

**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. System Requirements 2**](#_299iefckpwku)

[A. Functional Requirements 2](#_a62e1ei66r4)

[B. Non-Functional Requirements 3](#_jvgzrwopggw4)

[C. Data Requirements 4](#_xhieymsdm2o6)

[Types of Data 4](#_clch9ljck8wl)

[Data Quality Requirements 5](#_73ejt2dmvaod)

[**2. Technical and Operational Requirements 6**](#_493shep21mdc)

[A) Technical Requirements 6](#_i8uvjlfi7lti)

[B) Operational Requirements 8](#_qva9vj26tnim)

[**3. Prioritised Requirements 9**](#_3rdcrjn)

[A. Key Factors 10](#_vi454x6nktwc)

[B. Prioritizing User Needs 10](#_6ohynaiego3u)

[C. Proposed Needs from users 10](#_p2e8m6jj8v7d)

[**4. Development Model 11**](#_b5814h4h9h0r)

[**5. Assumptions and Constraints 13**](#_cx4ntnt3p1gt)

[Assumptions 13](#_krjpqttkt0o1)

[a. User Base and Technology Access: 13](#_e5x7t7pvb38b)

[b. Communication Infrastructure: 13](#_aj5b3rsbqhvw)

[c. Government and Institutional Support: 14](#_7w9csth2otk9)

[d. Community and Public Awareness: 14](#_2xk3kaq2ssum)

[Constraints 14](#_fzizbkoczqrg)

[**6. Conclusion 15**](#_ms9itkh6gu46)

# **System Requirements**

## Functional Requirements

Functional requirements specify the essential functions and behaviors that the system must perform. For our Disaster Management System, these include:

**a. User Authentication and Authorization:**

The system must authenticate users through secure login mechanisms (e.g., username/password, multi-factor authentication).

Different levels of access rights should be defined for various roles such as administrators, responders, and volunteers.

**b. Real-time Alerting System**

The system should be able to deliver real-time alerts to users based on their geographic locations and preferences.

Alerts should cover various types of emergencies such as natural disasters, public safety threats, and health emergencies.

Users should have the option to customize their alert preferences, including the types of alerts they receive and the communication channels (e.g., push notifications, SMS, email).

**c. Incident Reporting and Assistance**

Users should be able to report incidents directly through the mobile application, including providing details such as the type and location of the incident.

The system should facilitate coordination between emergency responders, volunteers, and affected communities to allocate resources and provide assistance promptly.

Users should have access to a directory of emergency contacts and resources for requesting assistance during emergencies.

**d. Geospatial Data Integration**

The system should integrate with geospatial data and mapping services to provide users with interactive maps displaying real-time information about disaster-affected areas, evacuation routes, shelter locations, and other relevant spatial data.

Users should be able to view overlays of various data layers (e.g., flood zones, fire risk areas) on the map to enhance situational awareness and decision-making during emergencies.

**e. Community Engagement Features**

The system should include features to promote community engagement and collaboration, such as forums, chat rooms, and social media integration.

These tools should facilitate information sharing, peer support, and collective action among users and stakeholders involved in disaster response and recovery efforts.

**f. Resource Management:**

Track and manage resources such as personnel, equipment, and supplies.

Allocate resources based on real-time needs and availability.

**g. First aid kit tools and best practices**

Efficiently manages and tracks first aid supplies. It maintains real-time inventory, plans and distributes kits based on needs, deploys resources swiftly during incidents, and generates usage reports for future planning. This ensures a prompt and effective medical response during disasters.

**h. Public Information Dissemination:**

Publish updates and information to the public via websites, social media, and other channels. Manage press releases and public service announcements.

## Non-Functional Requirements

Non-functional requirements focus on the quality attributes of the system, ensuring it performs effectively under various conditions. For a Disaster Management System, these include:

* **Performance:**

The system must have a quick response time, ideally under two seconds for most operations, to ensure timely data processing and decision-making.

It should handle a high volume of simultaneous users, especially during peak times of disaster events.

* **Scalability:**

The system should be able to scale horizontally to accommodate increasing data loads and user traffic.

