ChaRo Parcel Tracker - Project Report

Project Name: ChaRo Parcel Tracker
Platform: Flutter (Cross-platform: Android, iOS, Web, Windows, macOS, Linux)
Backend: Supabase (PostgreSQL Database, Edge Functions, Storage)
AI Integration: Google Gemini 2.5 Flash
Date: 2025

Table of Contents

1. App Overview

1.1 Purpose

ChaRo Parcel Tracker is a comprehensive parcel tracking application designed for the Kenyan market. The application enables users to track their parcels in real-time, view delivery routes on an interactive map, and interact with an AI-powered assistant for parcel-related queries. Administrators can manage all parcels, update their status and locations, and maintain detailed tracking history.

1.2 Key Features

User Features:

- **User Authentication**: Secure sign-up and sign-in with email and password (SHA-256 hashed)
- **Parcel Tracking**: Track parcels by ID with detailed status information
- **Interactive Map**: Visualize parcel routes on a live map with color-coded polylines
- **Tracking Timeline**: Horizontal timeline showing all status changes and location updates
- **AI Chat Assistant**: Natural language chatbot powered by Google Gemini for parcel queries
- **Profile Management**: Upload and manage profile photos stored in Supabase Storage
- **Parcel History**: View all parcels associated with the user account

Admin Features:

- **Admin Dashboard**: Dedicated admin tab with access to all parcels across all users
- **Parcel Management**: Search, filter, and update parcel status and locations
- **Status Updates**: Change parcel status (Pending, In Transit, Out for Delivery, Delivered)
- **Location Management**: Update current location from 47 Kenyan counties
- **Admin Notes**: Add notes to tracking history for each status change
- **Bulk Operations**: View and manage multiple parcels simultaneously

1.3 Technology Stack

Frontend:

- Flutter 3.9.2 (Dart)
- Material Design 3
- flutter_map for map visualization
- Image Picker for profile photos

Backend:

- Supabase (PostgreSQL database)
- Supabase Edge Functions (Deno runtime)
- Supabase Storage (profile photos)

AI/ML:

- Google Gemini 2.5 Flash API
- Natural Language Processing for chat queries

Key Dependencies:

- `supabase_flutter: ^2.8.4` - Backend integration
- `flutter_map: ^7.0.2` - Map visualization
- `latlong2: ^0.9.1` - Geographic coordinates
- `image_picker: ^1.1.2` - Image selection
- `crypto: ^3.0.3` - Password hashing
- `http: ^1.1.0` - HTTP requests

1.4 Target Users

- **End Users**: Kenyan residents tracking their parcels
- **Administrators**: Logistics staff managing parcel deliveries
- **Courier Services**: Companies using the platform for delivery tracking

---

2. Architecture & Flow Diagrams

2.1 System Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                    Flutter Application                         │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐       │
│  │   Main App   │  │  Dashboard   │  │   Services   │       │
│  │  (Auth UI)   │  │   (Tabs)     │  │  (AI Chat)   │       │
│  └──────────────┘  └──────────────┘  └──────────────┘       │
│         │                  │                  │               │
│         └──────────────────┼──────────────────┘               │
│                            │                                  │
└────────────────────────────┼──────────────────────────────────┘
                             │
```

```
                          │ HTTPS/REST API
                          │
    ┌─────────────────────┼───────────────────────────┐
    │                 Supabase Backend                │
    │  ┌───────────┐  ┌───────────┐  ┌───────────┐    │
    │  │ PostgreSQL │  │ Edge Function│  │  Storage  │    │
    │  │  Database  │  │ (Gemini API) │  │ (Photos)  │    │
    │  └───────────┘  └───────────┘  └───────────┘    │
    │        │              │              │          │
    └────────┼──────────────┼──────────────┼──────────┘
             │              │              │
             │              │              │
             ▼              ▼              ▼
       ┌──────────┐    ┌──────────┐    ┌──────────┐
       │  Users   │    │  Gemini  │    │  Files   │
       │ Parcels  │    │   API    │    │  (S3)    │
       │ Tracking │    │          │    │          │
       │ History  │    │          │    │          │
       └──────────┘    └──────────┘    └──────────┘
```

2.2 Database Schema

Users Table:

- `id` (UUID, Primary Key)
- `email` (TEXT, Unique)
- `full_name` (TEXT)
- `password_hash` (TEXT)
- `profile_photo_path` (TEXT, nullable)
- `is_admin` (BOOLEAN, default: false)
- `created_at` (TIMESTAMP)
- `updated_at` (TIMESTAMP)

Parcels Table:

- `id` (UUID, Primary Key)
- `parcel_id` (TEXT, Unique)
- `user_id` (UUID, Foreign Key → users.id)
- `from_county` (TEXT)
- `to_county` (TEXT)
- `current_location` (TEXT, nullable)
- `status` (TEXT: Pending, In Transit, Out for Delivery, Delivered)
- `courier_service` (TEXT)
- `recipient_name` (TEXT)
- `description` (TEXT)
- `created_at` (TIMESTAMP)
- `updated_at` (TIMESTAMP)

Tracking History Table:

- `id` (UUID, Primary Key)
- `parcel_id` (TEXT, Foreign Key → parcels.parcel_id)
- `status` (TEXT)
- `location` (TEXT, nullable)
- `notes` (TEXT, nullable)
- `updated_by` (TEXT, nullable - Admin user ID)
- `updated_at` (TIMESTAMP)
- `created_at` (TIMESTAMP)

Indexes:

- `idx_tracking_history_parcel_id` on `tracking_history(parcel_id)`
- `idx_tracking_history_updated_at` on `tracking_history(updated_at DESC)`

2.3 User Authentication Flow

