Namal University, Mianwali

Department of Computer Science

System Design Report

Project Milestone 3

FareShare

A Ride-Sharing and Fare-Sharing System

Team FareShare

Muhammad Naveed	NUM-BSCS-2024-54	Group Lead
Munawar Ali	NUM-BSCS-2024-60	Group Member
Areeba Tahir	NUM-BSCS-2024-15	Group Member

Instructor: Asiya Batool

Requirement Provider: Rana Muhammad Adeel

Submission Date: January 18, 2026

Contents

1 Introduction

The goal of Project Milestone 3 is to turn the approved Software Requirements Specification (SRS) into a full system design. This report explains the system's behavior, structure, and interactions. We have used standard software modeling methods to ensure the design is consistent and directly follows the project requirements.

2 Design Assumptions and Constraints

2.1 Design Assumptions

- **Single Active Role:** Users can be both Drivers and Riders, but they cannot perform both roles at the same time in a single session.
- **GPS Reliability:** We assume all mobile devices have accurate GPS to calculate the exact distances needed for the fare-sharing formula.
- **Third-Party Services:** We assume that Google Maps for navigation and SMS gateways for OTP services will be reliable.

2.2 Design Constraints

- **Connectivity:** The system must work even with slow (3G) internet connections by syncing data in the background.
- **Security:** All sensitive driver documents, such as ID cards and licenses, must be encrypted for safety.
- **Latency:** Real-time updates for shared rides must happen in less than 2 seconds to keep fare calculations accurate for all passengers.

3 Key Design Decisions

- **Functional Decomposition:** We separated the "Fare-Sharing Logic" into its own process. This keeps the main booking system simple and makes it easier to update the sharing rules later.
- **Class Relationships:** We linked payments directly to rides using Composition. However, riders exist independently of any single ride (Aggregation). This ensures financial records stay organized while user accounts remain permanent.
- **Standalone Safety Flow:** We created a separate process for SOS emergency alerts. This ensures that help requests are sent immediately without being delayed by other system tasks.

4 System Architecture

The architecture connects the mobile apps for riders and drivers and the web panel for admins to a central backend. It is designed to handle up to 500 active rides and 10,000 users while remaining available 99.5% of the time.

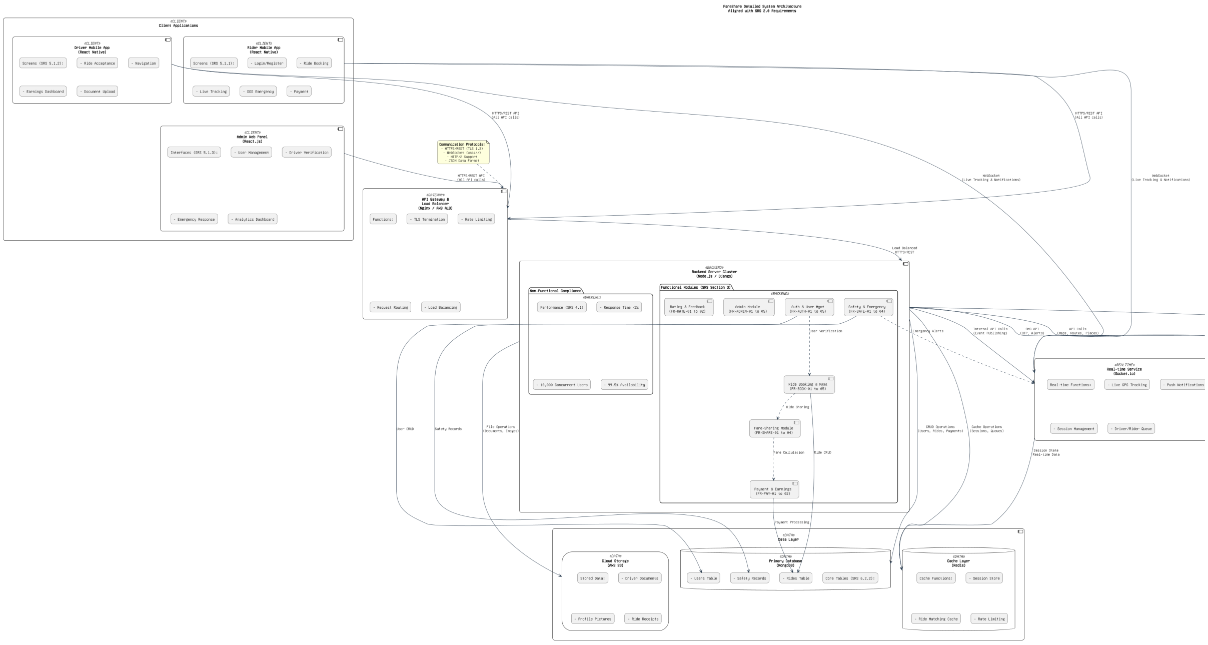

Figure 1: Detailed System Architecture (Aligned with SRS 2.0)

5 Use Case Model

The Use Case Diagram shows how different users interact with the system. It covers all functionalities for Riders, Drivers, and Administrators, including steps like phone verification and fare-sharing.

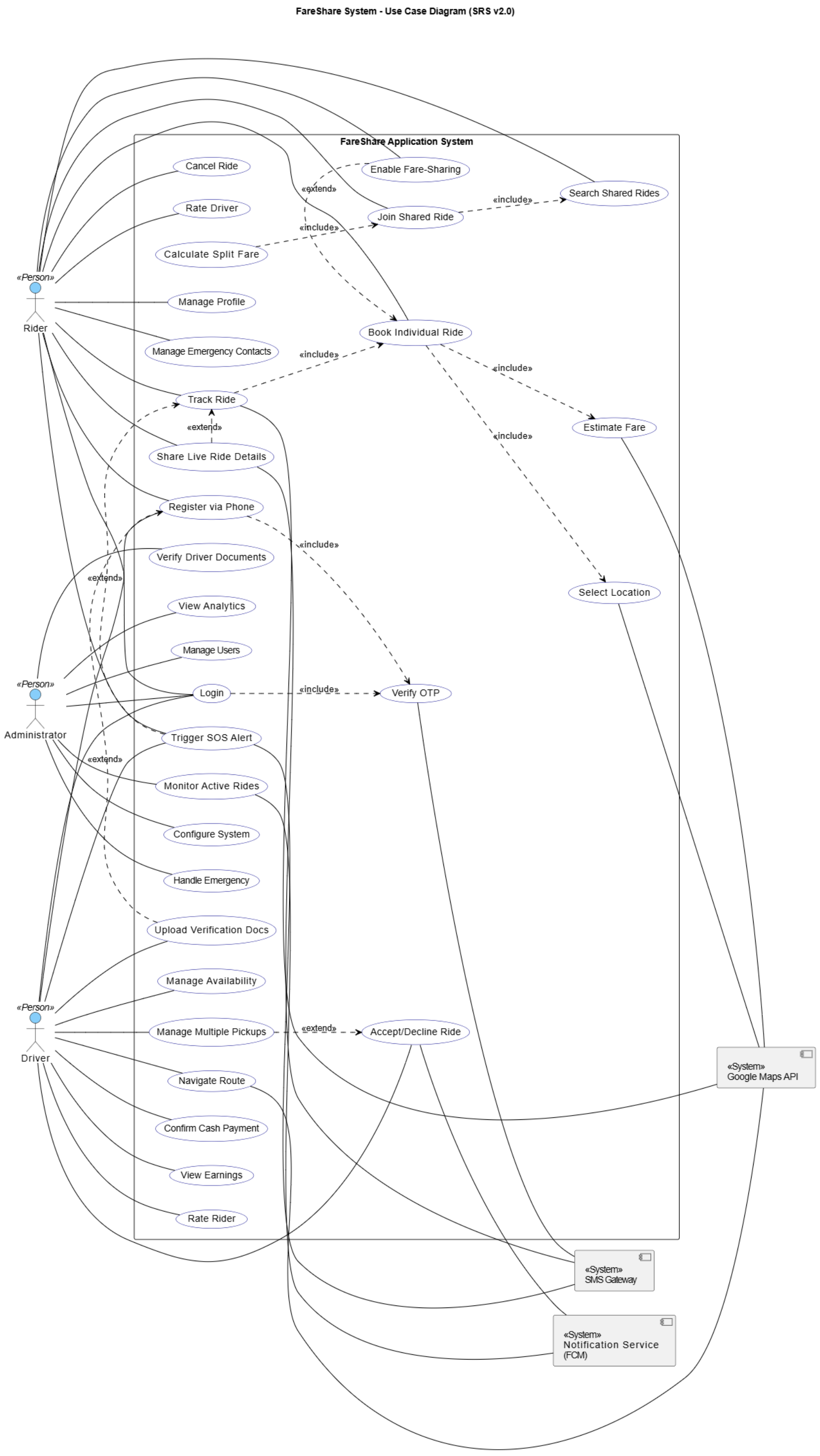

Figure 2: FareShare Use Case Diagram

6 Data Flow Diagrams

6.1 Level 0: Context Diagram

This diagram shows the system's boundaries and how it talks to external services like Google Maps and the SMS provider.

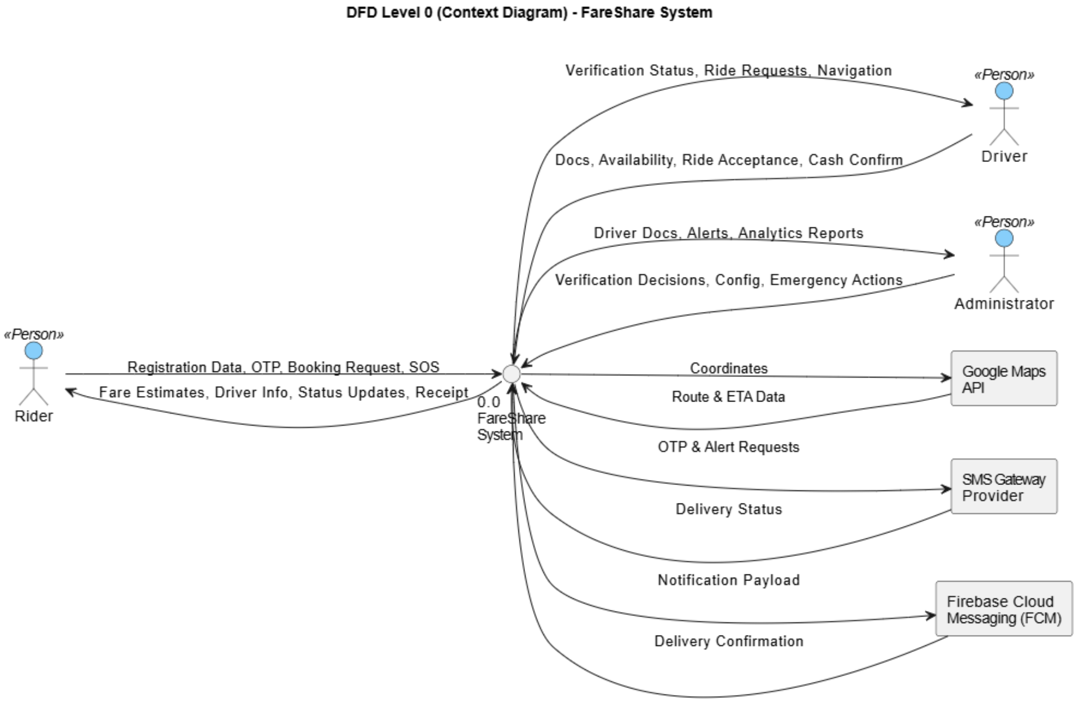
DFD Level 0 (Context Diagram) - FareShare System
Verification Status, Ride Requests, Navigation
«Person»
Driver
Docs, Availability, Ride Acceptance, Cash Confirm
Driver Docs, Alerts, Analytics Reports
«Person»
Administrator
Verification Decisions, Config, Emergency Actions
«Person»
Rider
Registration Data, OTP, Booking Request, SOS
Fare Estimates, Driver Info, Status Updates, Receipt
0.0
FareShare
System
Coordinates
Google Maps API
Route & ETA Data
OTP & Alert Requests
SMS Gateway Provider
Delivery Status
Notification Payload
Firebase Cloud Messaging (FCM)
Delivery Confirmation

Figure 3: DFD Level 0 - Context Diagram

6.2 Level 1: Major Processes

This view breaks the system into its core modules like User Auth, Ride Booking, and Fare-Sharing.

