FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Android mobile application for personal safety
Student:	Bich Phuong Phamová
Supervisor:	Ing. Eliška Šestáková
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2020/21

Instructions

Analyse existing mobile applications dealing with the problem of crime prevention in the Czech Republic and worldwide (focus on personal safety applications). Based on this survey, design and implement a prototype of your own personal safety application. Firstly, perform a detailed analysis of the requirements, based on which design the functionality of the application. Furthermore, design a GUI and discuss the architecture of the solution. Implement the prototype as an Android mobile app. Perform user testing of the final implementation and suggest further possible development.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 22, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Android Mobile Application for Personal Safety

Bich Phuong Phamová

Department of Software Engineering
Supervisor: Ing. Eliška Šestáková

May 6, 2020

Acknowledgements

First and foremost, I would like to pay special regards to my supervisor Ing. Eliška Šestáková for her guidance and valuable advisement that led to the completion of this thesis. I also wish to acknowledge the support of my friends and family throughout my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 6, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Phamová, Bich Phuong. *Android Mobile Application for Personal Safety.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstract

This bachelor thesis deals with the design and implementation of a personal safety mobile application for Android operating system. The final prototype is carried out using standard development practices as well as modern technologies such as Google Maps and the Firebase platform. The application provides users with the ability to call immediate help in critical situations. In case a criminal offense is witnessed, it is possible to publish a report to raise awareness. This thesis contains an analysis of existing solutions, requirements specification, design of the application's architecture, implementation, and usability testing. Possibilities of further development are discussed at the end of the thesis.

Keywords mobile application, Android, Kotlin, Firebase, GPS, SOS, personal safety

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací mobilní aplikace osobní bezpečnosti pro operační systém Android. Jsou využity nejen standardní praktiky vývojového procesu nýbrž i moderní technologické služby jako jsou Google Mapy či platforma Firebase. Výsledkem je funkční prototyp. Aplikace uživateli umožňuje přivolat okamžitou pomoc v krizové situaci. Naopak v případě, že se uživatel stane svědkem protiprávního jednání, má schopnost poslat hlášení varujíc tak své okolí. V práci je uvedena analýza současných řešení, specifikace požadavků, návrh architektury aplikace, její implementace a uživatelské testování. V poslední části jsou diskutovány možnosti budoucího vývoje.

Klíčová slova mobilní aplikace, Android, Kotlin, Firebase, GPS, SOS, osobní bezpečnost

Contents

List of Figures

List of Tables

List of Listings

Introduction

"But the chief problem in any community cursed with crime is not the punishment of the criminals, but the preventing of the young from being trained to crime." [1]

—William Edward Burghardt Du Bois

Even though criminality and violence are as old as humanity, this subject is still considered to be one of the most relevant topics in the Czech Republic [2]. It cannot be completely eliminated, but at some decent level, it can be reduced down.

How many times were you worried about your close ones, whether they had safely arrived home, or wondered if they really were where they are supposed to be? Have you ever felt scared when walking the city streets late at night? Have you ever hesitated to call the police when you were a witness of an unpleasant situation that another person was in, or did you walk by thinking someone else will certainly help? These are the questions that led to the idea of the application further described in this thesis.

In life-critical situations, even a second matters. It is essential for rescuers to know the exact location of the incident to save lives as quickly as possible. With the rapid development of technology, the world is open to new possibilities on how to protect itself. Using smart devices could be the easiest way to contact the rescue services, police officers, or a trusted group of people with a simple touch, sending all the information the other side will need to act immediately and precisely. Therefore, all the provided features, such as Global Positioning System (GPS), contact database, camera, the Internet connection, and more, should be taken advantage of at theirs utmost potential.

The goal of this thesis is to implement a crime prevention mobile application, mainly focusing on personal safety. The final result will be a working prototype on devices with Android operating system. One of the primary aims is to perform a detailed analysis of existing solutions. Based on the requirements specification, the final functionalities will be designed. Users will be able to notify their emergency contacts in life-threatening situations, providing them with the exact location. In addition to that, it will be possible to extend the database with crime reports to keep public awareness. Furthermore, the results of usability testing will be presented.

The thesis begins with Chapter 1 that includes common features of personal safety applications and an analysis of existing solutions. In Chapter 2, functional along with non-functional requirements are specified. The chapter also contains the list of use case scenarios that describe the user's behavior and interactions with the application. Following is Chapter 3, where the applied architecture patterns are discussed. What is more, a description of both domain and database structure is given here. Chapter 4 contains a graphical design of individual application screens and functionalities. Its primary focus is on simplicity and straightforwardness.

Chapter 5 along with Chapter 6 presents chosen technologies and development practices used for implementation of the final prototype. Reasons, why platforms such as Firebase and Google Maps are selected to integrate with the application, are also stated in this chapter. Testing methods and results of usability tests can be found in Chapter 7. In addition to that, the police response to the application's real usage is provided. And lastly, possible future enhancements are discussed in Chapter 8. Some of the improvements, such as an introductory tutorial or an offline alarm, were added based on the usability tests.

CHAPTER 1

State of the Art

In general, applications regarding personal safety can be divided into two categories—SOS[1] applications and family locators. Each category can be characterized by its common features. The architecture of the final prototype will be designed based on a comparison of these functionalities. The main components, as well as User Interface (UI) design of existing solutions on the market, are provided in Section 1.2.

1.1 Common Features

There are two main application categories worth mentioning when it comes to the topic of personal safety—SOS applications and family locators.

SOS applications In distress, SOS applications allow contacting bystanders or a close group of people with features such as GPS, Short Message Service (SMS), video, alerts, alarms, and more. Most often, the alarm is represented as an in-app button that, after touching, sends all required information to the emergency contacts in order for them to act immediately and precisely.

Family locators Locators target primarily small private circles, especially families and friends. With locators, it is possible to exchange instant messages, share real-time whereabouts among the members, and define the most frequently visited places.

[1] SOS is an internationally recognized signal or request for help or rescue [3].

1.2 Existing Solutions

The aim of this section is to describe existing mobile applications that have similar functionalities to the one designed in the thesis. The main purpose is to find the most beneficial result and to avoid the obstacles the existing solutions have encountered. The followings are five applications—chosen based on their distinct functionalities and great popularity—for Android Operating System (OS) with the description of their components, advantages, and disadvantages. All of the listed applications have more than 500 thousands of downloads on Google Play [4].

1.2.1 Záchranka

Záchranka [5] (or Ambulance in translation) is a mobile application officially supported by the rescue services of the Czech Republic (see Figure 1.1). It does not deal with crime prevention in the real sense of the word as it is mainly connected with the medical field. However, the analysis was conducted since it is the most popular life-saving mobile application in the country [5]. It is worth mentioning that there are no similar applications like *Záchranka* available for Czech citizens. The application consists of three primary components:

Alarm This element can be found on the main screen designed as an eye-catching red button that on pressing is able to put the user in contact with 155 emergency line or ski patrol. An SMS, with not only the exact location but also the medical issues and battery status, is sent at the same time. If the person is unable to talk or cannot hear, the application also supports communication via SMS. Another essential advantage is that no Internet connection is required. In this case, only the message is sent.

Locator This component shows users' exact GPS location and the nearest automated defibrillator, rescue services, or pharmacy. The application clearly displays points of interest with the option to navigate to their location quickly.

First aid First aid is made interactive in a very intuitive way for various health conditions from burns and bleeding to poisoning and unconsciousness. Paramedics highly use this function in education all over the country.

In the near future, *Záchranka* plans the City Crisis Management Department to send notifications about the life-threatening situations in the area. Users will be able to see them and avoid the potential danger. Support for wearables[2] can also be expected soon.

[2]Wearables are small computers or advanced electronic devices that are worn or carried on the body as part of the clothes or as an accessory [3].

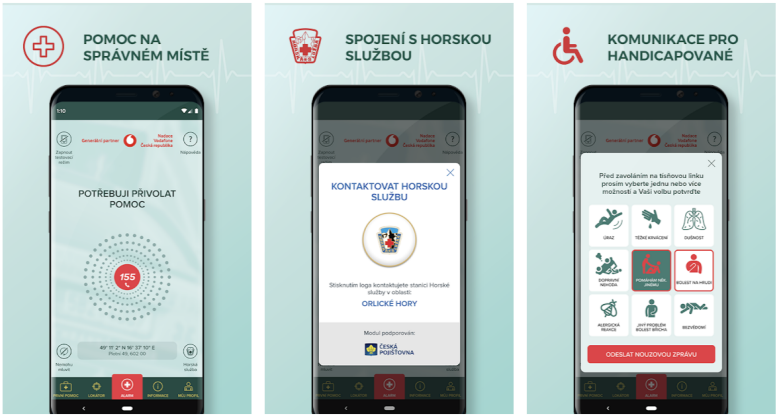

Figure 1.1: *Záchranka* screenshots [5]

1.2.2 Citizen

Citizen [6] is a social network as well as a mobile application first launched in New York City in 2017 (see Figure 1.2). It is till now available only in some parts of the United States of America. The main idea is that everyone should have access to information. Furthermore, it is believed that transparency, along with technology, are the keys to keep the citizens safe as well as aware of the danger around them. *Citizen* provides users with:

Instant alerts As the incidents occur, 911 emergencies notify users in the nearby area. Users can broadcast information in the form of images or user-generated video streams to give incident details from start to finish.

Map of emergencies This component clearly displays the location of crime incidents. They are designed as pinpoints which size depends on the level of severity. This functionality can be beneficial for authorities to determine the situation with the highest priority.

Community Along with proximity, the community is the number one priority in the application. Users can exchange instant messages with each other and comment on posts.

However, according to reviews [6], there are also many issues. The UI layer and navigation across the application are not well designed. Some unnecessary elements make most of the users confused and lead away from the primary goal of keeping awareness. Another problem is the number of notifications displayed in the case of the high post activity. Some can ask a question, whether the popularity of the posts does not lead to seeking dangerous situations on purpose in order to gain prominence.

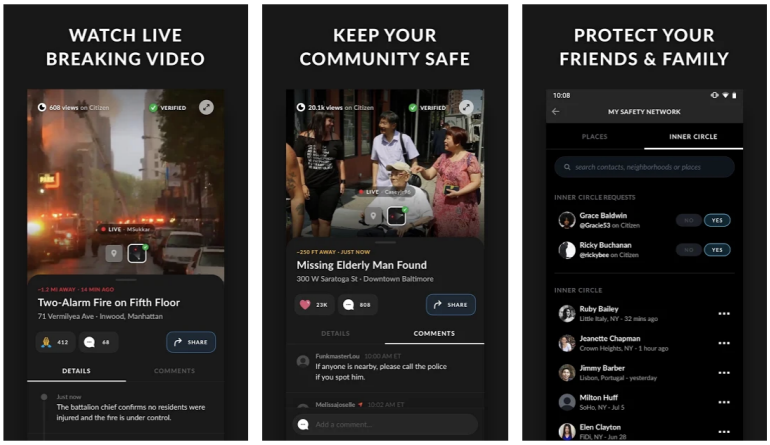

Figure 1.2: *Citizen* screenshots [6]

1.2.3 BSafe

BSafe [7] (see Figure 1.3) origin lies in a horrifying personal story of the company founder as his daughter was a victim of rape. With his daughter, universities, and many well-known organizations[3] by his side, he is leading the community to a safe environment mainly without rape and sexual assaults. Features such as voice activation, live streaming, and automatic audio and video recording were developed based on her experience with the hope of helping other youth worldwide. Major components of the application are:

SOS button This functionality can be activated by push or voice in order to notify emergency contacts. The contacts will not only receive a specific location but also automatically recorded audio and live video.

