EXPERIMENT 2

AIM:
Experiment based on React Hooks (useEffect, useContext, custom hooks)

THEORY:

React hooks have transformed the way React developers build applications, replacing class-based lifecycle methods and providing cleaner abstractions. To anchor the discussion, a real-time ambulance tracking application is used as an example context where these hooks find practical application. The experiment evaluates their role in managing side effects, sharing global state, and encapsulating reusable logic in complex, real-time environments.

1) Introduction

React Hooks introduced in React 16.8 revolutionized state and lifecycle management in function components. They allow developers to manage component state, perform side effects, and reuse logic without classes. This shift simplified codebases, promoted functional programming practices, and increased code reuse.

The real-time ambulance tracker project serves as a reference example. Such an application continuously receives GPS data from ambulances, updates a map interface, and displays live statistics. Building this requires careful handling of asynchronous events, global state management, and reusable logic — all areas where Hooks are critical.

2) Theory & Background

2.1 React Hooks — A Paradigm Shift

Traditionally, React used class components with methods like componentDidMount, componentDidUpdate, and componentWillUnmount to handle lifecycle events. Hooks abstract this into cleaner, more modular functions:

- useState for local state
- useEffect for side effects
- useContext for global state
- Custom hooks for encapsulated, reusable logic

This approach eliminates the need for classes and improves readability and maintainability.

2.2 The useEffect Hook

useEffect is used for handling side effects — operations that affect the outside world or depend on it. Examples include fetching data, subscribing to a WebSocket, or interacting with the DOM.

Key concepts:

- Execution timing: Runs after the component renders, ensuring the UI is updated before effects are executed.
- Dependency array: Determines when the effect runs. If dependencies change, the effect re-runs.
- Cleanup functions: Ensure that resources like subscriptions, intervals, or sockets are disposed properly.

Pitfalls and considerations:

- Stale closures: Forgetting to include all dependencies can cause outdated variables to be used.
- Over-subscription: Missing cleanups can lead to memory leaks.
- Strict Mode behavior: In development, React deliberately mounts/unmounts twice to detect unsafe effects.

Example (simplified for ambulance tracker):

```
useEffect(() => {
  const ws = new WebSocket("wss://ambulance-tracker.example");
  ws.onmessage = (msg) => console.log("Received update", msg.data);
  return () => ws.close();
}, []);
```

This shows how useEffect manages a WebSocket connection and cleans it up when the component unmounts.

2.3 The useContext Hook

useContext allows components to access values from a context provider without prop drilling. This is particularly useful for global state like themes, authentication, or connection state.

In a real-time tracker:

- Connection status (connected/disconnected)
- Shared ambulance data store
- User preferences (map type, filters)

Theory notes:

- Context updates trigger re-renders of all consumers; optimization often requires memoization.
- Multiple contexts can be combined to separate concerns.

Example:

```
const ConnectionContext = React.createContext();
const status = useContext(ConnectionContext);
```

Here, status might reflect whether the ambulance data stream is live or disconnected.

2.4 Custom Hooks

Custom hooks allow developers to bundle logic into reusable functions that use other hooks. They are powerful for abstraction and modularity.

Types of custom hooks:

- Data hooks: encapsulate data fetching (e.g., useAmbulanceData).
- Utility hooks: wrap browser APIs (e.g., useGeolocation).
- Policy hooks: manage behaviors like retries, debouncing, or throttling.

Benefits:

- Cleaner components by moving logic out.
- Reusability across different parts of an application.
- Easier testing and maintenance.

Example:

```
function useGeolocation() {
  const [pos, setPos] = useState(null);
  useEffect(() => {
    const watch = navigator.geolocation.watchPosition(setPos);
    return () => navigator.geolocation.clearWatch(watch);
  }, []);
  return pos;
}
```

This hook can be reused anywhere in the application where the operator's location is needed.

2.5 Hooks in Real-Time Applications

Hooks are particularly well suited for real-time applications:

- useEffect handles subscriptions (WebSocket, SSE, polling).
- useContext shares live state across components (e.g., current ambulance list).
- Custom hooks encapsulate recurring patterns like geolocation or retry logic.

In a real-time ambulance tracker, these hooks together provide a structured way to manage continuous data flow, avoid duplication, and keep the UI responsive.

CODE

Alerts.tsx

```tsx
import { useState, useEffect } from "react";
import { Bell, CheckCircle, AlertTriangle, Info, X, Battery, Wifi } from "lucide-react";
import { Button } from "@/components/ui/button";
import { Badge } from "@/components/ui/badge";
import { motion, AnimatePresence } from "framer-motion";
import { useQuery, useMutation, useQueryClient } from "@tanstack/react-query";
import { apiRequest } from "@/lib/queryClient";
import type { Alert } from "@/types";

export function Alerts() {
  const queryClient = useQueryClient();
  const [dismissingId, setDismissingId] = useState<string | null>(null);

  const { data: alerts, isLoading } = useQuery<Alert[]>({
    queryKey: ["/api/alerts"],
    refetchInterval: 5000, // Refetch every 5 seconds for real-time alerts
  });

  const markAsReadMutation = useMutation({
    mutationFn: async (alertId: string) => {
      const response = await apiRequest("PATCH", `/api/alerts/${alertId}/read`);
      return response.json();
    },
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ["/api/alerts"] });
    },
  });

  // Listen for custom events from other components
  useEffect(() => {
    const handleBatteryLow = (event: CustomEvent) => {
      // Battery low events are already handled by the backend alerts system
      console.log("Battery low event:", event.detail);
    };

    const handleVoiceSOS = (event: CustomEvent) => {
      console.log("Voice SOS event:", event.detail);
    };
```

Code 1.1

```
    const handleEmergencySOS = (event: CustomEvent) => {
      console.log("Emergency SOS event:", event.detail);
    };

    window.addEventListener("batteryLow", handleBatteryLow as EventListener);
    window.addEventListener("voiceSOS", handleVoiceSOS as EventListener);
    window.addEventListener("emergencySOS", handleEmergencySOS as EventListener);

    return () => {
      window.removeEventListener("batteryLow", handleBatteryLow as EventListener);
      window.removeEventListener("voiceSOS", handleVoiceSOS as EventListener);
      window.removeEventListener("emergencySOS", handleEmergencySOS as EventListener);
    };
  }, []);

  const handleDismissAlert = async (alertId: string) => {
    setDismissingId(alertId);
    try {
      await markAsReadMutation.mutateAsync(alertId);
    } catch (error) {
      console.error("Failed to dismiss alert:", error);
    } finally {
      setDismissingId(null);
    }
  };

  const getAlertIcon = (type: string) => {
    switch (type) {
      case "arrival":
        return CheckCircle;
      case "battery":
        return Battery;
      case "system":
        return Info;
      case "offline":
        return Wifi;
```

Code 1.2

```
      default:
        return AlertTriangle;
    }
  };

  const getAlertStyles = (severity: string) => {
    switch (severity) {
      case "success":
        return {
          bg: "bg-green-50 dark:bg-green-900/20",
          border: "border-green-200 dark:border-green-800",
          icon: "text-green-500",
          title: "text-green-800 dark:text-green-300",
          description: "text-green-700 dark:text-green-400",
          button: "text-green-600 hover:text-green-800 dark:text-green-400 dark:hover:text-green-200"
        };
      case "warning":
        return {
          bg: "bg-orange-50 dark:bg-orange-900/20",
          border: "border-orange-200 dark:border-orange-800",
          icon: "text-orange-500",
          title: "text-orange-800 dark:text-orange-300",
          description: "text-orange-700 dark:text-orange-400",
          button: "text-orange-600 hover:text-orange-800 dark:text-orange-400 dark:hover:text-orange-200"
        };
      case "error":
        return {
          bg: "bg-red-50 dark:bg-red-900/20",
          border: "border-red-200 dark:border-red-800",
          icon: "text-red-500",
          title: "text-red-800 dark:text-red-300",
          description: "text-red-700 dark:text-red-400",
          button: "text-red-600 hover:text-red-800 dark:text-red-400 dark:hover:text-red-200"
        };
      default:
        return {
```

