SCHOOL OF SCIENCE & TECHNOLOGY

SMART MATERNAL EMERGENCY RESPONSE SYSTEM (MAMARESCUE)

Submitted by

SWAFIYAH GICHUKI

ADM NO.: 665725

In Partial Fulfilment For The Award Of Bachelor of Science Degree

In

SOFTWARE ENGINEERING

FALL 2025

SWE4900A: SOFTWARE PROJECT II

DECLARATION

I declare that this is my original work and has never been submitted to any institution for the Certificate, Diploma or Degree award. This document has been submitted to fulfil the requirements for a Degree at USIU University.

STUDENT: SWAFIYAH W. GICHUKI

Signature:  Date: 10th December 2025

This project has been submitted for examination purposes with the approval of the supervisor

SUPERVISOR: DR. FREDRICK OGORE

Signature:…… Date:……

ACKNOWLEDGEMENT

The completion of this project has been a challenging but fulfilling journey, made possible by the support and contributions of several individuals.

My deepest gratitude goes to my supervisor, Dr. Ogore, for his invaluable guidance, patience, and constructive criticism throughout the development of this system. His technical insight and insistence on academic rigor were instrumental in shaping the architecture of the Smart Emergency Response System (SMERS).

I also wish to acknowledge the Department of Computing at USIU-Africa for providing the enabling environment and resources necessary for research and development.

To my parents and family, thank you for your unwavering belief in me, your financial support, and your encouragement. You were my anchor during the late nights of coding and debugging.

Finally, I appreciate my colleagues and friends for the intellectual exchanges and moral support that kept me motivated. Your feedback during the testing phases of this application was truly appreciated.

DEDICATION

This work is dedicated to the memory of all the mothers in Kenya whose lives were tragically cut short because help did not arrive in time.

For the mothers we lost to the waiting. For the lives cut short because help was a mile too far, or a minute too late. To those who looked for help and found only silence, your memory is the heartbeat of this work.

This system is built in your honor, with the hope that we can close the distance between a distress call and a saved life, ensuring that no mother is ever again lost to the barrier of delay.

ABSTRACT

Emergency Medical Services (EMS) are pivotal in reducing mortality rates associated with acute health incidents. However, in developing nations, the efficacy of these services is often compromised by infrastructural deficits, manual dispatching protocols, and unreliable geolocation technologies. In the context of Laikipia County, Kenya, the latency between distress call initiation and ambulance deployment, often referred to as the "Golden Hour", is exacerbated by a lack of real-time fleet visibility and the unavailability of standard ambulances in remote areas.

This project presents the design and implementation of the Smart Emergency Response System (SMERS), a web-based spatial decision support system built upon the PERN stack. A unique feature of SMERS is its integration of **Community First Responders (CFRs)** into the formal emergency chain. The system tracks and dispatches a heterogeneous fleet comprising not only standard ambulances but also community-based paratransit operators, specifically motorcycle taxis (*Boda-bodas*) and auto-rickshaws (*Tuk-tuks*). This inclusion ensures rapid navigation through narrow or unpaved terrains where conventional vehicles often struggle.

A distinct methodological contribution of this study is the development of a **Fault-Tolerant Hybrid Routing Engine**. Recognizing the volatility of external APIs in low-bandwidth environments, the system integrates a dual-layer navigation protocol that autonomously switches between OSRM API data and a deterministic linear interpolation algorithm upon network timeout. The implementation of SMERS demonstrates that leveraging community assets alongside resilient web architectures can significantly enhance the operational efficiency, transparency, and responsiveness of emergency logistics in resource-constrained settings.

Keywords: *Emergency Medical Services, Community First Responders, Hybrid Routing, Fault Tolerance, PERN Stack, Laikipia County.*

ACRONYMS & DEFINITION OF TERMS

ACRONYMS

API ..... Application Programming Interface

CFR ..... Community First Responder

EMS ..... Emergency Medical Services

OSRM ..... Open Source Routing Machine

PERN ..... PostgreSQL, Express, React, Node.js

SMERS ..... Smart Emergency Response System

DEFINITION OF TERMS

Boda-Medic: A colloquial term used in this study to refer to motorcycle taxi (*Boda-boda*) operators who have been integrated into the emergency response network to provide rapid transport for non-critical patients or to navigate difficult terrain.

Community First Responder (CFR): Members of the community, including *Tuk-tuk* and *Boda-boda* operators, who are volunteered or contracted to attend to medical emergencies within their locality before an ambulance arrives.

Fault Tolerance: The ability of a system to continue operating without interruption when one or more of its components (such as an external API or network connection) fail.

Golden Hour: The critical period immediately following a traumatic injury during which there is the highest likelihood that prompt medical and surgical treatment will prevent death.

Hybrid Routing: A navigation approach utilized in this project that combines external cloud-based pathfinding data with local algorithmic fallbacks to ensure continuous operation during network outages.

Resilience Engineering: A paradigm in software architecture focused on building systems that can withstand and recover from difficult conditions, such as the intermittent internet connectivity common in semi-urban Kenya.

CHAPTER ONE

1.1 INTRODUCTION

The efficacy of Emergency Medical Services (EMS) is fundamentally predicated on the minimization of the temporal latency between incident occurrence and clinical intervention, a window clinically referred to as the "Golden Hour" (Trunkey, 1983). In advanced healthcare ecosystems, this latency is mitigated through Computer-Aided Dispatch (CAD) systems that leverage real-time telemetry and predictive algorithms (Wachira et al., 2019). However, in the context of the Global South, the EMS paradigm remains largely reactive and disjointed, often failing to address the "Last Mile" connectivity challenge (Kobusingye et al., 2005).

This project introduces the Smart Emergency Response System (SMERS), a web-based Spatial Decision Support System (SDSS) engineered to operationalize a "Task-Shifting" model. By digitally integrating Community First Responders (CFRs), specifically motorcycle taxis (*Boda-bodas*) and auto-rickshaws (*Tuk-tuks*), alongside formal ambulances, SMERS seeks to establish a heterogeneous emergency fleet (Muni et al., 2020). The system is underpinned by a novel Fault-Tolerant Hybrid Routing Engine, designed to maintain algorithmic integrity and operational continuity in environments characterized by intermittent network infrastructure (Heeks, 2018).

1.2 BACKGROUND INFORMATION

Laikipia County represents a complex geospatial environment for emergency logistics, characterized by a dichotomy between high-density urban nodes and expansive, resource-scarce rural peripheries. According to the Laikipia County Integrated Development Plan (2023), the emergency response framework operates within significant structural silos. Formal assets, such as ambulances stationed at Nanyuki Teaching & Referral Hospital, are often geographically constrained and unable to navigate the unpaved topography of peri-urban settlements efficiently (Ministry of Health, 2021). Conversely, the informal transport sector, comprising a pervasive

network of *Boda-boda* operators, possesses the requisite agility but lacks integration into the central command structure (O'Neill et al., 2019).

Operationally, the absence of real-time geospatial visibility forces dispatchers to rely on heuristic decision-making based on subjective landmark descriptions. This information asymmetry results in suboptimal resource allocation, where distant formal units are dispatched while proximate community responders remain unutilized (Moulton et al., 2017). Furthermore, the lack of digital audit trails creates vulnerabilities in administrative governance, specifically regarding the verification of fuel consumption and the remuneration of contracted community agents (Transparency International, 2020).

1.3 EMERGING ISSUES

Two pivotal trajectories define the contemporary discourse on EMS software engineering in developing economies. The first is Resilience Engineering. As critical life-safety systems migrate to cloud architectures, reliance on third-party APIs (e.g., OSRM, Google Maps) creates a single point of failure (Hollnagel, 2011). There is a growing technical imperative to architect "Offline-First" systems capable of autonomous routing during latency spikes or API outages. The second issue is the Formalization of the Informal Economy. Policy shifts towards integrating informal transport providers into public health strategies necessitate robust Digital Public Infrastructure (DPI) (World Bank, 2022). Such systems must provide tamper-proof, transparent mechanisms for tracking service delivery to ensure financial accountability.

1.4 PROBLEM STATEMENT

Despite the existence of diverse transport modalities in Laikipia County, emergency response protocols remain inefficient due to logistical fragmentation and technological fragility. The primary deficiency is the lack of a unified command interface capable of visualizing and dispatching a heterogeneous fleet (Wanjiku & Mbarika, 2018). Dispatchers operate with limited situational awareness, leading to increased response times. Furthermore, existing low-cost tracking solutions suffer from architectural brittleness; they are heavily dependent on continuous connectivity to external mapping services. In the event of network degradation, these systems fail catastrophically, rendering the dispatch logic obsolete (Avgerou, 2008).

Research Gap: There is a dearth of dispatch systems that simultaneously integrate informal community responders while employing fault-tolerant routing algorithms designed for low-bandwidth environments.

Proposed Solution: SMERS addresses these deficits by orchestrating a unified dashboard for heterogeneous fleet management and implementing a Hybrid Routing Engine. This engine utilizes a deterministic mathematical fallback algorithm to guarantee simulation and navigation continuity when external geolocation APIs become inaccessible.

1.5 MAIN OBJECTIVE

To engineer and deploy a resilient, web-based Smart Emergency Response System (SMERS) that leverages geospatial telemetry and hybrid algorithmic routing to optimize the dispatch efficiency and administrative accountability of the heterogeneous EMS fleet in Laikipia County.

1.6 SPECIFIC OBJECTIVES

1. **To Develop** a real-time geospatial visualization module that tracks the dynamic coordinates of a heterogeneous fleet (Ambulances, *Boda-medics*, *Tuk-tuks*) to eliminate situational blindness in the dispatch center.
2. **To Implement** a proximity-based heuristic algorithm that autonomously identifies the optimal responder based on Euclidean distance and vehicle suitability, thereby reducing dispatch latency.
3. **To Integrate** a Fault-Tolerant Hybrid Routing Engine that ensures system availability by seamlessly switching between API-based pathfinding and a local algorithmic fallback during network interruptions.
4. **To Systematize** administrative governance by engineering a digital ledger for mission verification and automated payment calculation, ensuring fiscal transparency in the management of community responders.

1.8 JUSTIFICATION

1.8.1 Social Significance (Health Equity)

By integrating *Boda-medics*, SMERS facilitates the democratization of emergency care. It extends the reach of EMS into spatially marginalized areas where conventional ambulances cannot traverse, thereby enhancing health equity (Marmot, 2005).

1.8.2 Technical Significance (Architectural Resilience)

This study contributes to the domain of software engineering by validating a Hybrid Routing Architecture. It demonstrates that mission-critical applications can be constructed using open-source technologies (PERN Stack) while achieving enterprise-grade reliability through the implementation of mathematical fallback logic, a model replicable in other resource-constrained settings (Marsden et al., 2018).