```
┌─────────────┐
│   Splash    │
│   Screen    │
└──────┬──────┘
       │
       ▼
┌─────────────────┐
│  Auth Screen    │
│  (Sign In/Up)   │
└──────┬──────────┘
       │
       ├─── Sign In ──────────────┐
       │                          │
       │  1. Validate email/password │
       │  2. Hash password (SHA-256) │
       │  3. Query users table    │
       │  4. Verify password hash │
       │  5. Check is_admin flag  │
       │                          │
       └──────────────────────────┼───┐
                                  │   │
       ┌──────────────────────────┘   │
       │                              │
       ▼                              ▼
┌─────────────────┐         ┌─────────────────┐
│  Dashboard      │         │  Dashboard      │
│  (Regular User) │         │  (Admin User)   │
│  3 Tabs         │         │  4 Tabs          │
│  - Home         │         │  - Home         │
│  - Profile      │         │  - Profile      │
│  - Settings     │         │  - Settings     │
│                 │         │  - Admin        │
└─────────────────┘         └─────────────────┘
```

2.4 Parcel Tracking Flow

```
┌─────────────┐
│   User      │
│  Dashboard  │
└──────┬──────┘
       │
       │ Click "Track Parcel"
       ▼
┌─────────────────┐
│  Track Parcel   │
│    Dialog       │
└──────┬──────────┘
       │
       │ Enter Parcel ID
       │
       ▼
┌──────────────────────────────────┐
│  Query Database                  │
│  1. Fetch parcel by ID           │
│  2. Fetch tracking_history       │
│  3. Build route from locations   │
└──────┬───────────────────────────┘
       │
       │ Return Data
       ▼
┌──────────────────────────────────┐
│  Display Results                 │
│  - Parcel Details Card           │
│  - Horizontal Timeline           │
│  - Status Badge                  │
│  - Location Information          │
└──────────────────────────────────┘
```

2.5 Live Map Flow

```
┌─────────────┐
│   User      │
│  Dashboard  │
└──────┬──────┘
       │
       │ Click "Live Map"
       ▼
┌───────────────────────────────────┐
│  Fetch User Parcels               │
│  - Filter: status ≠ "Delivered"  │
│  - Include tracking_history       │
└──────┬────────────────────────────┘
       │
       │ For Each Parcel:
```

```
       ▼
┌──────────────────────────────────┐
│  Build Route Array                 │
│  1. Start: from_county             │
│  2. Add: tracking_history locations│
│  3. Add: current_location          │
│  4. Convert counties → LatLng      │
└──────────────────────────────────┘
       │
       │
       │ Assign Unique Color
       ▼
┌──────────────────────────────────┐
│  Render Map                        │
│  - Polylines (route lines)        │
│  - Markers (current locations)     │
│  - Legend (parcel list)            │
└──────────────────────────────────┘
```

2.6 AI Chat Flow

