NAME: SWAFIYAH GICHUKI

ID. NO.: 665725

SEMESTER: FALL 2025

COURSE: SWE4900A

SOFTWARE DESIGN DOCUMENT

1. Introduction

1.1 Purpose

This document provides a detailed design specification for the Smart Medical Emergency Response System (SMERS) high-fidelity front-end prototype. Its purpose is to serve as a comprehensive blueprint outlining the system's architecture, user requirements, and user interface (UI) design. It is intended to guide the development process and act as a formal record of the prototype's features and functionality for academic review.

1.2 Scope

The scope of this project is focused on the design and implementation of an interactive and realistic **front-end simulation** of the SMERS dispatch environment. The system's boundaries encompass a multi-page application with distinct, role-based user flows, a high-fidelity user interface built with the React framework, an interactive triage form simulating AI-assisted priority assignment, and a dynamic dispatch dashboard.

Functionality explicitly **out of scope** for this prototype includes the development of a production-ready back-end, real user authentication, and integration with live, third-party services. All such features are simulated on the client side for demonstration purposes.

1.3 Overview

This document is organized into five primary sections. Following this introduction, **Section 2** details the System Requirements. **Section 3** describes the System Architecture and Design. **Section 4** presents the User Interface Design. Finally, **Section 5** discusses the prototype's current Limitations and outlines potential Future Work.

2. System Requirements

2.1 Functional Requirements

The SMERS prototype is engineered to satisfy the following functional requirements that simulate a complete dispatch workflow:

1. **R-1 (Role-Based Authentication):** The system provides a simulated login interface with distinct access for **Dispatcher** and **Administrator** roles.
2. **R-2 (Triage Form):** The system allows a dispatcher to create a new emergency report, capturing patient details, contact information, location, and a description of the medical situation.
3. **R-3 (AI-Assisted Priority):** The triage form provides a real-time "Suggested Priority" (Critical, High, Medium) by analyzing keywords in the symptoms field as the dispatcher types.
4. **R-4 (Tabbed Dashboard Interface):** The main application view is a persistent, tabbed interface for navigation between the Dashboard, Jobs, Units, and Profile views.
5. **R-5 (Real-Time Simulation):** The prototype dynamically generates new emergency jobs at regular intervals and animates the real-time movement of dispatched responder units on the map.
6. **R-6 (Job Management View):** The system displays active jobs in a table and provides a sidebar with Key Performance Indicator (KPI) cards and a mini-map showing the location of pending emergencies.
7. **R-7 (Unit Management View):** The system displays a roster of all responder units and provides a sidebar with unit-specific KPIs and a mini-map showing the real-time location of all units.

2.2 Technology Stack

The prototype is developed using a modern, component-based front-end technology stack:

1. **Front-End Framework:** React.js
2. **UI Component Library:** Material-UI (MUI) v5
3. **Routing:** React Router v6
4. **Mapping Library:** React Leaflet with OpenStreetMap tiles
5. **State Management:** React Hooks (`useState, useEffect, useRef`)

3. System Architecture and Design

3.1 Architecture Overview

The prototype is architected as a Single-Page Application (SPA), providing a fluid user experience by rendering all views within a single HTML shell. A centralized state management pattern, "lifting state up," is employed. The top-level `App.js` component acts as the single source of truth, managing all application-wide state and logic. This central controller passes data and handler functions down to child page components as props, ensuring a predictable and unidirectional data flow.

3.2 UML Diagrams

This section utilizes the Unified Modeling Language (UML), a standardized graphical language for modeling, visualizing, and documenting software systems. The following diagrams are presented to provide a comprehensive, multi-faceted view of the SMERS prototype's design. Each diagram serves a distinct purpose:

1. The Class Diagram illustrates the static structure and relationships between the primary software components and data models.
2. The Object Diagram provides a concrete snapshot of the system's state during a specific runtime scenario.
3. The Sequence Diagram details the dynamic, time-ordered interactions between components during a key workflow.
4. The Entity-Relationship Diagram (ERD) models the logical schema for the future database.
5. The State Diagram describes the complete lifecycle of a core system entity.

3.2.1 Class Diagram

The Class Diagram provides a static, high-level blueprint of the application's logical structure. It defines the primary "classes" representing the main data models and React components along with their essential attributes and methods. As shown in Figure 1, the diagram illustrates the key relationships between these components, such as the composition relationship where the main `App` component manages collections of `Request` and `Responder` objects. It also shows the dependency relationships, demonstrating how view components like `JobsView` depend on the data models to render information.