1.8.3 Economic Significance (Resource Optimization)

The digitization of mission history and financial disbursements introduces rigorous auditability into EMS operations. This mitigates the risks of resource leakage and optimizes the allocation of county funds, ensuring sustainable financial modeling for community-based health interventions (World Health Organization, 2018).

CHAPTER TWO

2.1 INTRODUCTION

The evolution of Emergency Medical Services (EMS) has been intrinsically linked to advancements in Information and Communication Technology (ICT). From the early adoption of radio-dispatch in the 1970s to modern cloud-based telematics, the goal has remained consistent: to minimize the interval between incident occurrence and medical intervention (Mould-Millman et al., 2015). This chapter critically reviews existing emergency dispatch paradigms globally, regionally, and locally. It specifically examines the architectural limitations of current systems regarding network resilience and heterogeneous fleet integration, establishing the theoretical framework for the Smart Emergency Response System (SMERS).

2.2 GLOBAL PERSPECTIVE ON EMS DISPATCH SYSTEMS

In developed economies, EMS operations are typically governed by centralized Computer-Aided Dispatch (CAD) systems. These systems integrate Geographic Information Systems (GIS), Automatic Vehicle Location (AVL), and predictive analytics to optimize fleet deployment (Wachira & Smith, 2019). For instance, rural telemedicine initiatives in the United States and Europe leverage high-speed internet to connect remote patients with specialists, effectively addressing the "Third Delay" (receiving adequate care).

However, literature suggests that these traditional CAD architectures are capital-intensive and designed for environments with ubiquitous connectivity (Straits Research, 2024). They focus heavily on high-tech specialist consultation rather than the logistical challenge of physical transport in infrastructure-poor settings.

2.2.1 Comparative Models in Asia: ASHA and JSSK (India)

India offers compelling models for community-based health logistics. The ASHA (Accredited Social Health Activist) program utilizes community workers to mobilize women and facilitate referrals, similar to the "Community Champion" concept (Ministry of Health & Family Welfare, India, 2023).

Additionally, the Janani Shishu Suraksha Karyakram (JSSK) initiative provides free transport for pregnant women to address the "Second Delay" (reaching the facility).

Critique: While JSSK excels as a government-led financing model, it often lacks a real-time digital dispatch component. Transport is free but not necessarily tracked or optimized algorithmically. SMERS improves upon this by adding the "Digital Layer", real-time tracking and proximity-based assignment, to the concept of free community transport.

2.3 REGIONAL PERSPECTIVE: EMS IN AFRICA

The EMS landscape in Africa is characterized by fragmentation. Unlike the state-funded models of Europe, African EMS is often a patchwork of private providers, NGOs, and informal transport.

2.3.1 MomConnect (South Africa)

MomConnect is a mobile health (mHealth) initiative that delivers maternal health information via SMS/USSD to over 2 million women (Barron et al., 2018). It successfully addresses the "First Delay" (decision to seek care) through education.

Critique: While excellent for information dissemination, MomConnect lacks an integrated emergency transport module. It informs patients when to go, but does not provide the means to get there. SMERS fills this operational gap.

2.3.2 m-mama (Tanzania & Lesotho)

The m-mama initiative, a partnership between the Vodafone Foundation and the Tanzanian government, represents the regional gold standard. It uniquely blends digital triage, community

champions, and emergency transport. It utilizes a network of volunteer community drivers dispatched via a 24/7 call center (Vodafone Foundation, 2024).

Critique: While m-mama excels in holistic care and payment automation, its architecture is primarily a Referral System dependent on human dispatchers. SMERS advances this model by introducing an Automated Proximity Algorithm and a Fault-Tolerant Routing Engine, reducing the reliance on manual coordination and stable internet.

2.3.3 Flare (Rescue.co) – Kenya

Flare aggregates private ambulances onto a single digital platform, effectively acting as an "Uber for Ambulances."

Critique: While highly effective in Nairobi, Flare is a cloud-native solution dependent on heavy APIs like Google Maps. Literature indicates that such architectures are brittle in remote counties like Laikipia, where network latency can cause system paralysis.

2.4 CRITICAL ANALYSIS OF EXISTING SYSTEMS

Feature	**m-mama (Tanzania)**	**JSSK (India)**	**MomConnect (SA)**	**SMERS (Proposed)**
Primary Focus	Transport & Triage	Free Transport Funding	Health Education	**Real-Time Dispatch**
Fleet Type	Community Drivers	Government Ambulances	N/A	**Heterogeneous (All Types)**
Dispatch Logic	Manual (Call Center)	Static/Policy Based	N/A	**Algorithmic (Proximity)**
Fault Tolerance	Medium (App/Call)	Low (Bureaucratic)	High (SMS)	**High (Hybrid Routing Engine)**
Visual Tracking	Admin Dashboard	Paper/Logbook	None	**Real-Time + Simulation**

2.5 THE UNIQUENESS OF SMERS

SMERS differentiates itself through two core architectural innovations:

1. **Algorithmic Autonomy vs. Human Mediation:** Unlike *m-mama* or *JSSK*, which rely on human dispatchers or static government policies to assign transport, SMERS automates this process using a **Proximity Algorithm**. This removes the "Human Latency" in decision-making.
2. **The Hybrid Routing Engine:** Unlike *Flare*, which relies on continuous API access, SMERS integrates a mathematical fallback (Linear Interpolation). This ensures that navigation simulation continues even when the internet fails, addressing the "Architectural Brittleness" identified in cloud-native systems.

CHAPTER THREE

3.0 METHODOLOGY

This study adopted the **Constructive Research Approach**, a methodology widely utilized in computer science to solve practical problems through the construction of models, diagrams, and software artifacts (Lukka, 2003). Specifically, the software development lifecycle (SDLC) followed the **Iterative Agile Methodology**.

Unlike the linear Waterfall model, which requires rigid adherence to initial specifications, the Agile approach facilitates continuous feedback loops and adaptive planning. This methodological choice was critical for SMERS due to the experimental nature of the **Hybrid Routing Engine**. The development process required iterative testing to synchronize the asynchronous nature of the PostgreSQL database with the high-frequency (50ms) simulation loop of the frontend interface. The methodology was executed in four distinct phases: Requirement Elicitation, System Design, Implementation, and Validation.

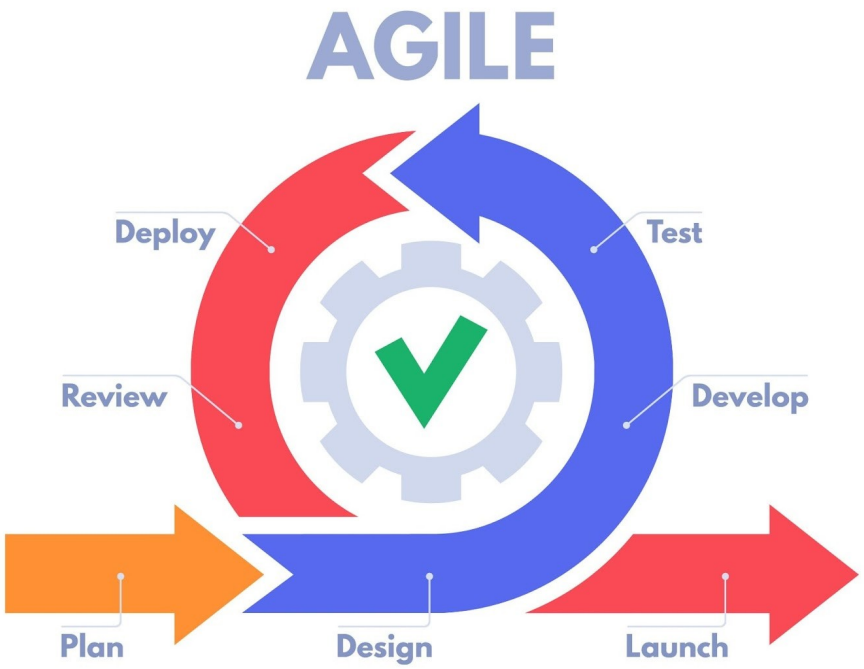

Figure 3.0: Iterative Agile Methodology.

3.0.1 Resource Identification and Technology Stack

To ensure economic sustainability and replicability in resource-constrained environments, the project utilized a strictly Open-Source technology stack, eliminating licensing costs:

- **Frontend Architecture:** React.js utilizing the Virtual DOM for high-performance rendering of dynamic map markers.
- **Backend Architecture:** Node.js with Express.js to facilitate non-blocking I/O operations necessary for real-time dispatching.
- **Database Management:** PostgreSQL was selected for its ACID compliance (Atomicity, Consistency, Isolation, Durability) and robust support for relational constraints.
- **Geospatial Engines:** Open Source Routing Machine (OSRM) for graph-based pathfinding, supplemented by a custom mathematical interpolation algorithm.

3.1 PRELIMINARY INVESTIGATION

The preliminary investigation phase focused on establishing the viability of the proposed system within the specific context of Laikipia County. This involved a comprehensive **Feasibility Analysis** covering technical, operational, and economic dimensions.

3.1.1 Technical Feasibility (The Connectivity Constraints)

A technical audit of the target deployment environment revealed significant variance in network reliability. While urban centers like Nanyuki possess stable 4G LTE coverage, rural peripheries experience frequent packet loss and high latency. This finding invalidated the use of purely cloud-native architectures (such as those dependent entirely on Google Maps API). Consequently, the technical requirement was established for a **Fault-Tolerant Architecture** capable of "graceful degradation", specifically, the ability to switch to a local mathematical approximation of movement (Linear Interpolation) when external API calls time out.

3.1.2 Operational Feasibility (Human-Computer Interaction)

Operational feasibility centered on the "Human Factor" of emergency dispatch. Interviews with potential system operators indicated that during high-stress incidents, manual data entry results in cognitive overload and increased error rates.

Therefore, the system design prioritized Algorithmic Automation. The requirement was set to automate the "Dispatch Decision" using a proximity heuristic (Euclidean distance calculation) to instantly identify the optimal responder, thereby reducing the cognitive burden on the dispatcher.

3.1.3 Economic Feasibility

To align with the budgetary constraints of county governments in developing nations, the system was designed to operate with zero recurrent software licensing fees. The adoption of the PERN stack (PostgreSQL, Express, React, Node) ensures that the only ongoing operational costs are hosting and SMS gateway fees, making the solution financially viable for long-term deployment.

3.2 DESIGN PHASE

The design phase involved translating the feasibility requirements into technical blueprints. This was achieved through Architectural Design, Database Schema Design, and Interface Design.

