UASIN GISHU SMART WASTE MONITORING SYSTEM (UGSWMS)

1. Introduction

1.1 Purpose

The purpose of this Software Design Specification (SDS) is to describe the overall architecture, module design, database schema, interface layouts, and operational logic of the Uasin Gishu Smart Waste Monitoring System (UGSWMS). This document translates the functional requirements outlined in the SRS into a detailed technical blueprint for developers, testers, and system administrators.

1.2 System Overview

The Uasin Gishu Smart Waste Monitoring System is a web and mobile-based platform that enhances waste management efficiency within Uasin Gishu County. The system uses simulated IoT data to represent bin fill levels in real time and displays them on an interactive dashboard. Administrators can monitor bin status, receive alerts, allocate trucks, and generate reports, while residents and businesses can pay for waste services digitally via M-Pesa using the Daraja API. Drivers use a mobile interface to view assigned bins, follow optimized routes, and update collection statuses. A built-in AI chatbot offers assistance and user support through natural language processing (NLP).

1.3 Design Goals

- To ensure real-time monitoring of simulated bins.
- To support smart allocation of trucks based on location.
- To provide secure payment processing through M-Pesa Daraja API.
- To deliver cross-platform compatibility using Flutter for mobile and web.
- To include AI-driven chatbot assistance for residents and admins.
- To maintain data integrity, scalability, and cloud-based reliability.

1.4 Design Constraints

- IoT readings are simulated, not from physical sensors.
- Requires stable internet connectivity for Firebase and Google Maps APIs.
- Limited free-tier usage for Firebase and Google APIs.
- Daraja API in sandbox mode for testing.
- Chatbot accuracy depends on Hugging Face free-tier API response limits.

2. System Architecture Design

2.1 Overview

UGSWMS follows a three-tier architecture:

- **Presentation Layer**: Flutter-based web and mobile interfaces for all users.
- **Application Layer**: Firebase Cloud Functions managing logic, alerts, payments, and notifications.

- **Data Layer**: Firebase Firestore and third-party APIs (Google Maps, Daraja, Hugging Face) for data persistence and interaction.

2.2 Components

- **Frontend (Flutter)**: Responsive UI for Admin, Driver, and Resident.
- **Backend (Cloud Functions)**: Executes alerts, payments, and chatbot queries.
- **Database (Firestore)**: Stores structured data collections (users, bins, drivers, payments).
- **External APIs**:
    - **Daraja API**: Payment processing.
    - **Google Maps API**: Mapping and tracking.
    - **Hugging Face NLP API**: Chatbot interaction.

2.3 Architecture Description

- Admin uses the web app to monitor the dashboard and assign trucks.
- Drivers access mobile app to view and complete assigned tasks.
- Residents use mobile app for payments, reports, and chatbot interaction.
- Firebase Cloud Functions handle logic for payments, bin alerts, and notifications.
- Real-time data synchronization ensures that changes in Firestore reflect instantly on all devices.

3. System Component Design

3.1 Admin Dashboard

- Provides real-time overview of all bins and trucks.
- Displays color-coded bins based on fill levels.
- Enables truck assignment and performance report generation.
- Includes access to payment summaries and analytics.

3.2 Driver App

- Displays assigned bins and optimized collection routes.
- Allows drivers to update collection progress.
- Sends live location updates to Firebase.
- Provides push notifications for new assignments.

3.3 Resident App

- Enables users to report full bins or request special pickups.
- Provides M-Pesa payment interface (Till/Paybill).
- Includes chatbot for inquiries and payment help.
- Shows collection schedules and digital receipts.

3.4 IoT Simulation Engine

- Randomly increases bin fill-levels every 5–10 seconds.
- Applies gradual updates for realistic behavior.
- Triggers alerts when bins exceed 80% capacity for consecutive cycles.
- Allows time acceleration for demo mode.

3.5 Payment Module (Daraja API)

- Integrates M-Pesa payment system (Till/Paybill).
- Supports STK Push and callback verification.
- Logs transaction details securely in Firestore.

3.6 AI Chatbot

- Powered by Hugging Face NLP API.
- Responds to FAQs, payment help, and bin reports.
- Logs chat sessions for performance improvement.

3.7 Google Maps Module

- Displays live bin and truck locations.
- Computes and visualizes shortest collection routes.
- Updates marker colors dynamically to show status changes.