Code 1.3

```
            bg: "bg-blue-50 dark:bg-blue-900/20",
            border: "border-blue-200 dark:border-blue-800",
            icon: "text-blue-500",
            title: "text-blue-800 dark:text-blue-300",
            description: "text-blue-700 dark:text-blue-400",
            button: "text-blue-600 hover:text-blue-800 dark:text-blue-400 dark:hover:text-blue-200"
          };
    }
  };

  const formatTimeAgo = (date: Date) => {
    const now = new Date();
    const diffInMinutes = Math.floor((now.getTime() - date.getTime()) / (1000 * 60));

    if (diffInMinutes < 1) return "Just now";
    if (diffInMinutes < 60) return `${diffInMinutes} minutes ago`;

    const diffInHours = Math.floor(diffInMinutes / 60);
    if (diffInHours < 24) return `${diffInHours} hours ago`;

    const diffInDays = Math.floor(diffInHours / 24);
    return `${diffInDays} days ago`;
  };

  if (isLoading) {
    return (
      <div className="bg-white dark:bg-gray-800 rounded-xl shadow-sm border border-gray-200 dark:border-gray-700">
        <div className="p-6">
          <div className="animate-pulse space-y-4">
            <div className="h-6 bg-gray-200 dark:bg-gray-700 rounded w-1/3" />
            {[...Array(3)].map((_, i) => (
              <div key={i} className="p-3 bg-gray-100 dark:bg-gray-700 rounded-lg">
                <div className="h-4 bg-gray-200 dark:bg-gray-600 rounded w-3/4 mb-2" />
                <div className="h-3 bg-gray-200 dark:bg-gray-600 rounded w-1/2" />
              </div>
            ))}
```

Code 1.4

```
  const unreadAlerts = alerts?.filter(alert => !alert.isRead) || [];

  return (
    <div className="bg-white dark:bg-gray-800 rounded-xl shadow-sm border border-gray-200 dark:border-gray-700">
      <div className="p-6">
        <h3 className="text-lg font-semibold text-gray-900 dark:text-white mb-4 flex items-center">
          <Bell className="w-5 h-5 text-medical-500 mr-2" />
          Alerts
          {unreadAlerts.length > 0 && (
            <Badge className="ml-2 bg-red-500 text-white">
              {unreadAlerts.length}
            </Badge>
          )}
        </h3>

        <div className="space-y-3 max-h-96 overflow-y-auto">
          <AnimatePresence mode="popLayout">
            {unreadAlerts.length === 0 ? (
              <motion.div
                initial={{ opacity: 0 }}
                animate={{ opacity: 1 }}
                className="text-center py-8 text-gray-500 dark:text-gray-400"
              >
                <Bell className="w-8 h-8 mx-auto mb-2 opacity-50" />
                <p className="text-sm">No new alerts</p>
                <p className="text-xs mt-1">You'll be notified of any important updates</p>
              </motion.div>
            ) : (
              unreadAlerts.map((alert, index) => {
                const Icon = getAlertIcon(alert.type);
                const styles = getAlertStyles(alert.severity);

                return (
                  <motion.div
                    key={alert.id}
                    initial={{ opacity: 0, y: 20, scale: 0.95 }}
                    animate={{ opacity: 1, y: 0, scale: 1 }}
```

Code 1.5

```
                    exit={{ opacity: 0, x: -20, scale: 0.95 }}
                    transition={{
                      delay: index * 0.1,
                      duration: 0.3,
                      type: "spring",
                      stiffness: 200,
                      damping: 20
                    }}
                    className={`flex items-start space-x-3 p-3 rounded-lg border ${styles.bg} ${styles.border}`}
                  >
                    <Icon className={`w-5 h-5 mt-1 flex-shrink-0 ${styles.icon}`} />

                    <div className="flex-1 min-w-0">
                      <div className={`font-medium ${styles.title}`}>
                        {alert.message}
                      </div>
                      {alert.description && (
                        <div className={`text-sm mt-1 ${styles.description}`}>
                          {alert.description}
                        </div>
                      )}
                      <div className="text-xs mt-2 opacity-75">
                        {alert.createdAt ? formatTimeAgo(new Date(alert.createdAt)) : 'Unknown time'}
                      </div>
                    </div>

                    <Button
                      variant="ghost"
                      size="sm"
                      onClick={() => handleDismissAlert(alert.id)}
                      disabled={dismissingId === alert.id}
                      className={`p-1 h-auto hover:bg-white/50 dark:hover:bg-gray-800/50 ${styles.button}`}
                    >
                      {dismissingId === alert.id ? (
                        <div className="w-4 h-4 border-2 border-current border-t-transparent rounded-full animate-spin" />
                      ) : (
```

const handleDismissAlert: (alertId: string) => Promise<void>

Code 1.6

```
                        <X className="w-4 h-4" />
                      )}
                    </Button>
                  </motion.div>
                );
              })
            )}
          </AnimatePresence>
        </div>

        {/* Alert History Toggle */}
        {alerts && alerts.length > unreadAlerts.length && (
          <div className="mt-4 pt-4 border-t border-gray-200 dark:border-gray-700">
            <Button
              variant="ghost"
              size="sm"
              className="w-full text-gray-500 hover:text-gray-700 dark:text-gray-400 dark:hover:text-gray-200"
            >
              View All Alerts ({alerts.length - unreadAlerts.length} read)
            </Button>
          </div>
        )}
      </div>
    </div>
  );
}
```

Code 1.7

AdminPanel.tsx

```tsx
import { useState } from "react";
import { Users, Car, BarChart3, Settings, MapPin, Clock, CheckCircle, XCircle } from "lucide-react";
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from "@/components/ui/card";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";
import { Badge } from "@/components/ui/badge";
import { Button } from "@/components/ui/button";
import { motion } from "framer-motion";
import { useQuery, useMutation, useQueryClient } from "@tanstack/react-query";
import { DriverManagement } from "./admin/DriverManagement";
import { BookingManagement } from "./admin/BookingManagement";
import { AnalyticsDashboard } from "./admin/AnalyticsDashboard";

export function AdminPanel() {
  const [activeTab, setActiveTab] = useState("overview");

  // Sample data - replace with real API calls
  const { data: stats, isLoading } = useQuery({
    queryKey: ["/api/admin/stats"],
    queryFn: async () => ({
      totalBookings: 156,
      activeDrivers: 12,
      completedTrips: 134,
      averageResponseTime: 8.5,
      revenue: 45600,
      pendingBookings: 5,
    }),
  });

  const { data: recentBookings } = useQuery({
    queryKey: ["/api/bookings"],
  });

  const { data: activeDrivers } = useQuery({
    queryKey: ["/api/drivers"],
  });

  if (isLoading) {
    return (
```

Code 2.1

```
      <div className="p-6">
        <div className="animate-pulse space-y-6">
          <div className="h-8 bg-gray-200 dark:bg-gray-700 rounded w-1/3" />
          <div className="grid grid-cols-1 md:grid-cols-4 gap-4">
            {[...Array(4)].map((_, i) => (
              <div key={i} className="h-24 bg-gray-200 dark:bg-gray-700 rounded" />
            ))}
          </div>
        </div>
      </div>
    );
  }

  return (
    <div className="p-6 max-w-7xl mx-auto">
      <motion.div
        initial={{ opacity: 0, y: 20 }}
        animate={{ opacity: 1, y: 0 }}
        className="space-y-6"
      >
        {/* Header */}
        <div className="flex items-center justify-between">
          <div>
            <h1 className="text-3xl font-bold text-gray-900 dark:text-white">Admin Dashboard</h1>
            <p className="text-gray-600 dark:text-gray-400">
              Manage drivers, bookings, and monitor system performance
            </p>
          </div>
          <Button className="bg-medical-500 hover:bg-medical-600 text-white">
            <Settings className="w-4 h-4 mr-2" />
            Settings
          </Button>
        </div>

        {/* Stats Overview */}
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-4">
          <motion.div whileHover={{ scale: 1.02 }}>
```

Code 2.2

```
            <Card>
              <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">
                <CardTitle className="text-sm font-medium">Total Bookings</CardTitle>
                <Car className="h-4 w-4 text-muted-foreground" />
              </CardHeader>
              <CardContent>
                <div className="text-2xl font-bold">{stats?.totalBookings || 0}</div>
                <p className="text-xs text-muted-foreground">
                  +12% from last month
                </p>
              </CardContent>
            </Card>
          </motion.div>

          <motion.div whileHover={{ scale: 1.02 }}>
            <Card>
              <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">
                <CardTitle className="text-sm font-medium">Active Drivers</CardTitle>
                <Users className="h-4 w-4 text-muted-foreground" />
              </CardHeader>
              <CardContent>
                <div className="text-2xl font-bold">{stats?.activeDrivers || 0}</div>
                <p className="text-xs text-muted-foreground">
                  2 new this week
                </p>
              </CardContent>
            </Card>
          </motion.div>

          <motion.div whileHover={{ scale: 1.02 }}>
            <Card>
              <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">
                <CardTitle className="text-sm font-medium">Avg Response Time</CardTitle>
                <Clock className="h-4 w-4 text-muted-foreground" />
              </CardHeader>
              <CardContent>
                <div className="text-2xl font-bold">{stats?.averageResponseTime || 0}m</div>
```