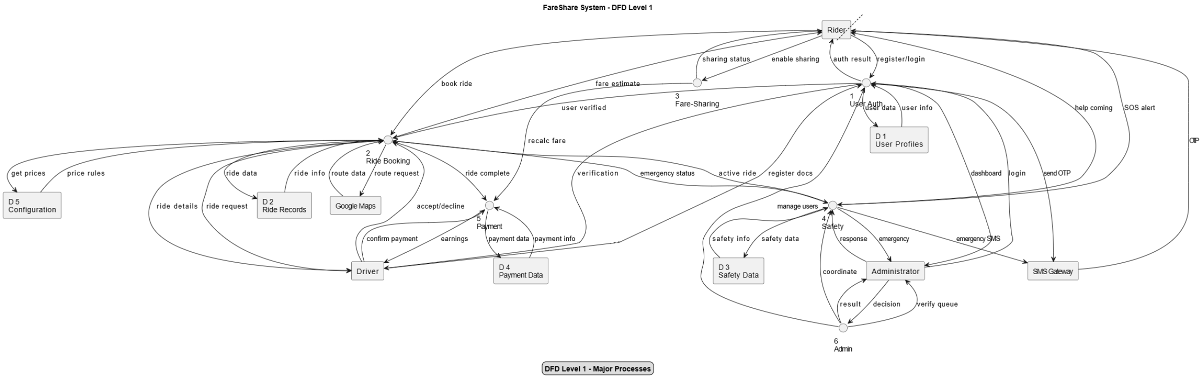

Figure 4: DFD Level 1 - Major Processes

6.3 Level 2: Process Decomposition

These diagrams break down major sections like User Authentication, Ride Booking, and Fare-Sharing into detailed steps.

