

**REPUBLIC OF CAMEROON
PEACE-WORK-FATHERLAND
UNIVERSITY OF BUEA
BUEA, SOUTH-WEST REGION
P.O BOX 63.**



**RÉPUBLIQUE DU CAMEROUN
PAIX-TRAVAIL-PATRIE
UNIVERSITÉ DE BUEA
BUEA, RÉGION DU SUD-OUEST
B.P. 63.**

**FACULTY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING
SOFTWARE ENGINEERING**

CEF440: Internet and Mobile Programming

DESIGN AND IMPLEMENTATION OF A DATABASE FOR A DISASTER MANAGEMENT MOBILE APPLICATION

Course Facilitator:
Dr. NKEMENI Valery, PhD

Date :
July 2024

Presented by:

GROUP 2

NAMES	MATRICULE
QUINUEL TABOT NDIP-AGBOR	FE21A300
SIRRI THERESIA ANYE	FE21A306
NGONCHI RAMATOU YOLAND	FE21A260
CHE BLAISE NJI	FE21A157
LIMA CHARLES	FE21A225

ABSTRACT

This abstract outline the design and implementation of a disaster management mobile application (DMA) aimed at enhancing community resilience and preparedness. The DMA leverages key conceptual frameworks like Disaster Risk Reduction (DRR) and Community-Based Disaster Management (CBDM) to provide a comprehensive platform for users. Drawing upon theoretical frameworks like Diffusion of Innovations Theory and Social Cognitive Theory, the application prioritizes user-centered design, accessibility, and ease of use. Key features include:

- Real-time alerts and warnings: Users receive timely notifications about impending natural hazards and emergencies based on location and threat level.
- Interactive risk assessment tools: Users can access customized risk assessments tailored to their location, providing information on potential threats and preparedness strategies.
- Community communication platform: Facilitates communication within local communities, allowing residents to share information, coordinate assistance, and report critical incidents.
- Resource mapping and navigation: Users can access location-based maps indicating evacuation routes, emergency shelters, and critical resources like medical facilities.
- Disaster preparedness training modules: Interactive training modules provide users with practical skills and knowledge to prepare for various disaster scenarios.

The DMA prioritizes data security and privacy, while promoting seamless integration with existing emergency response systems for improved coordination and information dissemination. The app aims to foster a proactive community approach to disaster management, empowering residents with the tools and knowledge to build resilience and enhance preparedness in the face of potential hazards.

Contents

ABSTRACT	iii
CHAPTER 1: GENERAL INTRODUCTION	1
CHAPTER 2: LITERATURE REVIEW	2
2.1. Conceptual Frameworks:	2
2.2. Theoretical Frameworks:	3
2.3. Design and Implementation Considerations:	4
2.4. Research Gaps and Future Directions:	4
CHAPTER 3: REQUIREMENTS GATHERING	6
3.1. Expected Outcomes	6
3.2. System Requirements For A Disaster Management Mobile Application	6
CHAPTER 4: REQUIREMENT ANALYSIS	10
4.1. Requirement Gathering Techniques:	10
4.2. Documenting Requirements:	10
4.4. Requirements classification	15
4.4.1. Functional Requirements	15
4.4.2. Non-Functional Requirements	17
4.4.3. Stakeholders and User Requirements	19
4.4.4. CAP Theorem Analysis	21
CHAPTER 5: SYSTEM MODELING AND DESIGN FOR A DISASTER MANAGEMENT MOBILE APPLICATION	22
5.1. CONTEXT DIAGRAM	22
1.1. System:	22
1.2. External Entities (Actors):	22
1.3. Data Flows	23
1.4. Context diagram of a Disaster Management Mobile Application	24
5.2. USE CASE DIAGRAM	24
2.1. Actors	24
2.2. Actors and their USE CASES and the Include and Extend Relationships	25
2.3. Use case diagram of a disaster management system	27
5.3. CLASS DIAGRAM	27
5.3. 1. Classes definition, attributes and methods	28
5.3.2. Relationships between classes	34
5.3.3. Class Diagram of a Disaster Management Mobile Application	37
5.4. SEQUENCE DIAGRAM	37

5.4.1.	Objects, functions and Sequential Procedure.....	37
5.4.3.	Sequence Diagram of a disaster management mobile application.	41
5.5.	DEPLOYMENT DIAGRAM.....	42
5.5.1.	Components:.....	42
5.5.2.	Communication Flow:	44
5.5.3.	Deployment:	44
5.5.4.	Deployment diagram of a disaster management system.....	45
CHAPTER 6: UI DESIGN AND IMPLEMENTATION FOR A DISASTER MANAGEMENT MOBILE APPLICATION		47
6.1.	DESIGN PHASE	47
i.	Admin persona	47
ii.	Emergency Responder	48
iii.	Civilian (normal user)	49
6.2.	SCENARIOS OF A DISASTER MANAGEMENT MOBILE APPLICATION.....	50
6.3.	USER JOURNEY MAP	52
6.4.	MAP VISUALISATION THROUGH WIRE FRAMING	53
i.	Wire framing objects and attributes of the main Screens.....	53
ii.	Wire framing sketches.....	57
6.5.	IMPLEMENTATION WITH FIGMA.	58
i.	Registration/Login	58
ii.	Home screen	60
iii.	Emergency Responders Contact	61
iv.	Incident reporting.....	62
v.	Alerts/Notifications	63
vi.	Setting/Updates	64
CHAPTER 7: DATABASE DESIGN AND IMPLEMENTATION.....		65
7.1.	Overview	65
7.2.	PROBLEM STATEMENT/ REQUIREMENT: DESIGNING DATABASE FOR A DISASTER MANAGEMENT MOBILE APPLICATION	65
7.3.	CONCEPTUAL DESIGN: ER DIAGRAM	66
7.4.	IMPLEMENTATION: FIREBASE (LOGICAL AND PHYSICAL DESIGN).....	69
7.5.	RESULTS	72
A.	Collection and Document	72
B.	Rules	75
CHAPTER 8: DEVELOPMENT AND RESULTS		76

8.1. Project Overview:	76
8.2. Technical Implementation:	77
8.3. CODE SCREEN SHOTS	81
8.4. Results	84
CHAPTER 10: CONCLUSION	94
CHAPTER 11: REFERENCES/ FUTURE RECOMMENDATION	95
10.1. References	95
10.2. Future Enhancements:	95

CHAPTER 1: GENERAL INTRODUCTION

A disaster management mobile application is a tool designed to help individuals and communities prepare for, respond to, and recover from natural or man-made disasters. These applications typically provide information on emergency preparedness, real-time alerts and updates during a disaster, resources for immediate assistance, and guidance on how to navigate the aftermath of a disaster.

The design of a disaster management mobile application should prioritize ease of use, accessibility, and reliability. The user interface should be intuitive and user-friendly, with clear navigation and easily accessible information. The application should also be designed to work in low-connectivity or offline environments, as network infrastructure may be compromised during a disaster.

In terms of implementation, a disaster management mobile application typically involves the following components:

1. Real-time alert system: The application should be able to send push notifications or SMS alerts to users based on their location and the type of disaster occurring in their area.
2. Emergency resources: The application should provide access to emergency contact information, evacuation routes, shelter locations, and other essential resources during a disaster.
3. Preparedness guides: The application should offer tips and guides on how to prepare for different types of disasters, such as creating an emergency kit or developing a family communication plan.
4. Community engagement: The application can facilitate communication and coordination among community members, allowing users to report incidents, request assistance, or offer help to others in need.
5. Recovery support: After a disaster has passed, the application can provide information on recovery efforts, assistance programs, and resources for rebuilding and recovery.

Overall, the design and implementation of a disaster management mobile application should prioritize user safety, accessibility, and functionality to effectively assist individuals and communities in managing and recovering from disasters.

CHAPTER 2: LITERATURE REVIEW

This literature review explores the design and implementation of disaster management mobile applications (DMAs), emphasizing the conceptual and theoretical frameworks guiding their development. It aims to provide insights into key considerations for building effective and user-centered DMAs that contribute to disaster risk reduction (DRR) and community resilience.

2.1. Conceptual Frameworks:

- Disaster Risk Reduction (DRR):

- * Hyogo Framework for Action (2005-2015): This framework emphasizes community engagement, information sharing, and multi-stakeholder partnerships in reducing disaster risks. (UNISDR, 2005)

- * Sendai Framework for Disaster Risk Reduction (2015-2030): Building on the Hyogo Framework, the Sendai Framework further emphasizes the need for early warning systems, risk assessments, and community-based preparedness. (UNISDR, 2015)

- * Applications: DMAs contribute to DRR by:

- * Disseminating timely warnings: (e.g., "DisasterAid" app by the Red Cross)

- * Providing access to evacuation routes: (e.g., "SafeCast" app)

- * Facilitating communication between authorities and citizens: (e.g., "Ushahidi" platform for citizen reporting)

- Community-Based Disaster Management (CBDM):

- * Emphasizes local community involvement in managing risks and fostering resilience through collective action and local knowledge.

- * Applications: DMAs can enhance CBDM by:

- * Enabling community mapping: (e.g., "MapAction" tool)

- * Facilitating resource mobilization: (e.g., "GoodSAM" app for volunteer coordination)

- * Supporting peer-to-peer communication: (e.g., "Nextdoor" app for local neighborhood communication)

- Information and Communication Technologies (ICTs) for Disaster Management:

- * Focuses on leveraging ICTs to improve communication, data collection, and coordination

in disaster response.

- * Applications: DMAs leverage ICTs for:

- * Real-time data sharing: (e.g., "Flood Watch" app by the U.S. National Weather Service)

- * Crowd-sourced information: (e.g., "Ushahidi" platform for citizen reporting)

- * Remote monitoring of disaster impacts: (e.g., "Earthquakes and Volcanoes" app by the USGS)

2.2. Theoretical Frameworks:

- Diffusion of Innovations Theory:

- * Explains how new ideas and technologies are adopted by individuals and communities.

- * Applications: This theory can guide DMA design and promotion by understanding factors like relative advantage, compatibility, and observability influencing app adoption. (Rogers, 2003)

- Social Cognitive Theory:

- * Emphasizes the role of self-efficacy, outcome expectancies, and social support in influencing behavior change.

- * Applications: DMAs can promote self-efficacy in disaster preparedness by:

- * Providing training materials: (e.g., interactive simulations)

- * Simulating disaster scenarios: (e.g., "Disaster Ready" app)

- * Fostering social interaction amongst users: (e.g., community forums within the app)

(Bandura, 1986)

- Risk Perception Theory:

- * Explores how individuals perceive and respond to risk, influenced by cultural factors, personal experiences, and media portrayals.

- * Applications: DMAs can leverage risk perception theory to:

- * Tailor messaging and information delivery: (e.g., using culturally relevant language and images)

- * Ensure users understand the severity of risks: (e.g., providing clear and concise risk assessment information) (Slovic, 1987)

2.3. Design and Implementation Considerations:

- **User-Centered Design:** Prioritize user needs and preferences through user research, iterative prototyping, and usability testing. (Norman, 2013)
- **Accessibility:** Ensure accessibility for diverse users, including those with disabilities, different literacy levels, and language barriers. (WCAG, 2021)
- **Functionality:** Include features that support various disaster phases (prevention, preparedness, response, recovery), such as:
 - * Early warning systems: (e.g., push notifications, alerts)
 - * Emergency communication tools: (e.g., two-way communication channels, SOS features)
 - * Resource mapping and navigation: (e.g., location-based services, evacuation routes)
 - * Information dissemination: (e.g., news updates, safety guidelines)
 - * Community engagement features: (e.g., forums, crowdsourcing)
- **Data Management and Security:** Implement robust data security measures to protect user privacy and ensure data integrity.
- **Integration with Existing Systems:** Seamlessly integrate the DMA with existing emergency response systems, early warning systems, and GIS platforms for efficient coordination and information exchange.
- **Sustainability:** Ensure long-term sustainability by considering factors like:
 - * Funding models: (e.g., government grants, private investment)
 - * Maintenance and updates: (e.g., regular updates, app store optimization)
 - * User engagement and retention strategies: (e.g., gamification, rewards)

2.4. Research Gaps and Future Directions:

- **User Experience and Accessibility:** Further research is needed on designing DMAs that are user-friendly, accessible, and culturally appropriate for diverse populations.
- **Impact Evaluation:** Rigorous evaluations are needed to assess the effectiveness of DMAs in reducing vulnerability, improving preparedness, and enhancing resilience.
- **Data Privacy and Security:** More research is needed to develop best practices for data privacy and security in DMA development, particularly in light of the increasing reliance on user data.

- **Integration with Existing Systems:** Continued research is needed to develop strategies for seamless integration of DMAs with existing emergency response systems and infrastructure.

This literature review highlights the potential of DMAs to play a crucial role in disaster risk reduction and community resilience. By incorporating conceptual and theoretical frameworks, prioritizing user-centered design, and addressing key design and implementation considerations, developers can create effective and impactful DMAs that empower communities to prepare for and respond to disasters.

CHAPTER 3: REQUIREMENTS GATHERING

In today's world, natural disasters and emergencies are a constant threat. To effectively manage these situations, we need robust disaster management systems. Traditional methods, often reliant on manual processes and scattered communication channels, struggle to deliver timely and coordinated responses. This project proposes a solution: a Mobile-Based Disaster Management System. Research, focus group, survey forms, interviews were used to gather these requirements.

3.1. Expected Outcomes

- A multifunctional mobile application supporting all stages of disaster management: preparedness, response, recovery, and mitigation. The application will have user-friendly interfaces catering to both individuals and emergency responders.
- A real-time alerting and notification system delivering critical information during emergencies. Alerts will be customized based on user location and preferences.
- The ability for users to report incidents and request assistance directly through the application. This will facilitate efficient resource allocation and timely response.
- Integration with geospatial data and mapping services, providing users with real-time information about disaster zones, evacuation routes, shelter locations, and other relevant spatial data.
- Features promoting community engagement and collaboration, such as forums, chat rooms, and social media integration. These tools will enable information sharing, peer support, and collective action during disaster response and recovery.

3.2. System Requirements For A Disaster Management Mobile Application

System requirements define the characteristics and functionalities a disaster management mobile application must possess to meet its objectives. These different requirements are:

1. Real-time Alerts and Notifications:

- System should deliver critical information like warnings about impending disasters, evacuation orders, and safety instructions.

- Allow for user preferences to customize alerts based on location severity (e.g., receive alerts only for major disasters or within a specific radius).
- Multiple notification channels can be used (push notifications, text messages, in-app alerts).

2. Incident Reporting and Assistance:

- Users can report incidents (e.g., flooding, fire damage, injuries) directly through the app.
- Reporting should include options to capture details like location (using GPS), type of incident, photos/videos (if possible), and any additional relevant information.
- The system should allow users to request assistance from specific emergency services (police, fire department, ambulance) based on the reported incident.

3. Geospatial Data and Mapping Services:

- Integrate with real-time geospatial data services to display disaster-affected areas on interactive maps.
- Maps should show key information like evacuation routes, shelter locations, flood zones, and critical infrastructure status.
- Users' current location should be displayed on the map for better situational awareness.

4. Community Engagement and Collaboration:

- Integrate features like forums, chat rooms, and social media feeds to facilitate information sharing and communication among users and stakeholders involved in disaster response.
- Users can share updates, request help, offer assistance, and connect with others in their community during emergencies.
- Allow for anonymous posting if desired to encourage broader participation.

5. Preparedness and Mitigation Resources:

- Provide a centralized repository of educational resources, guides, and checklists to help users prepare for different types of disasters.
- Content should cover topics like creating emergency plans, assembling disaster kits, and practicing safety measures.
- Resources can be downloadable for offline access.

6. Emergency Contacts and Services:

- Offer a comprehensive directory of emergency contact information for various services (police, fire department, hospitals, poison control).
- Allow users to easily call or message emergency services directly from the app.
- Information should be categorized and searchable for quick access during emergencies.

7. **Offline Capability and Data Sync:**

- The app should function to some extent even without an internet connection.
- Users should be able to access critical information like alerts, maps (pre-downloaded versions), and emergency resources offline.
- The system should queue incident reports and other user-generated data when offline, automatically syncing it with the server when an internet connection is re-established.

8. **Multi-Language Support:**

- The app interface and content should be available in multiple languages to cater to a diverse user base.
- Allow users to easily switch between languages based on their preference.
- Localization should consider cultural nuances to ensure clear and effective communication.

9. **Privacy and Security Measures:**

- **Data Encryption:** Implement strong encryption algorithms to protect user data during transmission and storage. This includes sensitive information like location, incident details, and potentially even personal information used for registration (if applicable).
- **Authentication and Authorization:** Consider user authentication mechanisms (logins, passwords) to restrict access to sensitive functionalities or user data (optional depending on app design).
- **Data Minimization:** Collect only the data essential for app functionality. Avoid unnecessary data collection to minimize privacy concerns.
- **Transparent Privacy Policy:** Clearly outline the app's data collection practices, how user data is used, and user rights regarding their data.

10. **Real-time Delivery of Alerts and Notifications:**

- **Low Latency Communication:** Utilize efficient communication protocols and server infrastructure to minimize delays in delivering critical alerts and notifications. This is crucial for ensuring timely warnings during emergencies.
- **Push Notification Optimization:** Optimize push notification delivery to minimize reliance on a stable internet connection. Explore alternative methods like SMS or local network broadcasts for critical alerts in case of internet outages.
- **Notification Retries:** Implement mechanisms to retry sending notifications if there are initial delivery failures due to network issues.

11. Accuracy and Reliability of Geospatial Data:

- **Data Source Validation:** Establish partnerships with reliable sources for geospatial data, such as government agencies or reputable mapping services.
- **Data Refresh Mechanisms:** Implement processes to regularly update geospatial data, especially during emergencies when information about disaster zones and evacuation routes can change rapidly.
- **Data Accuracy Checks:** Integrate data validation techniques to identify and correct any potential inaccuracies in the geospatial data before it's displayed in the app.

12. Accessibility of the Application:

- **WCAG Compliance:** Strive for conformance with the Web Content Accessibility Guidelines (WCAG) to ensure the app is accessible to users with disabilities. This includes features like:
 - **Screen Reader Compatibility:** The app interface should be compatible with screen reader software for visually impaired users.
 - **Increased Color Contrast:** Implement sufficient color contrast between text and background elements for better readability by users with visual impairments.
 - **Keyboard Navigation:** Allow users to navigate the app interface using a keyboard instead of a touchscreen, catering to users with motor impairments.

These requirements were gathered through various means in order to know what the end-users, stakeholders wants from the system.

CHAPTER 4: REQUIREMENT ANALYSIS

Developing a successful disaster management mobile application hinge on understanding the needs of its users and stakeholders. This document outlines a comprehensive approach to requirement gathering for this crucial technology.

4.1. Requirement Gathering Techniques:

We employed various techniques to gather requirements from stakeholders and users:

- **Stakeholder Workshops:** Facilitate discussions to identify critical needs, functionalities, and success metrics.
- **User Interviews and Surveys:** Gather user feedback on their expectations from the application and potential pain points during disaster scenarios.
- **Focus Groups:** Conduct moderated discussions with diverse user groups to explore needs, preferences, and accessibility considerations.
- **Scenario Mapping:** Simulate potential disaster situations and brainstorm functionalities that would be helpful in those scenarios.
- **Competitive Analysis:** Review existing disaster management applications to identify best practices and potential areas for improvement.

4.2. Documenting Requirements:

Capture all gathered requirements in a clear and organized manner. This involved:

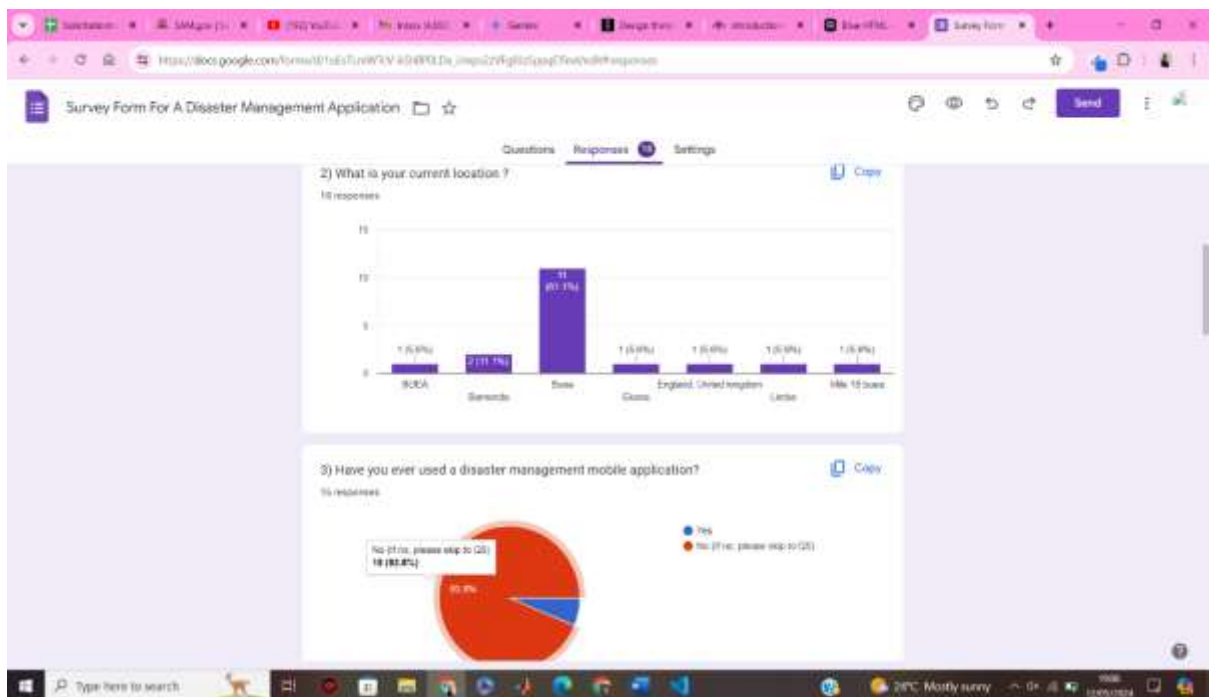
- **Use Cases:** Describe specific user interactions with the application to achieve particular goals.
- **User Stories:** Briefly describe functionalities from the user's perspective (e.g., "As a resident, I want to receive real-time alerts about approaching storms").
- **Functional Requirements:** Define the specific actions the application should be able to perform (e.g., display interactive maps, allow users to report incidents).
- **Non-Functional Requirements:** Specify performance, usability, security, and accessibility considerations.