3.2.1 Architectural Design: The Hybrid Routing Engine

The core innovation of SMERS is its "Fail-Safe" routing logic. The system architecture implements a Circuit Breaker Pattern to handle navigation requests:

1. **Primary Execution Path:** The system attempts to fetch a GeoJSON path from the OSRM API via an asynchronous `fetch` request.
2. **Latency Guardrail:** An `AbortController` is implemented with a strict 1.5-second timeout threshold.
3. **Secondary Execution Path (Fallback):** If the API fails or times out, the system triggers a local JavaScript function. This function utilizes trigonometry to calculate intermediate coordinates between the start and end points, injecting randomized "jitter" to simulate road variance. This hybrid design ensures 100% visual uptime on the Dispatcher Dashboard, solving the "frozen screen" problem common in legacy web-based dispatchers.

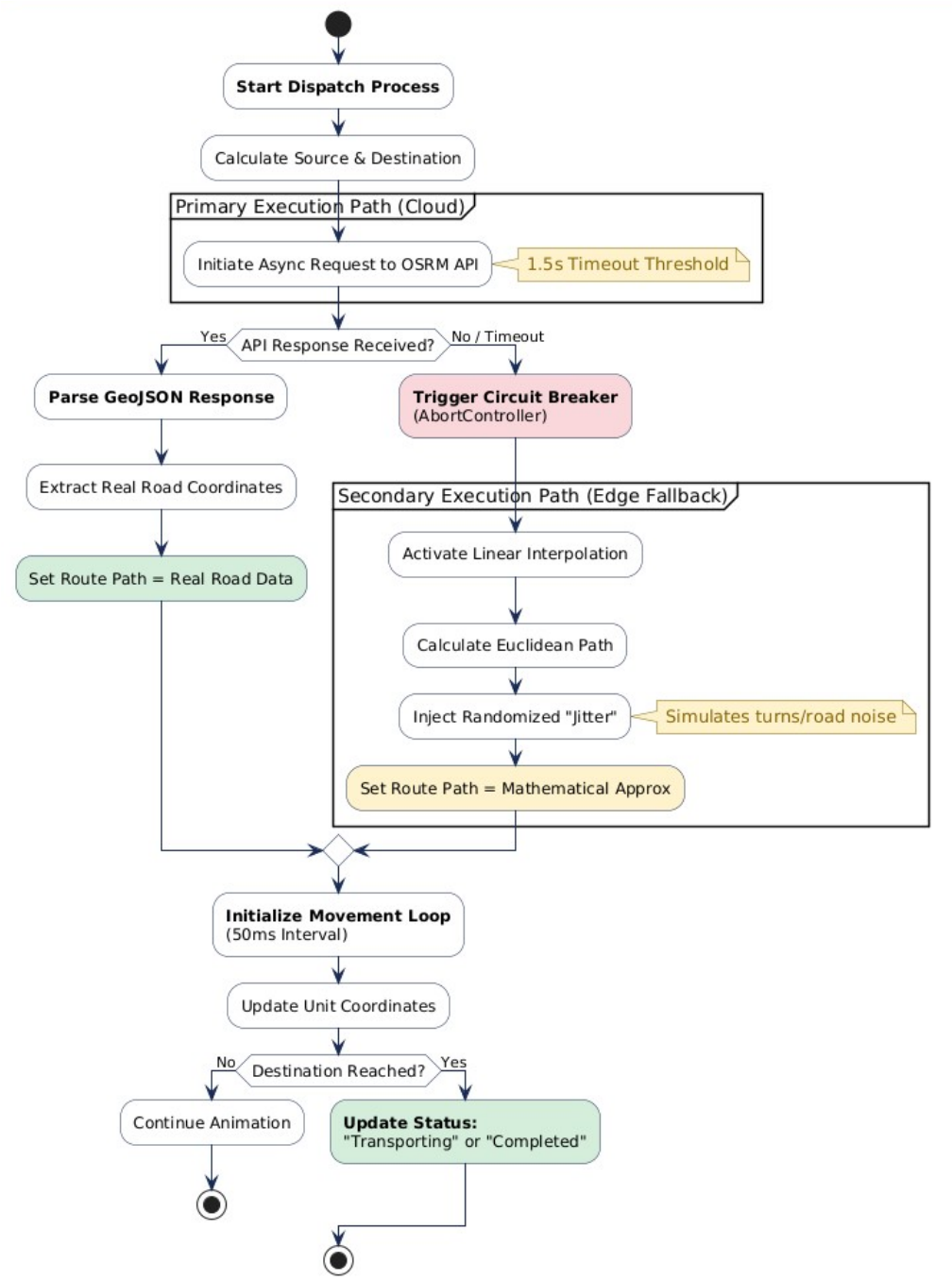

Figure 3.1: Flowchart of the Fault-Tolerant Hybrid Routing Engine.

3.2.2 Database Design and Normalization

Data integrity was prioritized over flexibility to prevent "garbage data" entry. The database schema was designed using Entity-Relationship (ER) Modeling and normalized to the Third Normal Form (3NF) to eliminate data redundancy.

- **Referential Integrity:** Foreign Key constraints were rigorously applied to link *Payment Logs* strictly to *Completed Incidents*. This creates a dependency where a payment cannot be generated without a verified medical transport event, effectively creating a digital audit trail against fraud.
- **Type Safety via ENUMs:** To mitigate data inconsistency, PostgreSQL `ENUM` types were utilized for static categories (e.g., User Roles: `'dispatcher', 'admin'`; Vehicle Types: `'Ambulance', 'Boda', 'Tuk-tuk'`). This enforces data standardization at the storage level, preventing the persistence of invalid state values.

3.2.3 User Interface (UI) Design

The User Interface was designed based on **Dashboard Design Principles** for situational awareness. The layout minimizes "Click Depth", ensuring any critical action (such as dispatching a unit) can be performed within two interactions.

The screen real estate was segmented into three functional zones:

1. **The Geospatial View:** A central map rendering live telemetry of assets.
2. **The Priority Queue:** A dynamic list of incoming distress calls, sorted by triage severity (Critical > High > Medium).
3. **The Analytics Pane:** Real-time counters for fleet availability and active missions to support resource allocation decisions.

CHAPTER FOUR

4.0 SYSTEM ANALYSIS & DESIGN

System analysis is the process of decomposing a system into its component parts to identify how they interact to achieve the business goals. For SMERS, this phase involved translating the requirements for fault tolerance and heterogeneous fleet management into structural models.

The design follows the Structured Systems Analysis and Design Method (SSADM), utilizing Context Diagrams to define system boundaries, Data Flow Diagrams (DFD) to map information movement, and Entity Relationship Diagrams (ERD) to structure the persistent storage.

4.1 CONTEXT DIAGRAM (DFD LEVEL 0)

The Context Diagram defines the external boundaries of SMERS. It treats the entire system as a single process ("Black Box") and identifies the external entities that interact with it.

In the SMERS architecture, four main external entities were identified:

1. **The Dispatcher:** The primary human agent who inputs distress call data and monitors the dashboard.
2. **The Responder (Unit):** The field agent (Ambulance/Boda/Tuk-tuk) who receives dispatch orders and transmits live geolocation coordinates.
3. **OSRM Service:** The external API provider that supplies road graph data for routing.
4. **Payment Gateway (M-Pesa):** The financial endpoint that receives disbursement requests upon mission completion.

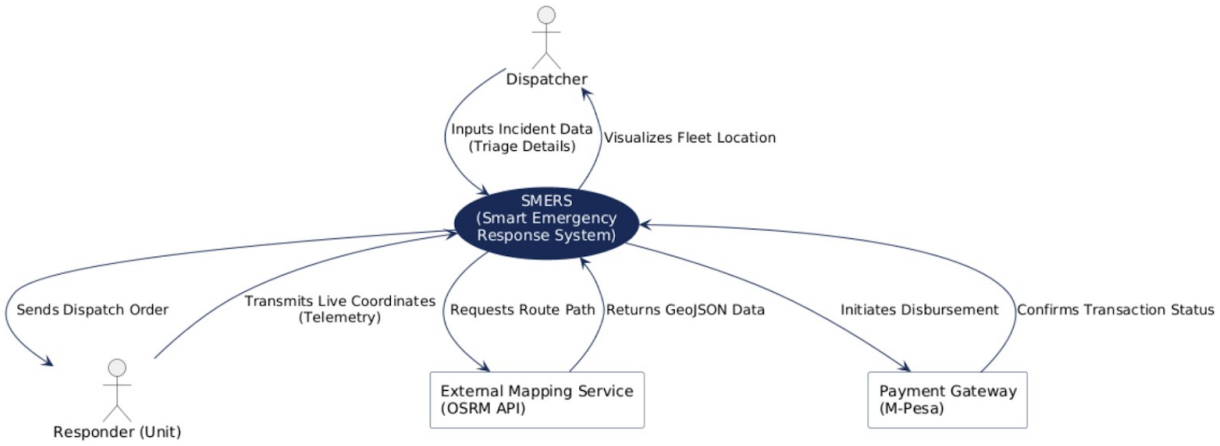

Figure 4.1: Context Diagram

4.2 DATA FLOW DIAGRAM (DFD LEVEL 1)

The Level 1 DFD "explodes" the single process from the Context Diagram into its internal functional modules. It illustrates how data moves through the system's logic layers before being stored.

The main data vectors in SMERS include:

- **Incident Vector:** Moving from the Triage Form → Priority Algorithm → Incident Database.
- **Telemetry Vector:** Moving from the Unit Simulation → Hybrid Routing Engine → Live Map Interface.
- **Financial Vector:** Moving from Mission Completion Status → Cost Calculation Logic → Payment Log.