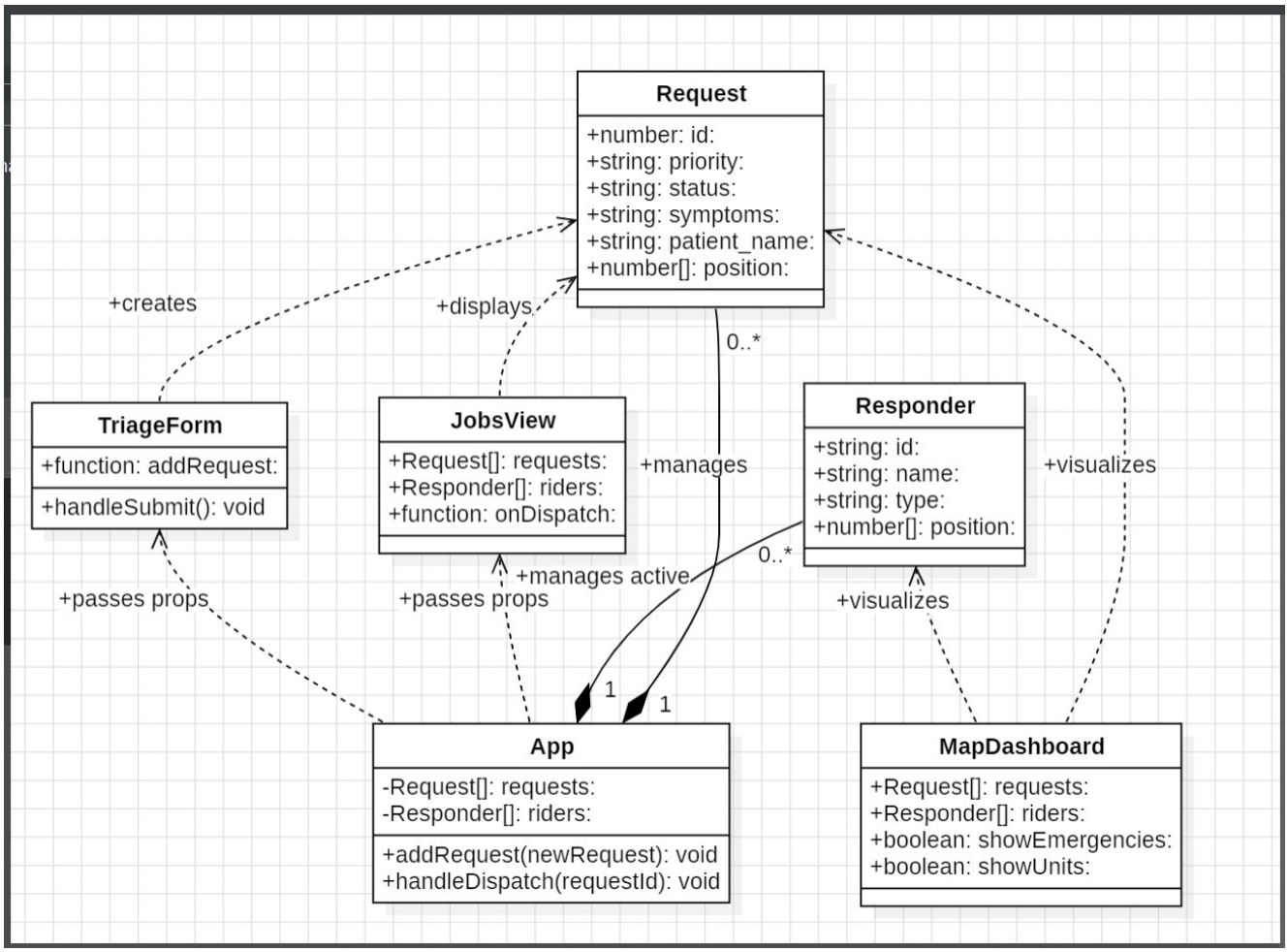

Figure 1: SMERS Class Diagram

3.2.2 Object Diagram

The Object Diagram presents a snapshot of the system's state at a single moment in time. Unlike the abstract Class Diagram, it models concrete instances of the classes. The scenario depicted in Figure 2 illustrates the `smersApp` instance managing two active emergencies. One instance, `request101`, is in a "Dispatched" state and is linked to an active responder, `boda01`. Simultaneously, `request102` remains "Pending." This diagram is valuable for understanding the system's state structure during a live operational scenario.

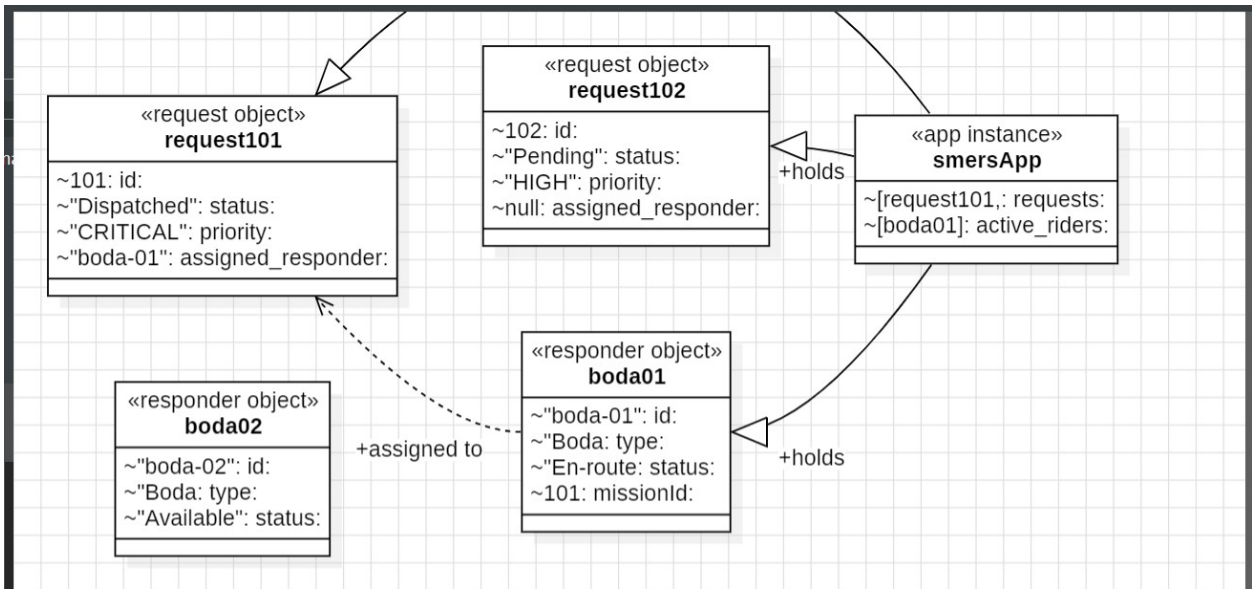

Figure 2: SMERS Object Diagram for an Active Dispatch

3.2.3 Sequence Diagram

The Sequence Diagram visualizes the interactions between different components over time, specifically during the "Dispatch Emergency" workflow. As illustrated in Figure 3, time flows downward, and the horizontal arrows depict messages and function calls between participants. This diagram effectively demonstrates React's unidirectional data flow: a user action (the `click`) triggers a function call that updates the central state in `App.js`. This state update then flows back down as new props, causing the UI components to re-render with the updated information.

