RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CB23332
SOFTWARE ENGINEERING LAB

Laboratory Record Note Book

Name : . . . . . .

Year / Branch / Section : . . . . . .

Register No. : . . . . . .

Semester : . . . . . .

Academic Year : . . . . . .

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)
RAJALAKSHMI NAGAR, THANDALAM – 602-105

BONAFIDE CERTIFICATE

NAME:_____**REGISTER NO.:** _____

ACADEMIC YEAR: 2024-25 **SEMESTER:** III **BRANCH:**_____B.E/B.Tech

This Certification is the bonafide record of work done by the above student in the

CB23332-SOFTWARE ENGINEERING - Laboratory during the year 2024 – 2025.

Signature of Faculty -in – Charge

Submitted for the Practical Examination held on _____

Internal Examiner

External Examiner

INDEX

S.No.	Name of the Experiment	Expt. Date	Faculty Sign
1.	Preparing Problem Statement		
2.	Software Requirement Specification (SRS)		
3.	Entity-Relational Diagram		
4.	Data Flow Diagram		
5.	Use Case Diagram		
6.	Activity Diagram		
7.	State Chart Diagram		
8.	Sequence Diagram		
9.	Collaboration Diagramt		
10.	Class Diagram		

1.Preparing problem statement

Aim:

Traditional ambulance management systems often encounter issues such as inefficient resource allocation, lack of real-time tracking, and limited integration with emergency response units. These challenges lead to delayed response times, ineffective coordination, and poor user experience. Patients and healthcare providers face difficulties in tracking ambulance availability, estimated arrival times, and booking processes. A Smart Ambulance Management System (SAMS)** is required to address these challenges by automating ambulance dispatch, enabling real-time tracking, and integrating emergency response services. The system will provide users with live updates on ambulance location, automated notifications for dispatch status, and a user-friendly interface for bookings, significantly enhancing the efficiency of ambulance operations and user satisfaction.

Algorithm:

1.Initialize System:

- Load the database containing details of ambulances, drivers, hospitals, and patients.

2.User Login:

- Allow users (patients, healthcare providers, and admin) to log in using credentials.

3.Ambulance Search:

- User searches for an ambulance by location, type, or availability.
- The system displays real-time availability of ambulances.

4.Ambulance Booking:

- User books an ambulance based on availability.
- The system records the transaction and assigns the nearest available ambulance.

5.Real-Time Tracking:

- Enable users to track the ambulance’s location via GPS.
- Provide estimated arrival times.

6.Notifications:

- Send automated notifications for booking confirmation, estimated arrival, or delays.

7.Emergency Requests:

- Enable users to prioritize emergencies for quicker response.
- Notify drivers and hospitals instantly for emergency cases.

8.Resource Allocation:

- Optimize ambulance allocation based on proximity, availability, and urgency.

9.Reporting:

- Generate reports on usage patterns, response times, and system performance.

10.Feedback and Review:

- Collect user feedback for continuous improvement.

Inputs for the Smart Ambulance Management System:

1.User Details:

- Username, password for login.
- Contact information for notifications and feedback.

2.Ambulance Details:

- Vehicle ID, type (basic life support, advanced life support), availability, and driver details.

3.Booking Details:

- User-initiated requests for ambulance services, including pick-up and drop-off locations.

4.Real-Time Data:

- GPS tracking data, traffic updates, and estimated travel times.

5.Notifications and Alerts:

- Inputs for automated alerts, such as booking confirmations, arrival estimates, and system updates.

6. Emergency Requests:

- Information on priority cases and required resources.

Stakeholder Problem Statement: Smart Ambulance Management System

Problem:

Existing ambulance management systems are inefficient, lacking real-time tracking and seamless integration with emergency services. Users face delays in booking ambulances, tracking their location, and coordinating with hospitals, leading to compromised response times and overall dissatisfaction.

Background:

Ambulance services in [Location Name] are struggling with outdated processes and a lack of automation. Patients and healthcare providers have expressed frustration over long waiting times, lack of real-time updates, and unreliable dispatch coordination. These inefficiencies impact emergency response effectiveness and user trust.

Relevance:

An efficient ambulance management system is essential for saving lives and ensuring timely healthcare. Providing real-time updates, optimized dispatch, and automated notifications will significantly enhance operational efficiency and user experience. A smart system will reduce delays, improve resource utilization, and streamline emergency services.

Objectives:

1. Automate ambulance booking, dispatching, and tracking.

2. Enable real-time GPS tracking and estimated arrival times.

3. Optimize resource allocation for emergencies and reduce response times.

4. Provide automated notifications for booking status and dispatch updates.

5. Integrate seamlessly with hospitals and emergency services.

6. Improve overall user experience and operational efficiency.

Result:

The problem statement was written successfully by following the steps described above.

2 .Write the software requirement specification document

Aim:

The aim of the Smart Ambulance Management System (SAMS) is to streamline and automate key operations related to ambulance services. These include real-time ambulance tracking, booking, and resource allocation. The system will enhance emergency response efficiency by integrating GPS tracking, automated notifications, and resource optimization for both users and administrators.

Algorithm:

1. Introduction

1.1 Purpose:

To design and implement a Smart Ambulance Management System that optimizes emergency ambulance services by enabling real-time tracking, booking, and efficient resource utilization.

1.2 Scope:

Defines functionalities such as ambulance booking, real-time tracking, integration of medical resources, and automated notifications.

1.3 Definitions, Acronyms, and Abbreviations:

A list of terms, acronyms, and abbreviations for better understanding (e.g., GPS: Global Positioning System, SAMS: Smart Ambulance Management System).

