

**FACULTY ENGINEERING**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF COMPUTER ENGINEERING, SE**

**CEF440: Internet and Mobile Programming**



**Presented by:**

| BOBGALA HARRISON DIMA | FE21A151 |
| --- | --- |
| KAMCHE YANN ARNAUD | **FE21A208** |
| MESSI II INNOCENT ROSTAND | **FE21A236** |
| NYANDO ONONGWENE | **FE21A290** |
| NKWI CYRIL AKINIMBOM | **FE21A281** |



**Table of Content**

[1. Introduction 2](#_bcslfkycl714)

[2. Context Diagram 3](#_299iefckpwku)

[Description: 3](#_mdv2agyxbs7c)

[Necessary Information: 3](#_glcdwbx0k7gm)

[Description of the Context Diagram 4](#_1d8yvin74gu)

[3. Use Case Diagram 5](#_493shep21mdc)

[4. Sequence Diagram 11](#_3rdcrjn)

[5. Class Diagram 12](#_b5814h4h9h0r)

[5.1. Class Diagram Overview 12](#_cdby8h67ck47)

[5.2. Classes, Attributes and Methods 12](#_ar3uhn7do1dx)

[5.3. Relationships 14](#_lzqgzqidkn3x)

[5.4. Visual Representation 15](#_txux6c3cjih8)

[6. Deployment Diagram 16](#_cx4ntnt3p1gt)

[**7. Conclusion 18**](#_5l38zv5ab5b9)

# **Introduction**

System modeling and design is a critical phase in the software development life cycle that involves creating abstract representations of the system to be developed. This phase typically follows the requirements analysis stage and precedes the implementation stage.

System Modeling and Design is the process of creating models to visualize, specify, construct, and document the structure and behavior of a software system. It involves various types of diagrams and models that represent different aspects of the system, including its architecture, components, interactions, and physical deployment.

The purpose of the system modeling and design stage is multifaceted:

1. Visualization:

To provide a visual representation of the system, making it easier for stakeholders to understand complex structures and processes. The outcome is clear and comprehensible diagrams that illustrate how the system will function and how its components will interact.

1. Specification:

To precisely define the components, their relationships, and their interactions within the system. Detailed specifications that guide developers during the implementation phase and ensure consistency and accuracy is the outcome.

1. Analysis:

To analyze different design alternatives and their implications on system performance, scalability, and maintainability.

1. Communication:

To facilitate communication among stakeholders, including developers, designers, clients, and end-users.

1. Documentation:

To create comprehensive documentation that serves as a reference throughout the development process and for future maintenance.

1. Validation:

To validate the system design against the requirements to ensure that all functional and non-functional requirements are met. The outcome is a validated design that aligns with the specified requirements and expectations of stakeholders.

1. Planning:

To aid in project planning by providing a clear outline of system components and their interactions, which helps in estimating time and resources needed for development.

**Key Activities in System Modeling and Design:**

* Creating Use Case Diagrams: To capture the functional requirements and interactions between the system and external actors.
* Developing Class Diagrams: To define the static structure of the system, including classes, attributes, methods, and relationships.
* Designing Sequence Diagrams: To illustrate the flow of messages and interactions between objects in specific scenarios.
* Constructing Deployment Diagrams: To show the physical deployment of system components on hardware nodes.
* Generating State Diagrams: To represent the state changes of various system components.
* Building Activity Diagrams: To detail the workflows and processes within the system.

# **Context Diagram**

### Description:

A diagram used to give an overview of an entire system. In a context diagram there is only one circle / process that represents the entire system. The purpose of this diagram is to display the expecting inputs and outputs from the system to and from various external entities. Through this display a system analyst can model what expected data is going to go into the system, then after it has been processed by the system, what information will be returned to the external entities.

### Necessary Information:

* **System**: Mobile-Based Disaster Management System
* **External Entities**:
* Users (General Public, Volunteers)
* Emergency Services (Police, Fire Department, Medical Services)
* Government Agencies
* Weather Data Providers
* Communication Networks
* **Interactions**:
* Users report incidents and receive alerts.
* Emergency services receive incident reports and provide response updates.
* Government agencies receive situation reports and provide directives.
* Weather data providers supply real-time weather information.
* Communication networks facilitate data transmission.

### Description of the Context Diagram

1. **Mobile-Based Disaster Management System**: The central system that facilitates disaster management through a mobile application, enabling various interactions with external entities.
2. **Users (General Public, Volunteers):**

* Interactions:
* Report incidents to the system.
* Receive alerts and updates from the system about ongoing or potential disasters.
* Purpose: Users provide real-time data about disaster events and receive crucial information to ensure their safety.