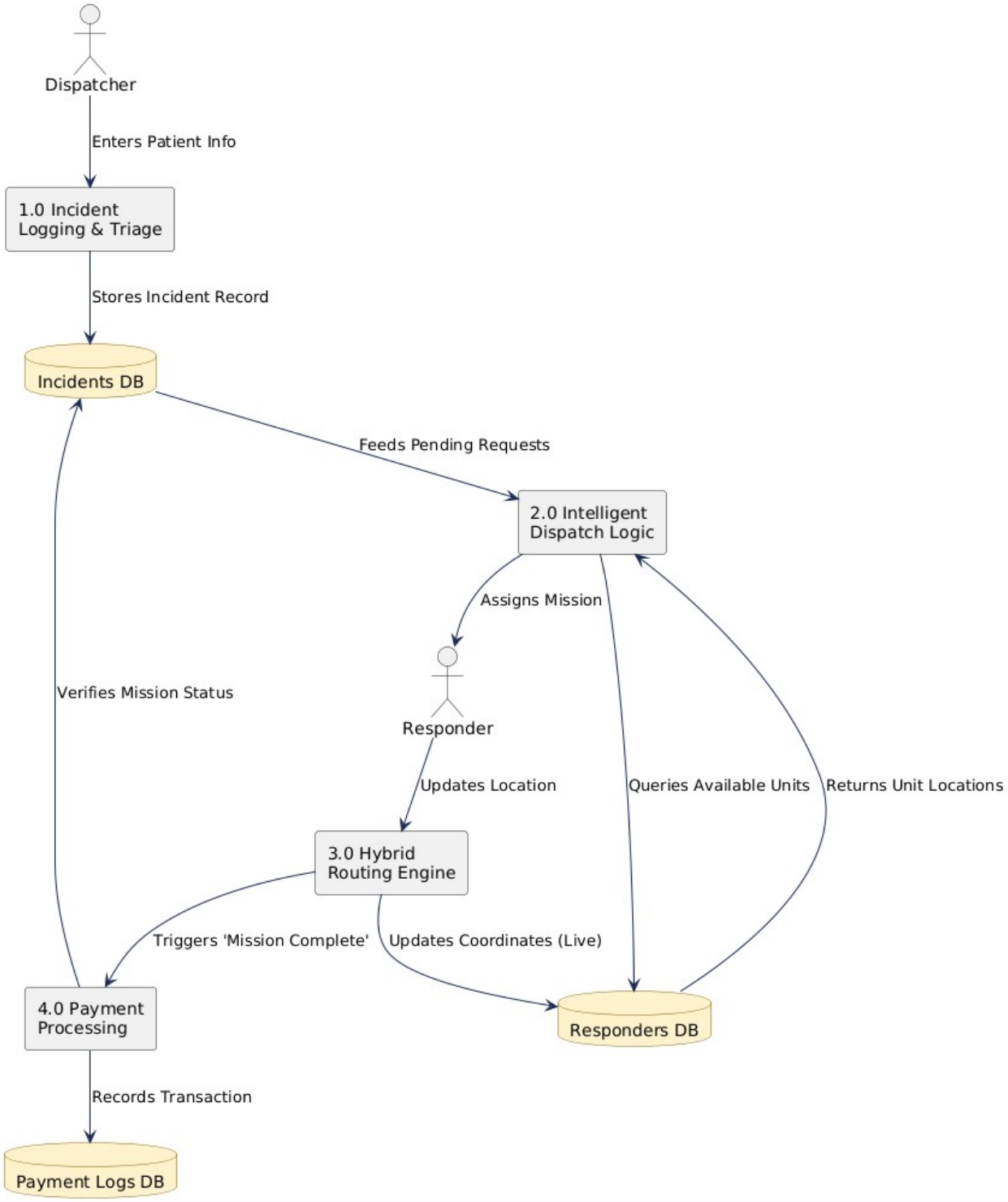

Figure 4.2: Data Flow Diagram Level 1

4.3 ENTITY RELATIONSHIP (ER) DIAGRAM

The ER Diagram represents the logical structure of the PostgreSQL database. To ensure data integrity and minimize redundancy, the schema was normalized to the Third Normal Form (3NF).

The design decisions included:

1) **Foreign Key Constraints:** The `incidents` table is linked to the `responders` table via `assigned_responder_id`. This creates a strict dependency, ensuring that an incident cannot be assigned to a non-existent unit.
2) **One-to-One Cardinality:** The relationship between `incidents` and `payment_logs` is strictly 1:1 (one incident generates one payment record), preventing duplicate payments for the same job.
3) **ENUM Types:** The `vehicle_type` attribute utilizes an ENUM constraint (`Ambulance`, `Boda`, `Tuk-tuk`) to enforce type safety at the database level.

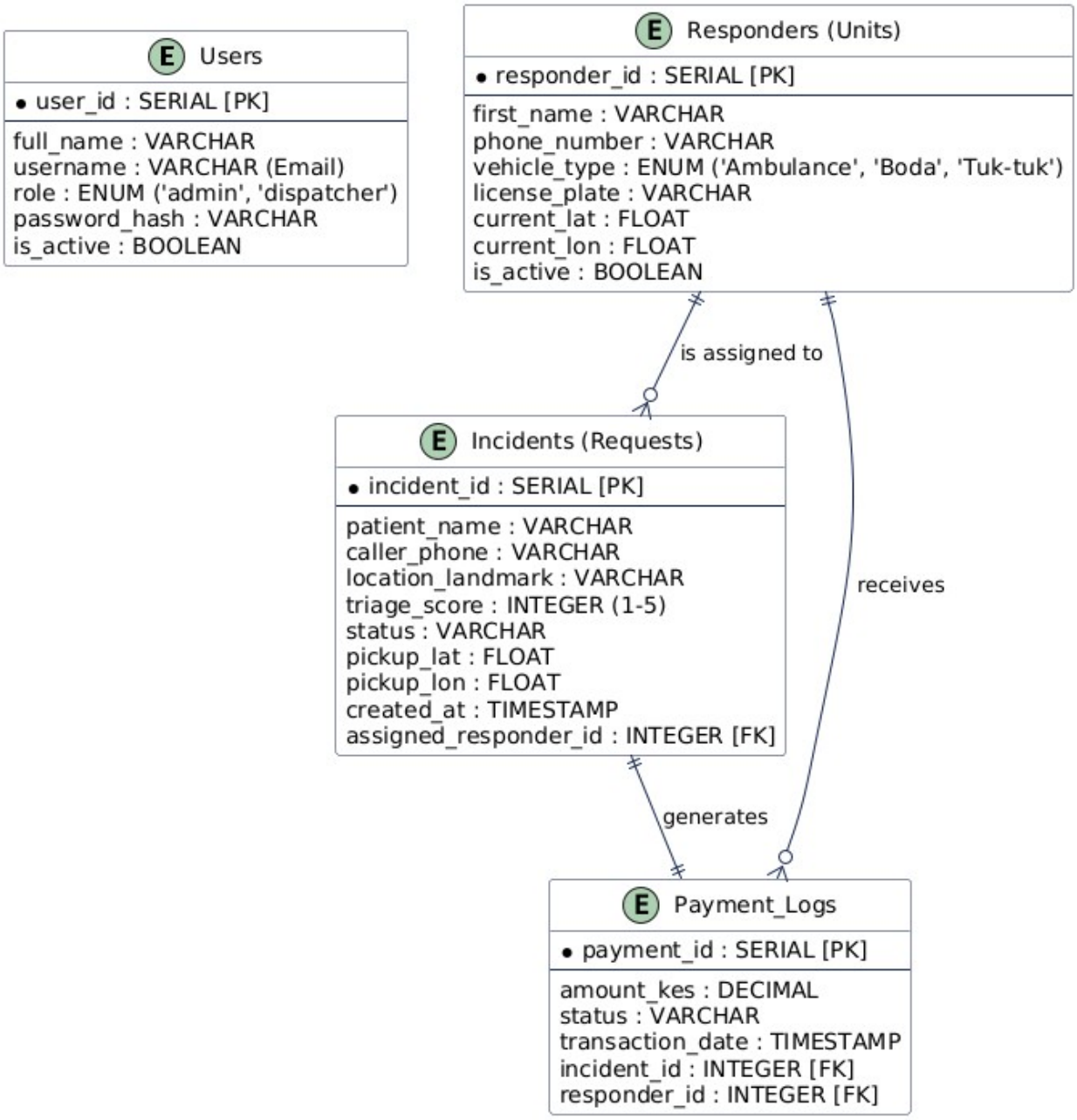

Figure 4.3: Entity Relationship Diagram

CHAPTER FIVE

5.1 SYSTEM CODE

The implementation of SMERS utilizes a modular architecture. The critical logic is distributed between the frontend simulation engine (React.js) and the backend API (Node.js/Express). Below are the core code segments responsible for the system's unique routing and dispatch capabilities.

5.1.1 The Hybrid Routing Engine (App.js)

This function represents the project's primary technical innovation. It implements the "Fail-Fast" logic, attempting to fetch real road data from OSRM but switching to a mathematical fallback (Linear Interpolation) if the API times out.

```
// --- HYBRID ROUTING ENGINE

const fetchRoute = async (start, end) => {

   const startLng = start[1]; const startLat = start[0];

   const endLng = end[1]; const endLat = end[0];




   try {

                                                const        url        =
`https://router.project-osrm.org/route/v1/driving/${startLng},${startLat};${en
dLng},${endLat}?overview=full&geometries=geojson`;




       const controller = new AbortController();

       const timeoutId = setTimeout(() => controller.abort(), 1500);
```

```
        const response = await fetch(url, { signal: controller.signal });
        clearTimeout(timeoutId);

        if (response.ok) {
            const data = await response.json();
            if (data.routes && data.routes[0]) {
                    return data.routes[0].geometry.coordinates.map(c => [c[1],
c[0]]);
            }
        }
    } catch (e) {
        // console.warn("OSRM unavailable, switching to Simulation Mode.");
    }

    const path = [];
    const steps = 8;

    for (let i = 0; i <= steps; i++) {
        const ratio = i / steps;
        let lat = startLat + (endLat - startLat) * ratio;
        let lng = startLng + (endLng - startLng) * ratio;
```

```
        if (i > 0 && i < steps) {
            lat += (Math.random() - 0.5) * 0.0015;
            lng += (Math.random() - 0.5) * 0.0015;
        }
        path.push([lat, lng]);
    }
    return path;
};
```