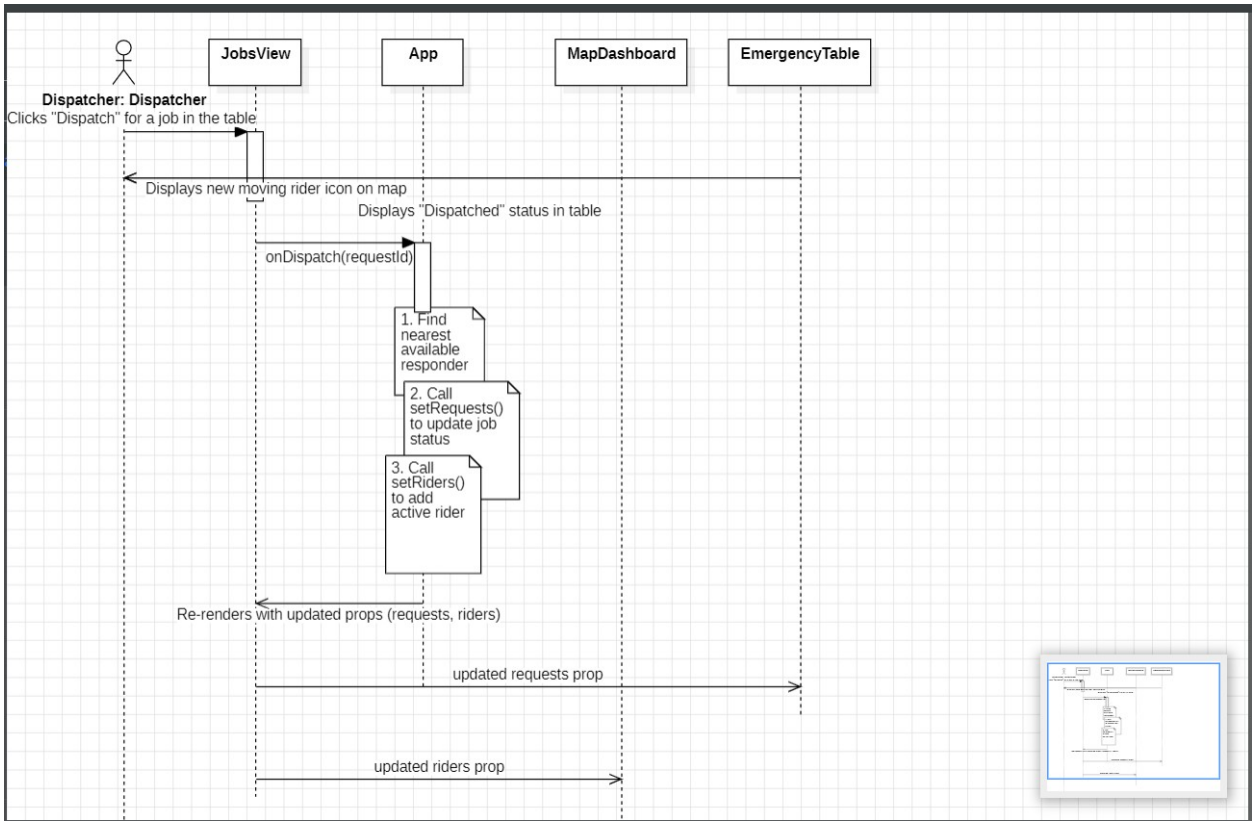

Figure 3: SMERS Sequence Diagram for Dispatch Workflow

3.2.4 Entity-Relationship Diagram (ERD)

The Entity-Relationship Diagram (ERD) models the logical data structure for a future production-level database. It defines the core entities (`DISPATCHER`, `EMERGENCY_REQUEST`, `RESPONDER_UNIT`) and their corresponding attributes. Figure 4 identifies the Primary Keys (PK) that uniquely identify each record and the Foreign Keys (FK) that establish links between the entities. The relationships and their cardinality are also defined, illustrating business rules such as "one `DISPATCHER` manages zero-or-many `EMERGENCY_REQUESTS`."

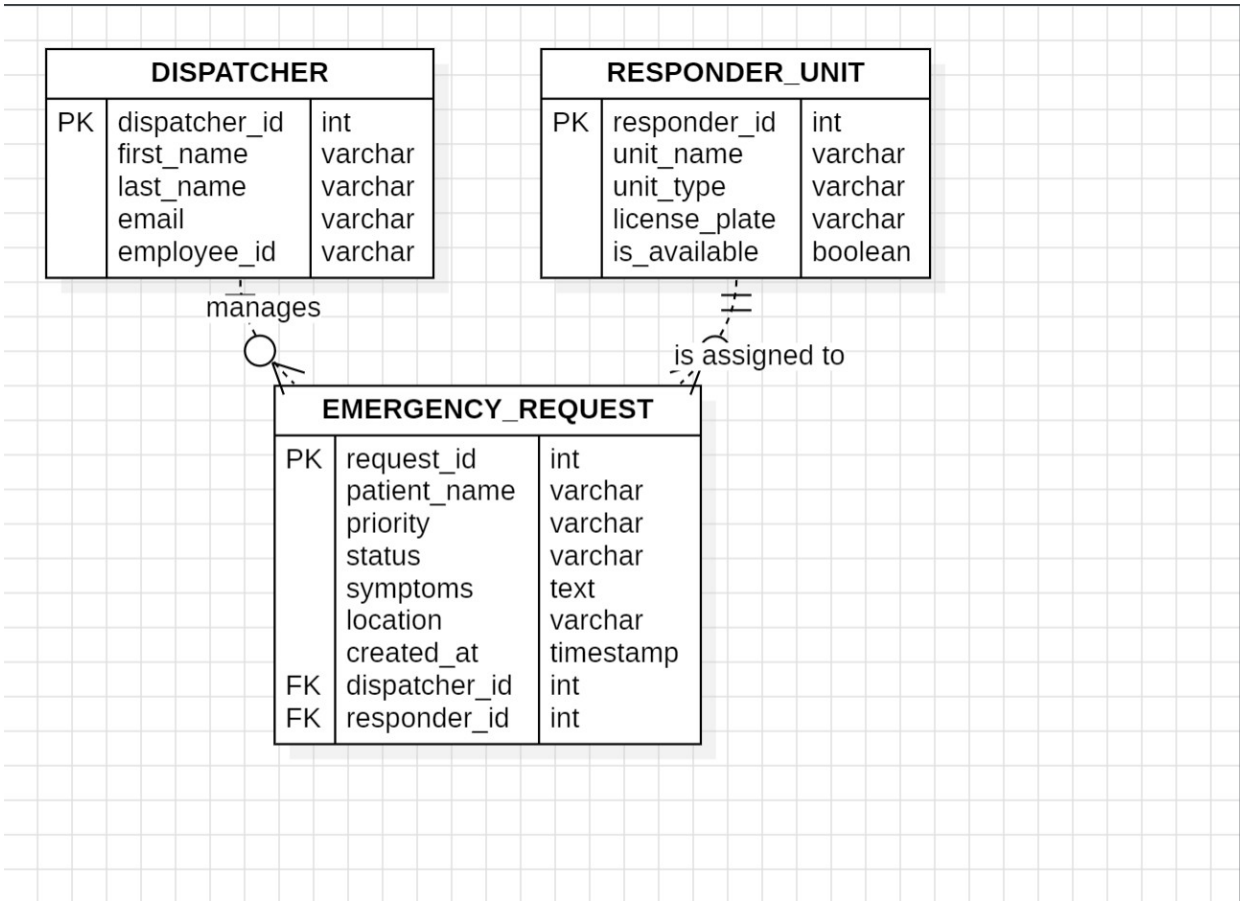

Figure 4: SMERS Entity-Relationship Diagram

3.2.5 State Diagram

The State Diagram describes the complete lifecycle of a single `Emergency_Request` object within the system. Figure 5 models the various states a request can transition through, from its creation (`Pending`) to its conclusion (`Completed` or `Cancelled`). The arrows represent the transitions, which are the events or actions that cause the request to move from one state to another (e.g., the event "Dispatcher assigns unit" transitions the request from `Pending` to `Dispatched`). This provides a clear and concise view of the business logic governing an emergency case's status.