Timer alarm The timer defines the maximum amount of time required to get to the exact location. If this period exceeds, the SOS button will be triggered.

Security network With *BSafe*, it is possible to set up a private group of people that shares live GPS location in stressed situations. This supports the company's motto "Never walk alone".

Fake calls Users can receive fake phone calls in order to leave the unpleasant companion or uncomfortable situations.

In-app sirene This functionality can be enabled with one touch and surely keeps away an impending danger.

[3] the National Sexual Violence Resource Center, Pennsylvania Coalition Against Rape, Infiniteshe and Beyond Harassment

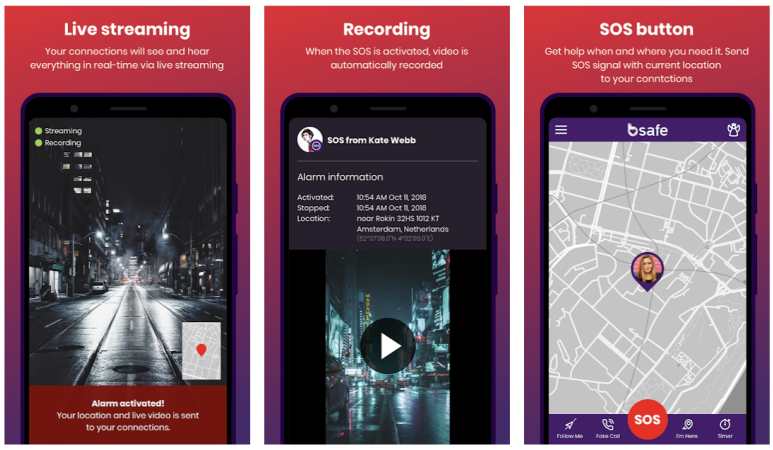

Figure 1.3: *BSafe* screenshots [7]

1.2.4 Life360

This very popular and well-designed application (see Figure 1.4) with more than 50 million downloads [8] has received multiple awards as the winner of both Google Android Developer Challenge[4], and Facebook fbFund[5].

Besides the typical characteristic of the family locator (as mentioned in Section 1.1) such as the ability to exchange instant messages and share real-time whereabouts among the members, *Life360* also provides users with:

Definition of frequently visited places The chosen circle of people is notified whenever the user is arriving or leaving the known area—most often it is home, school, or grocery store. This keeps the members in constant synchronization and assures the user's safety.

Driver support This component shows the driving speed of the member as well as reports crash detections that alert emergency contacts and sends an ambulance immediately. It can also be used in case of trouble on the road, such as a flat tire or engine problem, in which case the Roadside Assistance will help.

Crime reports The report's purpose is to raise awareness in the user's private circle about the nearby crime offenses.

[4]Google Android Developer Challenge is an annual Google contest where best application idea on Android OS with focus on specific technology is picked [9].

[5]Facebook fbFund provides micro-investments for companies developing websites and applications related to Facebook [10].

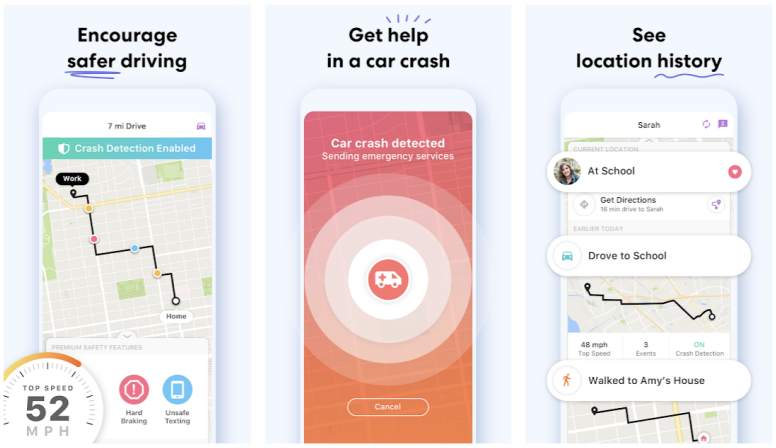

Figure 1.4: *Life360* screenshots [8]

Despite the application's popularity, there are some disadvantages worth mentioning. Crime reports, crash detection, emergency response, and roadside assistant are supported in the United States of America only. Users are required to keep the Internet connection at all times, which can lead to rapid battery loss of the device. Furthermore, despite the application being closed, a lot of unnecessary notifications were received. Applications such as *Life360* can give children and parents a sense of security, but they also raise questions about privacy and children's autonomy.

1.2.5 Noonlight

Noonlight [11] is a company that is not only focused on the same-titled mobile application but all Internet of Things (IoT)[6] that have the ability to secure end-users for instance wearables, smoke detectors, security cameras, and even Artificial Intelligence (AI) assistants. The company is working towards a world where homes, cars, and devices all work together to keep the community safe.

This simple, efficient, and well-designed application (see Figure 1.5) is primarily focused on one functionality—to alarm the authorities in case of an emergency. Similarly to all the existing solutions mentioned above, the alarm system is designed as an in-app button on the main screen. The alarm cannot only be initiated manually but also via an automatic trigger, such as a car crash detected through smartphone sensors. Once an alarm is created, the user will get into immediate contact with certified operators located in monitoring centers. The alarm can be canceled with verification code at any time.

[6]Internet of Things can be thought of as devices that are embedded in everyday objects and are connected to the Internet [3].

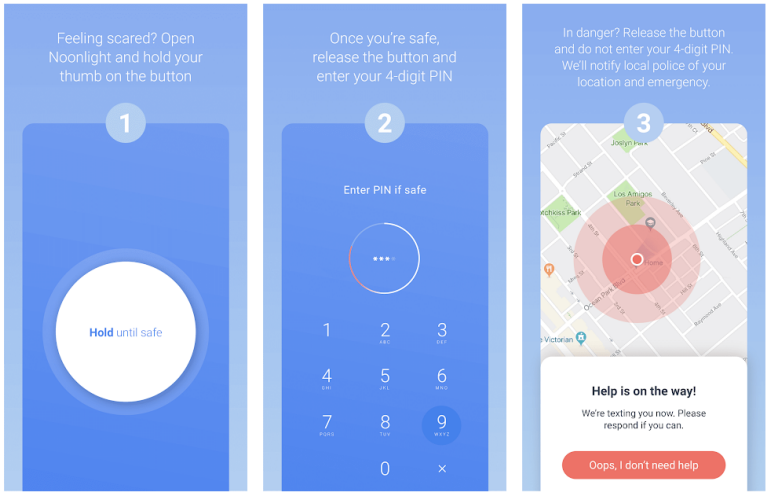

Figure 1.5: *Noonlight* screenshots [11]

1.3 Summary

There are two categories in which most of the personal safety applications can be divided into—SOS applications and family locators. SOS applications share a person's location and more via the alarm button in order to notify emergency contacts or police officers. On the other hand, family locators target smaller groups of people, providing them with instant messages exchange and the ability to define frequently visited places. The five applications from Google Play sharing the same purpose were described. Below is the list of highlights for each application.

- *Záchranka*—the life-saving application in the Czech Republic.
- *Citizen*—the application mainly focused on crime reports.
- *BSafe*—a classic SOS application.
- *Life360*—the most popular family locator on Google Play.
- *Noonlight*—a simply designed SOS application.

In conclusion, all functionalities and main advantages that were mentioned in the existing solutions can be found respectively in Table 1.1 and Table 1.2. Based on testing on multiple devices with Android OS, out of all applications described in this chapter, only *Záchranka* is fully supported in the Czech Republic and works correctly. Nevertheless, this application lacks crime reports, deeper connection with emergency contacts, and live GPS tracking.

Functionality	*Záchranka*	*Citizen*	*BSafe*	*Life360*	*Noonlight*
alarm	✓		✓		✓
contact management	✓	✓	✓	✓	
live GPS tracking		✓	✓	✓	
crime reports		✓		✓	
sirene			✓		
driver support				✓	✓
favourite places				✓	

Table 1.1: Functionalities coverage by existing solutions

Advantage	*Záchranka*	*Citizen*	*BSafe*	*Life360*	*Noonlight*
offline alarm			✓		
multiple languages	✓				
free of charge	✓				
connection with IoTs					✓

Table 1.2: Advantages coverage by existing solutions

Here are other main disadvantages that should be pointed out from the overall analysis:

- Paid versions are limiting important functionalities.
- A great number of unnecessary notifications are sent.
- Constant Internet connection is causing a rapid loss of battery.
- Disability to trigger the alarm when the application is turned off.

Based on the comparison, the requirements and core features for the final application can be specified. Functionalities such as the alarm, contact management, live GPS tracking, and crime reporting are included in the implementation of the resulted prototype. In addition to that, it will be possible to find the nearest police stations from the user's current location.

CHAPTER 2

Requirements Specifications

This chapter specifies all functional and non-functional requirements essential for the start of the development process. They are derived from the analysis and comparison of features that are implemented in existing solutions.

2.1 Functional Requirements

Functional requirements define features or functions a software must perform. They describe system behavior. [12] These requirements regard the core functionalities of the application, such as the alarm, live GPS-tracking, crime reporting, and contact management.

F1 Register new account To ensure data security, a person's phone number will be required for signup. After the verification of both phone number and confirmation code, the registration will be successful. From this moment on, there will be no need for logging in after each start of the application.

F2 Alarm With the application, it will be possible to activate an alarm in case of a dangerous situation. On a single click of the in-app button, emergency contacts will be notified immediately.

F3 Notify contacts On account of the alarm activation, emergency contacts will be notified either with an SMS message or an in-app notification in case they also have the application installed.

F4 Track user location In danger, the exact location will be shared with the emergency contacts in real-time. If the contacts have the application also installed, the position will be visible on a map.

F5 Show directions With the application, it will be possible to select a place on the map and show directions and the fastest route from the user's current location. This functionality will be especially beneficial in situations where the location of a person calling for help is visible on the map.

F6 Manage emergency contacts Users will be able to have as many connections as they wish to. However, the created relation has to be acknowledged by both sides. Moreover, contacts can be edited and even deleted.

F7 Create crime reports Users will be allowed to send a crime report via the application. The report will consist of a description, crime category, level of severity, and an optional image.

F8 Show selected report on the map On a report's click, the user will be redirected to the map where the selected report is in focus. In addition to that, its information, such as the report's title and the street address, will be visible.

