**NearbyNexus**

*Mini 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 ‘A’ 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.BinumonJoseph Ms. Meera Rose Mathew**

**Internal Guide Coordinator**

**Rev. Fr. Dr. Rubin Thottupurathu Jose**

**Head of the Department**

**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 Master of Computer Applications (MCA) from APJ Abdul Kalam Technological University, during the academic year 2023-2024.

**Date: 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 **Mrs. 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

The proposed project envisions a groundbreaking location-based application that seeks to forge connections between users and a diverse array of service providers, ranging from individual entrepreneurs to large-scale corporations. Its primary focus is on individuals venturing into unfamiliar cities or towns, aiming to fulfill their needs for essential services such as transportation, tailoring, legal assistance, manual labor, and much more. Through seamless integration, the system seamlessly incorporates individual service providers into its framework, guaranteeing optimal user convenience and satisfaction.

The application caters to three distinct user categories: Administrators, General Users, and Vendors, each endowed with unique functionalities:

Administrators oversee the platform's operations and ensure secure access through email or mobile number verification. This authentication process safeguards user and service provider privacy, prioritizing data security. Meanwhile, service providers possess the ability to register their services within the application, providing crucial details and descriptions. This empowers users to make informed decisions. Furthermore, users can engage in secure, cashless transactions within the application upon service completion, streamlining the payment process.

The application also boasts an efficient search function, enabling users to easily locate the services they require. Additionally, a robust rating and review system allows users to provide feedback for vendors after a service has concluded. The application further refines user experience by determining the current locations of both regular users and vendors, subsequently displaying pertinent vendors to users. Once a suitable service provider is identified, users can initiate a connection. Upon successful connection, the contact details of both parties are shared. This comprehensive suite of functionalities ensures a seamless and secure experience for users and service providers alike, positioning the proposed project as a transformative force in the domain of location-based service applications.

**CONTENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SL. NO** | | **TOPIC** | **PAGE NO** | |
| **1** | | **INTRODUCTION** |  | |
| **1.1** | | **PROJECT OVERVIEW** |  | |
| **1.2** | | **PROJECT SPECIFICATION** |  | |
| **2** | | **SYSTEM STUDY** |  | |
| **2.1** | | **INTRODUCTION** |  | |
| **2.2** | | **EXISTING SYSTEM** |  | |
| **2.3** | | **DRAWBACKS OF EXISTING SYSTEM** |  | |
| **2.4** | | **PROPOSED SYSTEM** |  | |
| **2.5** | | **ADVANTAGES OF PROPOSED SYSTEM** |  | |
| **3** | | **REQUIREMENT ANALYSIS** |  | |
| **3.1** | | **FEASIBILITY STUDY** |  | |
| **3.1.1** | | **ECONOMICAL FEASIBILITY** |  | |
| **3.1.2** | | **TECHNICAL FEASIBILITY** |  | |
| **3.1.3** | | **BEHAVIORAL FEASIBILITY** |  | |
| **3.1.4** | | **FEASIBILITY STUDY QUESTIONNAIRE** |  | |
| **3.2** | | **SYSTEM SPECIFICATION** |  | |
| **3.2.1** | | **HARDWARE SPECIFICATION** |  | |
| **3.2.2** | | **SOFTWARE SPECIFICATION** |  | |
| **3.3** | | **SOFTWARE DESCRIPTION** |  | |
| **3.3.1** | | **PHP** |  | |
| **3.3.2** | | **MYSQL** |  | |
| **4** | | **SYSTEM DESIGN** |  | |
| **4.1** | | **INTRODUCTION** |  | |
| **4.2** | | **UML DIAGRAM** |  | |
| **4.2.1** | | **USE CASE DIAGRAM** |  | |
| **4.2.2** | | **SEQUENCE DIAGRAM** |  | |
| **4.2.3** | | **STATE CHART DIAGRAM** |  | |
| **4.2.4** | | **ACTIVITY DIAGRAM** |  | |
| **4.2.5** | | **CLASS DIAGRAM** |  | |
| **4.2.6** | | **OBJECT DIAGRAM** |  | |
| **4.2.7** | | **COMPONENT DIAGRAM** |  | |
| **4.2.8** | | **DEPLOYMENT DIAGRAM** |  | |
| **4.2.9** | | **COLLABORATION DIAGRAM** |  | |
| **4.3** | | **USER INTERFACE DESIGN USING FIGMA** |  | |
| **4.4** | | **DATABASE DESIGN** |  | |
| **5** | | **SYSTEM TESTING** |  | |
| **5.1** | | **INTRODUCTION** |  | |
| **5.2** | | **TEST PLAN** |  | |
| **5.2.1** | **UNIT TESTING** | |  |
| **5.2.2** | **INTEGRATION TESTING** | |  |
| **5.2.3** | **VALIDATION TESTING** | |  |
| **5.2.4** | **USER ACCEPTANCE TESTING** | |  |
| **5.2.5** | **AUTOMATION TESTING** | |  |
| **5.2.6** | **SELENIUM TESTING** | |  |
| **6** | **IMPLEMENTATION** | |  |
| **6.1** | **INTRODUCTION** | |  |
| **6.2** | **IMPLEMENTATION PROCEDURE** | |  |
| **6.2.1** | **USER TRAINING** | |  |
| **6.2.2** | **TRAINING ON APPLICATION SOFTWARE** | |  |
| **6.2.3** | **SYSTEM MAINTENANCE** | |  |
| **7** | **CONCLUSION & FUTURE SCOPE** | |  |
| **7.1** | **CONCLUSION** | |  |
| **7.2** | **FUTURE SCOPE** | |  |
| **8** | **BIBLIOGRAPHY** | |  |
| **9** | **APPENDIX** | |  |
| **9.1** | **SAMPLE CODE** | |  |
| **9.2** | **SCREEN SHOTS** | |  |

