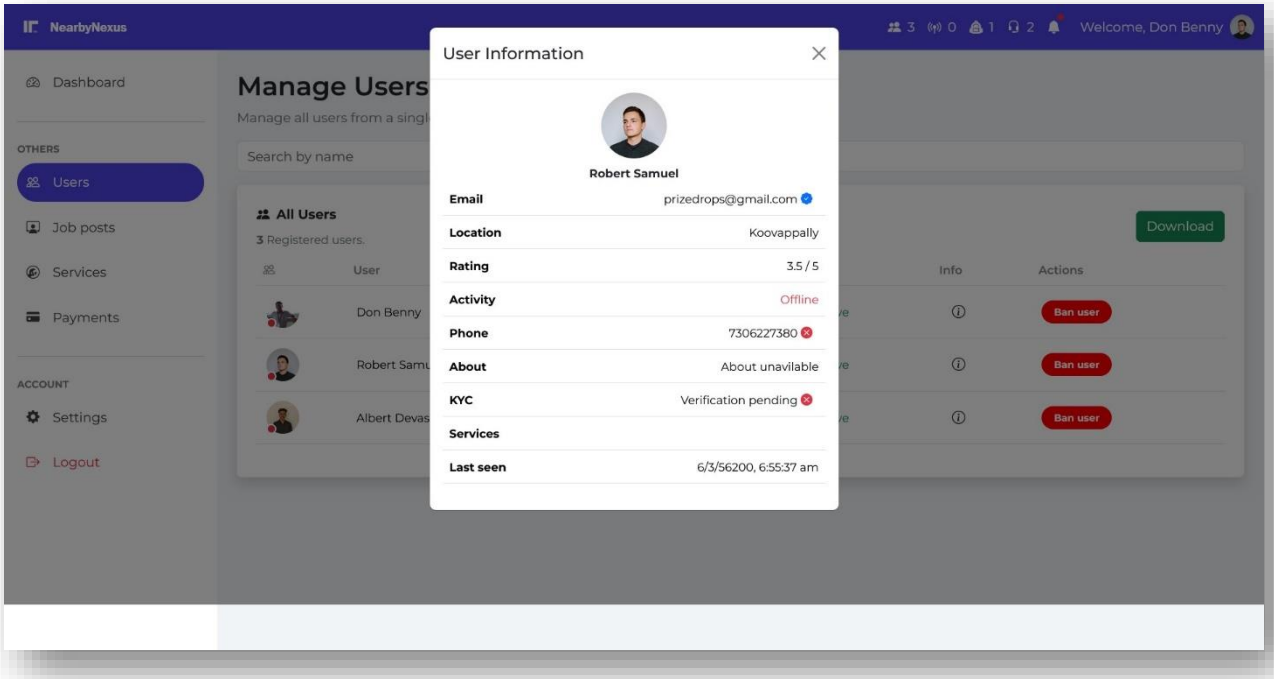


9.2.2 Dashboard

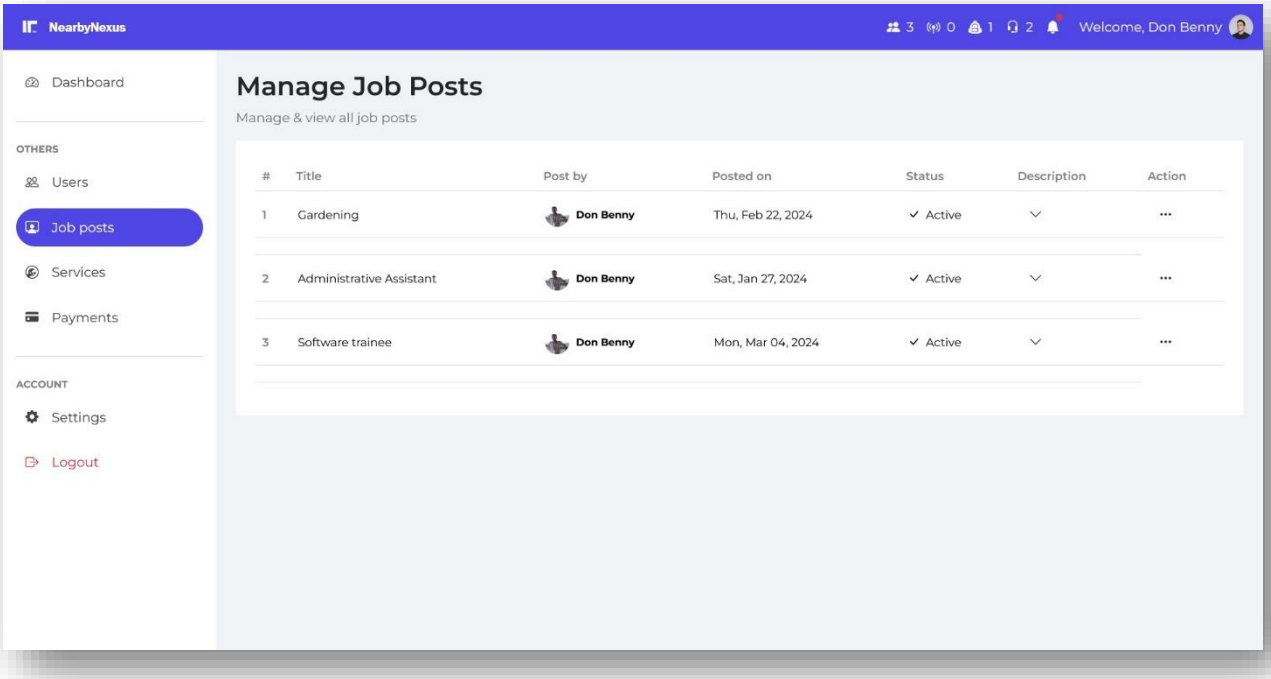


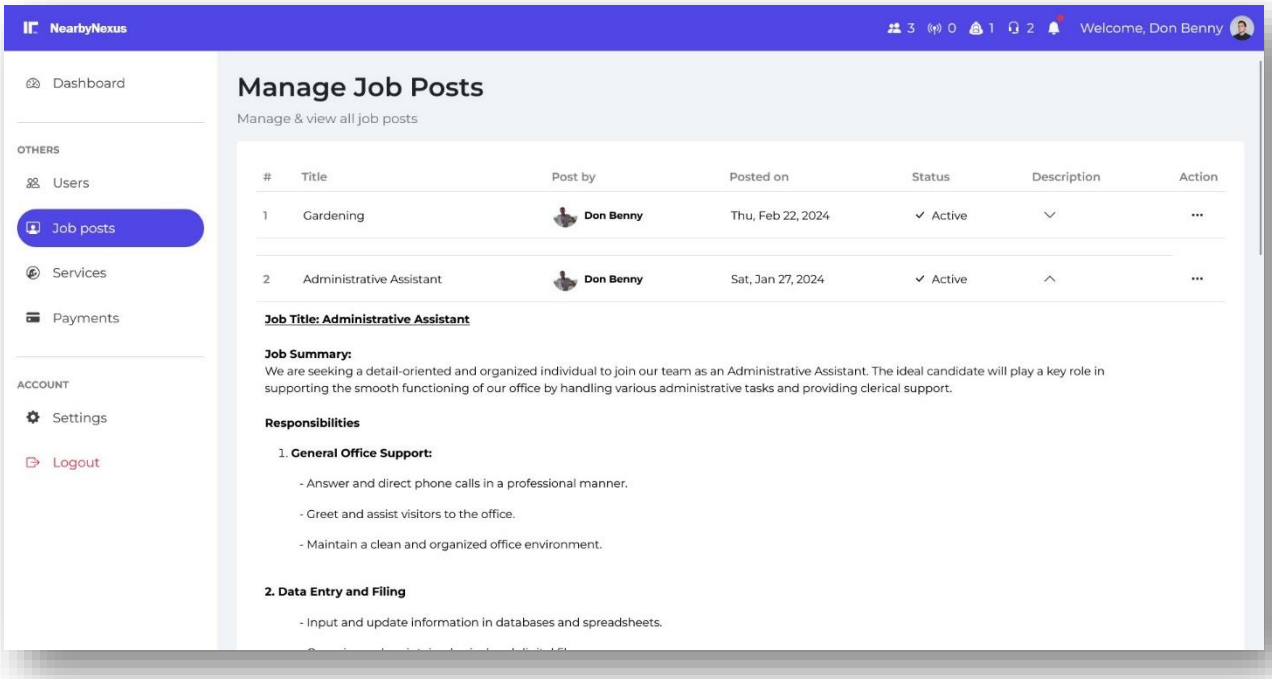
9.2.3 User management page