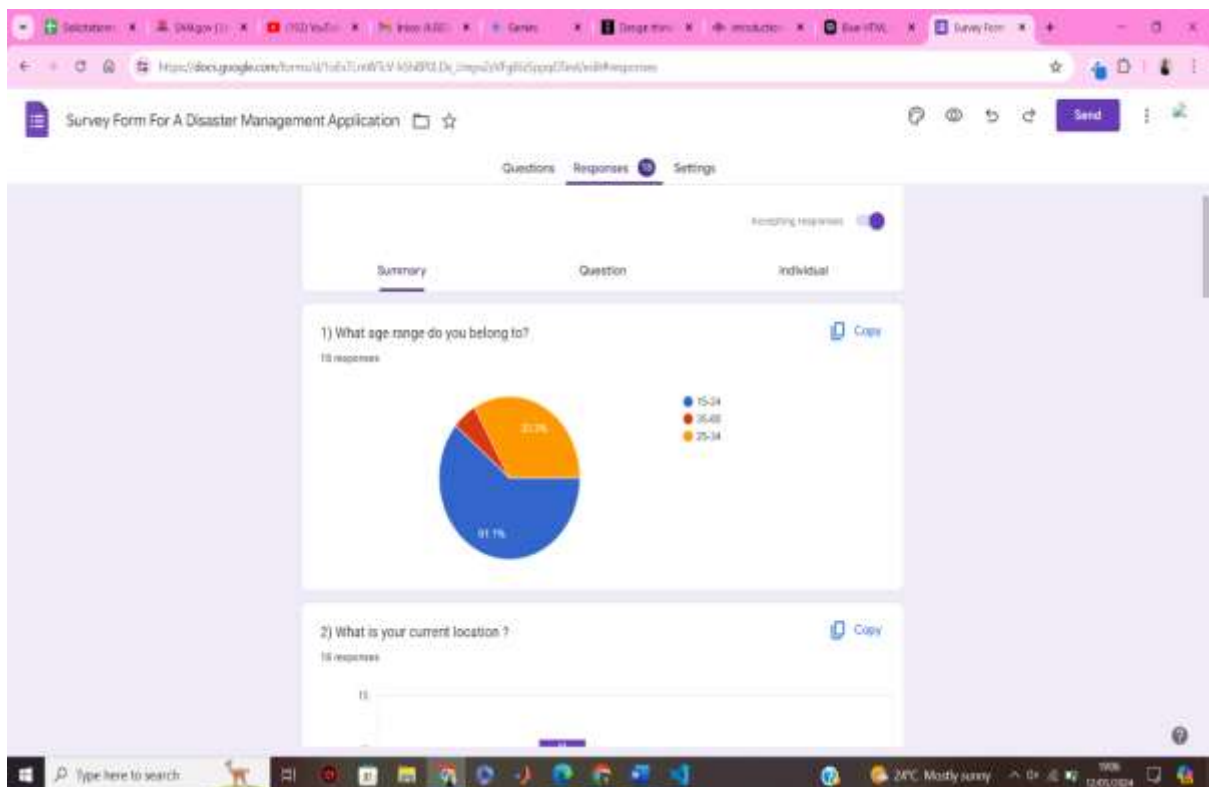
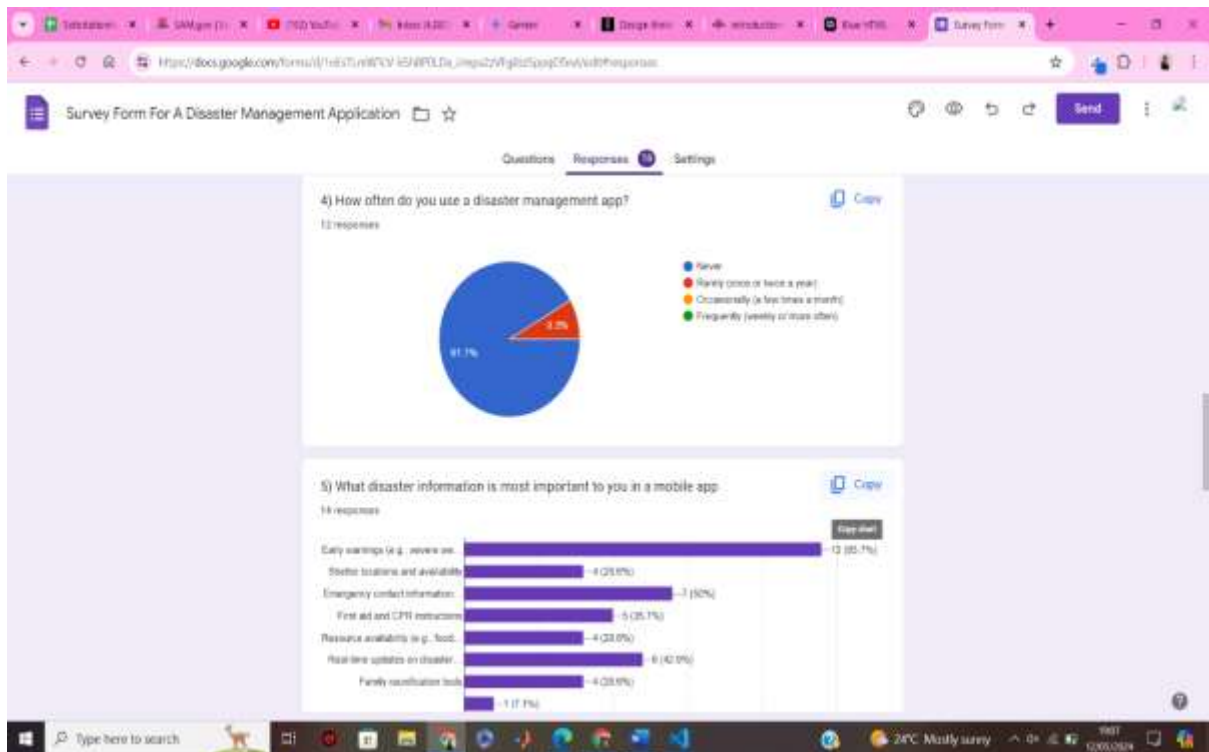
- **Prioritization:**

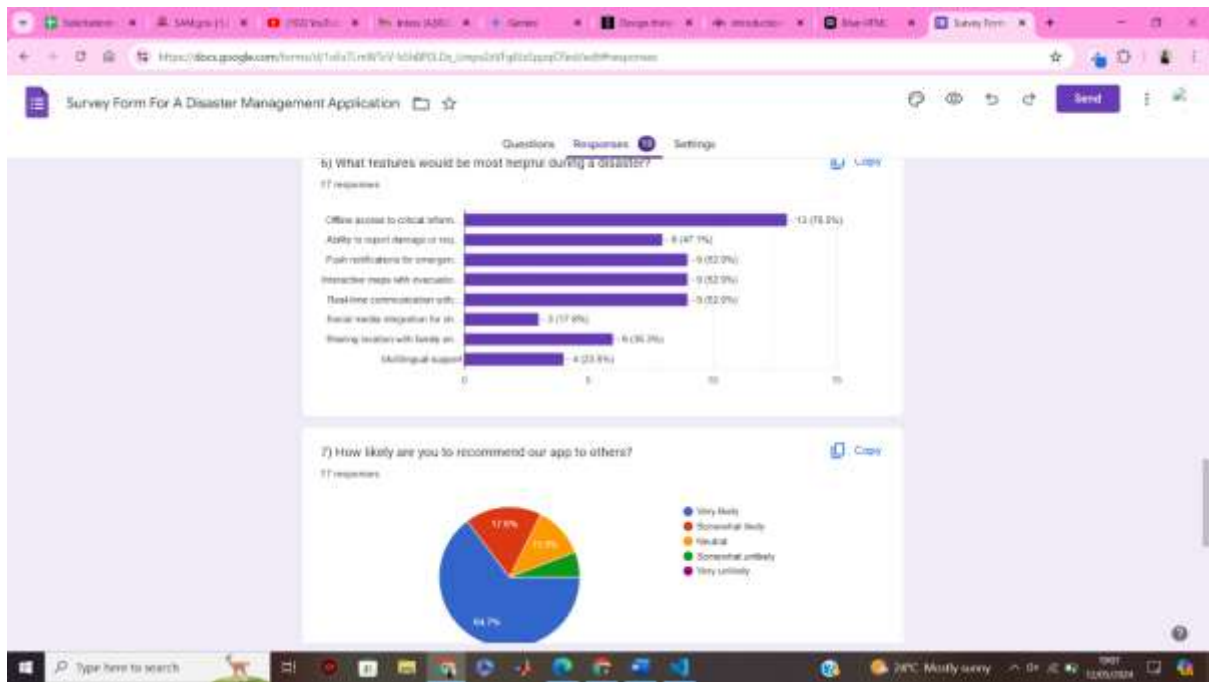
Not all requirements have equal importance. So we employed strategies like survey forms, interviews, and research .

➤ **Survey Form Analysis:**

We sent out a survey with 8 questions with over 30 responses using Google Forms which made is quite easy to collect and analysis statistics, As seen below:







8) What suggestions do you think will be necessary in a disaster management mobile application?

10 responses

- Have a general facilities
- Climate change report section
- Aid to those affected
- First aid
- The app should be efficient and reliable
- I don't know since I have never used one
- A way to know how fast and how far the disaster is going
- it should be free

➤ Research Analysis:

Developing a disaster management mobile application requires understanding user needs and existing solutions. Leveraging internet research and interviews to gather valuable insights:

➤ Internet Research:

- **Academic Papers and Reports:** Search for research papers and reports on disaster management mobile applications. These can provide insights into user needs, successful functionalities, and common challenges. Look for reputable sources like academic journals, government agencies (e.g., FEMA in the US), or disaster response organizations.
- **Existing Disaster Management Apps:** Analyze existing disaster management applications available on app stores.
 - Identify features they offer (e.g., real-time alerts, incident reporting, resource maps).
 - Read user reviews to understand user pain points and areas for improvement.
 - Explore competitor apps to identify potential gaps your application can address.
- **Industry Reports and News Articles:** Look for industry reports and news articles about disaster management trends and technological advancements. These can highlight emerging technologies like real-time data analytics or wearable device integration that could enhance your app.

➤ Interviews:

- **User Interviews:** Interview potential users of your disaster management application, focusing on individuals residing in disaster-prone areas.
 - Learn about their concerns and information needs during emergencies.
 - Identify what functionalities they would find most valuable and how they typically access information during disasters (e.g., smartphones, internet access).
 - Include people with disabilities to ensure the app is accessible to everyone.

4.4. Requirements classification

The system requirements analysis for a mobile disaster management application. Traditional disaster management methods often lack efficiency due to manual processes. These requirements can be categorized into different types:

- **Functional Requirements:** These define the specific actions the application should be able to perform. In our case, this includes features like real-time alerts, incident reporting, and access to resources.
- **Non-Functional Requirements:** These address overall qualities of the application, like performance, security, and usability. Examples include real-time delivery of alerts and ensuring the application is accessible to diverse users.
- **Business Requirements:** These focus on how the application aligns with the organization's goals. This could involve facilitating information sharing among stakeholders or coordinating response efforts.
- **Technical Requirements:** These specify the technical aspects needed to develop the application, such as integrating with mapping services or implementing multi-language support.
- **Stakeholder and User Requirements:** These capture the specific needs of the application's intended users. Examples include customized alerts based on location and forums for community engagement.

4.4.1. Functional Requirements

a. Real-time Alerts and Notifications:

- System should deliver critical information like warnings about impending disasters, evacuation orders, and safety instructions.
- Allow for user preferences to customize alerts based on location severity (e.g., receive alerts only for major disasters or within a specific radius).
- Multiple notification channels can be used (push notifications, text messages, in-app alerts).

b. Incident Reporting and Assistance:

- Users can report incidents (e.g., flooding, fire damage, injuries) directly through the app.
 - Reporting should include options to capture details like location (using GPS), type of incident, photos/videos (if possible), and any additional relevant information.
 - The system should allow users to request assistance from specific emergency services (police, fire department, ambulance) based on the reported incident.
- **Geospatial Data and Mapping Services:**
 - Integrate with real-time geospatial data services to display disaster-affected areas on interactive maps.
 - Maps should show key information like evacuation routes, shelter locations, flood zones, and critical infrastructure status.
 - Users' current location should be displayed on the map for better situational awareness.
- **Community Engagement and Collaboration:**
 - Integrate features like forums, chat rooms, and social media feeds to facilitate information sharing and communication among users and stakeholders involved in disaster response.
 - Users can share updates, request help, offer assistance, and connect with others in their community during emergencies.
 - Allow for anonymous posting if desired to encourage broader participation.
- **Preparedness and Mitigation Resources:**
 - Provide a centralized repository of educational resources, guides, and checklists to help users prepare for different types of disasters.
 - Content should cover topics like creating emergency plans, assembling disaster kits, and practicing safety measures.
 - Resources can be downloadable for offline access.
- **Emergency Contacts and Services:**
 - Offer a comprehensive directory of emergency contact information for various services (police, fire department, hospitals, poison control).
 - Allow users to easily call or message emergency services directly from the app.

- Information should be categorized and searchable for quick access during emergencies.
- **Offline Capability and Data Sync:**
 - The app should function to some extent even without an internet connection.
 - Users should be able to access critical information like alerts, maps (pre-downloaded versions), and emergency resources offline.
 - The system should queue incident reports and other user-generated data when offline, automatically syncing it with the server when an internet connection is re-established.
- **Multi-Language Support:**
 - The app interface and content should be available in multiple languages to cater to a diverse user base.
 - Allow users to easily switch between languages based on their preference.
 - Localization should consider cultural nuances to ensure clear and effective communication.

4.4.2. Non-Functional Requirements

1. Privacy and Security Measures:

- **Data Encryption:** Implement strong encryption algorithms to protect user data during transmission and storage. This includes sensitive information like location, incident details, and potentially even personal information used for registration (if applicable).
- **Authentication and Authorization:** Consider user authentication mechanisms (logins, passwords) to restrict access to sensitive functionalities or user data (optional depending on app design).
- **Data Minimization:** Collect only the data essential for app functionality. Avoid unnecessary data collection to minimize privacy concerns.
- **Transparent Privacy Policy:** Clearly outline the app's data collection practices, how user data is used, and user rights regarding their data.

2. Real-time Delivery of Alerts and Notifications:

- **Low Latency Communication:** Utilize efficient communication protocols and server infrastructure to minimize delays in delivering critical alerts and notifications. This is crucial for ensuring timely warnings during emergencies.
- **Push Notification Optimization:** Optimize push notification delivery to minimize reliance on a stable internet connection. Explore alternative methods like SMS or local network broadcasts for critical alerts in case of internet outages.
- **Notification Retries:** Implement mechanisms to retry sending notifications if there are initial delivery failures due to network issues.

3. Accuracy and Reliability of Geospatial Data:

- **Data Source Validation:** Establish partnerships with reliable sources for geospatial data, such as government agencies or reputable mapping services.
- **Data Refresh Mechanisms:** Implement processes to regularly update geospatial data, especially during emergencies when information about disaster zones and evacuation routes can change rapidly.
- **Data Accuracy Checks:** Integrate data validation techniques to identify and correct any potential inaccuracies in the geospatial data before it's displayed in the app.

4. Accessibility of the Application:

- **WCAG Compliance:** Strive for conformance with the Web Content Accessibility Guidelines (WCAG) to ensure the app is accessible to users with disabilities. This includes features like:
 - **Screen Reader Compatibility:** The app interface should be compatible with screen reader software for visually impaired users.
 - **Increased Color Contrast:** Implement sufficient color contrast between text and background elements for better readability by users with visual impairments.
 - **Keyboard Navigation:** Allow users to navigate the app interface using a keyboard instead of a touchscreen, catering to users with motor impairments.>9

4.4.3. Stakeholders and User Requirements

- **Stakeholders:** These are individuals or organizations that have a vested interest in the success of the application but may not directly use it themselves. They are impacted by the application's effectiveness and functionality.
- **Emergency Responders (Police, Fire Department, Ambulance):**
 - **Needs:** Real-time information on incidents, secure communication channels for coordination, efficient resource allocation based on user reports.
 - **Benefits:** Improved response times, better situational awareness, streamlined collaboration.
- **Government Agencies:**
 - **Needs:** Dissemination of critical information to the public, data collection for post-disaster analysis, centralized platform for resource management.
 - **Benefits:** Enhanced public safety and awareness, improved coordination between emergency services, informed decision-making for future preparedness efforts.
- **Non-profit Organizations and NGOs:**
 - **Needs:** Platform for sharing resources and connecting with affected communities, ability to coordinate volunteer efforts.
 - **Benefits:** Increased visibility and outreach, streamlined resource distribution, efficient deployment of volunteers.

Users (General Public): These are the individuals who will directly interact with the mobile application to access its features and functionalities.

- **Needs:** Timely and location-specific alerts, ability to report incidents and request assistance, access to emergency resources and evacuation information.
 - **Benefits:** Improved preparedness, increased sense of security, ability to connect with others during emergencies, access to critical resources during disasters.

- **Additional Considerations:**

- **Accessibility:** Ensure the application caters to users with disabilities (visual impairments, hearing impairments, etc.) through features like text-to-speech functionality or closed captions.
- **Multilingual Support:** The application should be available in multiple languages to reach a diverse user base.

4. Technical Requirements

Developing a feature-rich and effective disaster management app necessitates a clear understanding of the technical needs. Here's a breakdown of key requirements:

- **Front-End Development:**

- **Native App:** Developing a native Android app using tools like Android Studio ensures optimal performance, access to native device features (GPS, camera) and a seamless user experience tailored for the Android platform.
- **User Interface (UI) Design:** Figma, a web-based design tool, allows for collaborative UI design, prototyping, and creating a user-friendly and intuitive interface for the application.

- **Back-End Development:**

- **React Native:** This JavaScript framework enables building native-looking mobile apps using reusable components, streamlining development and potentially reducing maintenance costs.
- **Node.js:** This server-side JavaScript runtime environment facilitates building scalable and real-time functionalities like real-time alerts and communication features.

- **Database:**

- **Fire base:** This NoSQL document database offers flexibility and scalability for storing various types of data, including incident reports, user information, and resource locations.

- **Additional Considerations:**

- **Real-time Communication:** Integrate real-time messaging libraries like Socket.IO or Pusher to enable real-time communication features like chat rooms or push notifications.

- **Offline Functionality:** Implement mechanisms like caching and local storage to allow users to access critical information (e.g., alerts, maps) even without an internet connection.
- **Security:** Prioritize robust security measures like data encryption, user authentication, and secure API access to protect user data and system integrity.

4.4.4. CAP Theorem Analysis

The CAP Theorem, a fundamental concept in distributed systems, states that it's impossible to achieve all three properties (Consistency, Availability, Partition Tolerance) simultaneously. Let's analyse how our disaster management application might prioritize these properties:

- **Consistency:** Ensuring data consistency across all device instances is crucial. When a user reports an incident, the information should be immediately reflected on the server and other users' devices. MongoDB's eventual consistency model can provide an acceptable level of consistency for most functionalities, with data eventually becoming consistent across all replicas.
- **Availability:** During emergencies, ensuring application and data availability is paramount. A serverless architecture using cloud platforms like AWS Lambda can provide high availability by automatically scaling resources to handle increased user loads during disasters.
- **Partition Tolerance:** The app should function even if network connectivity is disrupted (partitioning). Offline functionalities with local data storage (e.g., cached maps) can ensure the app remains usable even in areas with limited internet access.

CHAPTER 5: SYSTEM MODELING AND DESIGN FOR A DISASTER MANAGEMENT MOBILE APPLICATION

In order to effectively manage disasters, it is essential to have a comprehensive plan in place that includes the use of mobile technology. The Unified Modeling Language (UML) is a standard modeling language that can be used to design and design software systems. UML provides a set of graphical notations that can be used to represent the different aspects of a software system, including its architecture, behavior, and data.

In this work, we present a UML-based modeling approach for a disaster management mobile application. The UML Diagrams that will be used are:

- Context diagram
- Use Case diagram
- Class case diagram
- Sequence diagram
- Deployment diagram

Each of the above diagrams will be defined by its various components, procedure of realisation and tools used and diagram

5.1. CONTEXT DIAGRAM

A **context diagram** is a high-level visual representation that provides an overview of a system's interactions with external entities. It helps define the scope, boundaries, and relationships of a system. In our case of a disaster management mobile application, we used **draw.io software** to draw the context diagram. In our case, we have a system and external entities.

1.1. System:

The Disaster Management System.

1.2. External Entities (Actors):

User: The primary user of the app.

Emergency Services: Entities like police, fire department, ambulance.

Geospatial Data Service: External service providing real-time geospatial data.

System Administrator: Manages and maintains the system.

Government Agency: Provides official alerts, guidelines, and receives incident reports.

1.3. Data Flows

i. User to System:

Submit incident reports.

Customize alert preferences.

Request assistance.

Access Preparedness resources.

Access emergency contacts.

Use offline capabilities.

Switch language preferences.

ii. System to User:

Deliver real-time alerts and notifications.

Display interactive maps.

Provide preparedness resources.

Sync data when back online.

iii. System to Emergency Services:

Forward incident reports and assistance requests.

iv. Emergency Services to System:

Provide response status.

v. System to Geospatial Data Service:

Request real-time geospatial data.

vi. Geospatial Data Service to System:

Provide geospatial data.

vii. System Administrator to System:

Manage users.

Update resources.

Perform system maintenance.

viii. Government Agency to System:

Provide official alerts and guidelines.

