**#** **Smart Local Services Project Notes**

**##** **1. Theme/Title Options**

- Smart Local Services

- Design and Implementation of a Smart Local Services Application

- Design and Implementation of a Smart Local Services Application: Case Study of [Your City/Region]

- Design and Implementation of a Smart Local Services Application (A digital platform to help users easily find, book, and pay for trusted local service providers such as plumbers, electricians, tutors, and delivery agents.)

**##** **2. Expanded Theme Statement**

Smart Local Services is a digital platform designed to make it easy for people to find, book, and pay for trusted local service providers—such as plumbers, electricians, tutors, cleaners, and delivery agents—directly from their mobile phones or computers. The app connects users with skilled professionals in their area, streamlines the booking process, and ensures secure, cashless payments, making daily life more convenient and reliable for everyone.

**##** **3. App Name Suggestions**

- Servigo

- Taskly

- HelpNow

- QuickTask

- ServEase

- BookIt

- FixIt

- HandyApp

- TaskUp

- Servio

**##** **4. Main Actors**

| Actor            | Role/Responsibility                                              |

|------------------|-----------------------------------------------------------------|

| User             | Books and pays for services, rates providers                     |

| Service Provider | Offers services, manages bookings, receives payments             |

| Admin            | Manages platform, verifies providers, handles support/disputes   |

| Payment Provider | Processes payments securely (external system)                    |

**##** **5. Key Features**

**###** **For Users (Customers):**

- Browse and search for services (plumbing, electrical, cleaning, tutoring, delivery, etc.)

- Book and schedule services

- Ratings & reviews for providers

- In-app payments (mobile money, card, wallet)

- Job tracking (pending, accepted, in progress, completed)

- Notifications (booking status, provider arrival, promotions)

- Chat/call with service provider

**###** **For Service Providers:**

- Register and create profile

- Manage bookings (accept/decline, update status)

- Receive payments

- Build reputation through ratings/reviews

- Offer promotions

**###** **For Admins:**

- Manage users and providers

- Verify provider credentials

- Monitor transactions

- Analytics dashboard

- Support and dispute resolution

**##** **6. Use Case Explanation**

A Local Services Marketplace App is a digital platform that connects users who need services with vetted, rated, and available local service providers. Users can browse, book, and pay for services directly from their phone, while providers can register, list their services, receive bookings, and get paid through the app.

**\*\*Example User Flow:\*\***

1. User searches for a plumber.

2. App shows available plumbers nearby with ratings and prices.

3. User books a service for a specific date/time.

4. Provider receives notification and accepts the job.

5. Provider completes the work and marks the job as done.

6. User pays through the app.

7. User leaves a rating and review.

8. Provider receives payment and a new review.

**##** **7. Monetization Strategies**

- Commission on each completed job

- Subscription for providers (premium placement, more bookings)

- Ads/promotions (featured providers)

- Service fees for users

**##** **8. Technical Stack Suggestions**

- Frontend: Flutter (iOS, Android, Web) or React Native

- Backend: Firebase (for rapid development) or Django/Node.js (for more control)

- Payments: MTN/Orange Mobile Money APIs

- Notifications: Firebase Cloud Messaging or Twilio

**##** **9. Simple Theme and App Name Pairings**

- Theme: Smart Local Services

  App Name: Servigo

- Theme: Easy Service Booking

  App Name: Taskly

- Theme: On-Demand Help App

  App Name: HelpNow

**##** **10. Formal Theme Example with Description**

Design and Implementation of a Smart Local Services Application

(A digital platform to help users easily find, book, and pay for trusted local service providers such as plumbers, electricians, tutors, and delivery agents.)

**---**

**\*\*You can use, edit, or expand this file for your project report, defense, or planning!\*\***

Great! Please specify which page or feature you want to work on next. Here are some common options for a local services platform:

1. **Service Selection for Providers:** Allow providers to select and manage the services they offer.
2. **Profile Editing:** Let users (customers and providers) update their profile information.
3. **Service Listing Page:** Show all available services and allow users to browse/filter them.
4. **Provider Listing Page:** Show all providers, with filters by service, city, etc.
5. **Service Request/Booking:** Allow customers to request/book a service from a provider.
6. **Messaging/Notifications:** Enable communication between customers and providers.
7. **Admin Dashboard:** Manage users, services, and requests.