Support for cloud-based deployment to dynamically adjust resources based on demand.

* **Security:**

Implement robust security measures to protect sensitive data, including encryption, secure communication protocols, and regular security audits.

Ensure compliance with relevant data protection regulations (e.g., GDPR, HIPAA).

* **Reliability:**

The system should have high availability, aiming for 99.9% uptime to ensure continuous operation during critical times.

Implement failover and disaster recovery mechanisms to maintain functionality during outages.

* **Usability:**

Provide an intuitive and user-friendly interface, ensuring that users with varying technical skills can navigate and use the system effectively.

Offer training modules and documentation to assist users in learning the system.

* **Compliance:**

Ensure the system meets all relevant regulatory standards and guidelines for disaster management and data handling.

Regularly update compliance measures to adhere to changing laws and regulations.

## Data Requirements

A Disaster Management System (DMS) relies on accurate, timely, and comprehensive data to effectively manage disaster response and recovery efforts. Here are the types of data we will use;

## Types of Data

**1. Disaster Information**

**Type:** Structured

**Description:** Details about the disaster, including type (e.g., earthquake, flood, hurricane), location, severity, and duration.

**Sources**: Meteorological data, geological surveys, emergency reports.

**2. Resource Data**

**Type:** Structured

**Description:** Information on available resources, such as personnel, equipment, supplies, and their current status and location.

**Sources:** Inventory management systems, real-time tracking tools.

**3. Incident Reports**

**Type:** Structured and Unstructured

**Description:** Reports on specific incidents, including time of occurrence, description, and status updates.

**Sources:** Field reports, emergency call centers, mobile reporting apps.

**4. Geospatial Data**

**Type:** Structured

**Description:** Geographic information systems (GIS) data, including maps, satellite images, and spatial analytics.

**Sources:** GIS databases, satellite imagery providers, drones.

**5. Communication Logs**

**Type:** Unstructured

**Description:** Records of communications between responders, agencies, and the public, including emails, messages, and calls.

**Sources:** Communication platforms, email servers, call logs.

**6. Public Information and Alerts**

**Type:** Structured and Unstructured

**Description:** Public advisories, alerts, and updates, including evacuation notices and safety instructions.

**Sources:** Government websites, social media, public broadcasting systems.

**7. Historical Data**

**Type:** Structured

**Description:** Historical records of past disasters, response efforts, and outcomes to inform current practices and decision-making.

**Sources:** Historical archives, research institutions, government databases.

## Data Quality Requirements

**1. Accuracy**

Data must be precise and free from errors to ensure reliable decision-making.

Regular validation and verification processes should be in place.

**2. Timeliness**

Data must be up-to-date and available in real-time, especially during disaster events.

Implement real-time data feeds and automated data collection where possible.

**3. Completeness**

Ensure all necessary data fields are populated to provide a comprehensive view of the situation.

Integrate data from multiple sources to fill gaps and provide a holistic perspective.

**4. Consistency**

Data should be consistent across different systems and reports.

Use standardized formats and coding systems to maintain uniformity.

**5. Security and Privacy**

Protect sensitive data from unauthorized access and breaches.

Comply with relevant data protection regulations (e.g., GDPR, HIPAA).