1.4. Context diagram of a Disaster Management Mobile Application.

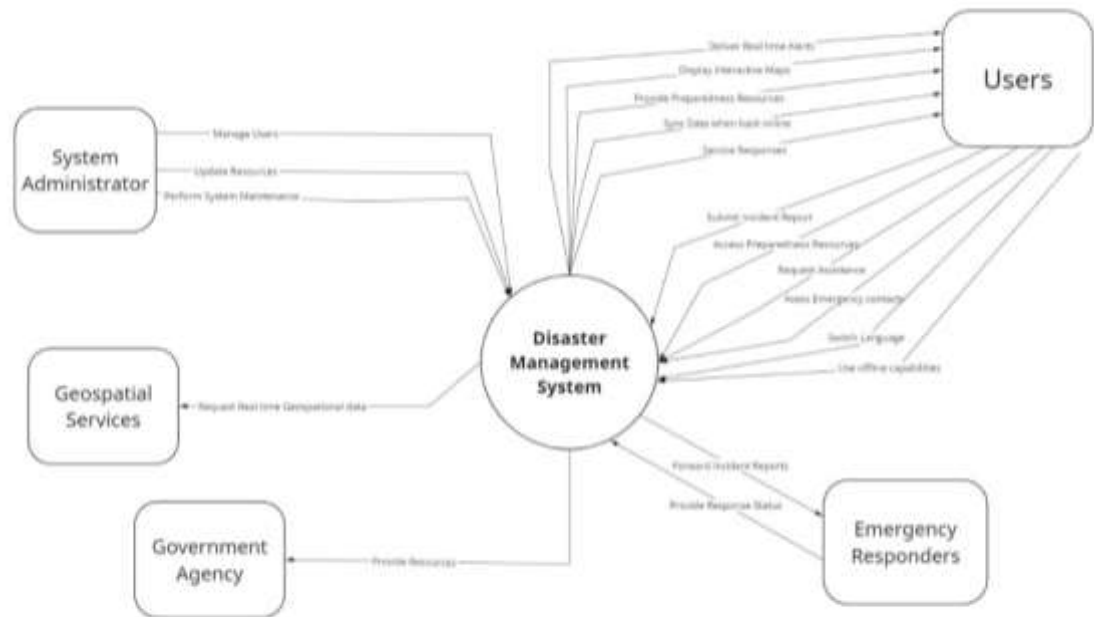


Figure 1 context diagram

5.2. USE CASE DIAGRAM

A **use case diagram** is a visual representation in the Unified Modeling Language (UML) that illustrates how users (actors) interact with a system. It provides a high-level view of the system's functionality, emphasizing the interactions between users and the system. In our case of a disaster management mobile application, we used **E-Drawmax software** to draw the use case diagram.

2.1. Actors

- Primary Actors:

Citizens/Users

Emergency Responders

- Secondary Actors:

Government Agencies

- System Actors:

System Administrator

2.2. Actors and their USE CASES and the Include and Extend Relationships

i. User/citizen

Report Incidents

Include Relationship

Authenticate User

It ensures that only verified and authorized citizens/users can report incidents, preventing potential misuse of the system.

Verify Alert

It validates the authenticity and accuracy of the emergency incident before sending out alerts, ensuring the information is reliable.

Request Emergency Assistance

Include Relationship:

Locate Incident

Receive Alerts and Notifications

Access Preparedness Resources

Include Relationship

Authenticate User: It validates the user's login credentials against the user accounts stored in the system's database.

Extend Relationship

Download Resource: downloading is part of accessing resources.

Access Emergency info

Include Relationship

Authenticate User: It validates the user's login credentials against the user accounts stored in the system's database.

Access Emergency Contact

Include Relationship

Search Contact: Users Can search contacts of emergency responders

Multilanguage Support

Include Relationship

Switch Language: Users can switch between Languages

Login

Include Relationship

Enable Single Sign-On: It allows users to log in using their existing credentials from trusted identity providers, such as social media or enterprise accounts.

Verify User Credentials: It validates the user's login credentials (username/email and password) against the system's user database.

Authorize User: It determine the user's permissions and access rights based on their role and profile, ensuring they can only perform authorized actions within the system.

ii. Emergency responders

Receive Incident Report

Include Relationship

Authenticate User: It validates the user's login credentials against the user accounts stored in the system's database.

Disseminate Emergency Information

Provide Medical Assistance

Login

Include Relationship

Enable Single Sign-On: It allows users to log in using their existing credentials from trusted identity providers, such as social media or enterprise accounts.

Verify User Credentials: It validates the user's login credentials (username/email and password) against the system's user database.

Authorize User: It determines the user's permissions and access rights based on their role and profile, ensuring they can only perform authorized actions within the system.

iii. System administrator

Manage User Account

Maintain the System

Update Resources

iv. Government agencies

Provide Resources

Manage Emergency Resources

Disseminate public information

2.3. Use case diagram of a disaster management system.

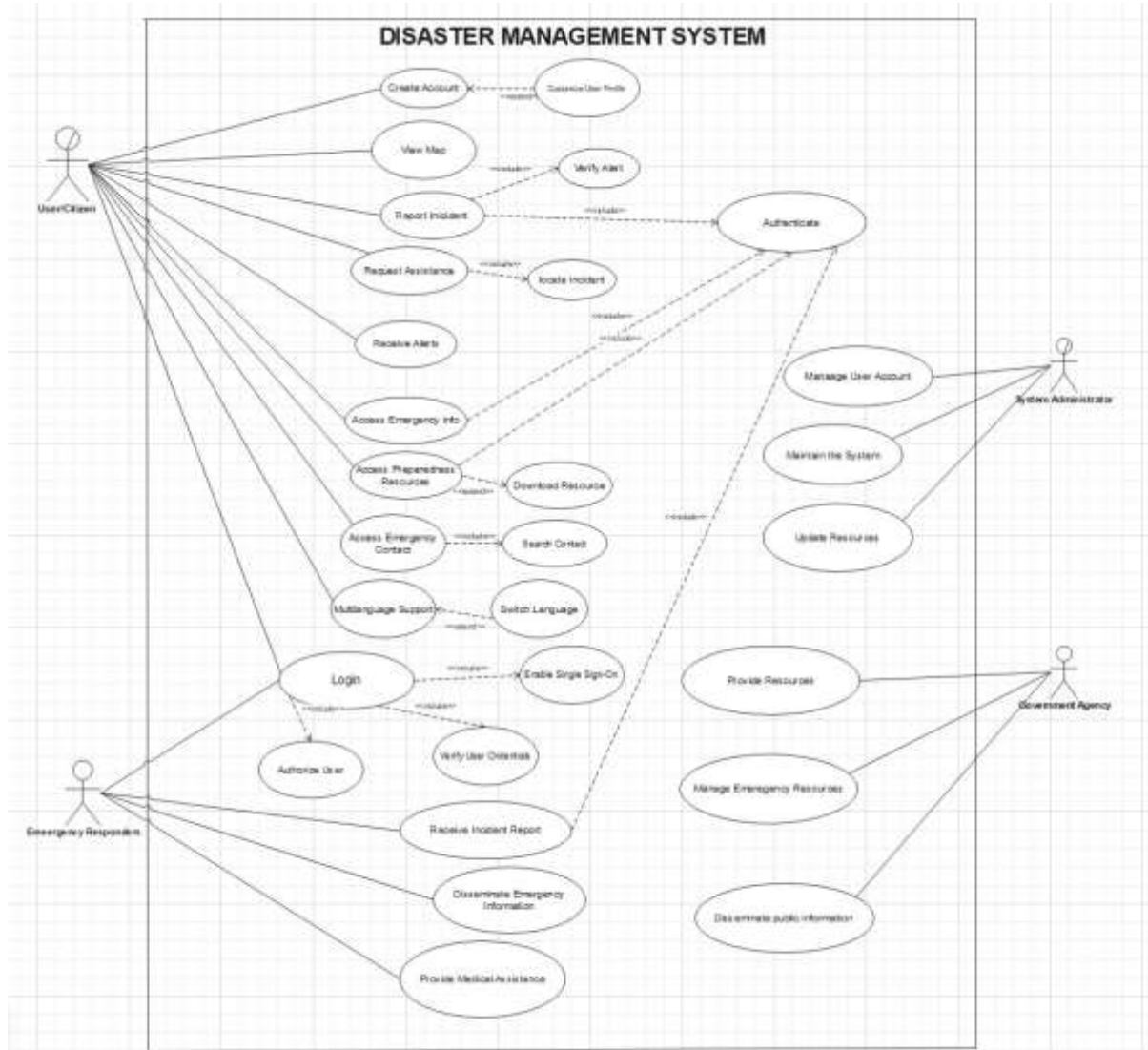


Figure 2: use case diagram

5.3. CLASS DIAGRAM

A **class diagram** is a type of static structure diagram in the Unified Modeling Language (UML) used in software engineering. It visually represents the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. In our context, we will give the various classes in our disaster management

mobile application, define them, give their attributes and methods, followed by the various relations that each class have with another.

5.3.1. Classes definition, attributes and methods

1. User:

- **Definition:**

- The **User** class represents individuals who interact with the disaster management mobile application. Users can report incidents, receive alerts, and access relevant information.

- **Attributes:**

- `userId (string)`: Unique identifier for each user.
- `username (string)`: User's display name.
- `email (string)`: User's email address.
- `languagePreference (string)`: Preferred language for notifications.
- `phoneNumber (string)`: User's contact number.
- `emergencyContacts (array of objects)`: Stores emergency contact information (name, phone number).

- **Methods:**

- `registerUser(username: string, email: string)`: Registers a new user.
- `updateProfile(username: string, email: string)`: Allows users to update their profile information.
- `setLanguagePreference(lang: string)`: Sets the preferred language.
- `addEmergencyContact(name: string, phoneNumber: string)`: Allows users to add emergency contacts.

- **Data Types:**

- `string, boolean, datetime, array.`

2. System:

- **Definition:**

- The **System** class represents the overall software infrastructure supporting the disaster management application.

- **Attributes:**

- `systemId (string)`: Unique identifier for the system.

- version (string): Current version of the application.
 - maintenanceMode (boolean): Indicates if the system is under maintenance.
 - lastUpdateTimestamp (datetime): Records the time of the last system update.
 - **Methods:**
 - checkSystemStatus(): Verifies system availability.
 - performSystemUpdate(): Initiates software updates.
 - **Data Types:**
 - string, boolean, datetime.
3. **Emergency Resources:**
- **Definition:**
 - The **Emergency Resources** class manages available resources (e.g., fire trucks, ambulances) for disaster response.
 - **Attributes:**
 - resourceId (string): Unique identifier for each resource.
 - resourceType (string): Type of emergency resource (e.g., medical, fire-fighting).
 - availability (boolean): Indicates if the resource is available.
 - location (string): Current location of the resource (GPS coordinates or address).
 - **Methods:**
 - requestResource(resourceType: string): Requests a specific resource.
 - updateResourceAvailability(resourceId: string, available: boolean): Updates resource availability.
 - updateResourceLocation(resourceId: string, location: string): Updates the resource's location.
 - **Data Types:**
 - string, boolean.
4. **Alert/Notification:**
- **Definition:**
 - The **Alert/Notification** class handles communication during emergencies.

- **Attributes:**
 - alertId (string): Unique identifier for each alert.
 - message (string): Content of the alert.
 - timestamp (datetime): Time when the alert was generated.
 - severityLevel (string): Severity level of the alert (e.g., low, moderate, high).
- **Methods:**
 - sendAlert(message: string, severity: string): Sends an emergency alert.
 - viewAlert(alertId: string): Displays alert details.
- **Data Types:**
 - string, datetime.

5. Preparedness and Mitigation:

- **Definition:**
 - The **Preparedness and Mitigation** class provides tips and guidelines for disaster preparedness.
- **Attributes:**
 - tipId (string): Unique identifier for each preparedness tip.
 - tipText (string): Practical advice for disaster preparedness.
 - category (string): Category of preparedness tip (e.g., earthquake, flood).
- **Methods:**
 - getRandomTip(): Retrieves a random preparedness tip.
- **Data Types:**
 - string.

6. Incident Reporting:

- **Definition:**
 - The **Incident Reporting** class allows users to report incidents, providing essential information for emergency responders.
- **Attributes:**
 - incidentId (string): Unique identifier for each reported incident.
 - location (string): Incident location description.
 - severity (string): Severity level (e.g., low, moderate, high).
 - reportedBy (string): User who reported the incident.

- **Methods:**
 - `reportIncident(location: string, severity: string)`: Allows users to report incidents.
 - `viewIncidentDetails(incidentId: string)`: Displays incident details.
 - `updateIncidentSeverity(incidentId: string, newSeverity: string)`: Allows authorized users to update incident severity.
- **Data Types:**
 - `string`.

7. Location:

- **Definition:**
 - The **Location** class deals with geospatial information related to incidents.
- **Attributes:**
 - `latitude (float)`: Latitude coordinate of an incident.
 - `longitude (float)`: Longitude coordinate of an incident.
 - `address (string)`: Human-readable address of the incident location.
 - `incidentType (string)`: Type of incident associated with this location (e.g., fire, flood).
- **Methods:**
 - `getIncidentLocation(incidentId: string)`: Retrieves the location details of a specific incident.
 - `convertCoordinatesToAddress(latitude: float, longitude: float)`: Converts GPS coordinates to an address.
- **Data Types:**
 - `float, string`.

8. Offline Functionalities:

- **Definition:**

- The **Offline Functionalities** class ensures that the app works even when there's no internet connection.
- **Attributes:**
 - `cachedData` (object): Stores essential data for offline use (e.g., maps, emergency contacts).
 - `offlineModeEnabled` (boolean): Indicates whether the app is currently in offline mode.
- **Methods:**
 - `syncDataWithServer()`: Synchronizes cached data with the server when online.
 - `enterOfflineMode()`: Activates offline functionality.
 - `exitOfflineMode()`: Deactivates offline mode when the device is back online.
- **Data Types:**
 - object, boolean.

9. Geospatial Mapping Services:

- **Definition:**
 - The **Geospatial Mapping Services** class handles map-related functionalities.
- **Attributes:**
 - `mapProvider` (string): Name of the map service provider (e.g., Google Maps, Leaflet).
 - `mapLayers` (array of strings): Available map layers (e.g., streets, satellite, topography).
 - `defaultZoomLevel` (integer): Default zoom level for the map.
- **Methods:**
 - `displayMap(location: string, zoomLevel: integer)`: Shows incident locations on the map.
 - `switchMapLayer(layerName: string)`: Allows users to switch between different map layers.
- **Data Types:**
 - string, array, integer.

10. Multilanguage Support:

- **Definition:**
 - The **Multilanguage Support** class ensures that the app can be used by speakers of different languages.
- **Attributes:**
 - `supportedLanguages` (array of strings): List of languages supported by the app.
 - `currentLanguage` (string): User's selected language.
 - `languageResources` (object): Contains translated strings for different languages.
- **Methods:**
 - `setLanguage(lang: string)`: Allows users to switch between supported languages.
 - `translateText(text: string, targetLanguage: string)`: Translates text to the specified language.
 - `getLocalizedString(resourceKey: string)`: Retrieves a localized string based on the current language.
- **Data Types:**
 - string, array, object.

11. Category:

- **Definition:**
 - The **Category** class organizes incidents into predefined categories, making it easier for users and responders to understand the nature of each incident.
- **Attributes:**
 - `categoryId` (string): Unique identifier for each category (e.g., fire, flood, earthquake).
 - `categoryName` (string): Descriptive name of the category (e.g., "Fire Incidents," "Flood Alerts").
 - `categoryIcon` (string): Icon representing the category (e.g., 🔥 for fire, 💧 for flood).
 - `incidentCount` (integer): Number of incidents associated with this category.

- `colorCode` (string): Hex code for the category color (e.g., “#FF5733” for fire).
- **Methods:**
 - `getCategoryName(categoryId: string)`: Retrieves the name of a specific category.
 - `getCategoryIcon(categoryId: string)`: Retrieves the icon associated with a category.
 - `getIncidentCount(categoryId: string)`: Returns the total number of incidents in this category.
 - `setCategoryColor(categoryId: string, colorCode: string)`: Allows customization of category colors.
- **Data Types:**
 - string, integer.

5.3.2. Relationships between classes

Describing the relationships between the classes in the context of a disaster management mobile application, considering various types of relationships:

1. User:

- **Relationships:**
 - **Association with Incident Reporting:** Users report incidents (association). Each user can report multiple incidents.
 - **Dependency on Multilanguage Support:** Users rely on the **Multilanguage Support** class for language preferences (dependency).

2. System:

- **Relationships:**
 - **Dependency on Emergency Resources:** The system relies on the **Emergency Resources** class for resource allocation (dependency).
 - **Dependency on Offline Functionalities:** The system depends on the **Offline Functionalities** class for offline functionality (dependency).

3. Emergency Resources:

- **Relationships:**

- **Association with Incident Reporting:** Emergency resources are dispatched based on incident reports (association). Each resource can be associated with multiple incidents.
 - **Dependency on Geospatial Mapping Services:** Emergency resources use geospatial data for navigation (dependency).
4. **Alert/Notification:**
- **Relationships:**
 - **Association with User:** Alerts and notifications are sent to users during emergencies (association). Each user can receive multiple alerts.
 - **Dependency on Multilanguage Support:** Alerts are translated into the user's preferred language (dependency).
5. **Preparedness and Mitigation:**
- **Relationships:**
 - **Association with User:** Preparedness tips and mitigation strategies are provided to users (association). Each user can access multiple tips.
6. **Incident Reporting:**
- **Relationships:**
 - **Association with User:** Users report incidents (association). Each user can report multiple incidents.
 - **Dependency on Location:** Incident locations are managed by the **Location** class (dependency).
7. **Location:**
- **Relationships:**
 - **Association with Incident Reporting:** Stores incident locations reported by users (association). Each incident is associated with a specific location.
 - **Dependency on Geospatial Mapping Services:** Converts GPS coordinates to addresses (dependency).
8. **Offline Functionalities:**
- **Relationships:**
 - **Dependency on System:** The system relies on the **Offline Functionalities** class for offline functionality (dependency).

- **Association with User:** Users benefit from offline functionality during emergencies (association).

9. Geospatial Mapping Services:

- **Relationships:**
 - **Dependency on Location:** Converts incident coordinates to addresses (dependency).
 - **Dependency on Emergency Resources:** Provides maps and navigation for emergency responders (dependency).

10. Multilanguage Support:

- **Relationships:**
 - **Dependency on User:** Users set their preferred language using the **Multilanguage Support** class (dependency).
 - **Dependency on Alert/Notification:** Translates alerts into the user's chosen language (dependency).

11. Category:

- **Relationships:**
 - **Association with Incident Reporting:** Incidents are categorized using the **Category** class (association). Each incident belongs to a specific category.
 - **Dependency on Preparedness and Mitigation:** Categories align with specific preparedness tips (dependency).

5.3.3. Class Diagram of a Disaster Management Mobile Application

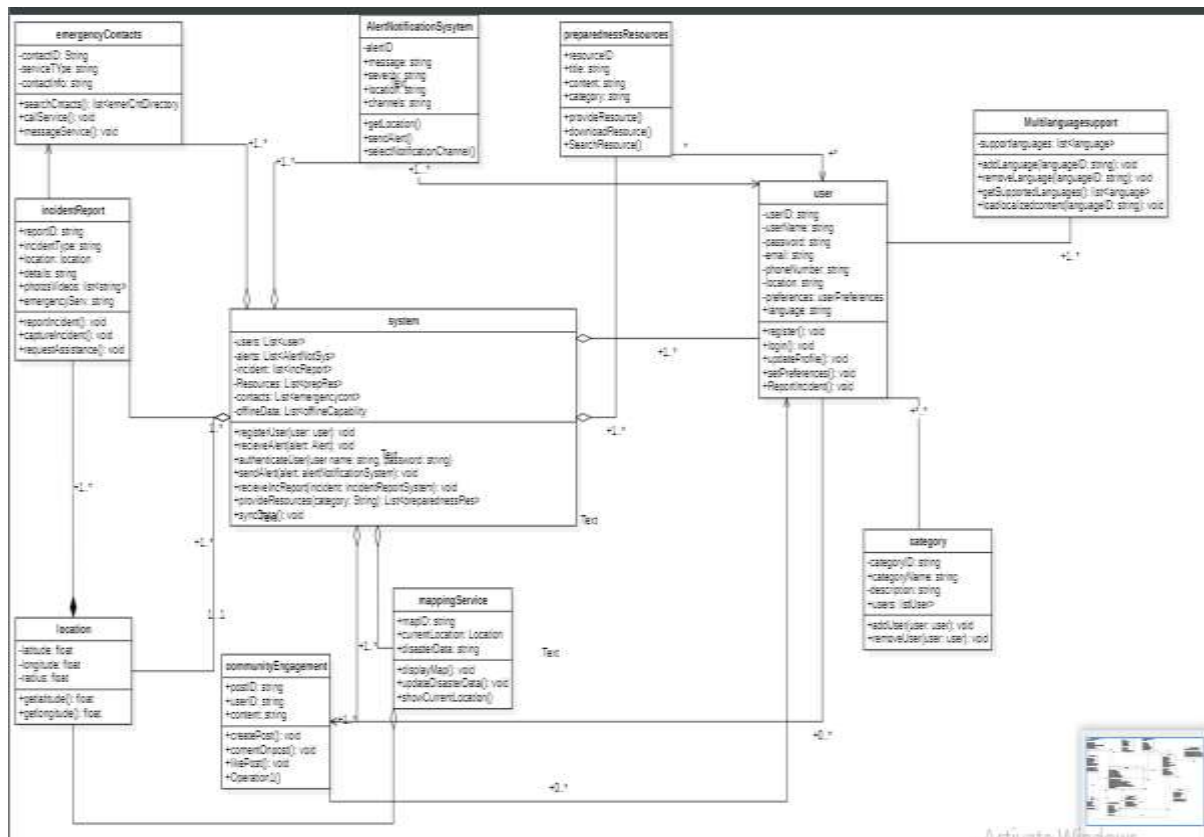


Figure 3: class diagram

These relationships define how classes interact within the system, ensuring effective communication and functionality.

5.4. SEQUENCE DIAGRAM

A **sequence diagram** for a disaster management mobile app illustrates the interactions and order of events between different components or actors within the system in our context, we will have the following **objects**:

5.4.1. Objects, functions and Sequential Procedure

i. User:

who registers in the application, by proving the required credentials; name, Gmail, location, password and etc. the user can login to the system, and he is capable of

receiving alerts/notifications from the system, in addition, the user can view displays options from the system when he/she login. The system in return prompts the user to select an option, while being able to access emergency resources/responders and also able to report incidents based on location and category.

ii. **System:**

- The **system** refers to the entire software infrastructure that supports the disaster management app. It includes the server, databases, APIs, and client-side applications (mobile app and web interfaces).
- Responsibilities:
 - Handles creation of new user account.
 - Handle user authentication and authorization.
 - Manage data storage and retrieval.
 - Coordinate communication between different components.
 - Ensure overall system reliability and scalability.

iii. **Alert/Notification:**

- The **alert/notification** component is responsible for informing users and emergency responders about critical events.
- Responsibilities:
 - Send real-time alerts to users during emergencies (e.g., push notifications).
 - Notify emergency responders about incidents.
 - Provide customizable alert settings for users.

iv. **Emergency Resources/Responders:**

- a. The **emergency resources/responders** component manages the availability and deployment of emergency personnel and resources.
- b. Responsibilities:
 - i. Maintain a database of available responders (firefighters, police, medical personnel).
 - ii. Assign responders to incidents based on their proximity and expertise.

