TaxiTap System
Service Contracts Documentation

Software Requirements Specification

Contents

1 Introduction

This document provides service contracts for all major services within the TaxiTap ride-sharing system. Each service contract defines a clear and consistent description of how the service can be used, including its purpose, required inputs, expected outputs, and interactions with other system components.

Service contracts serve as API specifications that outline the interface and behavior expectations for each service, ensuring clear communication between different parts of the system and external integrators.

2 User Account Management Service

The User Account Management service handles all user-related operations including registration, authentication, profile management, and role switching functionality.

2.1 signUpSMS Function

Field	Description
What the service does	Registers a new user account in the TaxiTap system using SMS-based verification. Creates user profile with provided information and sends SMS verification code to the provided phone number.
Inputs required	• `phoneNumber` (string) - Valid phone number in international format • `name` (string) - User's full name (2-0 characters) • `password` (string) - Secure password (minimum 8 characters) • `accountType` ("passenger" — "driver" — "both") - Account role type • `email` (optional string) - Valid email address • `age` (optional number) - User age (must be 18 or older)
Outputs returned	• `userId` (Id¡"taxiTap_users"¿) - Unique user identifier • `verificationStatus` (boolean) - SMS verification status • `sessionToken` (string) - Authentication token for session • `userProfile` (object) - Created user profile data • `success` (boolean) - Operation success indicator • `errorMessage` (optional string) - Error details if operation fails
Interactions with other systems/components	• Database Service - Stores user account data • Authentication Service - Authenticates user information • Validation Service - Validates input parameters • Notification System - Sends welcome notification

2.2 loginSMS Function

Field	Description

What the service does	Authenticates existing users using phone number and password combination. Validates credentials and establishes authenticated session with appropriate permissions based on user's account type.
Inputs required	• `phoneNumber` (string) - Registered phone number • `password` (string) - User password
Outputs returned	• `userId` (Id¡"taxiTap_users"¿) - Authenticated user identifier • `sessionToken` (string) - Authentication token for session • `userProfile` (object) - User profile information • `activeRole` ("passenger" — "driver") - Current active role • `permissions` (array) - User permission set • `success` (boolean) - Authentication result • `errorMessage` (optional string) - Error details if authentication fails
Interactions with other systems/components	• Database Service - Retrieves and validates user credentials • Authentication Service - Generates and manages session tokens • Security Service - Logs authentication attempts • User Profile Service - Retrieves user profile data

2.3 switchActiveRole Function

Field	Description
What the service does	Allows users with multiple account types to switch between passenger and driver roles. Updates active permissions and interface preferences based on selected role.
Inputs required	• `userId` (Id¡"taxiTap_users"¿) - Authenticated user identifier • `newRole` ("passenger" — "driver") - Target role to switch to
Outputs returned	• `activeRole` ("passenger" — "driver") - Updated active role • `permissions` (array) - Updated permission set • `interfaceConfig` (object) - Role-specific UI configuration • `success` (boolean) - Role switch result • `errorMessage` (optional string) - Error details if switch fails
Interactions with other systems/components	• Authentication Service - Updates session permissions • Database Service - Updates user's active role status • UI Configuration Service - Provides role-specific interface settings • Activity Tracking Service - Logs role switch event

2.4 updateUserProfile Function

Field	Description

What the service does	Updates user profile information including personal details, contact information, and emergency contacts. Validates all changes and maintains data integrity across the system.
Inputs required	• `userId` (Id¡"taxiTap_users"¿) - User identifier for profile update • `name` (string) - Updated full name • `phoneNumber` (string) - Updated phone number • `email` (optional string) - Updated email address • `profilePicture` (optional string) - Profile image URL or base64 data • `emergencyContact` (optional object) - Emergency contact details: – `name` (string) - Contact person's name – `phoneNumber` (string) - Contact phone number – `relationship` (string) - Relationship to user
Outputs returned	• `updatedProfile` (object) - Complete updated user profile • `changedFields` (array) - List of fields that were modified • `validationErrors` (array) - Any validation issues encountered • `success` (boolean) - Update operation result • `errorMessage` (optional string) - Error details if update fails
Interactions with other systems/components	• Database Service - Persists profile changes • Validation Service - Validates all input data • Image Processing Service - Processes profile picture uploads • Notification System - Notifies user of successful profile updates • Security Service - Logs profile modification events

3 Ride Request System Service

The Ride Request System service manages the complete ride request lifecycle from initial passenger request through driver matching, ride execution, and completion. This service consists of multiple handler functions that coordinate real-time interactions between passengers and drivers.

