REPUBLIC OF CAMEROON

PEACE-WORK-FATHERLAND

MINISTER OF HIGHER EDUCATION

UNIVERSITY OF BUEA

REPUBLIQUE DU CAMEROUN

PAIX-TRAVAIL-PATRIE

MINISTERE DE L'ENSEIGNEMENT SUPERIEURE

UNIVERSITE DE BUEA

EEF 440: Internet Program and Mobile Programming

TASK 4

Group 18

Course Instructor: ***Dr. NKEMENI Valery***

May 2024

Table of content

System Modelling and Design

Introduction

Developing a disaster management mobile application requires careful consideration of system modeling and design principles to ensure it is effective, reliable, and user-friendly. By leveraging modern technologies and adhering to best practices in system modeling and design, the application can significantly contribute to disaster preparedness, response, and recovery efforts.

Disaster management involves a series of activities aimed at minimizing the impact of natural disasters on human life and property. A mobile application for disaster management can play a critical role in achieving this by providing timely information, facilitating communication, and coordinating efforts among various stakeholders. This document outlines the system modeling and design principles for developing a robust and effective disaster management mobile application.

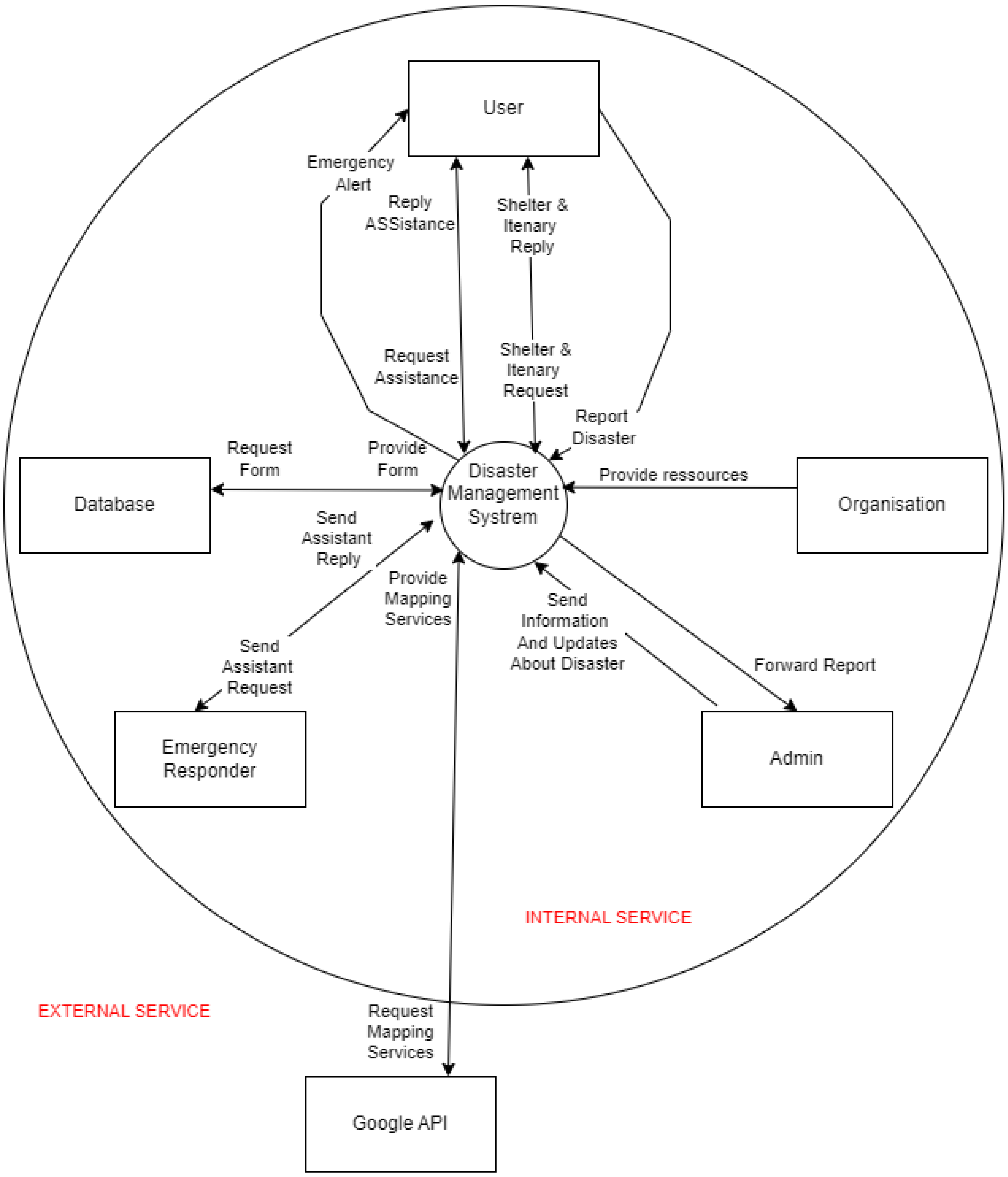
System Modelling

System modeling involves creating abstract representations of a system to understand and analyze its structure and behavior. For a disaster management mobile application, system modeling is essential to ensure all aspects of the application are well-planned and effectively designed. The primary components of system modeling include use case diagrams, class diagrams, sequence diagrams, and deployment diagrams.