- iii. Track their status and location during an emergency.
- v. **Incident Reporting:**
 - a. The **incident reporting** feature allows users to report emergencies or incidents.
 - b. Responsibilities:
 - i. Capture incident details (type, location, severity).
 - ii. Enable users to attach photos or videos.
 - iii. Forward incident reports to the server for processing.
- vi. **Category:**
 - a. The **category** component organizes incidents into predefined categories (e.g., fire, flood, earthquake).
 - b. Responsibilities:
 - i. Classify incidents based on their nature.
 - ii. Help emergency responders prioritize their actions.
- vii. **Location:**
 - a. The **location** component deals with geospatial information.
 - b. Responsibilities:
 - i. Retrieve the user's location (GPS coordinates).
 - ii. Display incident locations on maps.
 - iii. Assist responders in navigation.
- viii. **Preparedness and Mitigation:**
 - a. The **preparedness and mitigation** component focuses on proactive measures to reduce disaster impact.
 - b. Responsibilities:
 - i. Provide safety tips and guidelines to users.
 - ii. Educate users on disaster preparedness.
 - iii. Promote community resilience.
- ix. **Multilanguage Support:**
 - a. The **multilanguage support** feature ensures that the app can be used by speakers of different languages.
 - b. Responsibilities:
 - i. Offer language selection during app setup.
 - ii. Translate alerts, instructions, and user interfaces.

- iii. Facilitate communication between responders and users who speak different languages.
- x. **Offline Functionalities:**
 - a. The **offline functionalities** allow the app to work even when there's no internet connection.
 - b. Responsibilities:
 - i. Cache essential data (maps, emergency contacts, safety guidelines).
 - ii. Enable users to report incidents offline (data syncs when online).
- xi. **Geospatial Data:**
 - a. The **geospatial data** component handles geographic information.
 - b. Responsibilities:
 - i. Store and retrieve maps, and geospatial layers.
 - ii. Overlay incident locations on maps.
 - iii. Calculate distances and travel times.
- xii. **System Updates:**
 - a. The **system updates** component ensures that the app remains secure and up-to-date.
 - b. Responsibilities:
 - i. Regularly check for app updates.
 - ii. Install security patches.
 - iii. Notify users about new features or improvements.

Based on the above objects and their step-to-step relations of each other, **draw.io software** is used to realize the below sequence diagram.

5.4.3. Sequence Diagram of a disaster management mobile application.

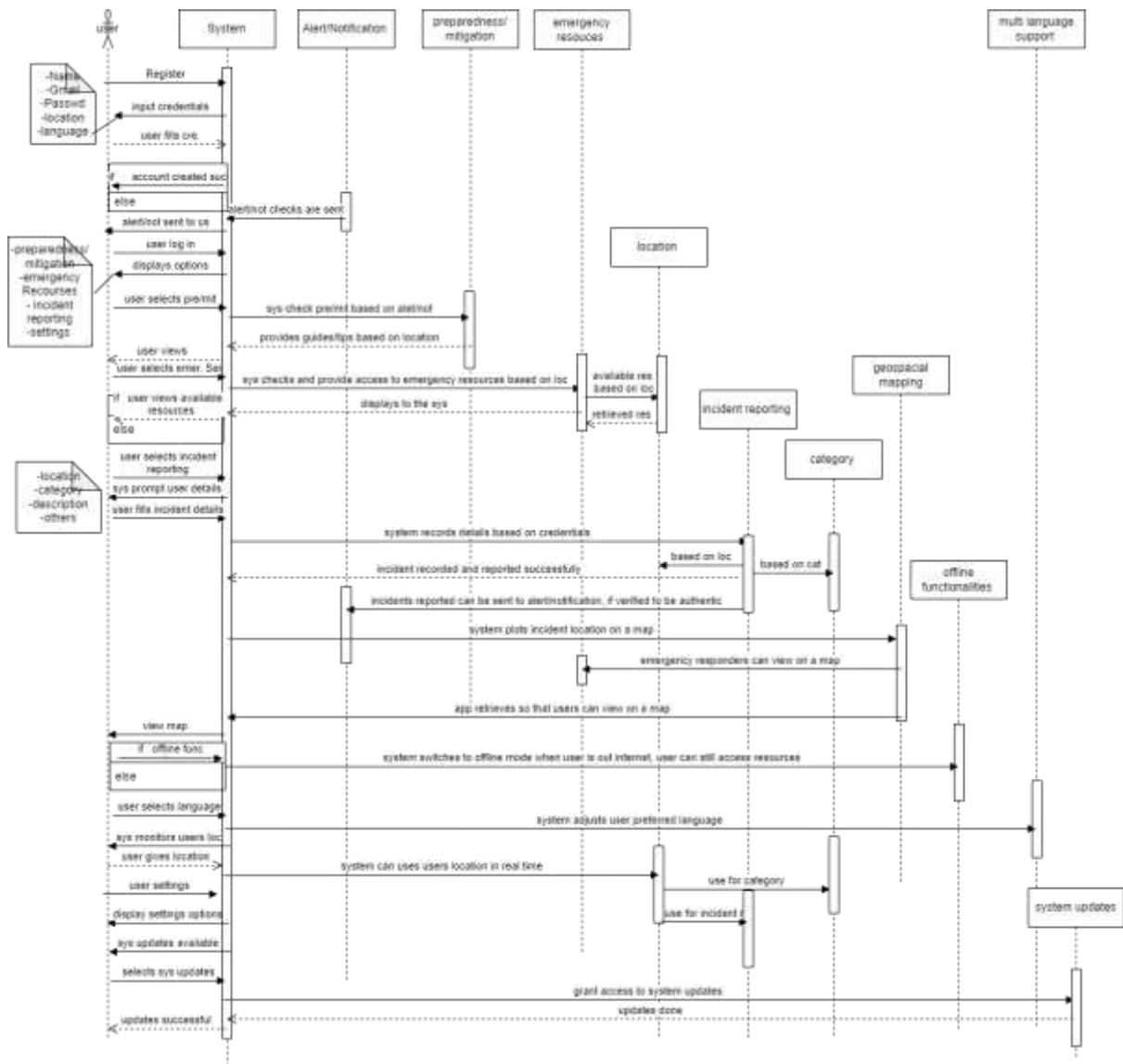


Figure 4: sequence diagram

5.5. DEPLOYMENT DIAGRAM

A **deployment diagram** in the context of a disaster management system mobile application illustrates how software components and hardware nodes interact and are distributed across a network or infrastructure. Let's break down the key aspects of a deployment diagram for such an application:

5.5.1. Components:

1. **Users:** These are the individuals who will utilize the disaster management mobile application. They interact with the app's features to receive alerts, access resources, and potentially report information during emergencies.
2. **Mobile Network:** This network provides internet connectivity for users' devices to communicate with the application server.
3. **React Native App (Frontend):**
 - Developed using the React Native framework, this app runs on users' smartphones and tablets.
 - It utilizes various React Native components to build the user interface (UI) for functionalities like:
 - Displaying disaster alerts and updates.
 - Allowing users to access resources like emergency shelters, evacuation routes, and contact information for aid organizations.
 - Potentially enabling features for reporting damage, missing persons, or resource needs (depending on app design).
 - The app communicates with the Node.js API server using well-defined APIs (Application Programming Interfaces) that follow protocols like REST (Representational State Transfer).
4. **Content Delivery Network (CDN) (Optional):**
 - This can be integrated to improve app performance by caching static content (images, Javascript files) geographically closer to users. This reduces latency and improves loading times, especially in remote areas.
5. **Node.js API Server (Backend):**
 - Developed using Node.js and Express.js framework, this server acts as the intermediary between the mobile app and the database.

- It handles incoming API requests from the mobile app and performs the following actions:
 - Validates user requests and performs authentication checks (if applicable).
 - Interacts with the MongoDB database to retrieve or store data relevant to the request (e.g., disaster alerts, resource locations).
 - Processes data as needed (e.g., filtering, aggregation) before sending responses back to the mobile app.
 - May handle real-time communication functionalities (push notifications) using technologies like websockets or Firebase Cloud Messaging (FCM).

6. API Gateway (Optional):

- This can be introduced as an additional layer of security and management for API access. It acts as a single entry point for API requests, offloading security concerns from the Node.js server.
- It can handle features like:
 - Rate limiting to prevent abuse of the API.
 - Authentication and authorization for different user roles.
 - API usage analytics.

7. Cloud Load Balancer:

- This service distributes incoming traffic across multiple instances of the Node.js server for scalability and fault tolerance. It ensures that the application remains responsive even during high user traffic situations.

8. MongoDB Database:

- This NoSQL database (MongoDB Atlas in the cloud) stores all the application's data relevant to disaster management. This includes:
 - Disaster information (type, severity, location)
 - Resource information (shelters, evacuation routes, aid organizations)
 - User information (if applicable) for functionalities like reporting or managing user profiles.

5.5.2. Communication Flow:

1. **User Interaction:** Users interact with the mobile app's UI elements, triggering actions that send API requests to the Node.js server.
2. **API Request:** The mobile app formulates an API request containing relevant data (e.g., user location, disaster type) and sends it over the internet through the mobile network.
3. **Content Delivery Network (Optional):** If a CDN is implemented, it might intercept the request and serve cached static content if available. This reduces load on the server and improves app performance.
4. **API Gateway (Optional):** The API Gateway receives the request, performs any necessary security checks (authentication, authorization), and routes it to the appropriate backend service (Node.js server).
5. **Node.js Server:** The Node.js server receives the request, validates it, and interacts with the MongoDB database.
 - It retrieves or stores data as needed based on the request.
 - It may perform data processing or manipulation before preparing a response.
6. **Response:** The Node.js server formulates a response containing the requested data or relevant messages (e.g., success/failure status) and sends it back to the mobile app.
7. **Mobile App Update:** The mobile app receives the response from the server and updates its UI accordingly. It might display retrieved data (e.g., disaster alerts, resource locations) or handle success/failure messages for user actions.

5.5.3. Deployment:

- **Mobile App:** The React Native app is typically compiled for Android platforms and deployed to their respective app stores (Apple App Store and Google Play Store).
- **Node.js Server:** The Node.js server application is deployed to a cloud platform that offers hosting services like

Based on the communication flow, **draw.io software** is used to realize the below deployment diagram.

5.5.4. Deployment diagram of a disaster management system.

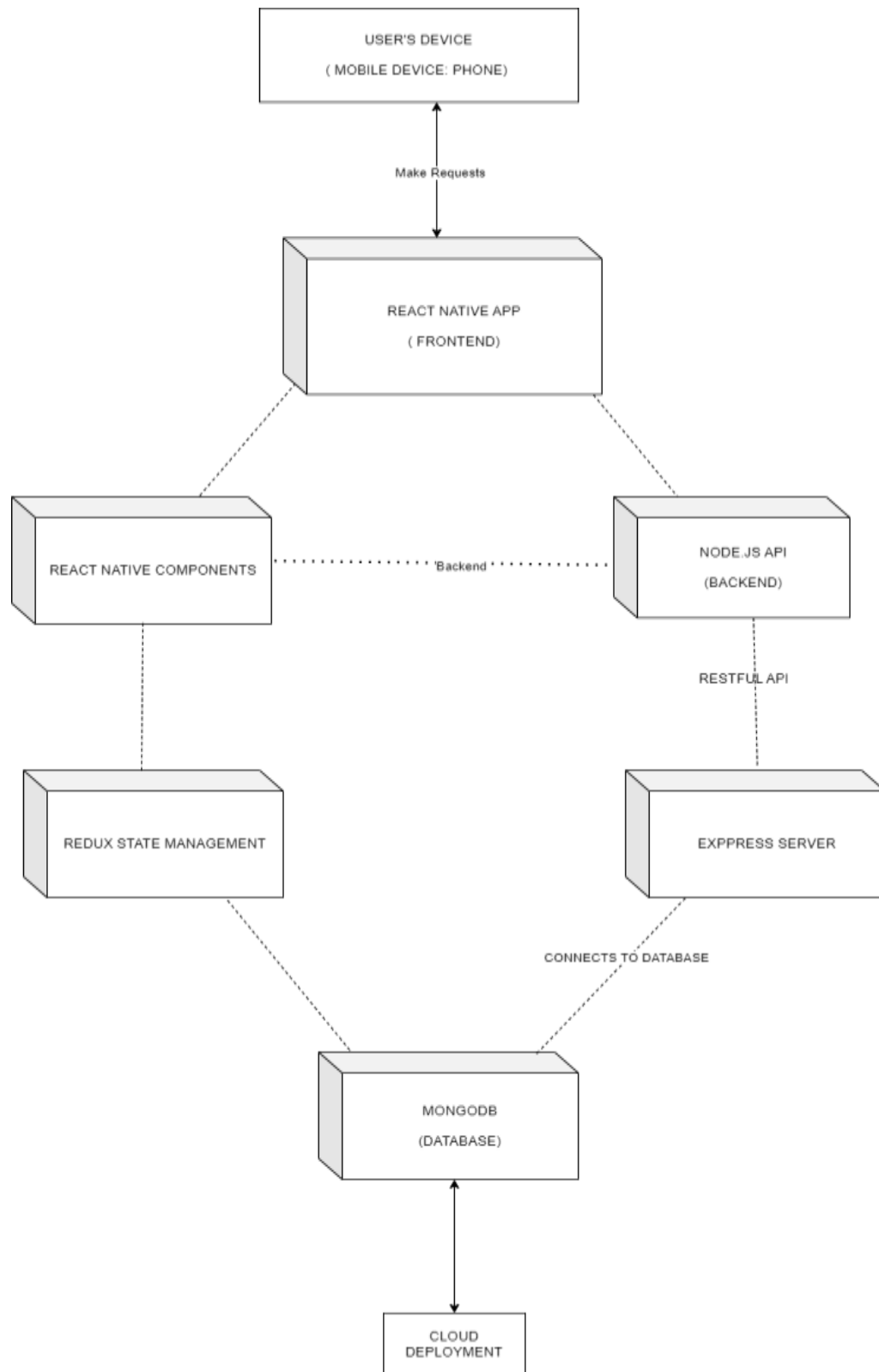


Figure 5: deployment diagram

This detailed deployment diagram provides a comprehensive view of the architecture for a disaster management mobile application built with React Native, Node.js, and MongoDB. By utilizing these technologies and following a well-structured approach, developers can create a robust and scalable application that empowers users with critical information and functionalities during emergencies.

Key benefits of this architecture:

- **Improved User Experience:** React Native allows for a native-like mobile app experience across different platforms.
- **Scalability and Performance:** Cloud deployment with load balancing ensures the application can handle high user traffic and maintain responsiveness.
- **Real-time Communication (Optional):** Integration with technologies like websockets or FCM enables real-time features like push notifications for critical updates.
- **Data Management:** MongoDB provides a flexible NoSQL database solution for storing various disaster management data.

Based on the above UML diagrams; context diagram, use case diagram, class diagram, sequence diagram and deployment diagram, the ideal disaster management mobile application task, can be passed across to the next stage, while being open for further corrections when need be.

CHAPTER 6: UI DESIGN AND IMPLEMENTATION FOR A DISASTER MANAGEMENT MOBILE APPLICATION

This design phase outlines the process of creating a disaster management mobile application aimed at providing timely assistance and resources to individuals affected by disasters. Through personas, user journey maps, wireframing, and implementation using Figma, the application seeks to address the diverse needs of users during crisis situations, fostering resilience and community support.

6.1. DESIGN PHASE

PERSONA

The number of personas in the disaster management mobile application depends on the specific target audience and the scope of the application. However, some common personas that we want to consider include:

- Admin (Disaster management officials): Individuals who are responsible for planning and coordinating disaster response and recovery efforts.
- Emergency responders: Emergency personnel who need to use the app to respond to and manage disasters.
- Civilians: Individuals who are at risk of being affected by disasters and need to be informed and prepared.
- People with disabilities: Individuals who may have specific needs or challenges during a disaster.

We will have a practical example with information of the various users of our system's persona below

i. Admin persona

Name: Sarah

- Age: 45
- Occupation: Emergency management director
- Goals: To ensure the safety and well-being of her community before, during, and after disasters.

•Touchpoints:

- Disaster management mobile app Emergency response software

- Social media
- Email
- Phone

Emotions:

* Stressful: When managing a disaster response, Sarah is under a lot of pressure to make quick decisions and coordinate multiple resources.

* Determined: Sarah is committed to protecting her community and ensuring that they have the resources they need to recover from disasters.

* Grateful: Sarah is thankful for the support of her team and the volunteers who help her to manage disaster response efforts.

• Pain points:

* Lack of resources: Sarah often has to make difficult decisions about how to allocate limited resources during a disaster.

* Communication challenges: It can be difficult to communicate with all of the stakeholders involved in disaster response, especially in areas with limited connectivity.

* Public resistance: Sarah sometimes encounters resistance from the public when implementing disaster preparedness measures or evacuations.

ii. Emergency Responder

- Name: John
- Age: 30
- Occupation: Firefighter/paramedic
- Goals: To save lives and property during disasters.

Touchpoints:

- * Disaster management mobile app
- * Emergency response vehicle
- * Radio
- * Flashlight
- * Medical equipment

• Emotions:

* Fearful: John is often exposed to dangerous situations when responding to disasters.

- * Adrenaline-fueled: John gets a sense of adrenaline and excitement when responding to emergencies.

- * Compassionate: John cares deeply about helping others and making a difference in their lives.

- **Pain points:**

- * Lack of information: John sometimes lacks access to real-time information about the disaster situation, which can make it difficult to make decisions.

- * Limited resources: John often has to work with limited resources, which can make it difficult to provide the best possible care to victims.

- * Physical and emotional exhaustion: Disaster response work can be physically and emotionally demanding, and John often has to work long hours in difficult conditions.

iii. Civilian (normal user)

- Name: Mary

- Age: 50

- Occupation: Teacher

- Goals: To keep herself and her family safe during a disaster.

- **Touchpoints:**

- * Disaster management mobile app

- * Social media

- * Television

- * Radio

- **Emotions:**

- * Anxious: Mary is worried about the safety of her family and her community during a disaster.

- * Confused: Mary may not always understand the instructions or information that is provided during a disaster.

- * Grateful: Mary is thankful for the help and support of her community during a disaster.

- **Pain points:**

- * Lack of information: Mary may not have access to accurate or timely information about the disaster situation, which can make it difficult to make decisions.

* Communication challenges: Mary may have difficulty communicating with her family and friends during a disaster, especially if there is limited connectivity.

* Lack of preparedness: Mary may not have the necessary supplies or knowledge to prepare for a disaster.

By understanding the personas, touchpoints, emotions, and pain points of our target users, we can design a disaster management mobile application that meets their needs and helps them to prepare for, respond to, and recover from disasters.

6.2. SCENARIOS OF A DISASTER MANAGEMENT MOBILE APPLICATION

Scenario 1: Registration/Login

- User downloads and opens the disaster management app
- Login if has an account, if not.
- Prompted to create an account for access to emergency resources and information
- Enters email address, creates a password, and provides basic personal information
- Clicks "Register"
- Receives a verification email
- Clicks verification link in the email to confirm their email address
- Redirected back to the app and prompted to log in with their email and password
- Enters credentials and clicks "Login"
- Gains access to real-time alerts, emergency resources, and communication tools

Scenario 2: Preparedness Before a Disaster

- User receives notifications about potential threats and prepares an emergency plan.
- User identifies nearby shelters, evacuation routes, and resource centers using the app.
- User shares their emergency contact information and medical history with trusted individuals.

Scenario 3: Alerts/Notifications During a Disaster

- User receives real-time emergency alerts and updates on the disaster's progress.
- User uses the app to navigate evacuation routes and find the nearest shelter.
- User communicates with family and friends using the app's messaging or social media integration.

Scenario 4: After a Disaster (recovery)

- User uses the app to locate nearby resources such as food banks, medical facilities, and counselling services.
- User connects with other survivors and volunteers to offer or receive assistance.
- User provides feedback on the disaster response and recovery efforts through the app.

Scenario 5: User with Disabilities

- User with visual impairment uses the app's screen reader functionality to access information and navigate the interface.
- User with hearing impairment uses closed captions for videos and text-based alerts.
- User with cognitive disabilities uses simplified language and clear visual cues to understand the app's functionality.

Scenario 6: User in a Remote Area (offline functionality)

- User with limited internet connectivity downloads essential data and resources offline before a disaster strikes.
- User uses the app's offline maps and GPS functionality to navigate evacuation routes.
- User communicates with emergency services via text messaging or satellite connection.

Scenario 7: Emergency Responders

Emergency Responders uses the app to access real-time disaster information, including incident reports and resource availability.

- Emergency Responders uses the app to communicate with other responders and coordinate relief efforts.
- Emergency Responders uses the app to document and share damage assessments.

Scenario 8: Incident Reporting

- User witnesses a disaster event and uses the app to report it to emergency services.
- User provides details about the incident, including location, type of disaster, and severity.
- User attaches photos or videos as evidence to support the report.
- Emergency services receive the report and dispatch responders accordingly.
- User receives updates on the incident's status and response efforts through the app.