3.1 requestRideHandler Function

Field	Description
What the service does	Initiates a new ride request from a passenger. Creates a ride record, calculates fare estimates, searches for available drivers in the vicinity, and broadcasts the ride request to eligible drivers. Manages the initial matching process and sets up real-time tracking.

Inputs required	• `passengerId` (string) - Unique identifier of the requesting passenger
	• `driverId` (string) - Specific driver identifier if passenger has a preference
	• `startLocation` (object) - Pickup location details:
	  – `coordinates` (object) - Latitude and longitude coordinates
	  – `address` (string) - Human-readable pickup address
	• `endLocation` (object) - Destination location details:
	  – `coordinates` (object) - Latitude and longitude coordinates
	  – `address` (string) - Human-readable destination address
	• `estimatedFare` (optional number) - Pre-calculated fare estimate
	• `estimatedDistance` (optional number) - Pre-calculated distance in kilometers
	• `estimatedDuration` (optional number) - Pre-calculated duration in minutes

Outputs returned	• `rideId` (string) - Unique identifier for the created ride request • `requestStatus` ("created" — "searching" — "pending_driver_response") - Current request status • `estimatedMatchTime` (number) - Expected time to find a driver in minutes • `nearbyDrivers` (array) - List of available drivers in the area • `fareDetails` (object) - Detailed fare breakdown including base fare, taxes, and fees • `routePreview` (object) - Basic route information and map data • `success` (boolean) - Request creation result • `errorMessage` (optional string) - Error details if request fails
Interactions with other systems/components	• Database Service - Creates ride record and retrieves passenger details • Driver Matching Service - Finds and filters available drivers • Routes Service - Calculates route and fare estimates • Notification System - Sends ride requests to drivers • Real-time Tracking Service - Initializes location tracking • Geolocation Service - Validates and processes location data • Fare Calculation Service - Computes accurate pricing

3.2 acceptRideHandler Function

Field	Description
What the service does	Processes a driver's acceptance of a ride request. Updates ride status, establishes the passenger-driver connection, cancels notifications to other drivers, and initiates the active ride phase with real-time coordination.

Inputs required	• `rideId` (string) - Unique identifier of the ride request being accepted • `driverId` (string) - Unique identifier of the driver accepting the ride
Outputs returned	• `rideStatus` ("accepted" — "driver_en_route") - Updated ride status • `driverDetails` (object) - Complete driver profile and vehicle information • `estimatedArrival` (datetime) - Expected driver arrival time at pickup location • `trackingInfo` (object) - Real-time tracking details for passenger • `contactInfo` (object) - Secure communication channels between passenger and driver • `ridePin` (string) - Security PIN for ride verification • `success` (boolean) - Acceptance processing result • `errorMessage` (optional string) - Error details if acceptance fails
Interactions with other systems/components	• Database Service - Updates ride status and driver assignment • Notification System - Notifies passenger of driver acceptance and cancels other driver notifications • Real-time Tracking Service - Activates live location sharing • Driver Availability Service - Updates driver status to busy • Communication Service - Establishes secure passenger-driver communication • Route Optimization Service - Provides optimal route to pickup location

3.3 cancelRideHandler Function

Field	Description
What the service does	Handles ride cancellation requests from either passengers or drivers. Processes cancellation fees if applicable, updates all parties involved, releases driver availability, and manages refunds or charges based on cancellation policy and timing.
Inputs required	• `rideId` (string) - Unique identifier of the ride to be cancelled • `userId` (string) - Identifier of the user initiating the cancellation (passenger or driver)