1.4 References:

Includes reference documents and tools used for requirements gathering, such as IEEE SRS templates and API documentation for GPS systems.

1.5 Overview:

Summarizes the document’s structure, explaining the purpose and features of the system.

2. Overall Description

2.1 Product Perspective:

Describes the context of the system and how it integrates with the current healthcare infrastructure to enhance emergency response services.

2.2 Product Features:

Key features include:

-Ambulance Booking and Scheduling: Instant booking of ambulances based on proximity.

- Real-time Ambulance Tracking: Users can track ambulance location through GPS.

-Automated Notifications: Alerts for booking confirmation, arrival, and estimated time of arrival (ETA).

-Integration with Hospitals: Direct linkage with hospital emergency rooms for better preparedness.

2.3 User Classes and Characteristics:

Identifies key user groups:

- Patients and Families:Users who book and track ambulances.

- Ambulance Operators: Manage ambulance operations and routes.

- Hospital Staff/Admins: Manage incoming cases and allocate resources efficiently.

2.4 Operating Environment:

The system will operate on web and mobile platforms with cloud-hosted databases. GPS tracking is enabled via Android/iOS apps, and the system integrates with existing hospital management software.

2.5 Design and Implementation Constraints:

- Compatibility with standard web browsers and mobile platforms.
- Adherence to healthcare data standards and privacy regulations.

2.6 Assumptions and Dependencies:

- Assumes users have basic digital literacy.
- Requires reliable internet connectivity and GPS accuracy.
- Depends on existing hospital databases for real-time integration.

3. System Features

3.1 User Account Management:

Allows users to create accounts, log in, and manage their profiles.

3.2 Ambulance Booking and Scheduling:

Users can book ambulances and schedule their services in advance.

3.3 Real-time Tracking:

Tracks ambulances using GPS to provide live location updates.

3.4 Notifications and Alerts:

Automated notifications for booking confirmation, ambulance arrival, and updates.

3.5 Resource Integration:

Integration of medical resources (e.g., ventilators, first-aid kits) for tracking availability.

3.6 Reporting and Analytics:

Provides reports on response times, bookings, and resource utilization for admins.

4. External Interface Requirements

4.1 User Interfaces:

Web and mobile interfaces will focus on simplicity, accessibility, and real-time updates.

4.2 Hardware Interfaces:

Integration with GPS devices for tracking and mobile devices for user interaction.

4.3 Software Interfaces:

Interaction with hospital management systems and healthcare databases.

4.4 Communication Interfaces:

APIs for GPS and real-time communication with external systems.

5. System Requirements

5.1 Functional Requirements:

- FR-01: User login and profile management.
- FR-02: Real-time GPS-based tracking of ambulances.
- FR-03: Automated notifications for bookings and updates.

5.2 Non-Functional Requirements:

Performance: Key operations (e.g., booking, tracking) should respond within 2 seconds.

Security: Encrypt user data with AES-256 encryption.

Usability: User-friendly interface for both technical and non-technical users.

Reliability: 99.9% uptime and efficient recovery mechanisms.

6. Use Case Diagrams

6.1 Use Case 1:User Login and Registration

Illustrates the steps for user registration and login.

6.2 Use Case 2: Ambulance Booking

Describes the booking process from user request to confirmation.

6.3 Use Case 3: Real-time Tracking

Explains how users track ambulances in real-time using GPS.

6.4 Use Case 4: Notifications and Alerts

Depicts the process of sending notifications for status updates.

6.5 Use Case 5: Resource Allocation

Shows how resources like ventilators are tracked and allocated.

7. System Models

7.1 Data Flow Diagram (DFD):

A visual representation of how data flows between users, GPS, and hospital systems.

7.2 Entity-Relationship Diagram (ERD):

Illustrates relationships between users, ambulances, bookings, and hospitals.

7.3 Sequence Diagram:

Depicts interactions between system components for booking and tracking.

7.4 State Chart Diagram:

Shows states like “Booked,” “In Transit,” and “Arrived.”

8. Performance Requirements

Response Time: Should respond to key operations within 2 seconds.

System Uptime:Maintain 99.9% uptime with minimal downtime.

9. Security and Privacy

9.1 Authentication and Authorization:

Role-based access to ensure only authorized personnel can access critical data.

9.2 Data Encryption:

User and booking data will be encrypted both in transit and at rest.

9.3 Privacy Compliance:

Comply with healthcare data privacy regulations (e.g., HIPAA).

10. Maintenance and Support

10.1 System Maintenance:

Regular updates and patches for new features and bug fixes.

10.2 Backup and Recovery:

Daily backups to ensure minimal data loss during failures.

10.3 Monitoring:

Real-time monitoring of system performance and error handling.

11. Appendices

11.1 Glossary of Terms:

Definitions of terms such as GPS, booking, and resources.

11.2 References:

Additional references such as GPS API documentation and healthcare standards.

Distributed System Diagram (PlantUML Code):

puml

@startuml

Actor Client

Node “Load Balancer” {

[Web Server]

[Mobile Server]

}

Node “Data Center 1” {

Database “Primary Database” as DB1 [App Server 1]}

Node “Data Center 2” {

Database “Secondary Database” as DB2 [App Server 2]

}

Client → “Web Server” : Request

Client → “Mobile Server” : Request

“Web Server” → “App Server 1” : API Call

“Mobile Server” → “App Server 2” : API Call

“App Server 1” → DB1 : Read/Write

“App Server 2” → DB2 : Read/Write

DB1 → DB2 : Data Replication

@enduml

Result:

The SRS for the Smart Ambulance Management System was successfully created following the steps described above.

