Diploma Project

RESQIA: A GEOSPATIAL ANDROID FRAMEWORK FOR REAL-TIME EMERGENCY INCIDENT REPORTING AND VOLUNTEER MOBILISATION

Eirini Alexandrou

University of Cyprus
Department of Computer Science

Department of Computer Science

May 2026

UNIVERSITY OF CYPRUS

Faculty of Pure and Applied Sciences

Department of Computer Science

RESQIA: A GEOSPATIAL ANDROID FRAMEWORK FOR REAL-TIME EMERGENCY INCIDENT REPORTING AND VOLUNTEER MOBILISATION

Eirini Alexandrou

Supervisor

Dr George A. Papadopoulos

Co-Supervisor

Dr Christos Mettouis

The Thesis was submitted in partial fulfillment of the requirements for obtaining the Computer Science degree of the Department of Computer Science of the University of Cyprus

May 2026

I declare that this dissertation and any accompanying software are my own original work, conducted in full compliance with the University of Cyprus Code of Conduct for Research https://www.ucy.ac.cy/legislation/kodikas-deontologias-kai-politikes/, and with full accountability on my part for the accuracy, integrity, and validity of all content.

Acknowledgements

I sincerely thank my supervisor, Dr George A. Papadopoulos, for his guidance, support, and valuable feedback throughout this project. Recognising his role helps convey my appreciation and makes him feel truly valued.

I also thank my co-supervisor, Dr Christos Mettouris, for his insightful suggestions and assistance, which provided clarity and direction during this thesis.

I would also like to express my sincere appreciation to the medical professionals who generously shared their knowledge and guidance on medical emergencies. Through our discussions, we collaboratively determined the most appropriate critical time window between the onset of an incident and the point at which the situation may become too severe for non-professionals to manage safely. Their expertise and input were essential in shaping the emergency-response aspects of this project.

Finally, I thank my family and friends for their continuous support, patience, and encouragement throughout my studies and the completion of this thesis.

ABSTRACT

Emergencies require prompt awareness and response. Traditional emergency communication methods, which depend on emergency calls and official responders, can delay information dissemination and hinder local situational awareness. This thesis introduces ResQia, a mobile emergency reporting and awareness application developed to facilitate real-time, community-based emergency management. The system aims to reduce the information gap between the occurrence of an emergency and the moment nearby citizens become aware of it by enabling users to report emergencies, share location-based information, and receive proximity-based notifications.

ResQia was implemented as a native Android application using Java and XML in Android Studio. The platform integrates Firebase Authentication, Cloud Firestore, Google Maps SDK, OneSignal notifications, and geospatial proximity filtering to support emergency reporting, volunteer coordination, and real-time information synchronisation. Furthermore, the system includes a helper verification mechanism that enables medically trained users to receive prioritised notifications for nearby medical emergencies.

The system was evaluated through a four-day user study involving 20 participants. The results indicated positive perceptions regarding usability, practicality, reporting speed, situational awareness, and overall usefulness, demonstrating the potential of community-driven mobile systems to support emergency awareness and response.

Keywords: Emergencies, Mobile Application, Smart Cities, Community-Based Response, Real-Time Notifications

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
BaaS	Backend-as-a-Service
EMS	Emergency Management Systems
GDACS	Global Disaster Alert and Coordination System
IoT	Internet of Things
SDK	Software Development Kit
UI	User Interface
UAV	Unmanned Aerial Vehicle
UX	User Experience
XML	Extensible Markup Language
GPS	Global Positioning System
IoT	Internet of Things
AI	Artificial Intelligence
EMS	Emergency Management Systems
UAV	Unmanned Aerial Vehicle
AED	Automated External Defibrillator
CPR	Cardiopulmonary Resuscitation
MHEWS	Multi-Hazard Early Warning System
OSOCC	On-Site Operations Coordination Centre
SMCS	Satellite Mapping Coordination System
API	Application Programming Interface
CPR	Cardiopulmonary Resuscitation
EMS	Emergency Medical Services
API	Application Programming Interface
CPR	Cardiopulmonary Resuscitation
ER	Entity–Relationship
GPS	Global Positioning System
MVVM	Model–View–ViewModel
RDBMS	Relational Database Management System
URI	Uniform Resource Identifier
FCM	Firebase Cloud Messaging
IDE	Integrated Development Environment
REST	Representational State Transfer
URL	Uniform Resource Locator

1 Introduction

1.1 Background

Nowadays, urban areas are the most populated, resulting in a growing demand for improved and sustainable public services. These public services, such as emergency management, ought to be smart by exploiting heterogeneous data, as this is a fundamental service and has a significant impact and influence on urban safety. Currently, due to technological developments, emergency management has transitioned from relying solely on emergency calls and non-automated dispatching of response vehicles to incorporating electronic sensors, smartphones, and other digital technologies. Therefore, cyber-physical systems have been created to provide detection, alerting, and mitigation of emergencies in a city, while automating most steps of the emergency management cycle [1].

Emergency management is directly related to protecting human life and public safety and reducing harm during unexpected events. In urban environments, emergencies may involve road accidents, fires, medical incidents, violence, natural disasters, or other situations that require immediate awareness and response. Traditionally, such incidents have been reported mainly through emergency calls, followed by the dispatch of official response vehicles. However, this approach may be affected by reporting delays, inaccurate location descriptions, and the time required for professional responders to reach the scene.

The development of smart cities, smartphones, Internet of Things technologies, and digital communication platforms has created new opportunities to collect and share emergency-related information. Smart cities use information and communication technologies to improve public services, support decision-making, and enhance citizens' quality of life [2, 3]. Within this environment, citizens can become active participants in emergency management by reporting incidents, sharing information, and supporting situational awareness. This reflects a modern governance approach, in which public services are supported through collaboration among authorities, organisations, communities, and citizens [4].

Citizens are especially valuable in emergencies because they are often physically close to an incident when it occurs. An eyewitness can provide immediate information about the type of emergency, its location, and its severity. In dense urban areas, where incidents can develop quickly, this real-time information can help reduce danger and improve awareness among nearby people. Similar to the concept of "social sensors", citizens and their mobile devices can serve as data sources that support event detection and situational awareness [5].

Emergency management can also be viewed as a life-cycle process that includes detection, re-

porting, response, mitigation, and recovery [6]. A mobile emergency application can support the early stages of this cycle by enabling users to report emergencies in real time. It can also support the response stage by notifying nearby users, including individuals who may be able to assist before official emergency services arrive.

1.2 Problem Statement

Although emergency services are essential, there are still several challenges in the way emergencies are reported and handled. One major problem is the delay between when an emergency occurs and when people nearby become aware of it. In many cases, only the official emergency services are informed, while nearby citizens remain unaware, even though they may be close enough to avoid danger or provide help.

Another problem is that eyewitness information is often not shared efficiently. A person who sees an emergency may call the authorities, but the information is usually not immediately available to other people in the surrounding area. This limits public awareness and reduces the possibility of community-based assistance. For example, in a medical emergency, a nearby person with first-aid knowledge may be able to help, but they cannot act if they do not know that the incident is happening nearby.

Furthermore, emergencies require quick and reliable information. Real-time systems must provide information quickly while reducing false reports and unnecessary alerts. As seen in real-time event recognition systems, speed, reliability, and low false alarms are important requirements when systems are used in safety-related environments [7]. Therefore, an emergency application must be designed carefully so that reports are useful, clear, and location-based.

A further challenge is the lack of connection between citizens, emergency awareness, and nearby assistance. While smartphones are widely used, there is still a need for applications that allow citizens to report incidents they have directly witnessed and notify others in real time. This creates an opportunity to use mobile technology to improve emergency awareness, support faster reaction, and encourage community participation in public safety.

1.3 Proposed Solution

To address these challenges, this thesis proposes developing an emergency mobile application that enables users to report emergencies they witness in real time. The application is designed to facilitate rapid communication among citizens by enabling eyewitnesses to submit emergency reports with relevant details, such as the type of emergency, location, and a description of the incident.

The main purpose of the application is to increase awareness of nearby emergencies. When an emergency is reported, nearby users can be notified so that they are aware of the situation. This can help people avoid dangerous areas, take precautions, or assist others if they can. In particular, users with medical knowledge or first-aid skills may be able to respond to nearby medical incidents before official emergency services arrive.

The proposed application is based on the idea that citizens can contribute to emergency management as active participants. This aligns with the broader concept of governance, in which different actors, including ordinary citizens, support the delivery of public services [4]. It also follows the smart city approach, where digital technologies and data are used to improve urban safety and public services [2, 3].

By using real-time reporting and location-based notifications, the application can support the detection and response stages of the emergency management life cycle [6]. It can also enhance situational awareness by collecting information directly from people present at the incident location. This form of citizen-generated data can be valuable because it provides immediate information about real-world events [5].

1.4 Thesis Structure

The remainder of this thesis is organised as follows.

Chapter 2: Background,

Chapter 2 presents the theoretical background of emergency management within smart-city services. It introduces the core concepts of Emergency Management Systems (EMS), including the emergency management cycle, hazard categories, emergency modelling approaches, and metadata for describing emergencies. Furthermore, the chapter discusses the role of heterogeneous sensing technologies, IoT infrastructures, and smart-city services in supporting emergency detection, alerting, and mitigation.

Chapter 3: Related Work

Chapter 3 reviews the existing literature and emergency response platforms on citizen participation, crowdsourced emergency reporting, and mobile emergency applications. The chapter discusses citizens as real-time information providers through smartphone technologies and examines the importance of geolocation, multimedia evidence, and rapid reporting interfaces during emergencies. Additionally, existing platforms, namely GDACS, Ushahidi, and PulsePoint Respond, are analysed to identify their strengths, limitations, and relevance to the proposed system.

Chapter 4: Proposed System – ResQia

Chapter 4 presents the proposed system, a native Android application designed to support real-time, community-based emergency management. The chapter introduces the system's purpose and objectives, target audience, operational scope, and how it differs from existing emergency-response platforms. Furthermore, it presents the functional and non-functional requirements, the trust and validation model, and the system modelling process through use-case analysis.

Chapter 5: System Architecture and Design

Chapter 5 presents the architectural design of the ResQia platform. It describes the distributed Backend-as-a-Service architecture, Android client structure, cloud backend infrastructure, notification and geospatial logic, database design, security architecture, and authorisation model that support the system's operation.

Chapter 6: System Implementation

Chapter 6 presents the implementation of the ResQia application. It describes the development environment, tools, and technologies used, including Android Studio, Java, XML, Firebase, Firestore, Firebase Authentication, Google Maps SDK, OneSignal, and external service integration. Additionally, the chapter explains the implementation of the core system components, including authentication and profile management, volunteer verification, geospatial incident discovery, emergency reporting, notification delivery, data synchronisation, inter-component communication, and UI/UX refinement.

Chapter 7: System Evaluation

Chapter 7 presents the evaluation of the proposed system. It describes the user-based evaluation methodology, participant profile, questionnaire design, functional evaluation results, reporting speed feedback, user experience evaluation, qualitative feedback, discussion, and the study's limitations.

Chapter 8: Conclusion

Chapter 8 summarises the work carried out throughout this thesis, discusses the main findings and contributions of the proposed system, highlights its limitations, and outlines possible improvements and directions for future work.

2 Background

2.1 Emergency Management within Smart-City Services

One of the most important public services in smart cities is emergency management, which relies on technological solutions to improve urban safety and resource coordination. Since modern cities are increasingly complex and large-scale, implementing emergency management systems (EMS) requires multidisciplinary approaches and the integration of various technologies [1]. In addition, modern emergency management involves multiple stakeholders, including public agencies, private-sector actors, and citizens, thereby highlighting the need for integrated platforms capable of processing diverse information sources [4]. Within this context, emergency management coexists with other urban systems, such as transportation, utilities, and energy, and it emerges as one of the most fundamental smart services [1].

2.2 Core Concepts of Emergency Management Systems

2.2.1 The Emergency Management Cycle

Emergency Management Systems in Smart Cities are commonly structured around a cycle that describes how abnormal situations are identified and handled. An emergency is understood as the initial stage of a dangerous situation that requires immediate action to prevent escalation into a disaster. In this context, hazards are the inherent sources of danger in urban environments, whereas disasters are the severe consequences that arise when emergencies are not addressed in time. The emergency management cycle begins when a detection service identifies that a hazard is manifesting and classifies it as an emergency, generating metadata to describe the event, such as its type, location, and severity. Once it is detected, an alert is created to notify all affected or relevant entities, thus enabling timely situational awareness. Afterwards, mitigation actions begin, aiming to contain the emergency, reduce its impacts, and prevent it from evolving into a disaster [1]. While many cities adopt this detection-alerting-mitigation structure, other works describe emergency management using broader life-cycle perspectives. One example is the “3+3” emergency life-cycle model, which organises the process into three phases:

1. *Forecast and early warning, which involves continuous hazard monitoring and early alert issuance.*
2. *Rescue and disposal, which focuses on urgent actions taken immediately after an emergency occurs.*

3. *Recovery and reconstruction, which aim to restore normal conditions and strengthen long-term resilience.*

These extended models emphasise the importance of continuous monitoring, early warning, and effective communication across stakeholders. This perspective aligns with the focus of this thesis, which aims to support the early stages of emergency management by enabling citizens to report incidents in real time and by notifying nearby users who may be able to assist before official responders arrive [6].

2.2.2 Hazards and Emergency Types

Hazards in urban environments are potential sources of harm that can threaten people, infrastructure, and the continuous operation of essential services within a city when specific conditions are met. A hazard becomes an emergency when its intensity exceeds safe thresholds or when it interacts with other factors in a way that demands immediate action. Since this transition is rapid and context-dependent, smart cities rely on continuous monitoring and accurate hazard modelling to detect hazardous conditions before they escalate [1]. Therefore, modern urban environments deploy diverse sensing systems, including IoT devices, environmental sensors, mobile platforms, UAVs, vehicle-installed modules, and software sensors derived from online user activity, to identify both natural and human-induced hazards at early stages [2, 7–9].