Outputs returned	• `cancellationStatus` ("cancelled" — "cancellation_failed") - Result of cancellation attempt • `affectedParties` (array) - List of users notified about the cancellation • `driverCompensation` (optional object) - Compensation for driver if applicable • `success` (boolean) - Cancellation processing result • `errorMessage` (optional string) - Error details if cancellation fails
Interactions with other systems/components	• Database Service - Updates ride status and records cancellation details • Payment System - Processes cancellation fees and refunds • Notification System - Notifies all affected parties of cancellation • Seat Availability Service - Updates driver seat status back to available • Real-time Tracking Service - Terminates active tracking sessions

3.4 completeRideHandler Function

Field	Description
What the service does	Finalizes a ride when the driver marks it as completed after reaching the destination. Calculates final fare based on actual route and time, processes payment, updates driver availability, and triggers post-ride activities like rating requests.
Inputs required	• `rideId` (string) - Unique identifier of the ride being completed • `driverId` (string) - Identifier of the driver marking the ride as complete
Outputs returned	• `rideStatus` ("completed") - Final ride status • `finalFare` (object) - Complete fare breakdown including actual charges • `paymentStatus` ("processed" — "pending" — "failed") - Payment processing result • `rideHistory` (object) - Complete ride details for records • `receiptInfo` (object) - Digital receipt details • `driverEarnings` (object) - Driver earnings breakdown • `ratingPrompt` (boolean) - Whether rating requests should be sent • `success` (boolean) - Completion processing result

Interactions with other systems/components	• Database Service - Updates ride completion status and stores final details • Payment System - Processes final payment and driver earnings • Notification System - Sends completion notifications and rating requests • Real-time Tracking Service - Finalizes tracking data

3.5 declineRideHandler Function

Field	Description
What the service does	Processes a driver's decision to decline a ride request. Records the decline reason, removes the driver from the current request pool, continues searching for alternative drivers, and may adjust driver matching algorithms based on decline patterns.
Inputs required	• `rideId` (string) - Unique identifier of the ride request being declined • `driverId` (string) - Identifier of the driver declining the ride
Outputs returned	• `declineStatus` ("declined") - Confirmation of decline processing • `remainingDrivers` (array) - List of other drivers still available for the ride • `searchContinues` (boolean) - Whether the search for drivers continues • `estimatedMatchTime` (number) - Updated estimated time to find a driver • `driverAvailability` ("available" — "temporarily_unavailable") - Driver's updated availability status • `success` (boolean) - Decline processing result • `errorMessage` (optional string) - Error details if decline processing fails
Interactions with other systems/components	• Database Service - Records decline and updates ride request status • Driver Matching Service - Continues search with remaining drivers • Analytics Service - Records decline patterns for algorithm improvement • Notification System - May notify passenger of continued search

3.6 endRideHandler Function

Field	Description
What the service does	Handles the final termination of a ride session, typically called after completion processing. Cleans up active sessions, terminates real-time connections, archives ride data, and ensures all resources are properly released for system efficiency.

Inputs required	• `rideId` (string) - Unique identifier of the ride session to end • `userId` (string) - Identifier of the user requesting to end the ride session
Outputs returned	• `sessionStatus` ("ended") - Confirmation that ride session has been terminated • `dataArchived` (boolean) - Whether ride data has been successfully archived • `resourcesReleased` (boolean) - Confirmation of system resource cleanup • `finalSummary` (object) - Complete ride summary for user records • `nextActions` (array) - Available post-ride actions (rate, report, share) • `success` (boolean) - Session termination result • `errorMessage` (optional string) - Error details if termination fails
Interactions with other systems/components	• Database Service - Archives ride data and cleans active sessions • Real-time Tracking Service - Terminates location tracking and connections • Session Management Service - Ends active ride sessions • Resource Management Service - Releases allocated system resources • Cache Service - Clears temporary ride-related cached data • Monitoring Service - Updates system performance metrics • Cleanup Service - Performs post-ride system maintenance tasks

4 Notification System Service

The Notification System service manages all communication between the system and users through multiple channels including SMS, push notifications, and in-app messages. This service handles notification delivery, user preferences, and message tracking.