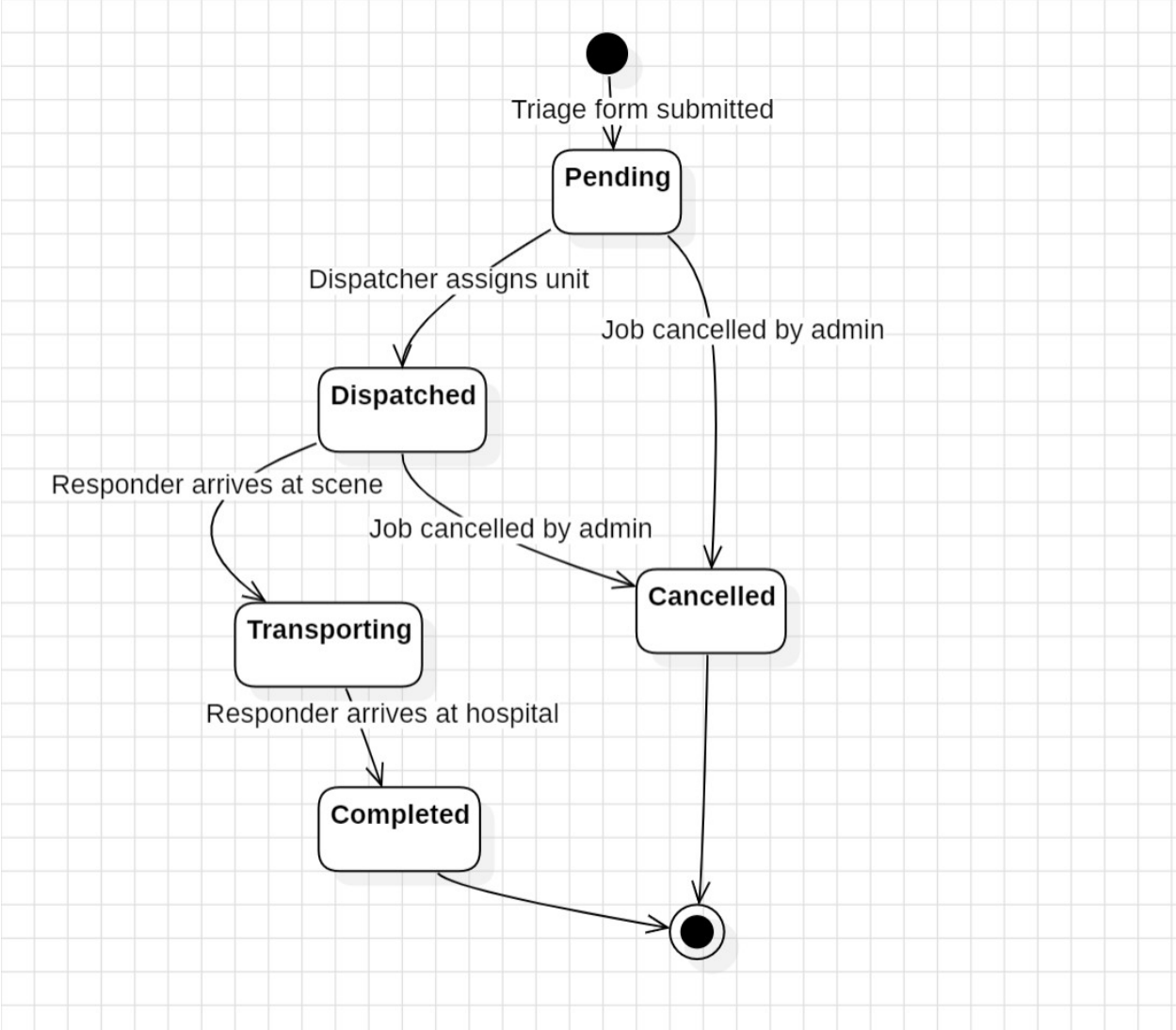

Figure 5: SMERS State Diagram for an Emergency Request

4. User Interface (UI) Design

The user interface is designed to be modern, intuitive, and data-rich, adhering to the principles of the Material Design system to ensure a professional and usable experience for dispatchers under pressure.

4.1 Login Page

The application's entry point is a responsive, two-column layout. The left panel contains the authentication form for dispatcher and admin roles, while the right panel displays a thematic medical illustration, creating a professional and welcoming first impression.

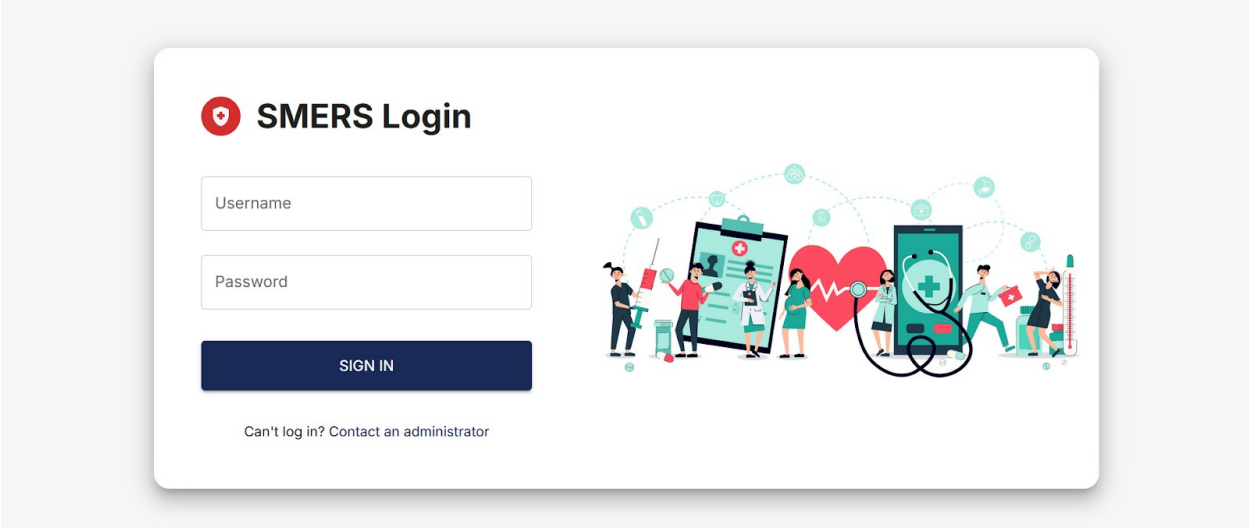

Figure 4.1: The SMERS Login Page User Interface

4.2 Main Application Layout

Upon authentication, the user is presented with a persistent header containing the SMERS logo, primary navigation tabs (Dashboard, Jobs, Units), and a clickable user avatar for accessing the profile. A floating action button provides global access to the triage form and utilizes a pulsing animation to indicate a simulated incoming call, ensuring the dispatcher is always aware of new events.