1. **Emergency Services (Police, Fire Department, Medical Services):**

* Interactions:
* Receive incident reports from the system, allowing them to respond promptly.
* Provide updates on their response actions back to the system.
* Purpose: Emergency services use the system to coordinate their disaster response activities effectively.

1. **Government Agencies:**

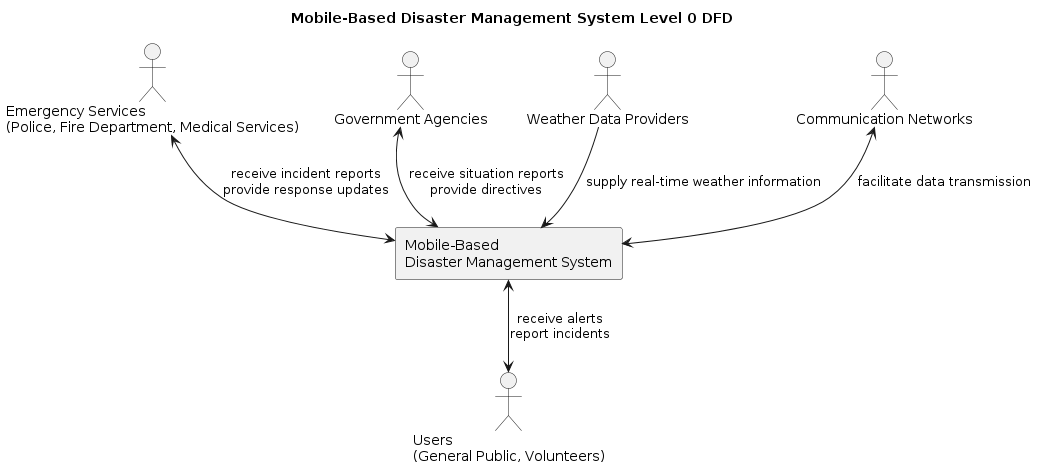
* Interactions:
* Receive situation reports from the system to stay informed about the disaster status.
* Provide directives and guidelines to the system to manage the disaster response.
* Purpose: Government agencies oversee disaster management strategies and provide necessary directives based on the situational reports received.

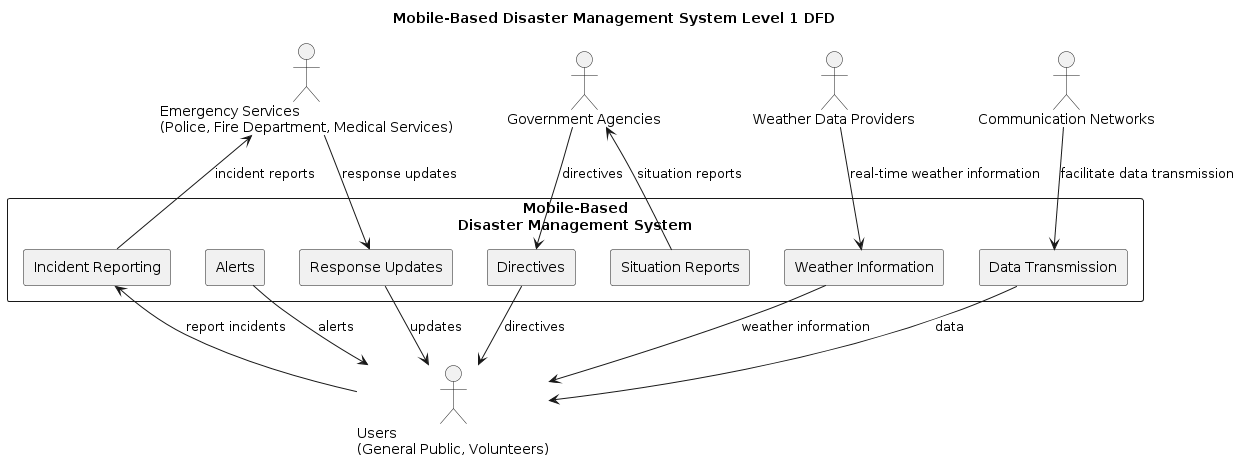
1. **Weather Data Providers:**

* Interactions:
* Supply the system with real-time weather information.
* Purpose: Real-time weather data is crucial for the system to predict and manage weather-related disasters accurately.