4.1 sendNotificationHandler Function

Field	Description
What the service does	Sends notifications to users through appropriate channels based on user preferences and message type. Handles message formatting, delivery method selection, queuing for reliable delivery, and tracks delivery status across multiple notification channels.
Inputs required	• `userId` (string) - Unique identifier of the target user • `message` (string) - The notification message content to be sent • `type` (string) - Notification category such as "ride", "payment", "system" • `metadata` (optional object) - Additional data for message customization and tracking

Outputs returned	• `notificationId` (string) - Unique identifier for the sent notification • `deliveryStatus` ("queued" — "sent" — "delivered" — "failed") - Current delivery status • `deliveryMethod` (string) - Method used for delivery (SMS, push, in-app, email) • `deliveryTimestamp` (datetime) - When the notification was sent • `estimatedDelivery` (datetime) - Expected delivery time • `retryCount` (number) - Number of delivery attempts if failed • `success` (boolean) - Notification sending result • `errorMessage` (optional string) - Error details if sending fails
Interactions with other systems/components	• Database Service - Stores notification records and retrieves user preferences • SMS Gateway - Sends SMS notifications • Push Notification Service - Delivers mobile push notifications • Email Service - Sends email notifications • User Preferences Service - Determines preferred notification channels • Template Engine - Formats messages using predefined templates • Queue Management Service - Manages notification delivery queues • Analytics Service - Tracks notification metrics and engagement

4.2 getNotificationsHandler Function

Field	Description
What the service does	Retrieves a user's notification history including unread messages, notification status, and metadata. Supports pagination for large notification lists and provides filtering options by notification type, date range, and read status.
Inputs required	• `userId` (string) - Unique identifier of the user whose notifications to retrieve
Outputs returned	• `notifications` (array) - List of user notifications with details • `unreadCount` (number) - Total number of unread notifications • `totalCount` (number) - Total number of notifications for the user • `latestNotification` (object) - Most recent notification details • `notificationTypes` (array) - Available notification categories • `paginationInfo` (object) - Pagination details for large lists • `success` (boolean) - Retrieval operation result • `errorMessage` (optional string) - Error details if retrieval fails

Interactions with other systems/components	• Database Service - Retrieves stored notification records • User Authentication Service - Validates user identity and permissions • Pagination Service - Handles large result set pagination • Notification Filtering Service - Applies filters and sorting options • Cache Service - Caches frequently accessed notification data • Analytics Service - Tracks notification access patterns

4.3 markNotificationAsReadHandler Function

Field	Description
What the service does	Updates the read status of a specific notification when a user views it. Records the read timestamp, updates unread counts, and may trigger follow-up actions based on notification type. Ensures accurate tracking of user engagement with notifications.
Inputs required	• `notificationId` (string) - Unique identifier of the notification to mark as read • `userId` (string) - Identifier of the user marking the notification as read
Outputs returned	• `readStatus` ("read") - Confirmation that notification has been marked as read • `readTimestamp` (datetime) - When the notification was marked as read • `updatedUnreadCount` (number) - New unread notification count for the user • `followUpActions` (array) - Any triggered actions based on the notification type • `notificationDetails` (object) - Updated notification information • `success` (boolean) - Read status update result • `errorMessage` (optional string) - Error details if update fails
Interactions with other systems/components	• Database Service - Updates notification read status and timestamp • User Authentication Service - Validates user permissions for the notification • Analytics Service - Tracks notification engagement and read patterns • Counter Service - Updates unread notification counters • Follow-up Action Service - Triggers actions based on notification type • Cache Service - Updates cached notification status data