Code 2.3

```
                <p className="text-xs text-muted-foreground">
                  -15% improvement
                </p>
              </CardContent>
            </Card>
          </motion.div>

          <motion.div whileHover={{ scale: 1.02 }}>
            <Card>
              <CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">
                <CardTitle className="text-sm font-medium">Monthly Revenue</CardTitle>
                <BarChart3 className="h-4 w-4 text-muted-foreground" />
              </CardHeader>
              <CardContent>
                <div className="text-2xl font-bold">${stats?.revenue?.toLocaleString() || 0}</div>
                <p className="text-xs text-muted-foreground">
                  +8% from last month
                </p>
              </CardContent>
            </Card>
          </motion.div>
        </div>

        {/* Main Content Tabs */}
        <Tabs value={activeTab} onValueChange={setActiveTab} className="space-y-4">
          <TabsList className="grid w-full grid-cols-4">
            <TabsTrigger value="overview">Overview</TabsTrigger>
            <TabsTrigger value="bookings">Bookings</TabsTrigger>
            <TabsTrigger value="drivers">Drivers</TabsTrigger>
            <TabsTrigger value="analytics">Analytics</TabsTrigger>
          </TabsList>

          <TabsContent value="overview" className="space-y-4">
            <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
              {/* Recent Bookings */}
              <Card>
                <CardHeader>
```

Code 2.4

```
            <CardContent>
              <div className="space-y-4">
                {[
                  { id: "BK001", patient: "John Doe", status: "completed", time: "2 hours ago", urgency: "high" },
                  { id: "BK002", patient: "Jane Smith", status: "in_transit", time: "30 min ago", urgency: "critical" },
                  { id: "BK003", patient: "Mike Johnson", status: "pending", time: "5 min ago", urgency: "medium" },
                ].map((booking, index) => (
                  <motion.div
                    key={booking.id}
                    initial={{ opacity: 0, x: -20 }}
                    animate={{ opacity: 1, x: 0 }}
                    transition={{ delay: index * 0.1 }}
                    className="flex items-center justify-between p-3 bg-gray-50 dark:bg-gray-700 rounded-lg"
                  >
                    <div className="flex items-center space-x-3">
                      <div className="w-8 h-8 bg-medical-500 rounded-full flex items-center justify-center text-white text-sm font-medium">
                        {booking.id.slice(-2)}
                      </div>
                      <div>
                        <div className="font-medium text-gray-900 dark:text-white">
                          {booking.patient}
                        </div>
                        <div className="text-sm text-gray-500 dark:text-gray-400">
                          {booking.time}
                        </div>
                      </div>
                    </div>
                    <div className="flex items-center space-x-2">
                      <Badge
                        variant={booking.urgency === "critical" ? "destructive" : "secondary"}
                      >
                        {booking.urgency}
                      </Badge>
                      <Badge
                        variant={booking.status === "completed" ? "default" : "outline"}
                        className={
                          booking.status === "completed" ? "bg-green-100 text-green-800" :
```

Code 2.5

```
                          booking.status === "in_transit" ? "bg-blue-100 text-blue-800" :
                          "bg-yellow-100 text-yellow-800"
                        }
                      >
                        {booking.status}
                      </Badge>
                    </div>
                  </motion.div>
                ))}
              </div>
            </CardContent>
          </Card>

          {/* Active Drivers */}
          <Card>
            <CardHeader>
              <CardTitle>Active Drivers</CardTitle>
              <CardDescription>Currently on duty</CardDescription>
            </CardHeader>
            <CardContent>
              <div className="space-y-4">
                {[
                  { id: "DR001", name: "John Smith", ambulance: "AMB-001", status: "assigned", location: "Downtown" },
                  { id: "DR002", name: "Sarah Johnson", ambulance: "AMB-002", status: "available", location: "Uptown" },
                  { id: "DR003", name: "Mike Wilson", ambulance: "AMB-003", status: "busy", location: "Hospital" },
                ].map((driver, index) => (
                  <motion.div
                    key={driver.id}
                    initial={{ opacity: 0, x: -20 }}
                    animate={{ opacity: 1, x: 0 }}
                    transition={{ delay: index * 0.1 }}
                    className="flex items-center justify-between p-3 bg-gray-50 dark:bg-gray-700 rounded-lg"
                  >
                    <div className="flex items-center space-x-3">
                      <div className="w-8 h-8 bg-green-500 rounded-full flex items-center justify-center">
                        <Users className="w-4 h-4 text-white" />
```

Code 2.6

```
                    </div>
                    <div>
                      <div className="font-medium text-gray-900 dark:text-white">
                        {driver.name}
                      </div>
                      <div className="text-sm text-gray-500 dark:text-gray-400 flex items-center">
                        <MapPin className="w-3 h-3 mr-1" />
                        {driver.location} • {driver.ambulance}
                      </div>
                    </div>
                  </div>
                  <Badge
                    variant={driver.status === "available" ? "default" : "secondary"}
                    className={
                      driver.status === "available" ? "bg-green-100 text-green-800" :
                      driver.status === "assigned" ? "bg-blue-100 text-blue-800" :
                      "bg-gray-100 text-gray-800"
                    }
                  >
                    {driver.status}
                  </Badge>
                </motion.div>
              ))}
            </div>
          </CardContent>
        </Card>
      </div>
    </TabsContent>

    <TabsContent value="bookings">
      <BookingManagement />
    </TabsContent>

    <TabsContent value="drivers">
      <DriverManagement />
    </TabsContent>
```

Code 2.7

React Hooks

use-mobile.tsx

```tsx
import * as React from "react"

const MOBILE_BREAKPOINT = 768

export function useIsMobile() {
  const [isMobile, setIsMobile] = React.useState<boolean | undefined>(undefined)

  React.useEffect(() => {
    const mql = window.matchMedia(`(max-width: ${MOBILE_BREAKPOINT - 1}px)`)
    const onChange = () => {
      setIsMobile(window.innerWidth < MOBILE_BREAKPOINT)
    }
    mql.addEventListener("change", onChange)
    setIsMobile(window.innerWidth < MOBILE_BREAKPOINT)
    return () => mql.removeEventListener("change", onChange)
  }, [])

  return !!isMobile
}
```

Code 3.1

useBatteryStatus.tsx

```tsx
import { useState, useEffect } from "react";
import type { BatteryInfo } from "@/types";

// Battery API is experimental and may not be available in all browsers
interface NavigatorWithBattery extends Navigator {
  getBattery?: () => Promise<BatteryManager>;
}

interface BatteryManager extends EventTarget {
  charging: boolean;
  chargingTime: number;
  dischargingTime: number;
  level: number;
  addEventListener(type: 'chargingchange' | 'levelchange', listener: EventListener): void;
  removeEventListener(type: 'chargingchange' | 'levelchange', listener: EventListener): void;
}

export function useBatteryStatus(): BatteryInfo {
  const [batteryInfo, setBatteryInfo] = useState<BatteryInfo>({
    level: 1, // Default to 100%
    charging: false,
    supported: false,
  });

  useEffect(() => {
    const nav = navigator as NavigatorWithBattery;

    if (!nav.getBattery) {
      setBatteryInfo(prev => ({ ...prev, supported: false }));
      return;
    }
                + interface BatteryManager
    let battery: BatteryManager | null = null;

    const updateBatteryInfo = () => {
      if (battery) {
        setBatteryInfo({
          level: battery.level,
```

Code 4.1

```
            charging: battery.charging,
            supported: true,
          });
        }
      };

      nav.getBattery()
        .then((batteryManager) => {
          battery = batteryManager;
          updateBatteryInfo();

          battery.addEventListener('chargingchange', updateBatteryInfo);
          battery.addEventListener('levelchange', updateBatteryInfo);
        })
        .catch(() => {
          setBatteryInfo(prev => ({ ...prev, supported: false }));
        });

      return () => {
        if (battery) {
          battery.removeEventListener('chargingchange', updateBatteryInfo);
          battery.removeEventListener('levelchange', updateBatteryInfo);
        }
      };
    }, []);

    return batteryInfo;
  }

  export function useAmbulanceBatteryAlert(threshold = 0.2) {
    const batteryInfo = useBatteryStatus();
    const [hasAlerted, setHasAlerted] = useState(false);

    useEffect(() => {
      if (batteryInfo.supported &&
```