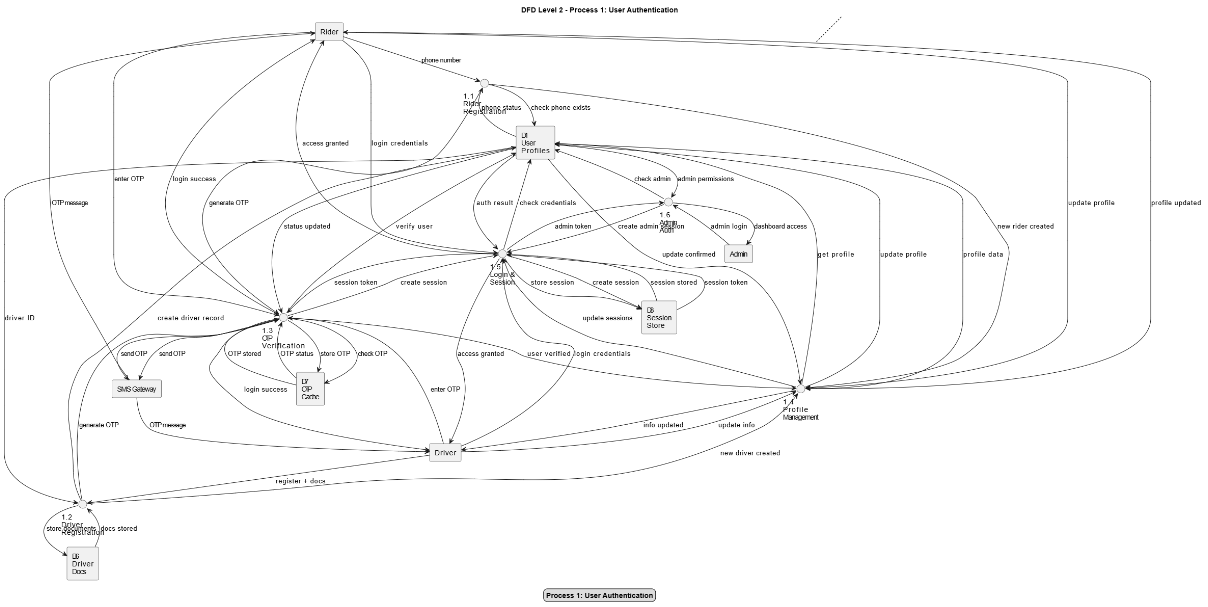

Figure 5: DFD Level 2 - Process 1: User Authentication

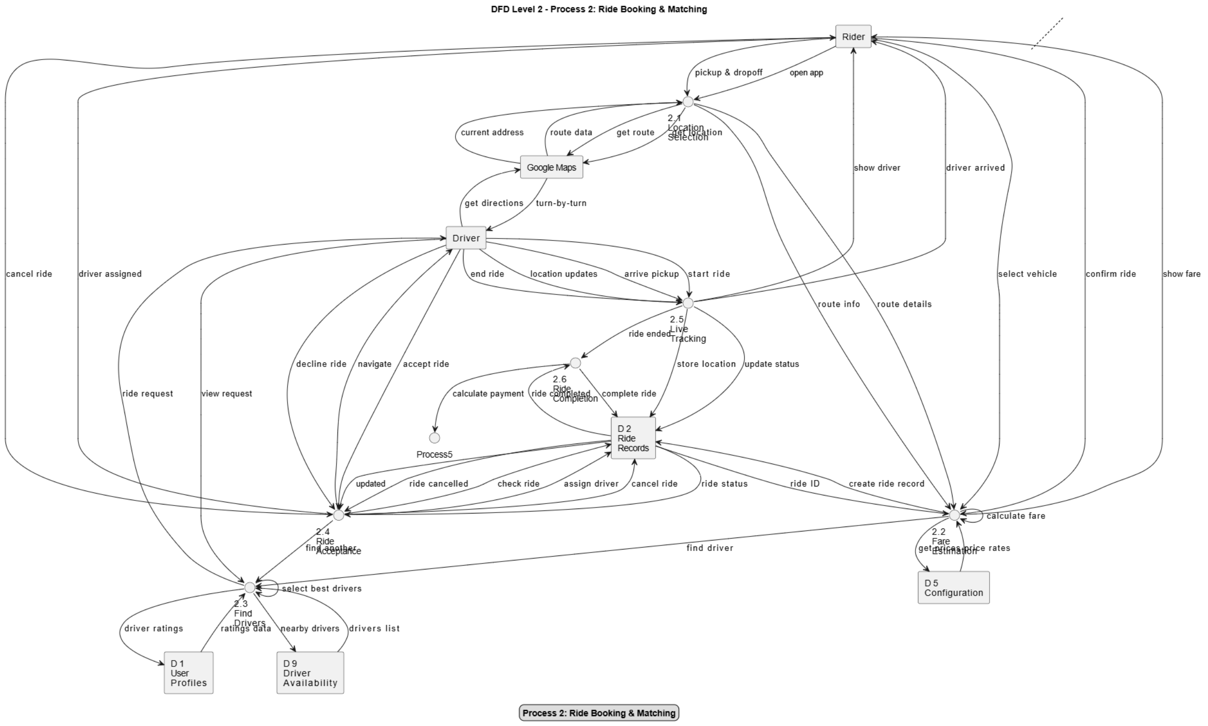

Figure 6: DFD Level 2 - Process 2: Ride Booking & Matching

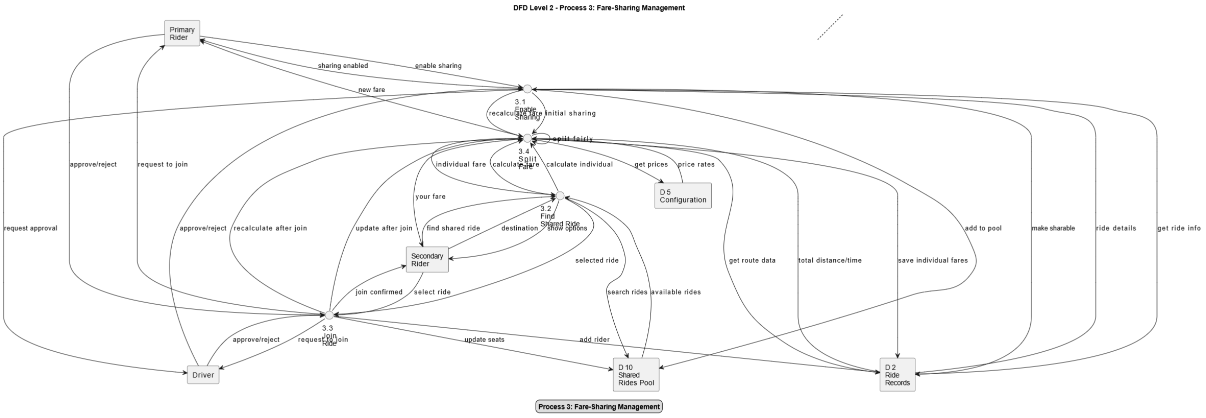

Figure 7: DFD Level 2 - Process 3: Fare-Sharing Management

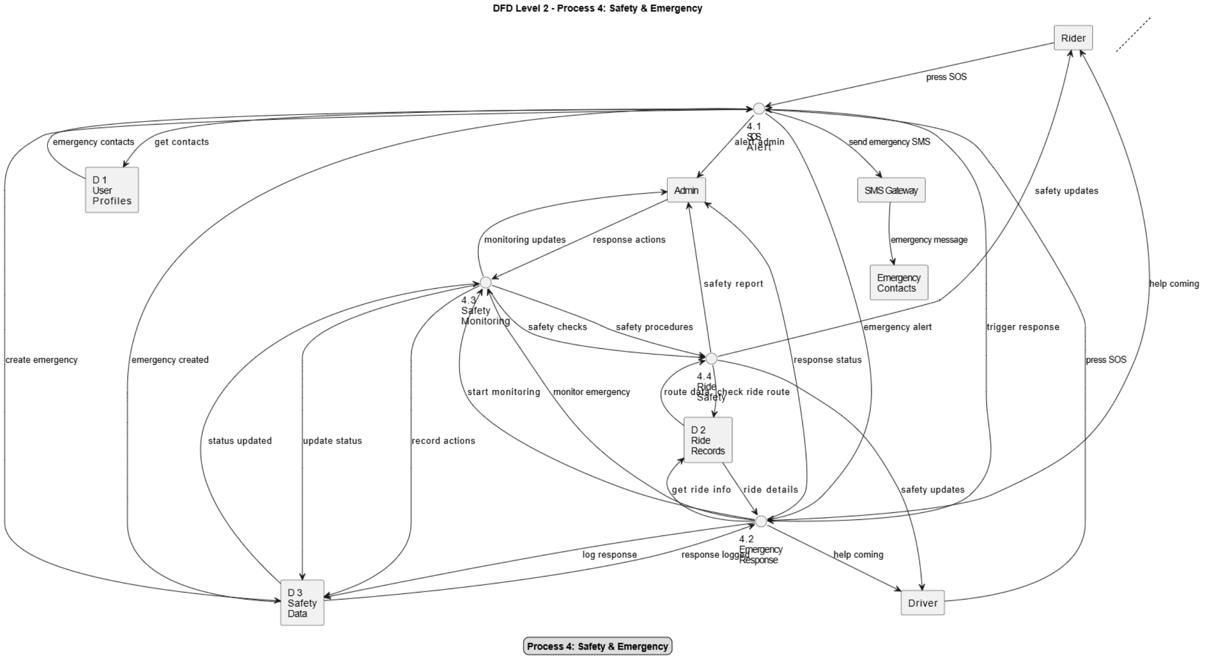

Figure 8: DFD Level 2 - Process 4: Safety & Emergency

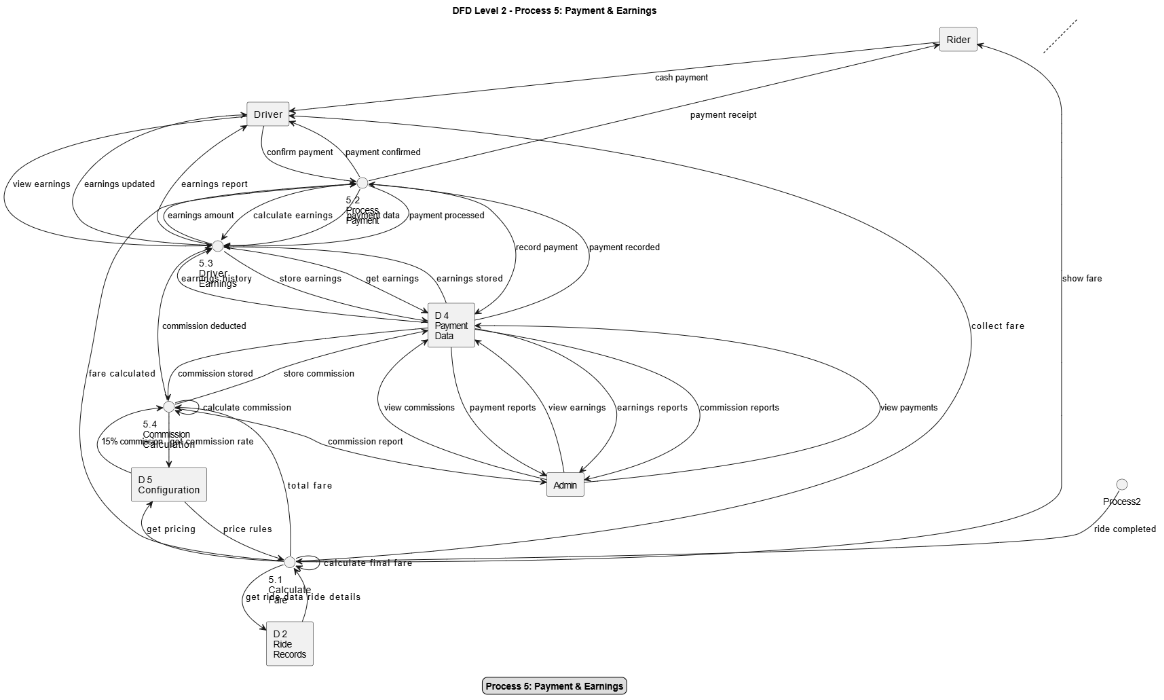

Figure 9: DFD Level 2 - Process 5: Payment & Earnings

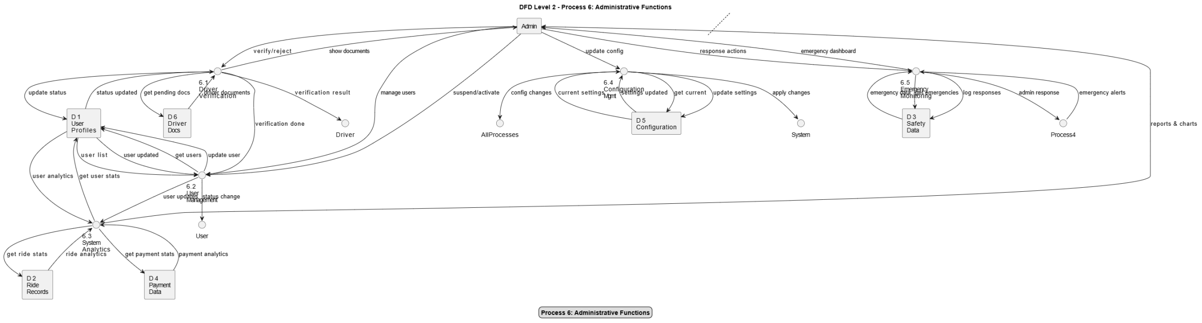

Figure 10: DFD Level 2 - Process 6: Administrative Functions

7 Sequence Diagrams

Sequence Diagrams show the order of actions for specific tasks. They explain how different parts of the system work together to complete actions like matching a rider with a driver.

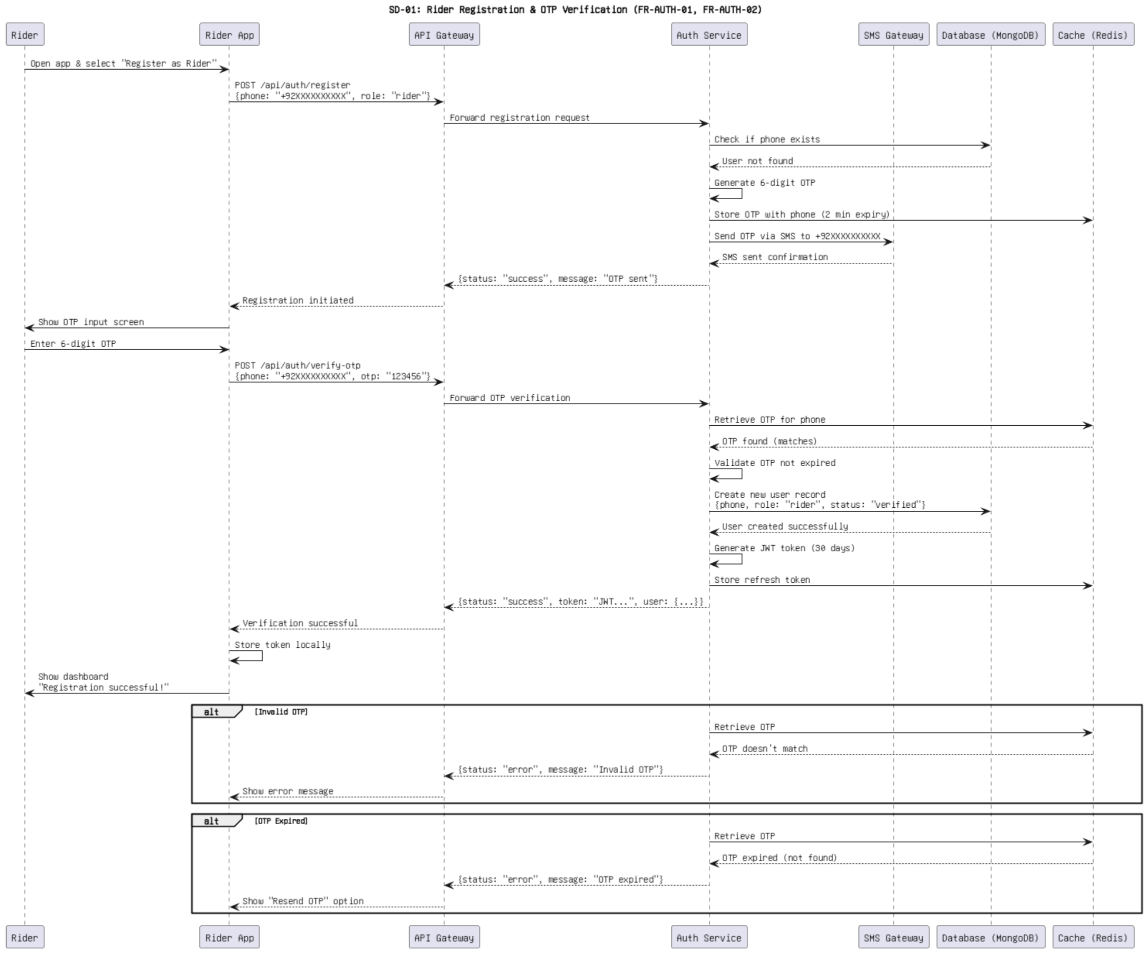

Figure 11: SD-01: Rider Registration & OTP Verification

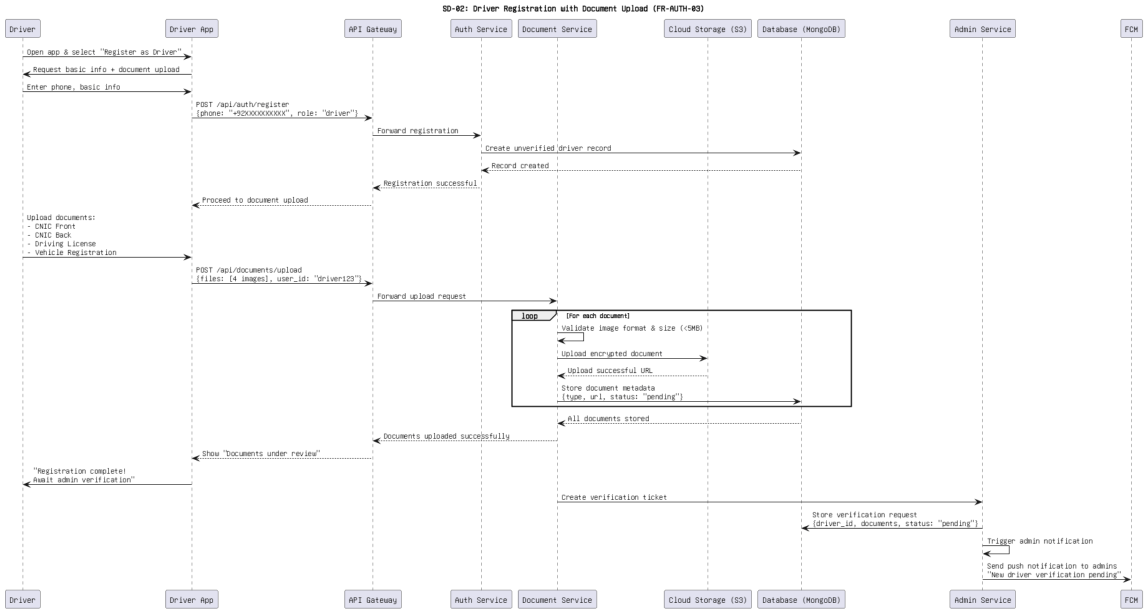
SD-02: Driver Registration with Document Upload (FR-AUTH-03)
Driver
Driver App
API Gateway
Auth Service
Document Service
Cloud Storage (S3)
Database (MongoDB)
Admin Service
FCM
Open app & select "Register as Driver"
Request basic info + document upload
Enter phone, basic info
POST /api/auth/register
{phone: "+92XXXXXXXXXX", role: "driver"}
Forward registration
Create unverified driver record
Record created
Registration successful
Proceed to document upload
Upload documents:
- CNIC Front
- CNIC Back
- Driving License
- Vehicle Registration
POST /api/documents/upload
{files: [4 images], user_id: "driver123"}
Forward upload request
loop
[For each document]
Validate image format & size (<5MB)
Upload encrypted document
Upload successful URL
Store document metadata
{type, url, status: "pending"}
All documents stored
Documents uploaded successfully
Show "Documents under review"
"Registration complete!
Await admin verification"
Create verification ticket
Store verification request
{driver_id, documents, status: "pending"}
Trigger admin notification
Send push notification to admins
"New driver verification pending"
Driver
Driver App
API Gateway
Auth Service
Document Service
Cloud Storage (S3)
Database (MongoDB)
Admin Service
FCM