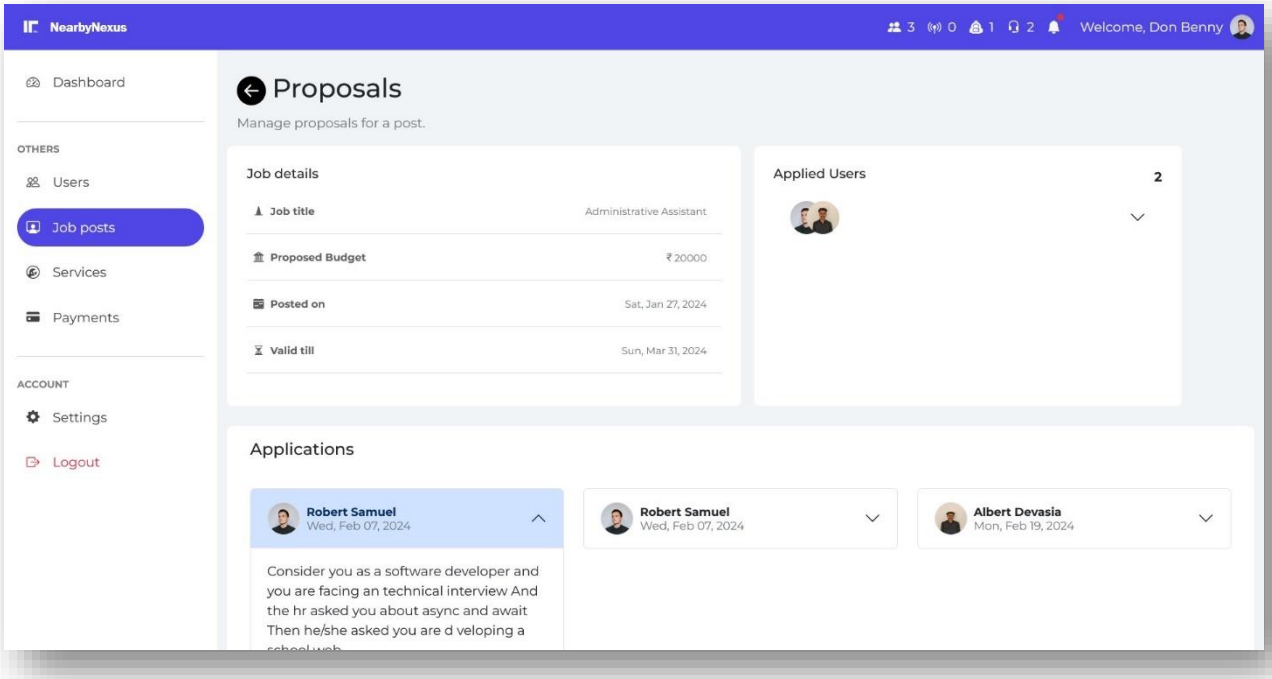


9.2.4 Jobs management page

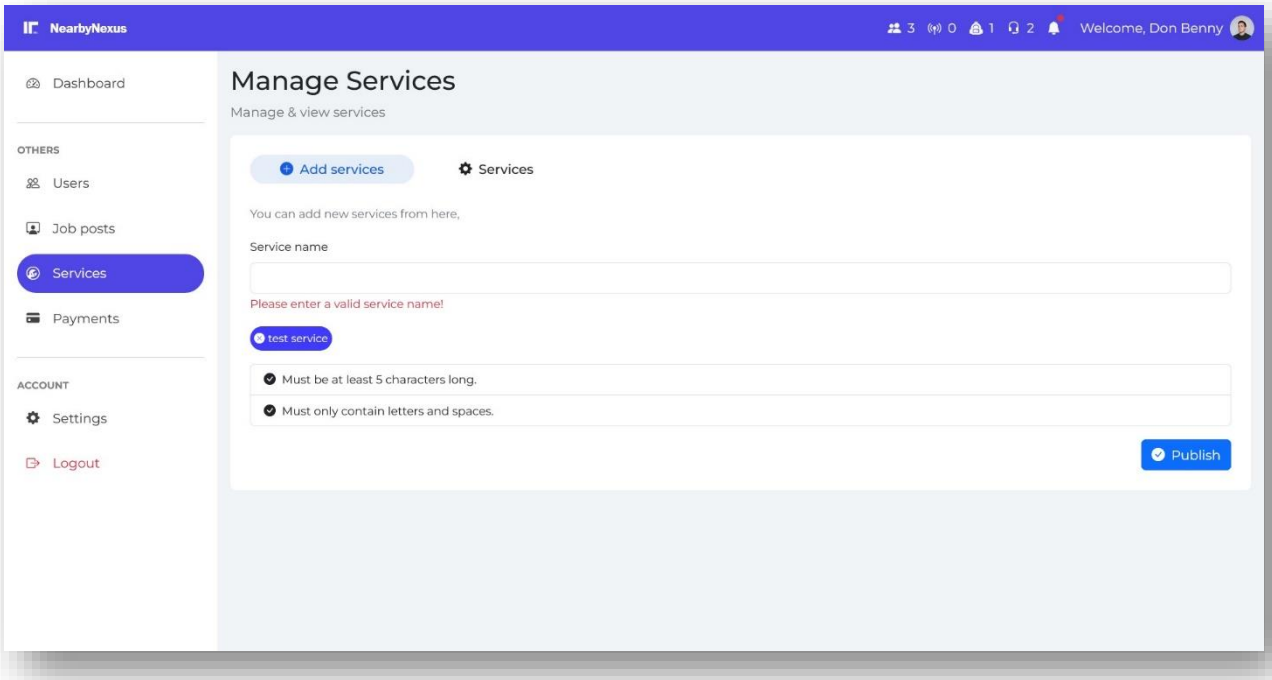




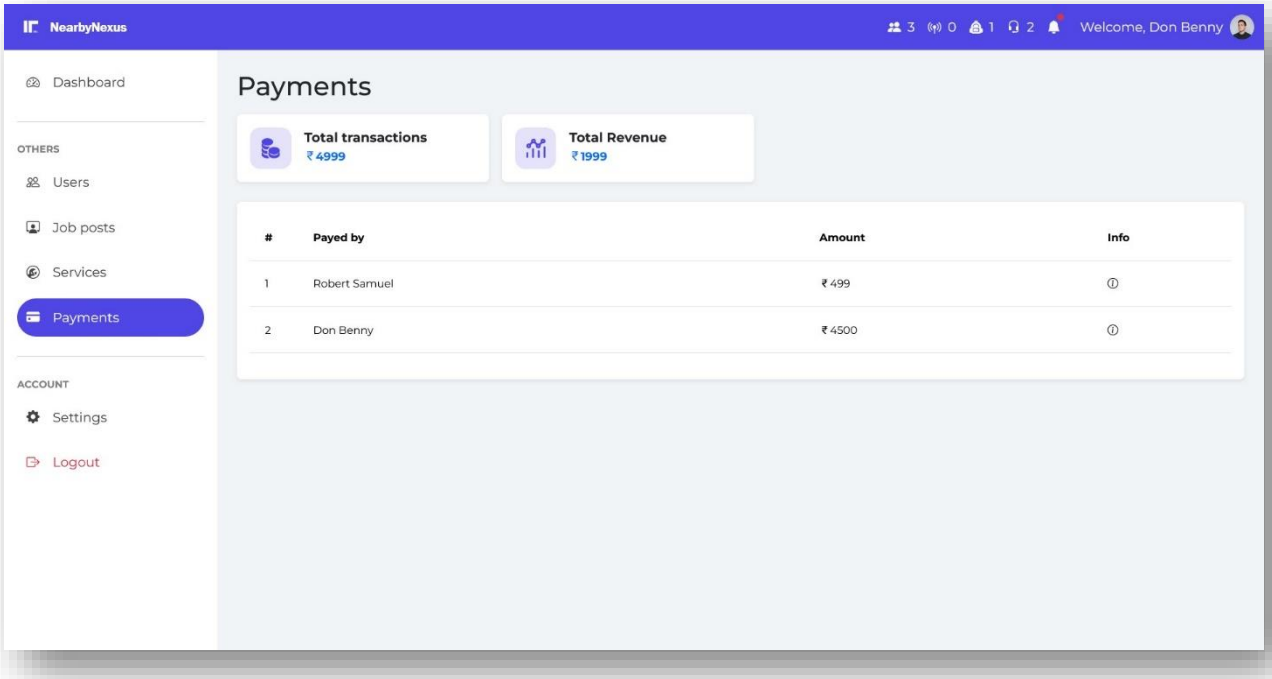
9.2.5 Proposal screen