1. **Communication Networks:**

* Interactions:
* Facilitate data transmission between the system and other external entities.
* Purpose: Ensure reliable communication channels for data exchange during disaster situations.





# **Use Case Diagram**

This UML diagram outlines the structure and interactions of an Emergency Management System, featuring four main actors:

**Actors:**

* **General Public:** Individuals who can report incidents and receive alerts.
* **Volunteers:** People who can offer help and receive alerts.
* **Emergency Responders (Police, Fire Department, Medical Services) :** Personnel who provide emergency responses and receive alerts.
* **Government Officials:** Authorities who issue directives and receive alerts.
* **Weather Data Provider:** Entities that supply weather data.

**System Components**

* **Mobile App**: Used by the general public and volunteers to report incidents, volunteer for help, and receive alerts.
* **Server:** Central component that manages interactions between other systems and the database.
* **Emergency Services System:** Secure network for emergency responders.
* **Emergency Response Application:** Application used by emergency responders.
* **Government Agency System:** Secure network for government officials.
* **Weather Data Provider System**: API used to supply weather data.
* **Database**: Stores data for the server to access and provide to other components.

**Data Flow and Connections**

1. Incident Reporting:

* Incidents are reported through the mobile app by the general public.
* The mobile app communicates with the server to relay this information.

2. Emergency Response:

* The server notifies the emergency services system about reported incidents.
* The emergency services system informs emergency responders via a secure network.
* Emergency responders use the emergency response application to coordinate responses, which connects to the server.

3. Alerts:

* The server sends alerts to the general public, volunteers, emergency responders, and government officials.

4. Volunteering:

* Volunteers use the mobile app to offer help.
* Data from volunteers is managed by the server and stored in the database.

5. Government Directives:

* Government officials issue directives through the government agency system.
* The government agency system communicates with the server.
* If the directives are for the general public, the server sends notifications via the mobile app.

6. Weather Data:

* Weather data is supplied via an API from the weather data provider system to the server.
* The server stores the weather data in the database for use by other components.

**Use Cases:**

* **Incident Reporting :**

**Actor:** General Public

**Description:** User reports a disaster event through the mobile app. **Precondition:** User has a registered account and location services enabled.

**Basic Flow:**

1. *User selects "Report Incident" on the app.*
2. *User chooses the type of disaster (e.g., earthquake, flood, fire).*
3. *User enters details like location description, severity, and any pictures/videos (optional).*
4. *User submits the report.*

**Alternative Flow 1:** User chooses to report anonymously (limited details captured).

**Alternative Flow 2:** User encounters network issues. System prompts user to retry or save report as draft.

**Postcondition:** System validates and transmits the incident report to the server. User receives confirmation and may be prompted for additional information.

* **Alert Notification**

**Actor:** General Public, Volunteers, Emergency Responders, Government Officials

**Description:** Actor receives alerts and updates regarding ongoing or potential disasters.

**Precondition:** Actor has opted-in for alerts (customizable based on location and type).

**Basic Flow:**

1. *System receives updates on disaster situations from various sources (reported incidents, weather data).*
2. *System generates targeted alerts based on actor's profile and location.*
3. *Alert is delivered to the actor's mobile device via push notification or SMS (if network allows).*

**Alternative Flow 1:** User can choose to adjust notification preferences within the app settings.

**Postcondition:** Actor is informed about the disaster and can take necessary precautions.

* **Volunteering:**

**Actor:** Volunteers

**Description:** Volunteer registers their availability and skills to assist during disasters.

**Precondition:** Volunteer has a registered account and completes a volunteer profile. **Basic Flow:**

1. *Volunteer logs in to the app and selects "Volunteer for Help."*
2. *Volunteer provides information on skills (e.g., first aid, search & rescue) and availability.*
3. *Volunteer submits the information.*

**Alternative Flow 1**: Volunteers can update their skills and availability at any time.

**Postcondition:** System stores volunteer information and may notify them of relevant opportunities based on their profile and ongoing incidents.

* **Emergency Response Providence:**

**Actor:** Emergency Responders

**Description:** Emergency responders receive and respond to incident reports, updating the system on their actions.

**Precondition:** Responder has a dedicated emergency response application with access to incident details.

**Basic Flow:**

1. *Emergency services receive notification of a new incident through their system.*
2. *Responder views details, location, and potential hazards on a map.*
3. *Responder confirms they are responding and dispatches resources as needed.*
4. *Responder updates the system on their progress and any critical information.*

**Alternative Flow 1**: Multiple responders may be assigned to an incident. System facilitates coordination.

**Alternative Flow 2:** Responder may request additional resources or support through the system.

**Postcondition:** System maintains a record of response activities and updates all relevant parties.

* **Issuing Directives :**

**Actor:** Government Officials

**Description:** Authorized officials issue directives and guidelines for disaster response through a secure web interface.