I. Context Diagram :

A **Context Diagram** is a high-level, visual representation of a system that shows the system as a single process and its interaction with external entities. It provides an overview of the system's boundary, its interactions with external entities, and the flow of information between them. Context diagrams are often used in the early stages of system design to establish a clear understanding of the system's scope and its environment.

Figure 1 below shows the context diagram for our disaster management mobile application



From the above **Context Diagram** for our Disaster Management System, we represent the system as a central entity and show its interactions with various external and internal entities. The arrows indicate the flow of information or services between the Disaster Management System and these entities.

Explanation of Components

1. **Disaster Management System (DMS)**:
    - The central system that coordinates disaster management activities.
2. **User**:
    - ➢ **Interactions**:
        - **Emergency Alert**: The system sends alerts to users in case of emergencies.
        - **Request Assistance**: Users can request assistance from the system.
        - **Shelter & Itinerary Request**: Users can request information about shelters and itineraries.
        - **Report Disaster**: Users report disasters to the system.
        - **Reply Assistance**: The system replies to user requests for assistance.
        - **Shelter & Itinerary Reply**: The system provides information about shelters and itineraries to users.
3. **Organization**:
    - ➢ **Interactions**:
        - **Provide Resources**: The system requests and receives resources from organizations.
4. **Admin**:
    - ➢ **Interactions**:
        - **Send Information and Updates About Disaster**: The admin sends information and updates about disasters to the system.
        - **Forward Report**: System forwards disaster reports to the admin.
5. **Emergency Responder**:
    - ➢ **Interactions**:
        - **Send Assistant Request**: The system requests assistance from emergency responders.
        - **Send Assistant Reply**: The emergency responders send replies to the system.

6. **Database**:
    - **Interactions**:
        - **Request Form**: The system requests forms from the database.
        - **Provide Form**: The database provides forms to the system.
7. **Google API (External Service)**:
    - **Interactions**:
        - **Request Mapping Services**: The system requests mapping services from Google API.
        - **Provide Mapping Services**: Google API provides mapping services to the system.

Services

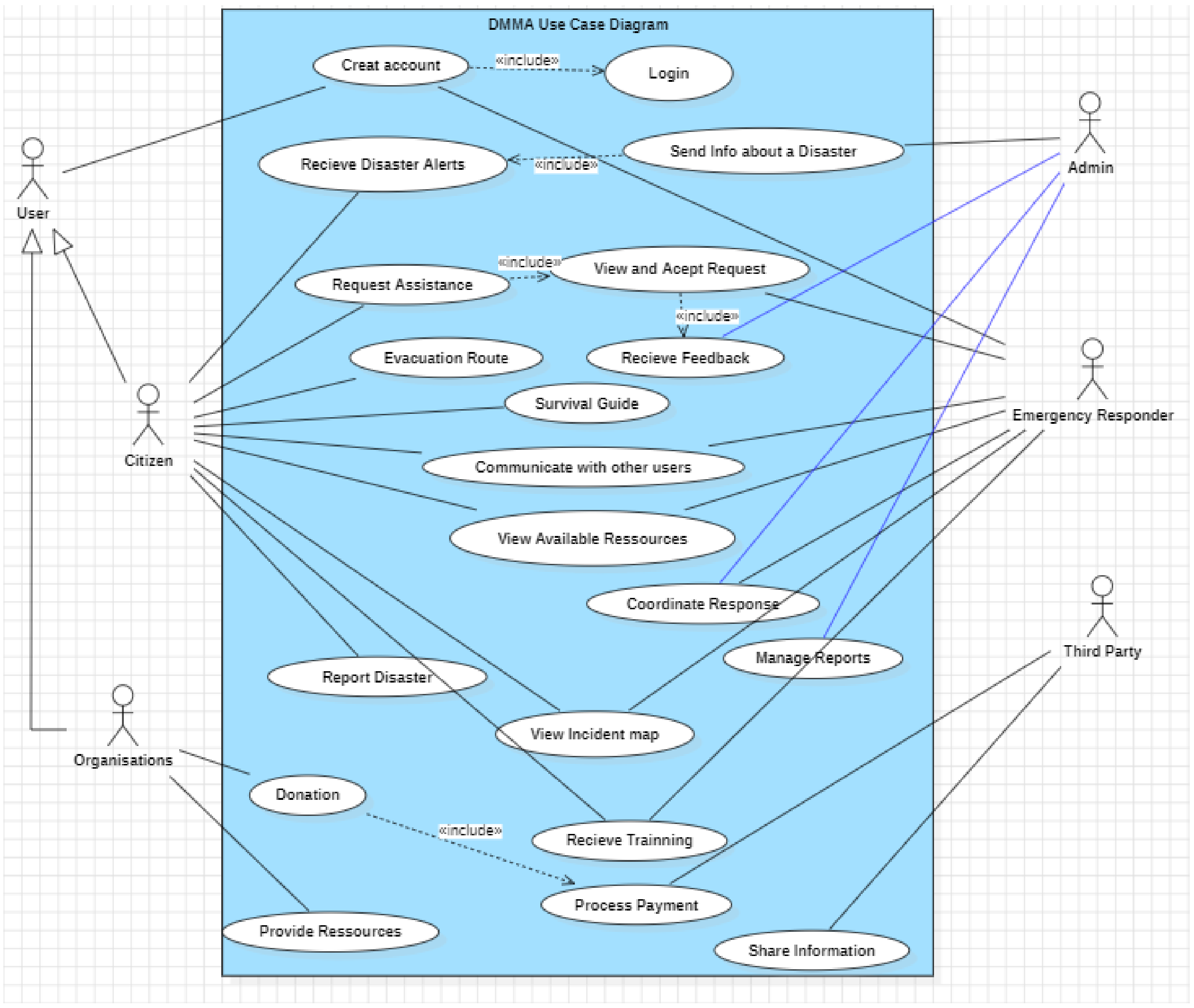
- **Internal Services**: Include interactions with entities within the system's organization, such as Admin, Organization, Database, User, Emergency Responder.
- **External Services**: Include interactions with entities outside the system's direct control, such as Google API for mapping services.
- **Flow of Information**: The arrows depict the direction of data or service flow, ensuring clarity on how the Disaster Management System communicates with various entities.