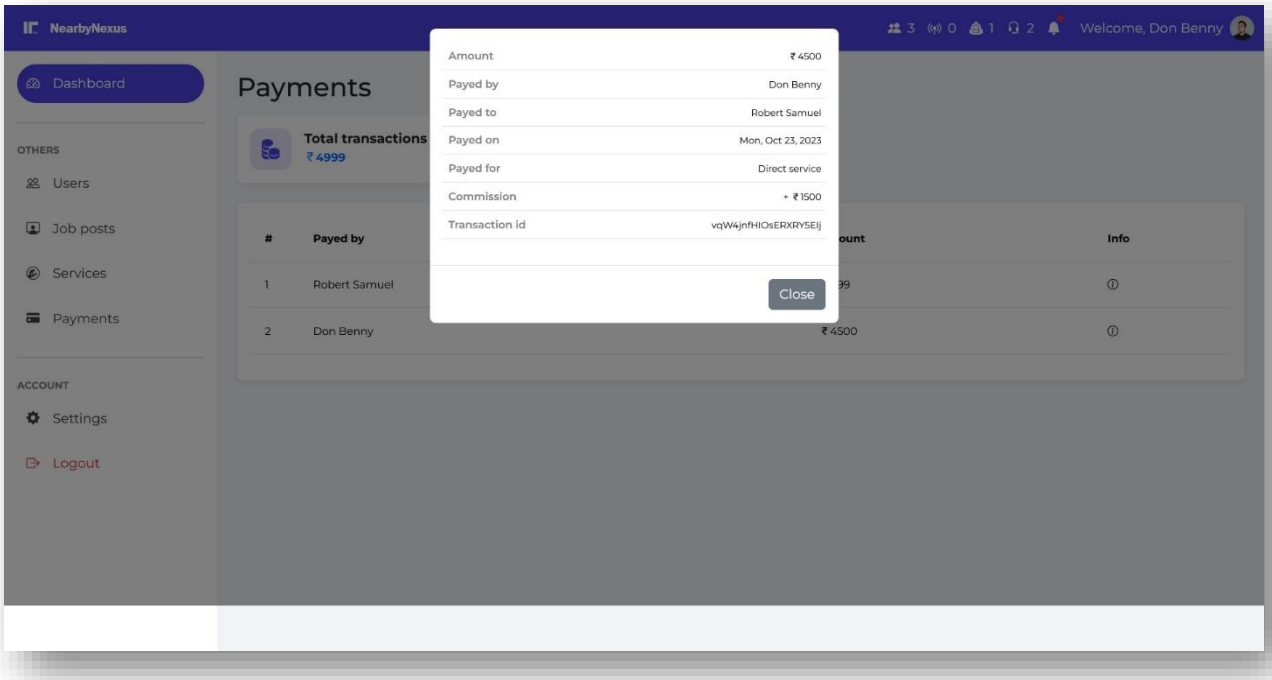


9.2.6 Manage services page



9.2.7 Payments page

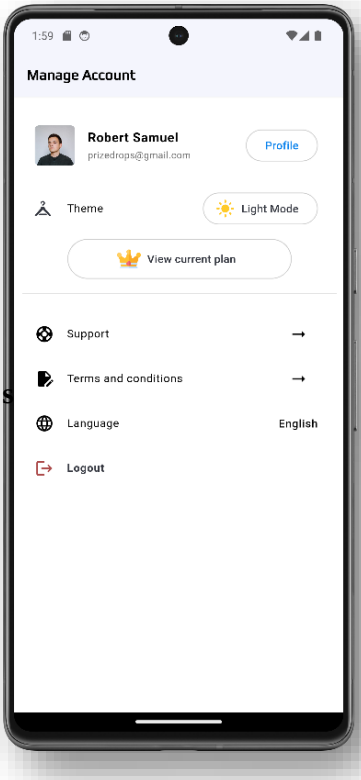




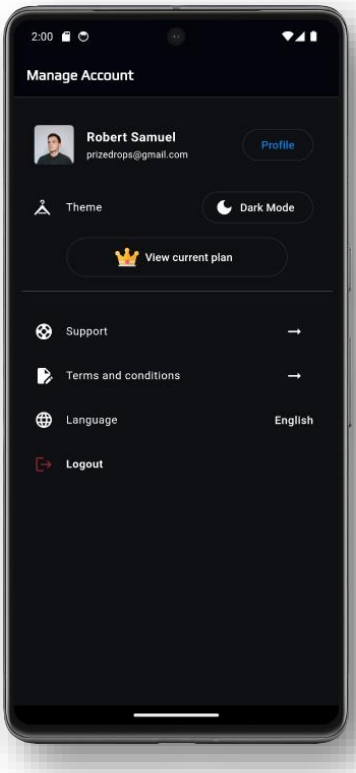