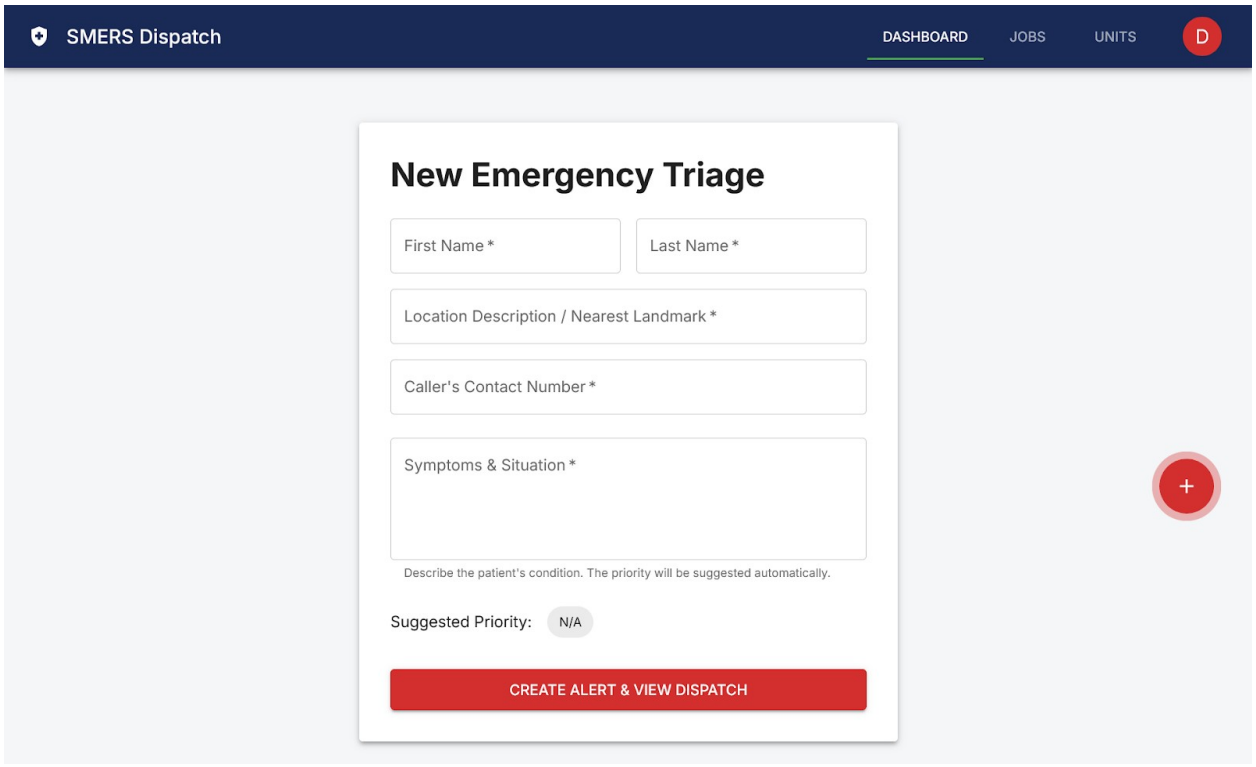

Figure 4.2: The SMERS Triage Form User Interface

4.3 Jobs Tab View

This is the primary operational view for dispatchers. It features a two-column layout where the main content area is dedicated to the "Active Jobs" table, showing all ongoing emergencies. A sidebar on the right displays high-level Key Performance Indicators (KPIs) such as 'Total Emergencies' and 'On-Going' cases, along with a mini-map focused on visualizing the geographic locations of all pending incidents

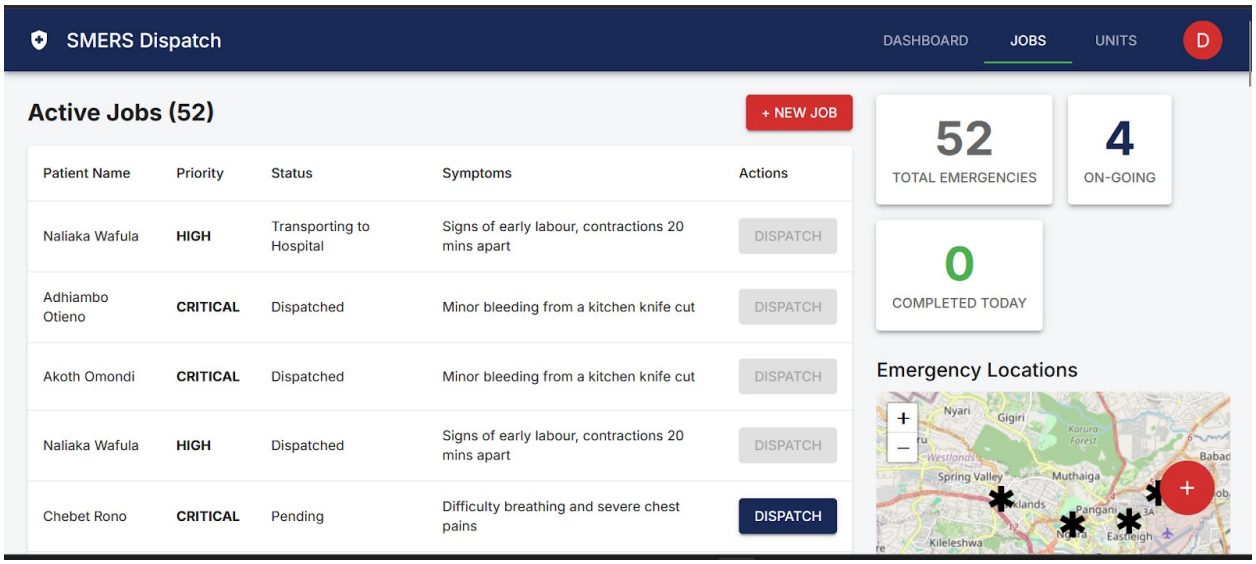

Figure 4.3: The SMERS Jobs Tabs View User Interface

4.4 Units Dispatch View

This view provides a complete operational picture of all responder assets. The main content area features a "Unit Roster" table, listing all responders, their vehicle type (Boda Boda or Tuk Tuk), and their real-time status (e.g., 'Available', 'En-route to Patient'). The sidebar contains unit-specific KPIs ('Total Units', 'Available', 'Engaged') and a dedicated mini-map that visualizes the current geographic location of all units, both idle and active.