F9 Display reports on the map The application will be able to display all the reports on the map. Reported incidents will be represented as pinpoints which size depends on the level of severity. The most pressing will be the most visible.

F10 Display reports in a list The application will be able to show all the reports in a list. Each item will contain the time passed, the exact location of the incident, and the distance from the user's current position.

F11 Sort reports It will be possible to sort reports by the time of posting, the level of severity, and by the distance from the closest to the furthest from the current location.

F12 Show the nearest police stations Users will be able to locate the nearest police stations according to their current position. Stations will be visible on the map with an option to quickly navigate to their location.

F13 Change application language The application will be localized in two languages for the user to choose from—English and Czech. However, the application should be designed in a way any other language could be easily added later.

F14 Introduction At the beginning of the application, a brief description of the core functionalities will be provided to put users into context. After signup, the introduction will no longer be displayed.

2.2 Non-functional Requirements

Non-functional requirements define the quality attribute of a software system [12].

N1 Synchronization with the server The application will be able to retrieve actual data from the server in real-time. It is especially important for GPS live-tracking functionality.

N2 Intuitive and straightforward design As the primary purpose of this application is to ensure the user's security, the design ought to be intuitive and straightforward without redundant functionalities and misleading elements.

N3 Security Each registration has to pass the verification process in order to use the application.

N4 Wide coverage To reach as many people as possible, the application will cover a wide range of Android devices, including various sizes and different versions of the operating system.

N5 Usability The UI will be designed in a way both children and the elderly have no difficulties with operating the application.

2.3 Use Case Scenarios

Based on functional requirements, use case scenarios that describe the user's behavior and interactions with the application are specified in this section. The functional coverage by use cases is detailed in Table 2.1. The UI design of individual components and screens is presented further in Chapter 4. The followings are basic use case scenarios in the application:

UC1 User acquaints with the application

1. User installs and launches the application.
2. The application shows a welcome page.
3. The application briefly introduces the core functionalities such as the alarm, crime reporting, and the nearest police stations search.

UC2 Change application language

1. User navigates to the device's settings.
2. User selects the language in which the application will be displayed.
3. By default, if the language is not supported, the application will be displayed in English.

UC3 Signup

1. The application displays the signup screen.
2. User enters his/her phone number in full format.
3. User confirms the number by clicking the "Send" button.
4. If the verification is successful, SMS with a 6-digit code is sent.
5. User enters the received code.
6. The system automatically evaluates the code after the 6th digit.
7. If the code is correct, access to the core of the application is given.

Alternative flow: SMS auto-detection

5A The application automatically detects the received SMS.

6A Access to the core of the application is given.

UC4 Create crime report

1. User navigates to the top-level destination with the label "News".
2. User clicks on the main action button to create a new crime report.
3. The application shows a report's form.
4. User fills the description in, selects the crime category, and a priority value of the report.
5. User can enclose a photo.
6. User publishes the report on "Send" button click.
7. The new report is displayed on the main screen, and on selection, it is visible on the map.

UC5 Display the newest/the most pressing/the nearest reports

1. User navigates to the top-level destination with the label "News".
2. User selects the attribute by which the items will be sorted.
3. Reports are arranged from the newest to the oldest, from the most pressing to the least important or from the nearest to the furthest from the current location according to the selected attribute.

UC6 Call for help

1. User navigates to the top-level destination with the label "Alarm".
2. User clicks on the SOS button.
3. The application shares the user's location with his/her emergency contacts via SMS or in-app notification.

UC7 Help the person in an emergency

1. User receives the SMS message from the person in danger.
2. The message contains the exact location of the caller along with the battery status of his/her device.

Extended flow: The contact is registered in the application

3 User receives an in-app notification regarding the person in danger.

4 On the notification click, the application is opened.

5 User navigates to the top-level destination with the label "Map".

6 The location of the person in danger is tracked and visible in real-time on the map.

7 On location's click, the user is provided with directions and the fastest route to this position.

8 The nearest police stations are displayed on the tab "The Police".

UC8 Add emergency contact

1. User navigates to the top-level destination with the label "Contact".
2. User clicks on the primary action to add a new emergency contact.
3. The application shows a contact's form.
4. User fills the name and the phone number of the contact.
5. User can enclose a photo.
6. User saves the contact on "Create" button click.
7. The new contact is visible in the list of emergency contacts.
8. The user's phone number is automatically added to the contact's list of wards.

UC9 Delete emergency contact

1. User navigates to the top-level destination with the label "Contact".
2. User selects a contact item that he/she wants to remove from the list of emergency contacts.
3. The application shows a contact's form.
4. User deletes the contact on "Delete" button click.
5. The contact is removed from the list of emergency contacts, and the user's location is no more shared with this person.
6. The user's phone number is automatically removed from the contact's list of wards.

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
F1			✓						
F2						✓			
F3						✓	✓		
F4						✓	✓		
F5							✓		
F6								✓	✓
F7				✓					
F8				✓					
F9				✓					
F10				✓	✓				
F11					✓				
F12							✓		
F13		✓							
F14	✓								

Table 2.1: Functionality coverage by use cases

CHAPTER 3

Design

At the beginning of this chapter, Android platform is introduced along with Kotlin language as it gained Google's support in the last years. The following sections give further details on the project structure and architecture. Lastly, the domain and database model is described.

3.1 Android Platform

Android is a Linux-based OS mostly developed by Google. It powers not only mobile devices but also tablets, watches, and even some types of car equipment. Android has been the best-selling OS for smartphones worldwide. [13] With a wide range of price rates along with various designs and functionalities manufactured by many companies, users can choose what suits them the best. This flexibility has allowed Android to grow incredibly quickly as a platform.

3.1.1 Versions

Android has seen numerous updates that have incrementally improved the operating system. Each major release is named in alphabetical order after a dessert or a sugary treat. However, this practice is ending with the latest stable version Android 10, which is simply named as Android Q. As one of the non-functional requirements is extensive coverage, the application will support Android devices with minimal Application Programming Interface (API) version of 19 (namely KitKat), which corresponds to 96.2 percent of all active devices. The supported versions are highlighted in Table 3.1 and Figure 3.1. The minimal version was also chosen due to its text messages management.

Codename	API	Distribution
Gingerbread	10	0.3%
Ice Cream Sandwich	15	0.3%
Jelly Bean	16–18	3.2%
KitKat	19	6.9%
Lollipop	21–22	14.5%
Marshmallow	23	16.9%
Nougat	24–25	19.2%
Oreo	26–27	28.3%
Pie	28	10.4%

Table 3.1: A relative number of devices running a given Android version [14]

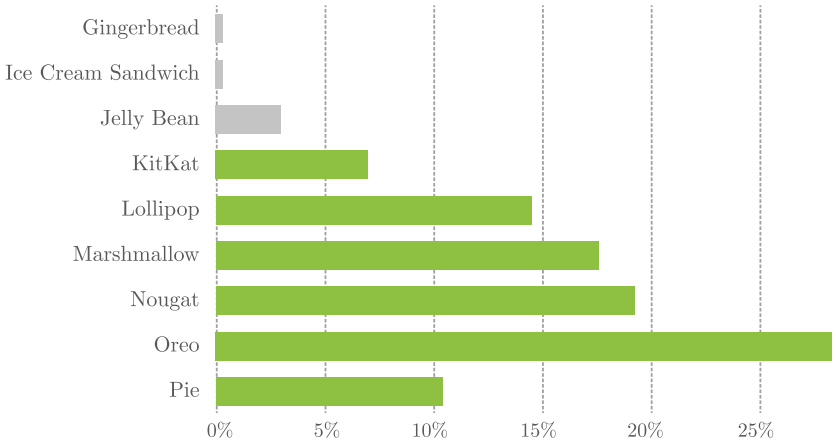

Figure 3.1: Android version distribution [14]

3.1.2 Kotlin

Kotlin is an open-source language developed by JetBrains, which is also a company behind the Android Studio Integrated Development Environment (IDE). It was never fully acknowledged until Google introduced Kotlin as an official language for Android development along with Java and C++ at its annual developer conference in 2017 [15]. Two years later, Google announced that the development would go Kotlin-first, and with that, it became the recommended choice for Android development. Nowadays, over 50 percent of professional Android developers write their code in Kotlin [15].

Kotlin provides safe and concise yet expressive programming in comparison to the verbosity of Java language. However, Kotlin is designed to interoperate fully with Java. Both Java and Kotlin classes can work side by side in a project and compile without any issues.

Followings are reasons why Kotlin was selected as an implementation language of the application:

- Nullpointer exception safety—Kotlin does not allow null assignment to variables by default.
- Extension functions that represent the ability to extend new functionality without having to create a class.
- Coroutines (more in Section 6.4) which provides simple multi-threads managing.
- Functional programming such as lambdas and lazy operations are supported.
- Manual definition of constructors, getter alongside with setter methods and further class functions is no more required.
- Explicit type specification of a variable while declaring is not necessary.
- It drastically reduces the amount of boilerplate code.

3.1.3 Gradle Kotlin DSL

Kotlin can also be integrated into the Gradle build system for Android. It replaces the traditional Groovy language and allows detachment of dependant libraries, plugins, modules, and versions into a separate file. This isolation makes the dependencies easier to maintain and refactor in the future as changes are updated from one place.

3.1.4 Base Components and Lifecycle

In this section, two crucial building blocks of every Android application are introduced. The `Activities` and `Fragments` are fundamental parts of an application model (see the visual representation in Figure 3.2). They are essentially what the user sees on a screen and can interact with. Both of these components are lifecycle aware, which means they go through several specific stages during configuration changes.

Activities `Activity` can be mostly thought of as a full-screen or a floating window where all the UI is laid out. Generally, one `Activity` represents a container for a group of screens that shares the same context.

Fragments `Fragments` are very similar to `Activities`. However, they provide an additional concept of modularity. They are used to create a responsive layout as well as scale the application between small and large devices. [16] Each `Fragment` is included in the `Activity`. Furthermore, both of their lifecycles are closely attached.

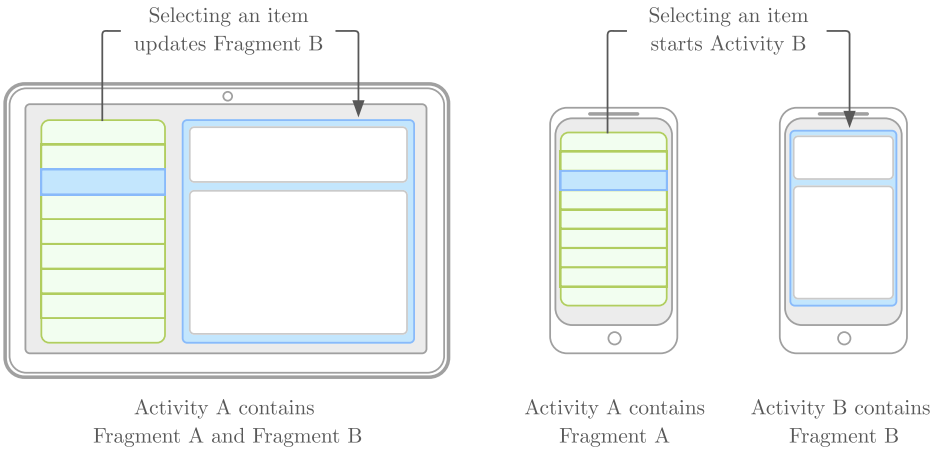

Figure 3.2: Example of Activities and Fragments on different devices [16]

Lifecycle

The order in which the `Activities` are displayed to the user is implemented as a stack with a running `Activity` on top. When a user navigates to a different screen, previous `Activity` will transfer below it and will be waiting to be resumed later on. And when the application is closed or crashes, then the `Activity` is destroyed and cleared from the memory. These stages are all part of the lifecycle that is visualized in the diagram in Figure 3.3.