```
┌─────────────┐
│   User      │
│  Dashboard  │
└─────────────┘
       │
       │
       │ Click Chat Button
       ▼
┌─────────────────┐
│  AI Chat Dialog │
└─────────────────┘
       │
       │
       │ User types message
       │
       ▼
┌──────────────────────────────────┐
│  Flutter App                      │
│  AIChatService.sendMessage()      │
└──────────────────────────────────┘
       │
       │
       │ POST /functions/v1/gemini-chat
       │ { message, userId }
       ▼
┌──────────────────────────────────┐
│  Supabase Edge Function            │
│  (gemini-chat/index.ts)            │
│                                     │
│  1. Validate request               │
│  2. Fetch user's parcels            │
│  3. Format context                  │
│  4. Build system prompt             │
└──────────────────────────────────┘
       │
```

```
        │
        │ POST to Gemini API
        │ (with API key from secrets)
        ▼
┌─────────────────────────────────┐
│  Google Gemini API               │
│  (gemini-2.5-flash)              │
│                                   │
│  Generate response based on:     │
│  - User's parcels                │
│  - Current question              │
└──────┬───────────────────────────┘
        │
        │ Return AI response
        ▼
┌─────────────────────────────────┐
│  Edge Function                  │
│  Return { response, success }    │
└──────┬───────────────────────────┘
        │
        │ JSON response
        ▼
┌─────────────────────────────────┐
│  Flutter App                    │
│  Display response in chat UI    │
└─────────────────────────────────┘
```

2.7 Admin Update Parcel Flow

```
┌─────────────┐
│  Admin Tab  │
└──────┬──────┘
        │
        │ Click "Edit" on parcel
        ▼
┌─────────────────────────────────┐
│  Update Parcel Dialog            │
│  - Pre-fill current values       │
│  - Status dropdown               │
│  - Location dropdown (47 counties)│
│  - Notes field                   │
└──────┬───────────────────────────┘
        │
        │ Admin updates & clicks "Update"
        ▼
┌─────────────────────────────────┐
│  Update Database                 │
│  1. Update parcels table:        │
│     - status                     │
│     - current_location           │
│     - updated_at                 │
│  2. Insert tracking_history:      │
│     - status                      │
│     - location                    │
│     - notes                       │
│     - updated_by (admin ID)       │
└──────┬────────────────────────────┘
       │
       │
       │ Success
       ▼
┌──────────────────────────────────┐
│  Refresh Parcel List              │
│  Show success message             │
│  Close dialog                     │
└──────────────────────────────────┘
```

---

3. Implementation Summary

3.1 Project Structure