Hazard detection in smart Cities increasingly depends on multimodal, heterogeneous data sources that capture complex urban dynamics. Time-series data from IoT networks are fundamental for monitoring environmental parameters such as temperature, humidity, smoke levels, atmospheric pollutants, water quality, and structural stress in buildings or critical infrastructure [2, 8]. More complex hazards require richer representations and are identified using computer vision, UAV-based monitoring, remote sensing, and AI-driven fusion for visual, spatial, environmental, and behavioural information [3, 5, 7, 9]. For example, fires, floods, earthquakes, and traffic can cause collisions and crowd disturbances, which may be detected using video analytics, aerial surveillance, or environmental sensor readings. At the same time, social media activity and citizen-generated content operate as "software sensors" that can reveal incidents not captured by physical monitoring systems [1, 5, 7, 9]. Hazards linked to human behaviour, such as distracted driving, abrupt braking, unsafe manoeuvres, and crowds' surges, can be anticipated by multimodal sensing frameworks combined with in-vehicle cameras, driver monitoring, vehicle telemetry and contextual environmental cues [3, 5]. Smart City platforms complement these sensing mechanisms by generating real-time alerts, dynamic risk maps, and decision-support insights, enabling timely emergency responses [8]. Hazards in Smart Cities are broadly classified into environmental, urban, and health-related categories [1]. Environmental hazards arise from natural processes, including floods, storms, droughts, wildfires, heatwaves, earthquakes,

and land subsidence, many of which are intensifying due to climate change and rapid urban expansion [2, 9, 10]. Additionally, risks such as poor air quality, dust, allergens, and water contamination contribute to environmental degradation and trigger secondary health impacts [8, 10]. As for Health hazards, they include individual medical emergencies, such as cardiac events and respiratory distress, as well as population-level threats, including pandemics, pollution-induced illnesses, and environmentally triggered reactions [8, 10]. The interaction among these categories highlights the interconnected nature of hazards in dense metropolitan settings.

As cities expand and adopt large-scale sensing infrastructure, the volume of and complexity of hazard-related data increase, requiring AI-based analytics for early detection and mitigation [2, 5, 8]. Machine-learning models process multimodal datasets to predict hazardous conditions such as fires, flooding, mobility risks, and health emergencies with improved accuracy [2, 5, 9]. Smart-health systems illustrate this capability by detecting falls or psychological distress and supporting faster medical response through dynamic traffic-light adjustments or personalised routing away from polluted areas [8]. UAVs extended detection coverage by providing high-resolution imagery of hazardous or inaccessible areas, strengthening situational awareness for emergency response [9]. Together, these technologies enhance smart cities' ability to identify emerging hazards promptly and coordinate effective interventions.

2.2.3 What Constitutes an Emergency

In smart-city Emergency management Systems, an emergency is modelled as a virtual entity representing a critical situation, defined by the manifestation of one or more hazards and contextual metadata, such as location, time, severity, and type [1]. A common and widely adopted approach models emergencies as self-contained, meaning that each emergency is linked to a single cause, such as fire, flood, earthquake, or traffic accident [1]. This modelling strategy is relatively simple to implement and is frequently used in systems designed to detect specific types of incidents, including fire detection platforms, heavy rainfall monitoring systems, or traffic accident detection frameworks [2, 3]. In these cases, emergency detection is typically achieved by monitoring a primary hazard and issuing an alert once abnormal conditions are observed. More advanced approaches define emergencies from an event-based perspective, where an emergency arises from a combination of multiple critical events or hazards occurring within a given spatial and temporal context [1]. In this view, individual events act as indicators of strange behaviour, while the emergency itself is inferred through their joint analysis. Event-based emergencies may be detected either through threshold-based triggering, where predefined limits activate emergency detection, or through aggregated analysis, where multiple heterogeneous signals are combined to infer complex situations [1]. This latter approach is particularly suitable for smart cities, where data from IoT sensors, crowdsourced user reports, public online content, UAV observations,

and health monitoring systems can be jointly analysed using machine learning and data fusion techniques [5, 8, 9]. Such modelling enables earlier detection and richer situational awareness, especially in complex scenarios where no single hazard is sufficient to indicate a developing emergency.

2.2.4 Metadata in Emergency Management Systems

When an emergency is detected, the virtual antiunity it is represented by is enriched with metadata describing the main characteristics and the reasoning behind its decision. Core metadata typically includes the affected area, the duration of the emergency, and its severity level. The affected area is usually defined by the geographic location of the associated hazards, often using GPS coordinates, and is extended to an influence zone to account for spatial propagation. Emergency duration captures the temporal persistence of a critical situation, which may continue beyond the active manifestation of the originating hazard. Severity levels reflect the potential impact of an emergency. They are commonly computed by considering hazard intensity together with spatial and temporal context, such as population density, infrastructure characteristics, and time of occurrence. Together, these metadata enable emergency management systems to prioritise concurrent emergencies and support effective alerting and mitigation actions [1].

3 Related Work

Citizens are the first to witness emergencies and can provide real-time information before official responders arrive, thus supporting emergency response efforts. Modern mobile phones further enable this process by allowing people to use embedded sensors such as GPS, Wi-Fi, accelerometers, and cameras to communicate relevant information, effectively turning users into human sensors. In this context, emergency notification applications have been developed to serve as intermediaries between citizens and emergency authorities, complementing traditional emergency calls by providing richer information rather than replacing them [11].

Among the information exchanged during an emergency, location is the most critical element; therefore, accurate geolocation is essential. Many individuals find it difficult to describe their location verbally, making GPS-based location reporting particularly important. Additionally, the ability to capture photos and videos is vital, as multimedia content is often more objective and actionable than textual descriptions, allowing emergency operators to infer critical details directly from visual data [11].

Given the stressful nature of emergencies, users must be able to report incidents as quickly as possible, which requires applications to minimise interaction and avoid complex forms. In practice, users tend to prefer taking photos rather than writing text, and multimedia reporting is perceived as faster and more comfortable than voice calls. Finally, while social media platforms are effective for large-scale disasters, smaller and localised incidents often go unnoticed, highlighting the importance of dedicated mobile emergency applications [11].

Following this, there are 3 emergency detection and response platforms worth mentioning, which will be mentioned below:

3.1 Global Disaster Alert and Coordination System (GDACS)

An international early-warning platform launched in 2004 to support humanitarian response to large-scale natural disasters. It was developed to address delays, information gaps, and coordination issues that arise when multiple actors collect and analyse disaster information through separate channels. GDACS aggregates science-based data from independent sources into a Multi-Hazard Early Warning System (MHEWS), thus providing real-time information and automatic notifications to subscribers for severe events such as earthquakes, tsunamis, and tropical cyclones, to support coordinated decision-making for events that may require international assistance.

Within this framework, GDACS is composed of three main components:

a) **Multi-Hazard Early Warning System (MHEWS)**, which is responsible for monitoring potential and ongoing natural disasters worldwide in real time and near real time, covering seven hazard types, specifically droughts, earthquakes, tsunamis, floods, forest fires, tropical cyclones, and volcanic eruptions. Additionally, it issues automatic, risk-based notifications using a standardised severity scale that combines hazard intensity, population and infrastructure exposure, and regional vulnerability to support early international situational awareness and decision-making.

b) **Virtual On-Site Operations Coordination Centre (Virtual OSOCC)**, which supports early-phase disaster response by enabling structural exchange of information and coordination among authorised international responders, thus helping to reduce duplication and delays.

c) **Satellite Mapping Coordination System (SMCS)**, which facilitates the sharing and coordination of satellite imagery and mapping products among multiple organisations, improving situational awareness while avoiding redundant mapping efforts during emergencies.

GDACS adopts a unified, impact-oriented approach to monitor and assess multiple natural hazards through its MHEWS. Emergencies are detected automatically or semi-automatically using authoritative scientific data sources. They are evaluated based on a combination of hazard intensity, population and infrastructure exposure, and regional vulnerability or coping capacity. This common framework allows different types of hazards to be assessed consistently while accounting for local conditions that influence potential humanitarian impact.

To support rapid and actionable decision-making, GDACS classifies events using a standardised three-level colour-coded alert system (green, orange, red), accompanied by a numerical score. Alert levels are designed to reflect the likelihood that an event may require international humanitarian assistance rather than the physical magnitude of a hazard alone. As a result, events with similar physical characteristics may receive different alert levels depending on population density, infrastructure exposure, and the affected regions' ability to cope with the event.

Geospatial modelling plays a central role in the GDACS impact assessment. Hazard data is combined with global population and settlement datasets to identify affected areas, estimate exposed populations, and define areas of interest for response planning and satellite tasking. For rapid-onset hazards such as earthquakes, tsunamis, and tropical cyclones, automated models prioritise timeliness while minimising false alerts. For slower-onset or complex hazards, including floods, droughts, volcanic eruptions, and wildfires, expert validation is incorporated to refine event characterisation and improve reliability.

[12]

3.2 Ushahidi

An open-source crowdsourcing and crisis-mapping platform created in 2008 by Ory Okolloh in response to post-election violence and media censorship following the Kenyan elections. It was developed to address information gaps, delays, and a lack of transparency during crises by allowing citizens to report events directly when traditional communication channels are unavailable, restricted, or slow. The platform gathers information from multiple public input channels, including SMS, web forms, email, social media posts (e.g. Twitter), photos, videos, and voice reports, allowing affected populations to act as real-time "human-sensors". Incoming resorts are translated, categorised, geolocated, and moderated using a combination of volunteer human intelligence and limited automation before being displayed. Rather than issuing automated alerts, Ushahidi primarily disseminated information through a live, interactive map, where incidents are symbolised spatially using markers, clusters, and categories, enabling responders, organisations, and the public to gain situational awareness. In several deployments, the platforms also support coordination by linking reported needs with offers of assistance, facilitating bottom-up humanitarian response. Ushahidi's effectiveness relies on deployment design, data verification process, and the social context in which it is implemented. [13]

3.3 PulsePoint Respond

A smartphone-based emergency notification application was developed in the early 2010s by the PulsePoint Foundation to increase the chances of survival for people experiencing out-of-hospital cardiac arrest. The platform was created to address a critical gap in emergency response: the delay between emergency dispatch and the initiation of life-saving interventions, such as cardiopulmonary resuscitation (CPR) and automated external defibrillators(AEDs). PulsePoint targets citizens who are willing and able to provide basic life support, including trained responders such as healthcare professionals and members of the public. The system is integrated directly with the local 911 emergency dispatch infrastructures. It automatically receives data from computer-aided dispatch systems when a suspected cardiac arrest is reported in a public location. When predefined dispatch criteria are met, PulsePoint sends real-time alerts to registered users within a predefined radius of the incident, displaying the event location on a map and nearby publicly accessible AEDs. By notifying nearby volunteers simultaneously with emergency medical services, PulsePoint aims to mobilise bystanders before professional responders arrive, therefore increasing rates of early CPR and defibrillation and strengthening community-based emergency response. [14]

4 Proposed System — ResQia

4.1 Overview of ResQia

The proposed system, **ResQia** (a portmanteau of Rescue and Immediate Assistance), is a native Android application designed to enable real-time, community-based emergency management. While modern smart cities rely on complex IoT sensors and professional dispatch, a critical ”zero-minute” gap remains between the occurrence of a hazard and the arrival of official responders. ResQia is developed to bridge this gap by transforming ordinary citizens into active ”human sensors” within a decentralised, hyper-local network.

4.1.1 Purpose and Objectives

The primary goal of ResQia is to minimise the ”information gap” that characterises the early stages of an emergency. By leveraging the ubiquity of mobile technology, the application enables the immediate dissemination of high-fidelity situational data, including precise GPS coordinates, time, photographic evidence and descriptive context.

The system serves two distinct strategic functions based on the nature of the event:

- **For Medical Emergencies:** The objective is to foster a ”human-sensor” ecosystem. By identifying and mobilising the closest possible helpers, the system facilitates life-saving interventions in the ”Critical Window”[1] before professional Emergency Medical Services (EMS) can arrive on the scene.
- **For General Urban Hazards:** The system functions as a decentralised situational awareness tool. It allows citizens to stay informed of surrounding threats through a transparent, peer-to-peer information stream. Unlike traditional social media platforms. which are often subject to algorithmic manipulation, echo chambers, or delayed trending cycles, ResQia ensures that verified citizens distribute information in real-time. This creates a ”Source-to-User” pipeline that prioritises raw, objective data over sensationalism, ensuring community members receive the most accurate and timely representation of local safety conditions.

[1] In this thesis, the term *Critical Window* refers to the short period immediately after a medical emergency occurs, during which rapid intervention can significantly reduce the risk of severe deterioration, injury, or death.

4.1.2 Target Audience

The system is designed to serve as two primary, overlapping user groups, both of whom are notified through a multi-tiered geospatial notification strategy:

- **General Users:** These are ordinary citizens who witness an emergency and require a streamlined, low-friction method to report it. Because these users often operate under high physiological stress, the system is designed to prioritise visual evidence and automated geolocation over complex text entry, making it accessible to anyone with an Android smartphone.
- **Community Helpers:** These are individuals with verified medical or emergency response backgrounds, such as off-duty doctors, nurses, paramedics or individuals who have completed certified first-aid training courses. Unlike general users, these users possess the professional expertise required to perform life-saving interventions. In addition to the standard features available to general users, Community Helpers are granted elevated privileges within ResQia.This allows them to receive high-priority alerts for medical emergencies in their immediate vicinity, enabling them to provide expert care during the ”Critical Window” before EMS arrives.