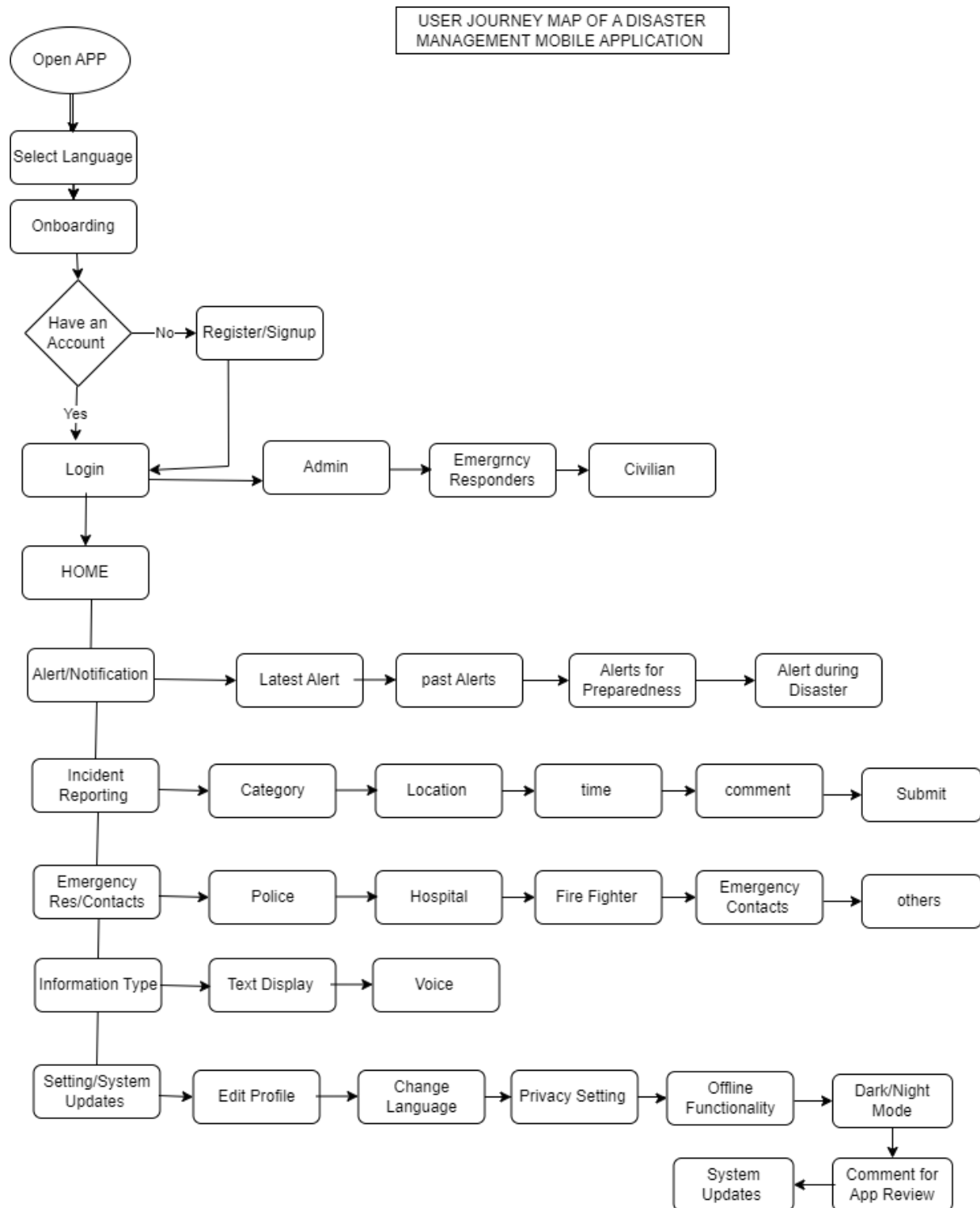
Scenario 9: Settings and Updates

- User customizes the app's settings to receive alerts for specific disaster types and locations.
- User sets up emergency contacts and shares their location with trusted individuals.
- User enables offline functionality to ensure the app can be used even without internet connectivity.
- User receives notifications about app updates and new features.

- User provides feedback on the app's functionality and suggests improvements.

6.3. USER JOURNEY MAP

Based on the above Personas and various Scenarios, we now develop the user journey map of our system.



6.4. MAP VISUALISATION THROUGH WIRE FRAMING

Based on the above user flow chart, the following main Screens are to be consider, which in return pops up or drop down other sub-screens, depending on the user's perspectives and wants.

i. Wire framing objects and attributes of the main Screens.

1. open App Onboarding Screen Wireframe:

[Logo]

[select language]

[Title:]

[welcome Field]

[introduction to features]

2. Registration Screen Wireframe:

[Logo]

[Title: Create Account]

[category of user]

[Email Address Input Field]

[Password Input Field]

[Confirm Password Input Field]

[First Name Input Field]

[Last Name Input Field]

[Phone Number Input Field]

[Register Button]

3. Verification Email Screen Wireframe:

[Logo]

[Title: Verify Your Email]

[Instructional Text: Please check your email and click the verification link to complete the registration process.]

[Resend Verification Email Button]

3. Login Screen Wireframe:

[Logo]

[Title: Log In]

[category of user]

[Email Address Input Field]

[Password Input Field]

[Forgot Password Link]

[Login Button]

4. Home Screen Wireframe

[Logo]

[Title: home]

[alerts]

[Report Incident]

[Emergency Responders contacts]

[type of message]

[setting/updates]

5. Preparedness Before a Disaster Wireframe:

[Logo]

[Title: Preparedness Before a Disaster]

[Checklist of Preparedness Items]

[Emergency Contact Information Input Fields]

[Emergency Kit Checklist]

[Evacuation Plan Input Field]

[Save and Update Button]

6. Alerts and Notifications During Disaster Wireframe:

[Logo]

[Title: Alerts and Notifications]

[Map with Disaster Location and Affected Areas]

[Emergency Alert Notifications]

[Weather Updates]

[Shelter Locations and Availability]

[Emergency Contacts Information]

7. Recovery After a Disaster Wireframe:

[Logo]

[Title: Recovery After a Disaster]

[Recovery Resources and Services]

[Community Support Information]

[Insurance Claim Filing Assistance]

[Rebuilding Checklist]

[Support Group Information]

8. Offline Capabilities Wireframe:

[Logo]

[Title: Offline Capabilities]

[Offline Mode Toggle Switch]

[Downloadable Emergency Resources]

[Offline Access to Emergency Contacts]

[Saving Incident Reports Locally]

9. Emergency Responders Wireframe:

[Logo]

[Title: Emergency Responders]

[Select Emergency Responders needed]

[Emergency Services Contact Information]

[Request Assistance Button]

[Location Sharing with Responders]

[Real-time Communication with Responders]

10. Incident Reporting Wireframe:

[Logo]

[Title: Incident Reporting]

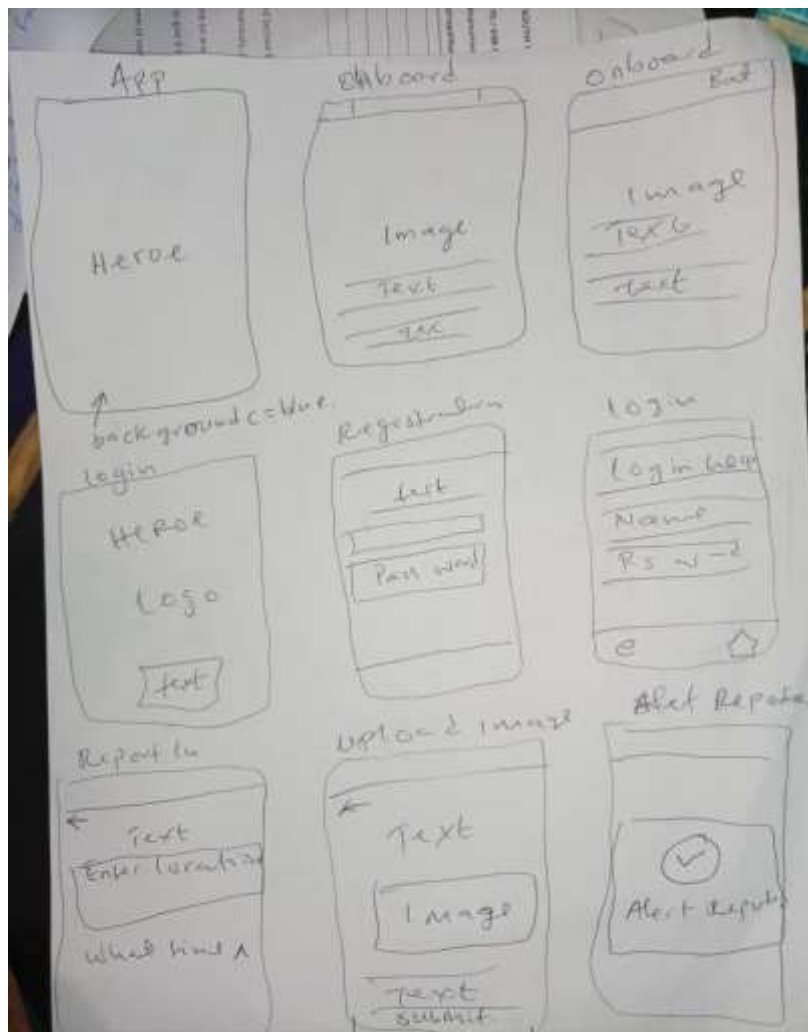
[Incident Type Selection]
[Location Tagging, time]
[Description of Incident Input Field]
[Upload Image/Video Evidence]
[Submit Report Button]

11. Settings and Updates Wireframe:

[Logo]
[Title: Settings and Updates]
[Profile Information Update Fields]
[Notification Settings]
[Language Selection Option]
[App Version Update Notification]

These wireframes provide a basic structure for the various features of the disaster management app. we further enhance them with visual elements, branding, and additional features as needed.

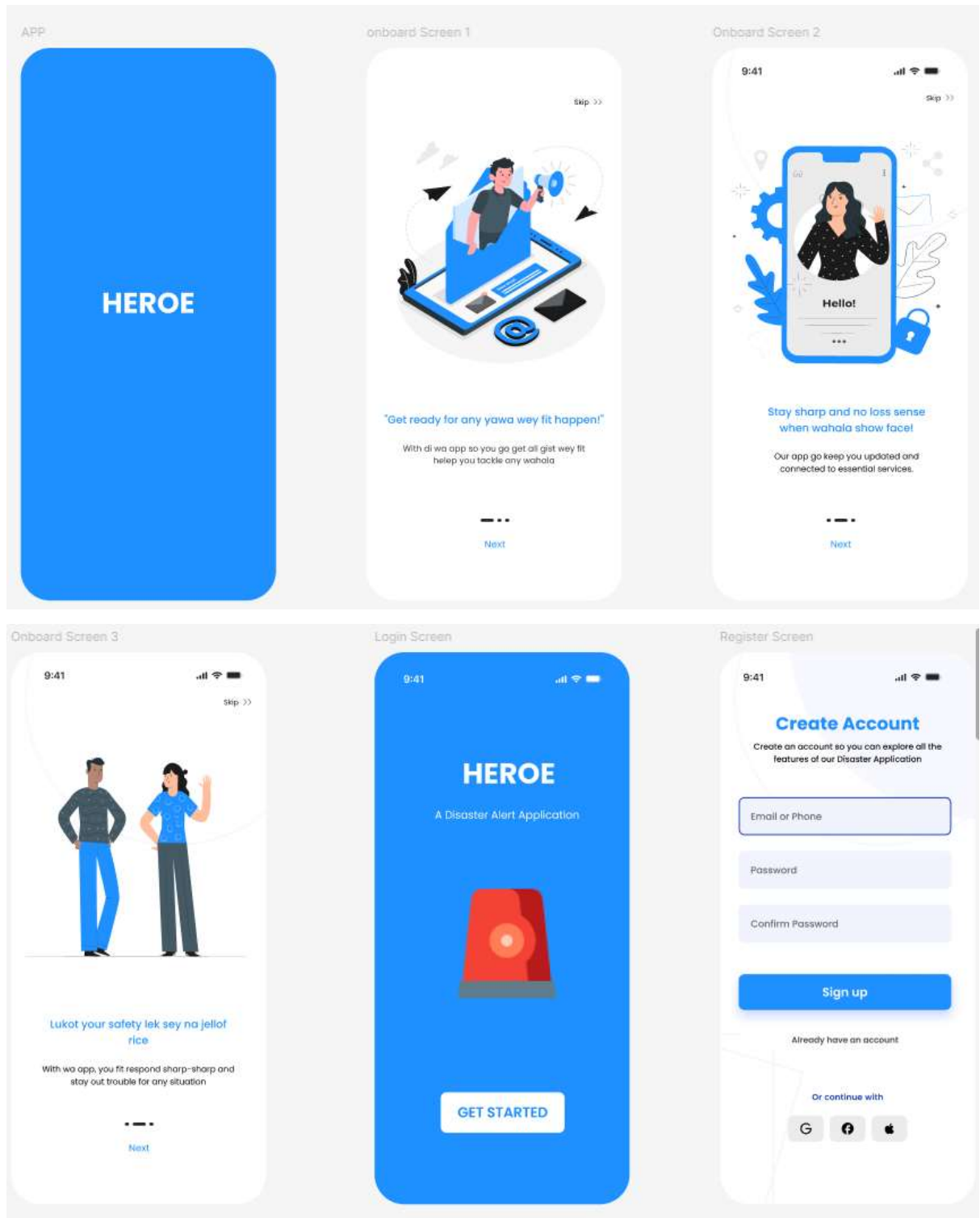
ii. Wire framing sketches

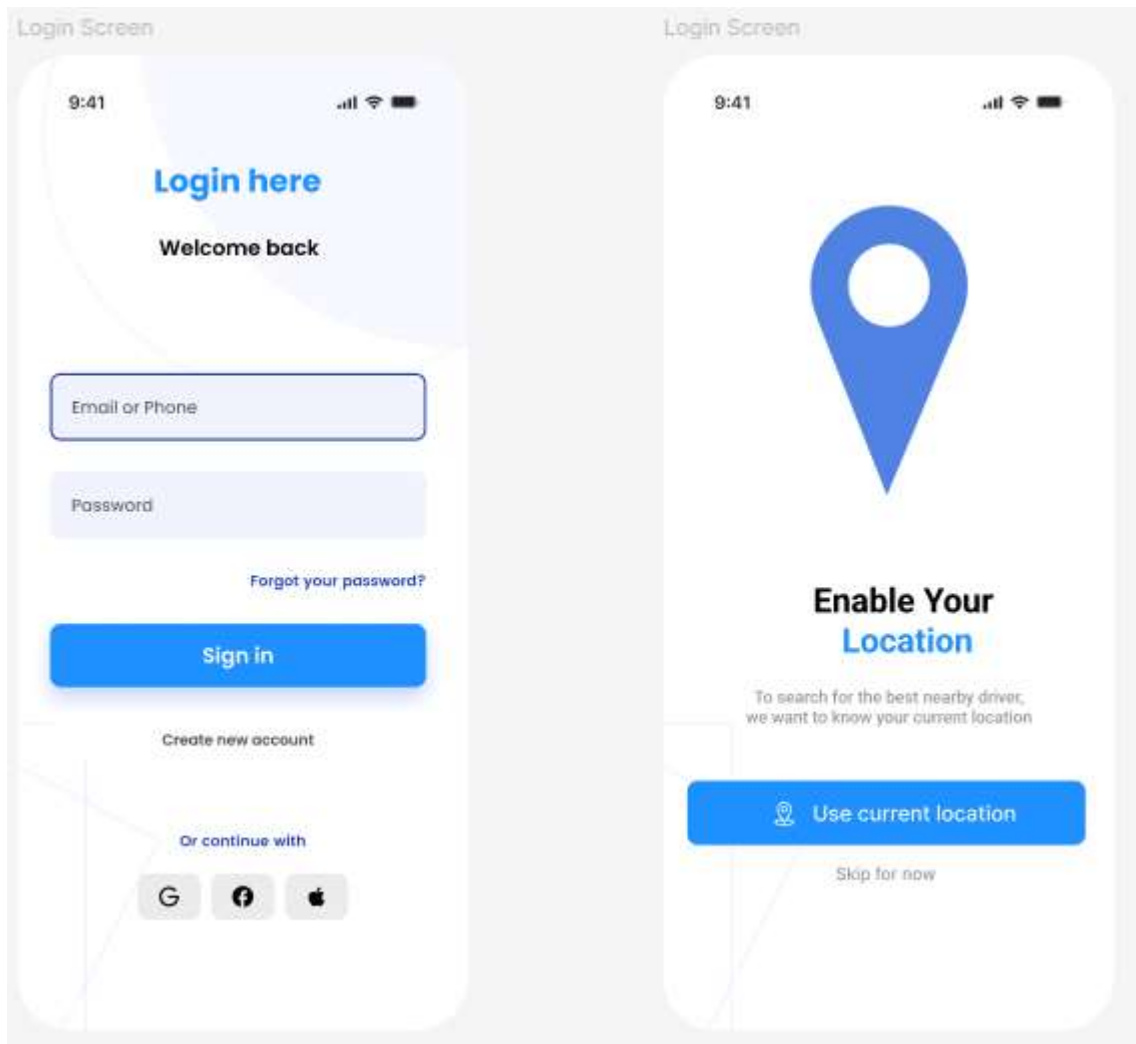


Using the 10 design principles, which are: simplicity, consistency, Hierarchy, Balance, Emphasis, unity, contrast, Functionality, Accessibility, Feedback, and the user centered design, we use colours and images in their appropriate positions, and backgrounds.

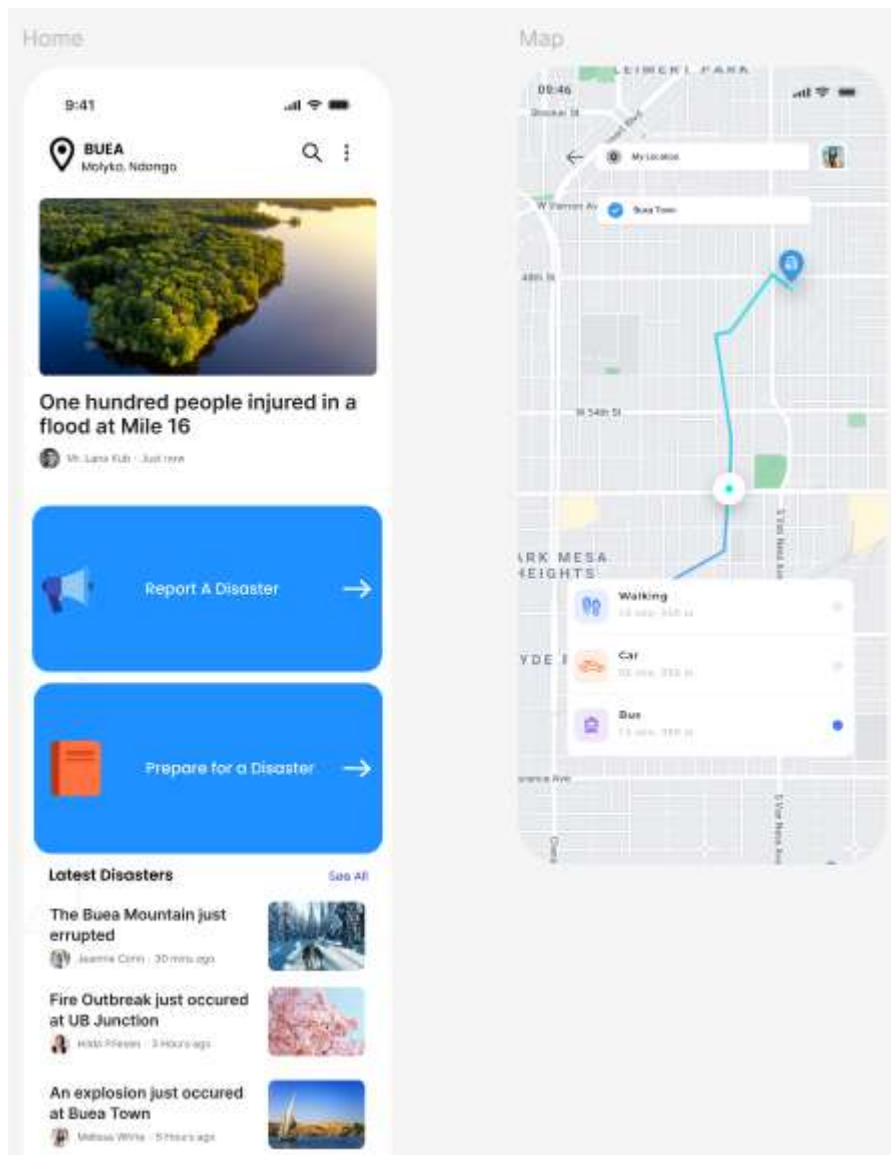
6.5. IMPLEMENTATION WITH FIGMA.