This context diagram provides a high-level overview of the Disaster Management System's interactions, making it easier to understand the system's scope and boundaries, as well as its external dependencies and internal responsibilities.

II. Use Case Diagrams :

A **Use Case Diagram** is a type of diagram defined by the Unified Modeling Language (UML) that represents the functional requirements of a system. It illustrates the interactions between the system and its external entities (actors), showing the system's functionalities (use cases) from a user's perspective.

Figure 2 below shows the use case diagram for a disaster management mobile application



From the above use case diagram illustrating the various functionalities (use cases) of our system and the interactions between the system and external entities (actors). This specific use case diagram for the Disaster Management Mobile Application (DMMA) includes several actors and their associated use cases.

Actors

1. **User**:
    - General category for all types of users.
2. **Citizen**:
    - Subset of User. Represents regular individuals using the application for personal safety and information.
3. **Admin**:
    - System administrators managing the backend, resources, and overall functionality of the system.
4. **Emergency Responder**:
    - Personnel or teams responding to disaster situations and coordinating relief efforts.
5. **Organizations**:
    - Entities providing resources, donations, and other support during disasters.
6. **Third Party**:
    - External entities that might interact with the system, such as payment processors or information propagators.

Use Cases

1. **Create Account**:
    - Allows users to create a new account by storing the user details in the system database.
    - Includes the **Login** use case, as users need to log in after account creation.
2. **Login**:
    - Enables users to access their accounts. The system verifies the user credentials before giving access to the system
3. **Receive Disaster Alerts**:
    - Users receive notifications about eventual upcoming disasters.