4.1.3 Problem Alignment and Differentiation

ResQia builds on the foundations of established emergency platforms and adds specific optimisations to address the functional limitations in their current deployment methods. The system is differentiated through three key dimensions:

- **Operational Scale:** While the **GDACS** system is optimised for macro-level, international humanitarian responses to large-scale natural disasters (e.g. tsunamis or earthquakes), ResQia is engineered for granular. street-level incidents. It provides a dedicated platform for ”everyday” emergencies, such as localised fires or individual medical distress, which may not trigger international alerts but require immediate, hyper-local intervention.
- **Automation and Latency:** Although **Ushahidi** pioneered the use of crowdsourced crisis mapping, its reliance on manual moderation and volunteer-led data processing can introduce critical delays in high-velocity situations. ResQia mitigates this latency by automating the reporting pipeline. Using native smartphone hardware (GPS and Camera APIs), the system generates and dispatches geolocated alerts instantly, eliminating the need for intermediary human filtering.)
- **Versatility of Functional Scope: PulsePoint Respond** provides a highly successful model for specialised medical interventions, specifically targeting out-of-hospital cardiac arrests. ResQia adopts this ”nearby responder” logic but expands the operation scope into a multi-

hazard framework. While the system provides a unified dashboard for situational awareness across various categories, such as fire, traffic accidents, and security threats, it utilises a specialised, high-priority notification stream for a broad range of medical emergencies. Unlike PulsePoint, which is primarily restricted to cardiac events, ResQia identifies and alerts verified helpers to any medical incident requiring intervention during the critical window, ensuring a more comprehensive approach to community-based emergency response.

4.1.4 Data Integrity and Trust Model

To ensure the reliability of crowdsourced data, ResQia implements a decentralised verification layer. Unlike traditional reporting systems that rely on centralised moderation, ResQia utilises a peer-to-peer validation mechanism in which users at the scene can confirm or flag reports in real time. This self-healing data model, combined with secure authentication for Community Helpers, ensures that the system maintains a high signal-to-noise ratio, prioritising safety and actionable information over unverified content.

4.2 Requirements Analysis and System Modelling

The requirements for ResQia are grounded in the necessity for rapid, high-fidelity data transmission and the mobilisation of specialised community resources during the ”Critical Window” of a medical emergency.

4.2.1 Functional Requirements (FR)

- **FR-1: One-Touch Reporting:** The system shall provide quick-access buttons for common emergencies (CPR, Fire, Car Accident) to minimise user input time during high-stress situations.
- **FR-2: Multi-media Evidence Collection:** The system shall allow users to optionally attach a photo or text description to an incident report.
- **FR-3: Automated Geospatial Tagging:** The system must automatically retrieve and attach the user’s GPS coordinates to every emergency report.
- **FR-4: Verified Volunteer Onboarding:** The system shall provide a workflow for users to upload medical training certificates for manual administration review.
- **FR-5: Multi-Tiered Geospatial Notification Strategy:** The system shall implement a two-tiered alert mechanism based on proximity and user role. A Standard Proximity Alert is dispatched to all users within a 7 Km radius of an incident to ensure broad situational

awareness. For medical emergencies, the system shall trigger a secondary, high-priority Call-to-Action for verified Community Helpers. This additional notification is sent only to the helper's real-time location that is within walking distance during the "Critical Window", as calculated by the system's logic.

- **FR-6: Peer-to-Peer Incident Validation:** The system shall implement a crowdsourced verification mechanism allowing users to "Like" or "Dislike" an active report. This pattern ensures data integrity; an incident may require multiple reports or positive validations to be elevated to a high-priority community alert, while "Dislikes" allow the community to flag resolved or false incidents.
- **FR-7: Integrated Emergency Navigation:** For users responding to an alert, particularly Community Helpers, the system shall provide a direct link to external mapping services (e.g., Google Maps). This feature shall automatically export the incident GPS coordinates to the navigation app to provide the most efficient route to the scene.

4.2.2 Non-Functional Requirements (NFR)

- **NFR-1: Low Latency:** The end-to-end delay from incident reporting to notification delivery must be minimised, particularly for medical emergencies, to ensure the "Critical Window" is utilised effectively.
- **NFR-2: High Availability:** Since the platform is for emergencies, the backend (Firebase) must ensure near-constant uptime.
- **NFR-3: Data Accuracy:** The system must rely on high-accuracy GPS providers to prevent incorrect location dispatch.
- **NFR-4: Resource Efficiency:** The mobile client shall utilise local caching for recent events to ensure faster load times and reduced data consumption.

4.2.3 Use Case Identification

The ResQia ecosystem involves three primary actors that interact with the system:

1. **Citizen (General User):** Reports hazards and receives awareness alerts.
2. **Verified Volunteer (Community Helper):** Additionally, from the actions a general user has, they also receive specialised medical alerts and provide on-site aid if possible.
3. **Administrator:** Additionally, from the actions a general user has, they also manage volunteer certifications.

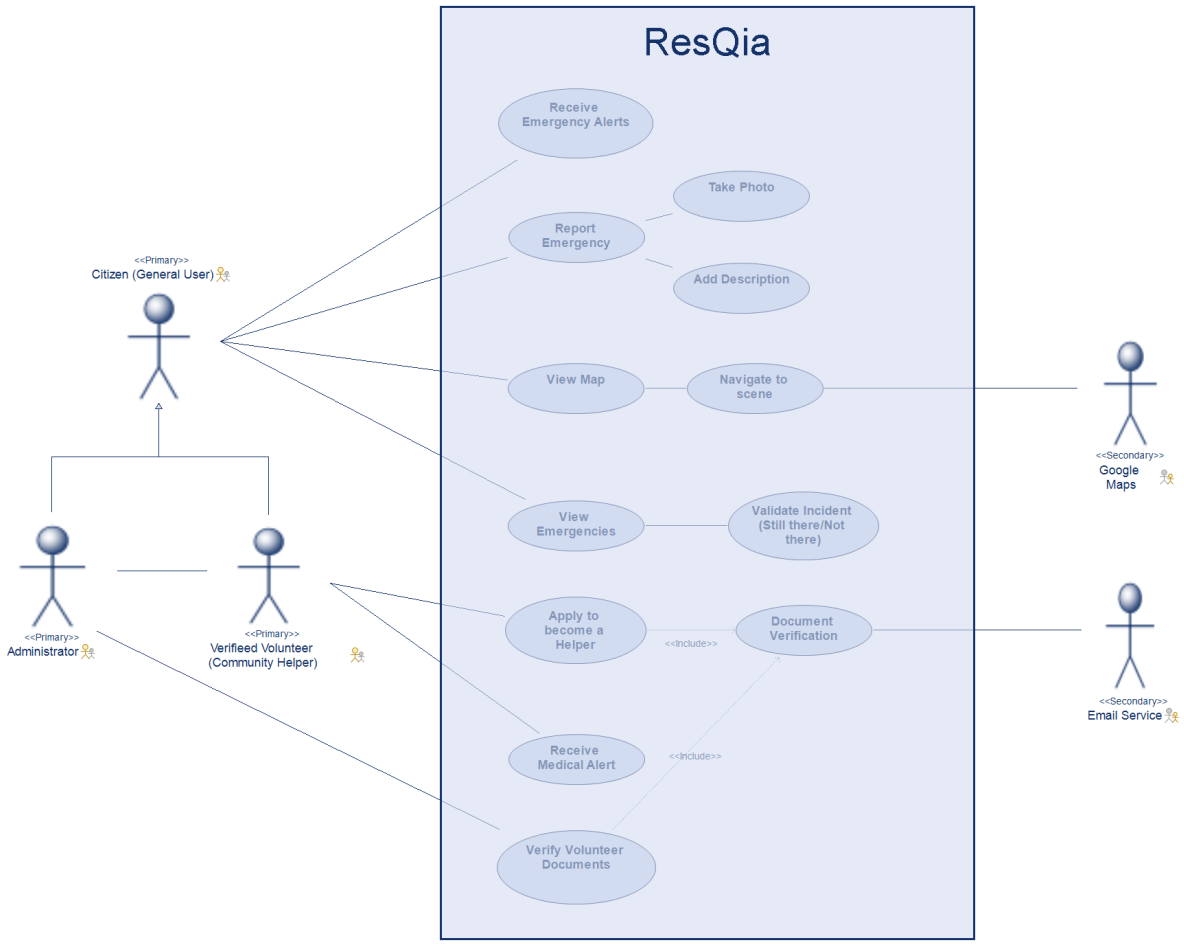

Figure 4.1: Use Case Diagram of the ResQia system.

4.2.4 Use Case Diagram

The use case diagram presented in Figure 4.1 delineates the functional requirements and the primary interactions between system actors and the ResQia platform. Furthermore, this Diagram serves as a high-level architectural representation, clarifying the application's operational scope and the role-based access control mechanisms governing user interaction.

The **Citizen** (General User) is the foundational actor on the platform. Their responsibilities include starting incident reports, optionally adding multimedia content (images, descriptions), and receiving real-time geo-targeted emergency alerts. Additionally, the Citizen actor plays an integral role in the crowdsourced data-validation process; by flagging incidents as either "active" or "resolved", users contribute to the overall data integrity and real-time reliability of the emergency information disseminated through the system.

The **Verified Volunteer**(Community Helper) actor extends the capabilities of the Citizen by facilitating specialised emergency response. Beyond standard reporting, this actor is designated to receive high-priority medical alerts based on geographical proximity. To improve response

efficiency, the platform provides integrated navigation services to help volunteers reach the incident site. This hierarchical extension demonstrates the system's commitment to a role-sensitive emergency mobilisation strategy.

The **Administrator** actor is responsible for the platform's certification governance and validation tasks. The Administrator role is a superset of the other platform roles. An Administrator has all the functional capabilities of a Citizen, such as reporting incidents and receiving alerts, and may also be granted Verified Helper status. This design enables Administrators to participate in the community while fulfilling their supervisory duties. Their primary administrative responsibility is to vet prospective community helpers by reviewing the credentials and supporting documents that users submit. By assigning specialised response privileges only to qualified personnel, the Administrator keeps the platform safe, credible, and running smoothly.

In short, the use case diagram shows that ResQia goes beyond simple emergency reporting. The platform fosters a collaborative, role-aware workflow that integrates incident reporting, community-driven verification, targeted notification dispatch, and administrative supervision into a cohesive unified system.

5 System Architecture and Design

5.1 System Architecture Design

The ResQia platform is a distributed, service-oriented mobile system designed to support real-time emergency reporting, geospatial coordination, and collaborative incident management. The system employs a **Backend-as-a-Service (BaaS)** model, strategically delegating resource-intensive backend operations, such as persistent data storage, identity management, and asynchronous notification dispatch to professionally managed cloud services.

The architecture is centred on the ResQia Android application, which serves as the primary client-side interface. By decomposing the platform into specialised, decoupled modules, the system ensures a strict separation of concerns, enabling high responsiveness in time-critical emergency scenarios. Figure 5.1 illustrates the high-level architectural design and the interaction flows between the internal software layers and the external service providers.

5.1.1 Architectural Domain Decomposition

The ResQia system is structured into four primary architectural domains:

1. **Android Client Layer:** The presentation and user interaction hub.
2. **Cloud Backend Layer:** The centralised infrastructure for state and identity management.
3. **Notification and Messaging Layer:** The engine for asynchronous, geo-targeted alerts.
4. **External Integration Layer:** Specialised third-party APIs for media and spatial processing.

5.1.2 Android Client Layer

The ResQia Android application is engineered as an event-driven interface. To ensure long-term maintainability and modularity, the client implementation adopts the **Model-View-ViewModel (MVVM)** architectural pattern, which logically separates the user interface from the application logic and data handling.

- **Presentation Layer:** Contains the visible UI components, including the Incident Dashboard, Reporting interface, and geospatial(map) views. These views are optimised for low-friction user interaction, essential for high-stress emergency reporting.
- **ViewModel Layer:** Serves as the application's "logic hub", managing the UI state across lifecycle events (e.g, screen rotations) and coordinating asynchronous workflows such as