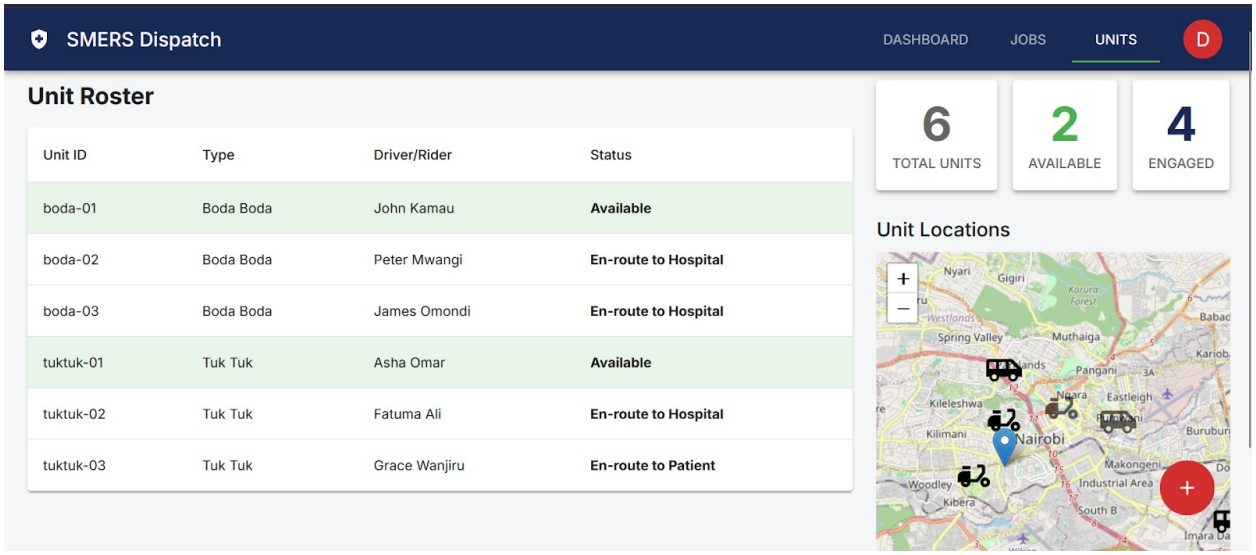

Figure 4.4: The SMERS Units Tab View User Interface

4.5 Profile View

This page uses a modern, card-based layout for viewing profile details and application settings, providing a clean and organized user experience for the authenticated dispatcher.

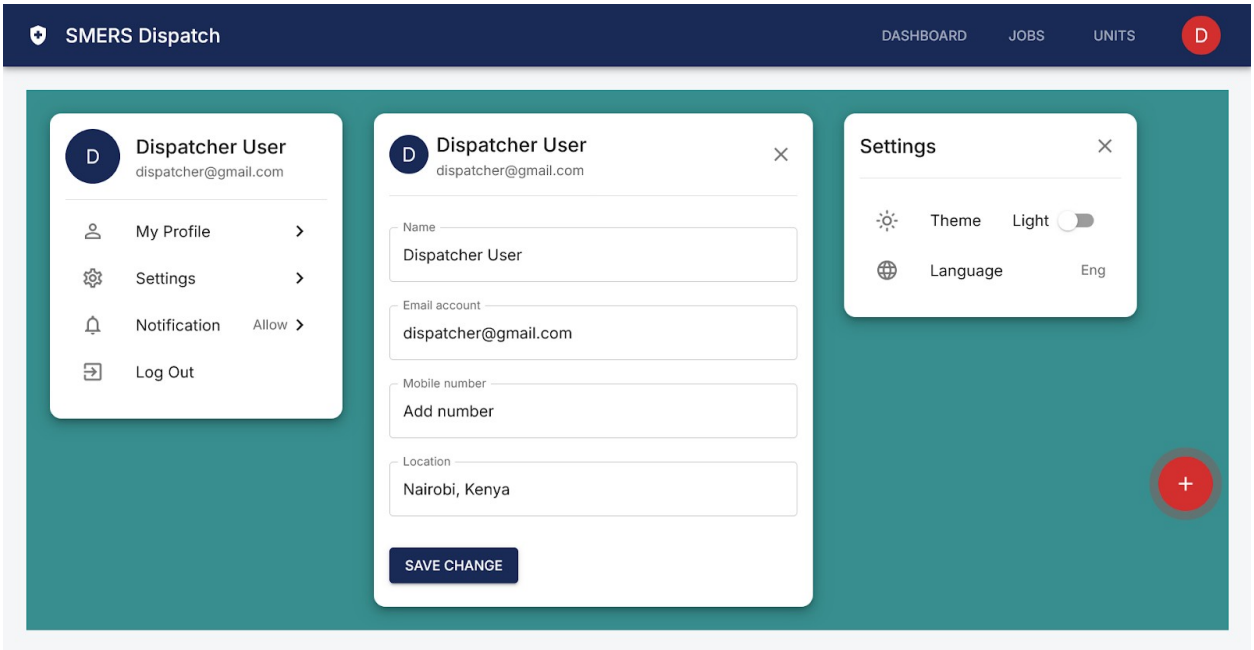

Figure 4.5: The SMERS Profile View User Interface

4.6. Admin Dashboard View

The Administrator Dashboard provides a high-level oversight of the entire SMERS system, focusing on operational analytics and management. The interface is organized into a two-column layout. The main content area features a tabbed interface for different administrative views, while the sidebar is dedicated to displaying system-wide analytics through Key Performance Indicator (KPI) cards. Each tab is detailed below.

4.6.1. Open Cases Tab

This tab provides the administrator with a real-time view of all active, ongoing emergencies that have not yet been completed. It utilizes the main `EmergencyTable` component, displaying critical information such as the patient's name, priority, and current status (e.g., 'Pending', 'Dispatched', 'Transporting to Hospital'). This allows for immediate oversight of all active operations.