## List of Abbreviation

**UML - Unified Modeling Language**

**IDE - Integrated Development Environment**

# CHAPTER 1

# INTRODUCTION

### PROJECT OVERVIEW

NearbyNexus is an application platform designed to connect users in a new town or city with various service providers based on their geographical location. The application caters to a wide range of services, from small-scale individual service providers to larger organizations. Its primary goal is to facilitate easy and convenient access to services without the need for users to have the contact information of the providers readily available. The users of the application need to be registered their account through a simple registration process which will grant the access to the application and its various functionalities.

The service provider or vendors also have to register for an account and verify their truthfulness to start servicing through the application. The administrator of the application has access to the user details and manages and monitor the activities of the user. The admin will take necessary actions upon a user or vendor based on his/her activity and other user reports. When the user is successfully login to then the user is displayed with a wide number of services that are available on geographical location. Through this application the users and vendors can sign up the services and achieve the goals

### 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.
* **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
* **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
  +  Receive service request notification

# CHAPTER 2

# SYSTEM STUDY

### INTRODUCTION

### EXISTING SYSTEM

**2.2.1 NATURAL SYSTEM STUDIED**

**2.2.2 DESIGNED SYSTEM STUDIED**

### DRAWBACKS OF EXISTING SYSTEM



### PROPOSED SYSTEM

### ADVANTAGES OF PROPOSED SYSTEM



# CHAPTER 3

# REQUIREMENT ANALYSIS

## 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.

### Economical 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.

### 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.

### 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 behaviour or service quality concerns will build trust and confidence in the application.

The behavioural 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 behavioural 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.

1. **To what extend the system is proposed for?**

The primary goal of the application is to provide users with easy access to a wide range of services in their local area. By utilizing the user's geographical location, the application connects them with nearby service providers, including individual service providers and organizations. The application seeks to enhance convenience for users by eliminating the need for prior contact information of service providers. Users can browse through various services, view service details, and request services through the app, making it convenient to avail of services quickly. The system is proposed to cover a diverse set of service categories, ensuring that users can find and avail services ranging from transportation (taxi drivers) to professional services (lawyers), household services (plumbers, electricians), and personal services (beauticians, tailors).

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

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

1. **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, etc..

1. **Identify the users in your project?**

Admin, General users (Public), Service providers.

1. **Who owns the system?**

Administrator

1. **System is related to which firm/industry/organization?**

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

1. **Details of person that you have contacted for data collection?**

Online resources

Athul Vinayakumar - IT professional

1. **Questionnaire to collect details about the project?**
   1. **How do you ensure the authenticity of vendors and users?**

NearbyNexus encourages service providers to provide information about their licenses, certifications, and insurance. This information helps customers assess the legitimacy and professionalism of the service provider. And for users are verified by their login credentials.

* 1. **How does the user registration process work? What information do you collect from users during registration?**

Initially the general users of the application are verified by technologies like OTP, SMS etc. this is done to ensure the authenticity of the user. The users are required to provide a good quality picture of them. During the registration process this application collects the necessary information’s like phone number, Emal id, facial images etc. After successful registration the geographical location are accessed and stored based on users’ permissions.

* 1. **Which payment gateway(s) are integrated into the application for online transactions?**

We often use Stripe for integrating payments and transaction through out the application which enable secure and successful transactions.

* 1. **How do users and vendors communicate within the application? Are there any additional features for user-vendor communication?**

The users and the vendors of the application can communicate with each other seamlessly through the in- app messaging system. Which enable safe and secure communication medium.

* 1. **How do you handle and display the 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 feedbacks about the users to whom they provided a service. Which describes the attitude and payment responsibility of the user is reviewed.

* 1. **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.

* 1. **How does the administrator handle user complaints, reports, and vendor verification's?**

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

* 1. **List the various service categories that the application covers. Are there any plans to expand the service categories in the future?**

As mentioned earlier the services of the applications are mostly created by the service provider who register with the application. When a service provider add a new service into their catalogue the service is automatically sync with the databases of the application. The admin reviews the service and approve the services added by the new vendor. Next time another user register for a service then the service list contains the newly approved services.

* 1. **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.

* 1. **Does the application integrate with other external systems or APIs for additional functionalities (e.g., maps, third-party services)?**

Yes, this application integrates other external systems and APIs for the smooth working and good user experience for the application and the user. Which may include maps, location services etc.