Application screens

- **Vendor screens**

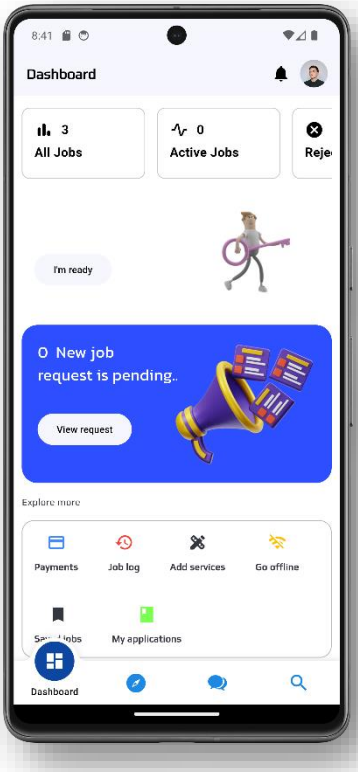
9.2.8 Light mode



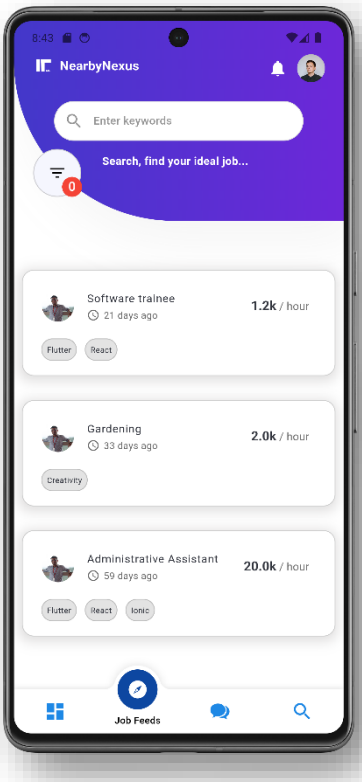
9.2.9 Dark mode