3.8 Notification System

- Sends push notifications for alerts, task assignments, and payments.
- Uses Firebase Cloud Messaging (FCM) for delivery.

4. Database Design

Collections

- **users**: Stores all system users with role-based access.
- **drivers**: Stores driver info, vehicle details, and routes.
- **bins**: Stores bin status, coordinates, and fill level.
- **payments**: Stores all payment transactions.
- **chatlogs**: Stores chatbot session logs.
- **notifications**: Stores alert and message records.
- **reports**: Stores generated reports and analytics.
- **system_logs**: Stores event history for auditing.

Relationships

- Drivers reference users by ID.
- Payments link to users.

- Bins reference assigned drivers.
- Notifications and reports reference admins.

Data Integrity & Validation

- Access rules managed via Firebase Security Rules.
- Real-time synchronization ensures data consistency.
- Encrypted storage for sensitive information.

5. Interface Design

5.1 Admin Web Dashboard

- Login → Map Dashboard → Alerts → Assign Truck → Reports.
- Real-time Google Map showing all bins and trucks.
- Alert panel with filter options (location, priority).
- Truck assignment modal with top three nearest suggestions.
- Reports with export options (PDF/Excel).

5.2 Driver Mobile App

- Login (OTP) → Tasks → Map → Update Status → History.
- Simple interface for quick updates (En Route, Arrived, Collected).
- Real-time route tracking and ETA estimation.
- "Available/Busy" toggle for active status.

5.3 Resident Mobile App

- Signup → Home (Nearby Bins) → Payment → Chatbot → History.
- Allows payment through M-Pesa Daraja API (Till/Paybill).
- Provides chatbot assistance for common queries.
- Displays transaction receipts and schedules.

5.4 API Interfaces

- **Firebase**: Authentication, Firestore, Cloud Functions, FCM for real-time updates.
- **Google Maps**: For map rendering, directions, and distance matrix.
- **Daraja API**: For M-Pesa payment processing and callback verification.
- **Hugging Face API**: For chatbot communication and NLP responses.

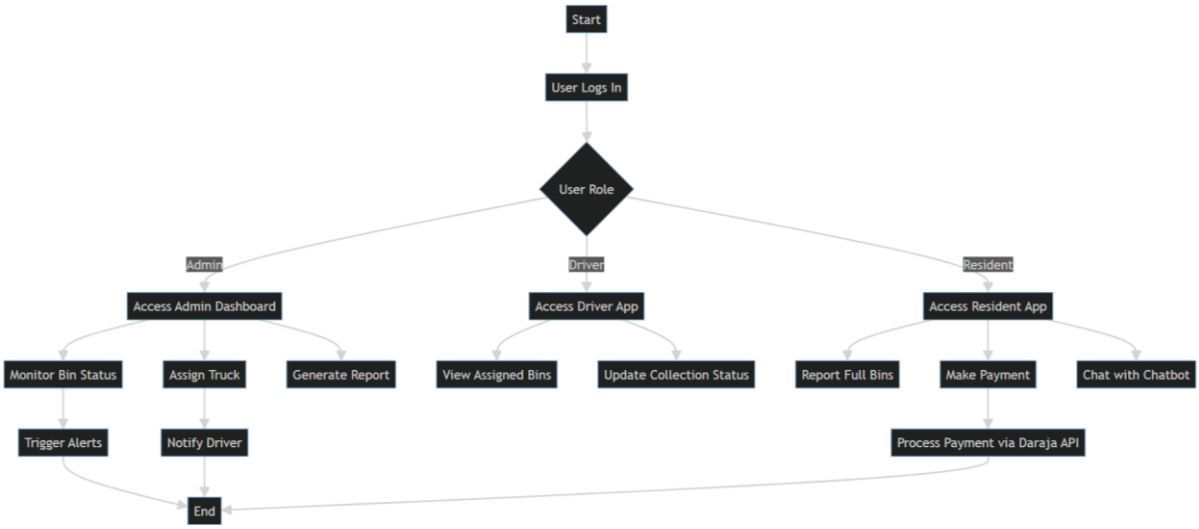
5.5 Example Data Flows

- Bin Simulation → Firebase: Bin fill-level updated → Alert triggered → Admin notified.
- Truck Assignment: Admin assigns → Cloud Function updates Firestore → Driver notified via FCM.
- Payment Process: Resident initiates payment → Daraja callback confirms → Firebase updates → Admin notified.