**Precondition:** Official has logged in to the government agency system with proper credentials.

**Basic Flow:**

1. *Official selects "Issue Directives" and chooses the target audience (e.g., public, responders).*
2. *Official drafts the directive message and defines its urgency and dissemination channels.*
3. *Official submits the directive for approval by designated authority (if applicable).*

**Alternative Flow 1:** System may offer pre-defined templates for common directives.

**Postcondition:** Approved directives are sent to the target audience via the mobile app or other communication channels.

* **Weather Data Suppliance :**

**Actor:** Weather Data Provider

**Description:** Provider transmits real-time weather data to the system through a secure API.

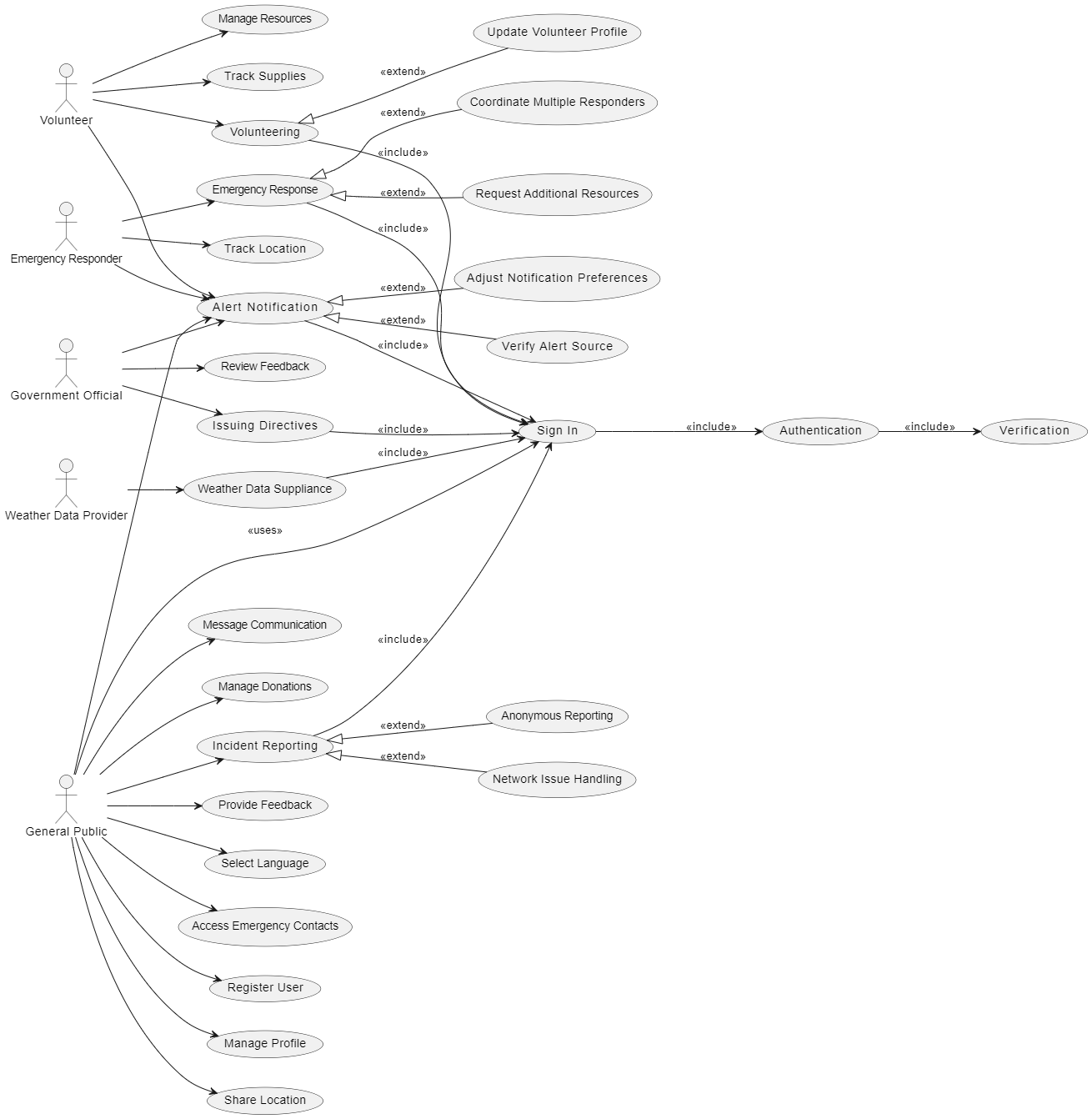
**Precondition:** Provider has established a connection with the system's API.