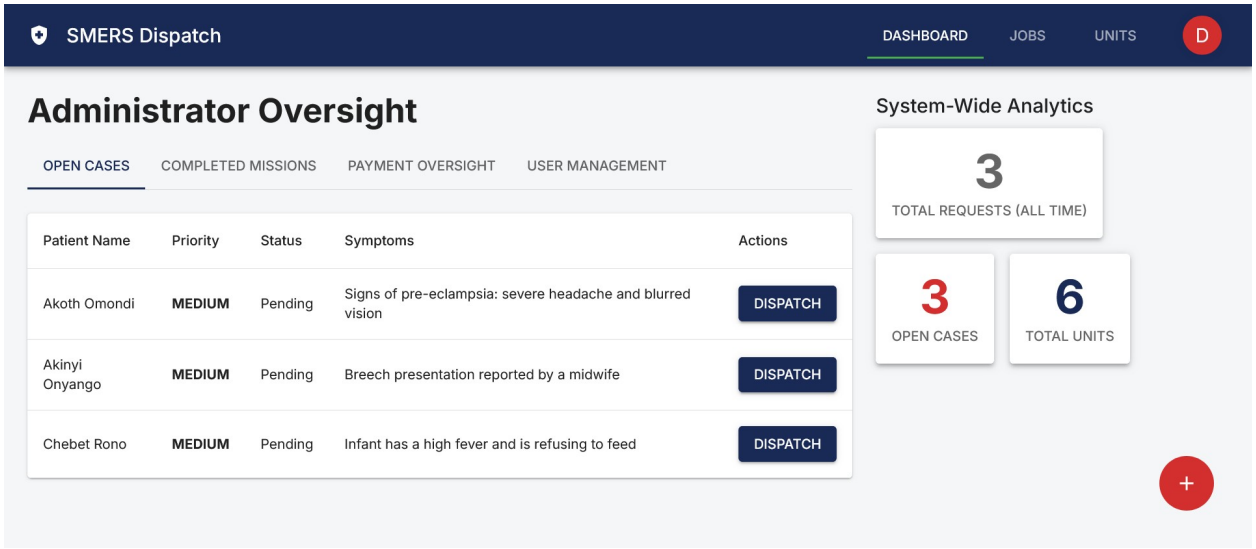

Figure 4.6: Admin Dashboard - Open Cases Tab

4.6.2. Completed Missions Tab

This tab serves as a historical log of all successfully completed missions. It displays the `CompletedMissionsTable`, providing a clear, archived record of past emergencies, including patient details, final status, and the dispatched responder. This is essential for record-keeping and performance review.

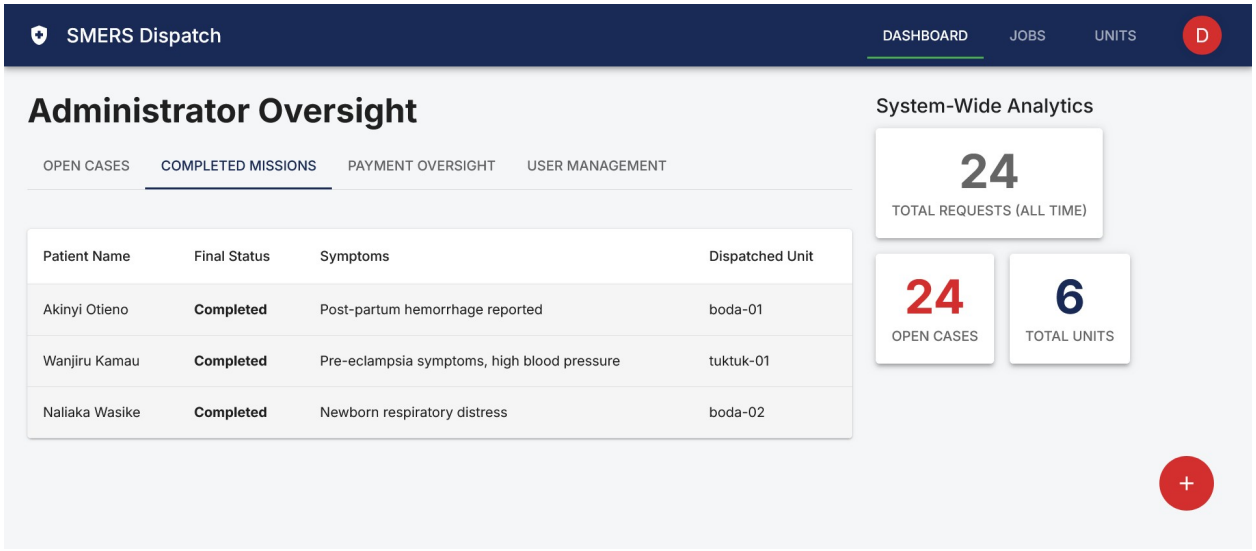

Figure 4.7: Admin Dashboard - Completed Missions Tab

4.6.3. Payment Oversight Tab

This view is designed for managing payments owed to responders. It features its own set of KPI cards for financial metrics (e.g., 'Total Pending Payout') and a detailed table listing each responder with completed missions. The table shows the number of missions, the total amount owed, and the current payment status. An interactive button allows the administrator to mark payments as "Paid."

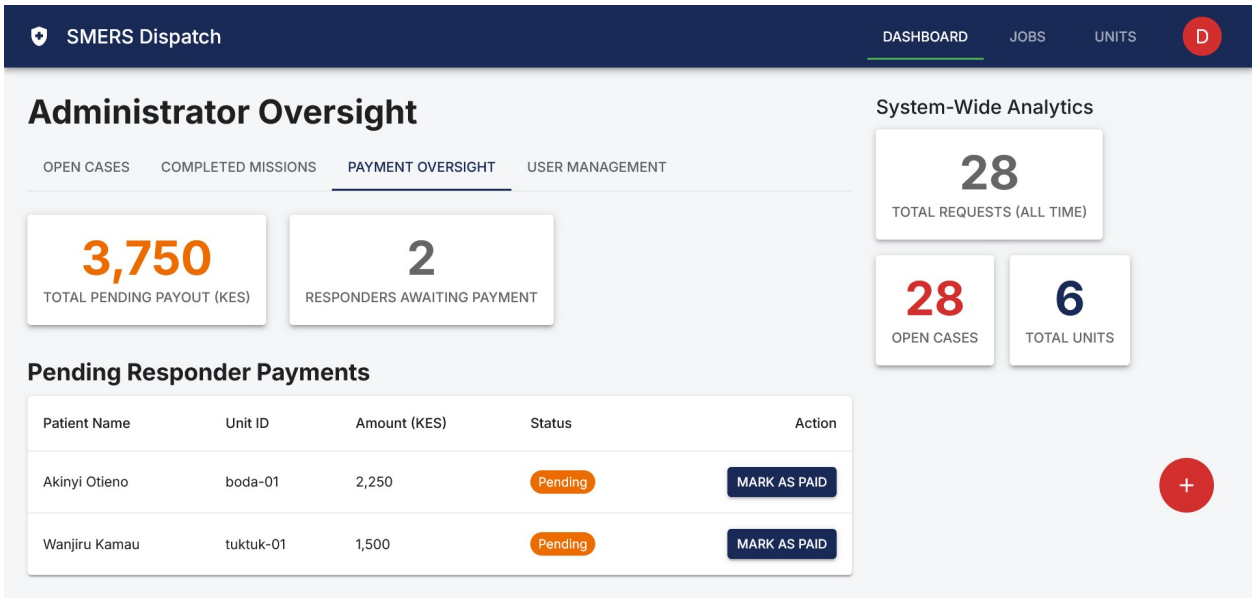

Figure 4.8: Admin Dashboard - Payment Oversight Tab

4.6.4. User Management Tab

This section provides an interface for managing system users. It features a two-part layout with a sidebar for actions (e.g., 'Manage Users', 'Add Users') and a main content area displaying a roster of all registered users. The table includes the user's name, title, and a color-coded chip indicating if their account is currently 'Active' or 'Inactive'.