# **Technical and Operational Requirements**

### Technical Requirements

* **APIs**: APIs play a crucial role in a disaster management app by enabling seamless integration with external services. These can include weather data providers, emergency services, and social media platforms. Integrating with weather data APIs allows the app to provide real-time updates on weather conditions, forecasts, and alerts, which are vital for preparing and responding to natural disasters. Connecting with emergency services APIs ensures that the app can disseminate alerts, coordinate rescue operations, and manage resources effectively. Additionally, social media integration allows the app to monitor and aggregate user-generated content, which can provide real-time insights into disaster conditions and facilitate rapid communication with the public. This multi-source integration ensures comprehensive situational awareness and enhances the app's ability to respond swiftly and efficiently to disasters.
* **Cloud-Based Servers (e.g., AWS, Azure, Google Cloud):** Cloud-based servers such as AWS, Azure, and Google Cloud offer significant advantages for a disaster management app. These platforms provide high availability, scalability, and flexibility, essential for handling the unpredictable nature of disaster scenarios. Cloud servers can scale up or down based on demand, accommodating spikes in user activity during emergencies. This elasticity is crucial for maintaining performance and availability during peak usage periods. The pay-as-you-go model makes cloud solutions cost-effective, as you only pay for the resources you use. Built-in redundancy ensures that services remain operational even if some servers fail, while the global reach of cloud providers allows for distributed deployments that enhance reliability and accessibility. Advanced security features, including encryption, identity management, and compliance certifications, help protect sensitive data and ensure regulatory compliance..
* **Network Infrastructure:** A robust network infrastructure is essential for ensuring continuous connectivity and performance in a disaster management app. Key components such as load balancers distribute traffic across multiple servers, preventing any single server from becoming overwhelmed and ensuring smooth operation even during peak usage. Redundant internet connections and high-bandwidth routers and switches are critical for maintaining connectivity and performance, even if part of the network fails. This redundancy is vital during disasters, as it helps maintain operational integrity and ensures that the app remains accessible to users when it is needed most. The use of virtual private networks (VPNs) and secure sockets layer (SSL) encryption can further enhance the security and reliability of network communications..
* **Security Software:** Implementing comprehensive security software is paramount for a disaster management app to safeguard against cyber threats. Regular audits and penetration testing are essential practices for identifying and addressing vulnerabilities. Security audits involve systematic evaluations of the app's security posture, ensuring compliance with industry standards and best practices. Penetration testing simulates cyber-attacks to identify and fix security gaps before they can be exploited by malicious actors. Additional security measures, such as firewalls, intrusion detection and prevention systems (IDPS), and data encryption, help protect against unauthorized access, data breaches, and other security threats. Regular updates and patch management are also crucial to address newly discovered vulnerabilities and maintain a robust security posture.
* **Architectural Frameworks:** Adopting architectural frameworks like microservices architecture significantly enhances the scalability and reliability of a disaster management app. In a microservices architecture, the application is broken down into smaller, independent components, each responsible for a specific function. This modularity allows individual components to be scaled independently based on their load, ensuring efficient resource utilization and improving overall scalability. Fault isolation is another key benefit, as issues in one component do not affect the entire system, enhancing the app's resilience. Microservices can be developed, deployed, and maintained independently, enabling faster updates and more agile responses to changing requirements. This architecture supports continuous integration and continuous deployment (CI/CD) practices, facilitating rapid development and deployment of new features and improvements.

### Operational Requirements

* **Data Backup**: Ensuring robust data backup mechanisms is crucial for protecting against data loss in a disaster management app. Implementing both incremental and full backups provides comprehensive protection. Incremental backups, which save only the changes made since the last backup, are performed frequently (e.g., hourly or daily) to ensure recent data is preserved with minimal storage requirements. Full backups, conducted at regular intervals (e.g., weekly), capture the entire dataset, providing a complete restore point. Additionally, cloud backups play a critical role in disaster recovery strategies. By storing backup data in the cloud, the system ensures that data remains safe and accessible even if the primary site is compromised due to local failures, such as hardware malfunctions or natural disasters. This multi-layered approach to data backup ensures that the system can recover quickly and effectively from any data loss incident.
* **Disaster Recovery:** A robust disaster recovery plan is essential for maintaining the operational continuity of a disaster management app. Implementing failover mechanisms automates the process of switching to backup servers in the event of a primary server failure. This ensures that the system can be quickly restored to operational status following a disruption, minimising downtime and data loss. Failover solutions can include real-time replication of data to secondary servers and automated detection of server failures to trigger the failover process. By maintaining backup servers that are always ready to take over, the system ensures high availability and reliability, which are critical during disaster scenarios when the app must remain operational to provide timely information and coordination.
* **Real-Time Monitoring:** Real-time monitoring is essential for maintaining the performance and reliability of a disaster management app. Utilizing advanced monitoring tools allows for continuous tracking of system performance, detecting anomalies, and alerting administrators promptly. These tools can monitor various metrics, such as server load, response times, database performance, and network traffic. Real-time alerts enable administrators to respond quickly to potential issues, preventing minor problems from escalating into major disruptions. Additionally, monitoring tools can provide valuable insights into usage patterns, helping to identify areas for optimization and ensuring the system performs efficiently under varying loads, especially during high-demand periods typical of disaster situations..
* **Optimization:** Optimizing the performance of a disaster management app is crucial for ensuring it runs smoothly and efficiently, especially during periods of high demand. Effective traffic distribution across servers using load balancers prevents any single server from becoming overwhelmed, ensuring consistent performance and availability. Database performance can be enhanced through indexing, query optimization, and partitioning. Indexing speeds up data retrieval operations, while query optimization ensures that database queries are executed as efficiently as possible. Partitioning large databases into smaller, more manageable segments improves performance and manageability. These optimization techniques collectively ensure that the app can handle increased loads during disaster events, providing reliable and timely information to users when it is needed most.