```
parcel_tracking_app/
├── lib/
│   ├── main.dart                  # App entry, authentication UI
│   ├── dashboard.dart             # Main dashboard with tabs (3964 lines)
│   ├── services/
│   │   └── ai_chat_service.dart # AI chat service
│   └── widgets/
│       └── ai_chat_dialog.dart  # Chat UI dialog
├── supabase/
│   ├── functions/
│   │   └── gemini-chat/
│   │       ├── index.ts         # Edge Function for Gemini API
│   │       └── deno.json        # Deno configuration
│   └── migrations/
│       ├── add_admin_support.sql
│       ├── add_current_location.sql
│       └── add_tracking_history.sql
├── android/                       # Android platform files
├── ios/                           # iOS platform files
├── web/                           # Web platform files
├── windows/                       # Windows platform files
├── macos/                         # macOS platform files
├── linux/                         # Linux platform files
└── pubspec.yaml                   # Flutter dependencies
```

3.2 Core Components

3.2.1 Authentication System (`lib/main.dart`)

Features:

- Animated splash screen with app branding
- Tab-based authentication UI (Sign In / Sign Up)
- Password hashing using SHA-256
- Email validation
- Error handling and user feedback
- Admin detection on login

Key Functions:

- `_hashPassword()`: SHA-256 password hashing
- `_handleSignIn()`: User authentication
- `_handleSignUp()`: New user registration
- Admin flag detection and passing to dashboard

3.2.2 Dashboard Screen (`lib/dashboard.dart`)

Architecture:

- Tab-based navigation (3 tabs for users, 4 for admins)
- State management using StatefulWidget
- Supabase client integration
- Image picker for profile photos

Tabs:

1. **Home Tab:**
    - Parcel list with status cards
    - "Track Parcel" dialog with timeline
    - "Live Map" dialog with route visualization
    - FloatingActionButton for AI chat
    - Pull-to-refresh functionality
2. **Profile Tab:**
    - User information display
    - Profile photo upload (camera/gallery)
    - Photo storage in Supabase Storage
    - Cache-busting for image updates
3. **Settings Tab:**
    - App settings and preferences
    - Logout functionality
4. **Admin Tab (Admin Only):**
    - Search and filter parcels
    - List of all parcels from all users
    - Update parcel dialog
    - Status and location management

  - Admin notes functionality

Key Features:

- **Parcel Tracking Dialog:**
  - Parcel ID input
  - Database query for parcel details
  - Horizontal timeline visualization
  - Status badges with color coding
  - Location information display
- **Live Map Dialog:**
  - Interactive map using flutter_map
  - Route polylines with unique colors
  - Location markers
  - County-to-coordinates conversion
  - Route legend panel
  - Filtering of delivered parcels
- **Update Parcel Dialog (Admin):**
  - Status dropdown (4 options)
  - Location dropdown (47 Kenyan counties)
  - Optional notes field
  - Database update with tracking history
  - Success/error feedback

3.2.3 AI Chat Service (`lib/services/ai_chat_service.dart`)

Functionality:

- HTTP POST requests to Supabase Edge Function
- Error handling for network issues
- User ID passing for context
- Response parsing and error management

API Endpoint:

- `POST /functions/v1/gemini-chat`
- Headers: Authorization, Content-Type
- Body: `{ message, userId }`

3.2.4 AI Chat Dialog (`lib/widgets/ai_chat_dialog.dart`)

UI Components:

- Full-screen dialog with gradient header
- Message list with user/AI distinction
- Text input field with send button

- Loading indicators
- Auto-scrolling to latest messages
- Welcome message on initialization

Message Display:

- User messages: Right-aligned, purple background
- AI messages: Left-aligned, grey background
- Avatar icons for visual distinction
- Timestamp support (stored but not displayed)

3.2.5 Supabase Edge Function (`supabase/functions/gemini-chat/index.ts`)

Functionality:

- CORS handling for cross-origin requests
- Environment variable management (GEMINI_API_KEY)
- User parcel fetching from database
- Context formatting for AI
- Gemini API integration
- Error handling and response formatting

AI Integration:

- Model: `gemini-2.5-flash`
- System prompt includes user's parcels
- Natural language understanding
- Context-aware responses

3.3 Database Implementation

3.3.1 Migrations

Phase 1: Admin Support

- Added `is_admin` column to `users` table
- Default value: `false`
- Enables role-based access control

Phase 2: Current Location

- Added `current_location` column to `parcels` table
- Stores county name as TEXT
- Used for map visualization and tracking

Phase 3: Tracking History

- Created `tracking_history` table
- Stores all status changes and location updates
- Includes admin notes
- Indexed for performance

3.3.2 Data Flow

Parcel Creation:

1. User creates parcel via admin or API
2. Parcel inserted into `parcels` table
3. Initial status: "Pending"
4. `from_county` set as starting location

Parcel Updates:

1. Admin updates status/location
2. `parcels` table updated
3. New entry inserted into `tracking_history`
4. Route array rebuilt for map visualization

Tracking Query:

1. User enters parcel ID
2. Query `parcels` table by `parcel_id`
3. Query `tracking_history` by `parcel_id`
4. Build timeline from history entries
5. Display chronological events

3.4 Map Visualization

Implementation Details:

- Library: `flutter_map` with OpenStreetMap tiles
- Coordinate conversion: County names → LatLng coordinates