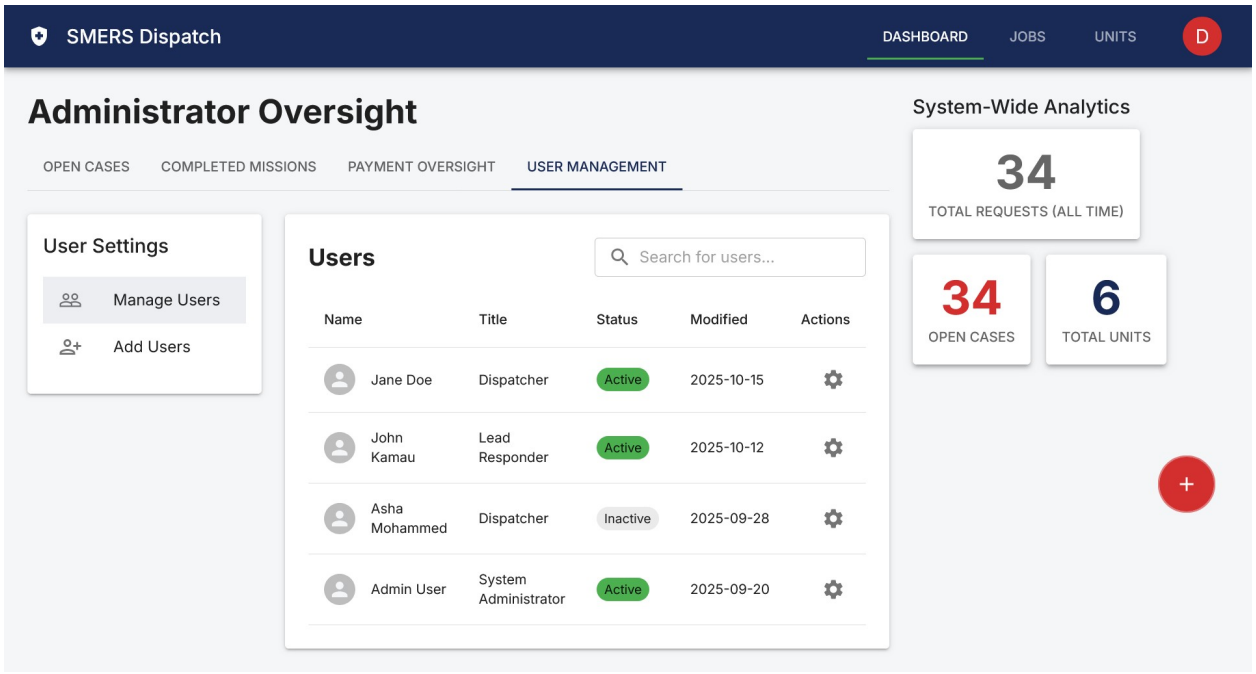

Figure 4.9: Admin Dashboard - User Management Tab

By selecting **'Add Users'** from the sidebar, an administrator is presented with the user creation form (Figure 4.10). This interface is used for the secure provisioning of new system accounts. The form is designed to capture essential employee details such as their name, role within the organization (e.g., Dispatcher, Responder), and work email.

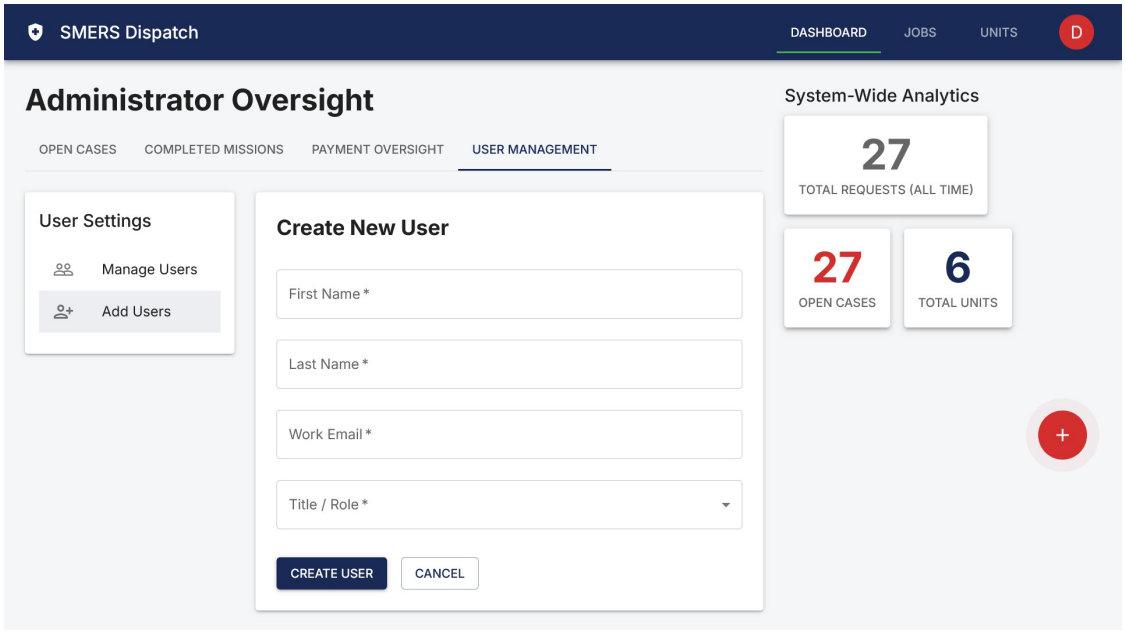

Figure 4.10: Admin Dashboard - Add User Form

5. Limitations and Future Work

5.1 Current Limitations

This prototype, while functionally rich, has several inherent limitations due to its focus on front-end simulation.

1. **Mock Data:** The system relies entirely on mock data generated and managed on the client side, with no persistent back-end or database.
2. **Simulated Movement:** The movement of responder units is a linear interpolation between two points and does not follow real-world road networks.
3. **Mock Authentication:** The authentication mechanism is mocked with hard-coded credentials and does not involve industry-standard security protocols.

5.2 Future Work

The successful implementation of this prototype lays the groundwork for the remaining areas of development required to create a production-ready system.

1. **Back-End Development:** Develop a secure back-end API using a framework such as Node.js with Express.
2. **Database Integration:** Integrate a robust relational database, such as PostgreSQL, for the persistent storage of all system data.
3. **Telephony Integration:** Integrate with a telephony service like Twilio to handle incoming toll-free calls.
4. **Advanced Routing:** Implement a road-based routing engine (e.g., OSRM) for accurate real-time tracking and estimated times of arrival.