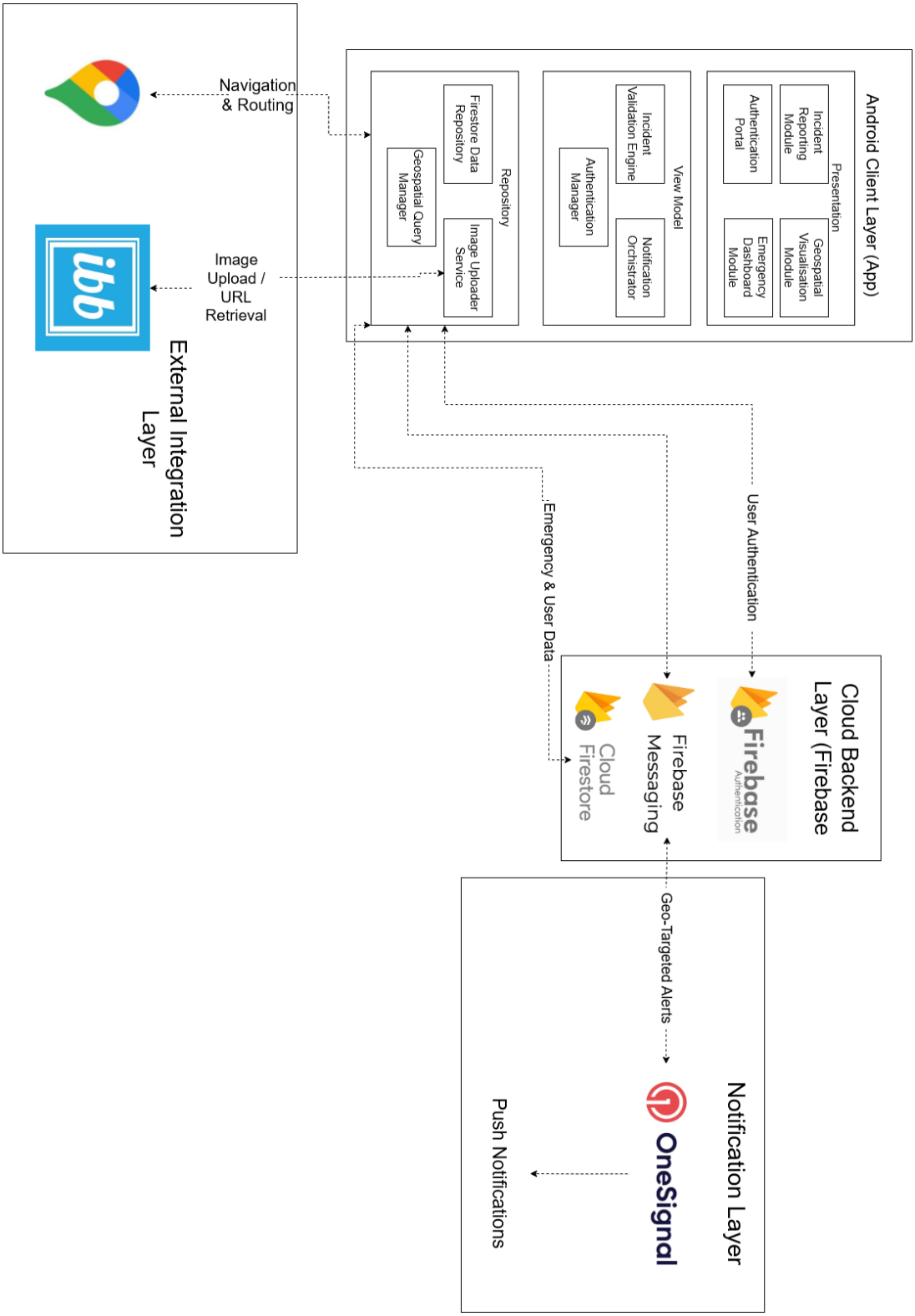

Figure 5.1: High-level System Architecture of the ResQia Platform

incident validation and authentication state management.

- **Repository Layer:** Acts as a data abstraction interface, providing a single source of truth. It abstracts the complexity of communication with external services, allowing the higher layers to remain agnostic of backend implementation details.

5.1.3 Cloud Backend Layer

The backend infrastructure is centralised within the Firebase ecosystem, leveraging managed services to ensure high availability.

- **Firebase Authentication:** Facilitates secure identity management, handling registration, session persistence, and role-based access control.
- **Cloud Firestore:** A NoSQL document-oriented database that serves as the system backbone. Its real-time synchronisation capabilities enable the instantaneous propagation of incident metadata, such as validation counts, GPS coordinates, and status updates, across all connected clients.

5.1.4 Notification and Geospatial Layer

- **OneSignal Integration:** Real-time alert dissemination is managed via One/signal. The platform uses advanced routing to evaluate proximity, ensuring high-priority medical reach is verified with minimal latency.
- **Google Maps API:** The navigation layer provides spatial awareness and route guidance. By allowing seamless transitions between the application interface and turn-by-turn navigation, the system reduces the cognitive load on responders during critical interventions.

5.1.5 System Operational Workflow

The system operates as a receive state machine. The operational sequence is defined as follows:

1. **Authentication:** The user verifies their identity via Firebase.
2. **Submission:** When an emergency occurs, the user completes and submits an incident report, thereby committing the associated metadata to Cloud Firestore.
3. **Media Persistence:** Concurrent to data storage, multimedia evidence is offloaded to the **imgbb** API, with only the resulting persistent URI stored in the database.
4. **Targetted Alerting:** The backend notification engine evaluates the geographical context of the incident and dispatches alerts via OneSignal.

5. **Collaborative Validation:** Users interact with the incident dashboard; these validations (active/resolved status) trigger real-time updates that synchronise across the entire user base.

5.1.6 Architectural Benefits

The proposed system offers several key engineering advantages:

- **Scalability:** Managed services ensure the system can handle growth for rapid emergency visibility.
- **Low Latency:** Real-time data synchronisation allows for rapid emergency visibility.
- **Maintainability:** The layered design simplifies updates and technical debugging.
- **Extensibility:** The modular structure allows for the future integration of features such as AI-based prioritisation or a municipal data dashboard, without disrupting core operations.

5.2 Database Design and Data Model

The ResQia platform adopts a document-oriented data model, utilising **Cloud Firestore** as its primary persistent layer. In contrast to traditional Relational Database Management Systems (RDBMS) that rely on fixed, normalised schemes and complex JOIN operations, Firebase employs a collection-document architecture. This paradigm offers significant advantages for the ResQia system, including schema flexibility, high-performance retrieval of semi-structured data, and native real-time synchronisation, ensuring state consistency across distributed mobile clients.

Figure 5.2 illustrates the conceptual Entity-Relationship (ER) model. It is essential to distinguish that this diagram is a **logical abstraction** of the system's data requirements rather than the exact blueprint of the physical database implementation. While the ER diagram represents the theoretical relationships among users, incidents, and interactions, the physical schema implemented in Firestore leverages denormalisation, a design strategy that embeds related data directly within documents to reduce read latency.

5.2.1 Conceptual Data Model

The ER model identifies the primary entities involved in the system and describes their relationships to provide a framework for how the application architecture perceives data.

- **User Entity:** The User Entity acts as the identity hub of the platform. It encapsulates profile data and role-based permissions, including the primary identified (UserUID), contact

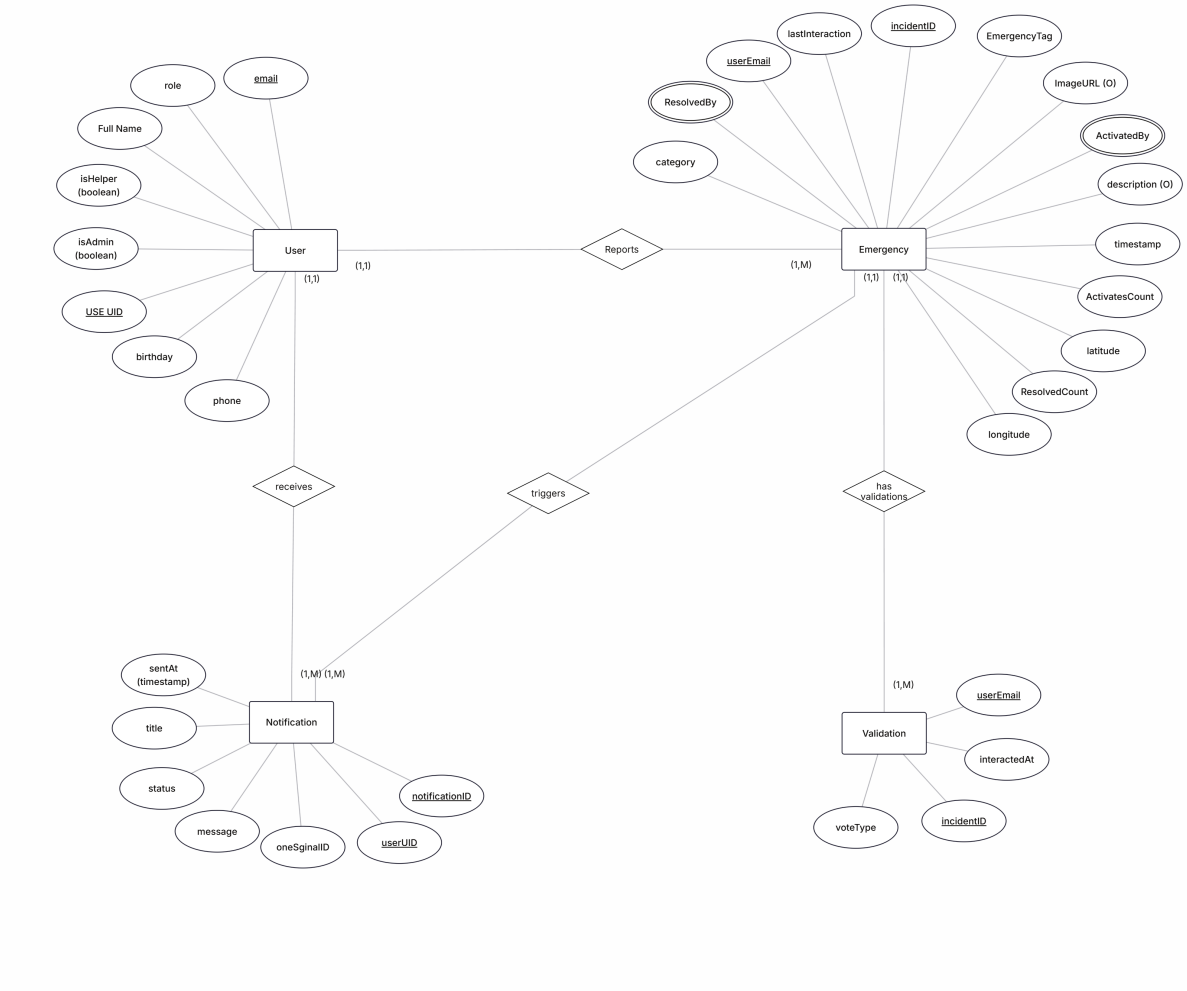

Figure 5.2: Entity–Relationship (ER) diagram of the emergency reporting system database.

details (email, phone), demographic attributes (birthday), and privileged flags (isAdmin, isHelper, and role).

- **Emergency Entity:** The Emergency Entity is the central operational unit of the system. It maintains the state of reported incidents, capturing metadata such as Incident ID, incident categorisation, user-defined descriptions, multimedia references (image URLs), and critical geospatial coordinates (latitude and longitude).
- **Logical Entities (Validation and Notification):** These entities are included in the conceptual model to preserve completeness. The Validation entity represents the community consensus mechanism, tracking interaction between users and specific reports. The Notification entity represents the asynchronous alert lifecycle, defining how information is delivered to users via OneSignal based on incident context.

5.2.2 Relationships Between Entities

The system defines five primary relation flows:

1. **User to Emergency (1:N):** A single user may report multiple incidents over time.
2. **Emergency to Validation (1:N):** A single incident can accumulate multiple validation entries.
3. **User to Validation (1:N):** A user may validate multiple distinct incidents.
4. **Emergency to Notification (1:1):** A reported emergency may trigger one geo-targeted alert.
5. **User to Notification (1,N):** A user may receive multiple emergency alerts.

These relationships are conceptual; in the actual implementation, they are managed through document references and embedded arrays rather than traditional foreign key constraints.

5.2.3 Physical Data Model (Firestore Implementation)

To optimise for mobile performance, the physical schema diverges from the normalised ER model toward a denormalised, read-optimised structure.

- **Users Collection:** Each document is keyed by a unique UserUID. Each document acts as a complete profile object containing authentication-related metadata and user role definitions, ensuring that session-specific checks (e.g, verifying if a user is a helper) are performant.
- **Emergencies Collection:** This collection stores each report as an autonomous document. By embedding validation metrics, such as activeCount, resolvedCount, and the ”Acti-

vatedBy" array (a list of user emails), the database eliminates the need for expensive cross-collection lookups.

5.2.4 Design Rationale

Three primary architectural requirements drive the decision to utilise a denormalised schema:

- **Read-Time Efficiency:** In high-concurrency emergency scenarios, reducing the number of database calls is vital. Denormalisation allows the application to retrieve a complete incident record, including its current community consensus, in a single, atomic read operation.
- **Network Optimisation :** By embedding interaction metrics within the emergency document, the system minimises the latency overhead inherent in asynchronous network requests.
- **System Responsiveness:** Real-time synchronisation is maximised when data is logically contained, ensuring that users receive the most current incident status without waiting for multiple query resolutions.

5.2.5 Summary

The ResQia data model effectively bridges the gap between conceptual clarity and physical performance. We use a logically structured ER diagram for design transparency and a denormalised Firestore Schema for real-world execution, ensuring rapid data propagation and a smooth user experience during time-critical emergency interventions.

5.3 Notification and Proximity Logic

The notification and Proximity engine is the core architectural component responsible for bridging the gap between an emergency event and a trained rescuer. The system is designed to minimise "alert latency", the duration between the occurrence of an incident and the receipt of a notification by a volunteer. This logic is governed by two parameters: the **temporal window** (critical response time) and the **spatial radius** (proximity threshold).

5.3.1 Clinically-Informed Temporal Window

The system assigns a critical response window, measured in minutes, to each type of medical emergency. These time windows were defined based on clinical guidance from a medical professional to approximate the maximum time within which nearby helpers can provide meaningful

assistance before the patient's condition becomes more critical. The values are not intended to replace official emergency medical protocols, but are used as practical thresholds for prioritising proximity-based notifications within the ResQia system.

- **Cardiac Arrest and Unconscious Person (3 minutes):** These cases require immediate attention because delays in basic support can rapidly reduce the patient's chance of survival. Therefore, the system uses a short 3-minute response window to prioritise helpers who are very close to the incident.
- **Choking (10 minutes):** Choking incidents may vary in severity depending on whether the airway is partially or fully blocked. The 10-minute window allows the system to notify nearby helpers while still prioritising quick intervention.
- **Severe Bleeding (3 minutes):** Severe bleeding can become life-threatening within a short period of time. For this reason, the system applies a 3-minute response window so that only helpers within very close proximity are targeted.
- **Anaphylaxis (15 minutes):** Anaphylaxis can progress quickly and may require urgent assistance. A 15-minute window is used to notify nearby helpers who may be able to support the affected person while professional responders are en route.
- **Seizure (5 minutes):** Seizures that continue for several minutes may become medically serious. The system therefore uses a 5-minute response window to prioritise helpers who can arrive quickly and provide immediate support if needed.