## SYSTEM SPECIFICATION

### Hardware Specification

Processor - AMDRyzen5

RAM - 8 GB

Hard disk - 1 TB

### Software Specification

Front End - Flutter, Dart

Back End - Firebase

Database - FirebaseFirestore

Client on PC - Windows 7 and above.

Technologies used - Dart, Flutter,Rapid API, Firebase Firestore & FirebasStorage

## SOFTWARE DESCRIPTION

### 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.

### 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

* 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.

## UML 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

## 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 functionalities.

## 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.

## 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:

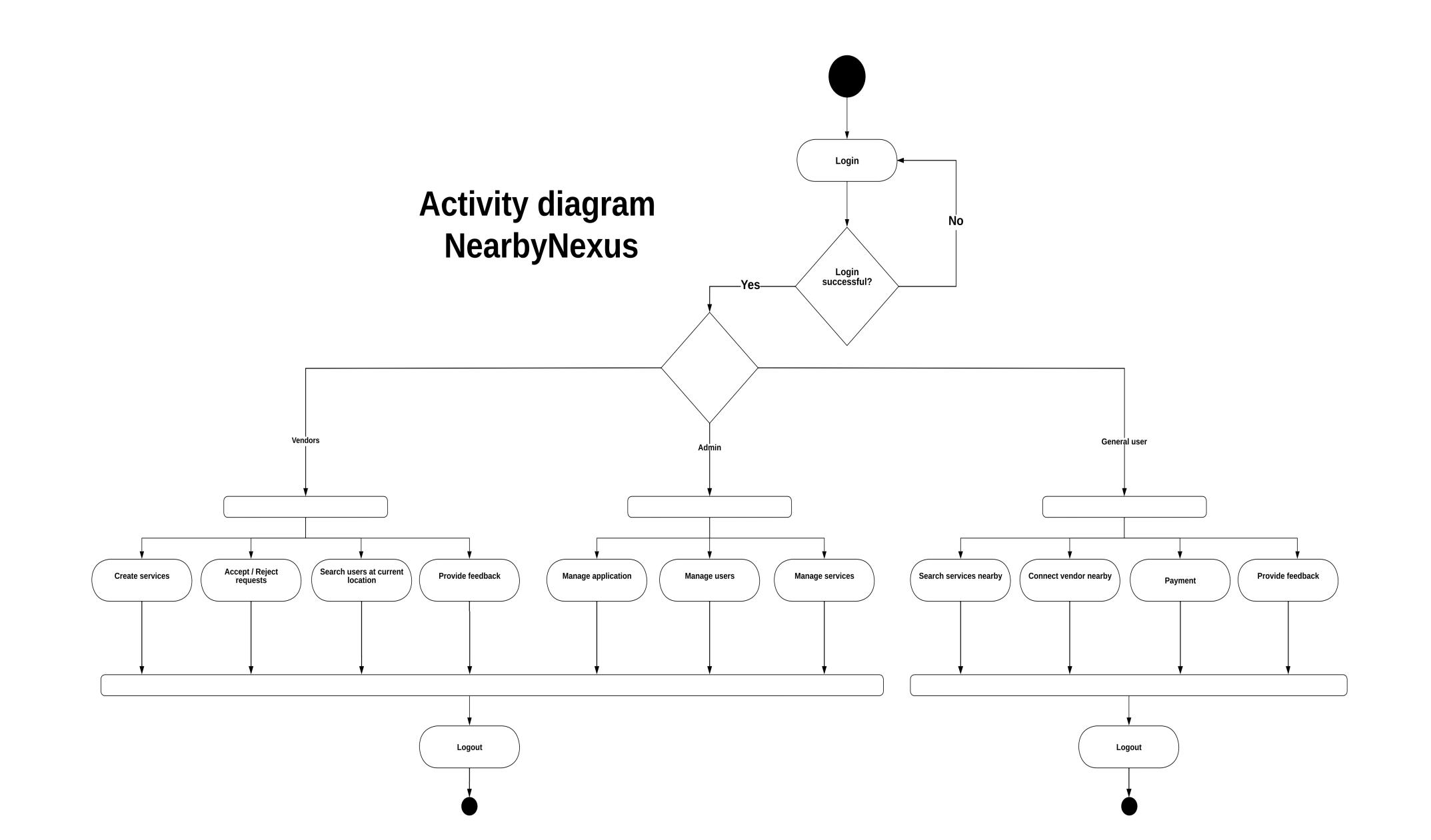
* **Initial State:** Represented by a filled circle, it signifies the starting point of the object's lifecycle.
* **State:** Depicted by rounded rectangles, states represent distinct phases in an object's existence.
* **Transition Arrow:**Arrows denote transitions between states, indicating the conditions 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.