**Basic Flow:**

1. *Weather data provider gathers weather data from various sources (e.g., weather stations, satellites).*
2. *Provider formats the data according to a pre-defined schema established by the system.*
3. *Provider transmits the formatted data securely through the system's API using HTTPS protocol.*
4. *The system receives the data and validates its authenticity and integrity.*
5. *Upon successful validation, the system parses and stores the weather data in a designated database.*
6. *The system may utilize the received data in various ways:* 
   1. *Update real-time weather information displayed within the mobile app for public awareness.*
   2. *Analyze weather patterns to predict potential disaster risks (e.g., flooding due to heavy rain).*
   3. *Trigger alerts for specific locations based on critical weather events.*

**Alternative Flow 1:** The system may encounter issues receiving or validating the data due to network problems or data format inconsistencies. In such cases, the system should notify the weather data provider and attempt to re-establish connection or request data correction.

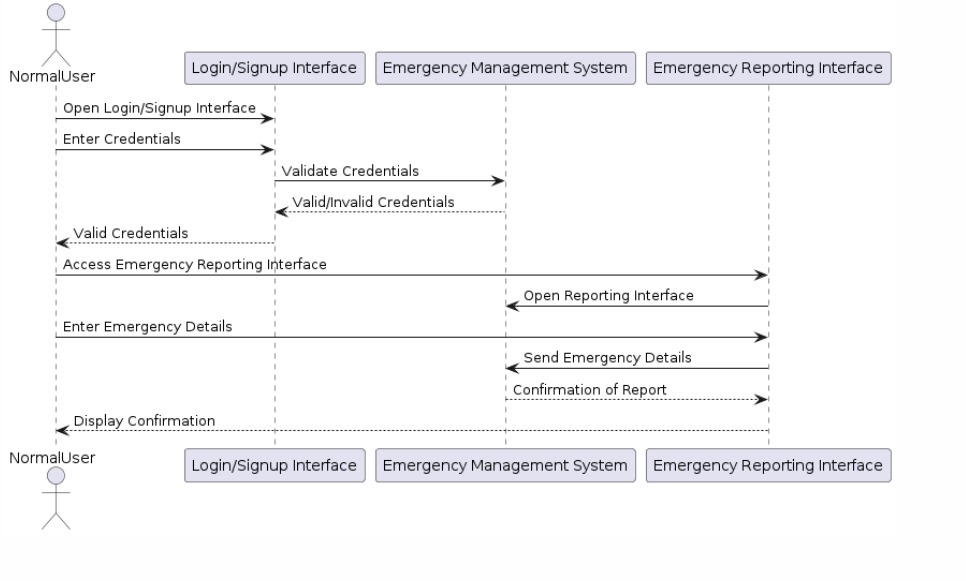
**Postcondition:** The system successfully integrates the received weather data for disaster management purposes. The weather data provider continues to transmit updates at regular intervals or as weather conditions change significantly.



# **Sequence Diagram**

A sequence diagram shows how objects interact in a particular scenario of a use case, focusing on the sequence of messages exchanged.

* **Scenario**: Reporting an Incident
* Objects:
  + User (General Public)
  + Mobile App
  + Server
  + Emergency Services
* Messages:
  + User -> Mobile App: Submit Incident Report
  + Mobile App -> Server: Forward Incident Report
  + Server -> Emergency Services: Notify Incident
  + Emergency Services -> Server: Acknowledge Notification
  + Server -> Mobile App: Confirm Report Submission
  + Mobile App -> User: Incident Report Confirmation



*Sequence Diagram for Disaster Management System. Scenario: Reporting an Incident*