**Which feature or page would you like to implement next?**Just reply with the number or

**PLANTUML CODE**

**GENERAL USE CASE DIAGRAM**

@startuml

actor Customer

actor "Service Provider" as Provider

actor Admin

Customer <|-- Provider

Customer <|-- Admin

actor "Payment Provider" as Payment

actor "Geolocation API" as GeoAPI

actor "AI Module" as AI

rectangle "Smart Service Finder System" {

Customer -- (Switch Role)

Customer -- (Become Service Provider)

Customer -- (Browse Services)

Customer -- (Search Services)

Customer -- (Book Service)

Customer -- (Pay Deposit)

Customer -- (Pay Balance)

Customer -- (Pay Milestone)

Customer -- (Track Provider Location)

Customer -- (Rate & Review Provider)

Customer -- (Chat with Provider)

Customer -- (Receive Notifications)

Customer -- (Get AI Recommendations)

Provider -- (List Services)

Provider -- (Manage Bookings)

Provider -- (Update Service Status)

Provider -- (Request Milestone Payment)

Provider -- (Receive Payments)

Provider -- (Get AI Insights)

Admin -- (Verify Providers)

Admin -- (Manage Customers)

Admin -- (Manage Providers)

Admin -- (Monitor Transactions)

Admin -- (Resolve Disputes)

Admin -- (View Analytics)