Figure 12: SD-02: Driver Registration with Document Upload

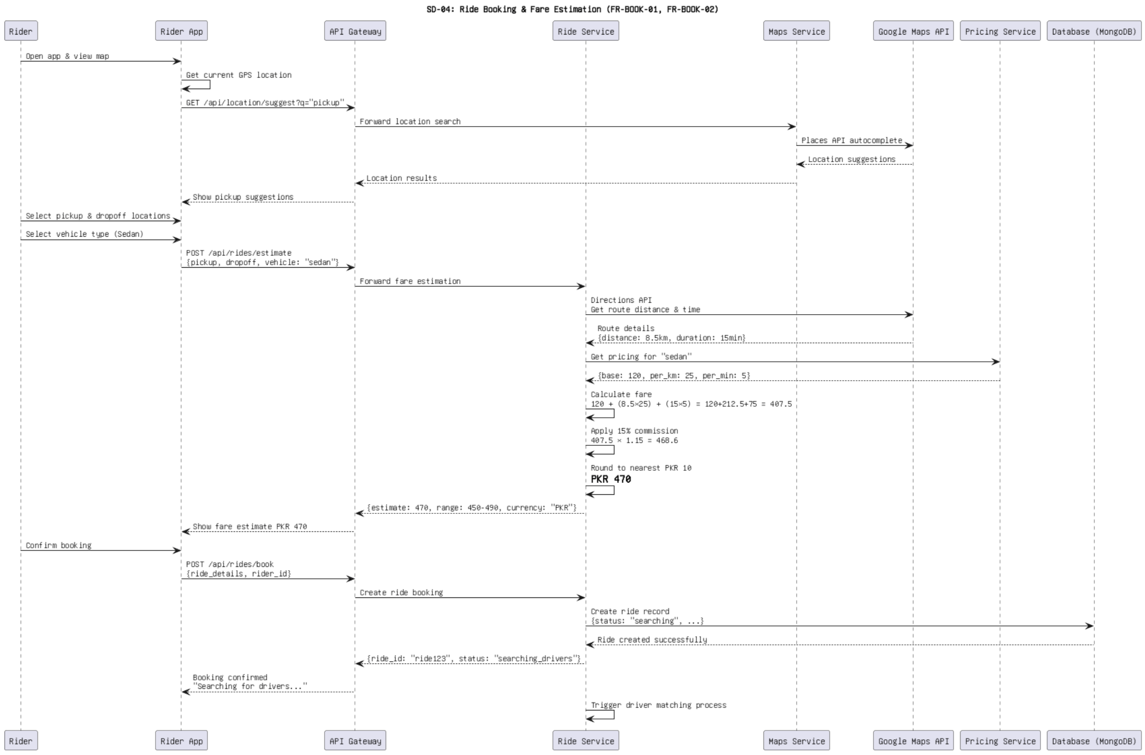

Figure 13: SD-04: Ride Booking & Fare Estimation

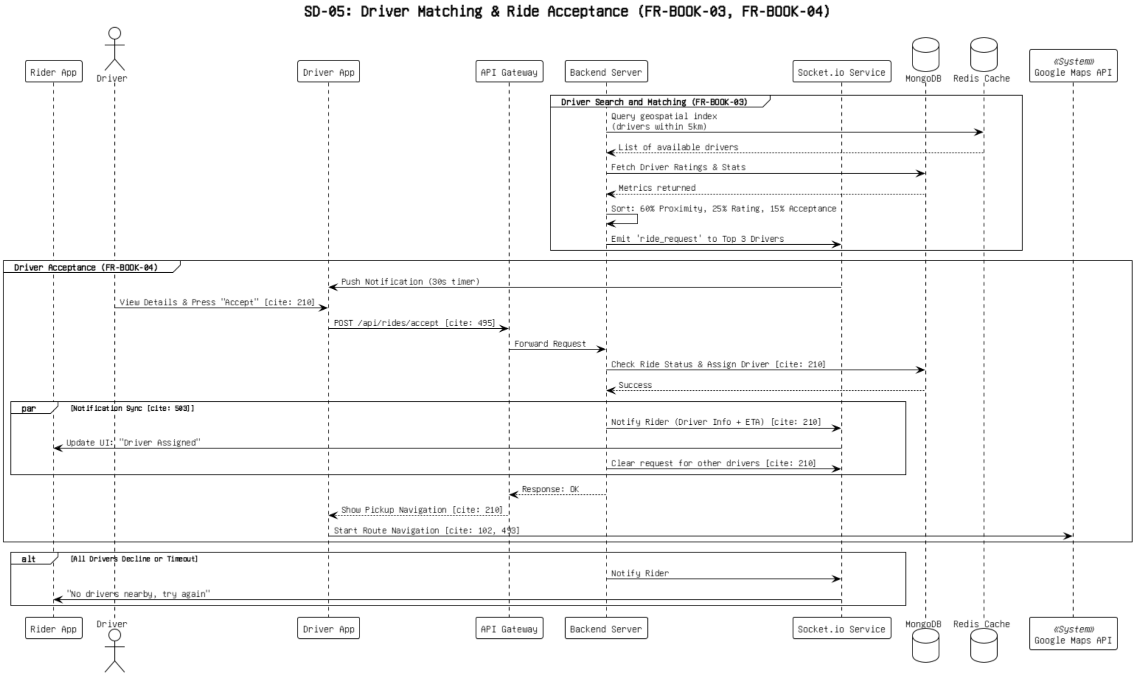
SD-05: Driver Matching & Ride Acceptance (FR-BOOK-03, FR-BOOK-04)
Rider App
Driver
Driver App
API Gateway
Backend Server
Socket.io Service
MongoDB
Redis Cache
«System»
Google Maps API
Driver Search and Matching (FR-BOOK-03)
Query geospatial index
(drivers within 5km)
List of available drivers
Fetch Driver Ratings & Stats
Metrics returned
Sort: 60% Proximity, 25% Rating, 15% Acceptance
Emit 'ride_request' to Top 3 Drivers
Driver Acceptance (FR-BOOK-04)
Push Notification (30s timer)
View Details & Press "Accept" [cite: 210]
POST /api/rides/accept [cite: 495]
Forward Request
Check Ride Status & Assign Driver [cite: 210]
Success
par
[Notification Sync [cite: 503]]
Notify Rider (Driver Info + ETA) [cite: 210]
Update UI: "Driver Assigned"
Clear request for other drivers [cite: 210]
Response: OK
Show Pickup Navigation [cite: 210]
Start Route Navigation [cite: 102, 493]
alt
[All Drivers Decline or Timeout]
Notify Rider
"No drivers nearby, try again"

Figure 14: SD-05: Driver Matching & Ride Acceptance

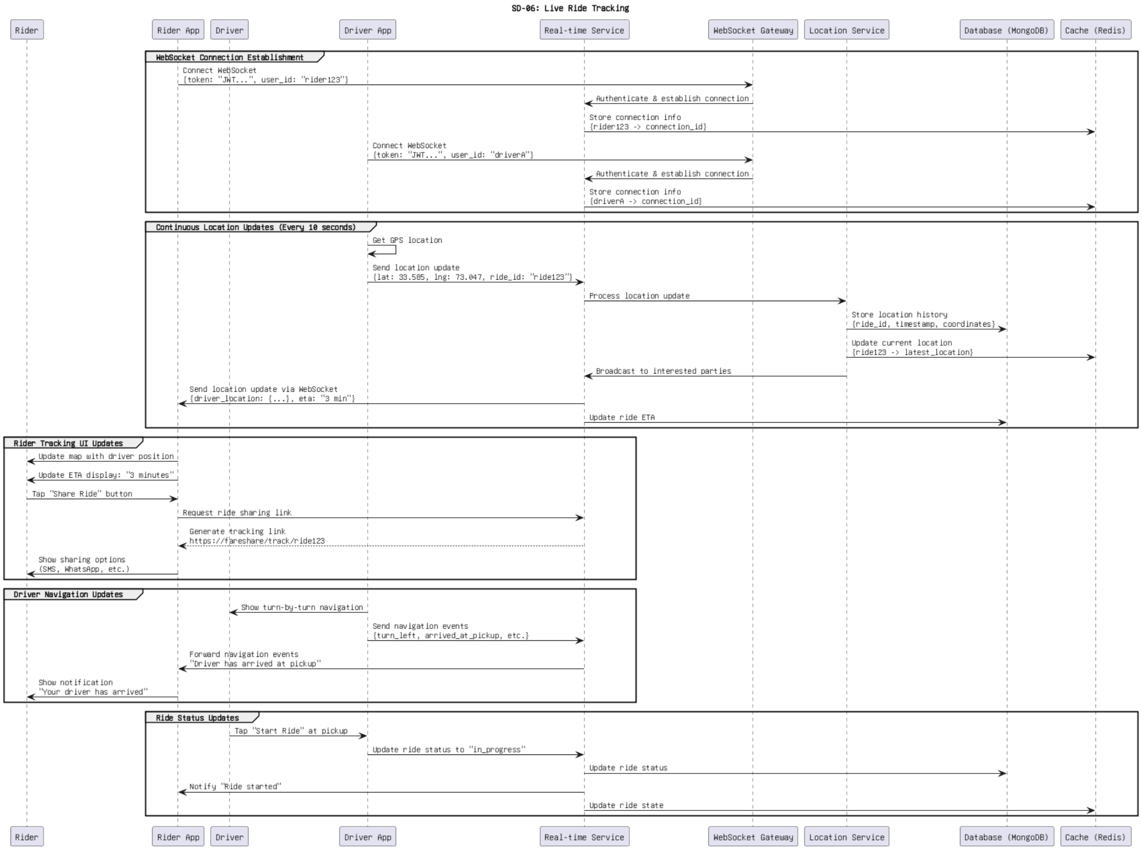
SD-06: Live Ride Tracking
Rider
Rider App
Driver
Driver App
Real-time Service
WebSocket Gateway
Location Service
Database (MongoDB)
Cache (Redis)
WebSocket Connection Establishment
Connect WebSocket
{token: "JWT...", user_id: "rider123"}
Authenticate & establish connection
Store connection info
{rider123 -> connection_id}
Connect WebSocket
{token: "JWT...", user_id: "driverA"}
Authenticate & establish connection
Store connection info
{driverA -> connection_id}
Continuous Location Updates (Every 10 seconds)
Get GPS location
Send location update
{lat: 33.585, lng: 73.047, ride_id: "ride123"}
Process location update
Store location history
{ride_id, timestamp, coordinates}
Update current location
{ride123 -> latest_location}
Broadcast to interested parties
Send location update via WebSocket
{driver_location: {...}, eta: "3 min"}
Update ride ETA
Rider Tracking UI Updates
Update map with driver position
Update ETA display: "3 minutes"
Tap "Share Ride" button
Request ride sharing link
Generate tracking link
https://fareshare/track/ride123
Show sharing options
(SMS, WhatsApp, etc.)
Driver Navigation Updates
Show turn-by-turn navigation
Send navigation events
{turn_left, arrived_at_pickup, etc.}
Forward navigation events
"Driver has arrived at pickup"
Show notification
"Your driver has arrived"
Ride Status Updates
Tap "Start Ride" at pickup
Update ride status to "in_progress"
Update ride status
Notify "Ride started"
Update ride state