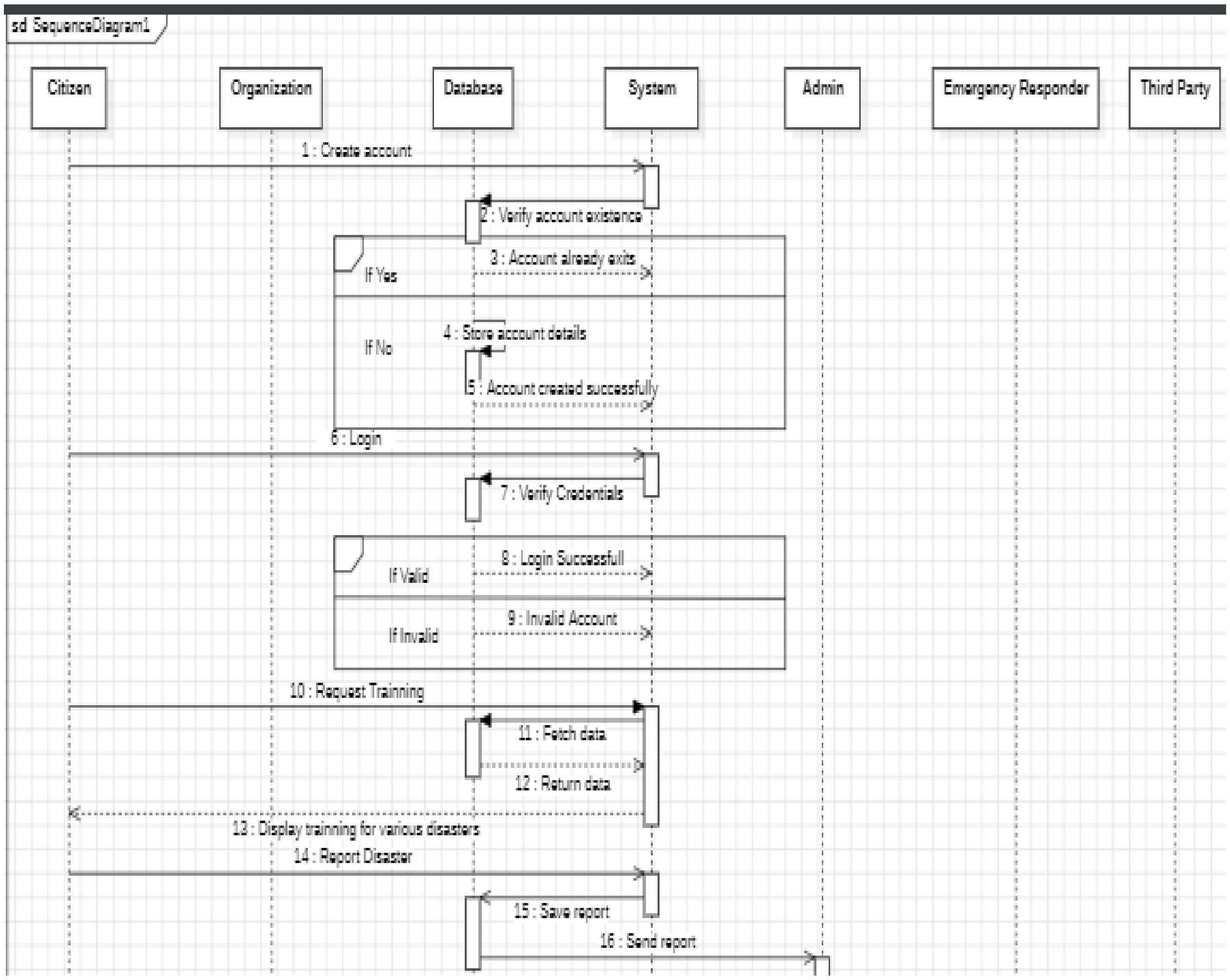
4. **Request Assistance**:
    - Citizens can request help during emergencies.
    - Includes **View and Accept Request** by emergency responders to manage these requests.
5. **Evacuation Route**:
    - Provides information on safe evacuation routes to escape from disasters.
6. **Survival Guide**:
    - Offers guidelines and tips for surviving during disasters.
7. **Communicate with Other Users**:
    - Enables users to communicate with each other for support and information sharing.
8. **View Available Resources**:
    - Allows users to see resources available for disaster management (e.g., shelters, medical aid).
9. **Report Disaster**:
    - Users report incidents or disasters to the system.
10. **Donation**:
    - Organizations can make donations to support disaster recovery.
    - Includes **Process Payment** by third party for handling financial transactions.
11. **Provide Resources**:
    - Organizations contribute resources needed during disasters response.
12. **Send Info About a Disaster**:
    - Admins send information and updates about ongoing disasters.
13. **View and Accept Request**:
    - Emergency responders view and accept assistance requests from citizens.
14. **Receive Feedback**:
    - Admin collect feedback from users to improve services.
15. **Coordinate Response**:
    - Emergency responders coordinate relief efforts.

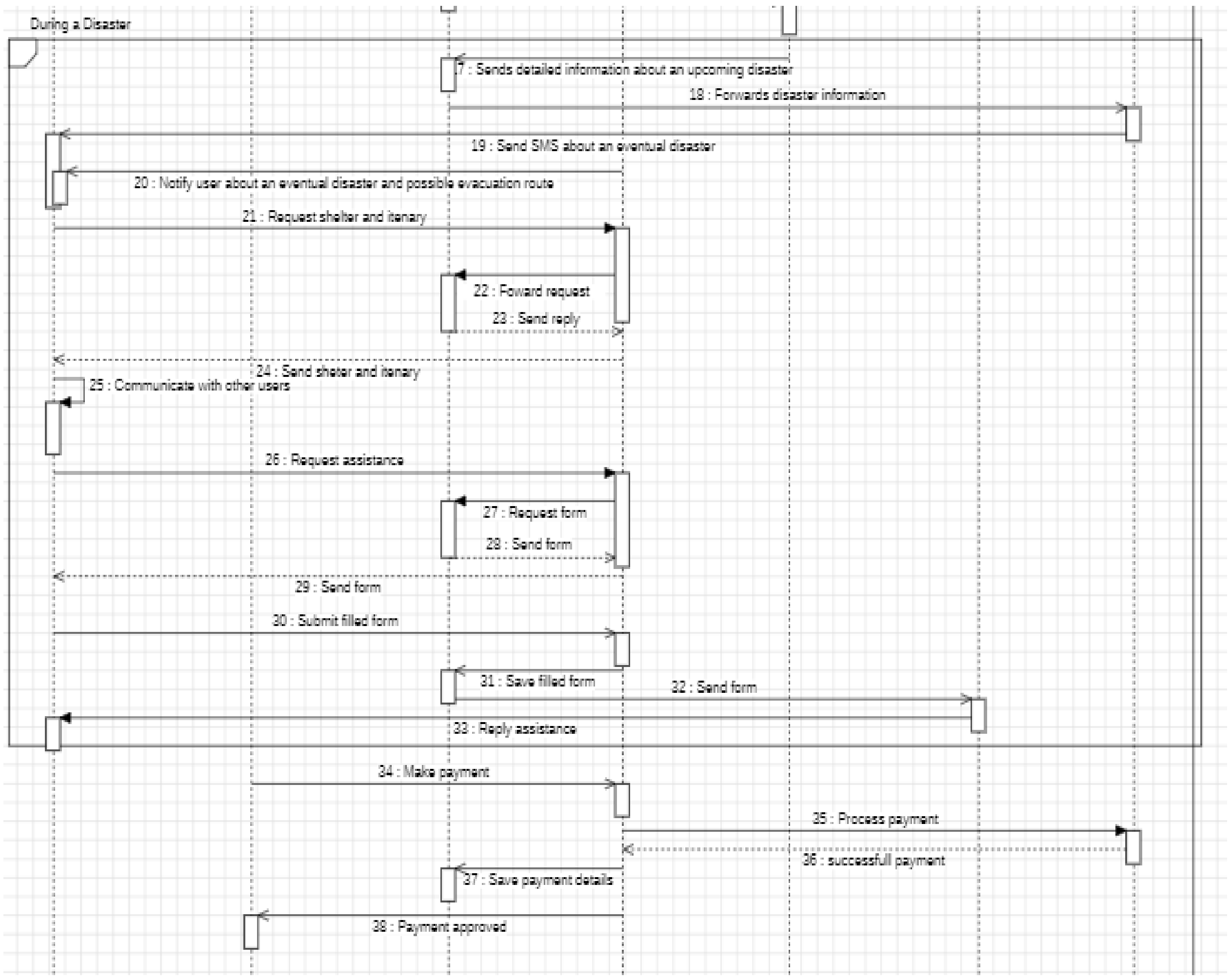
16. **Manage Reports**:
    - Admins manage and track disaster reports send by users.
17. **View Incident Map**:
    - Users can view a map highlighting disaster-affected areas.
18. **Receive Training**:
    - System provides training to users on disaster preparedness.
19. **Process Payment**:
    - Handles payment processing for donations or other financial transactions.
20. **Share Information**:
    - Third parties share relevant information with the system.