9.2.10 Dashboard



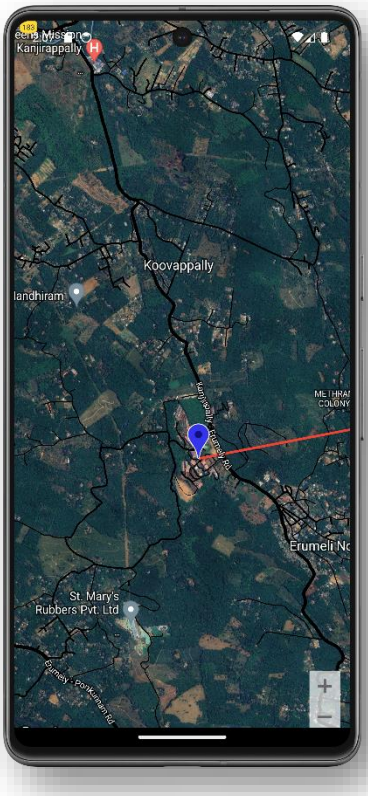
9.2.11 Job feeds



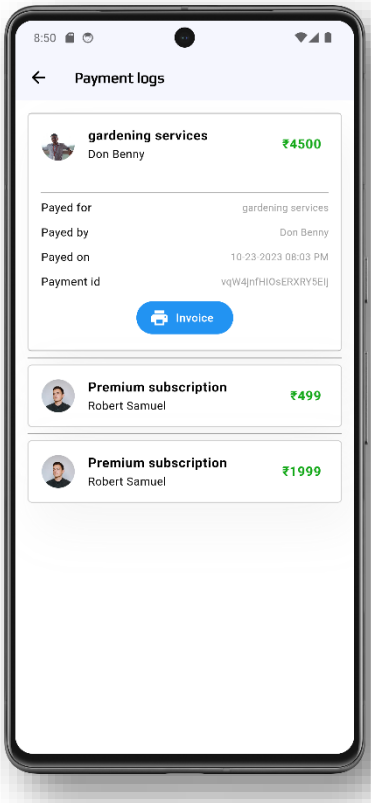
9.2.12 Inbox

9.2.13 Chat screen