# **Prioritised Requirements**

Here, we look at the key requirements we would be considering in order to design and build the system. First and foremost, we look at the key factors to consider in order to prioritize our requirements:

## Key Factors

* **Business Value and Urgency:** Evaluate the potential business value and urgency of each requirement. Focus on addressing high-value needs that have immediate impact on project success or customer satisfaction.
* **Feasibility:** Consider the technical feasibility and complexity of implementing each requirement within the project constraints, including time, budget, and available resources. Prioritize requirements that are achievable within the project scope and constraints to ensure successful delivery.
* **Risk:** Evaluate the potential risks and uncertainties associated with each requirement, including the likelihood of failure, impact on other project activities, and consequences of not addressing the requirement. Prioritize requirements that mitigate high-risk areas or address critical vulnerabilities to minimize project risks.
* **Stakeholder Needs and Dependencies:** Consider the needs and preferences of stakeholders, as well as dependencies between requirements. Engage stakeholders in the prioritization process and prioritize requirements that address critical dependencies or align with stakeholder expectations.

## Prioritizing User Needs

We divide all user needs according to percentage need from our end users

* Needs with 90 to 100% Acceptance
  + Real-time Alerts and Notifications in case of any disaster occurrence.
* Needs with 70 to 90% Acceptance
  + Location-based services
* Needs with 50 to 70% Acceptance
  + Emergency contacts
  + Offline functionality
  + Multi-language functionality.
  + Educational resources for sensitization
* Needs with 50% and below
  + Communication tools for messaging
  + Reporting incidents
  + Resource coordination