3. Entity Relationship Model

AIM:

To draw the Entity Relationship Diagram for the Ambulance Management System.

ALGORITHM:

1.Mapping of Regular Entity Types

Identify the main entities involved in the system:

- Ambulances
- Drivers
- Patients
- Requests

1.Mapping of Weak Entity Types

Determine if there are any weak entities (e.g., specific ride details may not exist independently without Requests). In this case, there are no weak entities as all identified entities can exist independently.

1. Mapping of Binary 1:1 Relation Types

Identify relationships where each entity is related to only one instance of another entity:

- Each Ambulance is assigned to one Driver (1:1).

2. Mapping of Binary 1: N Relationship Types

Define relationships where one entity can relate to multiple instances of another entity:

- One Driver can handle multiple Requests (1:N).
- One Ambulance can handle multiple Requests (1:N).

3. Mapping of Binary M:N Relationship Types

Define relationships where many instances of one entity can relate to many instances of another entity:

- Patients can generate multiple Requests, and each Request involves a Patient (M:N).

INPUT:

1. Entities:

- Ambulances
- Drivers
- Patients
- Requests

2. Entity Relationship Matrix:

Ambulances ↔ Drivers: 1:1

Each ambulance is assigned to one driver.

Ambulances ↔ Patients: 1:N

one ambulance can serve multiple patients.

Ambulances ↔ Requests: 1:N

one ambulance can handle multiple requests.

Drivers ↔ Patients: 1:N

one driver can serve multiple patients.

Drivers ↔ Requests: 1:N

one driver can be involved in multiple requests.

Patients ↔ Requests: M:N

one patient can generate multiple requests, and a single request involves one patient.

3. Primary Keys:

- Ambulances: `ambulance_id`
- Drivers: `driver_id`
- Patients: `patient_id`
- Requests: `request_id`

4. Attributes:
   - Ambulances:
     - `ambulance_id`
     - `vehicle_number`
     - `type`
     - `status`
   - Drivers:
     - `driver_id`
     - `name`
     - `license_number`
     - `contact_number`
   - Patients:
     - `patient_id`
     - `name`
     - `age`
     - `address`
     - `contact_number`
   - Requests:
     - `request_id`
     - `patient_id` (FK)
     - `ambulance_id` (FK)
     - `driver_id` (FK)
     - `request_date`
     - `pickup_location`
     - `drop_location`

Mapping of Attributes with Entities:

1. Ambulances:

- Attributes: `ambulance_id`, `vehicle_number`, `type`, `status`

2. Drivers:

- Attributes: `driver_id`, `name`, `license_number`, `contact_number`

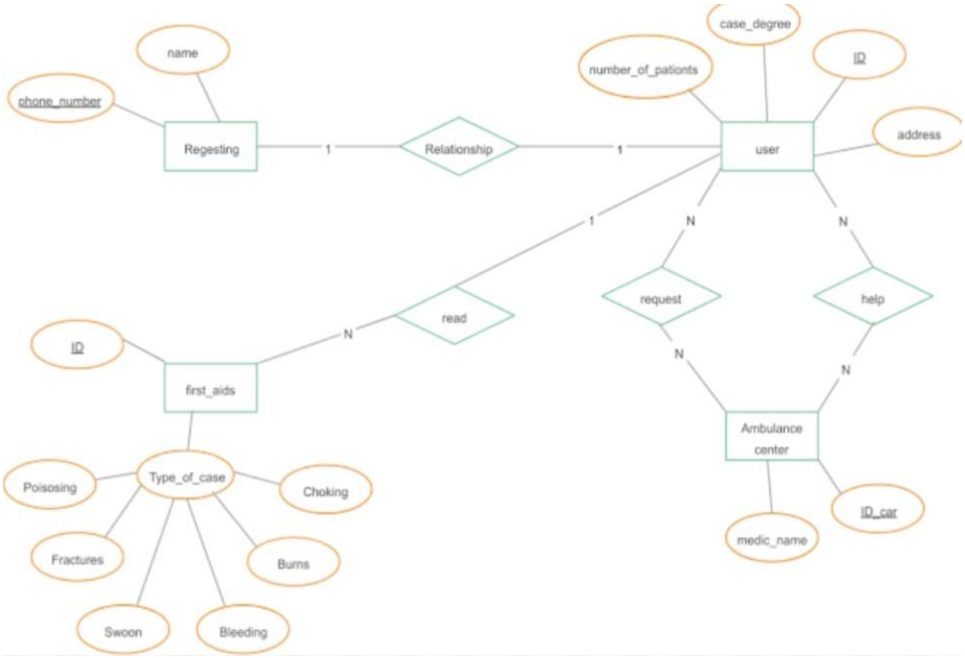
3. Patients:

- Attributes: `patient_id`, `name`, `age`, `address`, `contact_number`

4.Requests:

- Attributes: `request_id`, `patient_id` (FK), `ambulance_id` (FK), `driver_id` (FK), `request_date`, `pickup_location`, `drop_location`

OUTPUT:



RESULT:

The Entity Relationship Diagram for the Ambulance Management System was successfully created by following the steps described above, effectively illustrating the relationships and attributes relevant to the system.

4.Data Flow Diagram

AIM:

To Draw the Data Flow Diagram (DFD) for the Ambulance Management System and List the Modules in the Application.

ALGORITHM:

1. Open Visual Paradigm or Lucidchart:

Use a diagramming tool to create the DFD.

2. Select a DFD Template and name it “Ambulance Management System.”
3. Add External Entities:

Include entities like Patient and Emergency Operator.