5.3.2 Proximity Logic and Special Constraints

The ResQia notification architecture employed a multi-tiered spatial strategy to enhance community awareness and the efficiency of professional response.

1. **General Proximity Broadcast (Awareness Layer):**

   All registered users receive a notification if an emergency event is reported within a 7km radius of their current location. This default spatial parameter serves as a community awareness mechanism, ensuring that residents within a reasonable proximity are alerted to potential hazards in their immediate environment. The 7km threshold is calibrated to balance volunteer density with local situational awareness, ensuring a sufficient pool of observers without causing widespread ”alert fatigue”.

2. **Targeted Intervention Alert (Helper Layer):**

   For events categorised as medical emergencies, the system executes a secondary, high-priority notification pipeline. This alert is exclusively dispatched to users registered as ”Helpers” (e.g., individuals with medical or first-aid training). Unlike the general broad-

cast, the eligibility for this alert is determined by a strict spatio-temporal filter: a helper only receives the notification if their current location is within a *walkable distance* of the incident, such that the estimated transit time does not exceed the emergency-specific *criticalWindow* defined in 5.3.1.

By differentiating between general community awareness and time-critical professional mobilisation, the system ensures that ”Helpers” are not overwhelmed by non-urgent alerts, while simultaneously guaranteeing that trained responders are mobilised only when they can feasibly reach the victim within the clinically mandated timeframe.

5.4 Security and Authorization

The security architecture of the ResQia platform is designed to ensure data integrity, protect user identity, and enforce granular access control. By separating the authentication process from the authorisation model, the system maintains a robust defence against unauthorised data manipulation.

5.4.1 Identity Management and Authentication

User identity is established through Firebase Authentication, utilising a secure email-and-password-based registration flow. Upon successful authentication, the system links the generated unique user identifier to a persistent profile within the Firestore `users` collection. This mapping ensures that every unique authentication token corresponds to a centralised record maintaining the user’s metadata and role-based status (e.g., `isHelper`, `isAdmin`).

5.4.2 Data-Driven Authorization Model

The system enforces authorisation via a Data-Driven Model. Rather than relying on volatile client-side state management, the application utilises the database as the ”Single Source of Truth”.

- **Permission Enforcement:** Role-based checks are performed by validating the user’s role attribute (e.g., `role`, `isHelper`) stored within their unique Firestore document. When a user logs in, the application restricts access to features not available to them.
- **Architecture Security:** This approach ensures that privileges are managed centrally. Any updates to user access levels propagate immediately across the client environment, maintaining consistency and preventing privilege escalation attacks.

5.4.3 Reliability Through Crowdsourced Validation

To maintain the integrity of incident reports, the system implements a crowdsourced validation mechanism. Incident reliability is determined programmatically by computing an aggregate trust score from the community engagement (Active/Resolved ratio) stored in each `Emergency` document.

Design Rationale: This decentralisation of validation minimises the risk of false alarms. An incident is programmatically marked as ”High Reliability” only when it meets a defined consensus threshold. This ensures that high-acuity medical alerts are prioritised for the volunteer network based on objective community data rather than subjective, single-source reporting.

6 System Implementation

6.1 Development Environment and Tools

The development of the ResQia platform was carried out in the Android Studio Integrated Development Environment (IDE), utilising Java as the primary programming language. The project management and build process are orchestrated through Gradle, which handles dependency management, versioning, and the complication of the application's various modules.

The technical stack is composed of the following core technologies:

- **UI Architecture and Design:** The application's user interface was constructed using XML (Extensible Markup Language). This declarative approach allowed for a clear separation between the UI structure (layout Files) and the underlying Java logic. This separation of concerns ensured a maintainable codebase, as UI components could be refined independently of the backend functionality.
- **Backend Infrastructure:** The application utilizes Firebase as its Backend-as-a-Service (BaaS) platform. Specifically, Firestore is used for real-time, document-oriented data storage, while Firebase Authentication provides a secure, managed identity system for user account management.
- **Geospatial Services:** To provide location-aware functionality, the application integrates the Google Maps SDK for Android. This enables the rendering of interactive maps, the visualisation of emergency markers, and the calculation of spatial proximity via the Fused Location Provider API.
- **External Application Interfacing:** To maintain a lean application architecture and ensure adherence to Android's security best practices, the ResQia system employs an Intent-based design pattern to delegate specialised tasks to the user's native device applications. This approach effectively offloads complex, data-sensitive operations to established, high-fidelity native services.
    1. **Communication Interfacing:** For tasks such as submitting volunteer verification documentation, the application utilises the `ACTION_SENDTO` intent. Rather than implementing an internal email client, which would introduce unnecessary overhead, security risks related to password handling and maintenance complexity, the system constructs a structured URI intent. This delegates the email composition process to the user's preferred, pre-configured native email application. This ensures that the user maintains full control over their correspondence while benefiting from the

security of their established, authenticated email client.

2. **Geospatial Deep Linking:** For emergency response incident site navigation, the system employed Native Deep Linking. Upon the user selecting an incident, the application constructs a `google. navigation` URI intent. This triggers the device's installed Google Maps application, offloading the pathfinding, real-time traffic analysis, and turn-by-turn routing logic. With this architectural decision, the application connects incident discovery to real-time response without requiring an internal, resource-intensive routing engine or the additional costs of server-side navigation APIs.

- **Notification Dispatch Engine:** Push notification functionality is facilitated through a dual-service architecture. The application integrates OneSignal for advanced, filter-based notification delivery, which operates in tandem with Firebase Cloud Messaging (FCM) to ensure high-priority messages are delivered to users based on their geolocation and volunteer status.

This integrated toolchain facilitates a responsive, real-time user experience while maintaining security by offloading specialised tasks, such as email and location rendering, to established, high-fidelity native services.

6.2 Class Architecture and Component Organisation

The ResQia application is implemented using a modular, fragment-based structure. While the overall system architecture was introduced in Chapter 5, this section focuses on the internal class organisation of the Android application. Figure 6.1 presents the finalised class diagram, showing the main fragments, data adapter, service components, and data models used in the implementation.

The architecture is organised into four distinct functional areas:

- **Fragment-Based UI Composition:** The application logic is distributed across specialised `Fragment` classes (e.g., `HomeFragment`, `ProfileFragment`, `CapturePhotoFragment`). This "Single-Activity, Multi-Fragment" design allows each screen to manage its own lifecycle and UI state independently. This modularity ensures that the user interface remains responsive and the codebase is maintainable, as logic for specific features is isolated within their respective fragments.
- **Data Modelling and Binding:** The `EmergencyPost` class serves as the fundamental data model, acting as a bridge between the Firestore Database and the application's UI. The `EmergencyAdapter` serves as the controller for the dashboard; it binds incident data to the

`RecyclerView` components, ensuring the view updates dynamically as incident reports are fetched from the database.

- **Component Communication:** To maintain low coupling between components, the system utilises an event-driven communication pattern. AS depicted by the dashed associations in the class diagram, the system employs reactive listeners and the `FragmentResultListener` pattern to allow fragments to exchange data (such as image paths or authentication status) without direct instantiation. This ensures that the components remain decoupled and robust.
- **Service Isolation:** The architecture differentiates between core UI operations and high-latency, resource-intensive tasks. Components such as `OkHttpClient` and the OneSignal notification engine are isolated from the main application thread. This architectural separation prevents blocking operations during network-dependent tasks, such as image uploads or report submissions, ensuring the application remains functional even under poor network conditions.

This structure allows the application to achieve its primary objectives: real-time geospatial incident discovery, efficient reporting, and seamless volunteer coordination. The following sections detail the implementation of these specific modules, beginning with the authentication and profile management logic.

6.3 Core System Components

6.3.1 Authentication and Profile Management

The authentication and Profile Management module serves as the primary security gateway to the ResQia system. Its architecture relies on a synergistic integration between Firebase Authentication, which manages secure identity lifecycles, and Cloud Firestore, which provides a scalable, NoSQL document-based data layer for user profiles and role-based permissions.

Secure Identity Provisioning

The application manages user sessions through `LoginActivity` and `RegisterActivity`. The authentication pipeline is designed to be asynchronous, utilising Firebase's built-in token management to persist session states across application restarts. When a new user registers, the system executes an atomic operation: it first creates the user credential via `mAuth.createUserWithEmailAndPassword()`, upon success, it initialises a corresponding document in the Firestore `users` collection. This ensures data consistency by linking the Firebase-generated `UID` (Unique Identifier) to the application-specific profile metadata, such as phone numbers and verification roles.

Figure 6.1: Class Diagram

Listing 6.1: Atomically creating user credentials and initializing Firestore profile metadata

```
\\ Functions in RegisterActivity.java
private void performRegistration() {
// ...
// Validation logic implemented previously...
// ...
    mAuth.createUserWithEmailAndPassword(email, password)
                .addOnCompleteListener(task -> {
                    if (task.isSuccessful()) {
                        saveUserToFirestore(fullName, email, phone, birthday);
                    } else {
                        Toast.makeText(RegisterActivity.this, "Auth Failed: " +
                            task.getException().getMessage(),
                            Toast.LENGTH_LONG).show();
                    }
                });
}
private void saveUserToFirestore(String fullName, String email, String phone,
    String birthday) {
        if (mAuth.getCurrentUser() == null) return;
        String userId = mAuth.getCurrentUser().getUid();

        Map<String, Object> user = new HashMap<>();
        user.put("fullName", fullName);
        user.put("email", email);
        user.put("phone", phone);
        user.put("birthday", birthday);
        user.put("helperToggle", false);
        user.put("isHelper", false);
        user.put("isAdmin", false);
        user.put("role", "user");

        db.collection("users").document(userId).set(user)
                .addOnSuccessListener(aVoid -> {
                    Toast.makeText(RegisterActivity.this, "Account Created
                        Successfully", Toast.LENGTH_SHORT).show();
                    startActivity(new Intent(getApplicationContext(),
                        MainActivity.class));
                    finish();
```

```
            })
            .addOnFailureListener(e -> Toast.makeText(RegisterActivity.this,
                "Firestore Error: " + e.getMessage(),
                Toast.LENGTH_SHORT).show());
    }
```

Role-Based Authorisation and State Synchronisation

Once authenticated, the `ProfileFragment` serves as the interface for state synchronisation. The module performs real-time queries on Firestore to determine the user's privilege level. This is accomplished by retrieving the `isHelper`, `isAdmin`, and `helperToggle` flags. The application logic then dynamically modifies the UI, specifically, rendering administrative tables or helper status dashboards based on these flags.

This reactive approach ensures that the user interface always reflects the latest backend state. For instance, in the `loadPendingUsers()` method, the system queries the `users` collection using a compound filter that targets users where `helperToggle` is `true` and `isHelper` is `false`. This implementation demonstrates an effective use of backend-side filtering, which reduces the payload size and improves performance compared to client-side data filtering.

Listing 6.2: Querying pending helpers using backend-side filtering

```
\\Function in ProfileFragment.java
 private void loadPendingUsers() {
       db.collection("users")
              .whereEqualTo("helperToggle", true)
              .whereEqualTo("isHelper", false)
              .get().addOnCompleteListener(task -> {
                 if (task.isSuccessful()) {
                     if (adminTable.getChildCount() > 1) {
                         adminTable.removeViews(1, adminTable.getChildCount() -
                             1);
                     }
                     for (QueryDocumentSnapshot document : task.getResult()) {
                         addAdminRow(document);
                         addTableDivider();
                     }
                 }
              });
    }
```

6.3.2 Volunteer Role Management and Verification

The system includes a dedicated module for managing user roles, allowing registered users to transition into ”Helper” status. This module balances user-controlled state management with an asynchronous verification process.

State Management and Role Activation The `BecomeHelperFragment` acts as the interface for role-based configuration. It utilises a Firestore backend persistence model to store the `helperToggle` state, which is directly consumed by the notification filtering logic. By synchronising the switch state with the user's document in the `users` collection, the system ensures that user preferences are maintained across sessions and devices.

Listing 6.3: Implementation of volunteer state toggling and verification request

```
// 1. Sync Switch state with current Firestore value
        db.collection("users").document(userId).get().addOnSuccessListener(doc
            -> {
            if (doc.exists() && doc.getBoolean("helperToggle") != null) {
                helperSwitch.setChecked(doc.getBoolean("helperToggle"));
            }
        });

// 2. Handle Switch Toggle
        helperSwitch.setOnCheckedChangeListener((buttonView, isChecked) -> {
            db.collection("users").document(userId)
                    .update("helperToggle", isChecked)
                    .addOnFailureListener(e -> Toast.makeText(getContext(),
                        "Update Failed", Toast.LENGTH_SHORT).show());
        });
```

Verification Workflow

To maintain the integrity of the volunteer network, the system requires a manual verification step. When a user requests to become a helper, the application triggers a platform-native email intent. This pre-populates an email draft containing the user's metadata, prompting the user to provide necessary documentation (e.g., medical certificates or first-aid credentials) to an administrator. This hybrid approach, combining automated state toggling with human-in-the-loop verification, ensures that only qualified individuals receive high-priority medical incident alerts.

6.3.3 Geospatial Incident Discovery and Map Visualisation

The Geospatial Incident Discovery module is the primary interface for situational awareness within the ResQia system. It enables users to visualise emergency incidents in real-time, filtered by geographic proximity. The module leverages the **Google Maps SDK for Android** to provide interactive mapping capabilities, while the `ClusterManager` utility is used to manage high marker densities effectively.