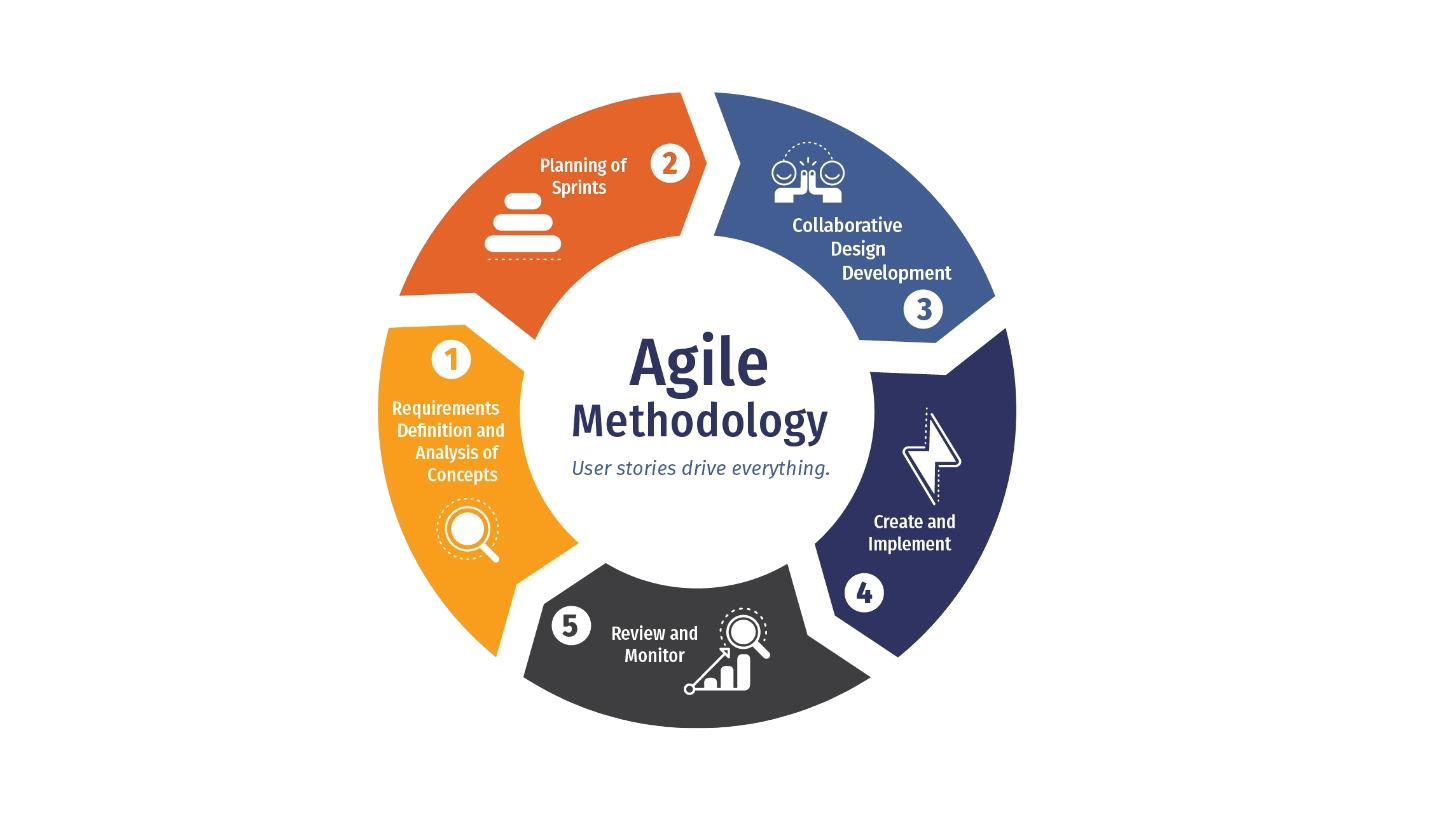
## Proposed Needs from users

Below are a list of needs users proposed to be in the system and could be included if the meet our key factors of prioritization.

1. Quick Medical Aid from Emergency Units
2. History of Hazardous Areas.
3. DIYs videos in Emergency situations
4. Fast response to these needs in case of an occurring disaster.

# **Development Model**

**Agile Development Model**

****

**What is Agile?**

Agile is a software development methodology that emphasizes iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. It promotes flexible responses to change, continuous improvement, and rapid delivery of functional product increments.

**Core Principles of Agile:**

1. Individuals and Interactions over processes and tools.

2. Working Software over comprehensive documentation.

3. Customer Collaboration over contract negotiation.

4. Responding to Change over following a plan.

**Reason for Choosing Agile:**

* **Flexibility and Adaptability:**

The nature of disaster management is inherently unpredictable, and the requirements might change based on new information, stakeholder feedback, or emerging technologies.  
Agile allows for changes to be made easily at any point in the development cycle, ensuring the application remains relevant and useful.

* **Frequent Delivery of Functional Software:**

Disaster management applications need to be reliable and quickly available to users.  
By delivering functional increments regularly, stakeholders can see progress, provide feedback, and ensure the application meets their needs effectively.

* **Stakeholder Engagement:**

Effective disaster management requires close collaboration with various stakeholders (emergency responders, community members, NGOs, etc.).  
Agile emphasizes customer collaboration, ensuring that stakeholder input is continuously integrated into the development process.

* **Continuous Improvement:**

Disaster scenarios and technologies evolve, necessitating ongoing enhancement of the application.  
Benefit: Agile’s iterative nature promotes regular reflection and improvement, ensuring the application evolves with the needs of users.

* **Risk Management**

Developing a disaster management application involves significant risks, including changing requirements and technological challenges.  
Agile helps mitigate risks by delivering small, functional parts of the application early and frequently, allowing for early detection and resolution of issues.