## 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.

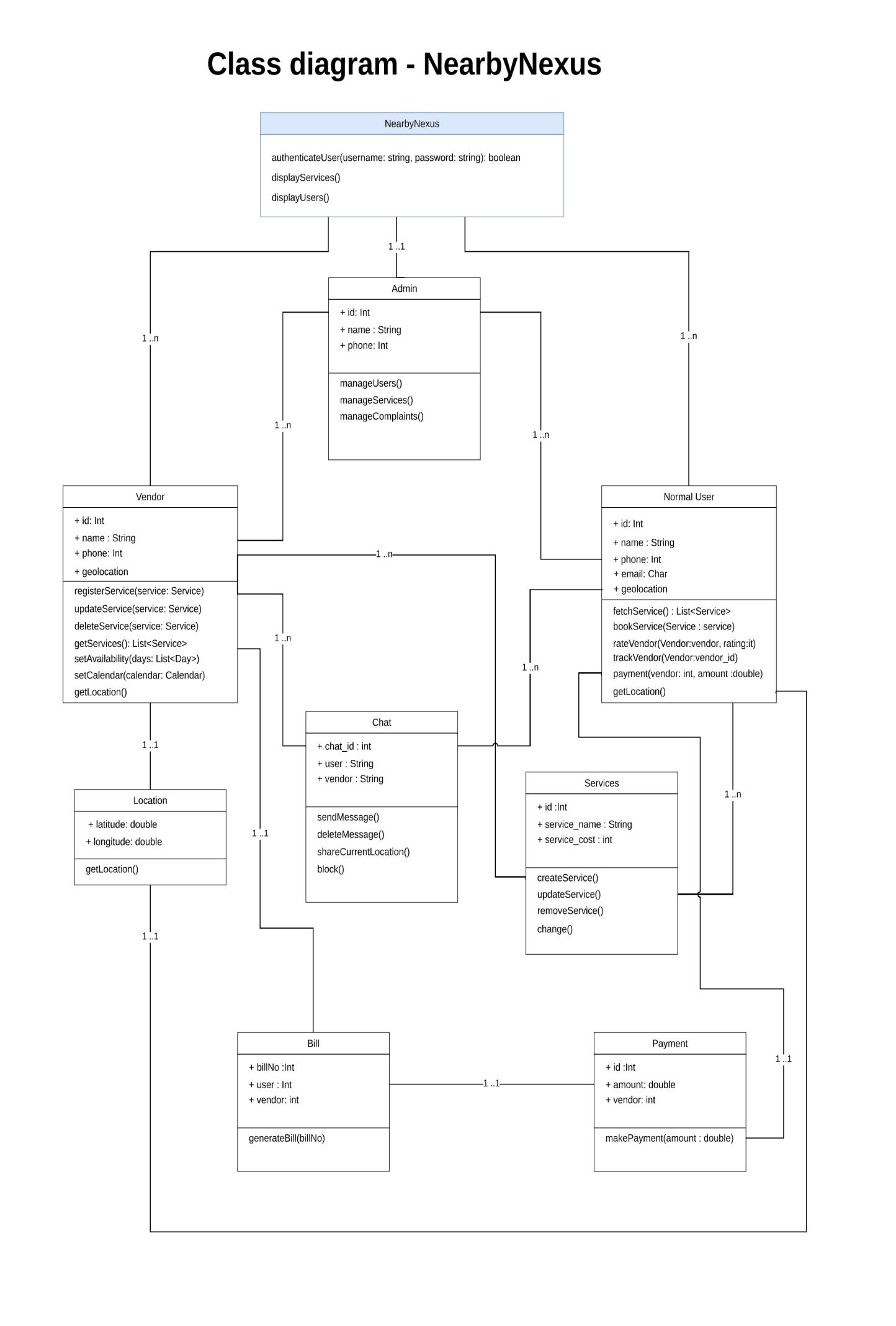


## CLASS DIAGRAM

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.