Proximity-Based Data Filtering

To optimise performance and maintain relevance, the application implements a radius-based filtering mechanism. Upon acquiring the user's current coordinates via the `FusedLocationProviderClient`, the system calculates the distance between the user and all incidents stored in Firestore. This is achieved using the `Location.distanceBetween()` method, which implements the Haversine formula. Mathematically, this determines the great-circle distance between the user's coordinate $P_1(\phi_1, \lambda_1)$ and the incident coordinate $P_2(\phi_2, \lambda_2)$ as:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

where ϕ represents latitude, λ represents longitude, and r denotes the Earth's radius. Incidents exceeding the `RADIUS_METERS` constant (defined as 7000 meters) are excluded from the dataset, ensuring that the map remains uncluttered and the visualised data remains actionable.

Listing 6.4: Proximity filtering logic using the Haversine distance calculation

```
// Inside function loadIncidents() in HomeFragment.java
if (incidentLat != null && incidentLng != null) {
                    // Haversine distance check
                    float[] results = new float[1];
                    Location.distanceBetween(userLat, userLng, incidentLat,
                        incidentLng, results);
                    float distanceInMeters = results[0];

                    // Filtering: only add if within 7km
                    if (distanceInMeters <= RADIUS_METERS) {
                        Timestamp ts = doc.getTimestamp("timestamp");
                        String dateStr = (ts != null) ? sfd.format(ts.toDate()) :
                            "Recent";

                        clusterManager.addItem(new EmergencyItem(
                                incidentLat, incidentLng,
```

```
                        doc.getString("emergencyTag"),
                        doc.getString("description"),
                        doc.getString("category"),
                        doc.getString("fullName"),
                        dateStr,
                        doc.getString("imageUrl")
                    ));
                }
            }
```

Map Visualisation and Marker Clustering

To handle the visualisation of multiple incident types (e.g., Fire, Medical, Security), the system employs a custom `ClusterRenderer`. This component dynamically maps incident categories to colour-coded markers, enhancing the user's ability to categorise emergency types at a glance. The use of `ClusterManager` is critical for scalability; it programmatically groups proximate markers into clusters at lower zoom levels, preventing UI occlusion and improving the overall readability of the map interface.

Listing 6.5: Dynamic category-based marker rendering

```
// Inside function onMapReady() in HomeFragment.java
clusterManager.setRenderer(new
    DefaultClusterRenderer<EmergencyItem>(requireContext(), mMap,
    clusterManager) {
            @Override
            protected void onBeforeClusterItemRendered(@NonNull EmergencyItem
                item, @NonNull MarkerOptions markerOptions) {
                // This pulls the Category (Natural, Medical, etc.) and picks
                    the hue
                float hue = getHue(item.getCategory());
                markerOptions.icon(BitmapDescriptorFactory.defaultMarker(hue));
            }
        });
```

Furthermore, the integration of a `Circle` overlay as a visual indicator of the user's active monitoring range. By programmatically updating its radius centred on the user's `LatLng` position, the application provides immediate visual configuration of the search area, bridging the gap between raw data visualisation and intuitive spatial navigation.

Listing 6.6: Dynamic rendering of the proximity radius overlay

```
// Inside HomeFragment.java
private void drawOrUpdateRadiusCircle(LatLng center) {
        if (mMap == null) return;
        if (radiusCircle != null) radiusCircle.remove();

        radiusCircle = mMap.addCircle(new CircleOptions()
                .center(center)
                .radius(RADIUS_METERS)
                .strokeWidth(3f)
                .strokeColor(0xFF1565C0)
                .fillColor(0x111565C0));
}
```

6.3.4 Emergency Reporting and Notification Pipeline

The Emergency Reporting and Notification Pipeline represents the system's core reactive capability. It transforms a localised incident report into a synchronised, location-aware alert disseminated to authenticated volunteers. This architecture relies on a RESTful integration between the Android Application and the OneSignal notification engine, facilitated by the `EmergencyDetailsReportFragment` and managed globally through the `MyApplication` lifecycle class.

Asynchronous Reporting and Cloud Persistence

The reporting workflow is designed for high-latency resilience. When a user submits an emergency report, the system performs an asynchronous multipart upload of image assets to the ImgBB cloud storage API. Upon successful retrieval of the image URL, the incident data, including geospatial coordinates, category, and timestamps, is committed to the Firestore `emergencies` collection. This two-phase commit process (Asset Upload followed by Metadata Persistence) ensures that the database remains lightweight while preventing UI blocking during network-intensive operations.

Intelligent Push Notification Logic

Once a report is successfully saved, the system triggers the notification pipeline. The application employs two distinct notification strategies based on the emergency context.

1. **General Incident Alerts:** Disseminated to all users within a 7km radius, excluding the reporter (via email tag filtering)
2. **Targeted Medical Alerts:** A high-priority, contextualised notification pipeline specif-

ically for medical emergencies. This logic utilises a `criticalWindow` calculation to estimate the required response time, dynamically adjusting the radius filter ($Radius = \text{Minutes} \times 83$) to ensure that alerts reach the first-aid providers.

Listing 6.7: Context-aware notification filtering using OneSignal REST API

```
// Inside EmergencyDetailsReportFragment.java in both
    sendOneSignalNotification() sendOneSignalNotificationSpecific()
    respectively.
// Radius Filter (Incident Centre)
        JSONObject loc = new JSONObject();
        loc.put("field", "location");
        loc.put("radius", String.valueOf(dynamicRadius));
        loc.put("lat", String.valueOf(latitude));
        loc.put("long", String.valueOf(longitude));
        filters.put(loc);

// Helper status filtering to ensure only volunteers receive medical
    alerts
        JSONObject helperFilter = new JSONObject();
        helperFilter.put("field", "tag");
        helperFilter.put("key", "isHelper");
        helperFilter.put("relation", "=");
        helperFilter.put("value", "true");
        filters.put(helperFilter);
```

Synchronisation of User Identity and Status

To ensure the integrity of the filtering pipeline, the system maintains a real-time synchronisation between the local Firebase Auth state and the OneSignal backend via MyApplication. By attaching tags, specifically `user_email` for sender exclusion and `isHelper` for role-based triggering, the system ensures that notification logic remains decoupled from the UI. This architectural approach, where tags are updated via Firestore `addSnapshotListener`, guarantees that a user's status (e.g., newly verified volunteer) is instantly reflected in the notification delivery logic without requiring an application restart or re-authentication.

Listing 6.8: Real-time synchronization of volunteer status with OneSignal tags

```
// Inside MyApplication.java in function syncHelperStatusWithOneSignal():
    Ensuring helper status is always current
db.collection("users").document(user.getUid())
                .addSnapshotListener((documentSnapshot, e) -> {
```

```
                if (documentSnapshot != null &&
                    documentSnapshot.exists()) {
                    Boolean isHelper =
                        documentSnapshot.getBoolean("isHelper");
                    String email = documentSnapshot.getString("email");

                    // 1. Sync Email Tag (Required for the 'Exclude
                        Sender' filter)
                    if (email != null) {
                        OneSignal.getUser().addTag("user_email", email);
                    }

                    // 2. Sync Helper Tag
                    // We use lowercase "true" as a string to match our
                        API filter
                    if (isHelper != null && isHelper) {
                        OneSignal.getUser().addTag("isHelper", "true");
                        android.util.Log.d("OneSignalSync", "Tag Set:
                            isHelper = true");
                    } else {
                        OneSignal.getUser().addTag("isHelper", "false");
                        android.util.Log.d("OneSignalSync", "Tag Set:
                            isHelper = false");
                    }
                }
            });
```

Finally, to maintain database integrity, the system implements a server-side clean-up routine within the `MainActivity`. The `cleanExpiredEmergencies()` method queries documents where the `last_interaction` timestamp exceeds a 24-hour threshold, ensuring the geospatial map is populated only with current, actionable data.

Listing 6.9: Automated cleanup of expired incident reports

```
for (QueryDocumentSnapshot document : task.getResult()) {

                    Timestamp lastInteraction =
                            document.getTimestamp("last_interaction");

                    if (lastInteraction != null) {
```

```
                long lastTime =
                        lastInteraction.toDate().getTime();

                if (lastTime < twentyFourHoursAgo) {
                    deleteEmergency(document.getId());
                }

            } else {
                // fallback if field missing
                Timestamp created =
                        document.getTimestamp("timestamp");

                if (created != null &&
                        created.toDate().getTime() <
                            twentyFourHoursAgo) {

                    deleteEmergency(document.getId());
                }
            }
        }
```

6.3.5 Data Persistence and Real-time Synchronisation

The data persistence layer of the ResQia system is architected to prioritise consistency and responsiveness, leveraging Firestore's real-time capabilities to ensure that all clients maintain an identical view of active emergency incidents. The system adopts a reactive programming model in which the UI layer observes the database state, ensuring that the `EmergencyDashboardFragment` remains up to date without requiring manual user intervention.

Real-time Data Observation

To achieve real-time synchronisation, the application employs the `addSnapshotListener` method on the Firestore `emergencies` collection. Unlike standard polling, which consumes unnecessary network bandwidth, this event-driven approach ensures that the client receives data updates only when the underlying database state changes. This is critical in high-pressure emergency scenarios where the most current information must be displayed immediately.

Listing 6.10: Reactive synchronization of the emergency dashboard via SnapshotListener

```
//Inside class EmergencyDashboardFragment
```

```
private void fetchDisplayPosts() {
      db.collection("emergencies")
              .orderBy("timestamp", Query.Direction.DESCENDING)
              .addSnapshotListener((value, error) -> {
                  if (error != null) {
                      // Handle error if necessary
                      return;
                  }
                  if (value == null) return;

                  emergencyList.clear();
                  for (DocumentSnapshot doc : value.getDocuments()) {
                      List<String> ActivatedBy = (List<String>)
                          doc.get("ActivatedBy");
                      List<String> ResolvedBy = (List<String>)
                          doc.get("ResolvedBy");

                      emergencyList.add(new EmergencyPost(
                              doc.getId(),
                              doc.getString("userEmail"),
                              doc.getString("fullName"),
                              doc.getString("description"),
                              doc.getString("imageUrl"),
                              doc.getString("emergencyTag"),
                              doc.getString("category"),
                              doc.getTimestamp("timestamp"),
                              doc.contains("ActivatesCount") ?
                                  doc.getLong("ActivatesCount") : 0L,
                              doc.contains("ResolvedCount") ?
                                  doc.getLong("ResolvedCount") : 0L,
                              ActivatedBy,
                              ResolvedBy
                      ));
                  }
                  adapter.notifyDataSetChanged();
              });
  }
```

Data Modelling and Defensive Mapping

The EmergencyPost class serves as a plain old Java <Object (POJO) that encapsulates the incident metadata. To ensure system stability, the implementation includes defensive null checks and type-safe casts during the mapping process. By handling missing fields (such as ActivatesCount or ResolvedCount) with ternary operators and initialising ArrayList objects for list-based attributes (ActivatedBy, ResolvedBy), the system prevents runtime NullPointerExceptions that often arise from sparse or malformed cloud data.

This reactive architecture ensures that, as soon as a new incident is created or an existing one is removed (via the cleanExpiredEmergencies() routine in MainActivity), the dashboard updates seamlessly across all active user sessions. This eliminates the need for manual page refreshes and provides a high-fidelity representation of local emergency situational awareness.

6.3.6 Inter-Component Communication

To maintain a decoupled and modular architecture, the application minimises direct dependencies between UI components. Instead, it utilises the Android FragmentManager API to facilitate inter-component communication. This approach ensures that fragments, such as the CapturePhotoFragment and the EmergencyDetailsReportFragment, remain independent, communicating through a centralised event bus rather than tight class-to-class coupling.

Fragment Result API implementation

The system leverages the FragmentResultListener for passing complex data, specifically local file paths. The CapturePhotoFragment serialises the file system path into a Bundle and dispatches it via the setFragmentResult method. This pattern ensures that the EmergencyDetailsReportFr does not need to know the camera module's internal implementation details, but merely acts as a listener for the expected "requestKey".

Listing 6.11: Implementation of the FragmentResult API for cross-fragment data transfer

```
// CapturePhotoFragment: Dispatching the captured file path
private void confirmPhoto() {
        if (lastSavedPath == null) return;
        Bundle res = new Bundle();
        res.putString("bundleKey", lastSavedPath);
        getParentFragmentManager().setFragmentResult("requestKey", res);
        Navigation.findNavController(requireView()).popBackStack();
    }

// EmergencyDetailsReportFragment: Listening for the returned result inside
    function onViewCreated()
    getParentFragmentManager().setFragmentResultListener("requestKey",
```

```
getViewLifecycleOwner(), (requestKey, bundle) -> {
    localPhotoPath = bundle.getString("bundleKey");
    if (localPhotoPath != null) {
        displayLocalPhoto(localPhotoPath);
    }
});
```

Navigational Lifecycle and State Management

This communication mechanism is inherently lifecycle-aware, meaning that results are only processed when the target fragment is in a valid state (e.g., `STARTED` or `RESUMED`). By decoupling the capture logic from the reporting logic, the system effectively handles configuration changes and navigation events without risking memory leaks or the persistence of stale data. This architectural choice establishes a modular foundation, allowing for future expansion, such as integrating image selection from a device gallery, without requiring structural modifications to the existing reporting pipeline.

6.4 UI/UX Evolution and Design Refinement

The UI/UX development of ResQia followed an iterative progression from conceptual design to functional deployment. Figure 6.2 illustrates the initial high-fidelity prototype, which focused on establishing the core spatial hierarchy, navigation patterns and the fundamental user workflows for rapid emergency reporting.

Figure 6.3 represents the final functional implementation, which evolved to support a complex, real-world data environment. Key differences and additions in the production version include:

- **Real-time Geospatial Logic:** The implementation replaces static layouts with a dynamic Geo-dashboard. This integrates real-time Firestore synchronisation, allowing for live incident mapping, location-based pinpoints, and interactive incident details.
- **Administrative and Verification Workflows:** Unlike the prototype, the final application includes a dedicated Admin Control Panel. This module manages user ”Helper” verification and report validation, ensuring the system remains a secure, trusted, and community-driven platform.
- **State Management**: The final interface incorporates advanced state management, as evidenced by features such as incident-reporting progress (empty vs filled states) and administrative status indicators. This ensures that users receive consistent, reliable feedback on their submissions and the status of ongoing emergencies.