III. Sequence Diagram

A **Sequence Diagram** is a type of interaction diagram in the Unified Modeling Language (UML) that shows how objects interact with each other and in what order. It visually represents the sequence of messages exchanged between the objects over time for a specific scenario or use case.

Figure 3 below shows the sequence diagram for a disaster management mobile application


sd SequenceDiagram1
Citizen
Organization
Database
System
Admin
Emergency Responder
Third Party
1 : Create account
2 : Verify account existence
If Yes
3 : Account already exits
If No
4 : Store account details
5 : Account created successfully
6 : Login
7 : Verify Credentials
If Valid
8 : Login Successfull
If Invalid
9 : Invalid Account
10 : Request Trainning
11 : Fetch data
12 : Return data
13 : Display trainning for various disasters
14 : Report Disaster
15 : Save report
16 : Send report


During a Disaster
17 : Sends detailed information about an upcoming disaster
18 : Forwards disaster information
19 : Send SMS about an eventual disaster
20 : Notify user about an eventual disaster and possible evacuation route
21 : Request shelter and itenary
22 : Foward request
23 : Send reply
24 : Send sheter and itenary
25 : Communicate with other users
26 : Request assistance
27 : Request form
28 : Send form
29 : Send form
30 : Submit filled form
31 : Save filled form
32 : Send form
33 : Reply assistance
34 : Make payment
35 : Process payment
36 : successfull payment
37 : Save payment details
38 : Payment approved

From the above sequence diagram for our disaster management system, our diagram effectively illustrates the dynamic behavior of the Disaster Management Mobile Application. It shows how users interact with the system to create accounts, report disasters, request assistance, and how the system coordinates with other entities like organizations and emergency responders.

Entities (Lifelines)

1. **Citizen**: Represents the user or individual using the mobile application.
2. **Organization**: Represents external organizations providing resources or assistance.
3. **Database**: The storage system that holds information related to users, requests, and resources.
4. **System**: The Disaster Management System (DMS) that handles the core operations.
5. **Admin**: System administrators managing and overseeing the operations.
6. **Emergency Responder**: Entities responding to emergencies and providing aid.

7. **Third Party**: External services or entities such as payment processors or additional support services.

Sequence of Interactions

1. **Create Account**:
    - **Citizen → System**: Sends a request to create an account.
    - **System → Database**: Verifies if the account details are unique and creates the account.
    - **Database → System**: Returns the status of the account creation (success or failure).
    - **System → Citizen**: Notifies the citizen of the account creation status.
2. **Login**:
    - **Citizen → System**: Sends login credentials.
    - **System → Database**: Verifies the credentials.
    - **Database → System**: Returns the verification status.
    - **System → Citizen**: Notifies the citizen of the login status.
3. **Request Training**:
    - **Citizen → System**: Requests training information.
    - **System → Database**: Fetches training details.
    - **Database → System**: Returns training information.
    - **System → Citizen**: Provides the training details to the citizen.
4. **Report a Disaster**:
    - **Citizen → System**: Reports a disaster.
    - **System → Admin**: Notifies the admin of the new disaster report.
    - **Admin → Organization**: Forwards the report to relevant organizations.
    - **Admin → Citizen**: Confirms receipt of the report to the citizen.
5. **During a Disaster**:
    - **Admin → Citizen**: Sends detailed information about an ongoing disaster.
    - **Admin → Emergency Responder**: Notifies responders about the disaster.