Figure 15: SD-06: Live Ride Tracking

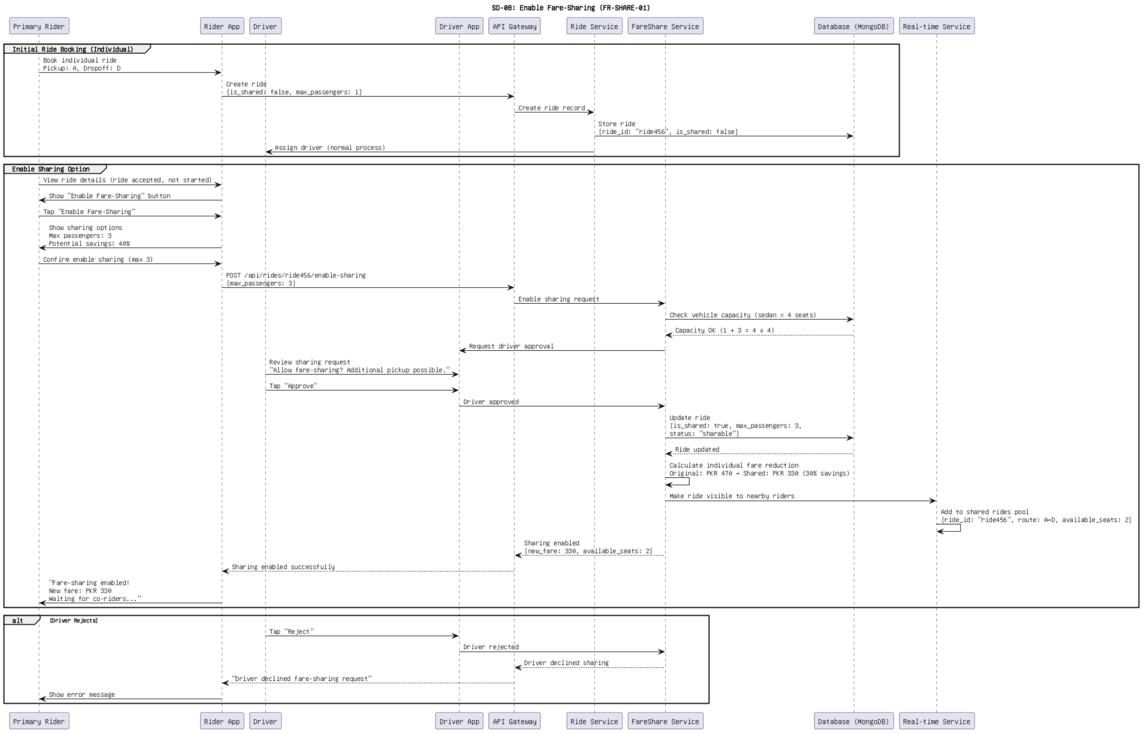
SD-08: Enable Fare-Sharing (FR-SHARE-01)
Primary Rider
Rider App
Driver
Driver App
API Gateway
Ride Service
FareShare Service
Database (MongoDB)
Real-time Service
Initial Ride Booking (Individual)
Book individual ride
Pickup: A, Dropoff: D
Create ride
{is_shared: false, max_passengers: 1}
Create ride record
Store ride
{ride_id: "ride456", is_shared: false}
Assign driver (normal process)
Enable Sharing Option
View ride details (ride accepted, not started)
Show "Enable Fare-Sharing" button
Tap "Enable Fare-Sharing"
Show sharing options
Max passengers: 3
Potential savings: 40%
Confirm enable sharing (max 3)
POST /api/rides/ride456/enable-sharing
{max_passengers: 3}
Enable sharing request
Check vehicle capacity (sedan = 4 seats)
Capacity OK (1 + 3 = 4 ≤ 4)
Request driver approval
Review sharing request
"Allow fare-sharing? Additional pickup possible."
Tap "Approve"
Driver approved
Update ride
{is_shared: true, max_passengers: 3,
status: "sharable"}
Ride updated
Calculate individual fare reduction
Original: PKR 470 → Shared: PKR 330 (30% savings)
Make ride visible to nearby riders
Add to shared rides pool
{ride_id: "ride456", route: A→D, available_seats: 2}
Sharing enabled
{new_fare: 330, available_seats: 2}
Sharing enabled successfully
"Fare-sharing enabled!
New fare: PKR 330
Waiting for co-riders..."
alt
[Driver Rejects]
Tap "Reject"
Driver rejected
Driver declined sharing
"Driver declined fare-sharing request"
Show error message

Figure 16: SD-08: Enable Fare-Sharing

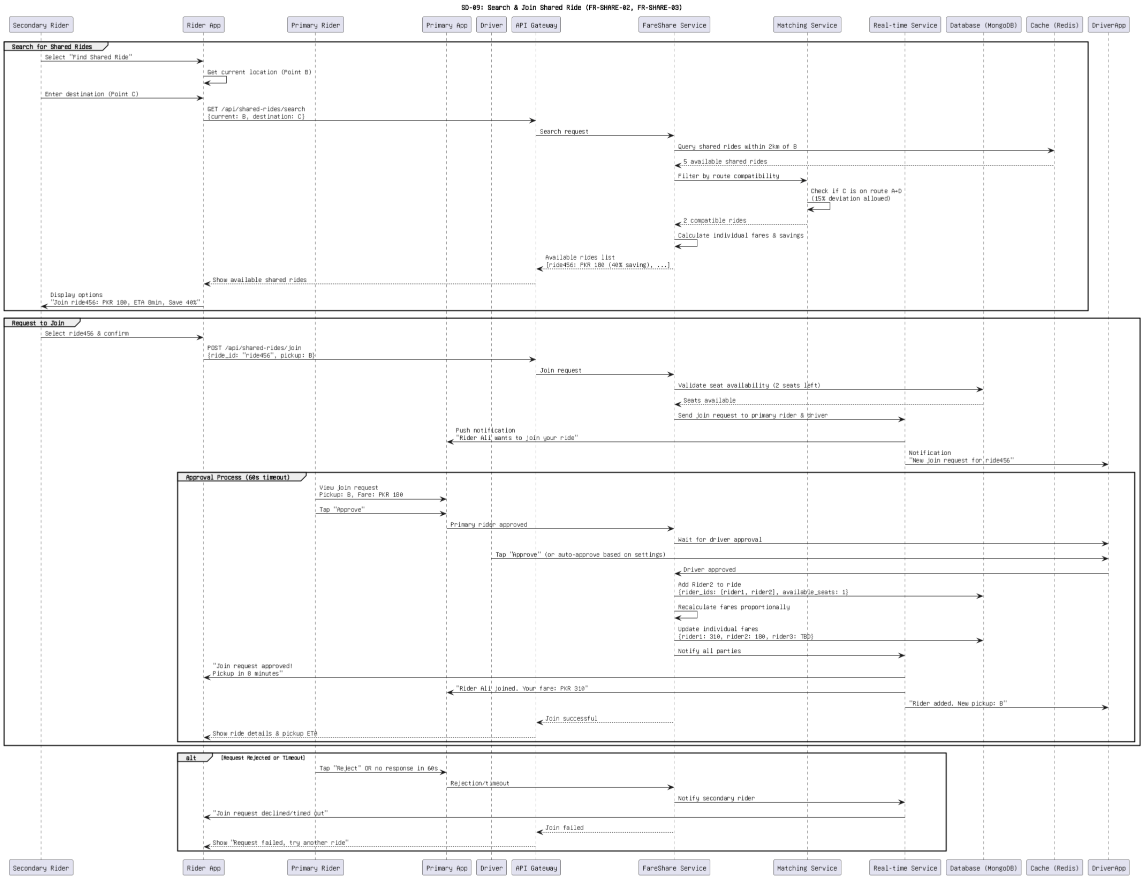
SD-09: Search & Join Shared Ride (FR-SHARE-02, FR-SHARE-03)
Secondary Rider
Rider App
Primary Rider
Primary App
Driver
API Gateway
FareShare Service
Matching Service
Real-time Service
Database (MongoDB)
Cache (Redis)
DriverApp
Search for Shared Rides
Select "Find Shared Ride"
Get current location (Point B)
Enter destination (Point C)
GET /api/shared-rides/search
{current: B, destination: C}
Search request
Query shared rides within 2km of B
5 available shared rides
Filter by route compatibility
Check if C is on route A→D
(15% deviation allowed)
2 compatible rides
Calculate individual fares & savings
Available rides list
[ride456: PKR 180 (40% saving), ...]
Show available shared rides
Display options
"Join ride456: PKR 180, ETA 8min, Save 40%"
Request to Join
Select ride456 & confirm
POST /api/shared-rides/join
{ride_id: "ride456", pickup: B}
Join request
Validate seat availability (2 seats left)
Seats available
Send join request to primary rider & driver
Push notification
"Rider Ali wants to join your ride"
Notification
"New join request for ride456"
Approval Process (60s timeout)
View join request
Pickup: B, Fare: PKR 180
Tap "Approve"
Primary rider approved
Wait for driver approval
Tap "Approve" (or auto-approve based on settings)
Driver approved
Add Rider2 to ride
{rider_ids: [rider1, rider2], available_seats: 1}
Recalculate fares proportionally
Update individual fares
{rider1: 310, rider2: 180, rider3: TBD}
Notify all parties
"Join request approved!
Pickup in 8 minutes"
"Rider Ali joined. Your fare: PKR 310"
"Rider added. New pickup: B"
Join successful
Show ride details & pickup ETA
alt
[Request Rejected or Timeout]
Tap "Reject" OR no response in 60s
Rejection/timeout
Notify secondary rider
"Join request declined/timed out"
Join failed
Show "Request failed, try another ride"