(Pay Deposit) -- Payment

(Pay Balance) -- Payment

(Pay Milestone) -- Payment

(Track Provider Location) -- GeoAPI

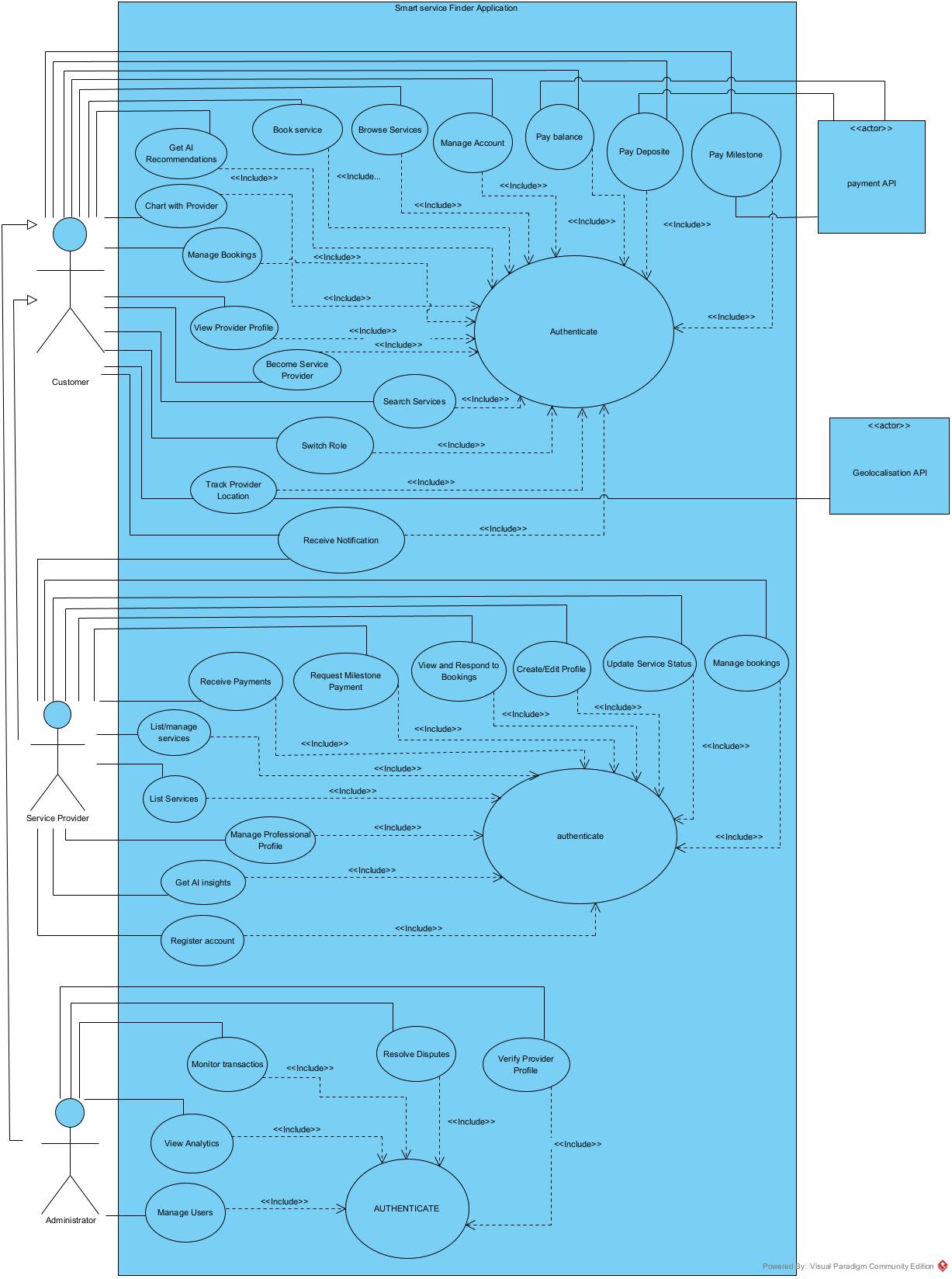
(Get AI Recommendations) -- AI

(Get AI Insights) -- AI

(Search Services) -- AI

}

@enduml



**SPECIFIC USE CASE DIAGRAMS**

**Book service**

@startuml

actor Customer

rectangle "Smart Service Finder System" {

(Book Service)

(Search Services)

(Select Provider)

(Pay Deposit)

(Rate Provider)

Customer -- (Book Service)

(Book Service) ..> (Search Services) : <<include>>

(Book Service) ..> (Select Provider) : <<include>>

(Book Service) ..> (Pay Deposit) : <<extend>>

(Book Service) ..> (Rate Provider) : <<extend>>

}

@enduml

**Manage users**

@startuml

actor Admin

rectangle "Smart Service Finder System" {

(Manage Users)

(View User List)

(Edit User)

(Delete User)

Admin -- (Manage Users)

(Manage Users) ..> (View User List) : <<include>>

(Manage Users) ..> (Edit User) : <<include>>

(Manage Users) ..> (Delete User) : <<include>>

}

@enduml

**Manage Bookings**

@startuml

actor "Service Provider" as ServiceProvider

rectangle "Smart Service Finder System" {

(Manage Bookings)

(View Bookings)

(View Booking Details)

(Accept Booking)

(Reject Booking)

(Cancel Booking)

(Update Booking)

(Contact Customer)

ServiceProvider -- (Manage Bookings)

(Manage Bookings) ..> (View Bookings) : <<include>>

(Manage Bookings) ..> (View Booking Details) : <<include>>

(Manage Bookings) ..> (Accept Booking) : <<extend>>

(Manage Bookings) ..> (Reject Booking) : <<extend>>

(Manage Bookings) ..> (Cancel Booking) : <<extend>>

(Manage Bookings) ..> (Update Booking) : <<extend>>

(Manage Bookings) ..> (Contact Customer) : <<extend>>

}

@enduml

**CLASS DIAGRAM**

@startuml

class User {

-userId: String

-name: String

-email: String

-phone: String

-password: String

-role: UserRole

+register()

+login()

+updateProfile()

}

class Customer {

-bookingHistory: List<Booking>

+searchServices()

+bookService()

+makePayment()

+rateProvider()

}

class ServiceProvider {

-services: List<Service>

-rating: float

-isVerified: boolean

+addService()

+updateService()

+manageBookings()

+receivePayment()

}

class Admin {

+verifyProvider()

+manageUsers()

+resolveDisputes()

+viewAnalytics()

}

class Service {

-serviceId: String

-name: String

-description: String

-price: double

-category: String

-status: ServiceStatus

-provider: ServiceProvider

+create()

+update()

+delete()

}

class Booking {

-bookingId: String

-service: Service

-customer: Customer

-provider: ServiceProvider

-status: BookingStatus

-totalAmount: double

-date: DateTime

+create()