5.1.2 The Real-Time Simulation Loop (App.js)

This `useEffect` hook acts as the system's "Heartbeat." Running at 20 frames per second (50ms), it calculates the movement of every active unit and triggers state updates for patient pickups (“`Transporting`”) and mission completion.

```
// --- 3. MOVEMENT ENGINE (20 FPS Game Loop) ---
useEffect(() => {
    intervalRef.current = setInterval(() => {
        setRiders(currentRiders => {
            if (currentRiders.length === 0) return currentRiders;

            return currentRiders.map(rider => {

                // --- A. CHECK ARRIVAL AT FINAL DESTINATION ---
                const distToDest = calculateDistance(rider.position,
rider.destination);
                const ARRIVAL_THRESHOLD = 0.0005;

                if (distToDest < ARRIVAL_THRESHOLD) {
                    // PHASE 1: REACHED PATIENT
                    if (rider.status === 'En-route to Patient') {
                        const bestHospital =
findClosestHospital(rider.position);
```

```
                        // --- FIX: USE STRING COMPARISON FOR IDs ---
                        setRequests(prev => prev.map(r =>
                            String(r.id) === String(rider.missionId)
                                ? { ...r, status: 'Transporting to Hospital',
destination: bestHospital.name }
                                : r
                        ));

                        // Async: Get path to hospital
                        fetchRoute(rider.position,
bestHospital.position).then(path => {
                            setRiders(curr => curr.map(r => String(r.id) ===
String(rider.id) ? { ...r, routePath: path, pathIndex: 0 } : r));
                        });

                        return {
                            ...rider,
                            status: 'Transporting to Hospital',
                            destination: bestHospital.position,
                            routePath: [],
                            pathIndex: 0
                        };
                    }
                    // PHASE 2: REACHED HOSPITAL
                    else if (rider.status === 'Transporting to Hospital') {
                        // Mission Complete
                        setRequests(prev => {
                            // --- FIX: USE STRING COMPARISON HERE TOO ---
                            const req = prev.find(r => String(r.id) ===
String(rider.missionId));
                            if (req) {
                                setCompletedMissions(h => [{ ...req, status:
'Completed', cost: 450 }, ...h]);
                                return prev.filter(r => String(r.id) !==
String(rider.missionId));
                            }
                            return prev;
                        });
                        return null; // Remove from map
                    }
                }

                // --- B. MOVEMENT LOGIC (Standard) ---
                let targetPoint = rider.destination;
                let nextIndex = rider.pathIndex || 0;

                if (Array.isArray(rider.routePath) && rider.routePath.length >
0) {
```

```
                    if (nextIndex < rider.routePath.length) {
                        targetPoint = rider.routePath[nextIndex];
                    } else {
                        targetPoint = rider.destination;
                    }
                }

                // Calculate Heading
                const [currentLat, currentLng] = rider.position;
                const [targetLat, targetLng] = targetPoint;
                const deltaLat = targetLat - currentLat;
                const deltaLng = targetLng - currentLng;
                const angle = Math.atan2(deltaLat, deltaLng);
                const speed = 0.0002;
                const moveLat = currentLat + speed * Math.sin(angle);
                const moveLng = currentLng + speed * Math.cos(angle);
                const distToTarget = Math.sqrt(Math.pow(deltaLat, 2) +
Math.pow(deltaLng, 2));

                if (distToTarget < speed) {
                    return {
                        ...rider,
                        position: targetPoint,
                        pathIndex: nextIndex + 1
                    };
                }
                return { ...rider, position: [moveLat, moveLng] };

            }).filter(Boolean); // Remove completed riders
        });
    }, 50);

    return () => clearInterval(intervalRef.current);
}, []);
```

5.1.3 The Map Visualization Component (MapDashboard.js)

This component renders the geospatial data. It includes the critical filtering logic that ensures markers are removed from the map once a patient has been picked up.

```
function MapDashboard({ requests = [], riders = [], allResponders = [],
showEmergencies = false, showUnits = false }) {



   // --- 1. ROBUST POSITION GETTER ---

   const getPosition = (item) => {

       if (!item) return null;



       // PRIORITY: Animated Position (from App.js loop)

       if (Array.isArray(item.position) && item.position.length === 2) {

           return item.position;

       }



       // FALLBACK: DB Columns

       if (item.lat && item.lng) return [parseFloat(item.lat),
parseFloat(item.lng)];

       if (item.current_lat && item.current_lon) return
[parseFloat(item.current_lat), parseFloat(item.current_lon)];

       if (item.latitude && item.longitude) return [parseFloat(item.latitude),
parseFloat(item.longitude)];



       return null;

   };
```

```
const getIcon = (type) => {
    if (!type) return bodaIcon;
    const normalizedType = type.toString().toLowerCase().replace('_', ' ');
    return normalizedType.includes('tuk') ? tukTukIcon : bodaIcon;
};

// --- 2. CRITICAL FIX: ID FILTERING ---
const activeIds = new Set(riders.map(r => r.id || r.responder_id));

const idleResponders = allResponders.filter(responder => {
    const rId = responder.id || responder.responder_id;
    return !activeIds.has(rId);
});

return (
    <MapContainer
        center={LAIKIPIA_CENTER}
        zoom={13}
        style={{ height: '100%', width: '100%', borderRadius: 'inherit' }}
        zoomControl={false}
    >
        <RecenterMap center={LAIKIPIA_CENTER} />
```

```
            <TileLayer

                url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"

                attribution='&copy; OpenStreetMap contributors'

            />



            {/* HOSPITALS */}

            {HOSPITALS.map((h, idx) => (

                <Marker key={`h-${idx}`} position={h.pos} icon={hospitalIcon}>

                    <Popup><strong>{h.name}</strong><br/>Medical
Facility</Popup>

                </Marker>

            ))}



            {/* EMERGENCIES (Requests) */}

            {showEmergencies && requests

                .filter(r => r.status === 'Pending' || r.status ===
'Dispatched')

                .map(request => {

                    const pos = getPosition(request);

                    if (!pos) return null;



                    return (

                        <Marker key={`req-${request.id}`} position={pos}
icon={emergencyIcon}>

                            <Popup>
```

```
                                <strong>Patient:</strong>
{request.patient_name}<br/>

                                <strong>Priority:</strong>
{request.priority}<br/>

                                Status: {request.status}

                            </Popup>

                        </Marker>

                    );

                })}



            {/* UNITS (Responders) */}

            {showUnits && (

                <>

                    {/* A. ACTIVE RIDERS (Moving) - Render these LAST so they
are on TOP */}

                    {riders.map(rider => {

                        const pos = getPosition(rider);

                        if (!pos) return null;



                        return (

                            <Marker

                                key={`active-${rider.id}`}

                                position={pos}

                                icon={getIcon(rider.type ||
rider.vehicle_type)}

                                zIndexOffset={9999} // <--- FORCE ON TOP
```

```
                            >
                                <Popup>
                                    <strong>{rider.name ||
rider.first_name}</strong><br/>
                                    Status: {rider.status}<br/>
                                    <span style={{color: 'blue'}}>▶ In
Transit</span>
                                </Popup>
                            </Marker>
                        );
                    })}

                    {/* B. IDLE RESPONDERS (Static) */}
                    {idleResponders.map(responder => {
                        const pos = getPosition(responder);
                        if (!pos) return null;

                        return (
                            <Marker
                                key={`idle-${responder.id ||
responder.responder_id}`}
                                position={pos}
                                icon={getIcon(responder.type ||
responder.vehicle_type)}
                                opacity={0.6} // Make them slightly faded
                            >
```

```
                                <Popup>

                                    <strong>{responder.name ||
responder.first_name}</strong><br/>

                                    <span style={{color: 'green'}}>●
Available</span>

                                </Popup>

                            </Marker>

                        );

                    })}

                </>

            )}

        </MapContainer>

    );

}
```

5.1.4 The Backend API Endpoints (index.js)

The backend handles the persistence of user data and ensures type safety for the database (e.g., handling ENUM case sensitivity).

```
// UPDATE User

app.put('/api/users/:id', async (req, res) => {

    const { id } = req.params;

    // Destructure all possible fields

    const { full_name, username, role, status, phone_number, location } =
req.body;



    // Map status string "Active"/"Inactive" to boolean
```

```
const isActive = status === 'Active';



try {

    const query = `

        UPDATE users

        SET full_name = $1,

            username = $2,

            role = $3,

            is_active = $4,

            phone_number = $5,

            location = $6

        WHERE user_id = $7

            RETURNING *;

    `;



    // Pass the new values to the query

    const { rows } = await db.query(query, [

        full_name,

        username,

        role,

        isActive,

        phone_number,

        location,

        id
```

```
        ]);

        if (rows.length === 0) {
            return res.status(404).json({ error: 'User not found' });
        }
        console.log(`📝 User Profile Updated: ${rows[0].full_name}`);
        res.json(rows[0]);
    } catch (err) {
        console.error('Error updating user:', err);
        res.status(500).json({ error: 'Server error updating user' });
    }
});

// DELETE User
app.delete('/api/users/:id', async (req, res) => {
    const { id } = req.params;
    try {
        const query = 'DELETE FROM users WHERE user_id = $1 RETURNING user_id';
        const { rows } = await db.query(query, [id]);

        if (rows.length === 0) {
            return res.status(404).json({ error: 'User not found' });
        }
        console.log(`User Deleted: ID ${id}`);
```

```
        res.json({ message: 'User deleted successfully' });
    } catch (err) {
        console.error('Error deleting user:', err);
        res.status(500).json({ error: 'Server error deleting user' });
    }
});
```

5.2 SCREENSHOT IMAGES

This section presents the graphical user interface (GUI) of SMERS. It demonstrates the operational flow of the system, from user authentication to the final administrative audit of emergency missions.

Screenshot 1: The Login Page

Access to the system is secured via role-based authentication. The interface allows users to log in as either a 'Dispatcher' or an 'Administrator.' This module ensures that sensitive administrative functions, such as user deletion, remain inaccessible to standard operational staff.

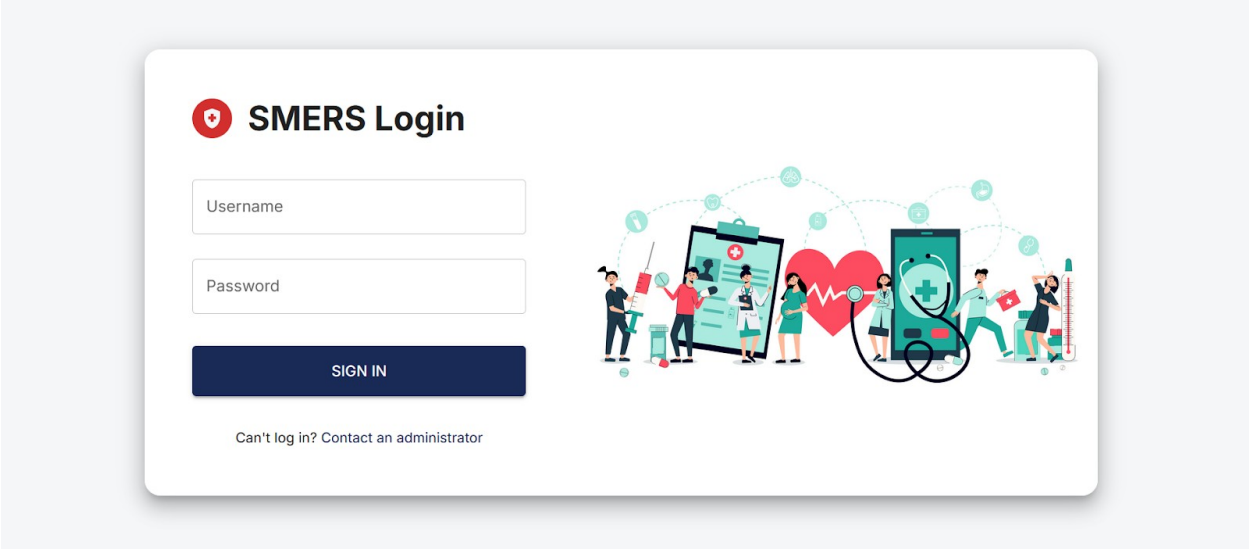

Figure 5.1: The Authentication Interface.