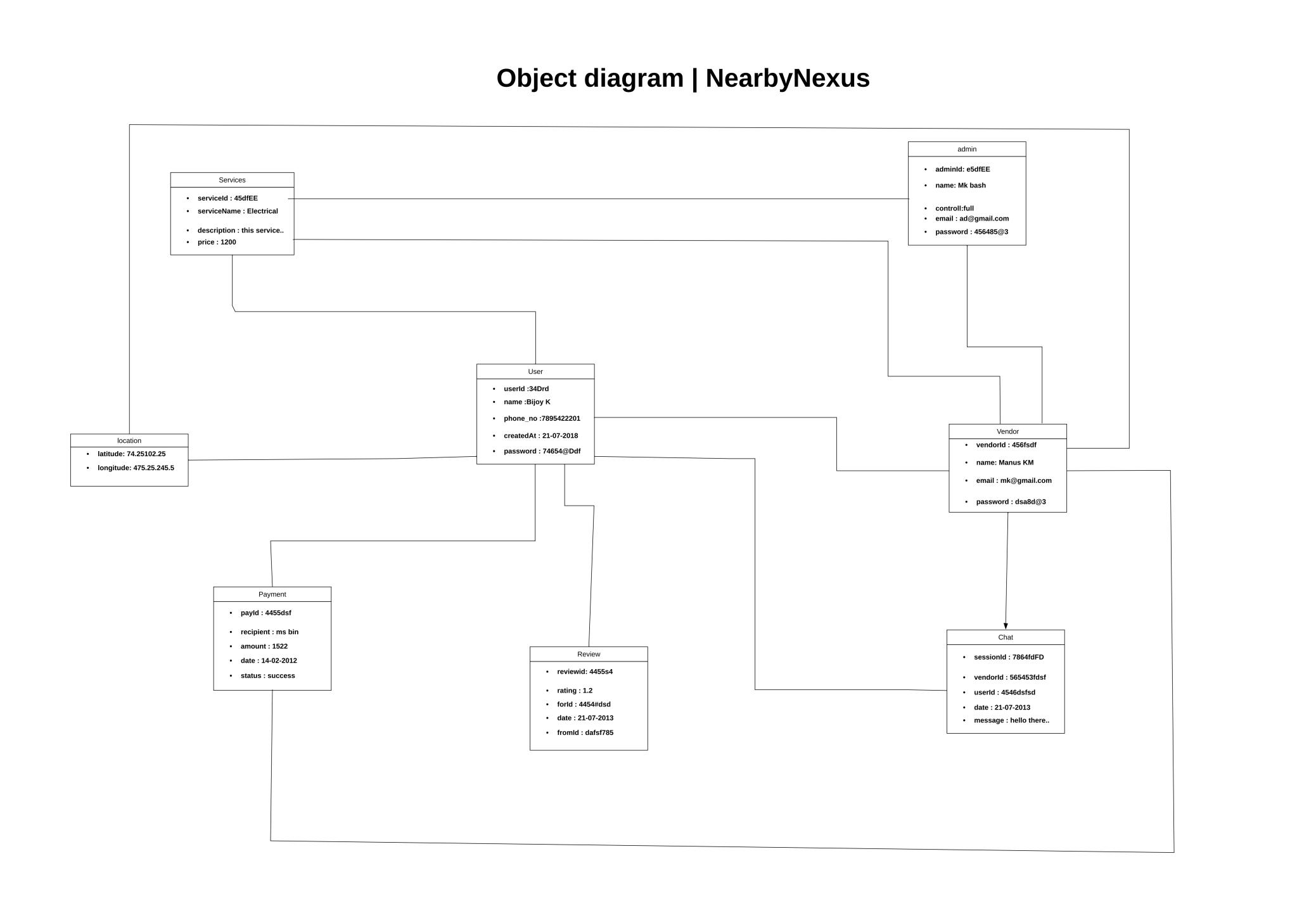


## OBJECT DIAGRAM

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.