# **Assumptions and Constraints**

When we plan anything, from building a house to developing an app, we make assumptions and encounter constraints.

## Assumptions

These are educated guesses about how things will be. They are based on our knowledge and experience, but they may not always be true. In the context of designing a mobile based disaster management system, we had the following assumptions:

### User Base and Technology Access:

* **Smartphone Penetration:** A certain baseline smartphone penetration within the target population was assumed. This helped determine the reach of the app and the need for alternative features for non-smartphone users.
* **Basic Literacy and Numeracy:** A base level of literacy and numeracy in the target population was assumed. This allowed for designing a user interface that is easy to understand and navigate, even in stressful situations.

### Communication Infrastructure:

* **Some Level of Connectivity:** We assumed that some level of cellular or Wi-Fi connectivity will be available, even if disrupted during a disaster. This allows for features that require an internet connection but necessitates designing for intermittent connectivity as well.
* **Standardised Emergency Communication Protocols:** The existence of standardised emergency communication protocols (e.g., emergency alert systems) was assumed. The app can integrate with these protocols to ensure a comprehensive communication strategy.

### Government and Institutional Support:

* **Collaboration with Emergency Responders:** Cooperation from government agencies and emergency responders was assumed. This allows for integrating real-time data feeds and official instructions into the app.
* **Open Data Sharing:** A willingness to share relevant data openly from the government and other institutions was assumed. This data is critical for populating the app with accurate maps, evacuation routes, and shelter locations.

### Community and Public Awareness:

* **Public Interest in Disaster Preparedness:** A general public interest in disaster preparedness was assumed. This justifies the development of the app and suggests a willingness to learn and use it.
* **Availability of Local Trainers and Promoters:** The existence of local organisations and volunteers who can be trained on the app and become promoters within the community was assumed. This is crucial for ensuring widespread adoption and effective utilisation.

## Constraints

These are limitations we have to work within. These can be due to factors like budget, time, or technology. For instance, a limited budget might constrain the features we can include in the app. These limitations can be technical, practical, or resource-based.

* **Technology Limitations**: Mobile devices themselves have limitations. Battery life is a major concern, especially during disasters. Processing power and storage capacity of phones can also restrict the complexity of features that can be incorporated.
* **Connectivity Issues:** Disasters can severely disrupt communication networks, making it difficult for the app to function or for users to receive updates. The system needs to be designed to work with limited or even no internet connectivity.
* **Cost and Development:** Developing and maintaining a robust mobile app can be expensive. There are ongoing costs for updates, security, and ensuring compatibility with different devices and operating systems.
* **Data Availability:** The app's effectiveness relies on access to accurate and up-to-date data (maps, evacuation routes, shelter locations). Gathering and maintaining this data can be challenging, especially in resource-constrained regions.
* **User Adoption and Training:** Encouraging widespread adoption of the app within the community requires overcoming factors like lack of awareness, limited digital literacy, or resistance to new technology. Training users on how to effectively leverage the app's features is another hurdle.
* **Integration with Existing Systems:** The app needs to integrate with existing emergency response systems and communication protocols to avoid creating a silo of information. This can involve overcoming technical hurdles and ensuring cooperation from different stakeholders.

These constraints pose challenges, but by acknowledging them during the development and planning stages, we can design a system that is more realistic, effective, and adaptable within the limitations.

# **Conclusion**

A thorough requirements analysis is the cornerstone of building a successful mobile disaster management app. By meticulously identifying user needs, technical limitations, and potential risks, we can develop a system that is:

* **Effective:** Focused on features and functionalities that truly empower users during emergencies.
* **Accessible:** Designed with inclusivity in mind, catering to diverse user bases and technological capabilities.
* **Sustainable:** Built with a clear understanding of ongoing costs, data management, and integration with existing infrastructure.

A mobile disaster management app is not just a piece of software; it is a potential lifeline. By investing in a comprehensive requirements analysis, we can ensure this app becomes a powerful tool for saving lives, minimising damage, and fostering resilience within communities most vulnerable to disasters.