i. Registration/Login

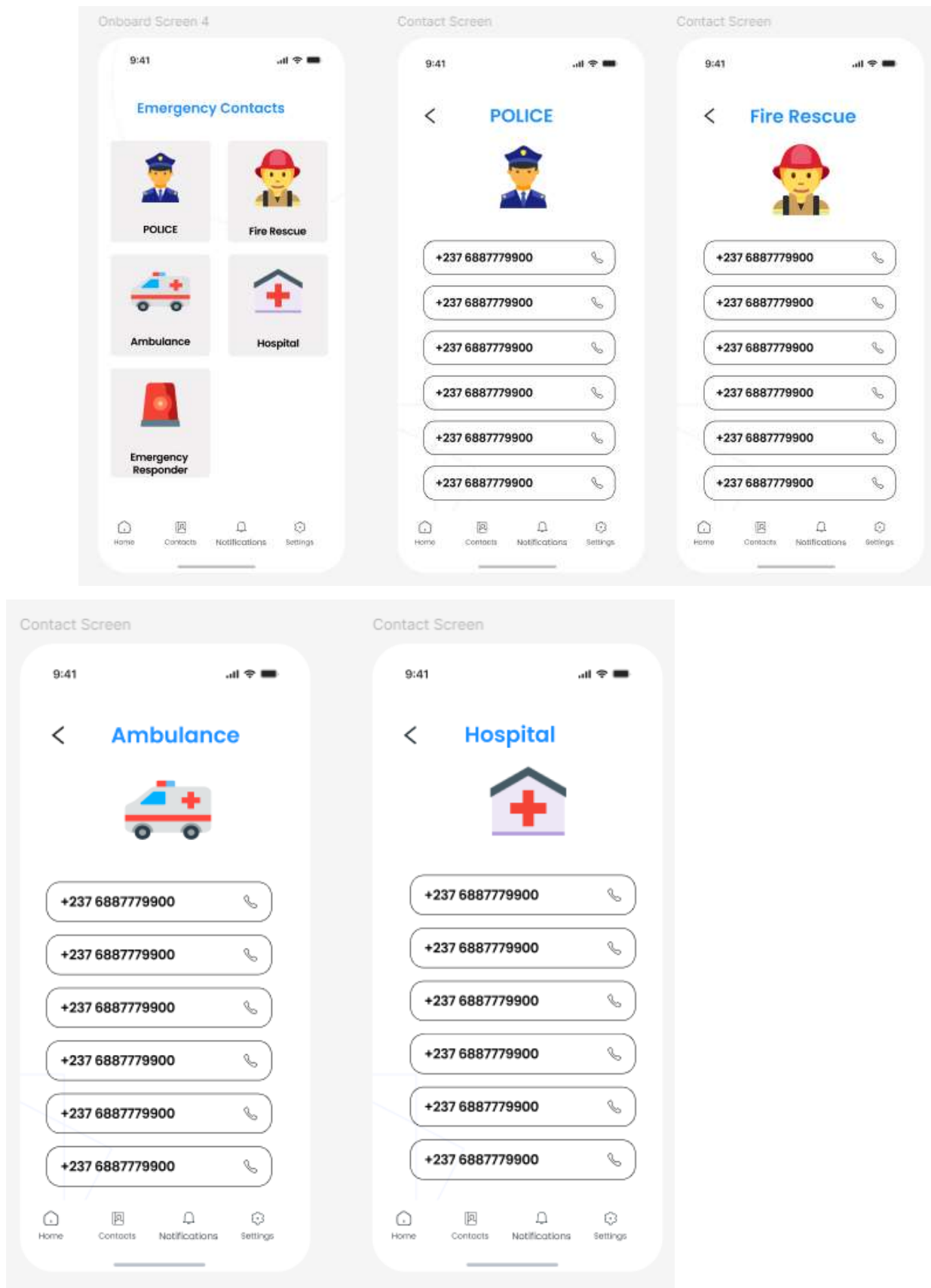




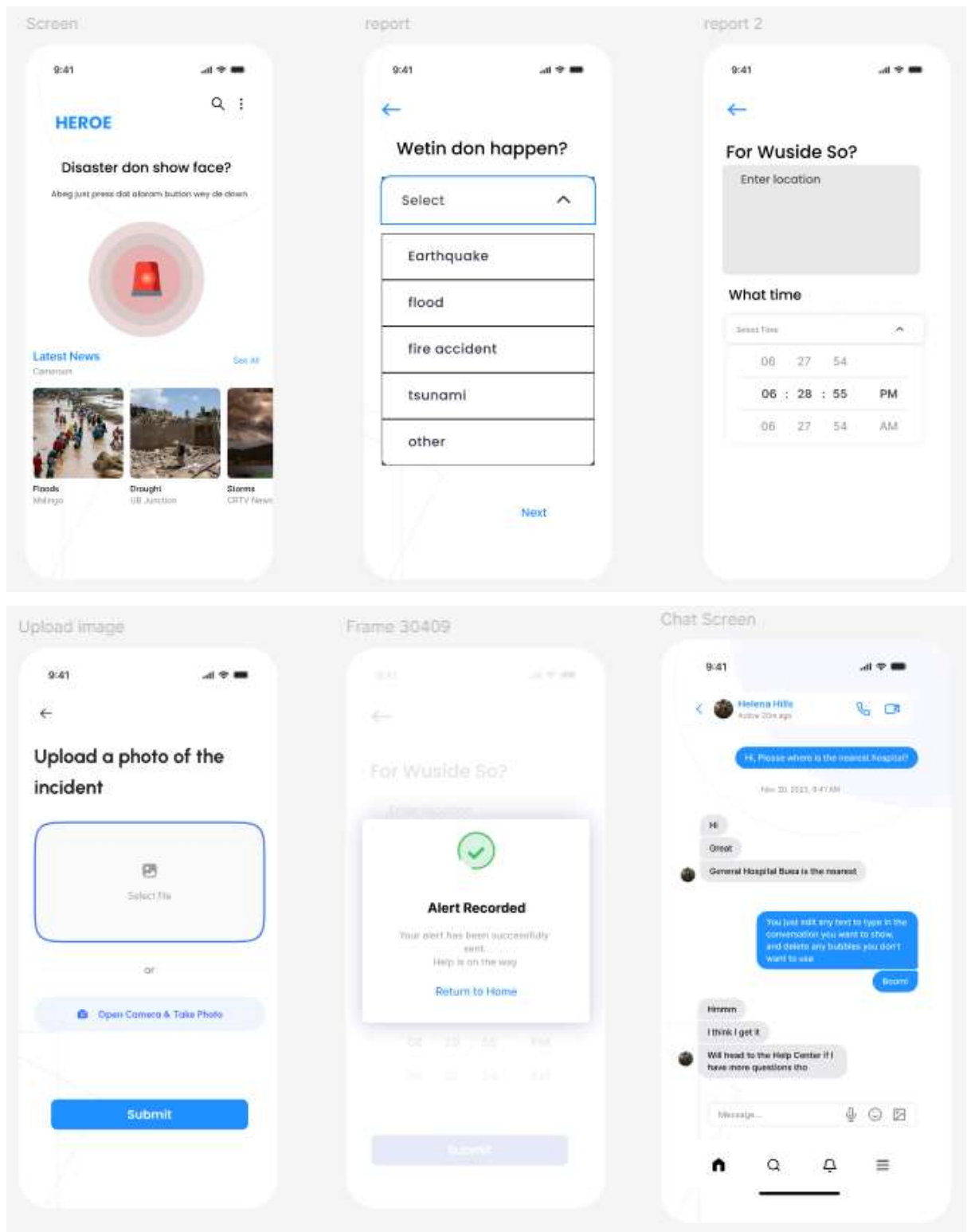
ii. Home screen



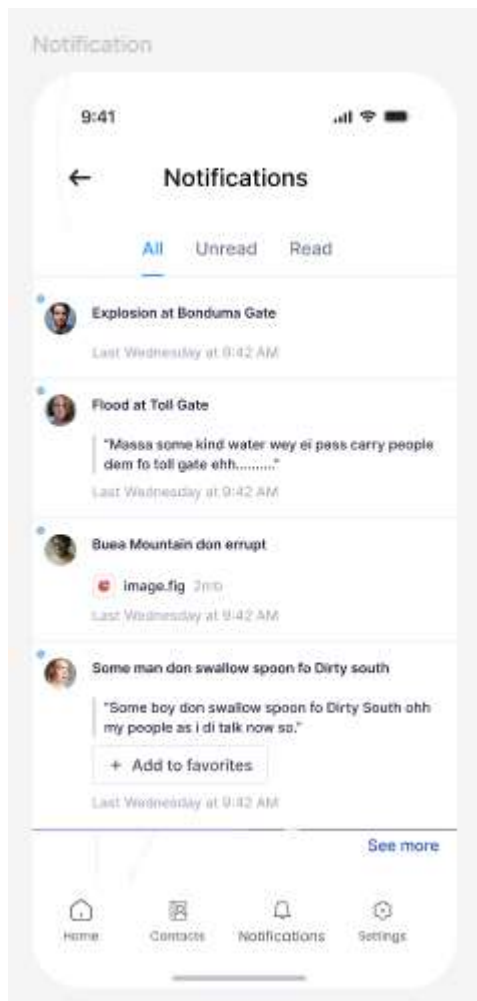
iii. Emergency Responders Contact



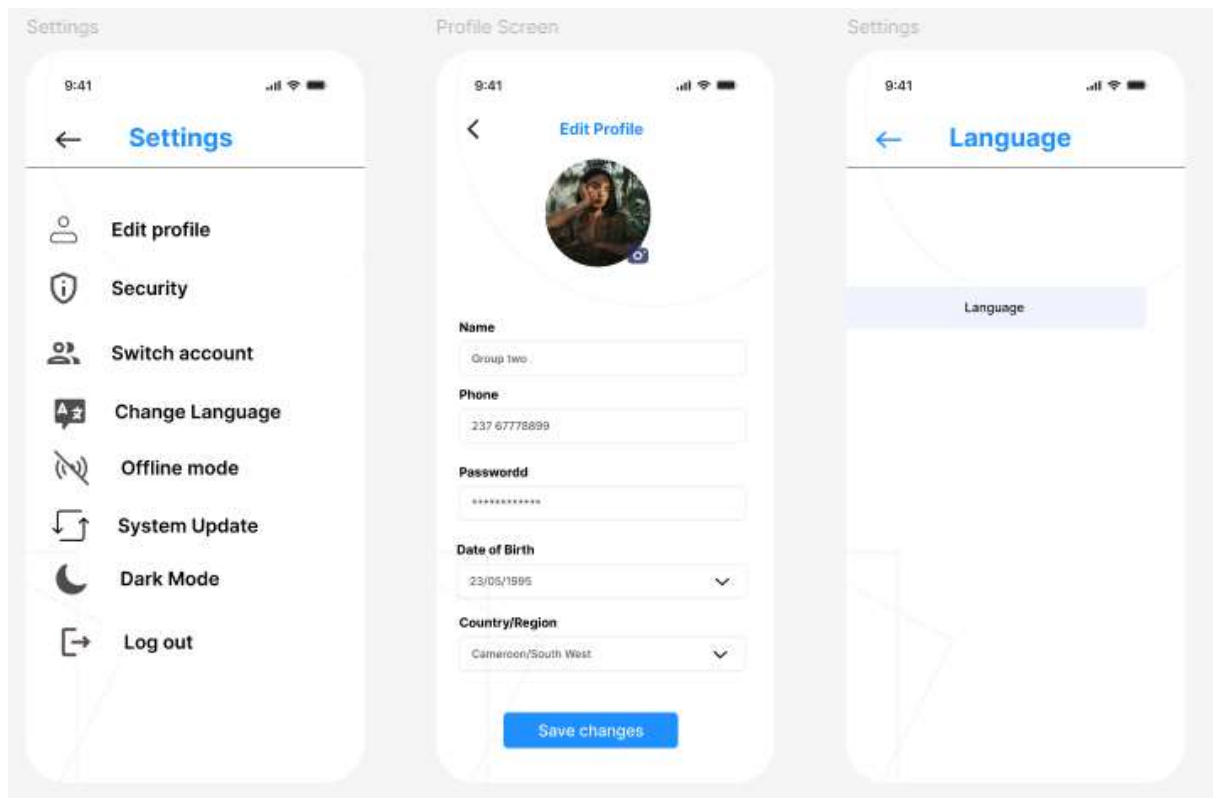
iv. Incident reporting.



v. Alerts/Notifications



vi. Setting/Updates



The design of the disaster management mobile application is based on a thorough understanding of the needs of users, as identified through user research, personas, scenarios, and a user journey map. The application is designed to be easy to use, accessible to all users, and effective in helping users to prepare for, respond to, and recover from disasters.

We plan to continue to develop and improve the application based on feedback from users and stakeholders. We are committed to making the application as user-friendly and effective as possible.

CHAPTER 7: DATABASE DESIGN AND IMPLEMENTATION

In this phase, we will design and implement a database for a disaster management system using Firebase Fire store. Our system is intended to manage various types of data, including user profiles, emergency responder contacts, real-time alerts, incident reports, locations, language preferences, and preparedness resources.

7.1. Overview

The disaster management system aims to:

- Enable users to report incidents and receive real-time alerts.
- Store and manage contact information for emergency responders.
- Track user locations during incidents.
- Support multilingual communication.
- Provide users with access to preparedness resources.

7.2. PROBLEM STATEMENT/ REQUIREMENT: DESIGNING DATABASE FOR A DISASTER MANAGEMENT MOBILE APPLICATION

We are tasked with designing and implementing a disaster management mobile application database that assists users during emergencies. The application will provide real-time information, facilitate communication, and enhance coordination among users, emergency responders, and relevant authorities. Below are the key requirements:

7.2.1. User Registration and Authentication:

Users can create accounts and log in securely.

Implement authentication mechanisms (e.g., OAuth, JWT) to protect user data.

7.2.2. Entities and Relationships:

User: Represents individuals using the app. Users can report incidents, receive alerts, and manage their profiles.

Emergency Responder Contacts: Stores contact information for emergency responders (e.g., police, fire department, medical services).

Alert/Notification: Allows sending emergency alerts to users based on their location and preferences.

Location: Tracks user location during incidents.

Incident Report: Captures details about incidents (e.g., natural disasters, accidents).

Language: Supports multilingual communication.

7.2.3. **Functionality:**

Incident Reporting: Users can submit incident reports with relevant details (location, severity, type).

Alerts and Notifications: Send real-time alerts to users based on their proximity to incidents.

Emergency Contacts: Users can access emergency responder contacts.

Multilingual Support: Provide information in different languages.

Map Integration: Display incident locations on a map.

User Profiles: Allow users to manage their profiles and preferences.

7.3. **CONCEPTUAL DESIGN: ER DIAGRAM**

In the context of a **disaster management mobile application**, an **Entity-Relationship Diagram (ERD)** is a visual representation of different entities within the system and how they

relate to each other. Below is a simplified representation of the entities and their relationships:

7.3.1. **User: Entities:**

User: Represents individuals using the app. Users can create accounts, log in, and manage their profiles.

Attributes: UserID (Primary Key), Username, Email, Password, LanguagePreference

- Relationships: One-to-Many with Incident (User can report multiple incidents)

7.3.2. **Emergency Responder Contacts:**

Emergency Responder Contacts: Stores contact information for emergency responders (e.g., police, fire department, medical services).

Attributes: ResponderID (Primary Key), ResponderName, ResponderType (e.g., police, fire, medical), Phone

Relationships: None (standalone entity)

7.3.3. **Alert/Notification:**

Alert/Notification: Allows sending real-time emergency alerts to users based on their location and preferences.

Attributes: AlertID (Primary Key), Message, Timestamp, Location

Relationships: Many-to-One with User (User receives multiple alerts)

7.3.4. **Location:**

Location: Tracks user location during incidents.

Attributes: LocationID (Primary Key), Address

Relationships: One-to-Many with Incident (Incidents occur at specific locations)

7.3.5. Incident Report:

Incident Report: Captures details about incidents (e.g., natural disasters, accidents).

Attributes: IncidentID (Primary Key), Type (e.g., earthquake, flood), Severity, Description, Timestamp, photo, video.

Relationships: Many-to-One with User (User reports incidents), Many-to-One with Location (Incident occurs at a specific location)

7.3.6. Language:

Language: Supports multilingual communication.

Attributes: LanguageID (Primary Key), LanguageName

Relationships: Many-to-Many with User (User preferences for multilingual support)

7.3.7. PreparednessResources

Attributes: ResourceID: A unique identifier for each preparedness resource.

ResourceType: Describes the type of resource (e.g., guide, checklist, emergency kit).

Description: Provides details about the resource (e.g., content, steps, recommendations).

Relationships: User-PreparednessResources: Users can access multiple preparedness resources and each resource can be accessed by multiple users.

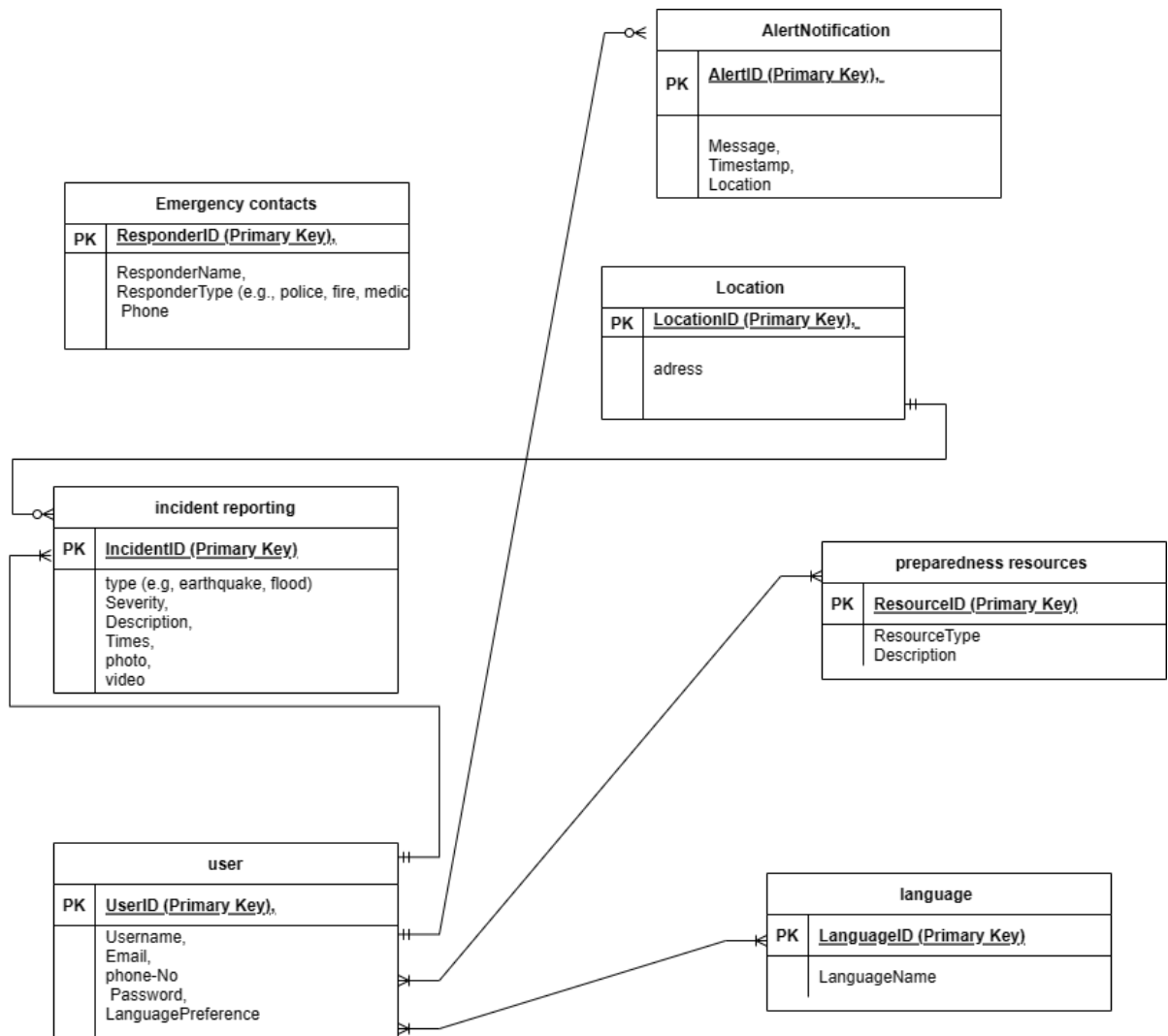


Figure 6: ER Diagram

7.4. IMPLEMENTATION: FIREBASE (LOGICAL AND PHYSICAL DESIGN)

Firebase is a cloud-based platform that offers a suite of services for mobile and web app development. One of its key components is the Firebase Realtime Database, which is a NoSQL database that provides real-time data synchronization.

Firebase Realtime Database provides a powerful, flexible, and easy-to-use NoSQL database with real-time data synchronization, making it ideal for building modern mobile and web applications that require instant data updates and seamless user experiences

i. Data Modeling:

> Document Structure: How will we structured our documents? We define the fields and their types (string, number, boolean, etc.) that represent your data.

> Relationships: How will you link documents together? You'll use references within documents to establish relationships. For example, a User document might contain a reference to a Post document.

ii. Data Flow and Relationships:

* How Data is Accessed and Updated: How does our app interact with the database? What operations (read, write, delete) are needed?

* Data Relationships: What are the relationships between different types of data? Think about how users, events, resources, or other entities in your app are related.

iii. Security Rules:

* Data Access Control: Who has access to different parts of your database? You need to define security rules in Firebase to control read and write permissions, ensuring data integrity and security.

v. Relationships

- User and Incident Report: One user can report multiple incidents.
- User and Alert/Notification: One user can receive multiple alerts.
- Incident Report and Location: Each incident occurs at a specific location.
- User and Language: Users can have preferences for multiple languages.
- User and Preparedness Resources: Users can access multiple preparedness resources and each resource can be accessed by multiple users.

vi. Mapping to Firebase Firestore

User Collection

- Attributes:
 - UserID: Automatically generated by Firestore as Document ID
 - Username
 - Email
 - Password
 - LanguagePreference: Array of LanguageIDs
- Subcollections:
 - IncidentReports (Documents representing individual incidents reported by the user)
 - Alerts (Documents representing alerts received by the user)
 - PreparednessResources (Documents representing resources accessed by the user)

Emergency Responder Contacts Collection

- Attributes:
 - ResponderID: Automatically generated by Firestore as Document ID
 - ResponderName
 - ResponderType
 - Phone

Alert/Notification Collection

- Attributes:
 - AlertID: Automatically generated by Firestore as Document ID
 - Message
 - Timestamp
 - Location: Reference to Location document

Location Collection

- Attributes:
 - LocationID: Automatically generated by Firestore as Document ID
 - Address

Incident Report Collection

- Attributes:
 - IncidentID: Automatically generated by Firestore as Document ID
 - Type
 - Severity
 - Description
 - Timestamp
 - UserID: Reference to User document
 - LocationID: Reference to Location document

Language Collection

- Attributes:
 - LanguageID: Automatically generated by Firestore as Document ID
 - LanguageName

Preparedness Resources Collection

- Attributes:
 - ResourceID: Automatically generated by Firestore as Document ID
 - ResourceType

- Description
- Subcollections:
- Users (References to User documents accessing this resource)

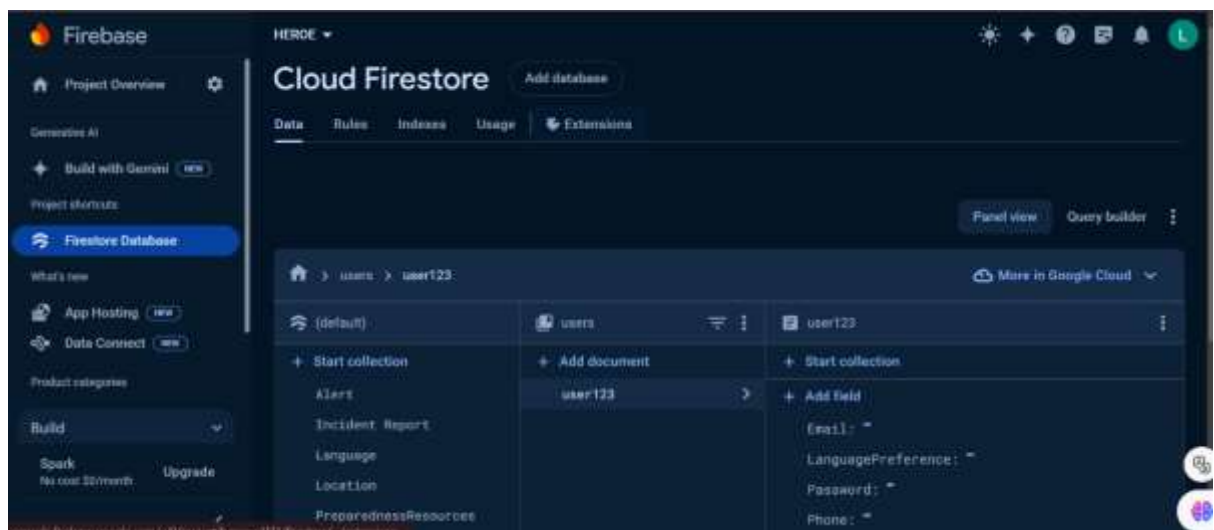
vii. Explanation of the Mapping

- User Collection: Stores user profiles with subcollections for incident reports, alerts, and accessed resources.
- Emergency Responder Contacts: Standalone collection for storing contact information.
- Alert/Notification Collection: Stores alerts with references to user and location.
- Location Collection: Stores locations with references in incidents and alerts.
- Incident Report Collection: Stores incident details with references to user and location.
- Language Collection: Stores language preferences.
- Preparedness Resources: Stores resource details with subcollections for users accessing these resources.