4. Add Processes:

Create processes such as:

- Login
- Request Ambulance
- Assign Driver
- Dispatch Ambulance
- Track Ambulance
- Generate Report

5. Add Data Stores:

Include data stores like:

- Ambulance Database
- Driver Database
- Patient Database
- Request Records

6. Connect Entities, Processes, and Data Stores:

Draw data flows like Login Credentials, Ambulance Details, and Request Status.

7. Label Data Flows:

Clearly label the data exchanged between entities, processes, and data stores.

8. Customize:

Adjust colors, fonts, and titles for clarity.

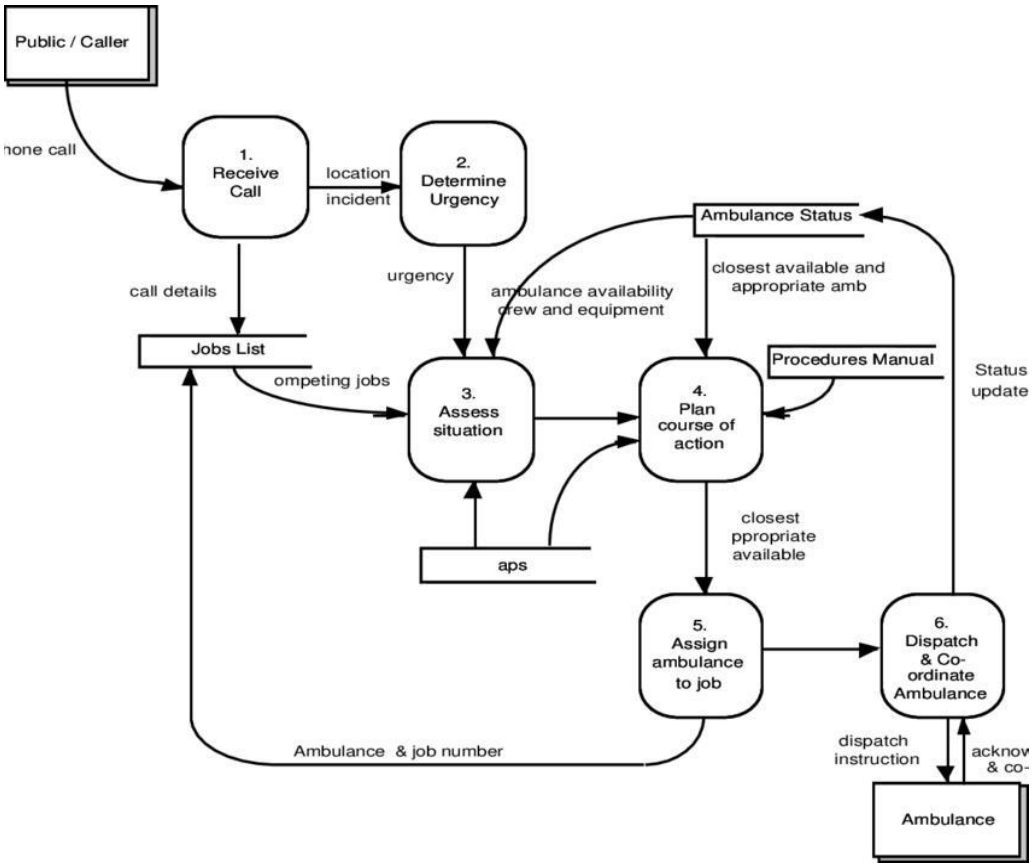
9. Export/Share:

Export or share the completed DFD.

INPUT:

1. Processes:
   - Login
   - Request Ambulance
   - Assign Driver
   - Dispatch Ambulance
   - Track Ambulance
   - Generate Report
2. Data Stores:
   - Ambulance Database
   - Driver Database
   - Patient Database
   - Request Records
3. External Entities:
   - Patient
   - Emergency Operator

Output:



Result: The Data Flow diagram was made successfully by following the steps described above.

5.Use Case Diagram

AIM:

To Draw the Use Case Diagram for the Ambulance Management System.

ALGORITHM:

1.Identify Actors:

- Patient
- Emergency Operator
- Driver

2. Identify Use Cases:

- Login
- Request Ambulance
- Assign Driver
- Dispatch Ambulance
- Track Ambulance
- Generate Report

3. Connect Actors to Use Cases:

- Patient →Login, Request Ambulance
- Emergency Operator → Login, Assign Driver, Dispatch Ambulance, Generate Report
- Driver → Login, Track Ambulance

4. Add System Boundary:

- Label the boundary as "Ambulance Management System."

5. Define Relationships:

- Use include or extend where applicable (e.g., “Request Ambulance” might include “Login”).

6.Review and Refine:

- Ensure all relevant actors and use cases are connected.