#### **Actors:**

* Normal User (General Public)
* Emergency Management System

#### **Objects:**

* Login/Signup Interface
* Emergency Reporting Interface

### Steps:

1. **User Accesses the System**:
   * **User** opens the **Login/Signup Interface**.
2. **Enter Credentials**:
   * **User** enters their login credentials (username and password) into the **Login/Signup Interface**.
3. **Validate Credentials**:
   * **Login/Signup Interface** sends the entered credentials to the **Emergency Management System** for validation.
4. **Credentials Valid/Invalid Response**:
   * **Emergency Management System** checks the credentials and sends back a response indicating if they are valid or invalid.
5. **Valid Credentials**:
   * If credentials are valid, the **Login/Signup Interface** confirms successful login to the **User**.
6. **Access Emergency Reporting Interface**:
   * **User** opens the **Emergency Reporting Interface**.
7. **Open Reporting Interface**:
   * **Emergency Management System** processes the request and opens the **Emergency Reporting Interface**.
8. **Enter Emergency Details**:
   * **User** enters details of the emergency (e.g., type, location, severity) into the **Emergency Reporting Interface**.
9. **Send Emergency Details**:
   * **Emergency Reporting Interface** sends the emergency details to the **Emergency Management System**.
10. **Confirmation of Report**:
    * **Emergency Management System** processes the details and sends a confirmation back to the **Emergency Reporting Interface**.
11. **Display Confirmation**:
    * **Emergency Reporting Interface** displays the confirmation message to the **User**.

### Sequence of Interactions:

1. **Open Login/Signup Interface**:
   * **User** opens the **Login/Signup Interface**. This is a synchronous action requiring immediate feedback.
2. **Enter Credentials**:
   * **User** enters their credentials into the **Login/Signup Interface**. This is also synchronous.
3. **Validate Credentials**:
   * **Login/Signup Interface** sends credentials to the **Emergency Management System** for validation. This is synchronous.
4. **Response: Valid/Invalid Credentials**:
   * **Emergency Management System** validates and returns whether credentials are valid or invalid. This is synchronous.
5. **Valid Credentials**:
   * If valid, **Login/Signup Interface** confirms successful login to the **User**. This is asynchronous.
6. **Access Emergency Reporting Interface**:
   * **User** opens the **Emergency Reporting Interface**. This is synchronous.
7. **Open Reporting Interface**:
   * **Emergency Management System** processes and opens the **Emergency Reporting Interface**. This is asynchronous.
8. **Enter Emergency Details**:
   * **User** enters emergency details into the **Emergency Reporting Interface**. This is synchronous.
9. **Send Emergency Details**:
   * **Emergency Reporting Interface** sends details to the **Emergency Management System**. This is synchronous.
10. **Confirmation of Report**:
    * **Emergency Management System** processes and confirms the report. This is asynchronous.
11. **Display Confirmation**:
    * **Emergency Reporting Interface** displays confirmation to the **User**. This is asynchronous.

# **Class Diagram**

A class diagram for a disaster management system is a type of static structure diagram in the Unified Modeling Language (UML) that describes the system by showing its classes, their attributes, methods, and the relationships among the objects. The diagram serves as a blueprint for the structure of the system, detailing how various components interact and function together. Here's a detailed look at the class diagram for a disaster management system, based on the previously provided information.

## Class Diagram Overview

The class diagram for the disaster management application includes the following entities

1. User
2. Incident
3. Volunteer
4. EmergencyService
5. EmergencyContacts
6. GovernmentAgency
7. WeatherDataProvider
8. AlertSystem

Each class includes specific attributes and methods that define its responsibilities within the system. Below is a detailed description of each class and its components.

## Classes, Attributes and Methods

1. **User**

The user represents any individual who interacts with the system.

* **Attributes:**
  + **userId**: Unique identifier for a user.
  + **Name**: Name of the user.
  + **contactInfo**: Contact Information of the user
* **Methods:**
  + **reportIncident():** Allow the user to report a new incident.
  + **receiveAlert():** Allows the user to receive alerts about incidents.

1. **Incident**

The Incident class captures information about reported incidents.

* **Attributes:**
  + **incidentId**: Unique identifier for an incident.
  + **location**: Location of the incident
  + **type**: Type of disaster like a flood or fire incident.
  + **description**: Detailed description of the incident.
  + **timestamp**: Timestamp when the incident was reported.
* **Methods:**
  + **getDetails():** Retrieves the details of the incident.
  + **updateStatus():** Updates the status of the incident.

1. **Volunteer**

The Volunteer class represents individuals who offer help during incidents.

* **Attributes:**
  + **volunteerId**: Unique identifier for the volunteer.
  + **skills**: Skills that the volunteer possesses.
  + **availability**: Availability status of the volunteer.
* **Methods:**
  + **register():** Allows the volunteer to register in the system.
  + **offerHelp():** Allows the volunteer to offer help for an incident.

1. **EmergencyService**

The EmergencyService class represents emergency response services such as police, fire department, and medical services.