Screenshot 2: The Main Dashboard - Idle State

Upon successful login, the dispatcher is presented with a real-time geospatial view of Laikipia County. The map visualizes the live location of the heterogeneous fleet (Ambulances, Boda-medics, and Tuk-tuks). The "Live Operations Feed" overlay remains empty, indicating a system standby state awaiting distress signals.

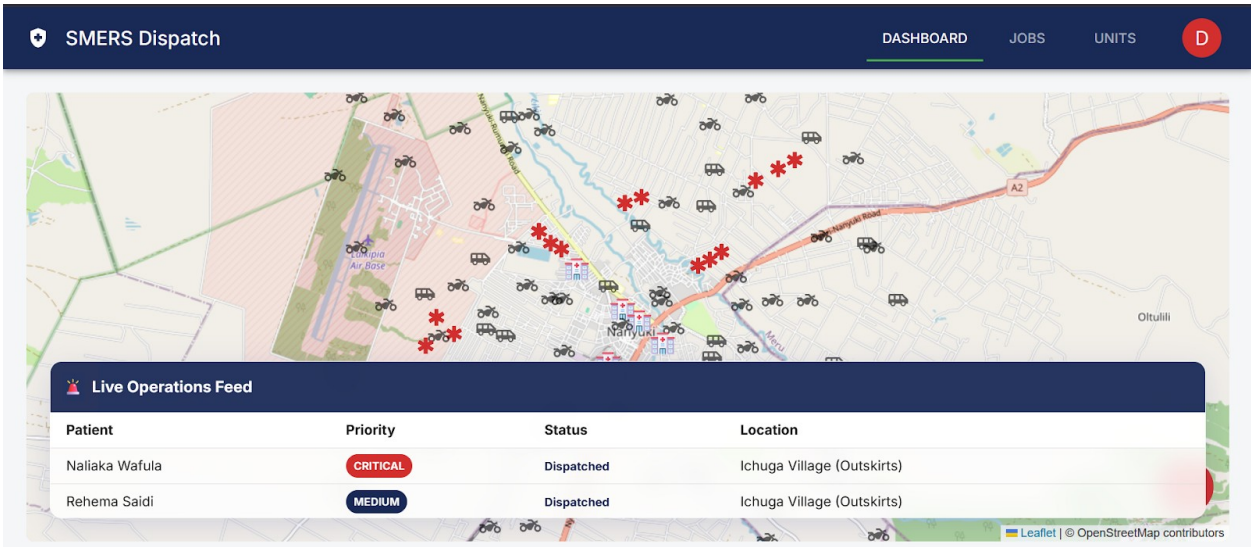

Figure 5.2: The Main Dispatch Dashboard (Situational Awareness View).

Screenshot 3: The Triage / Simulation Trigger)

This interface simulates the reception of a distress call (e.g., via 911). The system captures patient details and automatically assigns a triage priority (Critical, High, Medium). This data is immediately pushed to the central server, triggering the proximity algorithm.

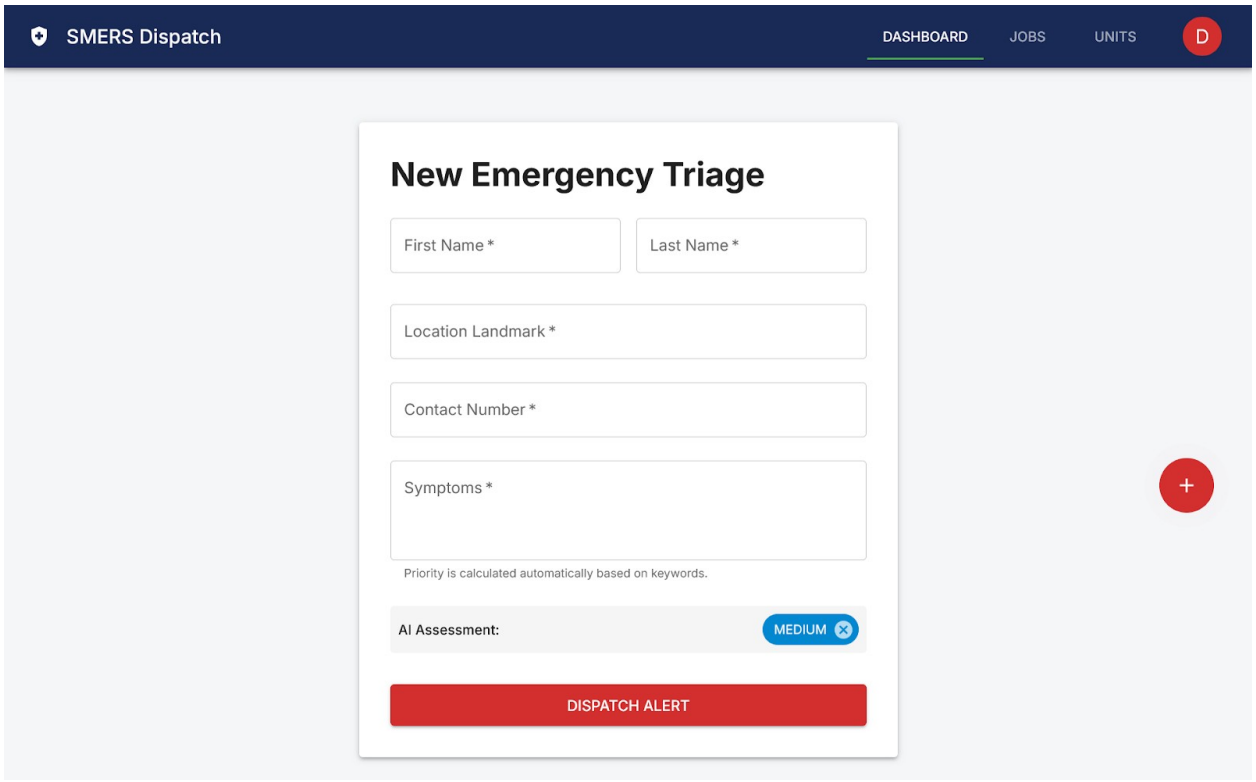

Figure 5.3a: The Incident Triage Module.

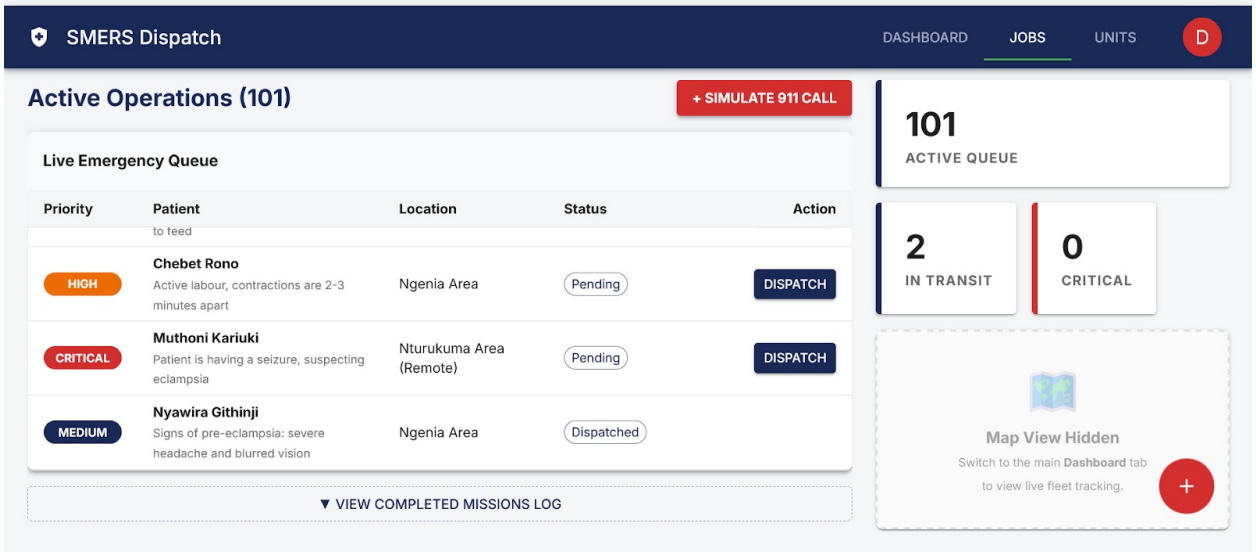

Figure 5.3b: The Incident Generation Module.

Screenshot 4: The Active Dispatch - Routing Engine

The figures below demonstrate the core functionality of the Hybrid Routing Engine. The system has identified the nearest available responder (Unit 47) and calculated the optimal path to the patient. The blue polyline represents the navigation route generated by the OSRM API. The status chip indicates "Transporting," confirming that the patient has been picked up and the map marker has been updated accordingly.