- Color assignment: 14-color palette, cycling for multiple parcels
- Polyline rendering: 3px width with white borders
- Marker placement: Current location for each active parcel
- Filtering: Excludes delivered parcels

Route Building Algorithm:

1. Start with `from_county` as first point
2. Fetch all `tracking_history` entries for parcel
3. Sort by `updated_at` chronologically
4. Extract unique locations
5. Add `current_location` if different from last point
6. Convert county names to coordinates
7. Draw polyline connecting all points

3.5 Security Implementation

Authentication:

- Password hashing: SHA-256 (one-way)
- No plaintext passwords stored

- Email validation on client and server
- Session management via Supabase

API Security:

- Gemini API key stored in Supabase secrets (not in code)
- Edge Function validates user authentication
- Row-Level Security (RLS) policies on database
- CORS properly configured

Data Access:

- Users can only access their own parcels
- Admins can access all parcels
- Tracking history linked to parcels
- Profile photos stored with user-specific paths

3.6 UI/UX Design

Design Principles:

- Material Design 3 guidelines
- Deep purple/indigo color scheme
- Gradient backgrounds for visual appeal
- Smooth animations and transitions
- Responsive layout for different screen sizes

Key UI Elements:

- Animated splash screen
- Tab-based navigation with custom indicators
- Card-based parcel displays
- Status badges with color coding
- Horizontal timeline for tracking
- Interactive map with markers and routes
- Modern chat interface

User Feedback:

- Loading indicators during async operations
- Success/error snackbars
- Pull-to-refresh on lists
- Empty state messages
- Error handling with user-friendly messages

---

4. Challenges & Improvements

4.1 Challenges Encountered

4.1.1 Database Schema Evolution

Challenge: The database schema evolved over time with multiple migrations (admin support, current location, tracking history). Managing these changes required careful coordination between code and database.

Solution: Created separate migration SQL files for each phase, allowing incremental updates. Documented each migration with clear instructions.

4.1.2 Map Coordinate Conversion

Challenge: Converting Kenyan county names to geographic coordinates (LatLng) for map visualization. No built-in geocoding service integrated.

Solution: Implemented a hardcoded mapping of county names to approximate coordinates. This works for visualization but could be improved with a proper geocoding service.

4.1.3 AI Chat Context Management

Challenge: Providing relevant context to the AI (user's parcels) while maintaining security and performance.

Solution: Edge Function fetches user's parcels server-side, formats context, and sends to Gemini API. This ensures security (API key not exposed) and provides accurate context.

4.1.4 State Management Complexity

Challenge: Managing complex state in the dashboard (parcels, map data, chat messages, profile photos) within a single StatefulWidget.

Solution: Used StatefulWidget with proper state management, but the file grew large (3964 lines). Could benefit from state management solutions like Provider or Riverpod.

4.1.5 Real-time Updates

Challenge: Parcel status changes by admins don't automatically reflect in user's view without manual refresh.

Solution: Implemented pull-to-refresh functionality. Could be improved with Supabase Realtime subscriptions for automatic updates.

4.1.6 Image Upload and Caching

Challenge: Profile photos needed cache-busting to show updated images immediately after upload.

Solution: Implemented cache-busting by appending timestamp query parameter to image URLs.

4.2 Improvements & Future Enhancements

4.2.1 Technical Improvements

1. State Management Refactoring

- **Current:** Single large StatefulWidget (3964 lines)
- **Improvement:** Implement Provider or Riverpod for better state management
- **Benefit:** Cleaner code, easier testing, better performance

2. Real-time Updates

- **Current:** Manual refresh required
- **Improvement:** Implement Supabase Realtime subscriptions
- **Benefit:** Automatic updates when parcels change status

3. Geocoding Service Integration

- **Current:** Hardcoded county-to-coordinate mapping
- **Improvement:** Integrate Google Maps Geocoding API or similar
- **Benefit:** Accurate coordinates, support for addresses beyond counties

4. Offline Support

- **Current:** Requires internet connection
- **Improvement:** Implement local caching with Hive or SQLite
- **Benefit:** View cached parcels offline, sync when online

5. Push Notifications

- **Current:** No notifications
- **Improvement:** Integrate Firebase Cloud Messaging (FCM)
- **Benefit:** Notify users when parcel status changes

6. Image Optimization

- **Current:** Images uploaded as-is
- **Improvement:** Implement image compression and resizing
- **Benefit:** Faster uploads, reduced storage costs

4.2.2 Feature Enhancements

1. Advanced Search and Filtering

- Filter parcels by status, date range, county
- Search by recipient name, courier service
- Sort by date, status, location

2. Bulk Operations (Admin)

- Bulk status updates
- Export parcel data to CSV/Excel
- Batch location updates

3. Analytics Dashboard (Admin)

- Statistics: Total parcels, delivery times, status distribution
- Charts and graphs for insights
- Performance metrics

4. Multi-language Support

- Support for Swahili and other Kenyan languages
- Localized UI text and messages

5. QR Code Integration

- Generate QR codes for parcels