Figure 17: SD-09: Search & Join Shared Ride

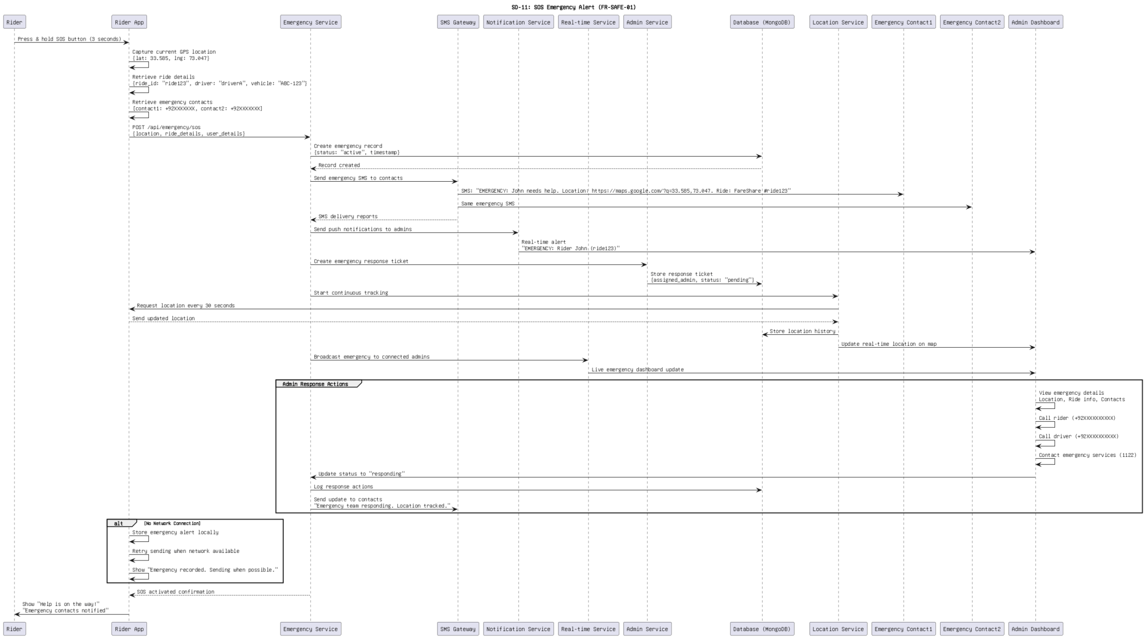
SD-11: SOS Emergency Alert (FR-SAFE-01)
Rider
Rider App
Emergency Service
SMS Gateway
Notification Service
Real-time Service
Admin Service
Database (MongoDB)
Location Service
Emergency Contact1
Emergency Contact2
Admin Dashboard
Press & hold SOS button (3 seconds)
Capture current GPS location
{lat: 33.585, lng: 73.047}
Retrieve ride details
{ride_id: "ride123", driver: "driverA", vehicle: "ABC-123"}
Retrieve emergency contacts
[contact1: +92XXXXXXX, contact2: +92XXXXXXX]
POST /api/emergency/sos
{location, ride_details, user_details}
Create emergency record
{status: "active", timestamp}
Record created
Send emergency SMS to contacts
SMS: "EMERGENCY: John needs help. Location: https://maps.google.com/?q=33.585,73.047. Ride: FareShare #ride123"
Same emergency SMS
SMS delivery reports
Send push notifications to admins
Real-time alert
"EMERGENCY: Rider John (ride123)"
Create emergency response ticket
Store response ticket
{assigned_admin, status: "pending"}
Start continuous tracking
Request location every 30 seconds
Send updated location
Store location history
Update real-time location on map
Broadcast emergency to connected admins
Live emergency dashboard update
Admin Response Actions
View emergency details
Location, Ride info, Contacts
Call rider (+92XXXXXXXXXX)
Call driver (+92XXXXXXXXXX)
Contact emergency services (1122)
Update status to "responding"
Log response actions
Send update to contacts
"Emergency team responding. Location tracked."
alt
[No Network Connection]
Store emergency alert locally
Retry sending when network available
Show "Emergency recorded. Sending when possible."
SOS activated confirmation
Show "Help is on the way!"
"Emergency contacts notified"

Figure 18: SD-11: SOS Emergency Alert Process

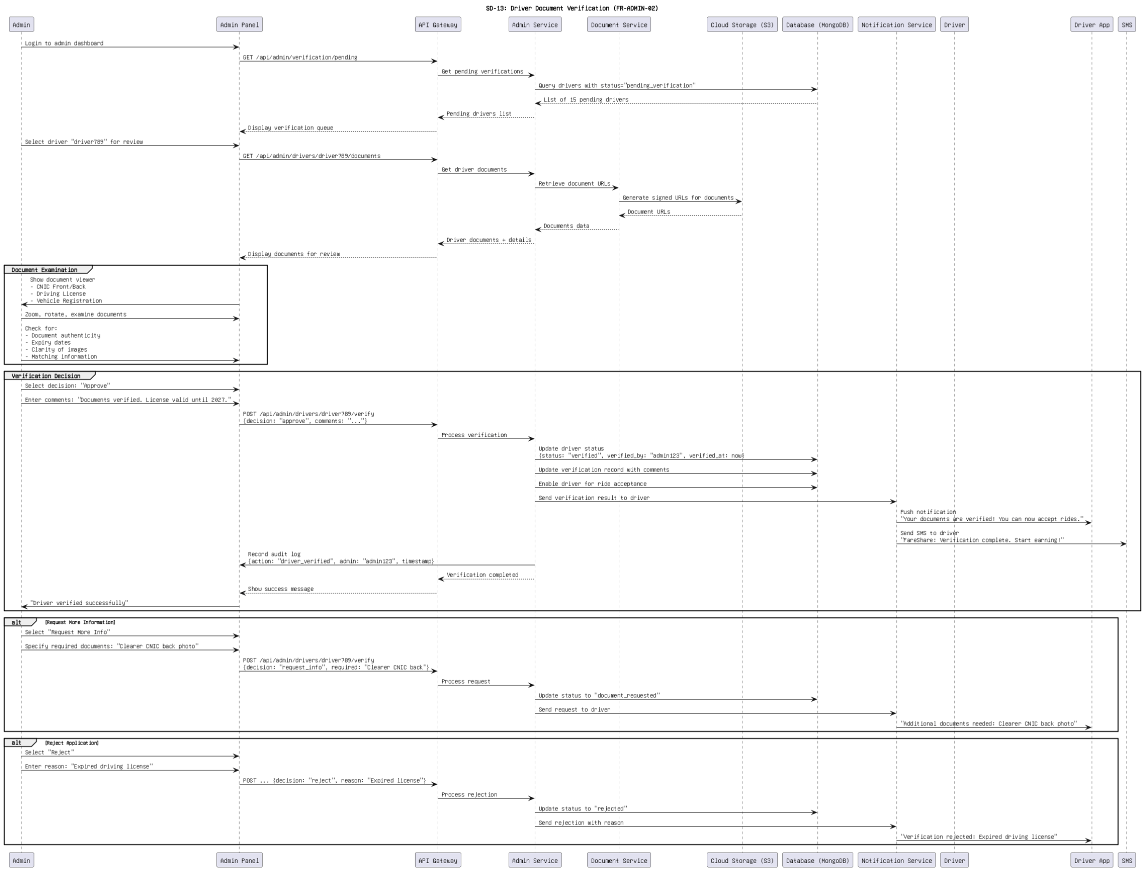
SD-13: Driver Document Verification (FR-ADMIN-02)
Admin
Admin Panel
API Gateway
Admin Service
Document Service
Cloud Storage (S3)
Database (MongoDB)
Notification Service
Driver
Driver App
SMS
Login to admin dashboard
GET /api/admin/verification/pending
Get pending verifications
Query drivers with status="pending_verification"
List of 15 pending drivers
Pending drivers list
Display verification queue
Select driver "driver789" for review
GET /api/admin/drivers/driver789/documents
Get driver documents
Retrieve document URLs
Generate signed URLs for documents
Document URLs
Documents data
Driver documents + details
Display documents for review
Document Examination
Show document viewer
- CNIC Front/Back
- Driving License
- Vehicle Registration
Zoom, rotate, examine documents
Check for:
- Document authenticity
- Expiry dates
- Clarity of images
- Matching information
Verification Decision
Select decision: "Approve"
Enter comments: "Documents verified. License valid until 2027."
POST /api/admin/drivers/driver789/verify
{decision: "approve", comments: "..."}
Process verification
Update driver status
{status: "verified", verified_by: "admin123", verified_at: now}
Update verification record with comments
Enable driver for ride acceptance
Send verification result to driver
Push notification
"Your documents are verified! You can now accept rides."
Send SMS to driver
"FareShare: Verification complete. Start earning!"
Record audit log
{action: "driver_verified", admin: "admin123", timestamp}
Verification completed
Show success message
"Driver verified successfully"
alt
[Request More Information]
Select "Request More Info"
Specify required documents: "Clearer CNIC back photo"
POST /api/admin/drivers/driver789/verify
{decision: "request_info", required: "Clearer CNIC back"}
Process request
Update status to "document_requested"
Send request to driver
"Additional documents needed: Clearer CNIC back photo"
alt
[Reject Application]
Select "Reject"
Enter reason: "Expired driving license"
POST ... {decision: "reject", reason: "Expired license"}
Process rejection
Update status to "rejected"
Send rejection with reason
"Verification rejected: Expired driving license"
Admin
Admin Panel
API Gateway
Admin Service
Document Service
Cloud Storage (S3)
Database (MongoDB)
Notification Service
Driver
Driver App
SMS

Figure 19: SD-13: Driver Document Verification

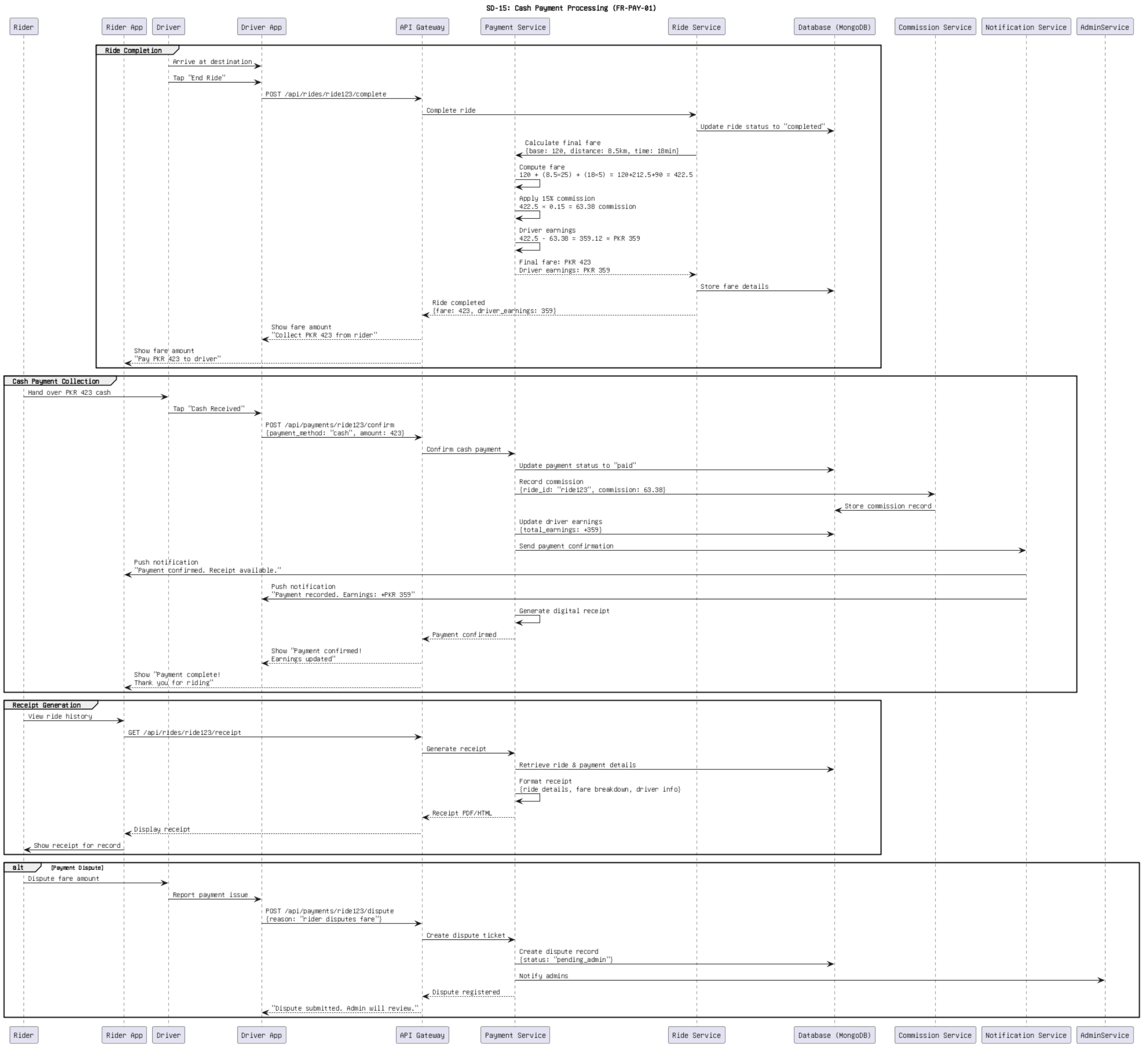

Figure 20: SD-15: Cash Payment Processing

8 Activity Diagrams

Activity diagrams represent the step-by-step operational workflows and decision logic within the system.