3.2 Multimodularisation

Multimodularisation refers to the process of isolating logical components or functionality of a project into separate modules that are distinct and independent [18]. Thus, modules are parts of an application that hold discrete responsibilities but can interact with each other. Although it requires more effort, the reasons why the modularization was chosen are:

- Build time is faster.
- Dependency control is less complex.
- Codebase is cleaner.
- Future refactoring is simpler.
- There is a possibility for dynamic features[7].

[7]Dynamic feature can be installed on demand. Therefore, making the core of the application much smaller in size.

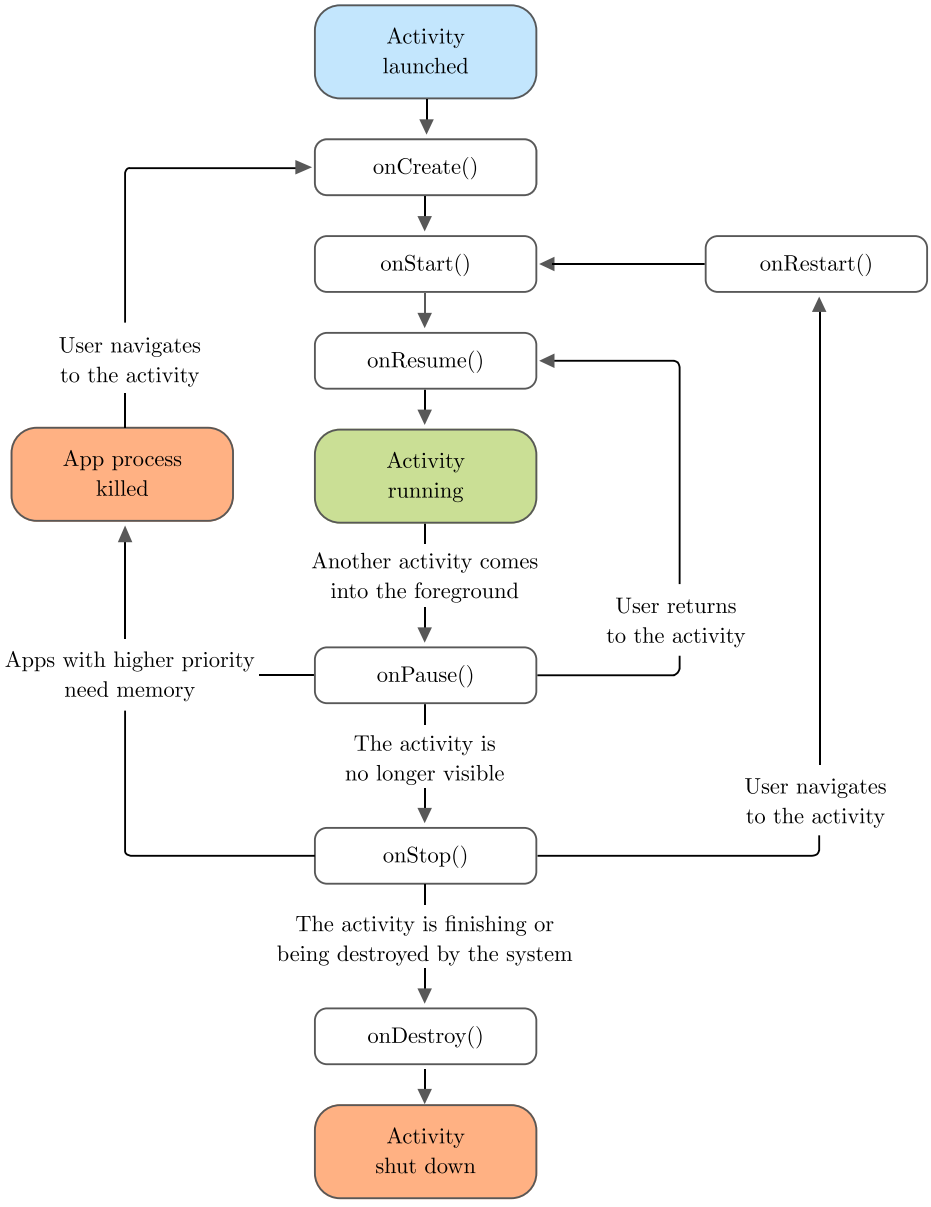

Figure 3.3: Activity lifecycle diagram [17]

There are a few ways to split the project structure—by feature, by layer, or both. By "layer", it is meant: database, network client, repository, or a user interface. Splitting by feature provides encapsulation. Components that work together are all in one place. That is why splitting by feature was chosen for this application. The main functionalities will be presented as individual modules. Therefore, the main project modules are authentication, alarm, reports, map, and a contact management module.

3.3 Design Pattern

There are many architectural patterns used in Android development. Some of the well-known are Model View Controller (MVC), Model View Presenter (MVP), and Model View ViewModel (MVVM). All of them are striving to follow a common principle—the Separation of Concerns[8]. Their goal is to detach the UI from the database model, and business logic as much as possible in order to create loose decoupled classes, robust projects that are easier to maintain and test. [19]

Besides the author's knowledge and MVVM being the recommended choice (MVVM flow is illustrated in Figure 3.4), there are two other significant reasons why it was determined this pattern would suit the application architecture:

- The pattern supports two-way data binding between the view properties and the model, which leads to an immediate update of UI when data changes (more about data binding in Section 6.2).
- `ViewModel` (read more in Section 6.1.2) is not aware of the view—it does not hold the reference to it. Instead, it only broadcasts the changes, and individual subscribers will handle the data separately. This respects the lifecycle of individual observers.

3.4 Domain Description

The underlying concept of the application is to enhance the awareness about dangerous situations around as well as to provide the ability to call for help in case of an emergency. The conceptual model of a domain using Unified Modeling Language (UML) classes can be seen in Figure 3.5.

Upon application installation, every user has to go through a verification process. The access is provided only upon successful validation of the person's phone number. The core of the application consists of the map and the news component that displays the crime reports (see their UI design in Section 4.4).

[8]Separation of Concerns is a principle which idea is to avoid co-locating different responsibilities within the code [19].

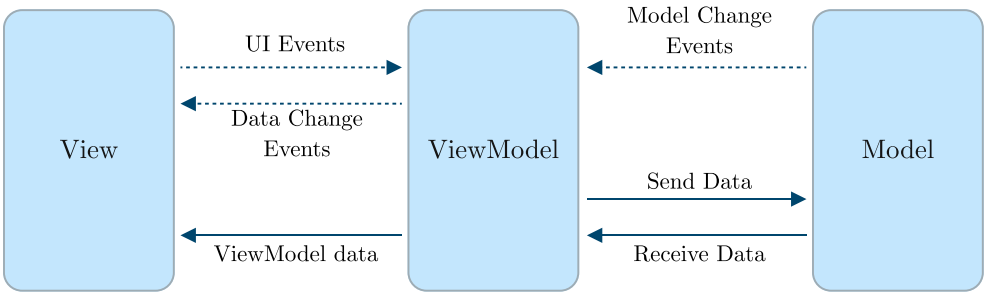

Figure 3.4: MVVM communication model [20]

Users can alarm their emergency contacts via application notifications or SMS messages on a single click of a button. The person's phone number, battery status, and current location are an essential part of the sent message. On the contrary, when witnessing a crime scene, a report can be sent to keep the surrounding informed. The report can be found on the map as well as in the news. The necessary attributes of the report are title, crime category, and priority specification.

3.5 Database Structure

The application data are stored in a local as well as a remote database. Local being the device's storage and remote refers to the databases provided by Firebase (see more in Section 5.1.2). This approach is chosen to make already loaded reports available even when there is no Internet connection. The database structure for the current scope of the application is rather simple as the only stored entities are `Contacts`, `Wards` and `Reports`.

Contacts and Wards

`Contacts` and `Wards` share common attributes such as a name, phone number, and an optional profile picture. However, from the user's perspective, a `Contact` can be thought of as an emergency contact that will be notified in case of danger, whereas `Ward` is a person on the opposite side that notifies and shares the location with the user when needed.

Reports

`Report` entity represents the crime report that is shared among the users. Its fundamental properties are description, exact location, category specification, priority level, and lastly, the time of posting. An optional image resource can be enclosed.

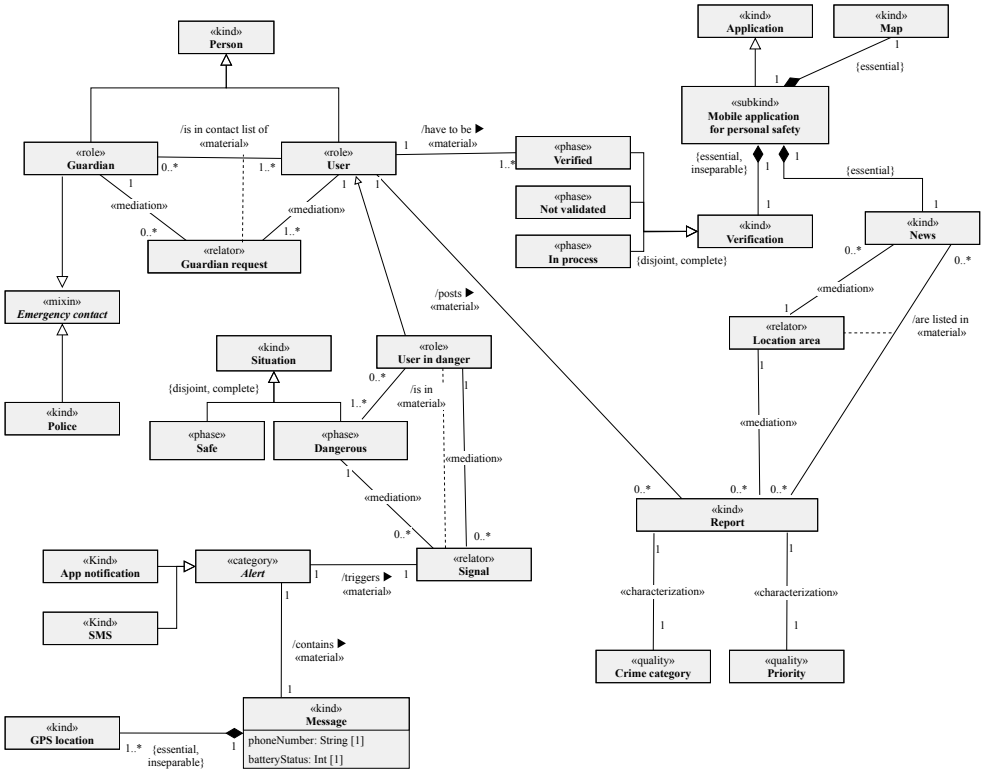

Figure 3.5: Conceptual model of the application domain

Crime Category

To shorten the time user has to spend with creating a report, a set of predefined crime categories is provided. The initial list of category keywords includes weapon, abuse, theft, terrorism, and fire.