Code 4.2

```
          batteryInfo.level <= threshold &&
          !batteryInfo.charging &&
          !hasAlerted) {

        // Create low battery alert
        const alertEvent = new CustomEvent('batteryLow', {
          detail: {
            level: Math.round(batteryInfo.level * 100),
            message: `Ambulance device battery is low (${Math.round(batteryInfo.level * 100)}%)`
          }
        });

        window.dispatchEvent(alertEvent);
        setHasAlerted(true);
      }

      // Reset alert if battery level improves
      if (batteryInfo.level > threshold + 0.1 || batteryInfo.charging) {
        setHasAlerted(false);
      }
    }, [batteryInfo.level, batteryInfo.charging, threshold, hasAlerted, batteryInfo.supported]);

    return {
      ...batteryInfo,
      isLow: batteryInfo.level <= threshold && !batteryInfo.charging,
      percentage: Math.round(batteryInfo.level * 100),
    };
  }

```

Code 4.3

useFetchAmbulance.tsx

```tsx
import { useQuery } from "@tanstack/react-query";
import { useEffect } from "react";
import type { Ambulance } from "@/types";

export function useFetchAmbulances() {
  const query = useQuery<Ambulance[]>({
    queryKey: ["/api/ambulances"],
    refetchInterval: 10000, // Refetch every 10 seconds
  });

  return query;
}

export function useFetchAmbulance(id: string) {
  const query = useQuery<Ambulance>({
    queryKey: ["/api/ambulances", id],
    enabled: !!id,
    refetchInterval: 5000, // Refetch every 5 seconds for individual ambulance
  });

  return query;
}

export function useFetchNearbyAmbulances(lat?: number, lng?: number, radius = 0.1) {
  const query = useQuery<Ambulance[]>({
    queryKey: ["/api/ambulances/nearby", { lat, lng, radius }],
    enabled: !!(lat && lng),
    refetchInterval: 10000,
    queryFn: async () => {
      if (!lat || !lng) throw new Error("Location required");

      const response = await fetch(
        `/api/ambulances/nearby?lat=${lat}&lng=${lng}&radius=${radius}`
      );

      if (!response.ok) {
        throw new Error("Failed to fetch nearby ambulances");
      }
```

Code 5.1

useLocalStorage.tsx

```tsx
import { useState, useEffect } from "react";

export function useLocalStorage<T>(
  key: string,
  initialValue: T
): [T, (value: T | ((val: T) => T)) => void] {
  // Get from local storage then parse stored json or return initialValue
  const [storedValue, setStoredValue] = useState<T>(() => {
    try {
      const item = window.localStorage.getItem(key);
      return item ? JSON.parse(item) : initialValue;
    } catch (error) {
      console.error(`Error reading localStorage key "${key}":`, error);
      return initialValue;
    }
  });

  // Return a wrapped version of useState's setter function that persists the new value to localStorage
  const setValue = (value: T | ((val: T) => T)) => {
    try {
      // Allow value to be a function so we have the same API as useState
      const valueToStore = value instanceof Function ? value(storedValue) : value;
      setStoredValue(valueToStore);
      window.localStorage.setItem(key, JSON.stringify(valueToStore));
    } catch (error) {
      console.error(`Error setting localStorage key "${key}":`, error);
    }
  };

  return [storedValue, setValue];
}

// Specific hooks for common use cases
export function useFavoriteHospitals() {
  return useLocalStorage<string[]>("favoriteHospitals", []);
}

export function useEmergencyHistory() {
```

Code 6.1

```tsx
  return useLocalStorage<Array<{
    id: string;
    date: string;
    hospital: string;
    duration: string;
    status: string;
  }>>("emergencyHistory", []);
}

export function useEmergencyContacts() {
  return useLocalStorage<Array<{
    id: string;
    name: string;
    phone: string;
    relationship: string;
  }>>("emergencyContacts", []);
}
```

Code 6.2

settingsContext.tsx

```tsx
import React, { createContext, useContext, useState, useEffect } from "react";
import type { SettingsContextType, Theme, Language, EmergencyStatus } from "@/types";

const SettingsContext = createContext<SettingsContextType | undefined>(undefined);

export function SettingsProvider({ children }: { children: React.ReactNode }) {
  const [theme, setTheme] = useState<Theme>("light");
  const [language, setLanguage] = useState<Language>("en");
  const [emergencyStatus, setEmergencyStatus] = useState<EmergencyStatus>("requested");
  const [isOnline, setIsOnline] = useState(true);

  // Initialize theme from localStorage
  useEffect(() => {
    const savedTheme = localStorage.getItem("theme") as Theme;
    if (savedTheme) {
      setTheme(savedTheme);
      document.documentElement.classList.toggle("dark", savedTheme === "dark");
    }
  }, []);

  // Initialize language from localStorage
  useEffect(() => {
    const savedLanguage = localStorage.getItem("language") as Language;
    if (savedLanguage) {
      setLanguage(savedLanguage);
    }
  }, []);

  // Monitor online status
  useEffect(() => {
    const updateOnlineStatus = () => setIsOnline(navigator.onLine);

    window.addEventListener("online", updateOnlineStatus);
    window.addEventListener("offline", updateOnlineStatus);

    return () => {
      window.removeEventListener("online", updateOnlineStatus);
      window.removeEventListener("offline", updateOnlineStatus);
```

Code 7.1

```
      return response.json();
    },
  });

  return query;
}

export function useSimulateAmbulanceMovement(ambulanceId: string, enabled = false) {
  useEffect(() => {
    if (!enabled || !ambulanceId) return;

    const interval = setInterval(async () => {
      try {
        await fetch(`/api/ambulances/${ambulanceId}/simulate-movement`, {
          method: "POST",
        });
      } catch (error) {
        console.error("Failed to simulate ambulance movement:", error);
      }
    }, 8000); // Move every 8 seconds

    return () => clearInterval(interval);
  }, [ambulanceId, enabled]);
}

```

Code 7.2

```
  const toggleTheme = () => {
    const newTheme = theme === "light" ? "dark" : "light";
    setTheme(newTheme);
    localStorage.setItem("theme", newTheme);
    document.documentElement.classList.toggle("dark", newTheme === "dark");
  };

  const toggleLanguage = () => {
    const newLanguage = language === "en" ? "hi" : "en";
    setLanguage(newLanguage);
    localStorage.setItem("language", newLanguage);
  };

  const updateEmergencyStatus = (status: EmergencyStatus) => {
    setEmergencyStatus(status);
  };

  const setOnlineStatus = (status: boolean) => {
    setIsOnline(status);
  };

  const value: SettingsContextType = {
    theme,
    language,
    emergencyStatus,
    isOnline,
    toggleTheme,
    toggleLanguage,
    updateEmergencyStatus,
    setOnlineStatus,
  };
```

Code 7.3

```
export function useSettings() {
  const context = useContext(SettingsContext);
  if (context === undefined) {
    throw new Error("useSettings must be used within a SettingsProvider");
  }
  return context;
}