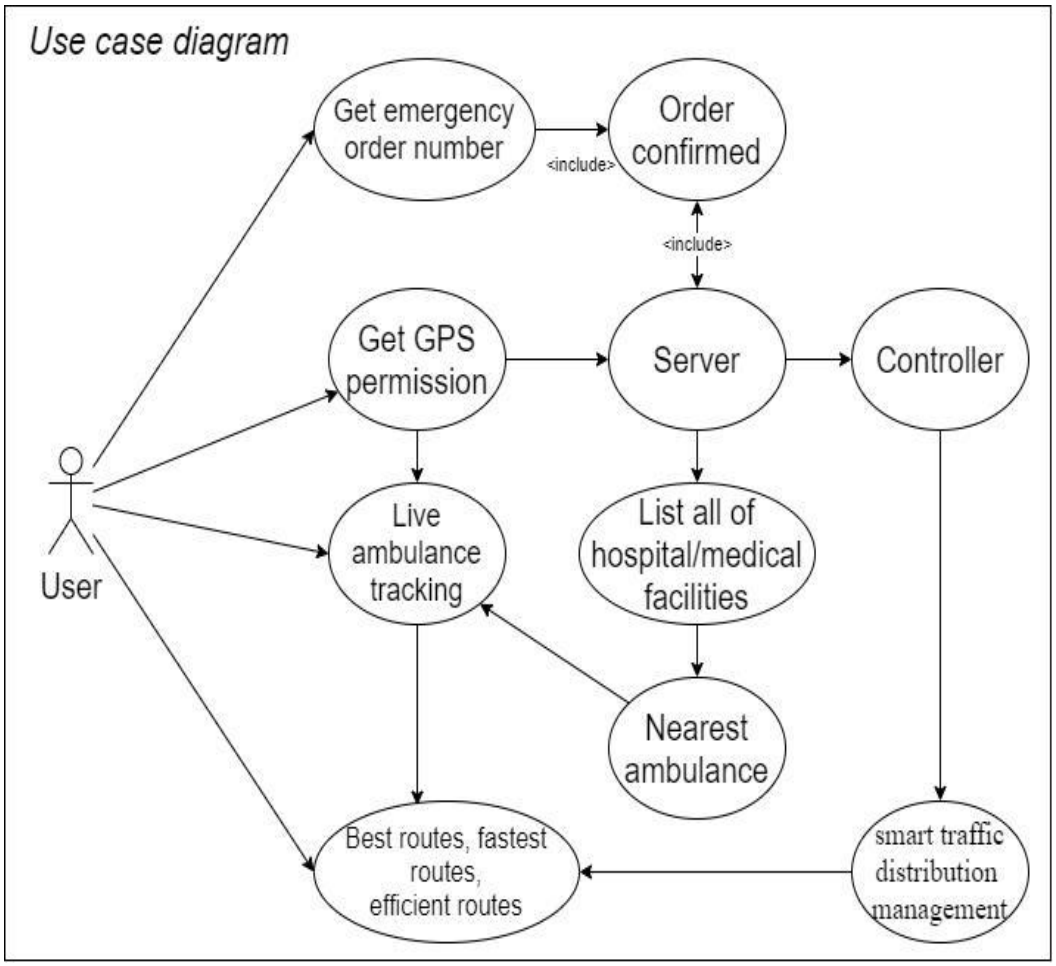
7. Validate Diagram Accuracy:

- Double-check for completeness and correct relationships.

INPUT:

1. Actors:
   - Patient
   - Emergency Operator
   - Driver
2. Use Cases:
   - Login
   - Request Ambulance
   - Assign Driver
   - Dispatch Ambulance
   - Track Ambulance
   - Generate Report
3. Relationships:
   - Patient → Login, Request Ambulance
   - Emergency Operator →Login, Assign Driver, Dispatch Ambulance, Generate Report
   - Driver →Login, Track Ambulance

Output:



Result :

The use case diagram has been created successfully by following the steps given.

6.Activity Diagram

AIM:

To Draw the Activity Diagram for the Ambulance Management System.

ALGORITHM:

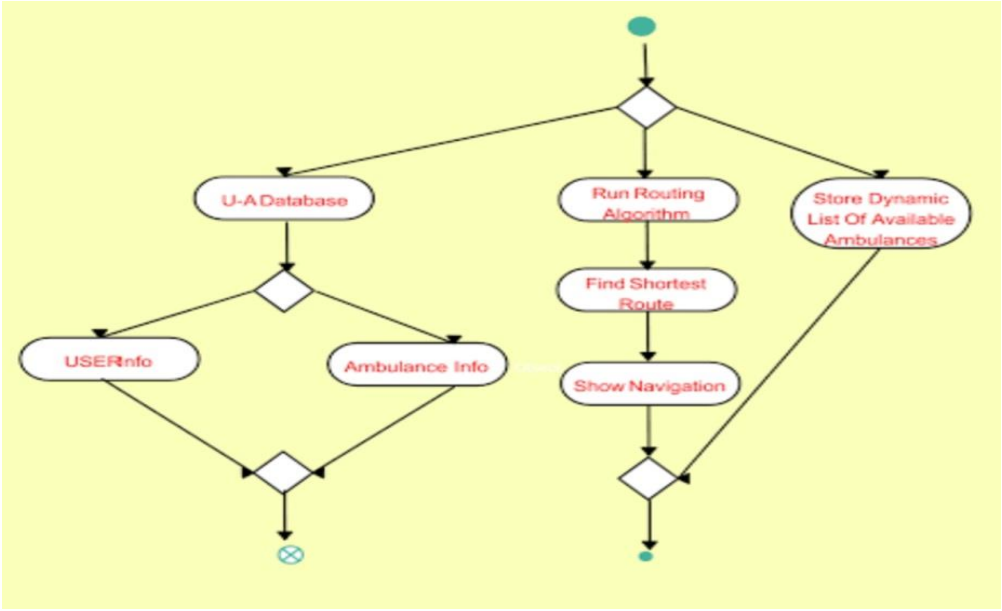
1. Identify Initial and Final States:
   - Initial State: System Idle / User Not Logged In.
   - Final State: Request Completed / User Logs Out.
2. Identify Intermediate Activities:
   - User Logs In
   - Request Ambulance
   - Assign Driver
   - Dispatch Ambulance
   - Track Ambulance
   - Generate Report
3. Identify Conditions or Constraints:
   - Successful Login
   - Ambulance Availability
   - Driver Assignment Confirmation
   - Ambulance Dispatch Confirmation
4. Draw the Diagram:
   - Use ovals for activities (e.g., Login, Assign Driver, Dispatch).
   - Use arrows for transitions between states.
   - Use the initial state symbol (solid circle) for the start and the final state symbol (circle with a border) for the end.