Priority

The priority attribute represents the report's level of severity ranged from low to high. Furthermore, this value serves as a filter option or to visually distinguish the reports on the map (see the features UI design in Section 4.4).

CHAPTER 4

UI Design

As the application does not target a specific group of age, the design focuses on welcoming anyone, including children and the elderly. The UI ought to be simple and straightforward, but most importantly, understandable. With this concept in mind, a lot of minimalistic informative illustrations are provided all over the application, and unnecessary elements that would only distract the user were avoided. In this chapter, the description of the chosen UI components and individual screens is presented.

4.1 Material Design

Material is a UI design standard in the Android development that provide guidelines and tools to support the best practices [21]. It is Google's visual language, inspired by the physical world and its textures. It focuses on how individual components reflect light and cast shadows.

The application UI follows the Material system as it is consistent, has Google's support, and millions of users are already familiar with its principles. Material Design specific Components that were used and customized (see Figure 4.1) for the overall application theme are:

Buttons Buttons allow users to take actions and make choices with a single tap. The design depends on the emphasis of the button's action. The more important the action is, the more the button is distinct.

Floating Action Buttons (FAB) A FAB is often presented as a rounded button with an icon in its center that is connected with a primary action on the screen. It has the highest elevation as it should be the most visible element. [21] FAB should perform a constructive action. Hence it found its place in crime report creations and contact items editing.

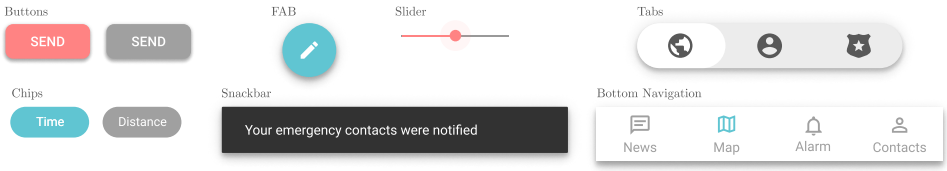

Figure 4.1: Material Design Components that were used and customized

Chips The chips' primary purpose is to provide a collection of choices, content filters, or action triggers [21]. In the application, these elements are used for reports sorting. A sorting attribute can be the time of publishing, priority, or distance from the current location.

Tabs Tabs present the distinct content that can be switchable on the screen [21]. This component helps to differentiate the types of markers on the map and to also navigate between the list of emergency contacts and wards.

Snackbars Snackbars are brief temporary messages at the bottom of the screen that informs the user about application processes. The message can be, for example, *"Your crime report was successfully published."*, or *"Your location is now being tracked."*.

Bottom Navigation This component allows navigation across the application's primary destinations [21]. It serves as a connection hub between the individual features making each one of them accessible from anywhere in the application.

Sliders Sliders allow users to select a value from a range of values along a bar. The selection is adjusted by dragging the finger along the track. In the application, it is used to determine the report's severity from low to high.

4.2 Google Maps Graphics

Besides the access to Google Maps servers, data downloading, map display, and response to map gestures, the Google Maps API also provides a set of graphical objects [22]. Followings are graphics that were used and customized in the application:

Markers Markers represent the crime reports, wards in danger, and the nearest police stations on the map. The markers were customized in a way not only info windows but also the map toolbar appear on an event click.

Info Windows Each marker is tied with an info window that contains text and an optional image. The view is displayed as a pop-up window overlaying the map. [22] Additional information on the marker such as a crime report's title, name of a ward or police facility, phone number, or the location address can be found in the info window's content.

Clusters Marker clustering is a process used when a large number of data points on the map are required on different zoom levels. It is making the map easier to read by displaying only the amount of markers that it is covering.

Shapes Google Maps offers a few simple ways to add basic shapes to the map. The location pin of the ward in danger was designed with animating circle objects replacing the marker.

With the Maps API, the way users can interact with the map was also defined. Some of the built-in UI components were determined to appear on the map, such as the map toolbar buttons. These buttons provide access to a map view, or directions request in the Google Maps native mobile app.

4.3 Animations

To illustrate what is happening on the screen, visual actions like animations can be used. They are particularly valuable when reflecting UI changes. [23] To indicate whether the alarm button has been pushed, and thus the location has been tracked, the pulsing animation was applied. The animation looks like as if the button was broadcasting signals which complements the actual act of calling for help (see the button's design in Figure 4.6). For consistency purposes, this visual effect can also be seen on the map.

4.4 Features UI

Based on the analysis of existing solutions, it was determined that the application would primarily include four features—crime reporting, map display, alarm in emergencies, and contacts management. Each of these functions was designed into a separate group of screens representing the top-level navigation destination.

4.4.1 Onboarding

The purpose of onboarding screens, which can be seen in Figure 4.2, is to describe the application's features briefly. Users can skip the whole introduction or swipe until the end, where they can further navigate to the signup. Once the user approaches the signup process, the onboarding is not further displayed in application future launches.

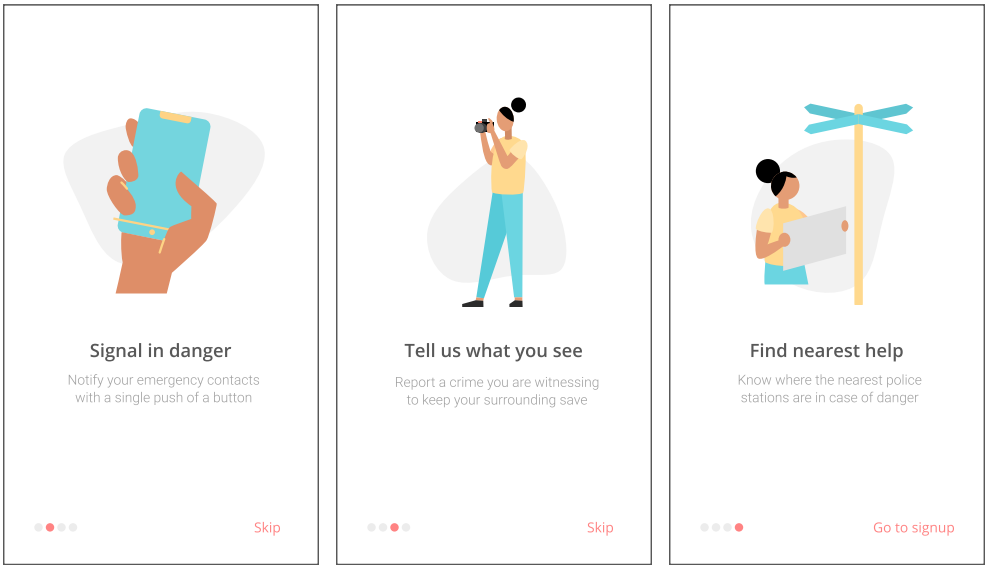

Figure 4.2: Onboarding UI design

4.4.2 Signup

Signup consists of only two screens that can be seen in Figure 4.3. The first screen contains a phone number field. If the device is not instantly verified by the Firebase Authentication (read more in Section 5.1.1), the SMS message with a 6-digit code is generated and sent. The code input is automatically validated after the 6th digit, and if correct, the access is given to the central core of the application. In future application launches, as the phone number is saved in the database, the user is instantly redirected to the main application features.

4.4.3 Crime Reporting

The main screen of this feature (see Figure 4.4) contains a list of all published crime reports and can be found behind the "News" label on the bottom navigation bar. The individual report's attributes are well-arranged for the purpose of readability. The priority of the report is represented by the number of red dots—three are for the most pressing situations. The sorting attributes are presented as chips that are hidable on scrolling. On a click event, the user will be redirected to the map where the selected report is in focus.

There is a FAB located in the bottom right corner, which on tap displays the form to create a new crime report. All attributes are designed in a way so that quick interaction is possible. Crime categories are presented as single selection buttons, and the priority value is determined by a slider.

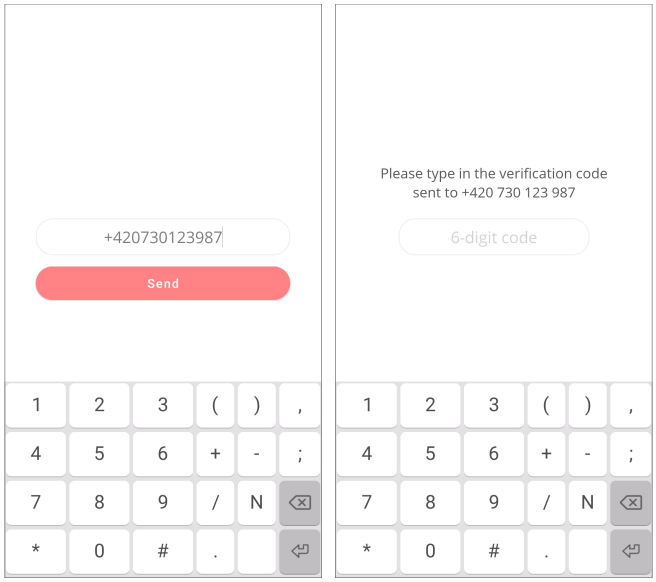

Figure 4.3: Signup UI design

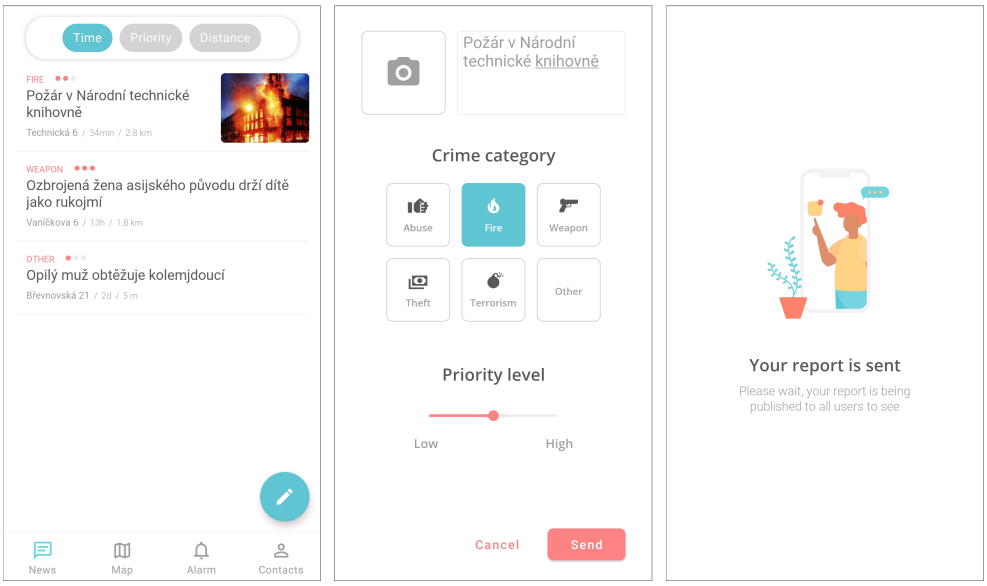

Figure 4.4: Crime reports UI design

4.4.4 Map

For the group of map screens (see Figure 4.5), the Google Maps API was used. Its native components were customized to suit the application context. There are tabs located on the top of the layout for navigation purposes. Each of the tabs represents a different type of map—to be exact different types of markers. The map levels are:

Reports map This kind of map presents all the published crime reports as pins placed on the reported location. Further information and direction to this place can be seen on a pin tap. If a particular zoom level causes the pins to accumulate in one place, making them overlap each other, they are replaced by a cluster (as described in Section 4.2) for transparency reasons.