+update()

+cancel()

}

class Payment {

-paymentId: String

-booking: Booking

-amount: double

-status: PaymentStatus

-type: PaymentType

+process()

+refund()

}

class Review {

-reviewId: String

-booking: Booking

-rating: int

-comment: String

-date: DateTime

+create()

+update()

}

enum UserRole {

CUSTOMER

PROVIDER

ADMIN

}

enum ServiceStatus {

ACTIVE

INACTIVE

PENDING

}

enum BookingStatus {

PENDING

ACCEPTED

REJECTED

COMPLETED

CANCELLED

}

enum PaymentStatus {

PENDING

COMPLETED

FAILED

REFUNDED

}

User <|-- Customer

User <|-- ServiceProvider

User <|-- Admin

ServiceProvider "1" -- "\*" Service

Customer "1" -- "\*" Booking

ServiceProvider "1" -- "\*" Booking

Service "1" -- "\*" Booking

Booking "1" -- "1" Payment

Booking "1" -- "0..1" Review

' Connect enums to their respective classes

User "1" -- "1" UserRole

Service "1" -- "1" ServiceStatus

Booking "1" -- "1" BookingStatus

Payment "1" -- "1" PaymentStatus

@enduml

**ADMIN VERIFIES SERVICE PROVIDER SEQUENCE DIAGRAM**

@startuml

title Admin Verifies Service Provider Sequence Diagram

skinparam sequence {

LifeLineBorderColor green

LifeLineBackgroundColor #eaffea

ParticipantBorderColor black

ParticipantBackgroundColor #f5f5f5

ActorBorderColor black

ActorBackgroundColor #f5f5f5

ArrowColor black

ActorStyle awesome

}

actor Admin

participant "System (Application)" as System

participant "DBMS" as DB

actor "Service Provider" as Provider

' Admin must authenticate before verifying providers

note over Admin, System: Admin must be authenticated (login) before verifying providers

Admin -> System: Authenticate (login)

System -> DB: verifyCredentials()

DB --> System: Authentication result

System -> Admin: Authentication success/failure

' Admin requests to verify a service provider

Admin -> System: Verify service provider (providerId)

System -> DB: getProviderDocuments(providerId)

DB --> System: Provider documents/status

System -> System: Review documents and status

' System updates provider verification status

System -> DB: updateProviderStatus(providerId, verified)

DB -> DB: Update provider record

DB --> System: Update result

' System notifies provider of verification result

System ->> Provider: Verification approved notification

hide footbox

@enduml

**Service Provider Update Service Status Sequence Diagram**

@startuml

title Service Provider Update Service Status Sequence Diagram

skinparam sequence {

LifeLineBorderColor green

LifeLineBackgroundColor #eaffea

ParticipantBorderColor black

ParticipantBackgroundColor #f5f5f5

ActorBorderColor black

ActorBackgroundColor #f5f5f5

ArrowColor black

ActorStyle awesome

}

actor "Service Provider" as Provider

participant "System (Application)" as System

participant "DBMS" as DB

' Service Provider must authenticate before updating service status

note over Provider, System: Service Provider must be authenticated (login) before updating service status

Provider -> System: Authenticate (login)

System -> DB: verifyCredentials()

DB --> System: Authentication result

System -> Provider: Authentication success/failure

' Provider requests to update service status/details

Provider -> System: Update service status/details (serviceId, newStatus, newDetails)

System -> System: Validate update request

' System updates service in DBMS

System -> DB: updateService(serviceId, newStatus, newDetails)

DB -> DB: Update service record

DB --> System: Update result

' System processes update result

System -> System: Process update result

alt Update successful

System --> Provider: Service update successful

else Update failed

System --> Provider: Service update failed (error message)

end

hide footbox

@enduml

**authentication (login)**

@startuml

' Visual Paradigm-like style and green lifelines

skinparam sequence {

LifeLineBorderColor green

LifeLineBackgroundColor #eaffea

ParticipantBorderColor black

ParticipantBackgroundColor #f5f5f5

ActorBorderColor black

ActorBackgroundColor #f5f5f5

ArrowColor black

ActorStyle awesome

}

actor User

participant "System (Application)" as System

participant "DBMS" as DB

' User initiates login

User -> System: Enter username & password (login)

System -> System: Validate input format

' System queries DBMS to verify credentials

System -> DB: SELECT \* FROM users WHERE username=? AND password=?