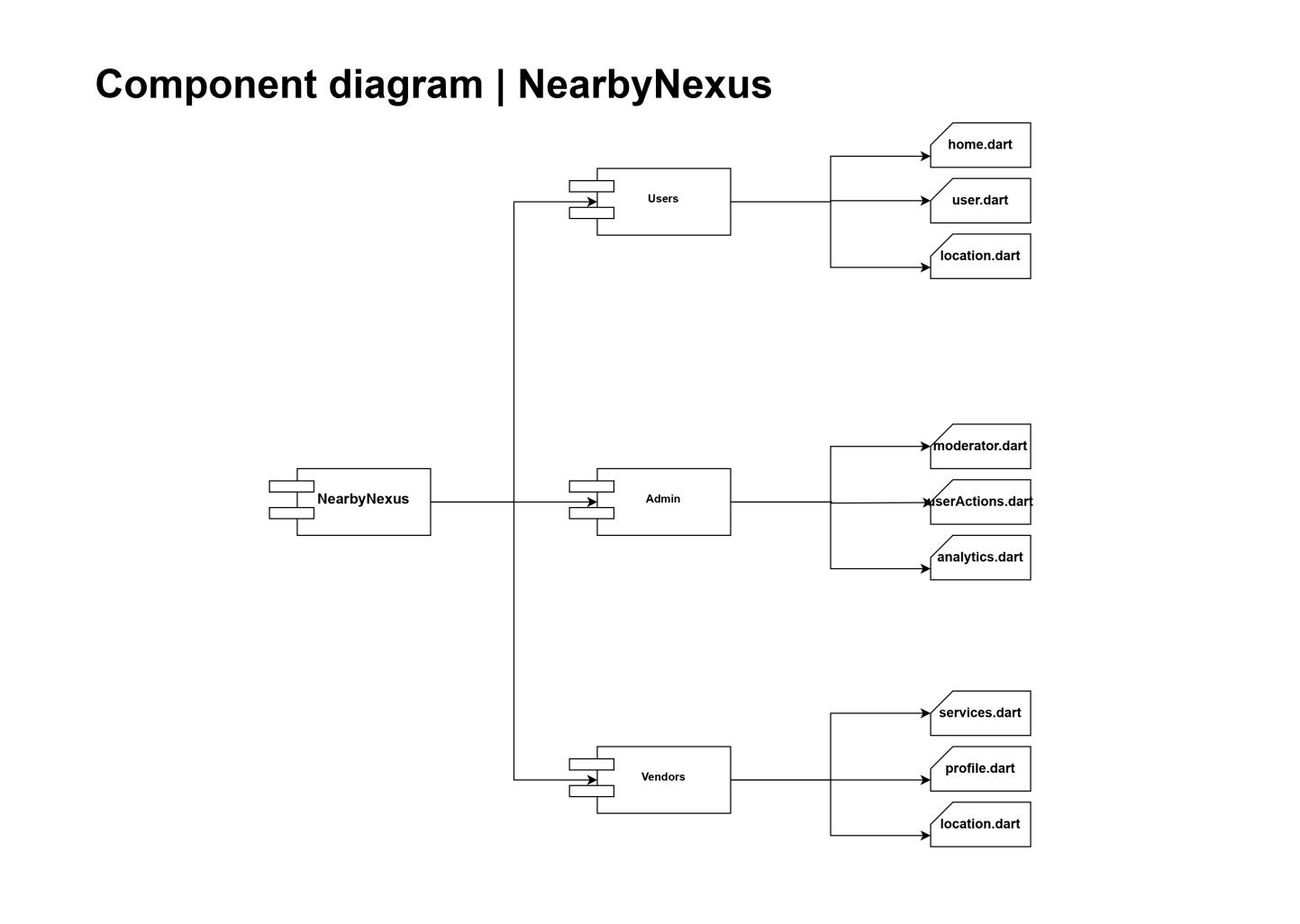


## COMPONENT DIAGRAM

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.

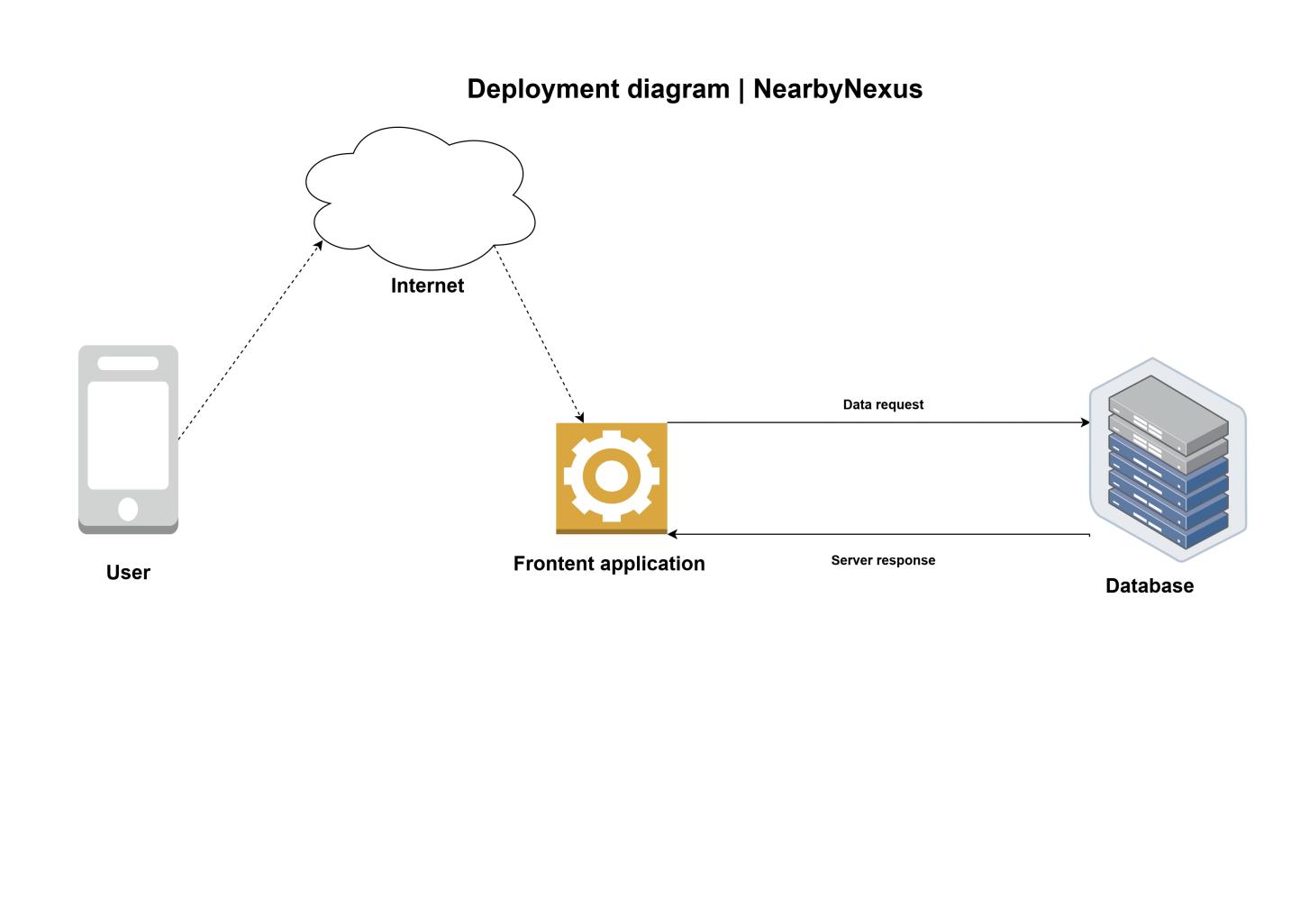
Key notations for Component Diagrams:

* **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.