Wards map In this level, it is possible to track the location of the ward in danger in real-time. The pin marker is designed as a pulsing circle that updates its position in a matter of seconds with the ward's name and the phone number as its description. Once more, all Google Maps native functions—such as directions—are available on pin's interaction.

Police map This map option displays all the police stations within a radius of two km from the actual location of the device. See the implementation in Section 5.2. The name of the facility and the street address is provided along with the marker.

4.4.5 Alarm

The essential element of the alarm (see Figure 4.6) is an eye-catching red button. On a single tap, it can notify emergency contacts in case of danger with both an SMS message and an in-app notification. As mentioned earlier, if contacts have the application installed, they can see the user's real-time location on the map feature. The process of active tracking is characterized by a pulsing animation.

4.4.6 Contacts Management

Users can find two classes of contacts list in this part of the application (see Figure 4.7)—the emergency contacts list and wards list. Users can switch between the contents using tabs that are located on the top of the screen.

The contact's form consists of a name and a phone number field. An optional image can be enclosed. On a selection, the emergency contacts can be edited and even deleted.

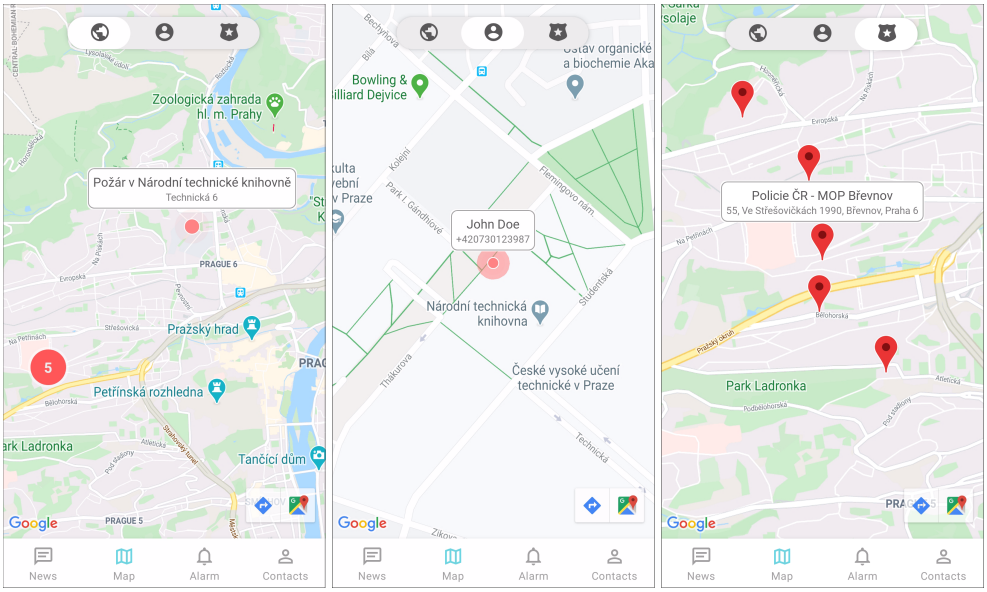

Figure 4.5: Map UI design

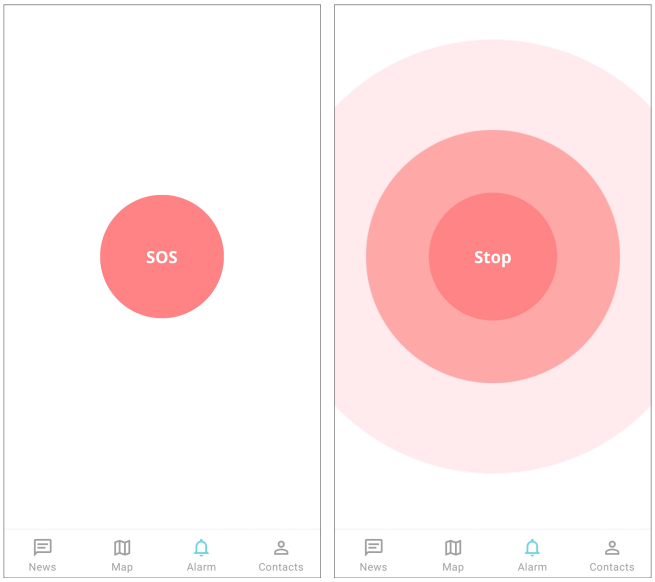

Figure 4.6: Alarm UI design

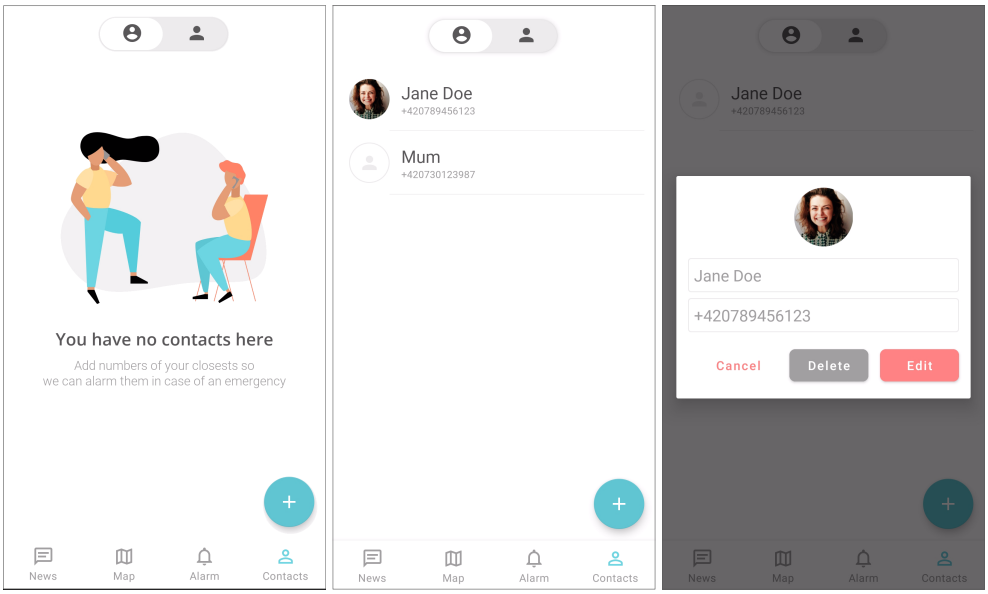

Figure 4.7: Contacts UI design

CHAPTER 5

Technologies

This chapter presents the fundamental technologies used in the application. Platform Firebase is chosen to integrate into the authentication, database, and notification solutions. Google Maps allow to visually mark the reports and user's current location on the map. Both Firebase and Maps fall within Google's competence, thus are preferred among other alternatives as the reliability and smooth cooperation with the Android OS is ensured.

5.1 Firebase

Firebase is a powerful platform for the rapid mobile and web development acquired by Google in 2014 [24]. The platform provides many backend services that are hosted in the cloud, including databases, file storage, authentication, analytics, A/B testing, push messaging, and the list goes on. All of these products are designed to work well together. The administrative access is provided through the Firebase console. Individual products that were used in the application are further described in the following subsections.

5.1.1 Authentication

Firebase Authentication takes care of restricting access to per-user data simply and securely. It supports identification using passwords and email, phone numbers, popular identity providers such as Google or Facebook, and more. [25] For successful signup into the application, only a phone number is requested (see Listing 5.1). Users can be instantly verified, or they can receive a 6-digit verification code via SMS message for further validation. The whole process of authentication is visualized in Figure 5.1. On some devices, Google Play services can automatically detect the incoming verification SMS and perform authentication without further user action.

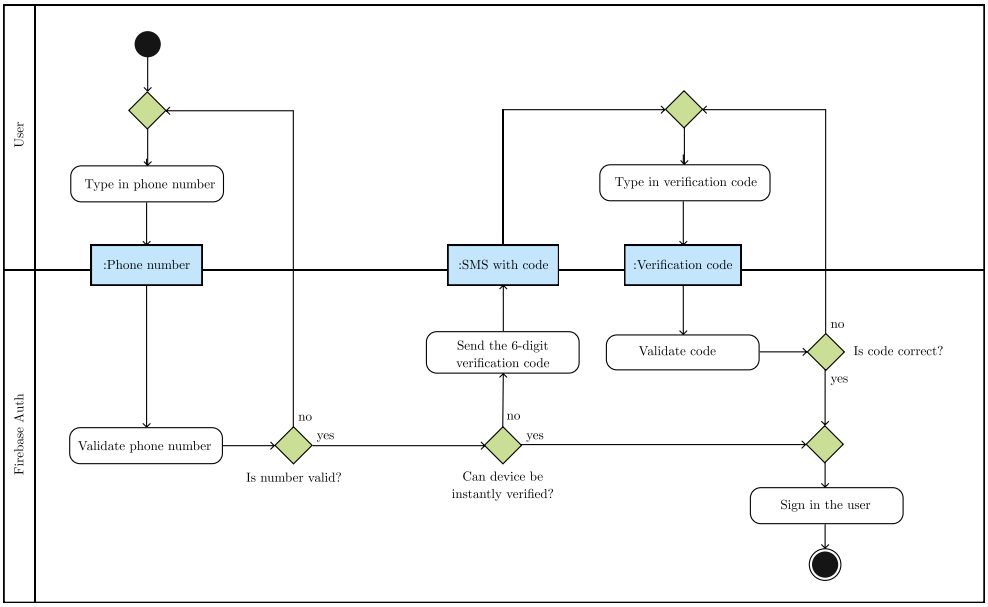

Figure 5.1: Authentication flow

```
private val auth = FirebaseAuth.getInstance()

 /**
  * Validate the provided verification code. Sign in or register
  * the user if the process is successful.
  *
  * @param activity the context for callbacks
  * @param credential the combination of verification code and id
  */
fun signInWithPhoneCredential(activity: Activity,
                              credential: PhoneAuthCredential)
{
    auth.signInWithCredential(credential)
        .addOnCompleteListener(activity) { task ->

            // Register the user if the signup is successful
            if (task.isSuccessful) {
                if (task.result?.additionalUserInfo?.isNewUser ==
                ↪  true) {
                    listener.onNewUserRegistered()
                }
            }
        }
}
```