DB -> DB: Check credentials

DB --> System: User record / Authentication result

' System processes authentication result

System -> System: Process authentication result

alt Login successful

System --> User: Login success (session/token issued)

else Login failed

System --> User: Login failed (error message)

end

hide footbox

@enduml

**Service Provider Booking Management Sequence Diagram**

@startuml

title Service Provider Booking Management Sequence Diagram

' Visual Paradigm-like style and green lifelines

skinparam sequence {

LifeLineBorderColor green

LifeLineBackgroundColor #eaffea

ParticipantBorderColor black

ParticipantBackgroundColor #f5f5f5

ActorBorderColor black

ActorBackgroundColor #f5f5f5

ArrowColor black

ActorStyle awesome

}

actor "Service Provider" as Provider

participant "System (Application)" as System

participant "DBMS" as DB

participant "Payment API" as Payment

actor User

' Service Provider must authenticate before managing bookings

note over Provider, System: Service Provider must be authenticated (login) before managing bookings

Provider -> System: Authenticate (login)

System -> DB: verifyCredentials()

DB --> System: Authentication result

System -> Provider: Authentication success/failure

' System notifies provider of new booking (asynchronous)

System ->> Provider: notifyNewBooking(bookingId)

' Provider views booking details (synchronous)

Provider -> System: View booking details

System -> DB: getBookingDetails(bookingId)

DB --> System: Return booking details

System -> System: Process booking details

System --> Provider: Show booking details

' Provider accepts booking (synchronous)

Provider -> System: Accept booking

System -> DB: updateBookingStatus(bookingId, accepted)

DB -> DB: Update booking status

DB --> System: Booking status updated

System -> System: Prepare user notification

' System notifies user (asynchronous)

System ->> User: Booking accepted notification

' Optionally, system may interact with payment API for provider payout

System ->> Payment: initiateProviderPayout(bookingId)

Payment -->> System: Payout confirmation

System -> System: Process payout confirmation

hide footbox

@enduml

**BOOK SERVICE**

@startuml

' Set lifeline color to green and style for Visual Paradigm-like look

skinparam sequence {

LifeLineBorderColor green

LifeLineBackgroundColor #eaffea

ParticipantBorderColor black

ParticipantBackgroundColor #f5f5f5

ActorBorderColor black

ActorBackgroundColor #f5f5f5

BoxBorderColor black

BoxBackgroundColor #f5f5f5

ArrowColor black

ActorStyle awesome

}

actor Customer

participant "System (Application)" as System

participant "DBMS" as DB

participant "Service Provider" as Provider

participant "Payment API" as Payment

' Customer must authenticate before using the system

note over Customer, System: Customer must be authenticated (login) before using the system

Customer -> System: Authenticate (login/signup)

System -> DB: verifyCredentials()

DB --> System: Authentication result

System -> Customer: Authentication success/failure

' Customer searches for service (synchronous)

Customer -> System: Search for service

System -> DB: queryServices(criteria)

DB --> System: Return matching services

System -> System: Process service results

System --> Customer: Show matching services

' Customer selects service (synchronous)

Customer -> System: Select service

System -> DB: getServiceDetails(serviceId)

DB --> System: Return service details

System -> System: Process service details

System --> Customer: Show service details

' Customer books service (synchronous)

Customer -> System: Book service

System -> DB: createBooking(serviceId, customerInfo)

DB --> System: Booking created

System -> System: Prepare payment initialization

' System initializes payment (asynchronous)

System ->> Payment: initializePayment(amount)

Payment -->> System: Return payment link/info

System -> System: Process payment link/info

System --> Customer: Show payment info

' Customer makes payment (asynchronous)

Customer ->> Payment: Make payment

Payment -> Payment: Process payment

Payment -->> System: Payment confirmation

System -> System: Process payment confirmation

' System updates payment status (synchronous)

System -> DB: updateBookingPaymentStatus(bookingId, paid)

DB -> DB: Update payment status

DB --> System: Payment status updated

System -> System: Prepare provider notification

' System notifies provider (asynchronous)