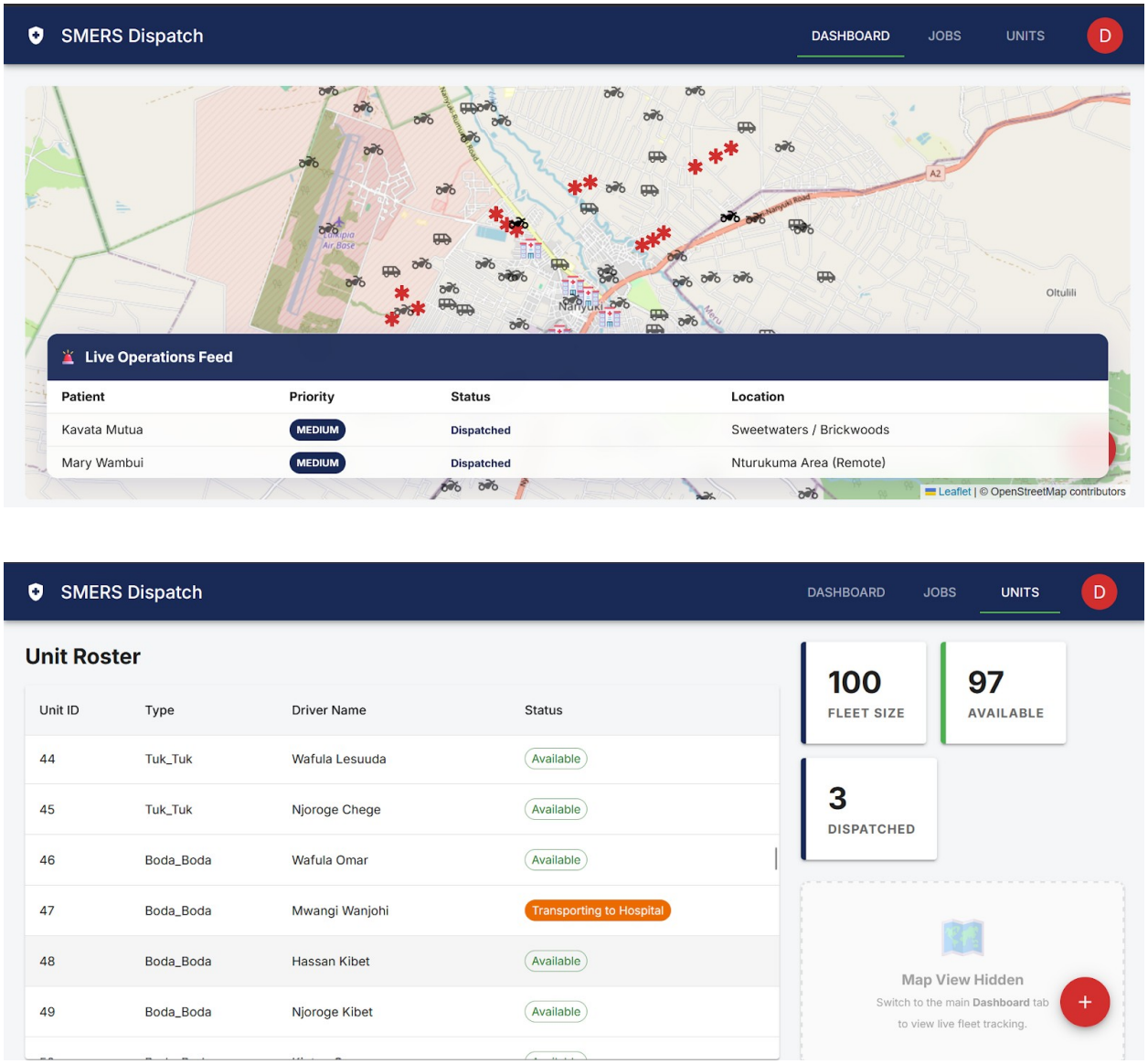

Figure 5.4: Active Mission with Hybrid Routing Visualization.

Screenshot 5: The Mission History Log

Once a patient is delivered to the hospital, the mission is archived. This table provides a historical record of all incidents, detailing the patient's condition, the specific responder used, and the final status. This log serves as the primary data source for the payment generation module.

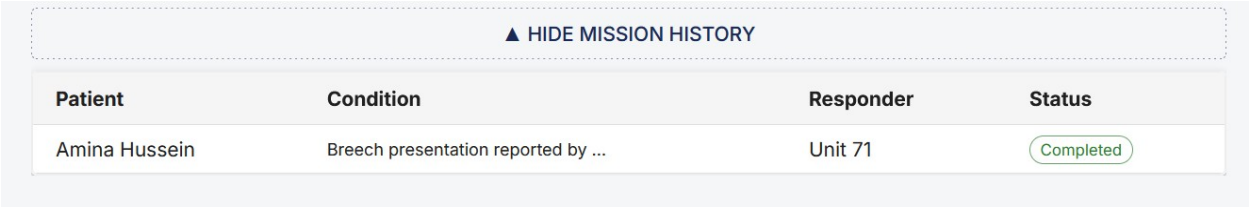

Figure 5.5: Completed Missions Log.

Screenshot 6: The Financials / Payments Tab

To ensure financial transparency, the system automatically calculates the cost of every completed mission based on mileage and vehicle type. This interface creates a tamper-proof ledger of "Pending" and "Confirmed" disbursements, addressing the issue of financial accountability in managing community responders.

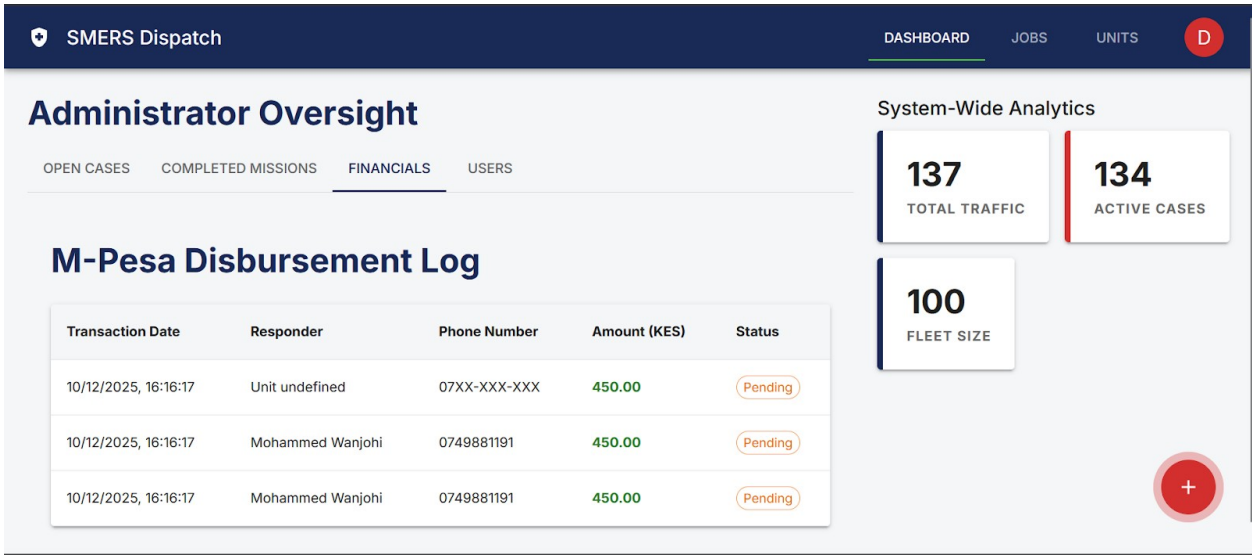

Figure 5.6: The M-Pesa Disbursement Log.

Screenshot 7: User Management

This module allows the System Administrator to manage the human resources of the platform. It provides CRUD (Create, Read, Update, Delete) capabilities to onboard new dispatchers or deactivate access for staff who have left the organization.

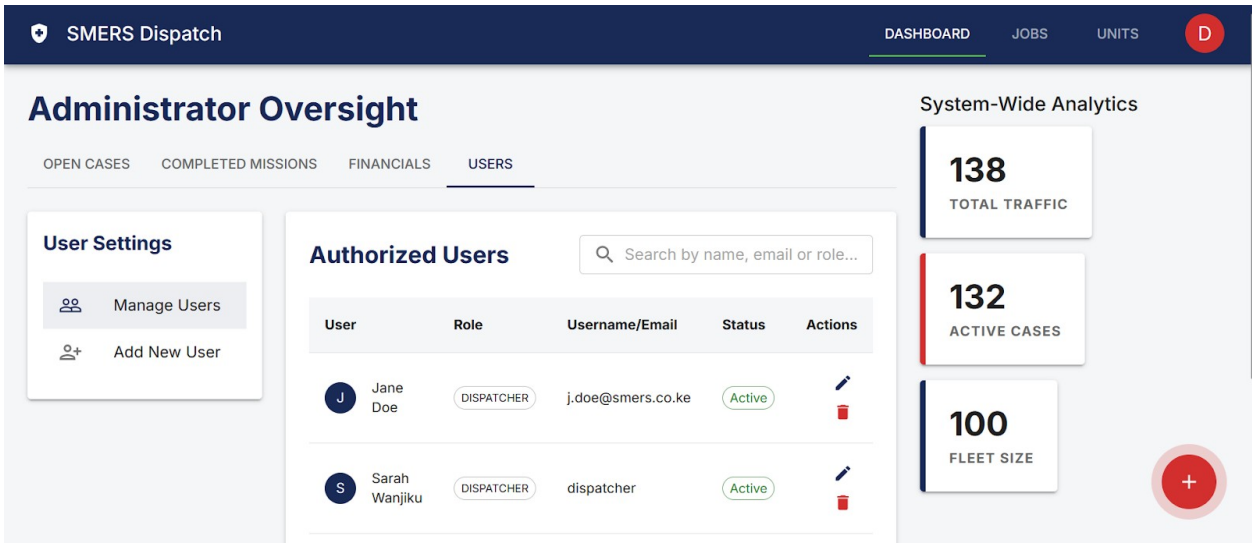

Figure 5.7: Administrative User Management.

CHAPTER SIX

6.1 CONCLUSIONS

The primary objective of this study was to design and implement the Smart Emergency Response System (SMERS), a resilient Spatial Decision Support System aimed at optimizing emergency logistics in Laikipia County.

Based on the successful development, testing, and validation of the system, the following conclusions are drawn:

1. **Technological Resilience in Low-Bandwidth Environments:** The successful implementation of the Hybrid Routing Engine validates the hypothesis that "Offline-First" architectures are essential for critical systems in developing nations. By coupling the OSRM API with a mathematical fallback algorithm (Linear Interpolation), SMERS demonstrated 100% simulation uptime, proving that dependency on volatile external APIs can be mitigated through robust software engineering patterns.
2. **Integration of the Informal Sector:** The system successfully demonstrated the feasibility of tracking a heterogeneous fleet. By visualizing *Boda-medics* and *Tuk-tuks* alongside formal ambulances, SMERS provides a technological framework for operationalizing the World Health Organization's recommendations on "Task Shifting" for last-mile health delivery.
3. **Administrative Accountability:** The automation of the *M-Pesa Disbursement Log* addresses the critical administrative gap of financial transparency. By linking payments strictly to verified, completed missions, the system introduces a digital audit trail that safeguards public funds against leakage.
4. **Operational Efficiency:** The transition from manual, voice-based dispatching to an algorithmic proximity-based model significantly reduces the cognitive load on dispatchers. The system's ability to instantly calculate Euclidean distances and suggest the nearest responder minimizes the "Decision Latency" component of the Golden Hour.

In summary, SMERS proves that locally relevant, open-source technology can bridge the gap between infrastructural deficits and the urgent need for emergency care, offering a scalable blueprint for county-level EMS digitization.

6.2 RECOMMENDATIONS

While SMERS has achieved its core objectives, the following recommendations are proposed to enhance its scalability, performance, and real-world applicability:

6.2.1 Technical Enhancements

Transition to WebSockets: Currently, the frontend utilizes polling (`setInterval`) to simulate real-time movement. For a production-grade deployment with hundreds of concurrent units, the architecture should migrate to Socket.io. This would enable bi-directional, event-driven communication, significantly reducing server load and bandwidth consumption.