```

Code 7.4

SCREENSHOTS

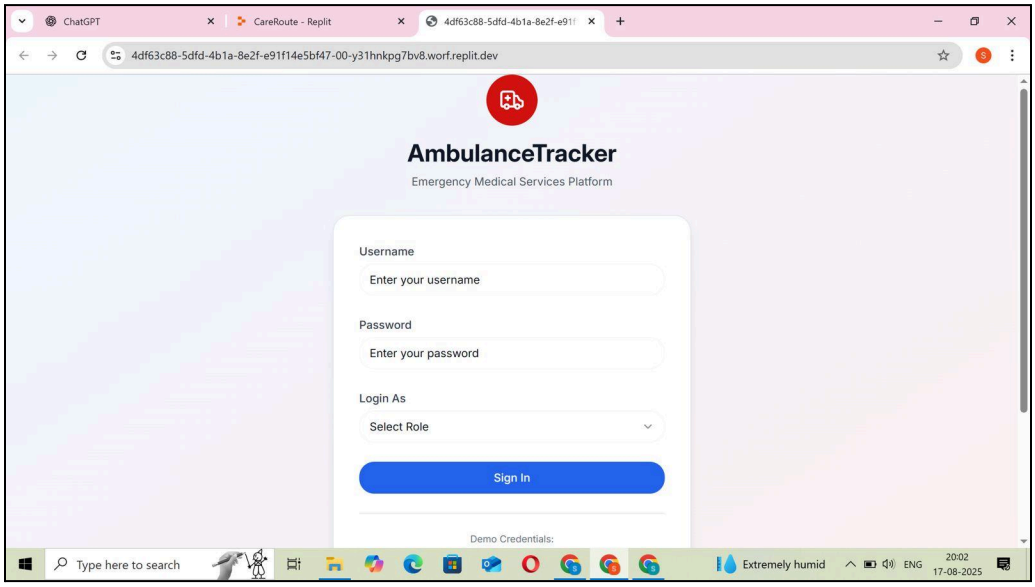

Figure 1.1

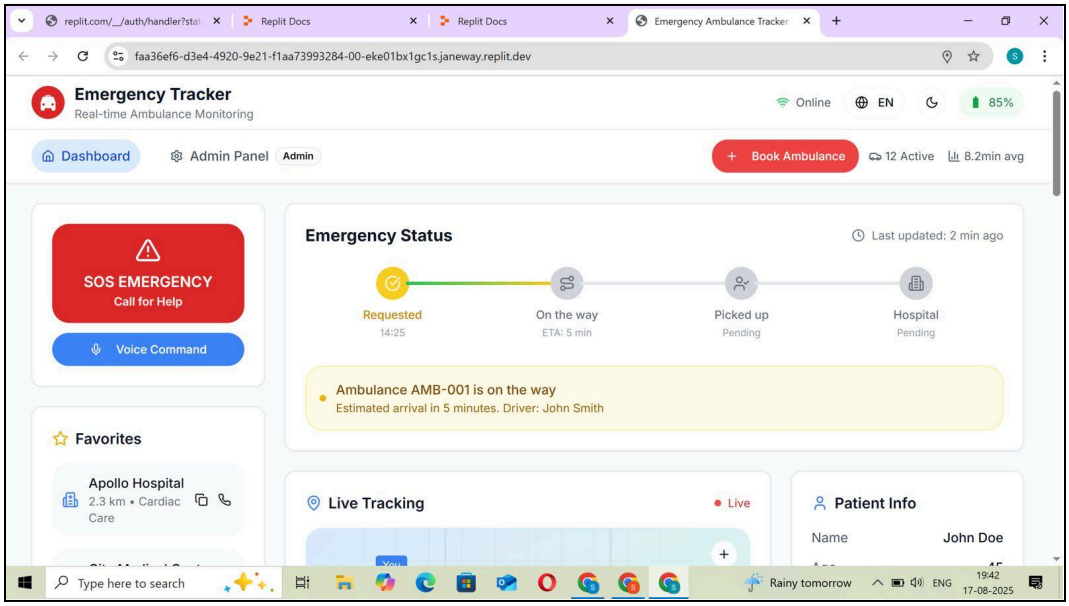

Figure 1.2

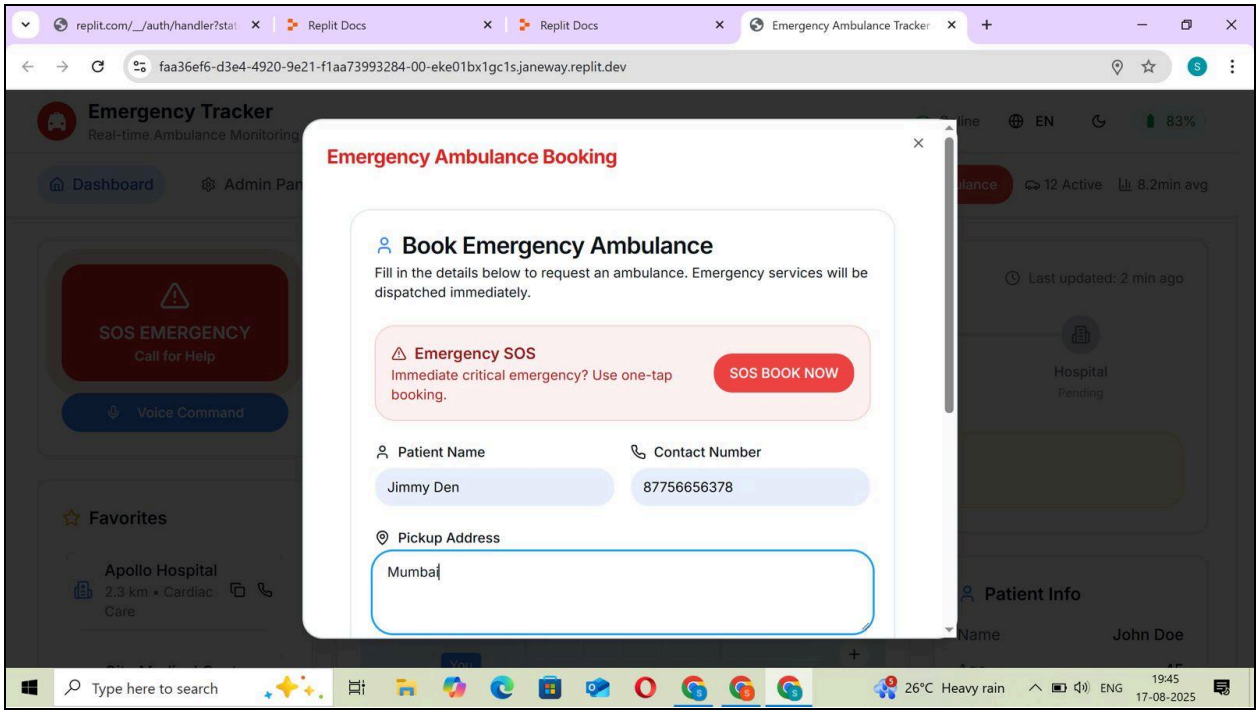

Figure 1.3

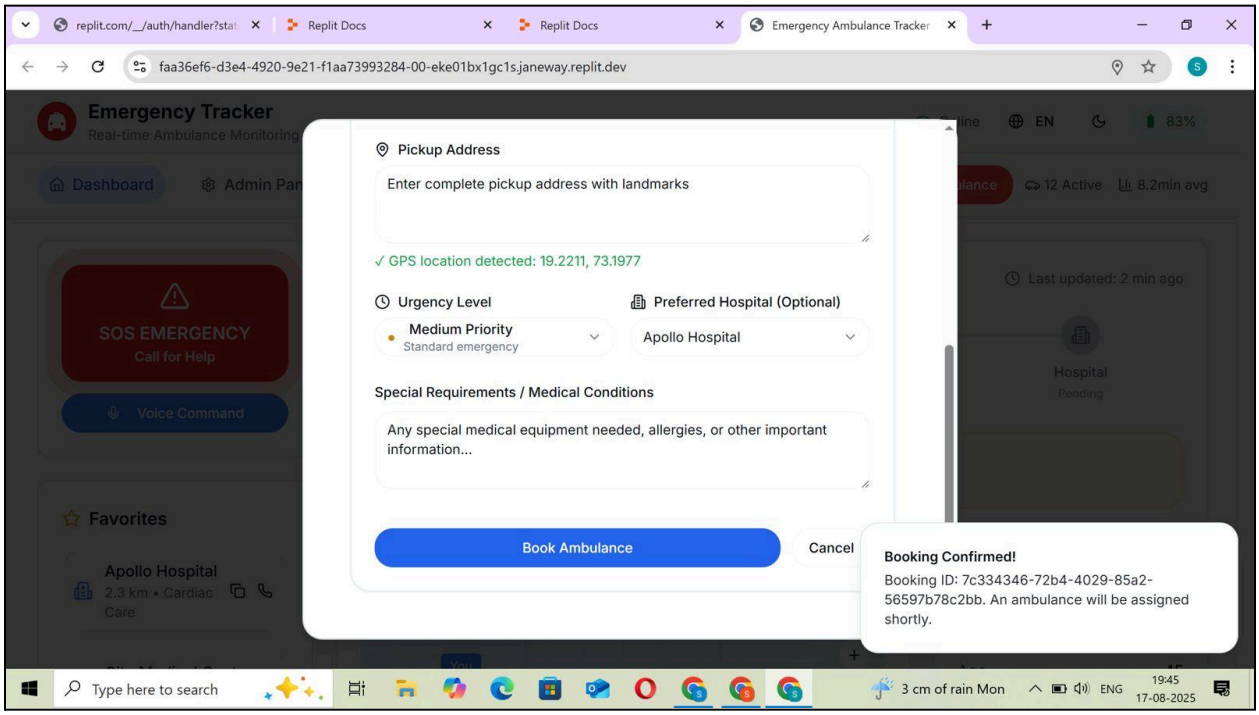

Figure 1.4

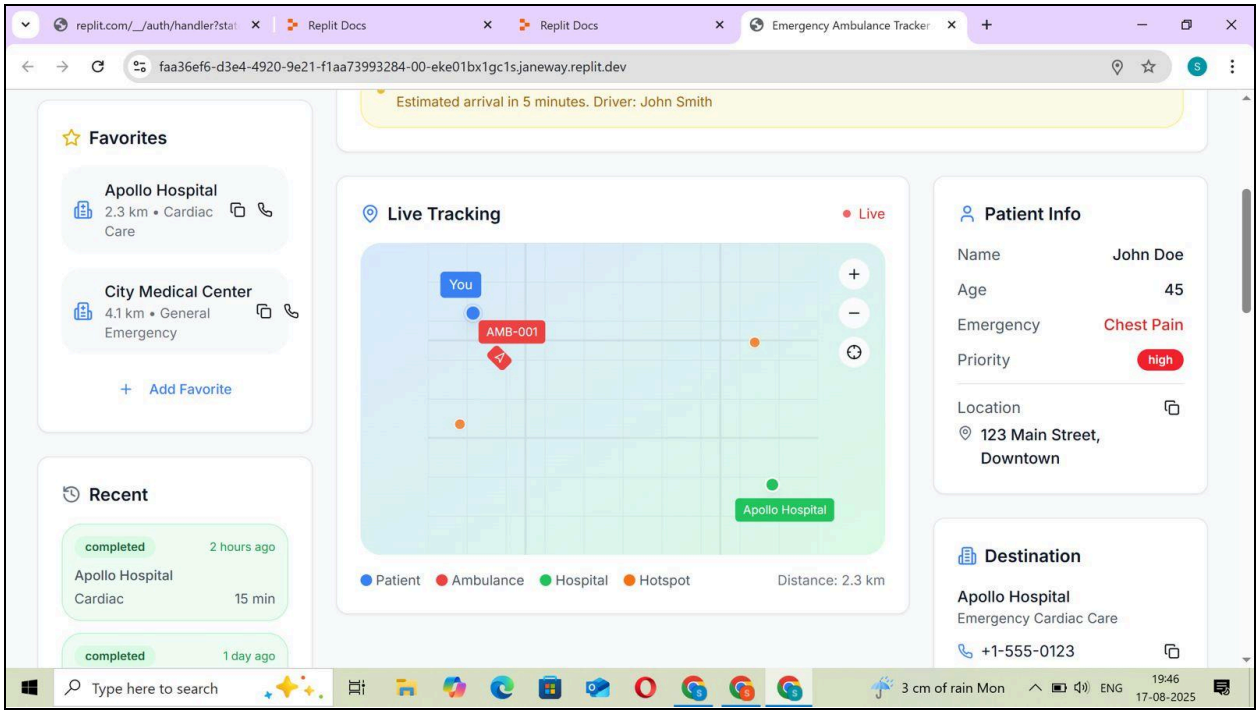

Figure 1.5

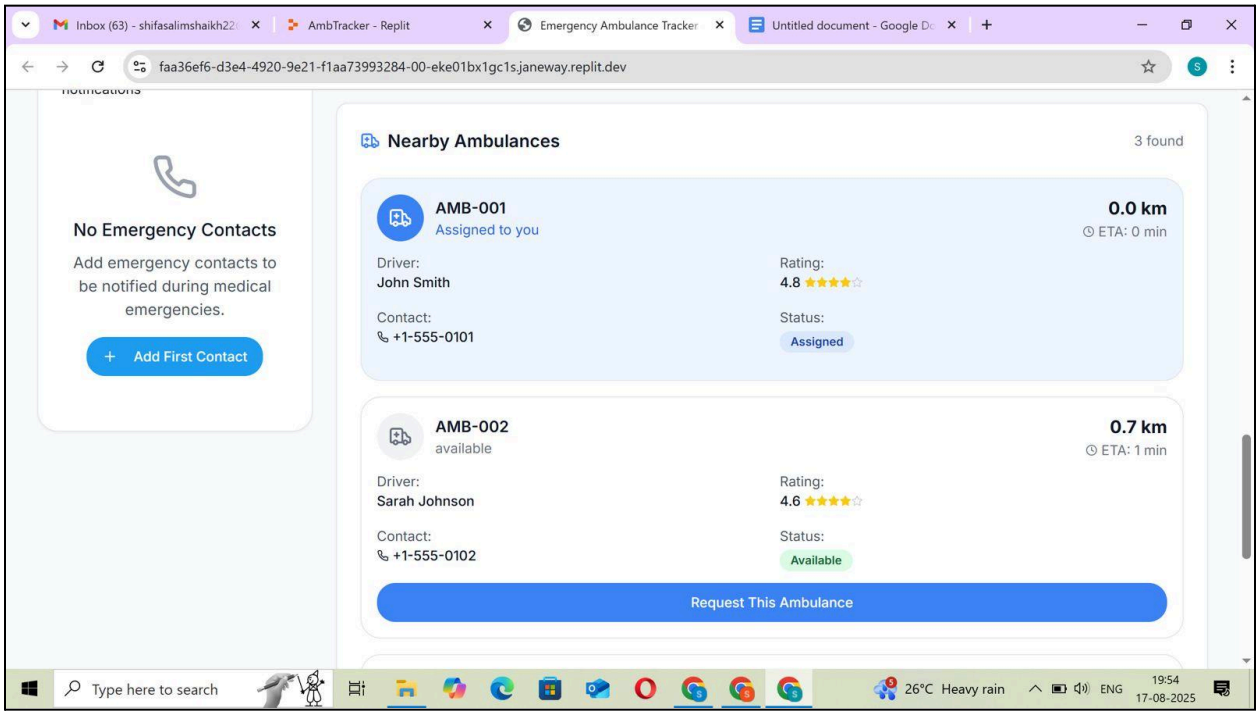

Figure 1.6

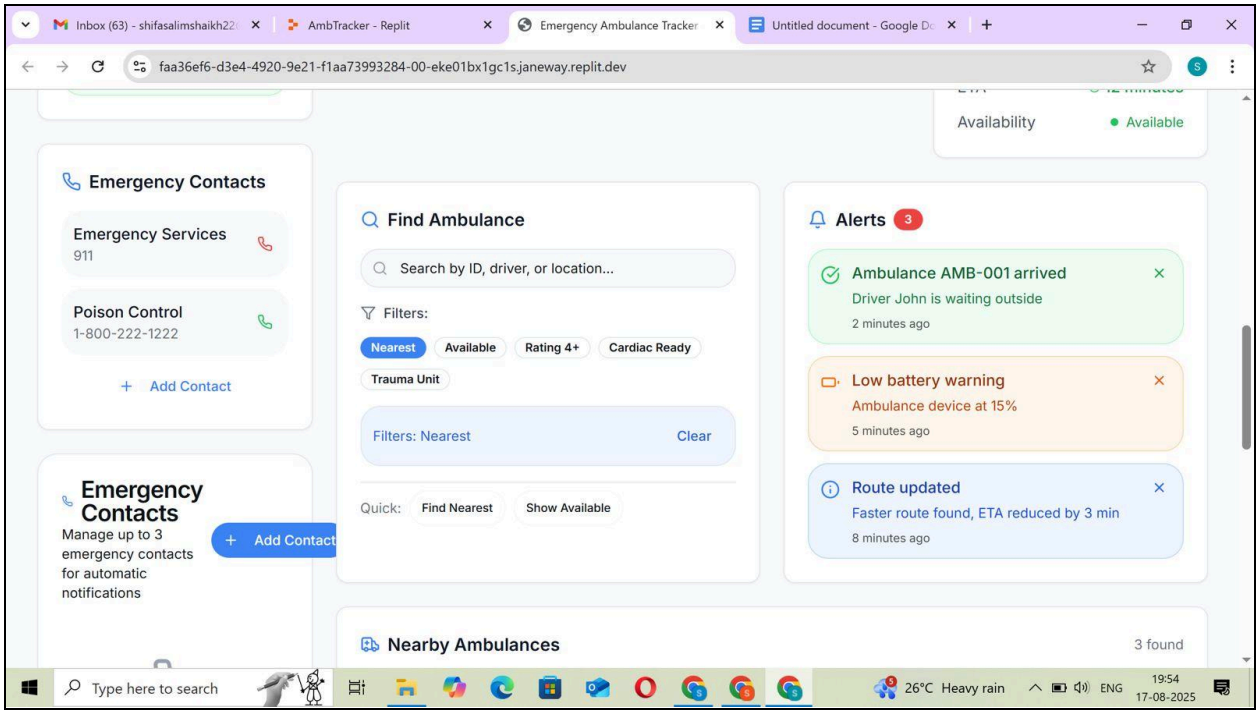

Figure 1.7

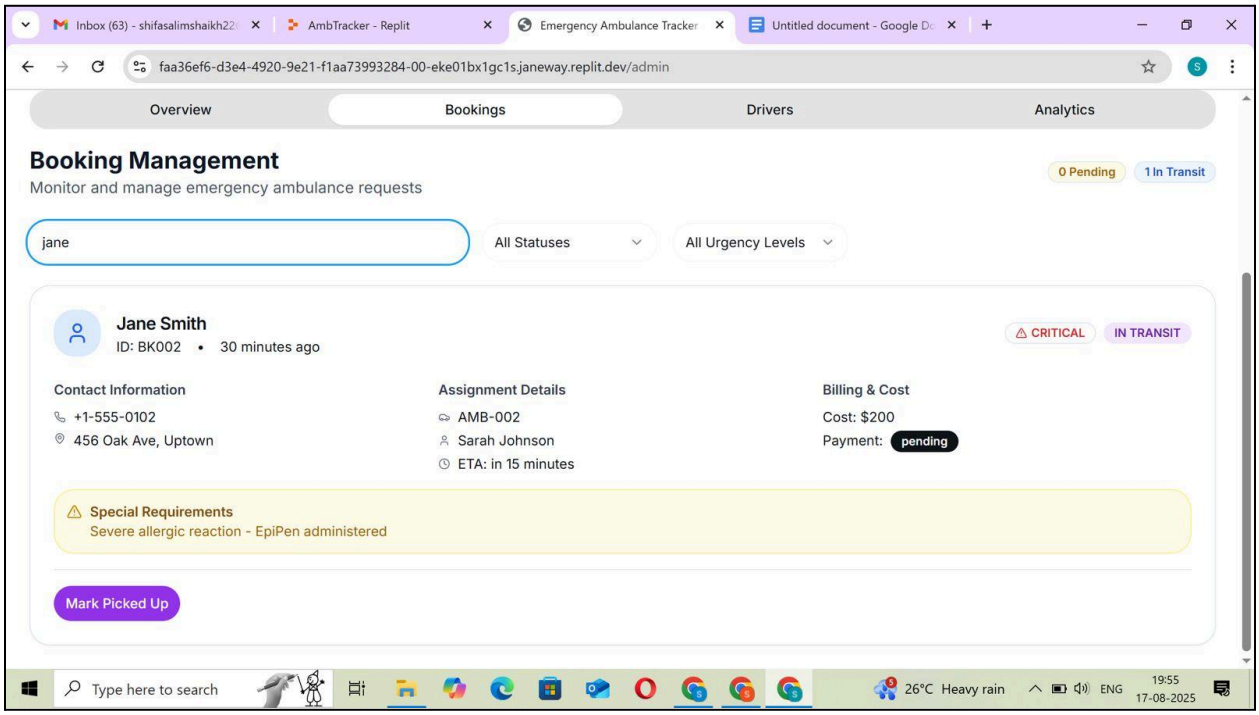

Figure 1.8

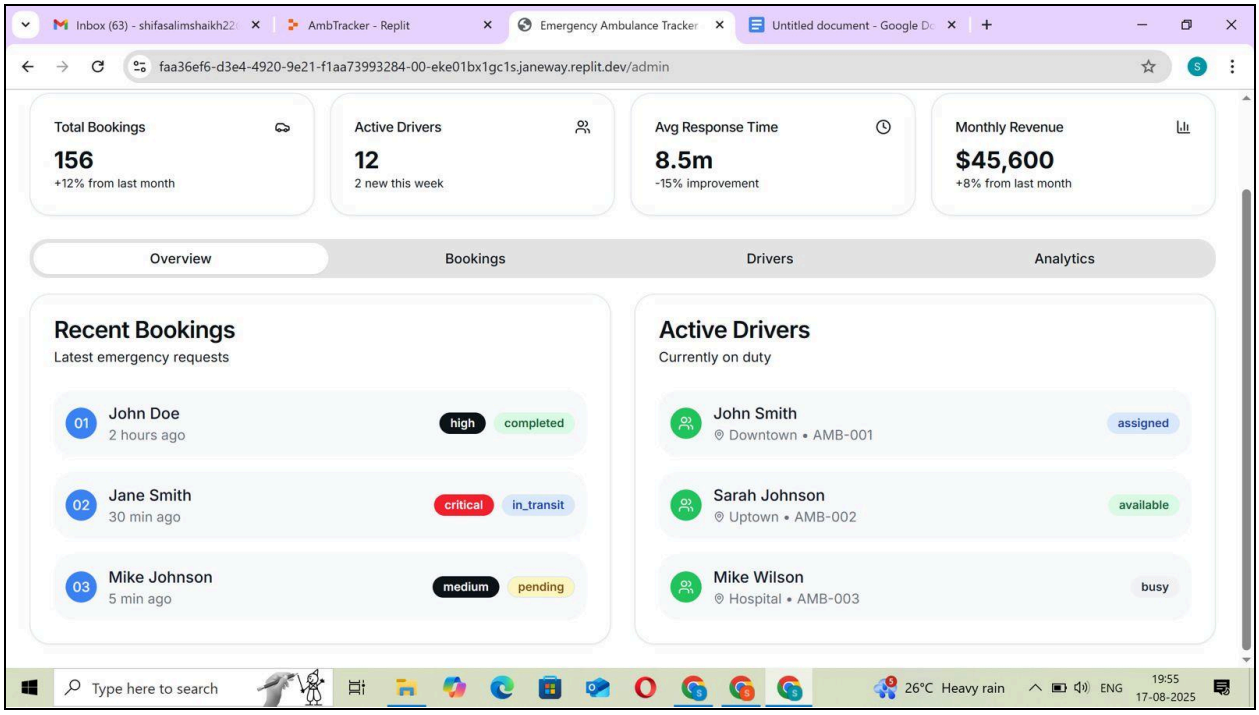

Figure 1.9

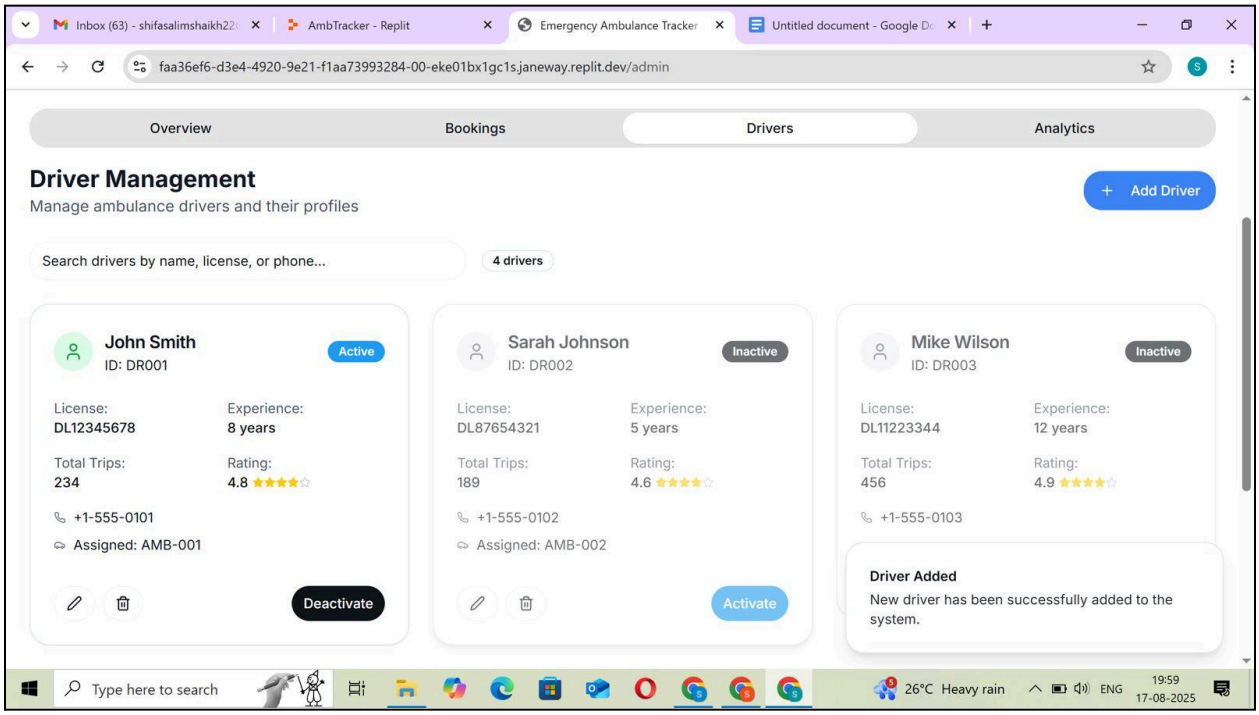

Figure 1.10

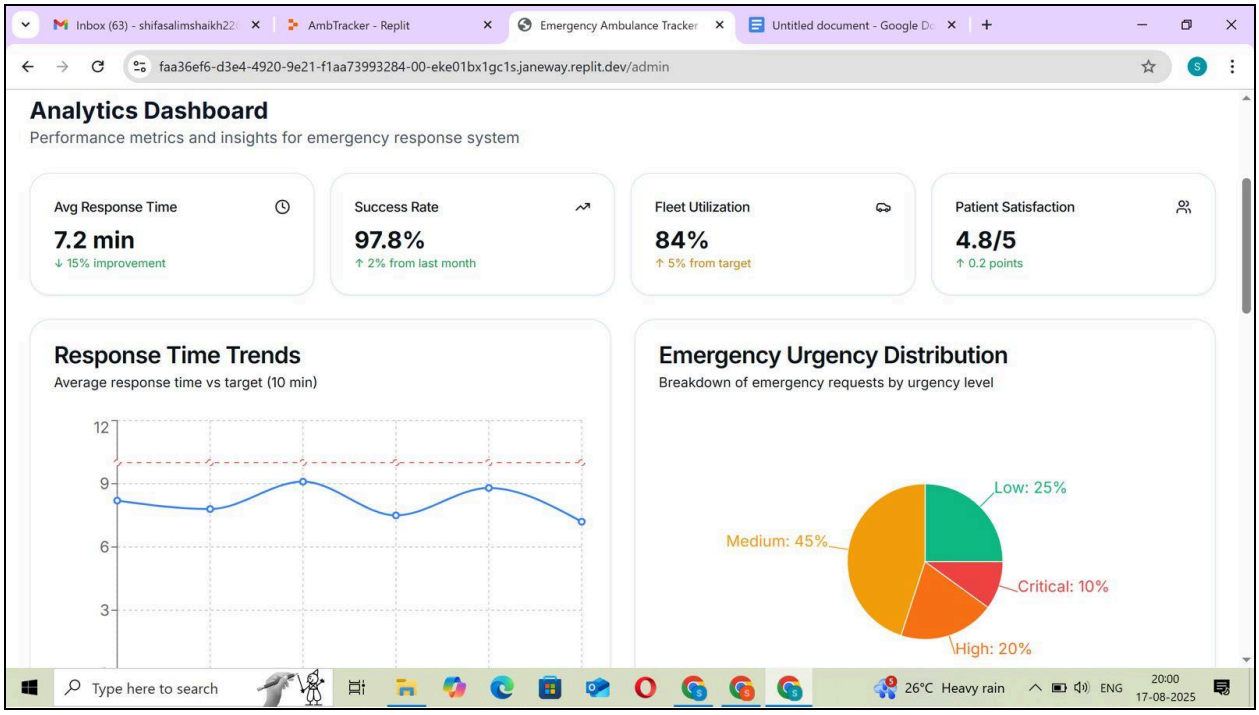

Figure 1.11

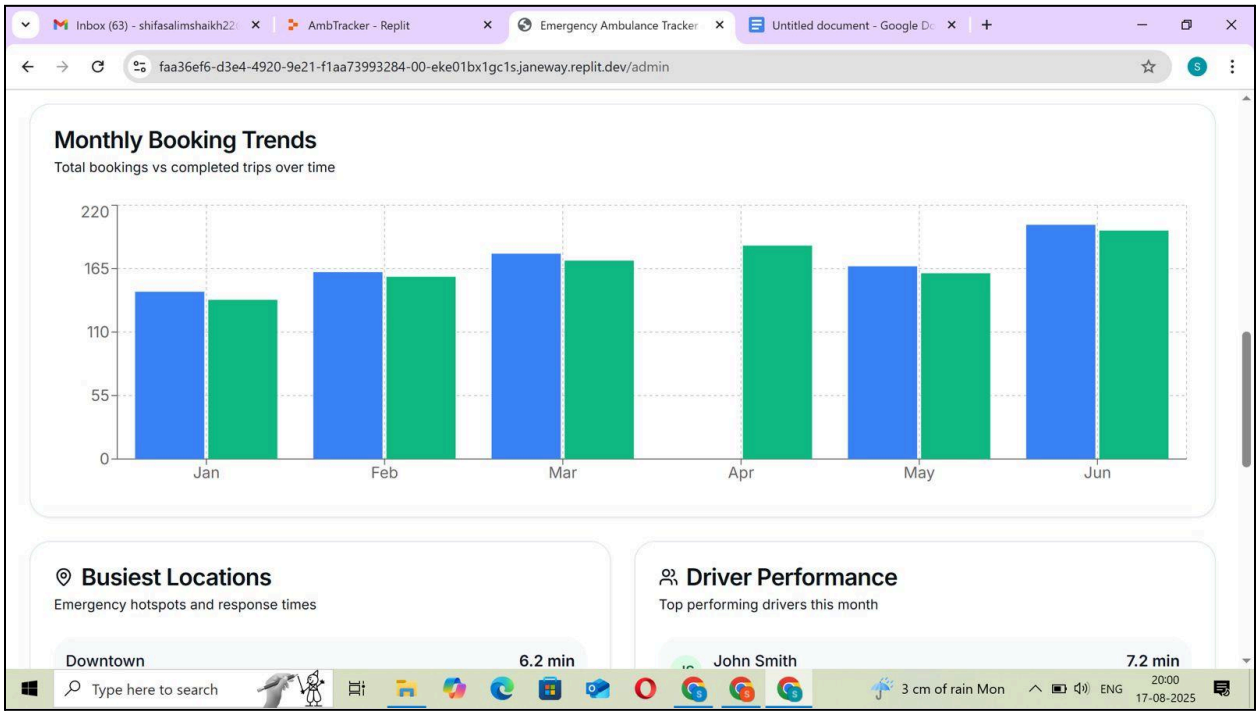