Listing 5.1: Firebase Authentication callback example

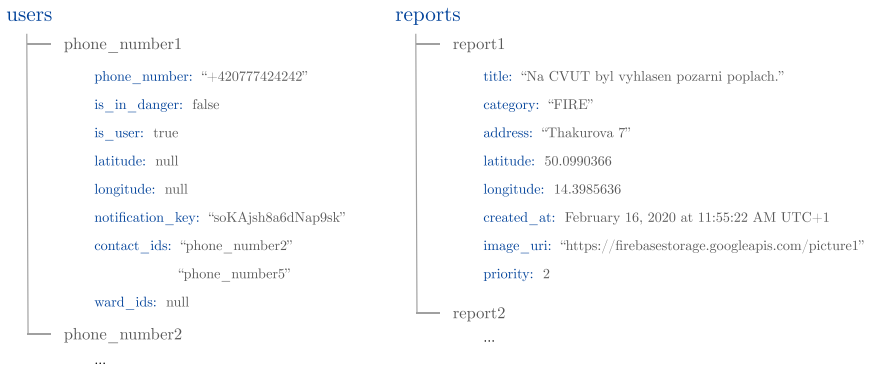

Figure 5.2: Cloud Firestore data structure

5.1.2 Cloud Firestore

Firebase provides two types of non-relational databases—Firebase Realtime Database and Cloud Firestore [26]. Both databases allow to store and sync data between users in real-time using listeners that gets invoked every time a change is observed. The main difference and the reason why the Firestore was chosen is that the data are more structured. Data in Realtime Database are stored in one JavaScript Object Notation (JSON) tree, whereas Firestore contains documents and collections which provide better querying (see the example in Listing 5.2). The application's data structure, as stored in Firestore, can be seen in Figure 5.2. `Reports` and `Users` correspond to collections and the instances with ids `report1` or `phone_number2` are called documents.

5.1.3 Cloud Storage

Cloud Storage is built to directly upload and download files such as images, audio, video, or other user-generated content [27]. In the application, it is primarily used when creating a crime report enclosing a photo (see Listing 5.3). The Cloud Storage will return the photo location path, which is then stored in a Firestore document of a given report.

5.1.4 Cloud Functions

With Cloud Functions for Firebase, it is possible to write and deploy code that automatically responds to events coming from other Firebase products [28]. The application notifications are sent to all emergency contacts whenever the attribute stored in Firestore, saying whether the user is in danger or not, updates (see Listing 5.4). The source code of the function is written in language JavaScript and is stored in Google's cloud, so there is no need for self-managing servers.

```
import com.google.firebase.firestore.DocumentChange.Type

private val firestore = FirebaseFirestore.getInstance()

// Listen to Reports collection changes
firestore.collection("REPORTS").addSnapshotListener { snap, _ ->
    snap?.documentChanges?.let { changesList ->
        for (change in changesList) {
            when (change.type) {
                Type.REMOVED ->  // The report was removed
                Type.ADDED ->    // The report was added
                Type.MODIFIED -> // The report was modified
                else -> throw
                ↪  RuntimeException("Change type not recognized.")
            }
        }
    }
}
```

Listing 5.2: Cloud Firestore listener example

```
val storage = FirebaseStorage.getInstance()

/**
 * Upload the image as a file (named as the last path segment
 * of the image address) to the Reports folder on Cloud Storage.
 *
 * @param imageUri the image address on device storage
 */
fun uploadReportImage(imageUri: Uri) {
    // Reference to the Reports folder on Cloud Storage
    val reportsRef = storage.reference.child("REPORTS")
    val lastPathSegment = imageUri.lastPathSegment
    val photoRef = reportsRef.child(lastPathSegment)

    photoRef.putFile(imageUri)
            .continueWithTask { photoRef.downloadUrl }
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    // Upload of report image was successful
                    val imageDownloadUrl = task.result
                } else {
                    // Upload of report image failed
                }
    }
}
```

Listing 5.3: Cloud Storage query to upload report image

```
// Listens for user updates in Firestore at users/:userPhoneNumber
exports.alarmNotification = functions.firestore
                                    .document('users/{phoneNumber}')
                                    .onUpdate((change, context) => {

    // Check whether the user is in danger and send a notification
    // to everyone in his list of emergency contacts
}
```

Listing 5.4: Cloud function Notification

5.1.5 Firebase Cloud Messaging

With Firebase Cloud Messaging, it is possible to send battery-efficient acknowledgments and notifications to client devices [29]. The recipient could be an individual, a group of users, or subscribers, which makes it easier to deliver relevant information according to the user's needs. This key functionality keeps the user engaged and connected. On the client-side, Firebase Messaging Service class is implemented to handle the received notifications and to customize the message content further. To deliver a notification, a unique device token is generated upon the application's installation and stored in Firestore database.

5.2 Google Maps

With 99 percent of the world coverage, Google Maps are definitely a number one map service. This platform comes with a wide variety of features for instance satellite imagery, robust UI controls, location tracking, and location markers. [30] To integrate the Maps into the application, Maps Software Development Kit (SDK) had to be enabled, and the unique API key to access the service had to be generated in the Google Developer Console[9]. Followings are the leading products of Google Maps platform that were used:

Maps This product gives users the ability to use zoom, rotate, and tilt with simple gestures to adjust the map according to their needs. It provides numerous types of map stylization along with a set of custom graphical elements that were described in Section 5.2.

Places According to the documentation, the library has over 150 million points of interest in its database. To ensure the information is always up-to-date, the data are open for the millions of daily active users to change. [30] With this product, the application can find the nearest police stations and provide their names, addresses, and contact information (see the search query in Listing 5.5).

[9]Google Developer Console stores all the project's settings, credentials, and APIs.

```
"https://maps.googleapis.com/" +
"maps/api/place/nearbysearch/json?" +  // Request for JSON file
"location=${latitude}," +              // User's location latitude
"${longitude}" +                       // User's location longitude
"&radius=2000" +                       // Within a radius in meters
"&type=police" +                       // Type of data
"&key=$APP_KEY"                        // The application key
```

Listing 5.5: Nearby police search query according to Map Places

```
// Launch DI and initialize the modules
class BaseApplication : Application() {
    override fun onCreate() {
        super.onCreate()
        startKoin {
            androidContext(this@Application),
            modules(listOf(firebaseModule))}
    }
}

// Define entities which will be injected at some point in the app
val firebaseModule: Module = module {
    single { FirebaseFirestore.getInstance() }
}

// Inject an instance when needed
private val firestore : FirebaseFirestore by inject()
```

Listing 5.6: Koin injection example

5.3 Koin

Koin is a lightweight Dependency Injection (DI)[10] framework written in pure Kotlin [31]. It is suitable for this application, especially due to its simplicity (see the example in Listing 5.6). It is also easier to test and has better documentation than other DI frameworks such as Dagger or Kodein.

Furthermore, the DI concept fits the multimodular structure of the application. Submodules are only aware of the interfaces but not specific implementations that may be located elsewhere. The actual instances are injected by Koin.

[10]Dependency Injection is a design pattern that allows the creation of dependent objects outside of a class that depends on them [19].

```
// Load the report's image into the view container
Glide.with(context)
     .load(report.imageUri))
     .transform(CenterCrop(), RoundedCorners(14))
     .into(vImage)
```

Listing 5.7: Glide image loading example

5.4 Glide

Glide is a robust Android framework that provides media management and image loading [32]. In the application, it is essentially used for displaying and resizing images that are fetched from the Firebase Storage as it efficiently uses media decoding, memory, and disk caching (see the example in Listing 5.7).

There is a similar library called Picasso. In comparison, Picasso is smaller in size, yet the first image load can take much longer. In this case, efficiency is preferred above the memory capacity. This choice is not excluded from future discussions as of this moment, the application does not contain a large amount of data.

CHAPTER 6

Implementation

This chapter contains the description of development practices used for the final prototype implementation. The solution was carried out with Android architecture components such as `LiveData`, `ViewModel`, and a persistence library Room. Further in the chapter, the answer to how UI updates, asynchronous tasks, and permission requests that plays a crucial role in the application can be found.

6.1 Architecture Components

There were times when all the code was written in the `Activity` class making it a so-called God component. As time went by, a different approach is followed. The `Activity` should essentially be only aware of its lifecycle rather than the business logic. In 2017 Google announced its recommended approach for solving this Separation of Concerns with Android architecture components [33]. The main elements introduced are a persistence library Room, `LiveData` and `ViewModel` where the main logic is moved to. The major benefits of using these components are:

- Application structure is easy to refactor.
- UI is instantly and automatically updated upon data changes.
- Data survives on configuration changes, and there are no memory leaks.
- The boilerplate code related to SQLite database is reduced.

```
class ReportViewModel(private val reportsRepo: ReportsRepo) :
    BaseViewModel() {

    // The list of all reports from the database
    val allReports: LiveData<List<Report>> = reportsRepo.allReports
}

class ReportsFragment : BaseFragment() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Observe the LiveData holding the list of all reports
        viewModel.allReports.observe(this) { reports ->
            // Do something with updated data here
        }
    }
}
```

Listing 6.1: LiveData observing example

6.1.1 LiveData

To put it plainly, `LiveData` are data holders following the Observer pattern[11] [19]. The meaning behind the name `LiveData` is that they are aware of the lifecycle of the interested observer. This ensures that only active subscribers are updated, which leads to no memory leaks and crashes to configuration changes. This feature takes much work from programmers' hands. Observers can then propagate the current data state into the UI layout. `LiveData` are designed to work well with `ViewModels` (see the example in Listing 6.1).

6.1.2 ViewModel

The main purpose of the `ViewModel` is to hold and manage UI-related data in the separation, which are then exposed usually through `LiveData` or data binding (read more about data binding in Section 6.2). Same as `LiveData`, it is lifecycle conscious, which makes data survive configuration changes, including screen rotations and keyboard availability. [19]

It communicates with both `Activities` and data repositories allowing to have not only user inputs but also data loaded from the database in one place. The data can be observed by multiple screens that share the same functionality. In the application, there is `AuthViewModel` accessible from both `PhoneNumberFragment` and `VerificationCodeFragment` (see signup UI design in Section 4.4.2).

[11] The data holder does not know where to send the updates. However, subscribers will automatically get the changes as soon as they are accessible. [19]

6.1.3 Room

Room is a persistence library that allows simple interactions over a robust database such as SQLite [34]. It provides an abstraction layer that maps the database entities into SQLite readable Plain Old Java Objects (POJOs) using elementary annotations. It reduces the amount of boilerplate code that traditionally follows the operations with database connection. Room can also validate the Structured Query Language (SQL) queries at compile time.