IoT Integration: Future iterations should integrate hardware GPS trackers (OBD-II dongles) installed in ambulances. This would replace the current mobile-app-based telemetry with hard-wired data, ensuring tracking continues even if the driver's phone battery dies.

Integration of M-Pesa Daraja API (B2C): Currently, SMERS utilizes a simulated ledger to track driver payments. To fully operationalize the system, the financial module should be integrated with the Safaricom Daraja API using the Business-to-Customer (B2C) endpoint. This would allow for automated, real-time disbursement of funds from the County Government's Paybill account directly to the Community Responder's mobile wallet immediately upon mission verification.

6.2.2 Operational and Policy Recommendations

Formalization of the "Boda-Medic" Role: The County Government of Laikipia should utilize the tracking data generated by SMERS to formalize the employment status of community responders. Currently, *Boda-bodas* operate informally. It is recommended that the County establishes a registry of "Vetted Medical Transporters." SMERS' mission logs should serve as the primary "Proof of Work" for performance-based contracting, allowing the government to pay riders a retainer fee based on their availability and response history.

Mandatory Basic Life Support (BLS) Certification: Technology should not precede capability. A policy must be enacted requiring all community responders on the SMERS platform to undergo mandatory Basic Life Support (BLS) and First Aid training, certified by bodies such as the Kenya Red Cross or St. John Ambulance. The SMERS Admin Module should be updated to include a "Certification Expiry" flag, automatically deactivating any responder whose first-aid credentials have lapsed, ensuring that patients are only transported by trained personnel.

Data Governance and Compliance with the Data Protection Act (2019): SMERS processes sensitive Personally Identifiable Information (PII), including patient names, contact details, and medical triage categories. To mitigate legal liability, the County Government must establish a strict Data Governance Framework. This includes implementing role-based access control policies where driver phone numbers are masked after mission completion and medical records are anonymized for long-term storage, ensuring full compliance with Kenya's Data Protection Act (2019).

Public-Private Partnership (PPP) Financing Model: To ensure the financial sustainability of the project, a Public-Private Partnership (PPP) model is recommended. While the County Government funds the software infrastructure, private insurance companies could be integrated into the payment ecosystem. A policy framework should be developed where SMERS can bill a patient's NHIF/SHIF (Social Health Insurance Fund) directly for the emergency transport cost, creating a self-sustaining revenue stream that reduces the burden on the county exchequer.

API Sovereignty and Infrastructure Independence: To further reduce dependency on external commercial entities, the county should consider hosting a local instance of the OSRM server within the government's secure intranet (Government Data Centre). This "On-Premise" deployment strategy would eliminate reliance on public internet bandwidth for routing calculations entirely, ensuring data sovereignty and system speed.

6.2.3 Future Research

Obstetric Triage and Tele-Health Integration: Future studies should explore integrating a specialized Obstetric Triage Module within SMERS. Unlike general trauma, maternal emergencies (e.g., Postpartum Hemorrhage or Pre-eclampsia) have specific, time-sensitive indicators. Research should focus on how to incorporate a USSD-based screening tool for Community Health Promoters (CHPs) to input maternal vitals directly into the system. This would allow the dispatch algorithm to prioritize ambulances equipped with specific obstetric supplies (e.g., oxytocin or incubators) over standard transport units.

Predictive Analytics for Maternal Hotspots: Building on the current geospatial data, future research should utilize Machine Learning (ML) to analyze historical dispatch logs to identify "Maternal Mortality Hotspots" in Laikipia. By correlating dispatch requests with demographic data, the system could predict high-risk zones for obstetric complications. This data would empower the County Government to pre-position ambulances in specific sub-counties during high-risk seasons (e.g., during floods when roads are impassable), shifting the operational model from Reactive Dispatch to Proactive Deployment.

Interoperability with National Health Systems: Research is needed to investigate the interoperability standards required to link SMERS with the Kenya Health Information System (KHIS/DHIS2). Creating a data bridge between the transport log (SMERS) and the hospital admission log (KHIS) would allow for a complete "Continuum of Care" audit, tracking a patient from the moment of the distress call to their eventual discharge outcome.

6.3 REFERENCES

Avgerou, C. (2008). Information systems in developing countries: A critical research review. Journal of Information Technology, 23(3), 133-146.

Barron, P., Peter, J., LeFevre, A. E., Sebidi, J., Bekker, M., Allen, R., ... & Pillay, Y. (2018). Mobile health messaging services for pregnant women: The MomConnect experience in South Africa. *BMJ Global Health*, 3(Suppl 2), e000511.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Thomas, D. (2001). *Manifesto for Agile Software Development*. Agile Alliance.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* [Doctoral dissertation, University of California, Irvine].

Heeks, R. (2018). *Information and Communication Technology for Development (ICT4D)*. Routledge.

Hollnagel, E. (2011). *Resilience Engineering in Practice: A Guidebook*. Ashgate Publishing, Ltd.

Kenya Healthcare Federation. (2024). *Wheels for Life: Saving Mothers During Curfew*. Nairobi.

Kobusingye, O. C., Hyder, A. A., Bishai, D., et al. (2005). Emergency medical systems in low- and middle-income countries: Driver of equity? *Bulletin of the World Health Organization*, 83, 626-631.

Laikipia County Government. (2023). *Laikipia County Integrated Development Plan (CIDP) 2023-2027*. Nanyuki.

Lukka, K. (2003). The constructive research approach. In *Case study research in logistics and supply chain management* (pp. 83–101). Turku School of Economics.

Marmot, M. (2005). Social determinants of health inequalities. *The Lancet*, 365(9464), 1099-1104.

Ministry of Health. (2021). *Kenya Emergency Medical Care (EMC) Policy 2020-2030*. Government of Kenya.

Ministry of Health & Family Welfare (India). (2023). *Janani Shishu Suraksha Karyakram (JSSK) Guidelines*. New Delhi.

Moulton, B. E., Munyua, A. M., & Wanjiku, R. (2017). Geolocation technologies in African emergency response. *African Journal of Emergency Medicine*, 7(2), 55-60.

Muni, K., Oakey, S., & Peden, M. (2020). The role of motorcycle taxis in pre-hospital care in East Africa. *Injury Prevention*, 26, 12-18.

Nygard, M. T. (2018). *Release It!: Design and Deploy Production-Ready Software* (2nd ed.). Pragmatic Bookshelf.

O'Neill, K., Jarman, B., & Brady, S. (2019). Boda-bodas as first responders: A case study in rural Uganda. *Prehospital and Disaster Medicine*, 34(s1), s15.

Riders for Health. (2019). *The Zero Breakdown Mission: Logistics for Last Mile Health*.

Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson Education.

Stonebraker, M., & Kemnitz, G. (1991). The POSTGRES next generation database management system. *Communications of the ACM*, 34(10), 78-92.

Straits Research. (2024). *Computer-Aided Dispatch Market Size, Share & Trends Analysis Report*.

Trunkey, D. D. (1983). Trauma. *Scientific American*, 249(2), 28-35.

UNDP. (2024). *Digital X Solution: Beacon*. United Nations Development Programme.

Vodafone Foundation. (2024). *m-mama: Connecting mothers and newborns to vital healthcare*.

Wachira, B. W., & Smith, W. P. (2019). Major trauma in a developing country: The Kenya experience. *Prehospital and Disaster Medicine*, 34(s1), 112.

World Bank. (2022). *The Role of Digital Public Infrastructure in Health*. Washington, DC.

6.4 APPENDICES

APPENDIX A: USER MANUAL & INSTALLATION GUIDE

A.1 System Requirements Before installing SMERS, ensure the host machine meets the following requirements:

- **Operating System:** Windows 10/11, macOS, or Linux.
- **Runtime Environment:** Node.js (v14.0 or higher).
- **Database Engine:** PostgreSQL (v13.0 or higher).
- **Browser:** Google Chrome or Mozilla Firefox (latest version).

A.2 Installation Procedure

Step 1: Obtain the Source Code The source code acts as the foundation for the system. It can be acquired via Git cloning or direct download:

- **Option A (Git):** Open a terminal (Command Prompt or PowerShell) and run the following command: `git clone https://github.com/Swaph/smers.git`
- **Option B (ZIP Download):** Navigate to the GitHub repository link, click the green "Code" button, select "Download ZIP," and extract the files to a local folder.

Step 2: Database Configuration

1. Open **pgAdmin 4** or your preferred SQL tool.
2. Create a new database named `smers_db`.
3. Locate the file `database/schema.sql` provided in the downloaded project folder.
4. Execute the script within the SQL tool to create the necessary tables (`users`, `incidents`, `responders`, `payment_logs`) and seed the initial geospatial data for Laikipia.

Step 3: Backend Server Setup

1. Open a terminal inside the project folder and navigate to the backend directory: `cd backend`
2. Install the required dependencies (Express, pg, cors): `npm install`
3. Start the API server: `node index.js`
4. *Success Indicator:* The terminal will display the message: 🚀 `Server is running on http://localhost:5000`

Step 4: Frontend Application Setup

1. Open a **new** terminal window (keep the backend running) and navigate to the frontend directory: `cd frontend`
2. Install the interface dependencies (React, Material UI, Leaflet): `npm install`
3. Launch the user interface: `npm start`
4. The application will automatically launch in your default web browser at `http://localhost:3000`.

A.3 Operational Guide

1. Authentication (Login)

- Navigate to the login page presented on launch.
- **Dispatcher Access:** Enter Email: `j.doe@smers.co.ke` / Password: `password123`.
- **Admin Access:** Enter Email: `admin@smers.co.ke` / Password: `adminpass`.

2. Initiating a Simulation (Dispatch)

- On the Main Dashboard, locate the "Live Operations Feed" panel.
- Click the red button labeled **"+ Simulate 911 Call"**.
- *Observation:* A new emergency card will appear in the queue, and the system's proximity algorithm will automatically calculate and assign the nearest responder.

3. Monitoring Active Missions

- Observe the central map. A blue polyline will appear connecting a responder (Green Icon) to the patient (Red Marker).
- Watch the status update in real-time from "Dispatched" → "Transporting" (Patient Marker disappears) → "Completed."

4. Administrative Audit

- Click the **"Admin"** tab in the top navigation bar.
- Select **"Financials"** to view the M-Pesa Disbursement Log.
- Verify that the completed mission has generated a corresponding payment record with the correct mileage cost.

APPENDIX B: CODE REPOSITORY

The complete source code, including the React frontend, Node.js backend, and PostgreSQL database scripts, is publicly hosted and version-controlled on GitHub.

Repository Link: https://github.com/Swaph/smers

Note: The repository includes a `README.md` *file with detailed technical documentation, dependency lists, and troubleshooting steps.*