- Add decision points (diamonds) to represent conditional logic, such as checking ambulance availability or login success.

INPUTS:

1. Activities:
   - User Logs In
   - Request Ambulance
   - Assign Driver
   - Dispatch Ambulance
   - Track Ambulance
   - Generate Report
2. Decision Points:
   - Is Login Successful?
   - Is Ambulance Available?
   - Is Driver Assigned?
3. Guards:
   - [Login Successful]
   - [Ambulance Available]
   - [Driver Assigned]
4. Parallel Activities:
   - Assign Driver and Track Ambulance can occur in parallel.
5. Conditions:
   - Login must be successful to proceed to requesting an ambulance.
   - Ambulance must be available to assign a driver.

Output:



Result : The Activity diagram has been created successfully by following the steps given.

7.State Chart Diagram

AIM:

To Draw the State Chart Diagram for the Ambulance Management System.

ALGORITHM:

1. Identify Important Objects:

   - Patient, Ambulance, Driver, System.

2.Identify the States:

   - Idle, Logged In, Requesting Ambulance, Assigning Driver, Dispatching Ambulance, Completing Request.

3. Identify the Events:

   - User Login, Ambulance Request, Driver Assignment, Ambulance Dispatch, Request Completion.

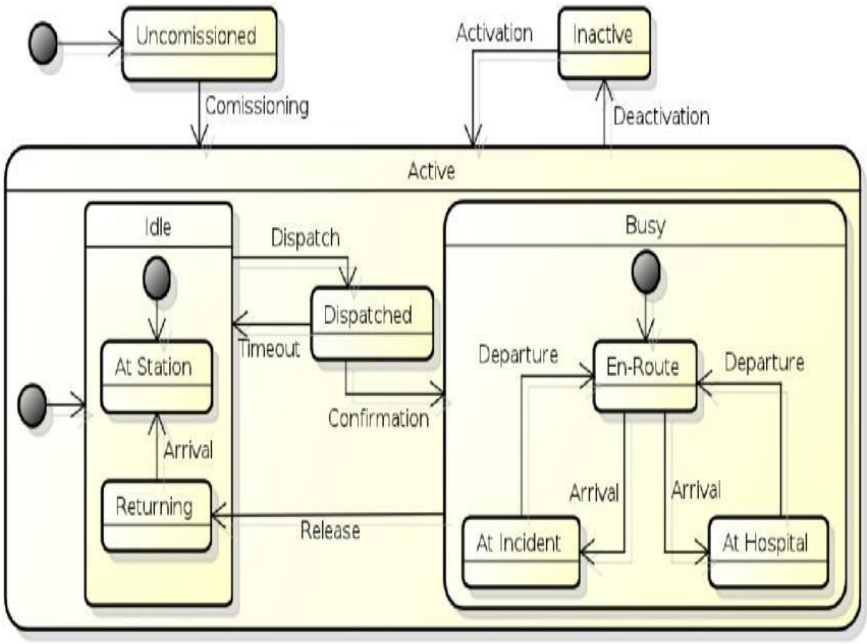
INPUTS:

- Objects:Patient, Ambulance, Driver, System.

- States: Idle, Logged In, Requesting Ambulance, Assigning Driver, Dispatching Ambulance, Completing Request.

- Events:User Login, Ambulance Request, Driver Assignment, Ambulance Dispatch, Request Completion.

Output:



Result : The State Chart diagram has been created successfully by following the steps given.

8.Sequence Diagram

AIM:

To Draw the Sequence Diagram for Ambulance Management System.

ALGORITHM:

1.Identify the Scenario:

- User requesting an ambulance.

2. List the Participants:

- Patient, Emergency Operator, Ambulance System.

3. Define Lifelines:

- Vertical lines for each participant (Patient, Emergency Operator, Ambulance System).

4. Arrange Lifelines:

- Place in order of interaction (Patient, Emergency Operator, Ambulance System).

5. Add Activation Bars:

- Show active periods with vertical bars.

6. Draw Messages:

- Arrows for messages exchanged (e.g., request, confirmation).

7. Include Return Messages:

- Dashed arrows for responses (e.g., success, failure).

8. Indicate Timing and Order:

- Messages in sequential order.

9. Include Conditions and Loops:

- Use brackets for conditional logic (e.g., ambulance availability).

10. Consider Parallel Execution:

- Represent simultaneous actions if needed.

11.Review and Refine:

- Check for accuracy.

12. Add Annotations:

- Clarify complex interactions.

13. Document Assumptions:

- Note assumptions made.