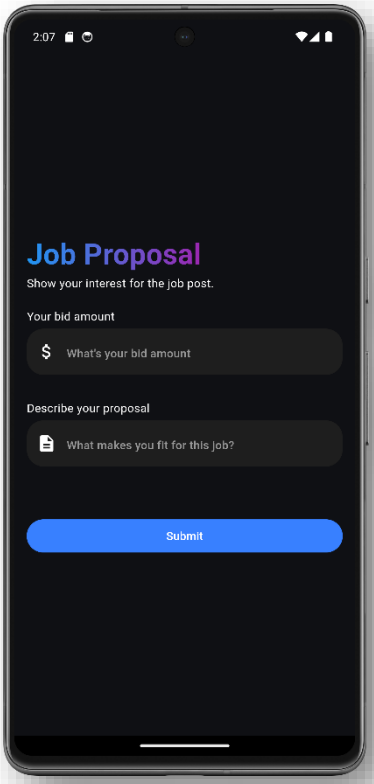
9.2.14 Map view



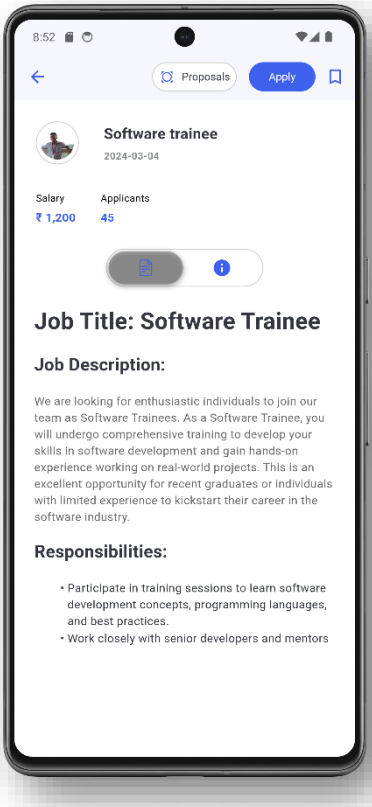
9.2.15 Payment's log



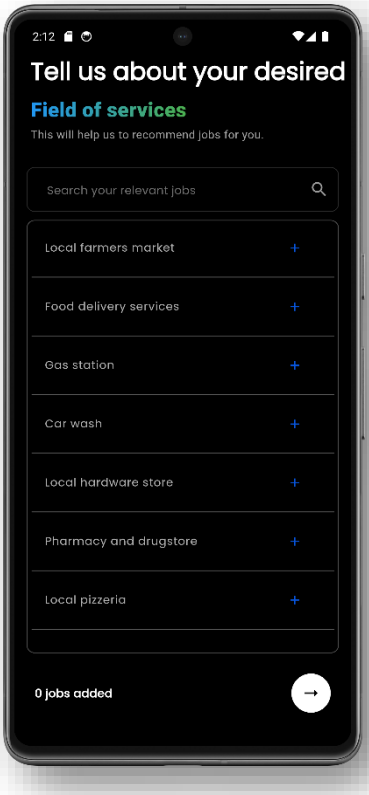
9.2.16 Add proposal.