**4.2.8 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.



**4.2.9 Collaboration Diagram**

Explanation, Diagram

## 4.3 USER INTERFACE DESIGN USING FIGMA

**Form Name: abcc**

Screenshot

**Form Name: abcc**

Screenshot

All Forms

## 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 **: 4.5.1**

Collection name **: users**

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Field name** | **Datatype** | **Description** |
| 2 | name | String | Stores the name of the user. |
| 4 | about | String | Stores the about description of the user. |
| 5 | address | String | Stores the address of the user. |
| 6 | geoLocation | String | Stores the geo-location of the user. |
| 8 | image | String | Contains the profile image of the user. |
| 11 | status | String | Stores the account status of the user whether he/she is disabled or banned from the application etc.. |
| 13 | userType | String | Specifies the user's type. |
| 15 | activityStatus | String | Specifies if the user is available or not. |
| 16 | actualRating | Number | Contains the rating score of the user out of 5 |
| 20 | totalRating | Number | Contains the sum of the all ratings |
| 3 | emailId | Map | Stores the email id of the user and the verification status of the email id. |
| 10 | phone | Map | Stores the phone number of the user and the verification status of the phone number |
| 14 | kycDetails | Map | Stores the necessary details related users KYC |
| 1 | docId | Document | This id is auto generated by Firebase & and act as individual user id. |
| 7 | iamRated | Array(references) | The list contains the users who have been rated by the user. |
| 9 | paymentLogs | Array(references) | Stores the reference id to the payments collection. |
| 12 | userFavourites | Array(references) | The system stores the reference IDs of users who have been added to the user's favorite list. |
| 17 | allRatings | Array(references) | This refers to the 'ratings' collection, which contains details pertaining to user ratings. |
| 18 | languages | Array | Lists user's spoken languages. |
| 19 | services | Array | Lists the services provided by the user. |
| 21 | workingDays | Array | Lists days the user available. |

**4.5.1 Table(Collection)**

Collection **: 4.5.2**

Collection name **: 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 **: 4.5.3**

Collection name **: 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 **: 4.5.4**

Collection name **: 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. |

**4.5.4 Table(Collection)**

# CHAPTER 5

# SYSTEM TESTING

* 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.

## 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

### 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.

### UnitTesting

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.

### 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.

### 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.
  + 1. **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.

* + 1. **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**

import 'package:flutter/material.dart';

import 'package:flutter\_test/flutter\_test.dart';

import 'package:integration\_test/integration\_test.dart';

import 'package:NearbyNexus/main.dart' as app;

void main() {

group('User Test', () {

IntegrationTestWidgetsFlutterBinding.ensureInitialized();

testWidgets("Login Scenarios", (widgetTester) async {

app.main();

// Ensure the splash screen is shown.

await widgetTester.pumpAndSettle(

Duration(seconds: 2)); // Adjust the duration as needed.

// Assuming the splash screen automatically navigates to the login screen.

final getStartedBtn = find.byKey(Key('getStartedBtn'));

await widgetTester.tap(getStartedBtn);

await widgetTester.pumpAndSettle();

final loginEmailTester = find.byKey(Key('LoginEmail'));

final loginPasswordTester = find.byKey(Key('LoginPassword'));

final loginButtonTester = find.byKey(Key('LoginButton'));

// Scenario 1: Press login button without entering anything

print("\x1B[34mScenario 1:\x1B[0m Press login button without entering anything");

await widgetTester.tap(loginButtonTester);

await widgetTester.pumpAndSettle(Duration(seconds: 5));

expect(find.text("Test Pass"), findsNothing); // Expecting no "Test` Pass" indicator

// Scenario 2: Enter wrong password and press login

print("\x1B[34mScenario 2:\x1B[0m Enter wrong password and press login");

await widgetTester.enterText(loginEmailTester, "donbenny916@gmail.com");

await widgetTester.enterText(loginPasswordTester, "wrong\_password");

await widgetTester.tap(loginButtonTester);

await widgetTester.pumpAndSettle(Duration(seconds: 5));

expect(find.text("Test Pass"), findsNothing); // Expecting no "Test Pass" indicator

// Scenario 3: Enter correct email and password

print("\x1B[34mScenario 3:\x1B[0m Enter correct email and password");

await widgetTester.enterText(loginEmailTester, "donbenny916@gmail.com");

await widgetTester.enterText(loginPasswordTester, "123456");

await widgetTester.tap(loginButtonTester);

await widgetTester.pumpAndSettle(Duration(seconds: 5));

expect(find.text("Test Pass"), findsNothing); // Expecting no "Test Pass" indicator