* **Attributes:**
  + **serviceId**: Unique identifier for the service.
  + **type**: Type of emergency service.
  + **contactInfo**: Contact information for the service.
* **Methods:**
  + **respondToIncident():** Responds to an incident.
  + **provideUpdate():** Provides updates on the incident response.

1. **EmergencyContacts**

The EmergencyContacts class represents important contacts to be notified during an emergency.

* **Attributes:**
  + **contactId**: Unique identifier for the contact.
  + **name**: Name of the contact.
* **Method:**
  + **callContact():** Calls the emergency contact.

1. **WeatherDataProvider**

The WeatherDataProvider class represents providers of weather data, which can be crucial during certain types of disasters.

* **Attributes:**
  + **providerId**: Unique identifier for the service.
  + **weatherData**: Type of emergency service.
* **Method:**
  + **supplyData():** Supplies weather obtained from the provider.

1. **Government Agency**

The GovernmentAgency class represents government entities involved in disaster management.

* **Attributes**:
  + **agencyId**: Unique identifier for the agency.
  + **name**: Name of the agency.
  + **directives**: Directives issued by the agency.
* **Method:**
  + **issueDirective():** Issues a directive in response to an incident.
  + **receiveReport():** Receives reports on incidents.

1. **AlertSystem**

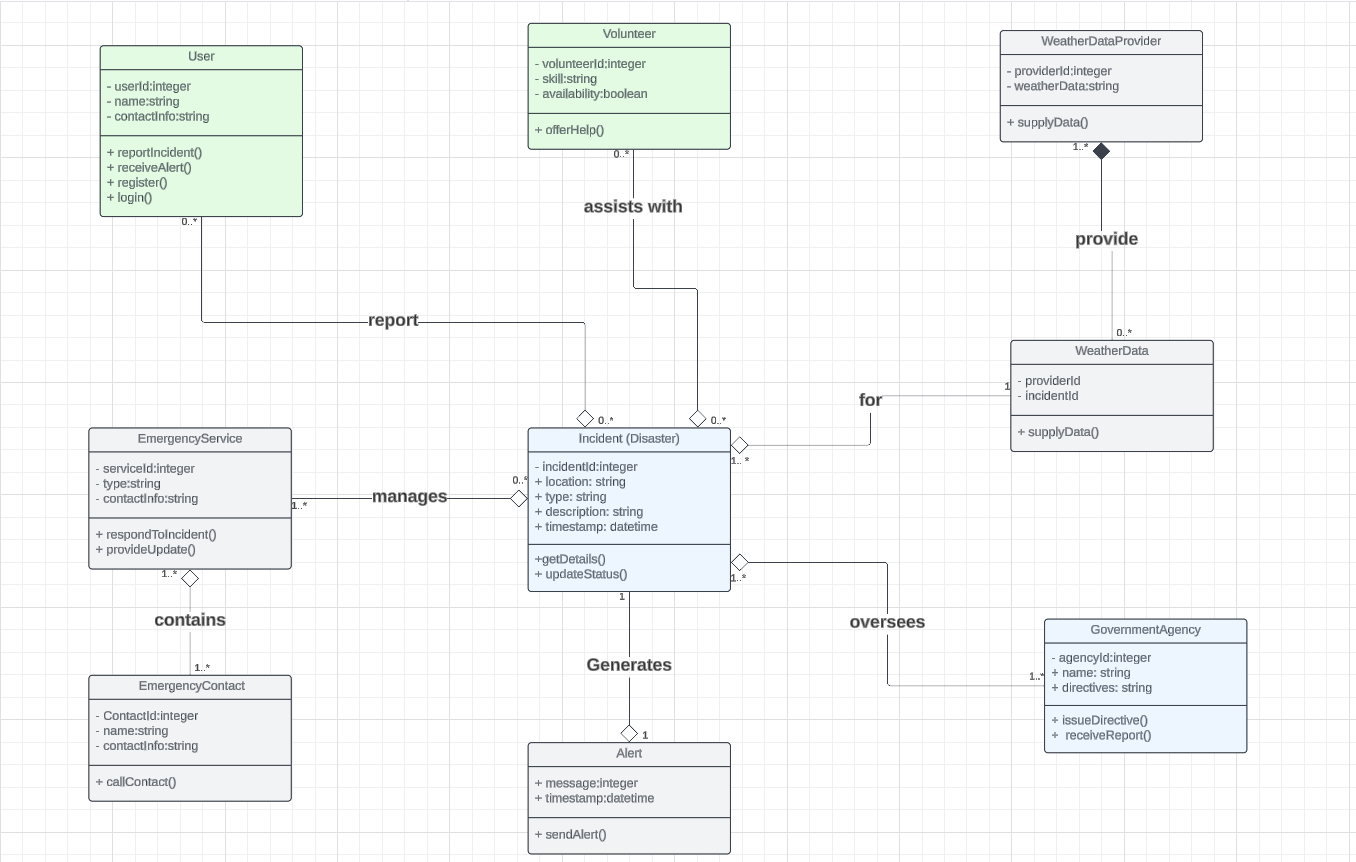
The AlertSystem class handles the triggering of alerts.