System ->> Provider: notifyNewBooking(bookingId)

' Provider accepts booking (synchronous)

Provider -> System: Accept booking

System -> DB: update Booking Status (bookingId, accepted)

DB -> DB: Update booking status

DB --> System: Booking status updated

System -> System: Prepare booking confirmation

' System confirms booking to customer (asynchronous)

System ->> Customer: Booking confirmed

' Hide actors at the bottom

hide footbox

@enduml

**COMMUNICATION DIAGRAM FOR BOOK SERVICE**

@startuml

' Communication Diagram for Book Service (with hierarchical numbering and sync/async)

object Customer

object "System/Application" as System

object DBMS

object "Payment API"

object "Service Provider"

' 1. Service Search and Selection

Customer -> System : 1: Search for service

System -> DBMS : 1.1: Query services (criteria)

DBMS -> System : 1.2: Return matching services

System -> Customer : 1.3: Show matching services

Customer -> System : 2: Select service

System -> DBMS : 2.1: Get service details

DBMS -> System : 2.2: Return service details

System -> Customer : 2.3: Show service details

' 3. Booking

Customer -> System : 3: Book service

System -> DBMS : 3.1: Create booking (serviceId, customer info)

DBMS -> System : 3.2: Booking created

' 4. Payment Initialization

System -> System : 4: Prepare payment initialization

System --> "Payment API" : 4.1: Initialize payment (amount)

"Payment API" --> System : 4.2: Return payment link/info

System -> Customer : 4.3: Provide payment link/info

' 5. Payment Processing

Customer --> "Payment API" : 5: Make payment

"Payment API" --> System : 5.1: Payment confirmation

System -> System : 5.2: Process payment confirmation

System -> DBMS : 5.3: Update booking payment status (paid)

DBMS -> System : 5.4: Payment status updated

' 6. Provider Notification and Confirmation

System --> "Service Provider" : 6: Notify new booking

"Service Provider" --> System : 6.1: Accept booking

System -> DBMS : 6.2: Update booking status (accepted)

DBMS -> System : 6.3: Booking status updated

System -> Customer : 6.4: Booking confirmation

@enduml

**COMMUNICATION DIAGRAM FOR AUTHENTICATION**

@startuml

' Communication Diagram for Authentication (with sync/async indication)

object User

object System

object DBMS

' Synchronous: User waits for interface

User -> System : 1: Launch System

System -> User : 1.1: Display application interface

' Synchronous: User waits for login screen

User -> System : 2: Click on Login Button

System -> User : 2.1: Display's Login screen

' Synchronous: User submits form and waits

User -> System : 3: Fill Form and Submit

' Synchronous: System waits for DBMS response

System -> DBMS : 3.2: Send verification query

DBMS -> DBMS : 3.2.1: Execution of query

DBMS --> System : 3.2.2: return data ' Asynchronous (could be async if DBMS responds via event/callback)

' Synchronous: System processes result

System -> System : 3.1: Treat result

System -> System : 3.3: Treat result

' Synchronous: System responds to user

System -> User : 3.4: Render User Page

System -> User : 3.5: Return user does not exist

System -> User : 3.6: Display Error Message

@enduml

**ADMIN VERIFIES SERVICE PROVIDER COMMUNICATION DIAGRAM.**

@startuml

' Communication Diagram for Admin Verifying Service Provider (without authentication)

object Admin

object System

object DBMS

' 1. Admin requests to verify service provider

Admin -> System : 1: Request to service provider (providerId)

' 2. System fetches provider documents/status

System -> DBMS : 1.1: Fetch provider documents/status

DBMS -> System : 1.2: Return provider documents/status

' 3. System reviews documents and status (internal)

System -> System : 2: Review documents and status

' 4. If documents valid, update provider status to verified

System -> DBMS : 3: Update provider record (status: verified)

DBMS -> System : 3.1: Confirm update

' 5. If documents invalid, show failure or request more info

System -> Admin : 4: Show failure or request more info

' 6. Notify admin of update result

System -> Admin : 5: Show update result

' 7. If verified, send verification approved notification

System -> Admin : 6: Send verification approved notification

@enduml