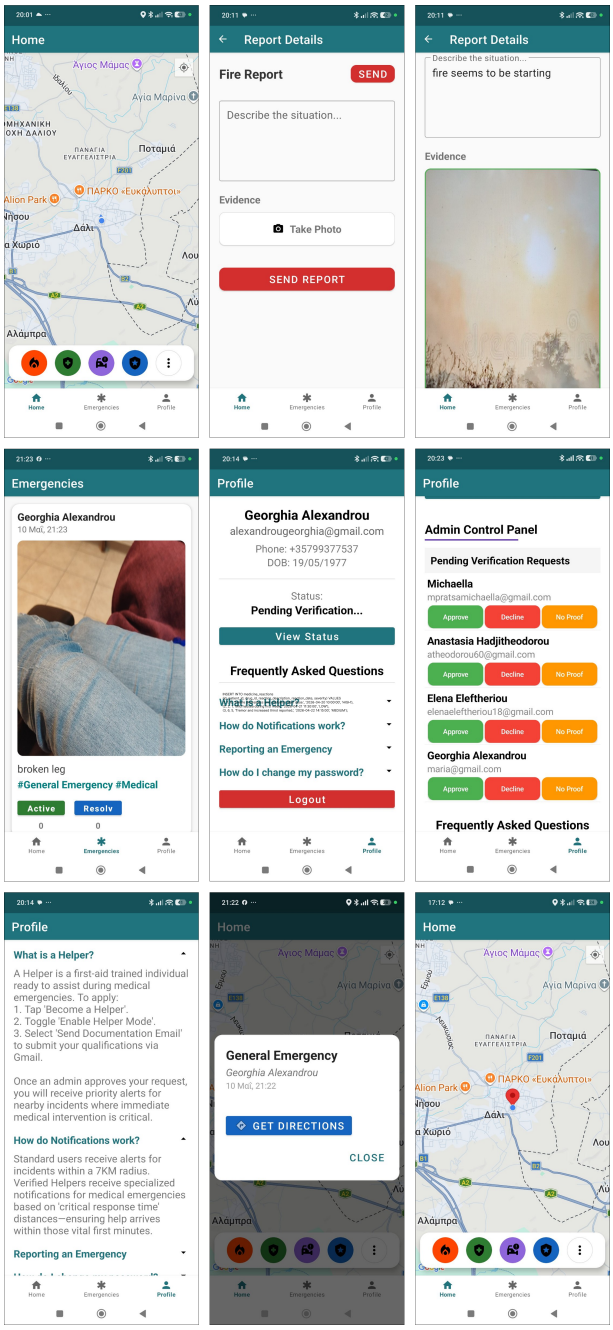

Figure 6.2: High-fidelity visual prototype of the ResQia interface.n

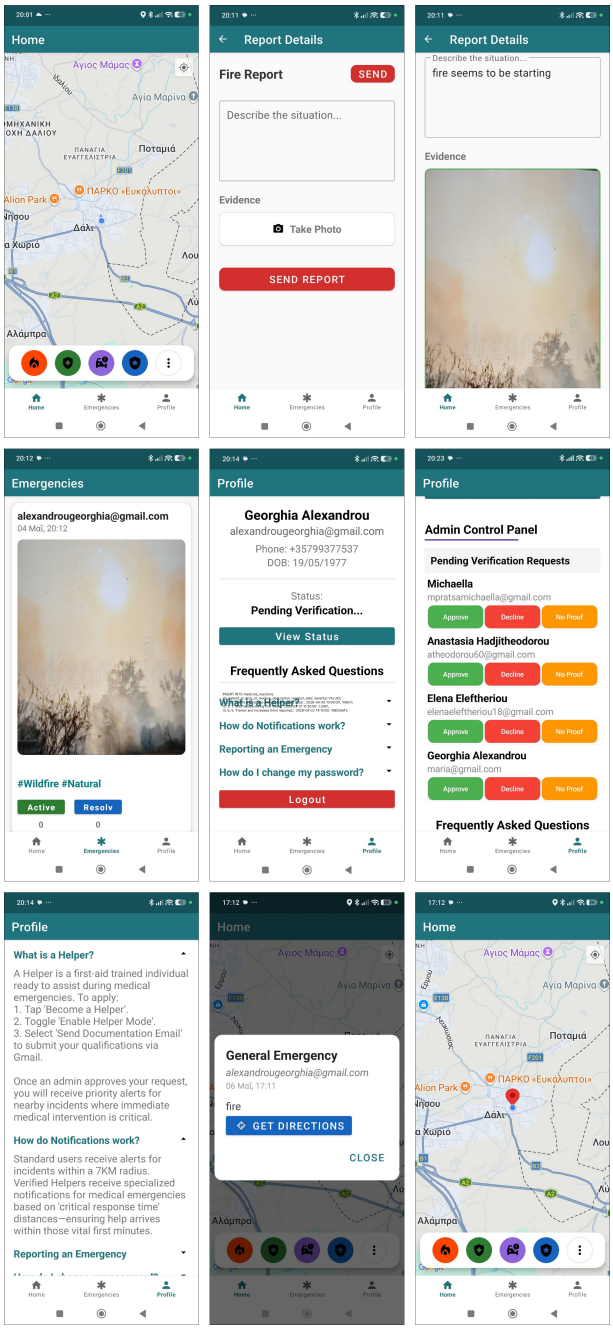

Figure 6.3: Functional implementation of the ResQia mobile application.

By comparing these two stages, we see that the transition from prototype to final product was defined by the integration of live data flows and robust authentication logic, moving beyond visual aesthetics to prioritise system reliability and user safety.

7 System Evaluation

7.1 Evaluation Methodology

ResQia was evaluated through a user-based study involving 20 Android users. Each participant was asked to download and install the application on their personal Android smartphone and use it for four consecutive days. During this period, participants were instructed to open the application at least once per day, explore its available features, monitor nearby emergency reports, and submit a report if they encountered or became aware of an emergency situation.

This approach aimed to allow participants to experience the application in a realistic usage scenario. Instead of evaluating the system immediately after a short demonstration, users were given time to become familiar with the interface, the reporting process, and the notification-based awareness mechanism. This helped gather more meaningful feedback on the system's usefulness, usability, and perceived value.

At the end of the four-day testing period, participants completed a questionnaire about their experience with ResQia. They were encouraged to answer without overthinking to capture their immediate impressions, natural reactions, and genuine intentions regarding the use of the application. This was considered important because emergency applications must be intuitive, fast, and trustworthy without requiring extensive training or explanation.

The evaluation focused on the key aspects important to an emergency reporting and awareness system. These included the perceived usefulness of ResQia, the speed and simplicity of the reporting process, the sufficiency of the information provided for situational awareness, the users' intention to continue using the application, and the overall usability and user experience of the system.

The questionnaire included demographic questions, questions about participants' previous habits of checking external sources such as social media, news, or radio for local incidents, questions about prior emergency experience, Likert-scale evaluation questions, a reporting-speed question, semantic differential user experience questions, and an optional comments section for additional feedback and suggestions.

The full questionnaire used for the evaluation is provided in Appendix A.

7.2 Participants

The participants represented different age groups and user roles. Most participants were in the 18–25 age group, with additional responses from users aged 26–35, 36–45, and 46–60. This allowed the evaluation to include feedback from both younger and older users.

Participants were also asked to identify their primary role when using ResQia. The majority selected the role of *Emergency Reporter*, while some participants selected *Emergency Reporter & Helper*. This distinction was important because ResQia is intended to support both users who report emergencies and users who may be able to assist nearby.

Table 7.1 summarises the participant profile.

Table 7.1: Participant profile

Category	**Group**	**Number**	**Percentage**
Age	18–25	14	66.7%
Age	26–35	3	14.3%
Age	36–45	1	4.8%
Age	46–60	3	14.3%
Role	Emergency Reporter	15	71.4%
Role	Emergency Reporter & Helper	6	28.6%
Emergency experience	Yes	15	71.4%
Emergency experience	No	6	28.6%

The results show that most participants had prior experience with emergencies, as victims, witnesses, or responders. This is useful for evaluation because these participants were more likely to understand the importance of fast, reliable emergency information.

7.3 Previous Emergency Awareness Habits

Before using ResQia, participants were asked how often they checked external sources such as social media, news platforms, or radio to stay informed about local incidents. The results showed that many users already relied on external information sources to stay informed about emergencies or incidents in their area.

Table 7.2 presents the frequency with which participants checked external sources before using ResQia.

Table 7.2: Frequency of checking external sources before using ResQia

Frequency	**Number**	**Percentage**
Multiple times a day	9	42.9%
Once a day	6	28.6%
A few times a week	3	14.3%
Rarely	2	9.5%
Only when something happened nearby	1	4.8%

These results indicate that a significant number of users were already interested in staying informed about local incidents. However, traditional methods such as social media and news platforms may not always provide immediate, location-specific, or structured emergency information. This supports the motivation behind ResQia, which aims to provide a more direct and localised method for emergency awareness.

7.4 Questionnaire Design

The questionnaire was designed to evaluate both the functional effectiveness and the user experience of ResQia; the first part focused on the application's perceived usefulness, using Likert-scale questions. Participants were asked to rate their agreement with statements related to information effectiveness, responder performance, response speed, data sufficiency, behavioural intention, and community trust.

The Likert-scale responses were interpreted using the following scale:

- 1 = Strongly Disagree
- 2 = Disagree
- 3 = Neutral
- 4 = Agree
- 5 = Strongly Agree

The second part of the questionnaire focused on the user experience of the system. Participants evaluated the application using adjective pairs such as *annoying/enjoyable*, *complicated/easy*, *inefficient/efficient*, and *impractical/practical*. These questions were rated on a 7-point scale, where the meaning of a high or low score depended on the direction of the adjective pair.

Finally, participants were allowed to provide open-ended comments and suggestions for future improvements.

7.5 Results of the Functional Evaluation

The functional evaluation aimed to assess whether ResQia achieved its main goals as an emergency awareness and reporting application. Table 7.3 summarises the results of the main Likert-scale questions.

Table 7.3: Functional evaluation results

Evaluation Item	**Valid Responses**	**Mean Score**	**Agree / Strongly Agree**
Information Effectiveness	21	4.43 / 5	100.0%
Responder Performance	8	4.50 / 5	100.0%
Response Velocity	21	4.38 / 5	85.7%
Data Sufficiency	21	4.57 / 5	100.0%
Behavioral Intention	21	4.38 / 5	95.2%
Community Trust	21	4.48 / 5	95.2%

The results were generally very positive. The highest mean score was observed for *Data Sufficiency*, with a mean value of 4.57 out of 5. This suggests that participants considered the information provided by ResQia, such as the incident type, location, and photos, sufficient to understand the situation.

The *Information Effectiveness* item also received a high score, with all participants agreeing or strongly agreeing that ResQia helped them stay more effectively informed about local incidents compared to traditional methods. This result is important because one of the application's main goals is to improve local emergency awareness.

The *Response Velocity* item received a mean score of 4.38 out of 5. Most participants agreed that the application allowed them to identify and respond to emergencies more quickly than checking social media or news sources. However, this item had a slightly lower agreement percentage than some other categories, which may indicate that response speed depends on factors such as notification delivery, user availability, and the clarity of the emergency information.

The *Responder Performance* item was answered only by participants who used or considered the role of *Community Helper*. The results were very positive, with all valid respondents agreeing or strongly agreeing that the application improves their ability to act as a helper during a medical emergency.

The results for *Behavioural Intention* and *Community Trust* were also strong. Most participants stated that they intended to keep ResQia installed and use it regularly, while the majority also indicated that they would recommend the application to friends or family. This suggests that the

application was not only considered useful but also trustworthy and acceptable for personal and community safety.

7.6 Reporting Speed

A key requirement for an emergency reporting application is that the reporting process must be fast and simple. In emergencies, users may be stressed or under time pressure, so the application should allow them to submit a report with minimal effort.

Participants were asked to estimate how long the entire reporting process took, from opening the application to submitting the report. Ten participants answered this question directly through the questionnaire. Table 7.4 shows the results.

Table 7.4: Estimated reporting process duration

Reporting Time	**Number**	**Percentage of Valid Responses**
Less than 30 seconds	3	30.0%
30 seconds to 1 minute	7	70.0%

In addition to the questionnaire responses, the remaining 11 participants were asked informally during an interview about how easy it was to locate and complete the reporting process. These participants reported that the emergency reporting function was easy to find and that submitting a report required at most 5 steps. They also indicated that the process could be completed within approximately 45 seconds.

Overall, both the questionnaire results and the interview feedback suggest that the reporting process in ResQia is fast, simple, and suitable for time-sensitive situations. All questionnaire responses indicated that a report could be submitted in under one minute. At the same time, the interview feedback further supported that users were able to find and complete the reporting process quickly. This is a positive result, as a fast and simple reporting flow increases the likelihood that users will submit emergency reports when needed, especially in stressful real-world scenarios.

7.7 User Experience Evaluation

In addition to functional effectiveness, the evaluation also examined the overall user experience of ResQia. Participants rated the application using a set of semantic differential scales. These scales were used to capture users' feelings about the application regarding usability, clarity, attractiveness, efficiency, innovation, and trust.

Table 7.5 presents the mean and median scores for the user experience adjective pairs.