5 Routes Service

5.1 calculateRoute Function

Field	Description
Service Name	findBestMatchingRoutes
What the service does	Finds optimal transportation routes based on geographical proximity to user-specified start and end locations. Calculates distance-based scores using Haversine formula, filters routes within configurable distance limits, and returns routes sorted by suitability with comprehensive proximity metrics and stop details.
Inputs required	• `startLat` (number) - Latitude of departure location • `startLon` (number) - Longitude of departure location • `endLat` (number) - Latitude of destination location • `endLon` (number) - Longitude of destination location • `maxStartDistance` (optional number) - Maximum acceptable distance from start point in km (default: 2.0) • `maxEndDistance` (optional number) - Maximum acceptable distance from end point in km (default: 8.0) • `maxResults` (optional number) - Maximum routes to return (default: 10)
Outputs returned	• `success` (boolean) - Operation status • `matchingRoutes` (array) - Optimized route recommendations with: – Route identification and metadata – Proximity measurements to start/end points – Closest stop details with distances – Route suitability scoring – Direct route availability flag • `totalRoutesChecked` (number) - Total active routes evaluated • `routesWithinRange` (number) - Routes meeting distance criteria • `searchCriteria` (object) - Parameters used for search • `message` (string) - Status or error information
Interactions with other systems/components	• Route Database - Retrieves all active routes • Enriched Stops Database - Gets enhanced stop information • Distance Calculation Service - Haversine formula computations • Scoring Engine - Calculates route suitability scores • Filtering Service - Applies distance constraints • Sorting Service - Orders results by priority

5.2 displayRoute Function

Field	Description

What the service does	Provides comprehensive details for a specific route including all stops with optional distance calculations from a user's current location. Retrieves enriched stop information and calculates proximity metrics when user coordinates are provided.
Inputs required	• `routeId` (string) - Unique identifier of the target route • `userLat` (optional number) - User's current latitude for distance calculations • `userLon` (optional number) - User's current longitude for distance calculations
Outputs returned	• `success` (boolean) - Operation status • `route` (object) - Complete route details including: – Route metadata and operational information – Complete stop list with ordering – Distance calculations from user (if coordinates provided) – Enrichment status indicator • `message` (string) - Status or error information
Interactions with other systems/components	• Route Database - Retrieves specific route by ID • Enriched Stops Database - Gets enhanced stop data • Distance Calculation Service - Computes user-to-stop distances • Data Enrichment Service - Provides stop name enhancement status • Error Handling - Manages missing route scenarios

5.3 storeRecentRoute Function

Field	Description
What the service does	Discovers all transportation stops within a specified radius of a given location across all active routes. Provides comprehensive stop information including associated route details and exact distance measurements.
Inputs required	• `lat` (number) - Center point latitude for search • `lon` (number) - Center point longitude for search • `radiusKm` (optional number) - Search radius in kilometers (default: 2.0) • `maxResults` (optional number) - Maximum stops to return (default: 20)

Outputs returned	• `success` (boolean) - Operation status • `nearbyStops` (array) - Stop objects within radius containing: – Stop identification and metadata – Precise distance measurement – Associated route information – Geographical coordinates • `searchLocation` (object) - Center coordinates used for search • `radiusKm` (number) - Search radius applied • `totalFound` (number) - Number of stops discovered • `message` (string) - Status or error information
Interactions with other systems/components	• Route Database - Retrieves all active routes • Enriched Stops Database - Gets enhanced stop data • Geographical Search Service - Radial distance calculations • Result Aggregation - Collects stops from multiple routes • Sorting Service - Orders stops by proximity • Pagination Service - Limits results according to parameters

6 Payment System Service

6.1 earnings Function

Field	Description
Service Name	getWeeklyEarnings
What the service does	Calculates comprehensive weekly earnings and performance metrics for a driver over the last 4 weeks. Aggregates trip data, work sessions, and provides detailed daily breakdowns including earnings, online hours, reservations, and hourly averages.
Inputs required	• `driverId` (string) - Unique identifier of the driver
Outputs returned	• `Array of weekly objects` containing: – `dateRangeStart` (number) - Start timestamp of the week – `earnings` (number) - Total earnings for the week – `hoursOnline` (number) - Total hours worked (rounded) – `averagePerHour` (number) - Average earnings per hour – `reservations` (number) - Number of reservation trips – `dailyData` (array) - Daily breakdown with earnings per day – `todayEarnings` (number) - Earnings for current day (current week only)

Interactions with other systems/components	• Trip Database - Retrieves driver's trip records • Work Sessions Database - Gets driver's work session data • Date/Time Service - Calculates week boundaries and day ranges • Aggregation Service - Computes totals and averages • Data Transformation - Formats daily breakdowns
Special Features	• Handles 4-week rolling window analysis • Calculates real-time today's earnings for current week • Provides day-by-day earnings breakdown • Computes hourly productivity metrics • Differentiates reservation vs regular trips