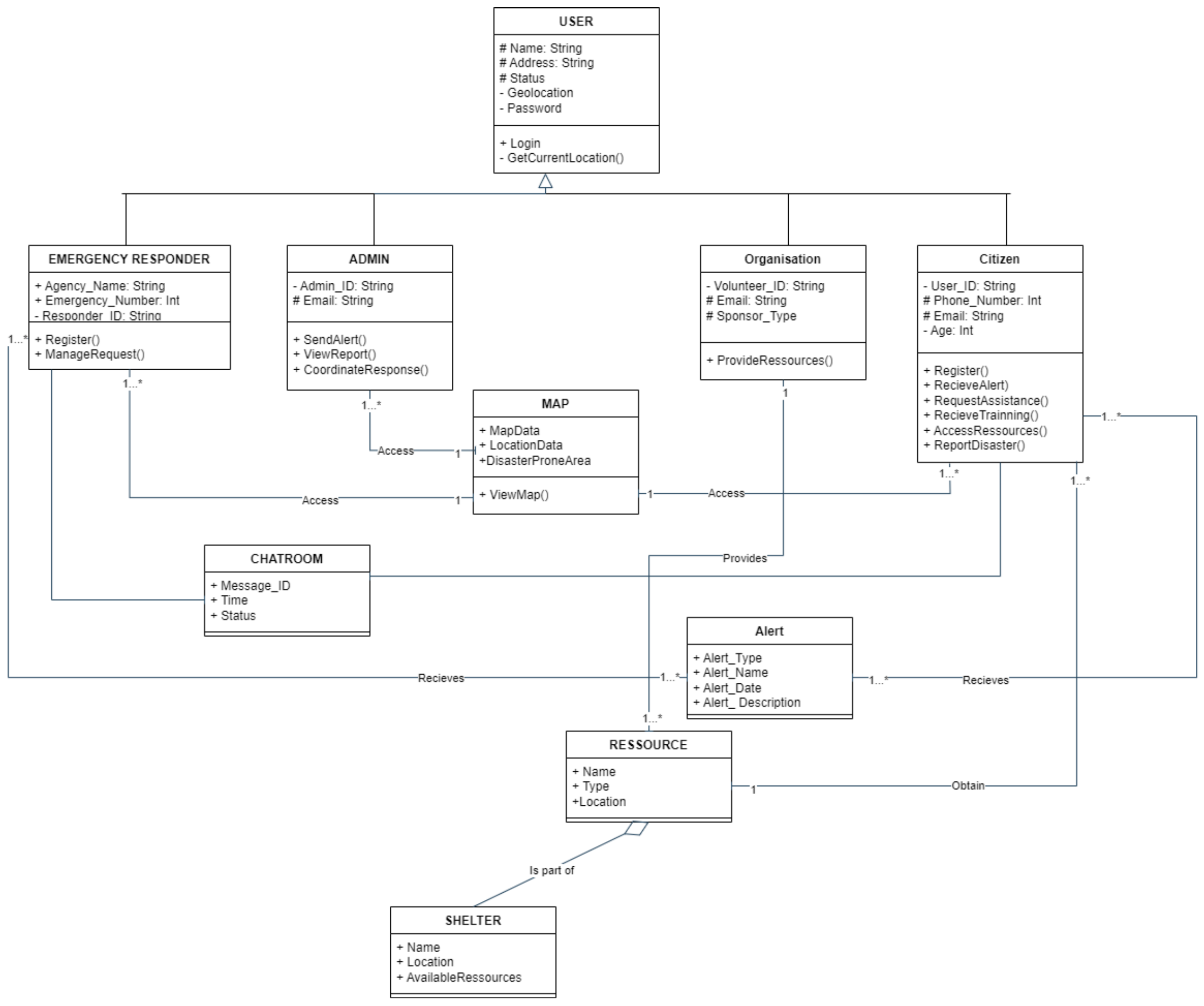
- **Citizen → System**: Requests shelter and itinerary information.
- **System → Database**: Fetches shelter information.
- **Database → System**: Returns shelter data.
- **System → Citizen**: Provides the requested shelter information.

6. **Communication and Assistance**:
    - **Citizen → System**: Communicates with other users.
    - **Citizen → System**: Requests assistance.
    - **System → Database**: Fetches necessary forms or information.
    - **Database → System**: Returns the forms or data.
    - **System → Citizen**: Sends the requested forms to the citizen.
    - **Citizen → System**: Submits filled forms requesting assistance.
    - **System → Emergency Responder**: Forwards the assistance request to emergency responders.
    - **Emergency Responder → System**: Confirms the provision of assistance.
    - **System → Citizen**: Notifies the citizen of the assistance status.
7. **Donation Process**:
    - **Organization → System**: Offers a donation.
    - **System → Third Party**: Processes the payment through an external payment service.
    - **Third Party → System**: Confirms the payment.
    - **System → Organization**: Notifies the organization of the successful donation.