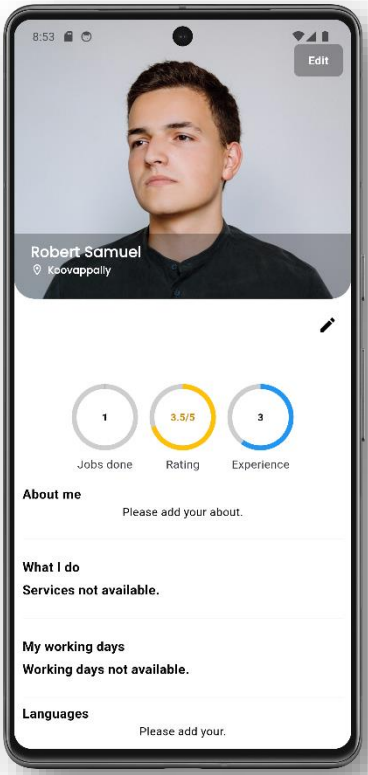
9.2.17 Job description



9.2.18 Add Service

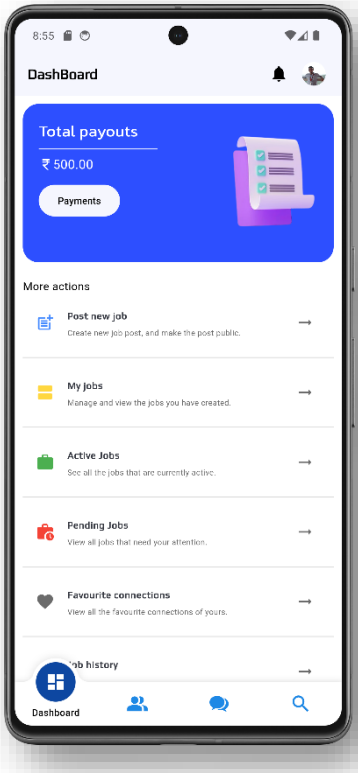


9.2.19 Portfolio

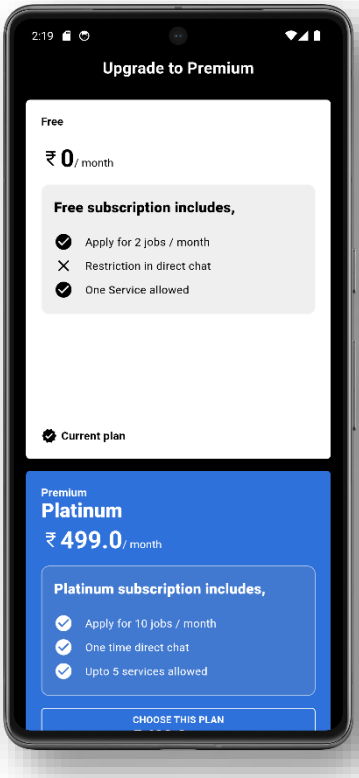


- **Users screens**

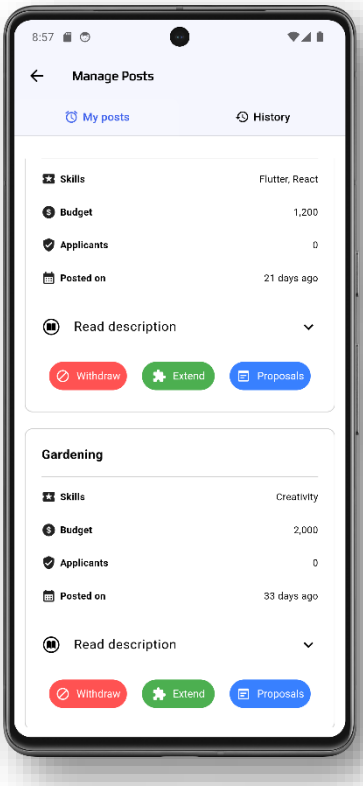
9.2.20 Dashboard



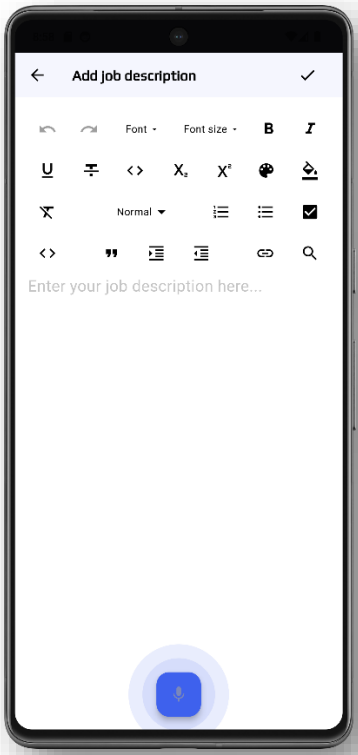
9.2.21 Subscription