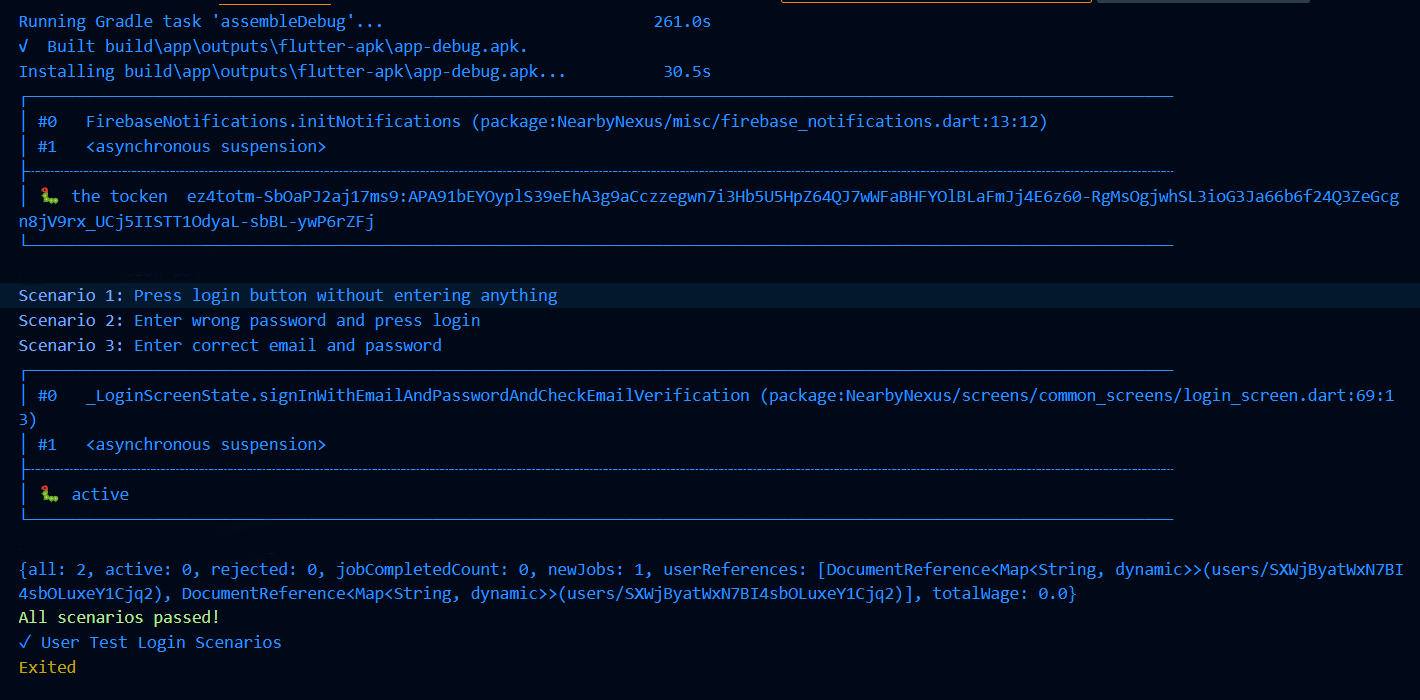
print("\x1B[32mAll scenarios passed!\x1B[0m");

});

});

}

**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:** | | |
| **Module Name**: Login Screen | | | **Test Executed By : Mr. Binumon Joseph** | | |
| **Test Title : Login\_Scenario** | | | **Test Execution Date:** | | |
| **Description: Test the login page** | | |  | | |
| **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 without any credentials |  | Error message should be displayed | Error message displayed | Pass |
| 3 | Enter wrong password then press login | wrong\_password | Invalid credential message should shown | Error message displayed | Pass |
| 4 | Enter valid Email and password | [donbenny916@gmail.com](mailto:donbenny916@gmail.com)  123456 | No error should displayed | No error occurred | Pass |
| **Post-Condition:**After successful validation with the database, the user is able to log into their account. The session details associated with the account are then appropriately logged into the database for future reference and security purposes. This confirms that the user's interaction with the system is authenticated and systematically tracked, ensuring a secure and reliable user experience. | | | | | |

**Test Case 2:**

**Code**

**Screenshot**

**Test report**

**Minimum 4 test cases (1 login 3 functionalities)**

# CHAPTER 6

# IMPLEMENTATION

## INTRODUCTION

Explanation

## IMPLEMENTATION PROCEDURES

Explanation

### User Training

Explanation

### Training on the Application Software

Explanation

### System Maintenance

Explanation

# CHAPTER 7

# CONCLUSION AND FUTURE SCOPE

## CONCLUSION

## 

.

* 1. **FUTURE SCOPE**

.

# CHAPTER 8

# BIBLIOGRAPHY

### REFERENCES:

* + - ..
    - ..
    - ..
    - ..
    - ...

### WEBSITES:

* + - [..](http://www.w3schools.com/)
    - [..](http://www.jquery.com/)
    - [..](http://homepages.dcc.ufmg.br/%7Erodolfo/es-1-03/IEEE-Std-830-1998.pdf)
    - [..](http://www.agilemodeling.com/artifacts/useCaseDiagram.html)

# CHAPTER 9

# APPENDIX

## Sample Code

Main functionalities

## Screen Shots