14.Use a Tool:

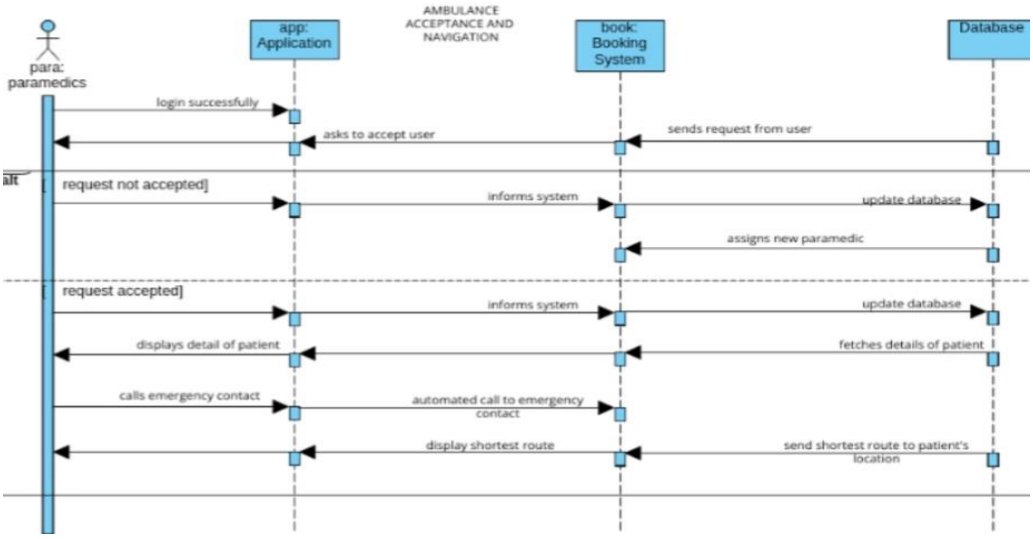
- Create the diagram using software (e.g., Lucidchart).

INPUTS:

- Objects:
  - Patient, Emergency Operator, Ambulance System.
- Message Flows:
  - Patient → Request Ambulance → Emergency Operator
  - Emergency Operator → Check Availability → Ambulance System
  - Ambulance System → Confirm Dispatch → Emergency Operator
  - Emergency Operator → Notify Patient → Patient

- Sequence:
  - Messages flow sequentially from the Patient to the Emergency Operator, then to the Ambulance System, and back with responses.
- Object Organization:
  - Lifelines are organized by the order of interaction, from left to right (Patient, Emergency Operator, Ambulance System).

Output:



Result :

The Sequence diagram has been created successfully by following the steps given.

9.Collaboration Diagram

Aim:

To Draw the Collaboration Diagram for the Ambulance Management System.

Algorithm:

1. Identify Objects/Participants:

   - Patient, Emergency Operator, Ambulance System.

2. Define Interactions:

   - Patient requests ambulance, Emergency Operator processes request, Ambulance System checks availability.

3. Add Messages:

   - “Request Ambulance,” “Check Availability,” “Confirm Availability,” “Dispatch Ambulance.”

4. Consider Relationships:

   - Patient interacts with Emergency Operator, Emergency Operator interacts with Ambulance System.

5. Document the Collaboration Diagram:

   - Show objects and message exchanges with arrows indicating interaction sequence.

Inputs:

- Objects:

   - Patient, Emergency Operator, Ambulance System.

- Message Flows:

   - Patient → Request Ambulance → Emergency Operator

   - Emergency Operator → Check Availability → Ambulance System

   - Ambulance System → Confirm Availability → Emergency Operator

- Emergency Operator → Dispatch Ambulance → Ambulance System

- Ambulance System → Notify Dispatch → Patient.

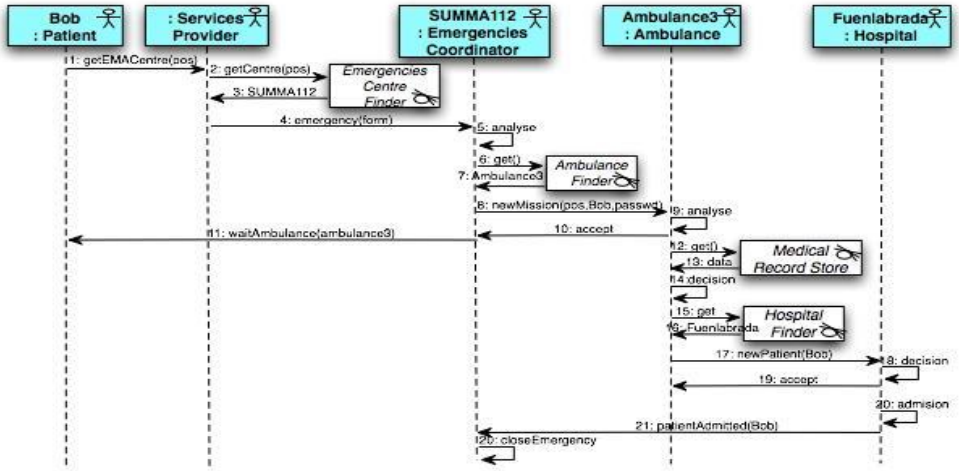
- Sequence:

- Messages flow sequentially from the Patient to the Emergency Operator, then to the Ambulance System and back.

- Object Organization:

- Objects organized from left to right: Patient, Emergency Operator, Ambulance System.

Output:



Result :

The Collaboration diagram has been created successfully by following the steps given.

10.Class Diagram

Aim:

To Draw the Class Diagram for Ambulance Management System.

Algorithm:

1. Identify Classes:

   - Patient, Emergency Operator, Ambulance, Dispatch, System.

2. List Attributes and Methods:

   - Patient: patientID, name, contactInfo; methods: requestAmbulance(), provideLocation().

   - Emergency Operator: operatorID, name; methods: processRequest(), assignAmbulance().

   - Ambulance: ambulanceID, location, availability; methods: checkAvailability(), updateStatus().

   - Dispatch: dispatchID, ambulanceID, time, location; methods: trackAmbulance(), confirmArrival().

3. Identify Relationships:

   - Patient ↔ Dispatch (A patient can request multiple dispatches).

   - Dispatch ↔ Ambulance (Each dispatch involves one ambulance).

   - Emergency Operator ↔ Dispatch (Emergency operator assigns ambulances).