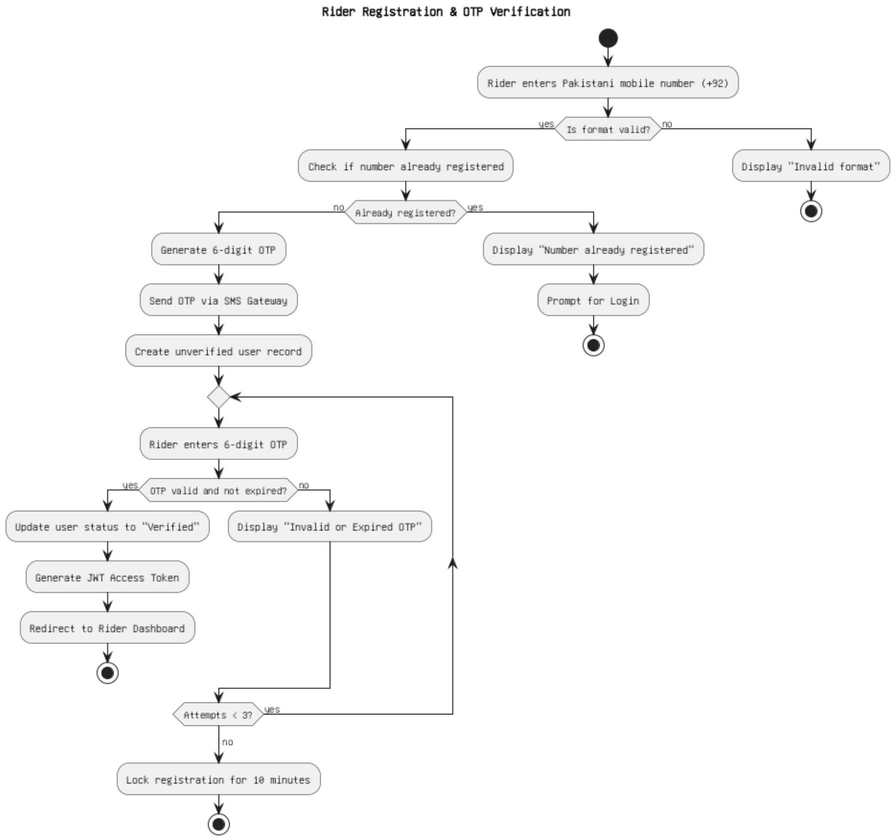
Rider Registration & OTP Verification
Rider enters Pakistani mobile number (+92)
yes
Is format valid?
no
Check if number already registered
Display "Invalid format"
no
Already registered?
yes
Generate 6-digit OTP
Display "Number already registered"
Send OTP via SMS Gateway
Prompt for Login
Create unverified user record
Rider enters 6-digit OTP
yes
OTP valid and not expired?
no
Update user status to "Verified"
Display "Invalid or Expired OTP"
Generate JWT Access Token
Redirect to Rider Dashboard
Attempts < 3?
yes
no
Lock registration for 10 minutes

Figure 21: Workflow: Rider Registration & OTP Verification

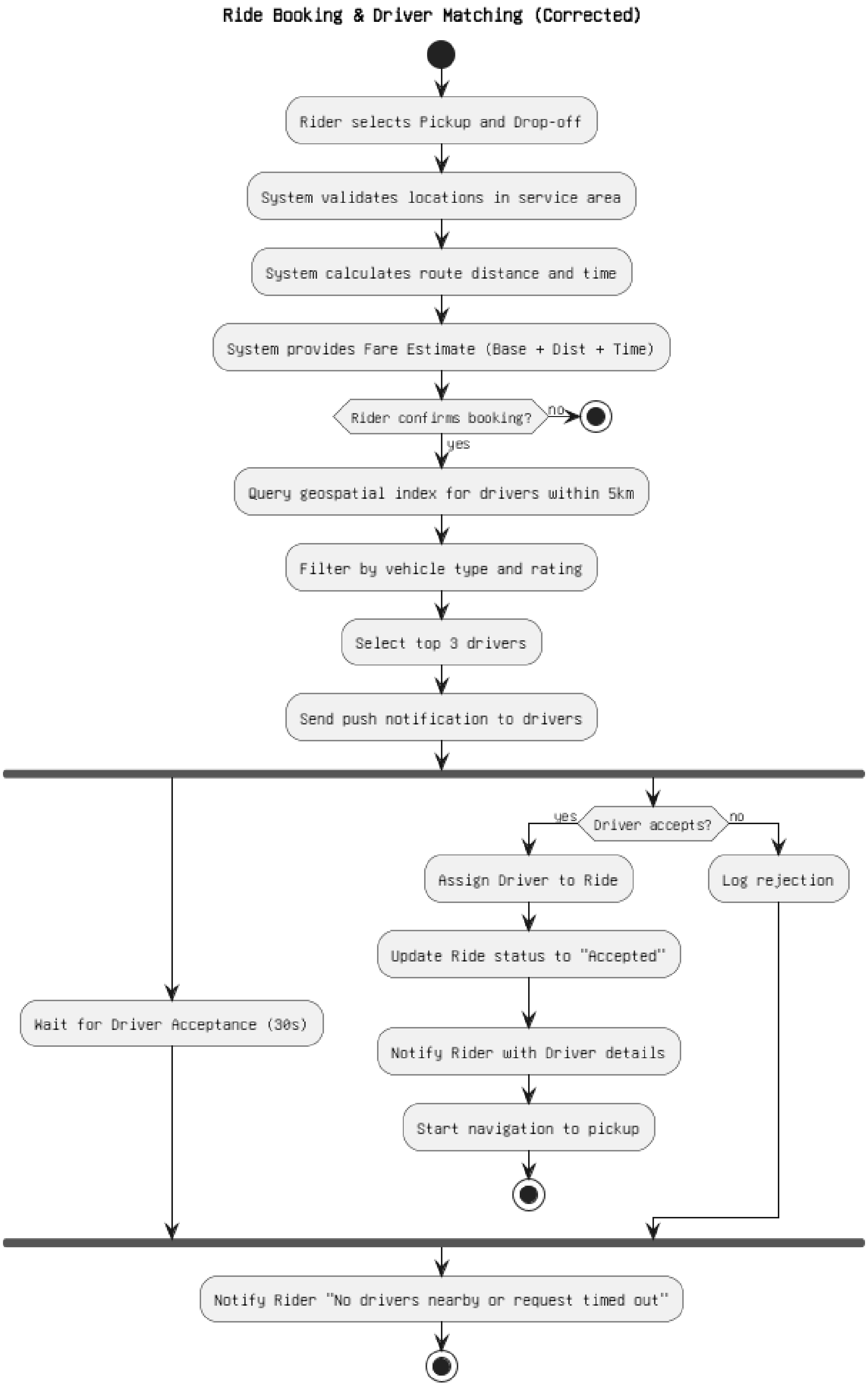
Ride Booking & Driver Matching (Corrected)
Rider selects Pickup and Drop-off
System validates locations in service area
System calculates route distance and time
System provides Fare Estimate (Base + Dist + Time)
Rider confirms booking?
no
yes
Query geospatial index for drivers within 5km
Filter by vehicle type and rating
Select top 3 drivers
Send push notification to drivers
Driver accepts?
yes
no
Assign Driver to Ride
Log rejection
Update Ride status to "Accepted"
Wait for Driver Acceptance (30s)
Notify Rider with Driver details
Start navigation to pickup
Notify Rider "No drivers nearby or request timed out"

Figure 22: Workflow: Ride Booking & Driver Matching

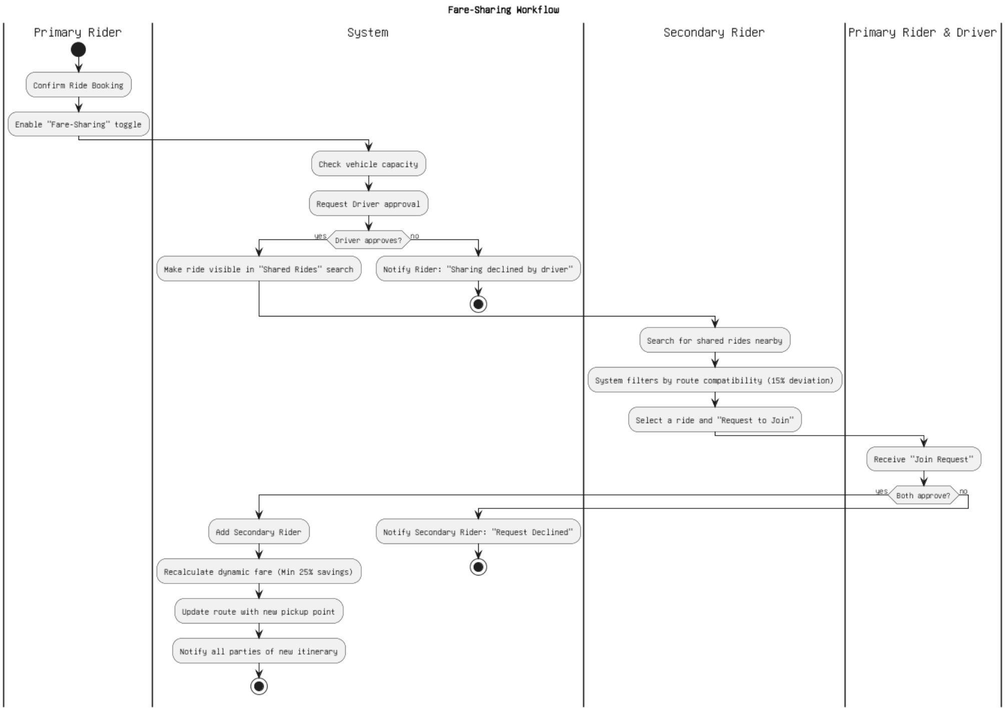
Fare-Sharing Workflow
Primary Rider
System
Secondary Rider
Primary Rider & Driver
Confirm Ride Booking
Enable "Fare-Sharing" toggle
Check vehicle capacity
Request Driver approval
yes
Driver approves?
no
Make ride visible in "Shared Rides" search
Notify Rider: "Sharing declined by driver"
Search for shared rides nearby
System filters by route compatibility (15% deviation)
Select a ride and "Request to Join"
Receive "Join Request"
yes
Both approve?
no
Add Secondary Rider
Notify Secondary Rider: "Request Declined"
Recalculate dynamic fare (Min 25% savings)
Update route with new pickup point
Notify all parties of new itinerary