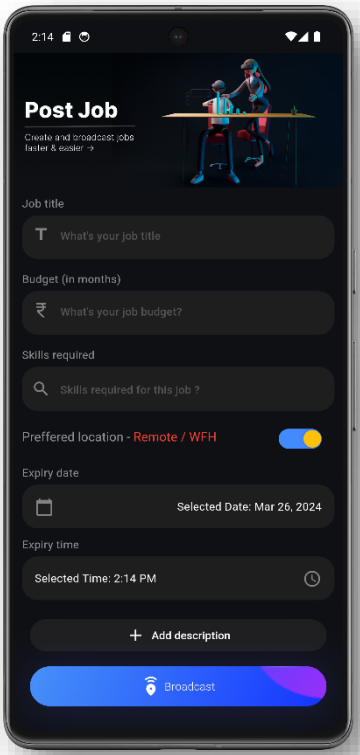
9.2.22 Manage posts



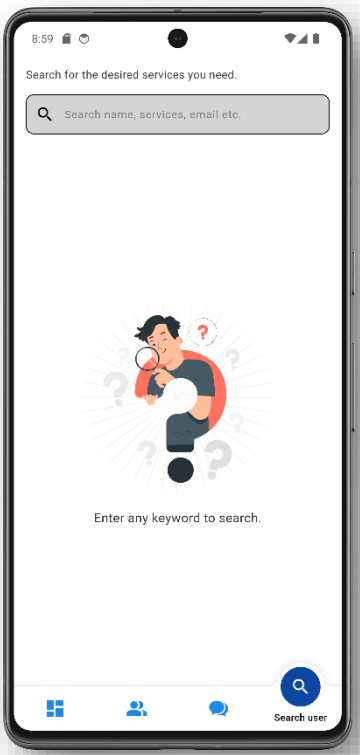
9.2.23 Description



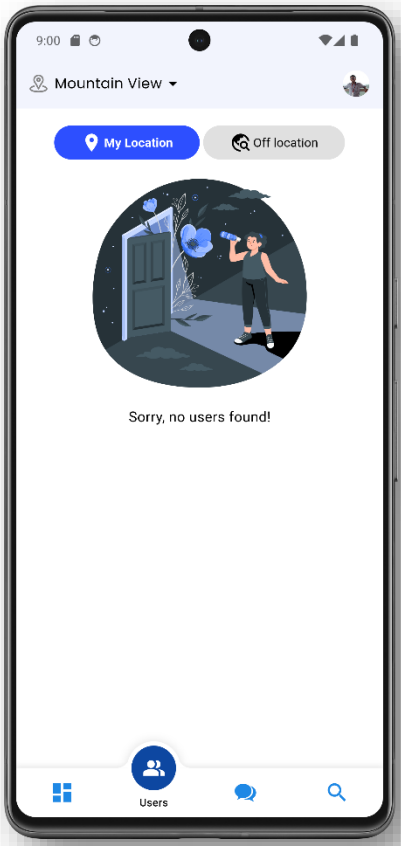
9.2.24 Create jobs



9.2.25 Search users

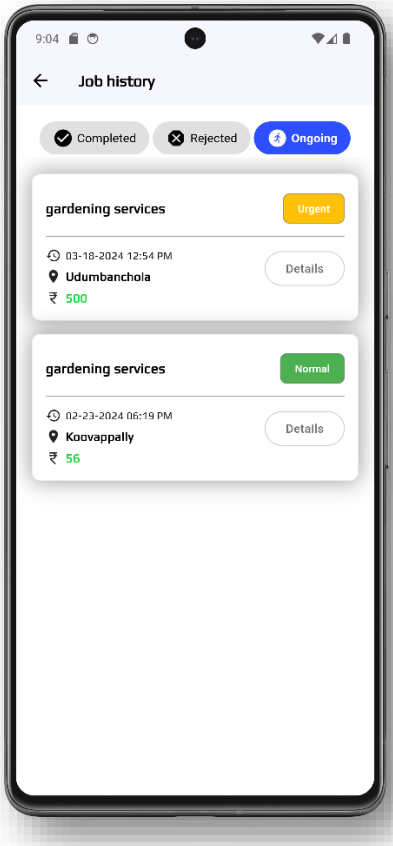


9.2.26 Nearby users



9.2.27 Favourites

9.2.28 Job history



9.2.29 Notifications