# HealthMap: Real-Time Tracking Solution for Hospital Logistics

## 1. Introduction

**HealthMap** is a real-time tracking system designed to optimize hospital logistics by providing real-time visibility into the movement and location of critical hospital assets and staff. This system is built as a web application using React for the frontend and integrates cloud-based services and tracking devices.

***Purpose:***

This documentation serves as a guide for the development, deployment, and maintenance of the HealthMap web application.

## 2. System Architecture

**2.1 Overview** HealthMap consists of several interconnected components designed to offer real-time asset tracking within a hospital environment:

- **Frontend (Web Interface)**
- **Backend (API Server)**
- **Database**
- **Real-Time Tracking Layer**
- **Cloud Infrastructure**

**2.2 Components Breakdown**

- **Frontend**: Built using **React** (for web)
- **Backend**: **Node.js** and **Express.js** (for API management)
- **Database**: **MongoDB**
- **Real-Time Tracking**: **GPS** tracking devices
- **Cloud Infrastructure**: **AWS EC2**, **AWS Lambda**, **AWS S3**
- **Data Processing Layer**: **Apache Kafka** for real-time stream processing
- **Security**: **OAuth2** for authentication, **AES** for data encryption, **SSL/TLS** for secure communication

## 3. Technologies Used

### 3.1 Frontend:

- **React**: For building dynamic and interactive user interfaces.
- **Redux**: To manage global state across the application.
- **React Router**: For handling page navigation within the application.
- **Axios**: To make HTTP requests to the backend API.
- **CSS/SCSS**: For styling the web pages and ensuring a responsive UI.

### 3.2 Backend:

- **Node.js**: JavaScript runtime used to build scalable server-side applications.
- **Express.js**: A fast and minimalist web framework for Node.js to build RESTful APIs.
- **JWT/OAuth2**: For secure user authentication and token management.

### 3.3 Real-Time Tracking:

- **MQTT/WebSockets**: For continuous communication between devices and the backend for real-time data updates.
- **GPS/RFID/IoT Devices**: To track the location of hospital assets and staff.

### 3.4 Database:

- **PostgreSQL**: A relational database for storing structured data such as asset information, user details, and tracking logs.
- **MongoDB**: Used for storing real-time data (such as GPS location data) in a flexible format.

### 3.5 Cloud Infrastructure:

- **AWS EC2**: For hosting the backend server.
- **AWS S3**: For storing static files and logs.
- **AWS Lambda**: For serverless functions that handle specific tasks like alerts and notifications.

### 3.6 Data Processing:

- **Apache Kafka**: For handling real-time data streams and ensuring that large volumes of tracking data can be processed efficiently.

### 3.7 Security:

- **OAuth2/JWT**: For handling secure authentication and user sessions.
- **SSL/TLS**: Ensures secure data transmission between the client and server.
- **AES Encryption**: Used for encrypting sensitive data at rest and during transmission.

## 4. Functional Specifications

### 4.1 User Roles

- **Admin**: Full access to all system data and features, including user management and asset control.
- **Manager**: Access to asset tracking, location history, and reporting.
- **Staff**: Restricted access based on roles (e.g., tracking specific assets, viewing personal information).

### 4.2 Features

- **Real-Time Asset Tracking**: Visualizes the live location of hospital assets.
- **Asset Management**: Add, update, and manage hospital assets.
- **Alerts & Notifications**: Notifies users about critical events, such as asset misplacements or unauthorized movement.
- **Analytics & Reporting**: Allows users to generate detailed reports on asset usage and hospital logistics.

### 4.3 Non-Functional Requirements

- **Scalability**: The system should be scalable to accommodate increasing data and hospital assets.
- **Reliability**: High availability (99.99% uptime) for real-time tracking.
- **Performance**: Low-latency data processing to deliver real-time updates to users.

## 5. Database Schema

### 5.1 Tables

- **Users**: Stores user data (id, name, email, password, role).
- **Assets**: Stores information about hospital assets (id, type, location, status, last updated).

- **Location_Logs**: Stores real-time location data for assets (id, asset_id, timestamp, location).
- **Tracking_Devices**: Stores data about tracking devices associated with assets (id, device_type, status, asset_id).
- **Events**: Stores event data like asset misplacements (event_id, asset_id, event_type, timestamp).

### 5.2 Relationships

- Users can be associated with assets (e.g., staff assigned to specific equipment).
- Assets have associated location logs, updated continuously based on real-time tracking.
- Tracking devices are linked to specific assets to track their location.

## 6. API Documentation

### 6.1 Authentication API

- **POST /auth/login**: Logs in a user and returns an access token.
  - **Request**: `{ "email": "user@example.com", "password": "password123" }`
  - **Response**: `{ "token": "jwt_token_here" }`
- **POST /auth/logout**: Logs out the user and invalidates the token.

### 6.2 Asset Management API

- **GET /assets**: Retrieves all assets in the hospital.
  - **Response**: `[ { "id": 1, "type": "Wheelchair", "location": "Room 101", "status": "Available" } ]`
- **POST /assets**: Adds a new asset to the system.
  - **Request**: `{ "type": "Defibrillator", "description": "Portable", "location": "Emergency Room" }`
  - **Response**: `{ "id": 2, "type": "Defibrillator", "location": "Emergency Room" }`

### 6.3 Real-Time Tracking API

- **GET /tracking/{asset_id}**: Returns the real-time location of an asset.
  - **Response**: `{ "location": "Room 105", "timestamp": "2024-11-08T10:00:00" }`

### 6.4 Event API

- **GET /events**: Retrieves events like asset misplacements.
    - **Response**: `[ { "event_id": 1, "asset_id": 3, "event_type": "Misplaced", "timestamp": "2024-11-08T10:15:00" } ]`

## 7. Deployment

### 7.1 CI/CD Pipeline

- **GitHub Actions** or **Jenkins** for automating tests, builds, and deployment.
- **Docker**: For containerizing the backend and frontend services.
- **Kubernetes**: For orchestrating the containers and ensuring scalability.

### 7.2 Hosting & Infrastructure

- The backend is hosted on **AWS EC2**.
- **AWS S3** is used for storing static files, images, and logs.
- **AWS Lambda** handles serverless functions for alerts and notifications.

## 8. Security

### 8.1 Authentication

- **OAuth 2.0 / JWT**: Secure user authentication with token management.

### 8.2 Data Security

- **SSL/TLS**: For encrypted communication.
- **AES encryption** for sensitive data at rest and during transmission.

### 8.3 Access Control

- Role-based access to ensure proper user permissions and data security.

## 9. Future Enhancements

- **Machine Learning**: Integrating predictive models for asset management and route optimization.
- **Integration with other Hospital Systems**: For seamless operations.
- **Expanded Device Support**: Integration with more IoT-based devices for better asset tracking.