* **Attributes:**
  + **alertId**: Unique identifier for the service.
  + **message**: Type of emergency service.
  + **Timestamp:** Timestamp when the alert was created.
* **Method:**
  + **sendAlert():** Sends an alert to users.

## Relationships

* **User —Alert:** Users receive alerts.
* **Incident — User:** Users can report incidents.
* **Volunteer — Incident:** Volunteers offer help for incidents.
* **EmergencyService — Incident:** Emergency services respond to incidents.
* **EmergencyContacts — Incident**: Emergency contacts are notified about incidents.
* **WeatherDataProvider — Incident:** Weather data providers supply data relevant to incidents.
* **GovernmentAgency — Incident:** Government agencies issue directives and receive reports about incidents.

## Visual Representation



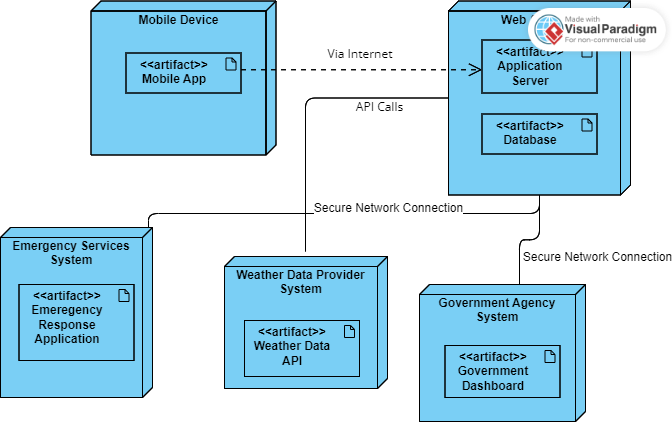
*Class Diagram of Disaster Management System*

# **Deployment Diagram**

A Deployment Diagram illustrates how software architecture, designed on a conceptual level, translates into the physical system architecture where the software will run as nodes. It maps out the deployment of software components onto hardware nodes and depicts their relationships through communication paths, enabling a visual representation of the software’s execution environment across multiple nodes.

For our mobile-based Disaster Management System, below are the nodes, artifacts and connections for the Deployment Diagram:

* Nodes:
* Mobile Device
  + Artifacts: Mobile App
* Web Server
  + Artifacts: Application Server, Database
* Emergency Services System
  + Artifacts: Emergency Response Application
* Government Agency System
  + Artifacts: Government Dashboard
* Weather Data Provider System
  + Artifacts: Weather Data API
* Connections:
* Mobile Device <-> Web Server (via Internet)
* Web Server <-> Emergency Services System (secure network connection)
* Web Server <-> Government Agency System (secure network connection)
* Web Server <-> Weather Data Provider System (API calls)



*Deployment Diagram for Disaster Management System*

# **Conclusion**

In navigating the complexities of modern software development, the system modeling and design stage stands as a critical pillar, ensuring that abstract ideas are transformed into tangible, functional solutions. Through the meticulous crafting of context diagrams, use case diagrams, class diagrams, sequence diagrams, and deployment diagrams, this phase provides a clear and comprehensive blueprint that bridges the gap between concept and implementation.

By visualizing and specifying every aspect of the system, we facilitate seamless communication among stakeholders, foster collaborative efforts, and lay the groundwork for a robust, scalable, and maintainable system. This structured approach not only enhances the clarity and precision of the development process but also mitigates risks and aligns the final product with the stakeholders' expectations and requirements.

As we conclude this phase, we embark on the next steps with a well-validated design that promises efficiency, effectiveness, and innovation. The models and diagrams created serve as both a guide and a testament to the foresight and detailed planning that underpin successful software development projects. In essence, the system modeling and design stage is not just a preparatory step; it is the cornerstone of building resilient, impactful systems that can adeptly respond to the dynamic demands of the real world.