Table 7.5: User experience evaluation results

UX Pair	**Mean**	**Median**
Annoying / Enjoyable	5.76	6
Not understandable / Understandable	5.62	6
Creative / Dull	2.19	2
Easy to learn / Difficult to learn	1.76	1
Valuable / Inferior	1.43	1
Boring / Exciting	5.19	5
Not interesting / Interesting	5.76	6
Unpredictable / Predictable	5.14	6
Fast / Slow	1.48	1
Inventive / Conventional	2.33	2
Obstructive / Supportive	5.57	6
Good / Bad	1.76	1
Complicated / Easy	5.81	6
Unlikable / Pleasing	5.67	6
Usual / Leading Edge	4.10	4
Unpleasant / Pleasant	5.48	6
Secure / Not Secure	1.95	2
Motivating / Demotivating	2.05	1
Meets Expectations / Does Not Meet Expectations	1.76	1
Inefficient / Efficient	5.76	6
Clear / Confusing	1.52	1
Impractical / Practical	5.90	6
Organized / Cluttered	1.62	1
Attractive / Unattractive	1.67	2
Friendly / Unfriendly	1.48	1
Conservative / Innovative	5.19	6

The user experience results show that participants generally perceived ResQia positively. The application received strong ratings for being practical, easy, efficient, understandable, enjoyable, supportive, and friendly. In particular, the adjective pair *Impractical/Practical* received a mean score of 5.90, indicating that users considered the application highly practical.

The pair *Complicated/Easy* also received a high mean score of 5.81, which suggests that the application was perceived as easy to use. This is an important result because emergency systems must be simple enough to use quickly, even by users who are not technically experienced.

The *Inefficient/Efficient* pair received a mean score of 5.76, indicating that users found the application efficient. This result is consistent with the reporting-speed results, where all valid responses indicated that the reporting process was completed in less than one minute.

The application was also rated positively for clarity and organisation. The low mean scores for *Clear/Confusing* and *Organised/Cluttered* indicate that users generally found the interface clear and well structured. Similarly, the low score for *Friendly/Unfriendly* indicates that the application was perceived as friendly and approachable.

Some adjective pairs require careful interpretation because the positive adjective appears on the left side of the scale rather than the right side. For example, the low mean score for *Valuable/Inferior* indicates that users perceived the application as valuable. In contrast, the low mean score for *Fast/Slow* indicates that the application was perceived as fast. Similarly, the low score for *Secure/Not Secure* suggests that participants generally perceived ResQia as secure.

Overall, the user experience results indicate that participants received ResQia positively and that the application provides a user-friendly experience suitable for emergency use.

7.8 Qualitative Feedback

Participants were also allowed to provide additional comments and suggestions. The comments were generally positive and supported the quantitative results. Several participants described the application as useful, practical, and helpful. One participant specifically highlighted the value of receiving notifications when an emergency occurs nearby, stating that this helps users stay up to date with local incidents.

Another participant commented that the application could provide reassurance to parents, especially when their children may be driving at night. This comment suggests that ResQia may have value not only for direct emergency reporting but also for increasing the sense of safety among family members and community members.

A recurring suggestion was to add an SOS button that would allow users to immediately call emergency services, such as an ambulance or the police. This feature could improve the application by providing a direct connection between emergency reporting and official emergency response services. Another suggestion was the future implementation of automatic car crash detection using phone movement sensors. This could allow the application to detect serious incidents automatically and notify nearby helpers or emergency services.

The main suggestions collected from the qualitative feedback were:

- adding an SOS button for direct emergency calls;
- adding automatic car crash detection using mobile phone sensors;
- continuing to improve the notification system for nearby emergencies;
- maintaining the requirement for photos in emergency reports, as this helps responders better understand the situation.

These comments provide useful guidance for future development and show that users were able to identify practical ways to improve ResQia.

7.9 Discussion

The evaluation results suggest that ResQia successfully addresses its main objectives. Participants generally agreed that the application helps them stay informed about local incidents more effectively than traditional methods such as social media, news platforms, or radio. The high scores for information effectiveness and data sufficiency indicate that the system provides useful, relevant information to support situational awareness.

The results also suggest that ResQia can improve the speed of emergency response. Most participants agreed that the application allows them to identify and respond to emergencies more quickly than they would when relying on external sources. This is particularly important because emergencies often require immediate awareness and action. By using location-based notifications and structured incident reports, ResQia provides users with information that is more direct and relevant to their surroundings.

The positive ratings for behavioural intention and community trust indicate that participants were willing to continue using the application and recommend it to others. This is an important finding because the success of a community-based emergency application depends on user adoption. If users do not trust the application or do not intend to keep it installed, the system cannot achieve its full potential.

The user experience results further support the system's effectiveness. Participants described the application as practical, easy to use, efficient, clear, organised, and friendly. These characteristics are essential for emergency applications, where users may need to act quickly and under pressure. A complicated or confusing interface could reduce the likelihood of successful emergency reporting.

However, the evaluation also identified areas for improvement. The request for an SOS button suggests that users would benefit from a more direct way to contact official emergency services. In addition, automatic crash detection could further improve the system by enabling emergency

detection without requiring manual input from the user. These suggestions could be considered in future versions of ResQia.

7.10 Limitations

Although the evaluation results were positive, there are some limitations to consider. First, the number of participants was relatively small. A larger sample size would provide more reliable and generalizable results. Second, most participants were in the 18–25 age group, suggesting that the evaluation may not fully represent the opinions of older users or those with less experience using mobile applications.

Another limitation is that the evaluation period lasted four days. While this allowed users to interact with the application over multiple days, a longer evaluation period could provide more detailed insights into long-term usage, notification reliability, and continued user engagement.

Finally, the evaluation was based on self-reported questionnaire responses. Although this method is useful for understanding user perception, future evaluations could also include objective performance measurements, such as the exact time required to submit a report, the number of reports submitted, or the response time to emergency notifications.

8 Conclusion

8.1 Thesis Overview and Results

This thesis presented the design, implementation, and evaluation of **ResQia**, a mobile emergency reporting and awareness application. The main goal of the system was to reduce the information gap between the occurrence of a local emergency and the moment when nearby people become aware of it. By allowing users to report incidents, receive location-based notifications, and act as community helpers, ResQia aims to support faster awareness and encourage community-based response.

The implemented system combined emergency reporting, real-time database updates, image upload, location services, push notifications, helper registration, and administrative management. These components worked together to provide users with a structured way to report emergencies and receive relevant information about nearby incidents.

The evaluation results were positive. Participants generally found the application useful, practical, easy to use, and suitable for emergency-related situations. The questionnaire showed strong agreement that ResQia helps users stay informed about local incidents and provides sufficient information for situational awareness. The reporting process was also considered fast, as users could submit a report in under one minute.

Overall, ResQia achieved its main objectives by providing a fast and accessible platform for emergency reporting and local incident awareness. The results suggest that a mobile, location-based system can improve how citizens receive and share emergency information.

8.2 Future Work

Although ResQia successfully implements its main functionality, several improvements could be considered in future versions.

First, the application could include a direct SOS button that allows users to immediately call official emergency services, such as **112**. This would make the application more useful in urgent situations where the user needs immediate professional help.

Another useful extension would be integrating social media. Verified emergency reports could be shared automatically or semi-automatically on platforms such as X. At the same time, the application could also retrieve emergency-related information from public sources, including X posts and local Facebook groups. This could help ResQia collect and distribute emergency

information more widely, although verification would be necessary to avoid false reports.

Future versions could also display nearby **Automated External Defibrillators (AEDs)** on the map. This would be useful for community helpers responding to medical emergencies, as they could identify and collect an AED on their way to the incident if needed.

Other possible improvements include automatic crash detection using phone sensors, better helper verification, multilingual support, iOS or cross-platform development, and limited offline functionality for areas with poor internet connections.

In conclusion, ResQia provides a strong foundation for a community-based emergency awareness system. With further development, it could become a more complete tool for supporting citizens, helpers, and emergency response during critical situations.

BIBLIOGRAPHY

[1] D. G. Costa, J. P. J. Peixoto, T. C. Jesus, P. Portugal, F. Vasques, E. Rangel, and M. Peixoto, "A survey of emergencies management systems in smart cities," *IEEE Access*, vol. 10, pp. 61 843–61 872, 2022.

[2] Y. Zhang, P. Geng, C. Sivaparthipan, and B. A. Muthu, "Big data and artificial intelligence based early risk warning system of fire hazard for smart cities," *Sustainable Energy Technologies and Assessments*, vol. 45, p. 100986, 2021.

[3] A. I. Voda and L. D. Radu, "Artificial intelligence and the future of smart cities," *BRAIN. Broad Research in Artificial Intelligence and Neuroscience*, vol. 9, no. 2, pp. 110–127, 2018.

[4] N. Kapucu, "Disaster and emergency management systems in urban areas," *Cities*, vol. 29, pp. S41–S49, 2012.

[5] Y. Wang and M. S. Kankanhalli, "Tweeting cameras for event detection," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 1231–1241.

[6] J. Han, P. Zhang, and Y. Song, "The construction of emergency management whole process model based on the emergency life-cycle: wenchuan case study," *Technology for Education and Learning*, pp. 235–242, 2012.

[7] E. Ohn-Bar, A. Tawari, S. Martin, and M. M. Trivedi, "On surveillance for safety critical events: In-vehicle video networks for predictive driver assistance systems," *Computer Vision and Image Understanding*, vol. 134, pp. 130–140, 2015.

[8] M. K. Al-Azzam and M. B. Alazzam, "Smart city and smart-health framework, challenges and opportunities," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 2, 2019.

[9] F. Mohammed, A. Idries, N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Uavs for smart cities: Opportunities and challenges," in *2014 international conference on unmanned aircraft systems (ICUAS)*. IEEE, 2014, pp. 267–273.

[10] S. Chatterjee and A. K. Kar, "Smart cities in developing economies: A literature review and policy insights," in *2015 international conference on advances in computing, communications and informatics (ICACCI)*. Ieee, 2015, pp. 2335–2340.

[11] M. Romano, T. Onorati, I. Aedo, and P. Diaz, "Designing mobile applications for emergency response: citizens acting as human sensors," *Sensors*, vol. 16, no. 3, p. 406, 2016.

[12] D. MASANTE, D. BARANTIEV, E. DESTRO, M. MASTRONUNZIO, S. PARIS, C. PROIETTI, V. SALVITTI, M. SANTINI *et al.*, "Multi-hazard early warning system global disaster alert and coordination system (gdacs)," *JRC Publications Repository*, 2025.

[13] F. Mora, "Innovating in the midst of crisis: A case study of ushahidi," *Submitted for publication to SAGE Convergence Journal*, vol. 3, no. 5, pp. 231–245, 2011.

[14] T. Smida, J. Salerno, L. Weiss, C. Martin-Gill, and D. D. Salcido, "Pulsepoint dispatch associated patient characteristics and prehospital outcomes in a mid-sized metropolitan area," *Resuscitation*, vol. 170, pp. 36–43, 2022.

APPENDICES

A Evaluation Questionnaire

This appendix presents the questionnaire used to evaluate the ResQia application. The questionnaire was distributed to participants after they had used the application for four consecutive days. It consisted of participant profile questions, usefulness and impact questions, a reporting-speed question, user experience questions, and an optional comments section.

A.1 Participant Profile and Background

1. **What is your age?**
    - Under 18
    - 18–25
    - 26–35
    - 36–45
    - 46–60
    - 60+
2. **What is your primary role in using ResQia?**
    - Emergency Reporter
    - Emergency Reporter & Helper
3. **Before using ResQia, how often did you check external sources, such as social media, news, or radio, to stay informed about local incidents?**
    - Multiple times a day
    - Once a day
    - A few times a week
    - Only when I heard something happening nearby
    - Rarely
    - Never
4. **Have you ever been involved in an emergency, as a victim, witness, or responder?**
    - Yes
    - No

A.2 Usefulness and Impact

Participants were asked to rate their agreement with the following statements on a scale of Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree, and N/A.

1. **Information Effectiveness:** ResQia helps me stay more informed about local incidents than traditional methods.
2. **Responder Performance:** For Helpers, the app improves my ability to perform as a Community Helper during a medical emergency.
3. **Response Velocity:** The app enables me to identify and respond to emergencies more quickly than checking social media or news.
4. **Data Sufficiency:** The information provided, including incident type, location, and photos, is sufficient for my situational awareness needs.
5. **Behavioural Intention:** I plan to keep ResQia installed and use it regularly to monitor safety in my area.
6. **Community Trust:** I would recommend ResQia to friends or family as a reliable tool for their personal safety.
7. **Speed of Use:** Approximately how long did the entire reporting process take you, from opening the app to submitting the report?
    - Less than 30 seconds
    - 30 seconds to 1 minute
    - 1 minute to 2 minutes
    - More than 2 minutes

A.3 User Experience Questionnaire

This section consisted of pairs of contrasting attributes. Participants were asked to select a value from 1 to 7 based on their immediate impression of the application.

Table A.1: User Experience Questionnaire items

Negative/Positive Attribute	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Opposite Attribute**
annoying								enjoyable
not understandable								understandable
creative								dull
easy to learn								difficult to learn
valuable								inferior
boring								exciting
not interesting								interesting
unpredictable								predictable
fast								slow
inventive								conventional
obstructive								supportive
good								bad
complicated								easy
unlikable								pleasing
usual								leading edge
unpleasant								pleasant
secure								not secure
motivating								demotivating
meets expectations								does not meet expectations
inefficient								efficient
clear								confusing
impractical								practical
organized								cluttered
attractive								unattractive
friendly								unfriendly
conservative								innovative

A.4 Additional Comments

The final question allowed participants to provide optional comments or suggestions regarding ResQia.

Additional comments or suggestions

B Source Code Repository

The final implementation of this thesis project is available in a public GitHub repository. The repository contains the final version of the source code, including the application files and the relevant documentation required for installation and execution.

- **Repository title:** ResQia Thesis Project
- **Repository link:** https://github.com/eirinialxnd/ResQia.git
- **Access:** Public repository
- **Access date:** 29 May 2026