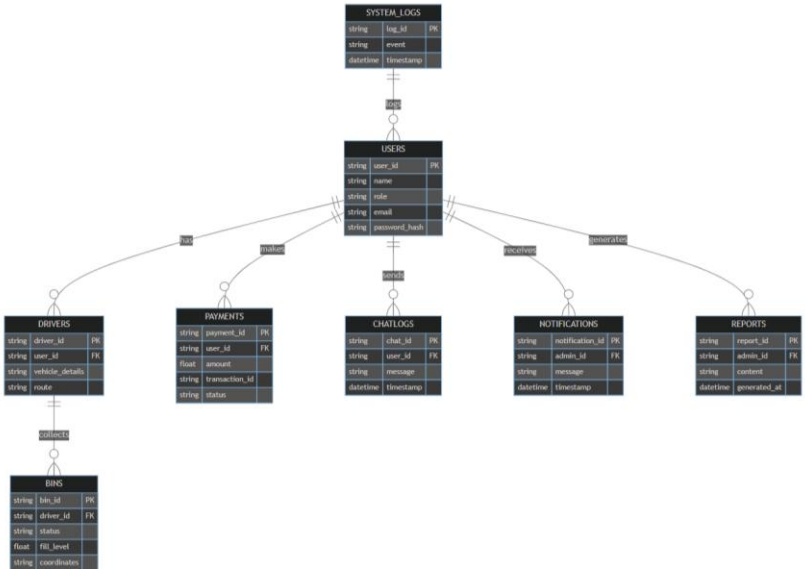
- Chatbot Interaction: User sends query → Backend relays to Hugging Face → Response returned → Log stored.

5.6 Diagrams

Activity Diagram



Entity Relationship Diagram



6. System Flow Design

6.1 Data Flow

- IoT simulation updates bin fill-level in Firestore.
- Cloud Function monitors bin data and triggers alerts if threshold exceeded.
- Admin sees alert and assigns nearest truck.
- Truck assignment saved → Driver receives notification.
- Driver updates status → Dashboard reflects completion.
- Resident pays via M-Pesa → Payment verified via Daraja callback.
- System logs transaction and updates reports automatically.

6.2 Sequence Flow Example (Admin → Driver → Bin)

- Bin fill level exceeds 80%.
- Firebase triggers alert and sends notification.
- Admin views alert, checks suggested trucks.
- Admin assigns truck; Cloud Function updates assigned_truck.
- Driver receives push notification with route details.
- Driver completes collection and updates status.
- System updates bin to “Empty” and logs event.

7. Security and Access Control Design

- Authentication: Managed through Firebase Auth with email-password and OTP.
- Authorization: RBAC controls user privileges (Admin, Driver, Resident).
- Data Encryption: All transactions and API communications use HTTPS.

- Payment Security: Daraja API credentials stored securely in environment variables.
- Database Rules: Firestore rules prevent unauthorized writes.
- Session Security: Automatic logout on inactivity or expired tokens.

8. Error Handling and Logging

- Error Logging: Firebase Cloud Functions log all runtime errors.
- User Feedback: Friendly error messages ("Network Error, Please Retry").
- Retry Logic: Automatic retries for failed network calls.
- Payment Failures: Logged and marked as "Failed"; users can retry.
- System Logs: Every major event (login, assignment, payment) recorded in system_logs collection.

9. Deployment Design

- Hosting: Web app hosted on Firebase Hosting.
- Mobile app: Built and deployed via Flutter (Android primary).
- Backend: Firebase Cloud Functions handle all server logic.
- Database: Firebase Firestore for structured data; Realtime DB for quick simulation updates.
- APIs: Integrated via HTTPS endpoints (Google Maps, Daraja, Hugging Face).
- Monitoring: Firebase Console for metrics, logs, and crash reports.
- Backup: Automated daily backups and export to secure cloud storage.

10. Future Enhancements

- Integration with real IoT sensors for live bin data.
- Machine Learning model to predict fill times and optimize collection routes.
- Offline-first mobile support for drivers in low-connectivity zones.
- User reward system for proper waste disposal.
- County-level expansion with multi-region dashboards.