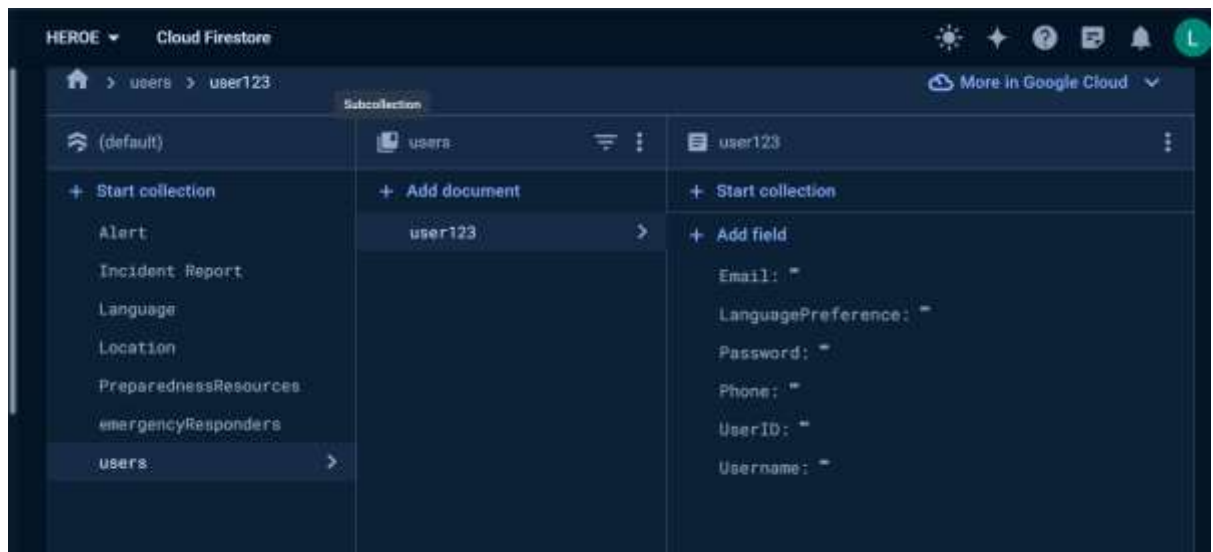
7.5. RESULTS

A. Collection and Document

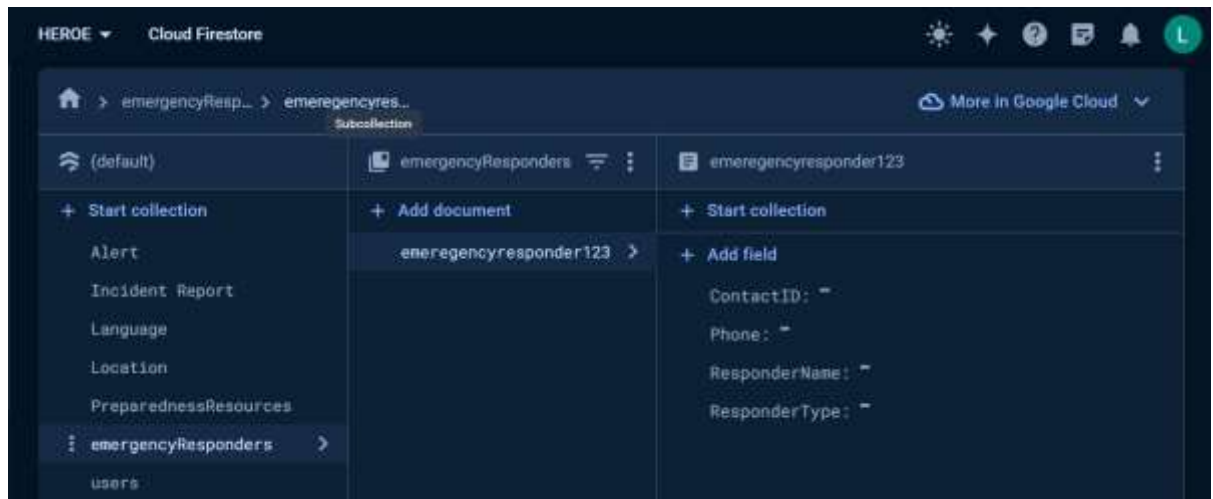
It has the results gotten from each collection and Documents that have their various attributes



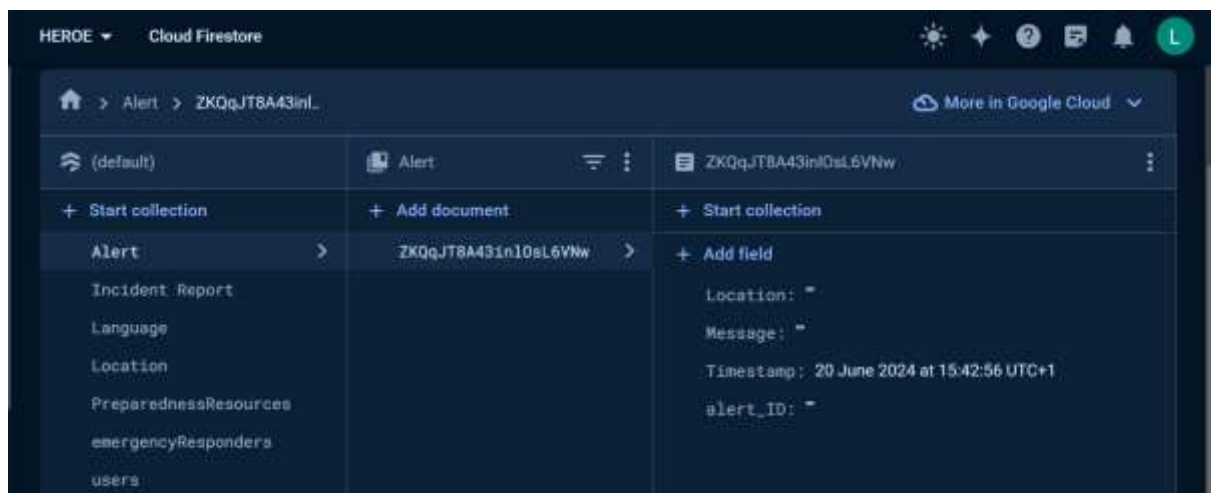
i. user/admin collection



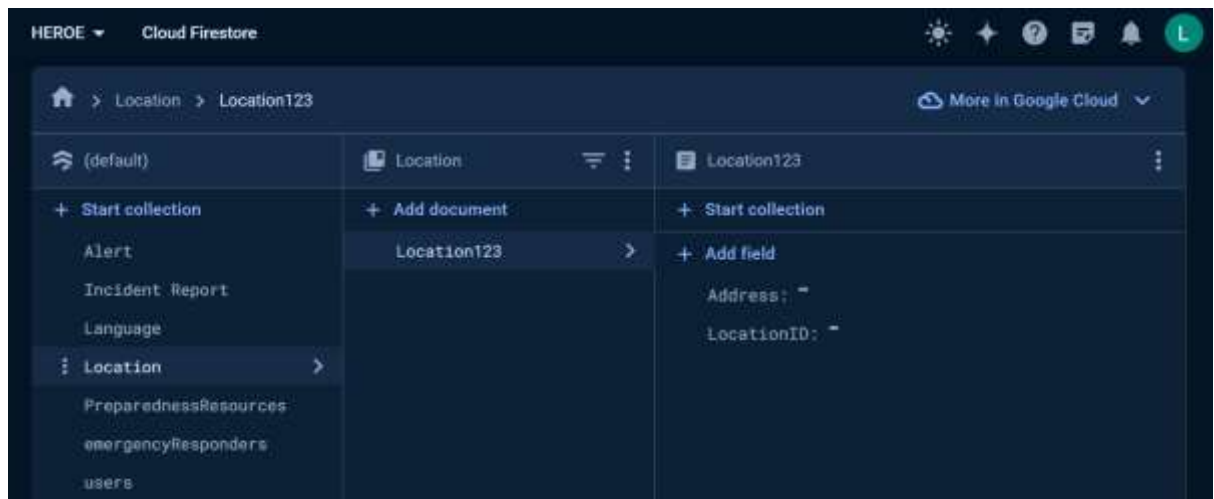
ii. Emergency Responders Collection



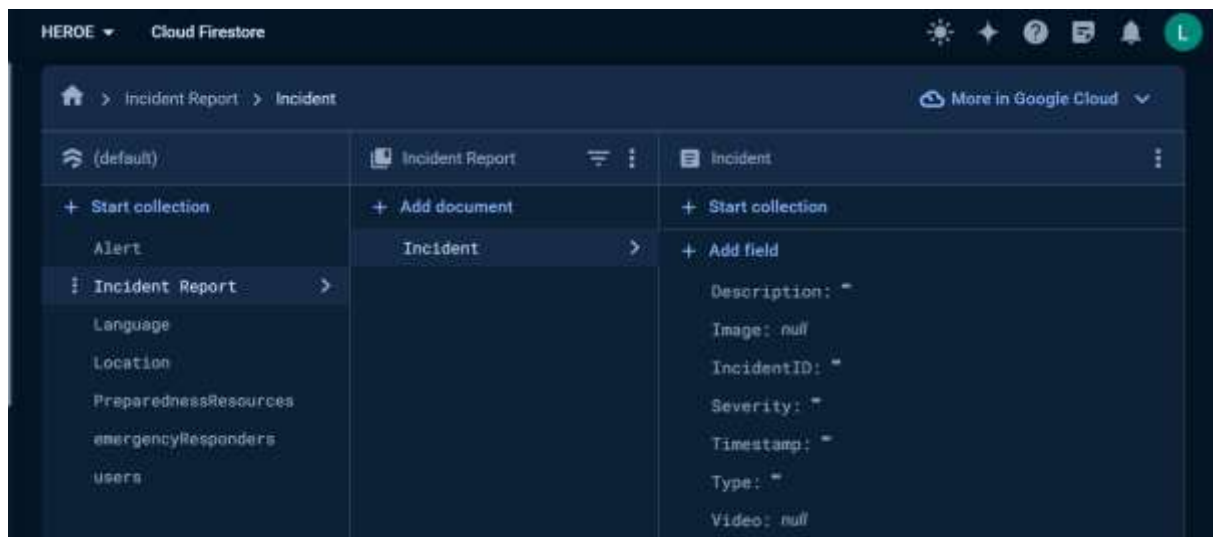
iii. Alert/Notification Collection



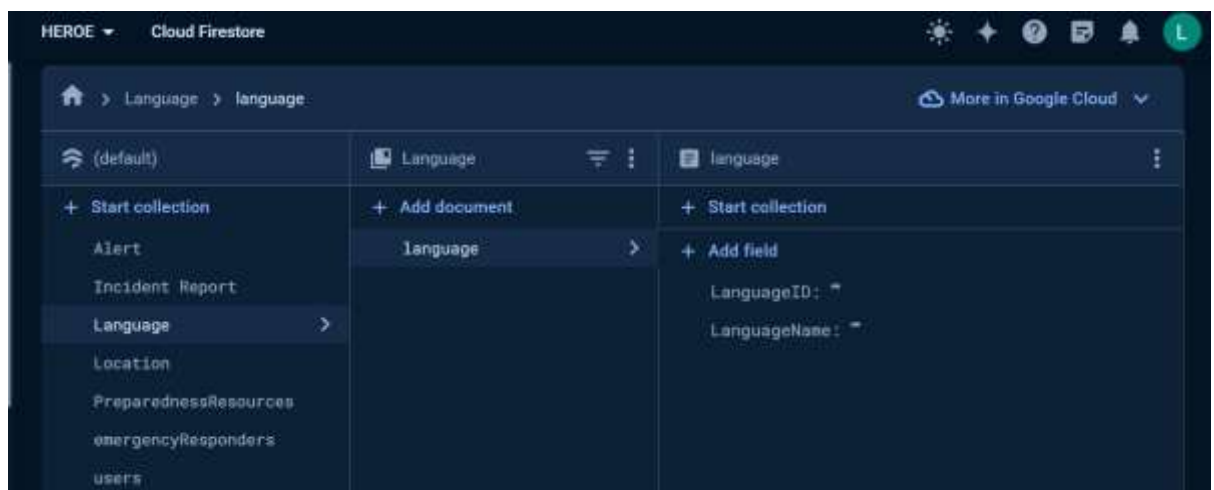
iv. Location Collection



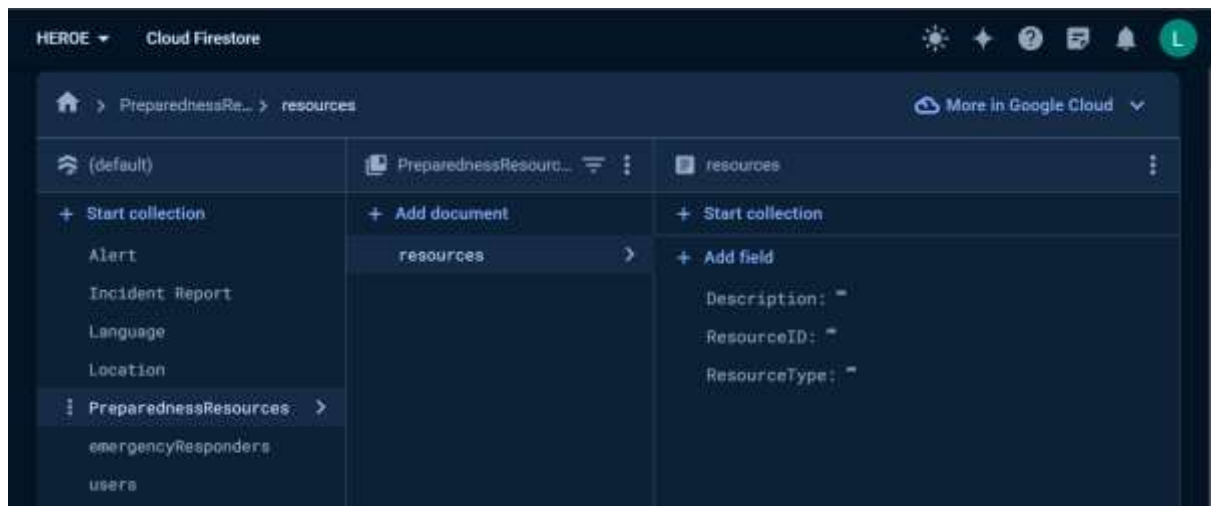
v. Incident Report Collection



vi. Language Collection

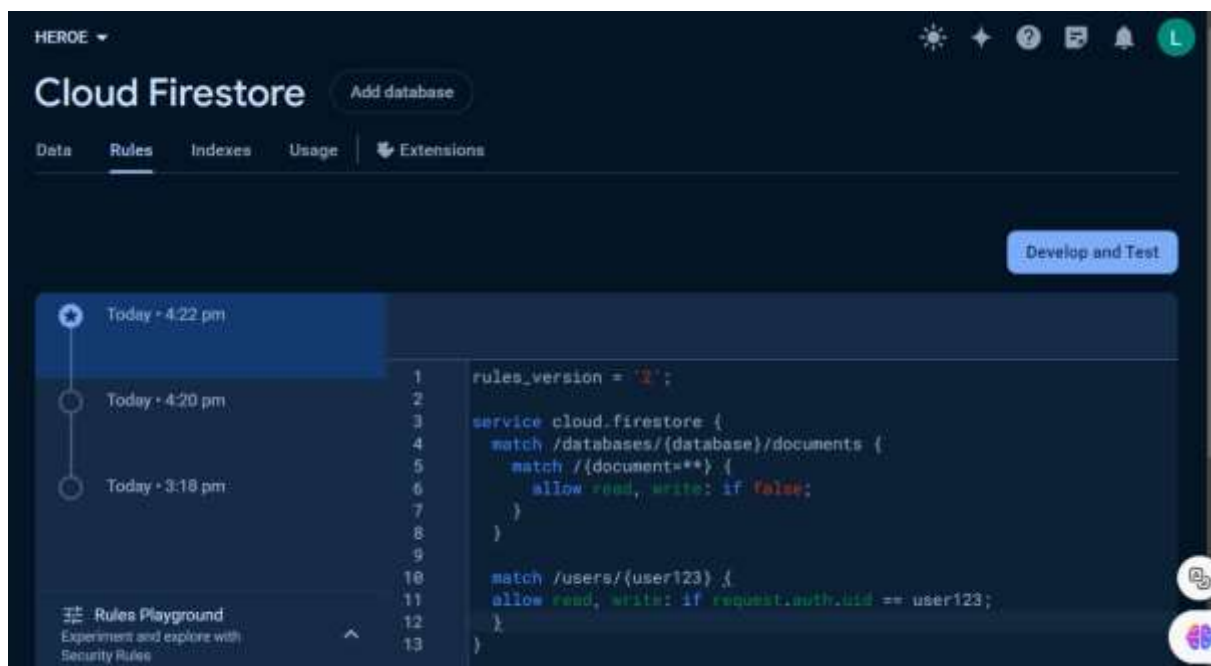


vii. Preparedness Resources



B. Rules

Set of rules on how users can read, write and perform other actions in the database



Detailed database design and implementation for a disaster management mobile application using Firebase Firestore. The design includes well-defined entities, attributes, and relationships, mapped effectively to Firestore collections and documents. This structure ensures efficient data management and real-time capabilities essential for a disaster management application.

CHAPTER 8: DEVELOPMENT AND RESULTS

The implementation of a disaster management mobile application (DMA) developed using React Native and Firebase. The application aims to empower users with the tools and information necessary for preparedness, communication, and response during emergencies.

8.1. Project Overview:

Objective: To develop a mobile application that enhances community preparedness, communication, and response to natural disasters and emergencies.

8.1.1. Technology Stack:

- React Native: Cross-platform mobile framework for building native iOS and Android applications.
- Firebase: Backend-as-a-Service (BaaS) platform for real-time database, authentication, storage, and other essential functionalities.
- Expo Go: mobile application used to view the developed application

8.1.2. Functionalities:

- User Authentication and Management:
 - * User registration, login, and profile management using Firebase Authentication (Email/Password, Google Sign-In).
 - * Secure user data storage in Firebase Firestore.
- Emergency Responders Contact:
 - * Database of verified emergency responders (police, fire department, ambulance) organized by region/location.
 - * Easy access to contact information (phone numbers, websites) with a single click.
- Alert Notification System:
 - * Real-time push notifications triggered by:
 - * Geo-location based alerts for specific disaster events (earthquakes, floods, etc.) using Firebase Cloud Messaging (FCM).
 - * Custom alerts from authorities (e.g., evacuation orders, shelter information).
- Preparedness Resources:
 - * Comprehensive library of disaster preparedness information categorized by hazard type (hurricanes, wildfires, earthquakes, etc.).

- * Access to downloadable resources (checklists, emergency kits, survival guides) in multiple languages.

- Incident Reporting:

- * Users can report incidents (power outages, road closures, injuries, etc.) with detailed information and location using integrated maps.

- * Real-time incident feed displays reported events with timestamps and location for community awareness.

- Location Services:

- * Utilizes device location services to personalize alerts and provide location-based information.

- * Maps integrated for navigation to emergency shelters, evacuation routes, and resource centers.

- Language Support:

- * Multiple language support using React Native's I18n library to cater to diverse user populations.

- * Users can choose their preferred language for UI and content.

8.2. **Technical Implementation:**

- React Native Architecture: Components are organized based on functionality (authentication, alerts, resources, reporting) for maintainability and scalability.

- Firebase Integration:

- * Firestore Database: Stores user data, emergency responder contact information, incident reports, and preparedness resources.

- * Authentication: Handles user registration, login, and account management.

- * Cloud Messaging (FCM): Powers real-time push notifications for disaster alerts and emergency updates.

- UI Design: Focuses on simplicity, clear navigation, and user-friendly design for optimal accessibility during emergencies.

- Data Management: Efficient data handling using Firestore to ensure real-time updates and synchronization across multiple devices.

- Security and Privacy: Data encryption, user consent mechanisms, and robust security measures implemented to protect user data.

Technical Implementation steps.

- Set up a new React Native project.
- Install necessary dependencies (Firebase, geolocation, localization).
- Configure Firebase project and add necessary credentials.
- Design UI components for each feature.
- Implement Firebase Firestore for data storage.
- Integrate geolocation services.
- Implement language preferences using localization libraries.
- Test thoroughly on both iOS and Android devices.