Figure 1.12

30% EXTRA CONTRIBUTION

In addition to the base implementation of the experiment, we contributed an extra 30% effort by introducing improvements in both the technical workflow and the way results were presented.

- **Contextual State Management with useContext**
  Instead of relying on prop drilling, we implemented global state management using useContext. This allowed ambulance location, availability, and status to be shared across components without external libraries, ensuring cleaner and more scalable code.

- **Custom Hooks for Code Reusability**
  We encapsulated repeated logic into custom hooks (such as useLiveLocation, useMapUpdate, and useAmbulanceStatus). This modular approach promoted reusability, maintainability, and clear separation of concerns within the project.

- **Enhanced Visualizations & Plots**
  To make the results more interpretable and visually clear, we included additional visualizations such as before vs. after comparison graphs, stress-level variation plots, confusion matrices, time–frequency spectrograms, heatmaps, and waveform plots. We also provided summary dashboards for easy-to-read insights.

Through these contributions, we extended the project beyond the standard implementation, adding value in terms of both **technical depth** and **visual clarity of results**.

CONCLUSION

React Hooks provide a concise, functional, and modular approach to building real-time applications.

- useEffect manages side effects like connections and subscriptions.
- useContext provides efficient global state management.
- Custom hooks enable abstraction and reuse.

Through the example of a real-time ambulance tracker, it was observed that Hooks improve maintainability, reduce complexity, and allow better separation of concerns. However, developers must remain cautious about common pitfalls such as dependency mismanagement in useEffect or excessive re-renders from useContext.