IV. Class Diagrams :

A **Class Diagram** is a type of static structure diagram in the Unified Modeling Language (UML) that describes the structure of a system by showing its classes, their attributes, methods, and the relationships among the objects. Class diagrams are the main building blocks of object-oriented modeling and are used for visualizing, describing, and documenting different aspects of a system.

Figure 4 below shows the class diagram for a disaster management mobile application



The above diagram is a class diagram for a disaster management system, detailing the different entities involved, their attributes, and the relationships between them. Here's a breakdown of each class and their connections:

1. **USER**

   - Attributes: Name, Address, Status, Geolocation, Password.

- Methods: Login(), GetCurrentLocation().
- Relationships:
    - Generalization: USER is a general class for other specific user types such as Emergency Responder, Admin, Organisation, and Citizen.

2. **EMERGENCY RESPONDER**
    - Attributes: Agency_Name, Emergency_Number, Responder_ID.
    - Methods: Register(), ManageRequest().
    - Relationship: Inherits from USER.
3. **ADMIN**
    - Attributes: Admin_ID, Email.
    - Methods: SendAlert(), ViewReport(), CoordinateResponse().
    - Relationship: Inherits from USER.
4. **ORGANISATION**
    - Attributes: Volunteer_ID, Email, Sponsor_Type.
    - Methods: ProvideResources().
    - Relationship: Inherits from USER.
5. **CITIZEN**
    - Attributes: User_ID, Phone_Number, Email, Age.
    - Methods: Register(), ReceiveAlert(), RequestAssistance(), ReceiveTraining(), AccessResources(), ReportDisaster().
    - Relationship: Inherits from USER.
6. **MAP**
    - Attributes: MapData, LocationData, DisasterProneArea.
    - Methods: ViewMap().
    - Relationships:
        - ADMIN accesses MAP.
        - EMERGENCY RESPONDER accesses MAP.
        - CITIZEN accesses MAP.

7. **CHATROOM**
    - Attributes: Message_ID, Time, Status.
    - Relationships:
        - EMERGENCY RESPONDER uses CHATROOM.
        - ADMIN uses CHATROOM.
8. **ALERT**
    - Attributes: Alert_Type, Alert_Name, Alert_Date, Alert_Description.
    - Relationships:
        - ADMIN sends ALERT.
        - CITIZEN receives ALERT.
9. **RESOURCE**
    - Attributes: Name, Type, Location.
    - Relationships:
        - ORGANISATION provides RESOURCE.
        - CITIZEN accesses RESOURCE.
        - SHELTER is part of RESOURCE.
10. **SHELTER**
    - Attributes: Name, Location, AvailableResources.
    - Relationships:
        - SHELTER is part of RESOURCE.

Relationships:

- **Inheritance (Generalization)**
    - USER class is a parent class to EMERGENCY RESPONDER, ADMIN, ORGANISATION, and CITIZEN.
- **Association**
    - ADMIN, EMERGENCY RESPONDER, and CITIZEN have access to the MAP.
    - EMERGENCY RESPONDER and ADMIN interact with the CHATROOM.

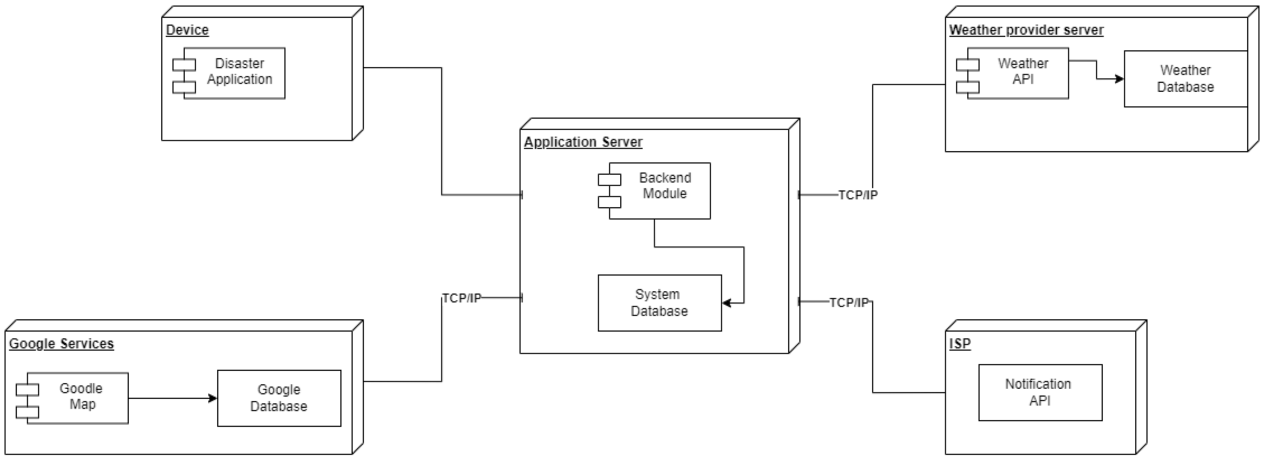
- ADMIN sends ALERTS, which are received by CITIZEN.
- ORGANISATION provides RESOURCES, which are accessed by CITIZEN.
- RESOURCE is part of SHELTER.