Security and Privacy:

- Ensure secure communication with Firebase (HTTPS).
- Implement Firebase security rules to restrict access.
- Handle sensitive user data (e.g., location) carefully

Implementation structure

CEF440-GROUP2/

| SourceCode

|— .expo/

|— .expo-shared/

|— assets/

| └— (assets like images, fonts, etc.)

|— components/

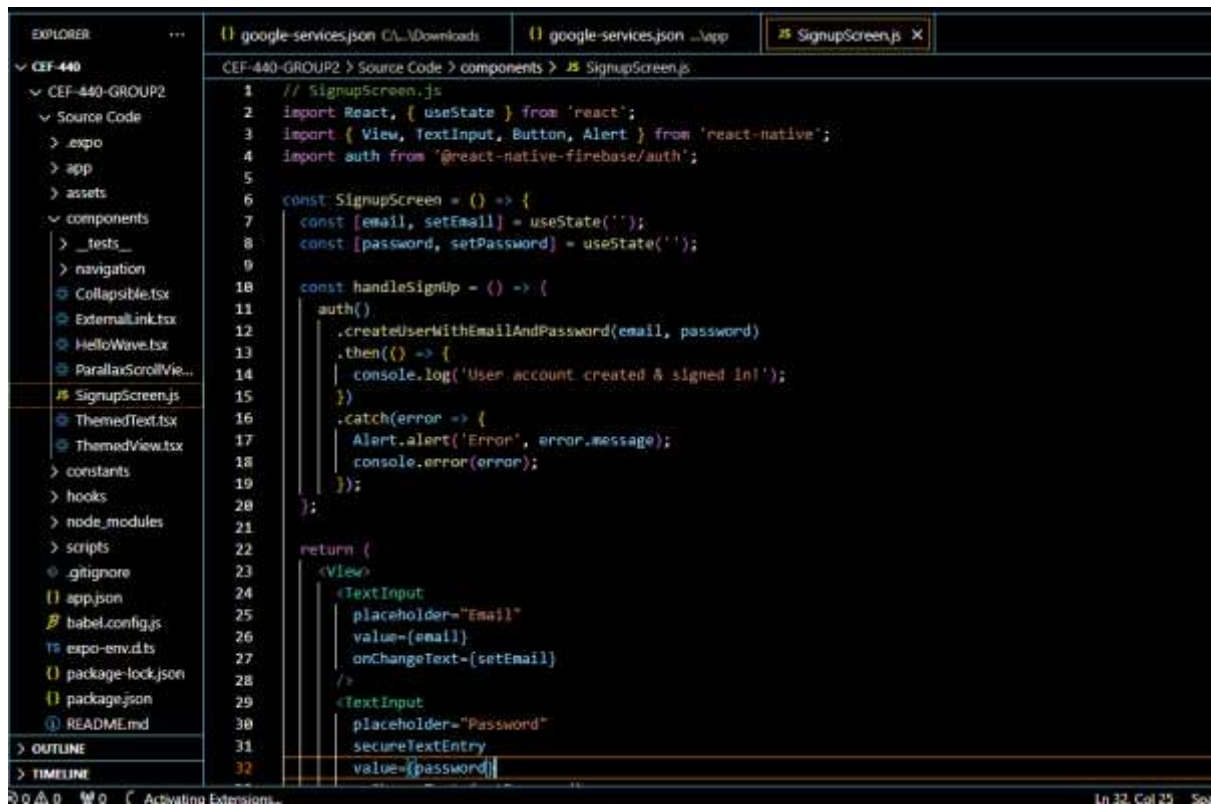
| └— AddDisaster.js

| └— ReadDocuments.js

- | | — UpdateDisaster.js
- | | — DeleteDisaster.js
- | | — IncidentReportForm.js
- | | — GeospatialMap.js
- | | — EmergencyContacts.js
- | | — AlertsNotifications.js
- | | — LanguageSelector.js
- | | — PreparednessResources.js
- | | — UserProfile.js
- | — contexts/
 - | | — AuthenticationContext.js
 - | | — GeolocationContext.js
 - | | — LanguageContext.js
 - | | — NotificationsContext.js
- | — firebase/
 - | | — firebase.js
- | — screens/
 - | | — HomeScreen.js
 - | | — LoginScreen.js
 - | | — IncidentReportScreen.js

```
| |── MapScreen.js
| |── EmergencyContactsScreen.js
| |── AlertsScreen.js
| |── ResourcesScreen.js
| └── ProfileScreen.js
|── utils/
| |── geolocation.js
| |── notifications.js
| |── language.js
| └── api.js
|── node_modules/
|── .gitignore
|── App.js
|── app.json
|── babel.config.js
|── package.json
└── README.md
```

8.3. CODE SCREEN SHOOTS



```
1 // SignupScreen.js
2 import React, { useState } from 'react';
3 import { View, TextInput, Button, Alert } from 'react-native';
4 import auth from '@react-native-firebase/auth';
5
6 const SignupScreen = () => {
7   const [email, setEmail] = useState('');
8   const [password, setPassword] = useState('');
9
10  const handleSignUp = () => {
11    auth()
12      .createUserWithEmailAndPassword(email, password)
13      .then(() => {
14        console.log('User account created & signed in!');
15      })
16      .catch(error => {
17        Alert.alert('Error', error.message);
18        console.error(error);
19      });
20  };
21
22  return (
23    <View>
24      <TextInput
25        placeholder="Email"
26        value={email}
27        onChangeText={setEmail}
28      />
29      <TextInput
30        placeholder="Password"
31        secureTextEntry
32        value={password}
33      />
```

Components Screens.

EXPLORER

CEP-440

CEP-440-GROUP2

Source Code

.expo

app

assets

components

tests

snapshots

ThemedText-test.tsx

Alerts

Alerts

JS NotificationScreen.js

navigation

Collapsible.tsx

JS EmergencyRespondersCantactScreen.js

ExternalLink.tsx

HelloWave.tsx

JS HomeScreen.js

InsidentReportingScreenjs

JS LanguageScree.js

JS LocationScreen.js

ParallaxScrollView.tsx

JS PreparednessResourcesScree.js

JS SignupScreen.js

ThemedText.tsx

ThemedView.tsx

OUTLINE

JS HomeScreen.js U

JS LocationS

CEP-440-GROUP2 > Source Code > con

1

EXPLODER

CEP-440

Source Code

components

tests

Alerts

Alerts

NotificationScreen.js

navigation

Collapsible.jsx

EmergencyRespondersContactScreen.js

ExternalLink.jsx

HelloWave.jsx

HomeScreen.js

IncidentReportingScreens.js

LanguageScreen.js

LocationScreen.js

ParallaxScrollView.jsx

PreparednessResourcesScreen.js

SignupScreen.js

ThemedText.jsx

ThemedView.jsx

USERSCREEN.JS

constants

hooks

node_modules

scripts

OUTLINE

TIMELINE

SignupScreen.js U

USERSCREEN.JS U

IncidentReportingScreens U

EmergencyRespond

CEP-440-GROUP2 > Source Code > components > SignupScreen.js > inc SignupScreen

```

1 // SignupScreen.js
2 import React, { useState } from 'react';
3 import { View, TextInput, Button, Alert } from 'react-native';
4 import auth from '@react-native-firebase/auth';
5
6 Complexity is 9 It's time to do something.
7 const SignupScreen = () => {
8   const [email, setEmail] = useState('');
9   const [password, setPassword] = useState('');
10
11   Complexity is 3 Everything is cool
12   const handleSignUp = () => {
13     auth()
14       .createUserWithEmailAndPassword(email, password)
15       .then(() => {
16         console.log('User account created & signed in!');
17       })
18       .catch(error => {
19         Alert.alert('Error', error.message);
20         console.error(error);
21       });
22   };
23
24   return (
25     <View>
26       <TextInput
27         placeholder="Email"
28         value={email}
29         onChangeText={setEmail}
30       />
31       <TextInput
32         placeholder="Password"
33         secureTextEntry

```

TASKS-2*

Ln 32, Col 25

Spaces: 2

UTF-8

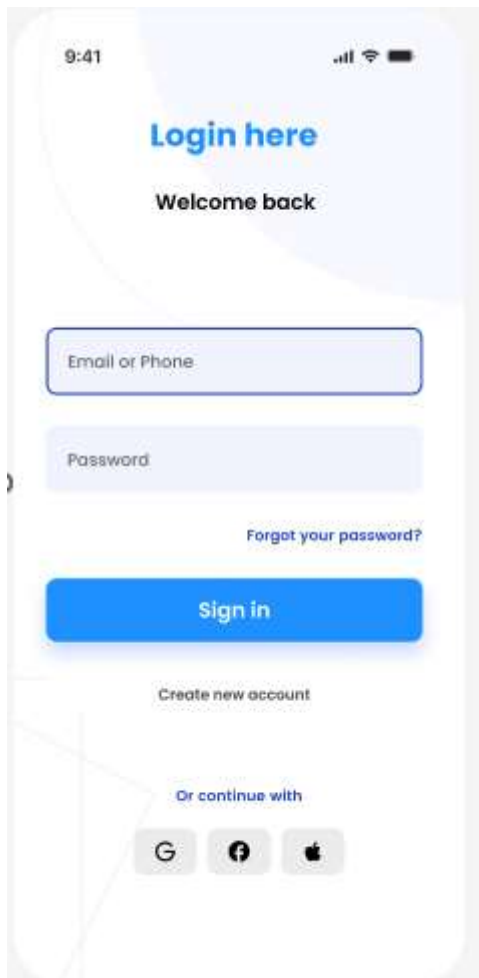
CRLF

JavaScript

Go Live

83

8.4. Results



CHAPTER 9: IMPLEMENTATION AND RESULT

Technologies Used

Frontend:

React Native: A JavaScript framework for building cross-platform mobile apps (iOS and Android) for the core app functionality.

Database and Backend:

Firebase Realtime Database: A NoSQL database by Firebase that stores and manages real-time data on disasters and emergency reports (optional).

Authentication:

Clerk: A third-party solution that handles user login securely and efficiently.

Implementation of the UI**Splash Screen**

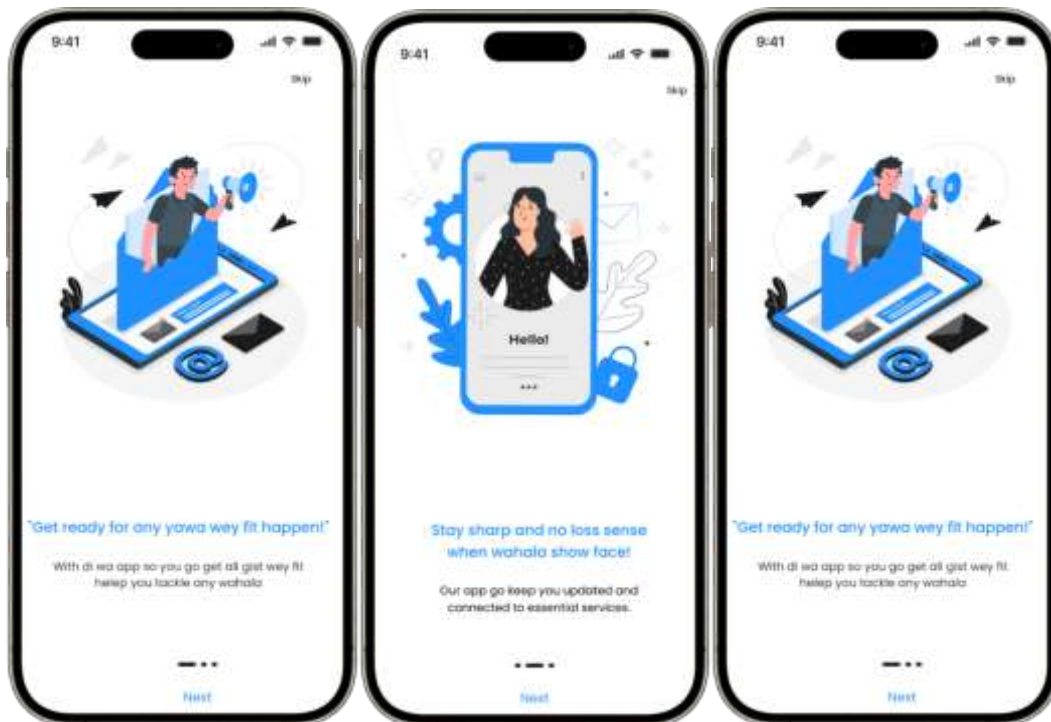
Library: A popular library like react-native-bootsplash can be used to manage the splash screen.

App Launch: The library typically integrates with the app's launch process to display the splash screen momentarily while the app initializes in the background.

**Onboarding Screens**

Components: Onboarding screens are built using standard React Native components like View, Image, Text, and potentially UI libraries for styling.

Navigation: A navigation library like react-navigation is used to manage screen transitions between the onboarding screens and the main app.



Authentication Implementation

Implementing SignUp and Login Screens

For the authentication, we made use of clerk. Clerk is a comprehensive authentication solution and it is known for its ease of use, security features, and seamless integration with React Native.

We chose clerk because;

Simplified Development: Clerk offers pre-built components and APIs, streamlining the login, registration, and password recovery processes. This reduces development time compared to building a custom solution.

Robust Security: Clerk prioritizes security by adhering to best practices. Features like secure password hashing and token-based authentication ensure user data protection.

Scalability: Clerk's infrastructure scales efficiently, accommodating a growing user base for the disaster management app.

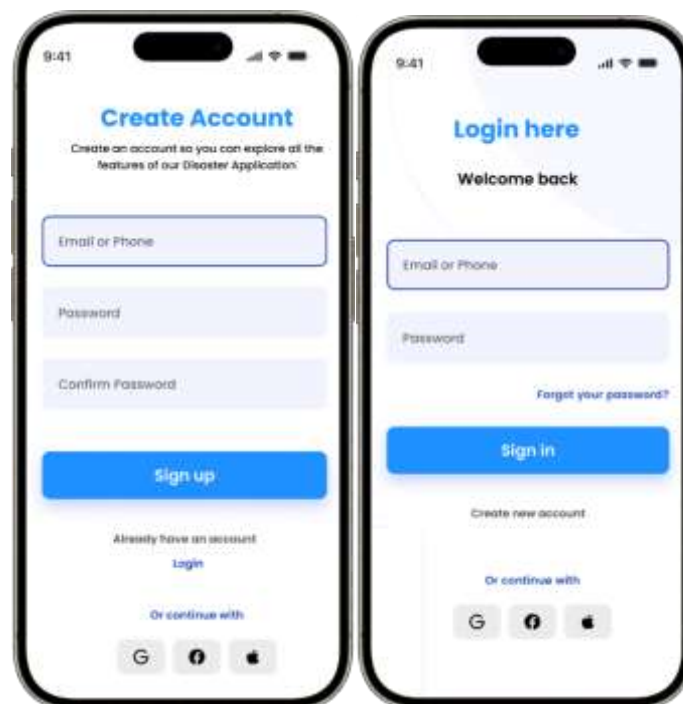
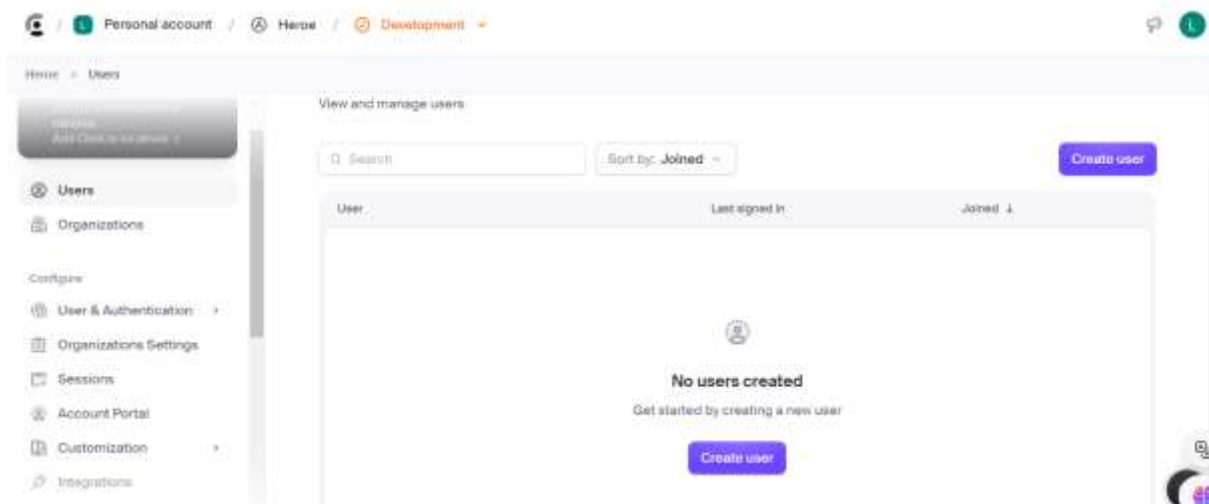
Frontend-focused: Clerk integrates seamlessly with the React Native frontend, removing the need for complex backend development for authentication.

How we Implemented:

Clerk Account Creation: A Clerk account was created and configured within the Clerk dashboard. This included defining user attributes relevant to the disaster management app (e.g., name, organization).

Clerk SDK Installation: The Clerk React Native SDK was installed within the React Native project.

The Clerk dashboard: This is the central repository for managing users who create accounts in our mobile application



Implementing the Home Screen

The home screen of the disaster management app serves as a central hub for users to stay in-

formed, report emergencies, and initiate contact with emergency services. This report outlines the tools and technologies used to achieve these functionalities.

Disaster Information Display:

Firebase Realtime Database: This NoSQL database serves as the backend, storing real-time data on the latest disasters.

React Native Components: The retrieved disaster data from Firebase is parsed and displayed on the home screen using React Native components like Text, FlatList, and Image (for displaying headlines, summaries, and thumbnails).

Emergency Reporting:

Formik: This form library simplifies building a user-friendly form for reporting emergencies (e.g., fire, flood, etc.). It handles data validation and form management.

SOS Button:

React Native Linking: This library allows the app to initiate a phone call to a pre-defined emergency responder number when the SOS button is pressed.

Implementation Flow:

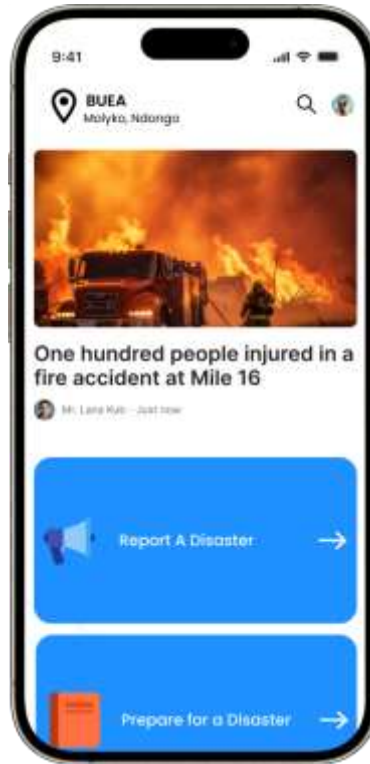
Upon successful user login via Clerk, the home screen retrieves real-time disaster data from the Firebase Realtime Database.

The disaster data is parsed and displayed in a user-friendly format on the home screen.

Users can report emergencies through a form built with Formik.

Form data is validated and then written to the Firebase Realtime Database (optional, depending on emergency reporting needs).

Clicking the SOS button triggers a phone call to a pre-defined emergency responder number using React Native Linking.



Implementation of Alert/Notifications

The Firebase Realtime Database would be extended to store data submitted by users through the reporting form on the home screen.

Form Submission:

When a user submits an emergency report through the form on the home screen, the data is captured using Formik. This data is then sent to the backend using functions like fetch or a dedicated library for interacting with Firebase.

Data Writing:

Upon receiving the emergency report data on the backend, code is implemented to write it to the emergency_reports section within the Firebase Realtime Database. This creates a new entry with a unique identifier and stores the submitted details.



Emmergency Resources



Implementation of the SOS Button

SOS Button Action:

The SOS button on the home screen serves as a quick way for users to initiate a phone call to a pre-defined emergency responder number.

The Calling Functionality:

It's important to note that this functionality is not currently implemented due to limitations with React Native.

Currently, React Native offers limited capabilities for directly initiating phone calls within the app itself.

Pre-defined Emergency Number: A specific emergency responder phone number is configured within the app. This ensures users are connected to the appropriate service.

And for our disaster mobile application we already have some pre-defined Emergency Numbers



Calling Screens



Implementation of the Backend

Firebase provides a scalable and real-time NoSQL database solution that is well-suited for the app's requirements.

Data Storage:

Disaster Information: Firebase Realtime Database stores real-time data on various disasters. This data can include details like disaster type, location, severity, timestamps, and any additional relevant information.

Emergency Report: Depending on the specific requirements, our app leverage Firebase to store user-submitted emergency reports. This data includes the type of emergency, location, and any additional details provided by the user.

Real-time Updates:

A crucial aspect of the implementation is Firebase's real-time functionality. When new disaster information or emergency reports (if enabled) are added or updated in the database, the changes are reflected immediately on the app's home screen for all connected users. This ensures everyone has access to the latest information.

Security Rules:

Firebase offers security rules to control data access and modification within the database. These rules can be implemented to restrict unauthorized access or actions on the data.

Backend Logic

Data Storage:

Disaster Information: Real-time data on disasters (type, location, severity, etc.) is stored in the database.

Emergency Reports (Optional): If enabled, user-submitted emergency reports (location, type, etc.) are also stored.

Real-time Updates: Whenever disaster information or emergency reports (if enabled) are added, updated, or deleted, these changes are reflected immediately for all connected users on the app's home screen.

Data Access (Optional): Depending on the implementation, security rules might be defined to control who can access or modify data within the database.

This real-time functionality ensures everyone using the app has access to the latest critical information.

CHAPTER 10: CONCLUSION

We have come to the end of the design and implementation of a disaster Management Mobile Application. Realising this project wasn't easy, but with the help of God and consistent follow up from our supervisor, we were able to overcome some challenges. Some challenges faced were:

- Consistent interruption of electrical supply
- Unstable and low internet connection
- Having access in downloading libraries and dealing with hocks and API

With continual research and persistent working, we tried to a certain extent to overcome these shortcomings.

CHAPTER 11: REFERENCES/ FUTURE RECOMMENDATION

10.1. References

- 1) Enhanced 911 <http://library.ema.gov.au/LIBERO/WebOpac.cls>
- 2) SMS Alerts <http://www.egov4dev.org/mgovapplic.htm>
- 3) SMS emergency broadcasting <http://www.egov4dev.org/mgovapplic.htm>
- 4) Mobile devices to government authorities www.jsce-int.org/Report/report/flood_euro.pdf
- 5) The value of GSM in a natural disaster situation. http://www.gsmworld.com/news/press_2000/press_releases_43.shtml
- 6) New emergency response system helps fight disasters (EGERIS) http://www.innovations-report.com/html/reports/communication_media/report-29094.html
- 7) Disaster Emergency Management Information System (AFAYBIS) <http://www.isprs.org/istanbul2004/comm4/papers/339.pdf>
- 8) <https://www.figma.com/design/Ud6Hh9suPpsFRma0GNVRdZ/H%C3%A9roe?node-id=0-1&t=KKhJFXQIBiqZTXNN-0>
- 9) Firebase: <https://console.firebase.google.com/>
- 10) Building Real-time Disaster Response Applications with Firebase: <https://firebase.google.com/docs/firestore/disaster-recovery>
- 11) Firebase Medium Review: <https://medium.com/@louisjaphethkouassi/how-to-use-google-firebase-realtime-database-9df362548de7>

10.2. Future Enhancements:

- Integration with External APIs: Utilize APIs from meteorological agencies and disaster relief organizations to provide more accurate and comprehensive data.
- AI-Powered Features: Explore integrating AI algorithms for:
 - * Personalized risk assessment: Predicting individual risk based on user location and hazard profile.

* Automated incident categorization: Utilizing machine learning to classify reported incidents for faster response.

- Gamification and Incentives: Introduce gamefic
- Bandura, A. (1986). Social foundations of thought and action: A social cognitive theory. Prentice-Hall.
- Norman, D. A. (2013). Design of everyday things. Basic Books.
- Rogers, E. M. (2003). Diffusion of innovations (5th ed.). Free Press.
- Slovic, P. (1987). Perception of risk. Science, 236(4799), 280-285.
- UNISDR. (2005). Hyogo Framework for Action 2005-2015: Building the Resilience of Nations and Communities to Disasters.
- UNISDR. (2015). Sendai Framework for Disaster Risk Reduction 2015-2030.
- WCAG (Web Content Accessibility Guidelines) (2021). Retrieved from <https://www.w3.org/WAI/standards-guidelines/wcag/>