Figure 23: Workflow: Fare-Sharing Process

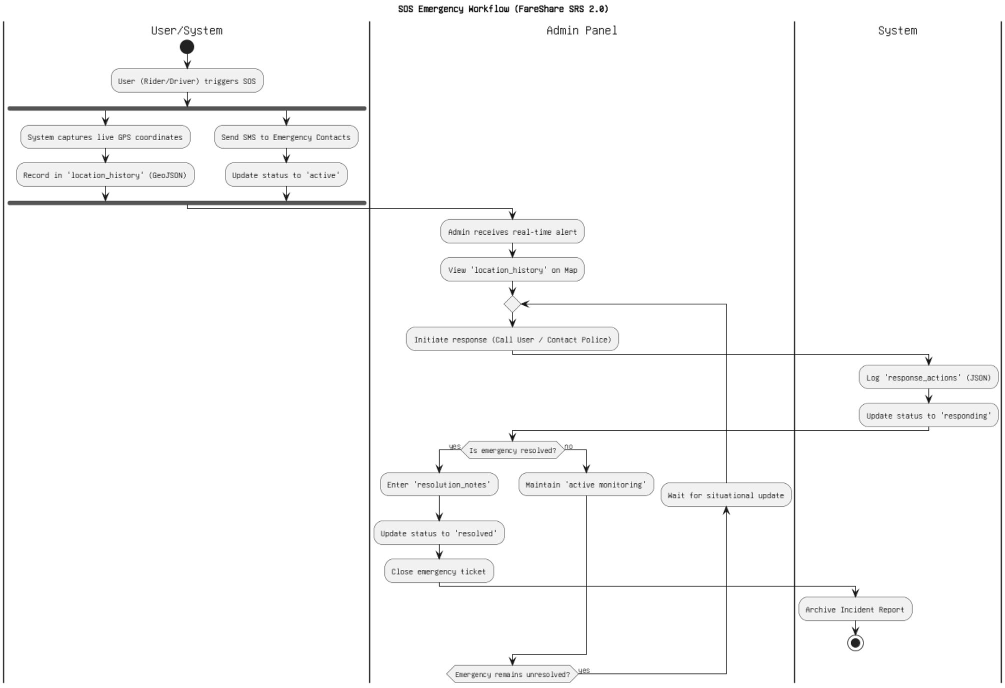
SOS Emergency Workflow (FareShare SRS 2.0)
User/System
Admin Panel
System
User (Rider/Driver) triggers SOS
System captures live GPS coordinates
Send SMS to Emergency Contacts
Record in 'location_history' (GeoJSON)
Update status to 'active'
Admin receives real-time alert
View 'location_history' on Map
Initiate response (Call User / Contact Police)
Log 'response_actions' (JSON)
Update status to 'responding'
yes
Is emergency resolved?
no
Enter 'resolution_notes'
Maintain 'active monitoring'
Wait for situational update
Update status to 'resolved'
Close emergency ticket
Archive Incident Report
Emergency remains unresolved?
yes

Figure 24: Workflow: SOS Emergency Alert

9 Structural Diagrams

9.1 Class Diagram

The Class Diagram reflects the static structure of the system, including classes, attributes, and relationships.

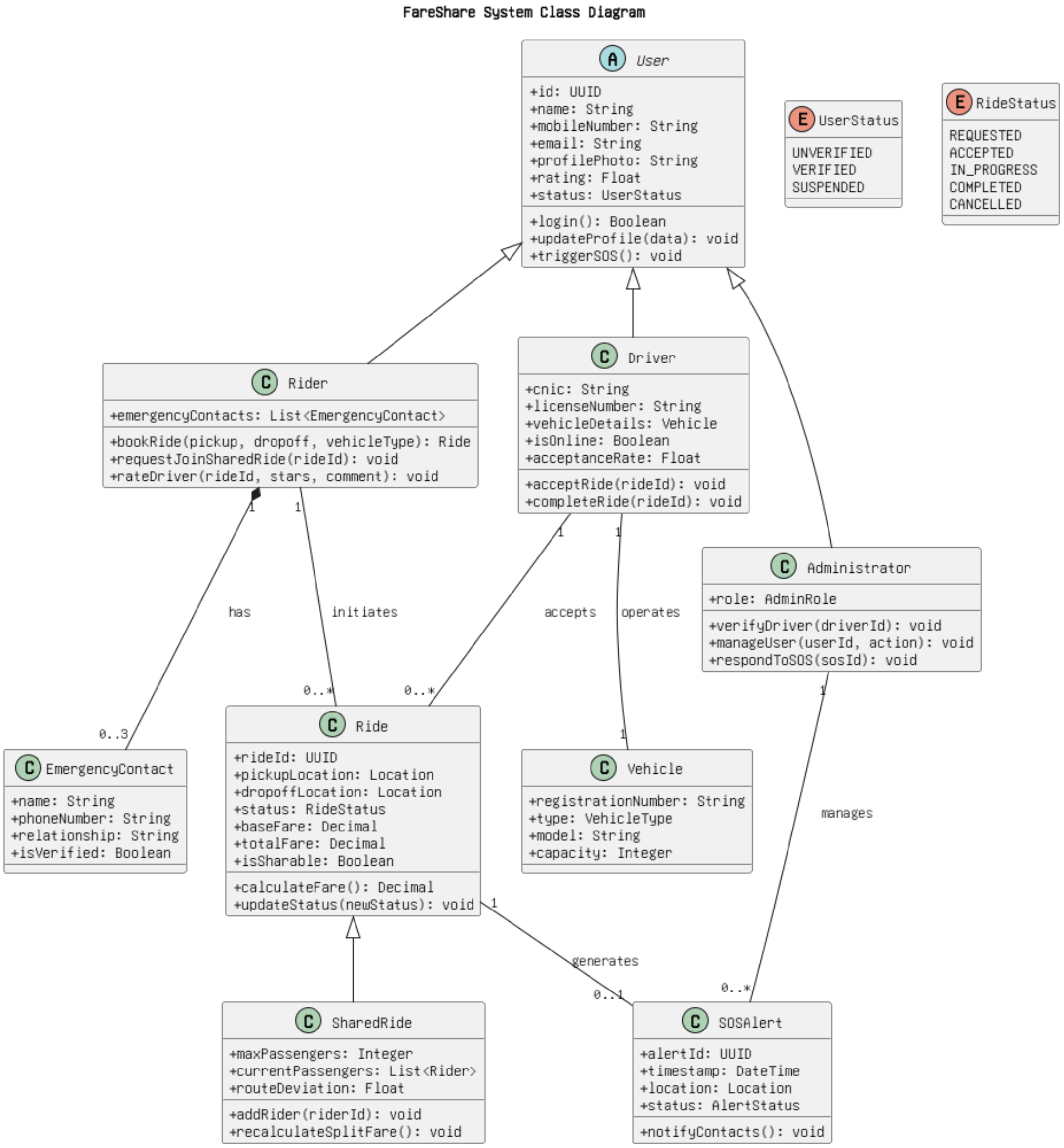

Figure 25: FareShare System Class Diagram

9.2 Component Diagram

This illustrates the high-level system components and how they depend on each other.

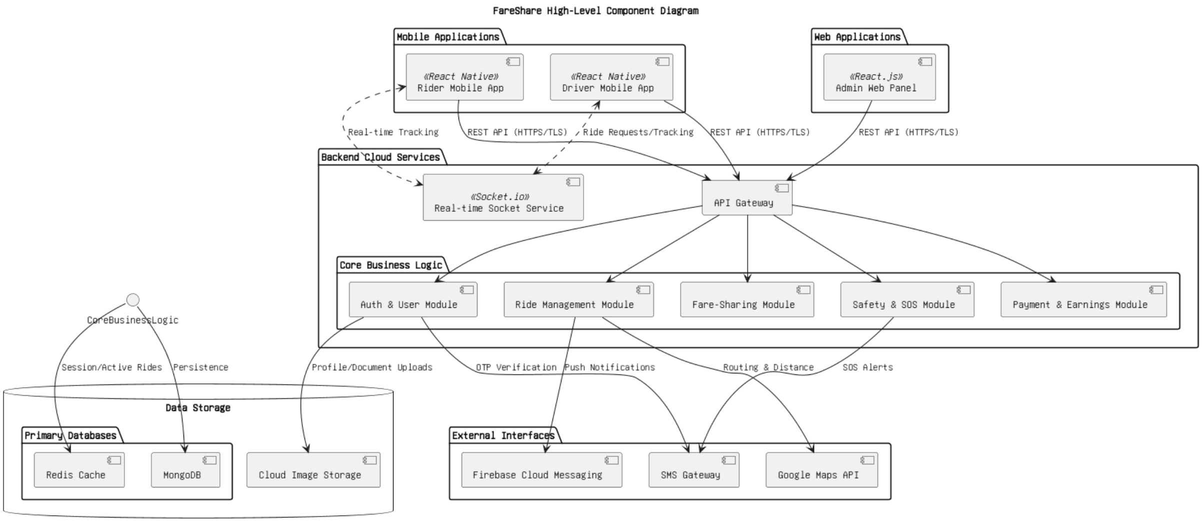

Figure 26: FareShare High-Level Component Diagram

10 Requirements-Design Traceability Table

This table connects our functional requirements to our designs, ensuring that every feature requested in the SRS is properly addressed.

Table 1: Requirements-to-Design Traceability Matrix

Req ID	**Requirement**	**Use Case**	**DFD Process**	**Logic Module**
FR-AUTH-01	User Registration	Register User	P1.1 Verify Phone	UserController
FR-AUTH-03	Driver Onboarding	Register Driver	P1.2 Upload Docs	DocManager
FR-BOOK-01	Location Selection	Book Ride	P2.1 Fetch Maps	MapService
FR-BOOK-02	Fare Estimation	Get Fare	P2.2 Calc Cost	FareCalculator
FR-BOOK-03	Driver Matching	Match Driver	P2.3 Query Driver	MatchingEngine
FR-SHARE-01	Share Initiation	Initiate Share	P3.1 Set Sharable	Ride Logic
FR-SHARE-04	Dynamic Fare Split	Split Fare	P3.4 Compute	Split Engine
FR-SAFE-01	SOS Emergency	Trigger SOS	P4.1 Send Alert	AlertManager
FR-ADMIN-02	Doc Verification	Verify Driver	P5.2 Verify Docs	AdminPanel
FR-PAY-01	Cash Payment	Complete Ride	P6.1 Record	Transaction

11 Project Links

The following links provide access to the complete project assets:

- **GitHub Repository:** https://github.com/Naveed83067/FareShare-Ride-Sharing-System
- **Figma Interactive Prototype:** https://www.figma.com/design/X1RGz517AJMZ1P3mxR0kug Prototype?node-id=0-1t=vAqoUkrh7UyChOHb-1

12 Conclusion

This design report has translated all requirements from the SRS into a technical plan. By using clear architecture, detailed data flows, and interactive prototypes, we have ensured that the system is ready for the development phase. All designs have been verified against the original project goals to ensure quality and safety.