This system is designed to ensure coordination and communication during disaster management, enabling different user roles to interact with the necessary resources and information effectively.

V. Deployment Diagram :

A **Deployment Diagram** is a type of diagram in the Unified Modeling Language (UML) that models the physical deployment of artifacts (software or system components) on nodes (hardware or server components). It shows the hardware and software components that make up a system and how they are configured and deployed. Deployment diagrams are essential for understanding the physical architecture and infrastructure of a system, especially in distributed systems.

Figure 5 below shows the class diagram for a disaster management mobile application



The above deployment diagram represents the architecture of a disaster management system, showing the interaction between different components and servers. Here's a breakdown of each component and their roles:

Components

1. **Device**
   - **Disaster Application**: This is the client-side application installed on users' devices (e.g., smartphones, tablets). It interacts with the application server to receive and send data related to disasters.

2. **Application Server**
    - **Backend Module**: The core logic of the system is implemented here. It processes requests from the disaster application and other components.
    - **System Database**: Stores all the data necessary for the disaster management system, including user information, disaster reports, resource data, etc.
    - **Communication**: Uses TCP/IP to communicate with other servers and services.
3. **Weather Provider Server**
    - **Weather API**: Provides weather data to the application server. It is an interface through which the application server can request current weather information and forecasts.
    - **Weather Database**: Stores weather-related data which the Weather API can access.
4. **Google Services**
    - **Google Map**: A mapping service that provides geographic data and mapping functionalities.
    - **Google Database**: Stores geographic and map-related data. The Google Map service retrieves data from this database to provide mapping services.
5. **ISP (Internet Service Provider)**
    - **Notification API**: A service provided by the ISP to send notifications to users. This can be used to send alerts and updates related to disasters.

Communication

- **Device to Application Server**: The disaster application on the user's device communicates with the application server over TCP/IP. This is used to send user data, disaster reports, and receive alerts and updates.
- **Application Server to Weather Provider Server**: The application server communicates with the weather provider server via the Weather API over TCP/IP. This allows the application server to fetch real-time weather data which can be crucial during a disaster.
- **Application Server to Google Services**: The application server communicates with Google Services over TCP/IP to access geographic data through Google Maps. This helps in mapping disaster-prone areas and resources.

- **Application Server to ISP**: The application server communicates with the ISP's Notification API over TCP/IP to send notifications to users. This ensures that users are promptly informed about any updates or alerts related to disasters.

System Design

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It involves translating the logical design of a system into a physical blueprint, which includes both hardware and software specifications. The goal of system design is to create a blueprint that provides the functionality, performance, and scalability required by the system's stakeholders.

I. Hybrid Architecture

1. Client Layer

- **Hybrid Mobile Application**:
    - Developed using React Native or Flutter to ensure cross-platform compatibility.
    - Features include user login, disaster reporting, assistance requests, and resource location.
    - Access to native device features (camera, GPS, push notifications) for a seamless user experience.

2. Backend Layer

- **API Gateway**:
    - Centralized entry point for all client requests.
    - Handles routing, rate limiting, and security (e.g., JWT validation).
- **Microservices**:
    - **User Service**: Manages user registration, authentication, and profiles.
    - **Disaster Management Service**: Handles disaster reports, updates, and notifications.
    - **Resource Management Service**: Manages resource inventories, allocations, and logistics.
    - **Communication Service**: Facilitates communication between users and other users or responders.

3. Data Layer

- **NoSQL Database (e.g., MongoDB)**:
    - Stores unstructured data such as disaster reports, multimedia content, and user feedback.
    - Provides scalability and flexibility in handling large volumes of data.
- **Relational Database (e.g., PostgreSQL)**:
    - Stores structured data, including user accounts, roles, and resource details.
    - Ensures transactional integrity and supports complex queries.

4. Integration Layer

- **External APIs**:
    - **Google Maps API**: Provides location services, geocoding, and route planning.
    - **Payment Gateway API**: Facilitates secure donations and payments.
    - **SMS/Email Gateway**: Ensures notifications are sent to users without internet access.

5. Notification Layer

- **Push Notification Services (e.g., Firebase Cloud Messaging)**:
    - Sends real-time alerts to mobile users about disasters, updates, and assistance requests.
- **SMS/Email Gateway**:
    - Ensures critical notifications reach users who may not have access to the internet during disasters.

6. Security Layer

- **Authentication and Authorization**:
    - Implements secure login mechanisms (e.g., OAuth, JWT).
    - Role-based access control to restrict access to sensitive functionalities.
- **Data Encryption**:
    - Encrypts data both at rest (in databases) and in transit (using TLS/SSL).

Conclusion

The UML diagrams and the hybrid architecture provide a comprehensive blueprint for developing a disaster management mobile application. By following this structured approach, we are well-positioned to develop a disaster management mobile application that significantly enhances disaster response and management capabilities, ultimately saving lives and improving resilience in disaster-prone areas.