6.2 fare Function

Field	Description
Service Name	getFareForLatestTripHandler
What the service does	Retrieves the fare amount from the most recent trip for a given user, checking both passenger and driver roles. Returns the latest trip fare regardless of whether the user was a passenger or driver in that trip.
Inputs required	• `userId` (string) - Unique identifier of the user
Outputs returned	• `fare` (number) - Fare amount from the most recent trip • `null` - If no trips found for the user
Interactions with other systems/components	• Trip Database - Queries trip records by passengerId • Trip Database - Queries trip records by driverId • User Role Service - Checks user's potential roles in trips • Sorting Service - Orders trips by recency (descending) • Result Resolution - Returns first non-null result from role checks
Special Features	• Dual-role support (passenger and driver) • Returns most recent trip regardless of role • Graceful null handling for users with no trip history • Efficient querying with database indexes

7 Rating and Feedback System Service

7.1 averageRating Function

Field	Description

What the service does	Calculates the average rating for a specific driver based on all submitted feedback. Processes all valid ratings, computes the arithmetic mean, and returns the result formatted to one decimal place. Returns 0 if no ratings are available.
Inputs required	• `driverId` (Id) - Unique identifier of the driver
Outputs returned	• `averageRating` (number) - Calculated average rating (0.0 to .0) • Returns 0 if no valid ratings found
Interactions with other systems/components	• Feedback Database - Retrieves all feedback records for the driver • Data Validation Service - Filters out invalid or non-numeric ratings • Calculation Service - Computes arithmetic mean • Formatting Service - Rounds result to one decimal place
Special Features	• Handles empty rating sets gracefully • Filters invalid ratings (non-numbers, zero/negative values) • Precision formatting to one decimal place • Efficient database indexing by driver
Error Handling	• Returns 0 instead of throwing errors for no ratings • Silent filtering of invalid rating values

7.2 saveFeedback Function

Field	Description
What the service does	Stores passenger feedback for a completed ride. Ensures only one feedback submission per ride, validates input data, and creates a comprehensive feedback record with rating, comments, and location details.
Inputs required	• `rideId` (string) - Unique identifier of the ride • `passengerId` (string) - Identifier of the passenger submitting feedback • `driverId` (string) - Identifier of the driver being rated • `rating` (number) - Numerical rating (typically 1-) • `comment` (string) - Optional textual feedback • `startLocation` (object) - Ride starting location details • `endLocation` (object) - Ride ending location details
Outputs returned	• `id` (string) - Unique identifier of the created feedback record • Throws error if feedback already exists for the ride

Interactions with other systems/components	• Feedback Database - Checks for existing feedback by rideId • Database Insertion Service - Creates new feedback record • Validation Service - Ensures ride-based uniqueness • Timestamp Service - Records creation time
Special Features	• Prevents duplicate feedback for the same ride • Comprehensive ride context preservation • Automatic timestamping • Ride-based indexing for efficient lookups
Error Conditions	• "Feedback already submitted for this ride" - Duplicate prevention • Database insertion failures • Invalid input data types

7.3 showFeedback Function

Field	Description
What the service does	Provides two specialized functions for retrieving feedback: one for passengers to view their submitted feedback with driver details, and one for drivers to view feedback received about their service.
Components	• `showFeedbackPassengerHandler` - Gets feedback submitted by a passenger • `showFeedbackDriverHandler` - Gets feedback received by a driver
Inputs required	• `passengerId` (string) - For passenger view • `driverId` (string) - For driver view
Outputs returned	• `Array of feedback objects` containing: – Rating, comments, timestamps – Ride location details – Driver information (passenger view only) – Chronological ordering (descending)
Interactions with other systems/components	• Feedback Database - Retrieves feedback records by user • User Database - Enriches feedback with driver details (passenger view) • Data Enrichment Service - Adds driver names to passenger feedback
Special Features	• Dual perspective: passenger and driver views • Chronological sorting for easy review • Efficient database indexing by user IDs

8 Conclusion

These service contracts provide a comprehensive foundation for the TaxiTap system architecture. Each contract clearly defines the interface, behavior, and integration points for major system services, enabling reliable communication between components and facilitating system maintenance and expansion.