- Scan QR codes to track parcels
- Faster parcel identification

6. Delivery Estimates

- Calculate estimated delivery time based on route
- Show expected delivery date
- Notify users of delays

7. Parcel Photos

- Allow users to upload photos of received parcels
- Admin can attach photos to tracking history
- Photo gallery in parcel details

8. SMS/Email Notifications

- Send SMS when parcel status changes
- Email updates with tracking links
- Delivery confirmation messages

9. Route Optimization (Admin)

- Suggest optimal delivery routes
- Group parcels by location
- Reduce delivery time and costs

10. User Reviews and Ratings

- Rate delivery experience
- Review courier services
- Feedback system

4.2.3 Performance Optimizations

1. Database Query Optimization

- Add more indexes for frequently queried fields
- Implement pagination for large parcel lists
- Use database views for complex queries

2. Image Loading

- Implement image caching with cached_network_image
- Lazy loading for parcel lists
- Progressive image loading

3. Map Performance

- Limit number of parcels shown on map simultaneously
- Cluster markers for better performance
- Lazy load map tiles

4. API Response Caching

- Cache AI chat responses for common queries
- Cache parcel data with TTL
- Reduce API calls

4.2.4 Security Enhancements

1. Enhanced Authentication

- Implement JWT token refresh
- Add two-factor authentication (2FA)
- Password strength requirements

2. Data Encryption

- Encrypt sensitive data at rest
- Use HTTPS for all API calls (already implemented)
- Encrypt admin notes

3. Rate Limiting

- Implement rate limiting for API calls
- Prevent abuse of AI chat
- Limit parcel creation per user

4. Audit Logging

- Log all admin actions
- Track user activities
- Security event monitoring

4.2.5 User Experience Improvements

1. Onboarding Flow

- Welcome tutorial for new users
- Feature highlights
- Interactive guide

2. Accessibility

- Screen reader support
- High contrast mode
- Font size adjustments

3. Dark Mode

- Implement dark theme
- System theme detection
- User preference toggle

4. Improved Error Messages

- More descriptive error messages
- Actionable error suggestions
- Help documentation links

5. Loading States

- Skeleton screens instead of spinners
- Progressive loading
- Better loading indicators

4.3 Scalability Considerations

Current Limitations:

- Single Supabase project (may need scaling)
- No load balancing for Edge Functions
- Database may need optimization for large datasets

Scalability Improvements:

1. **Database Scaling:**
   - Implement read replicas for queries
   - Partition large tables
   - Archive old tracking history
2. **Edge Function Scaling:**
   - Implement caching layer (Redis)
   - Rate limiting per user
   - Queue system for high traffic
3. **Storage Scaling:**
   - Implement CDN for image delivery
   - Compress images automatically
   - Archive old profile photos
4. **Monitoring and Logging:**
   - Implement application monitoring (Sentry, LogRocket)
   - Performance metrics tracking
   - Error tracking and alerting

4.4 Testing Improvements

Current State: Limited testing implemented

Recommended Testing:

1. **Unit Tests:**
   - Test authentication logic
   - Test password hashing
   - Test coordinate conversion
2. **Widget Tests:**
   - Test UI components
   - Test dialog interactions
   - Test form validation
3. **Integration Tests:**
   - Test database operations
   - Test API integrations
   - Test end-to-end flows
4. **Performance Tests:**
   - Load testing for Edge Functions
   - Database query performance
   - Map rendering performance

4.5 Documentation Improvements

Current State: Basic documentation in markdown files

Recommended Documentation:

1. **API Documentation:**
   - OpenAPI/Swagger specification
   - Endpoint documentation
   - Request/response examples
2. **Code Documentation:**
   - Inline code comments
   - Function documentation
   - Architecture diagrams
3. **User Documentation:**
   - User guide with screenshots
   - FAQ section
   - Video tutorials

4. **Admin Documentation:**
    - Admin user guide
    - Troubleshooting guide
    - Deployment guide

Conclusion

The ChaRo Parcel Tracker application successfully implements a comprehensive parcel tracking system with user authentication, real-time tracking, interactive maps, and AI-powered assistance. The application leverages modern technologies (Flutter, Supabase, Google Gemini) to provide a seamless user experience.

Key Achievements:

- ✅ Cross-platform Flutter application
- ✅ Secure authentication system
- ✅ Real-time parcel tracking with timeline
- ✅ Interactive map with route visualization
- ✅ AI chatbot integration
- ✅ Admin dashboard for parcel management
- ✅ Profile management with photo uploads
- ✅ Comprehensive tracking history

Areas for Future Development:

- State management refactoring
- Real-time updates with Supabase Realtime
- Enhanced geocoding for accurate locations
- Push notifications for status updates
- Advanced analytics and reporting
- Multi-language support
- Offline functionality

The application provides a solid foundation for a production-ready parcel tracking system and can be extended with the improvements outlined above to meet growing user needs and scale requirements.

Report Generated: 2025
Project Status: Functional - Ready for Testing and Deployment
Total Lines of Code: ~5,000+ (Flutter/Dart)
Database Tables: 3 (users, parcels, tracking_history)
Edge Functions: 1 (gemini-chat)
Platforms Supported: Android, iOS, Web, Windows, macOS, Linux