4. Create Class Boxes:

   - Draw separate boxes for each class and include their attributes and methods.

5. Draw Relationships:

   - Connect classes with lines to show relationships (e.g., association).

6. Label Relationships:

   - Label relationship types (e.g., one-to-many, many-to-one).

7. Review and Refine:

- Ensure accuracy and completeness.

8. Use Tools for Digital Drawing:

- Use digital tools like Lucidchart or Visual Paradigm to create the diagram.

Inputs:

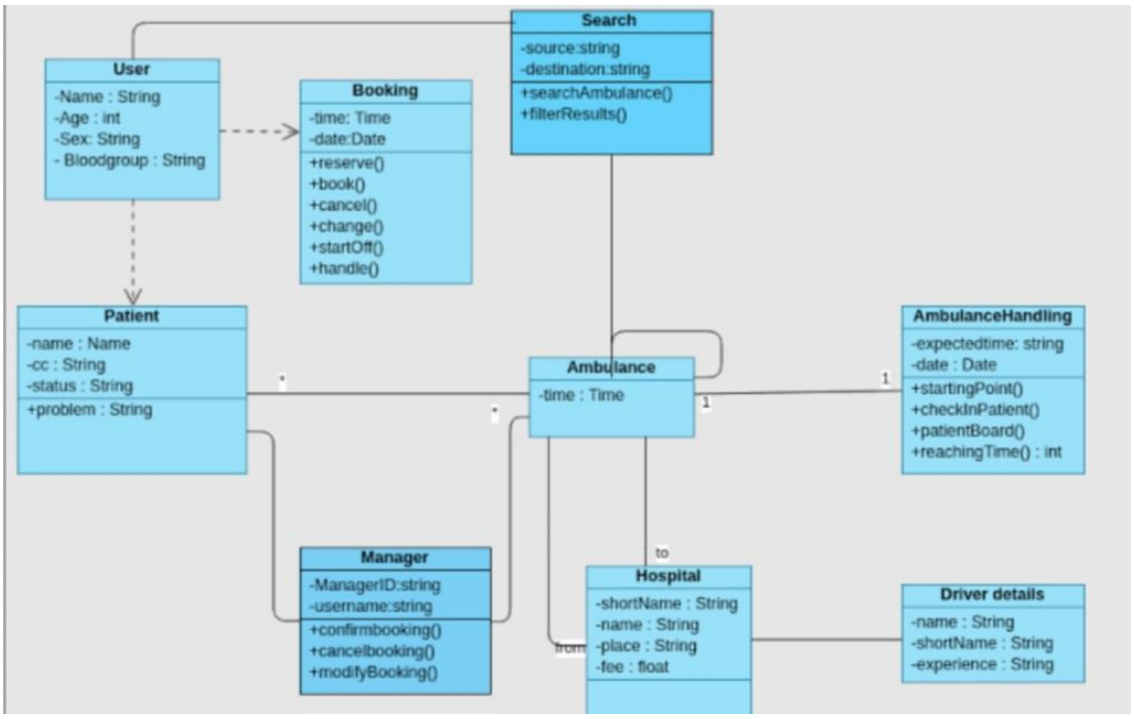
1.Class Name

2. Attributes

3. Methods

4. Visibility Notation

Output:



Result:

The Class diagram has been created successfully by following the steps given.

Code for mini project

```
Import time
Import random
From datetime import datetime

# Simulating a list of ambulances
Ambulances = [
    {“id”: “AMB123”, “location”: “12.9716,77.5946”, “status”: “En Route”, “eta”: 5},
    {“id”: “AMB456”, “location”: “13.0827,80.2707”, “status”: “Available”, “eta”: 0},
]

# Function to send notification
Def send_notification(ambulance_id, message):
    Timestamp = datetime.now().strftime(“%Y-%m-%d %H:%M:%S”)
    Print(f”[{timestamp}] Notification for Ambulance {ambulance_id}: {message}”)

# Function to simulate ambulance tracking and notifications
Def track_ambulances():
    For ambulance in ambulances:
        If ambulance[“status”] == “En Route”:
            Send_notification(
                Ambulance[“id”],
                F”Your ambulance is {ambulance[‘eta’]} minutes away. Current Location: {ambulance[‘location’]}.”
            )
        Elif ambulance[“status”] == “Available”:
```

```
        Send_notification(
            Ambulance[“id”],
            “Ambulance is available for dispatch.”
        )

# Simulate real-time updates
Def simulate_updates():
    While True:
        For ambulance in ambulances:
            # Randomly change ETA for active ambulances
            If ambulance[“status”] == “En Route”:
                Ambulance[“eta”] = max(0, ambulance[“eta”] – random.randint(1, 2))
                If ambulance[“eta”] == 0:
                    Ambulance[“status”] = “Arrived”
                    Send_notification(ambulance[“id”], “Your ambulance has arrived.”)

        # Track ambulances every 10 seconds
        Track_ambulances()
        Time.sleep(10)

# Start the simulation
If __name__ == “__main__”:
    Print(“Starting Notification System…”)
    Simulate_updates()
```

Output:

[2024-11-21 10:15:00] Notification for Ambulance AMB123: Your ambulance is 5 minutes away. Current Location: 12.9716,77.5946.

[2024-11-21 10:15:00] Notification for Ambulance AMB456: Ambulance is available for dispatch.

[2024-11-21 10:15:10] Notification for Ambulance AMB123: Your ambulance is 3 minutes away. Current Location: 12.9716,77.5946.

[2024-11-21 10:15:20] Notification for Ambulance AMB123: Your ambulance has arrived.