As mentioned before, it creates seamless integration with other architecture components. How the data sources architecture was designed can be seen in Figure 6.1. The application contains three main Room components (see the code example in Listing 6.2):

Entity Room creates a table within a database for each class that has `@Entity` annotation. The fields in the class will correspond to columns in the table. The names can be customized or generated by Room by default. Another annotation, such as `@PrimaryKey`, specifies the unique id of the object. According to the application's database model, the entities are `Report`, `Contact`, and `Ward`.

Data Access Object (DAO) This component is responsible for defining methods that communicate with the database. Once more, queries can be build using explicit annotations in a DAO class such as `@Insert` or `@Delete` without any additional attributes.

Database Database serves as the main access point for the underlying connection to the application's persisted data. To create a database, an abstract class that extends `RoomDatabase` has to be defined. The annotation `@Database`, including the contained entities and the DAO that have access to the database, has to be declared.

6.2 Data Binding

To further simplify the development of UI, the data binding concept is incorporated into the application. The architecture components are included in the layout, which allows views to communicate directly with the `ViewModel` objects. This ensures the UI logic is moved out of the layout, which leads to easier testing, and at the same time, it reduces the number of UI calls in the `Activities`. Furthermore, the view attributes can listen to the `ViewModel` and propagate data source changes instantly into the UI (see data binding example in Listing 6.3).

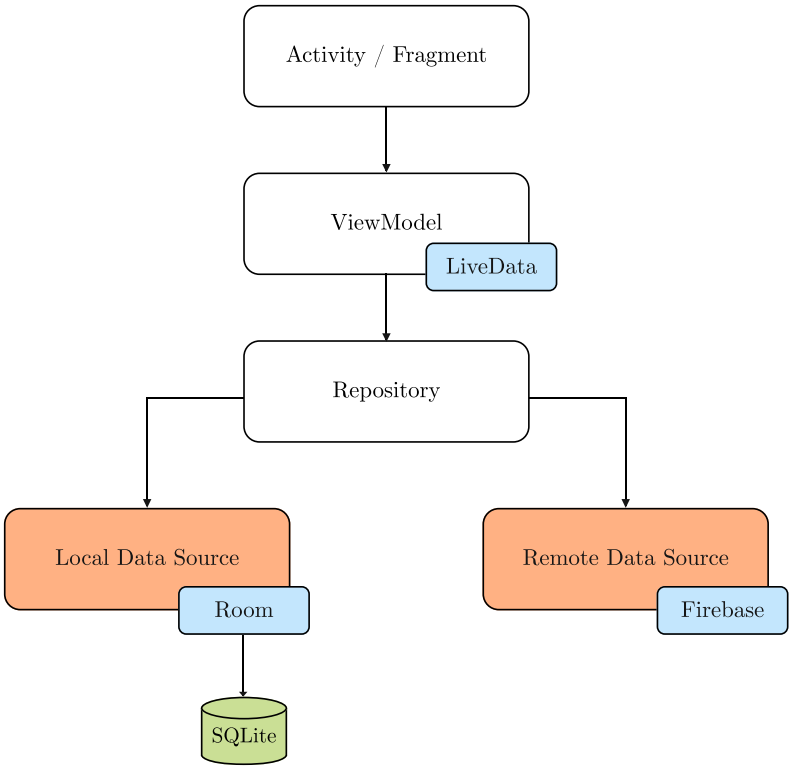

Figure 6.1: Data sources architecture

6.3 Layout Views

Devices running the Android OS are very distinct regarding the wide variety of sizes and resolutions. That is the reason why hardcoded values are considered a bad approach when it comes to UI implementation, as the layout may not look the same on different devices. The application uses `ViewGroups` such as `ConstraintLayout` that supports working with percentages and scaled dimensions as much as possible. Moreover, to correctly display items in a scrolling list, the `ViewGroup` `RecyclerView` is presented in this section.

6.3.1 ConstraintLayout

`ConstraintLayout` is a `ViewGroup` that can build large and complex layouts with a flat view hierarchy, which is beneficial to not only readability but also the efficiency of the layout display. The view's position is outlined by its relations to other sibling and parent views. [35] At least two constraints (horizontal and vertical) have to be defined in order to anchor the view. What is more, `ConstraintLayout` supports chaining the views in numerous styles.

```
@Dao
interface ReportsDAO {

    // Get the list of all reports sorted by time by default
    @Query("SELECT * FROM reports ORDER BY timeAgo DESC")
    fun getAllReports(): LiveData<List<Report>>
}

// The Room database for storing Report entities
@Database(entities = [Report::class], version = 1)
abstract class ReportsDatabase : RoomDatabase() {

    // Reports data access object for database interactions
    abstract fun reportsDAO(): ReportsDAO
}

// The crime report that is shared among the users
@Entity(tableName = "REPORTS")
data class Report(

    // The report's unique id used as a primary key in database
    @PrimaryKey
    val id: String
)
```

Listing 6.2: Room components example

Guidelines

`Guidelines` are helper objects that are not visible by default. They serve as constraints or borders for views inside the `ConstraintLayout`. They can be oriented either horizontally or vertically, and positioned by defining the percent value of the screen (see the example in Listing 6.4).

6.3.2 RecyclerView

There are various practices for displaying a scrolling list of elements of a large amount or that changes frequently. One of the solutions is to use `RecyclerView`. The main reason why this component was chosen is that it creates only as many view containers as are needed to display on the screen plus a few extra for smooth scrolling. As a user scrolls the items, views are not destroyed and recreated but are reused and rebinded to the new content, which leads to a faster response. [36] As the previously visible views are stored in a cache for later reuse, the performance is drastically improved. To bind the content with the views, `RecyclerView` has to be connected with the `RecyclerView Adapter`.

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <data>
        <variable
            name="vm"
            type="cz.cvut.fit.phamobic.safety.ContactViewModel" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        style="@style/Layout">

        <cz.cvut.fit.phamobic.safetyUI.PrimaryButton
            style="@style/PrimaryButton"
            android:onClick="@{_ -> vm.savePerson()}"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"/>

    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

Listing 6.3: Data binding with contact's editing

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    style="@style/Layout">

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/vGuidelinePhoto"
        style="@style/Layout.Wrapped"
        android:orientation="vertical"
        app:layout_constraintGuide_percent="0.7" />

    <ImageView
        android:layout_height="0dp"
        android:layout_width="0dp"
        app:layout_constraintDimensionRatio="1:2"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="@id/vGuidelinePhoto"
        android:layout_margin="@dimen/scaled_dimension_size_8dp"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Listing 6.4: Guideline example

Adapter

The elements in the list are represented by view holder objects. These are managed by the `RecyclerView Adapter`. It binds the views with corresponding data based on its list position. [36] It also supports connection with click listeners on individual items. In the application, adapters are used for both reports and contacts correct display. On a click, reports redirect the user to the map screen where the selected report is in focus. On the other hand, contacts on a click event display the contact's form for editing.

6.4 Asynchronous Tasks

There are many different approaches on how to handle heavy and long-running tasks such as network calls or database operations that should be moved to the background to not overload the main UI thread. The programmers are familiar with `AsyncTasks` and RxJava. With `AsyncTasks` comes a lot of potential issues such as memory leaks. Therefore, they are not further supported. RxJava is a compelling yet both heavy and complex library, and therefore in this manner, it would be very inefficient to use only for threading purposes. Hence for this application, the coroutines were found as the most suitable option.

Google introduced coroutines in 2018 as a new way of writing asynchronous and non-blocking code [37]. Coroutines are written in pure Kotlin and can be thought of as a light-weight thread (see the example in Listing 6.5). Following are the coroutines characteristics and benefits that they bring to the application:

- Coroutines are very simple to read and also understand as they are written sequentially as a non-asynchronous operation. No code nesting is required.
- Coroutines can be safely terminated if the application process is destroyed. They fit naturally in a `ViewModel` scope, which is aware of the lifecycle.
- Like threads, coroutines can run in parallel, wait for each other and communicate. However, in contrast, coroutines are very cheap, almost free. A great number of coroutines can be created within a single thread, and with a small price in terms of performance.
- A thread can suspend a coroutine to work on a different task with higher priority. The coroutine process can then be resumed later on, or another thread could even take over.

```
class ContactViewModel(private val personRepo: PersonRepo) :
    BaseViewModel() {

    // Delete the selected contact from the database
    fun onDeleteContactClicked() {
        selectedContact?.id.let { contactId ->
            viewModelScope.launch {
                personRepo.deleteContactById(contactId)
            }
        }
    }
}

class PersonRepoImpl(private val contactsDAO: ContactsDAO) :
    PersonRepo, CoroutineScope {

    // The coroutine scope that defines used threads
    override val coroutineContext = SupervisorJob() + Dispatchers.IO

    override suspend fun deleteContactById(contactId: Long) {
        contactsDAO.deleteContactById(contactId)
    }
}

@Dao
interface ContactsDAO {

    // Suspend modifier says it has to be executed using coroutines
    @Query("DELETE FROM contacts WHERE id =:contactId")
    suspend fun deleteContactById(contactId: Long)
}
```

Listing 6.5: Coroutines example with a database operation

6.5 Permissions

With the announcement of Android Marshmallow (SDK 23), a new runtime permission model (see Figure 6.2) was introduced [38]. Not only the permissions have to be declared in the application manifest as before, but now they also have to be requested at runtime (see the example in Listing 6.6). This solution gives users full control over the application permissions and enough context on why they are needed. Permissions can be divided into two groups:

Normal permissions These permissions do not directly affect the user's privacy. If the permission is listed in the application manifest, then it is automatically granted by the system upon installation. Unlike dangerous permissions, they do not have to be checked at runtime.

Dangerous permissions These permissions give the application access to the user's sensitive data such as location, camera, contacts, or storage access. They can potentially affect the system and other apps. Not only they have to be listed in the manifest, but the user has to give permission to use them explicitly.

Before this concept, the user had to grant all the permissions before the installation. However, now, when the user does not accept a permission check, the application should still be working even with the limited set of functionalities. As the user can manually revoke the approval at any time in the device's settings, permissions have to be checked every time the dangerous functionality is required. For the features' proper behavior, four dangerous permissions are listed in the application manifest:

Access Fine Location This permission allows the API to fuse the data from all available location providers, including the GPS, as well as Wi-Fi and mobile cell data, to return as precise location as possible [39]. This is the key permission for locating both users calling for help and crime reports published by the users.

Write External Storage With this permission, the application is able to write to the external storage of the device [39]. It is especially requested upon saving the contacts profile pictures into the device's photo gallery.

Camera This permission is required to access the camera hardware of the device and other related features [39]. It is requested upon optional enclosing of the reports images and contacts profile pictures.

SMS SMS permission allows the application to receive, read, and send SMS messages regarding the alarm button. All the emergency contacts receive an SMS that contains the user's current location and battery status of the device.

6.5.1 Location Updates

The purpose of `LocationRequests` is, as the name suggests, to ask for the location of the device. As shown in Listing 6.7, many attributes can be customized according to application needs, such as the accuracy level, the update interval, or the amount of power consumed by requests. The location is retrieved with a `FusedLocationProviderClient` under Google API. When the system is not running, the location updates are removed. To use this feature, the location permission has to be granted, and also GPS turned on.

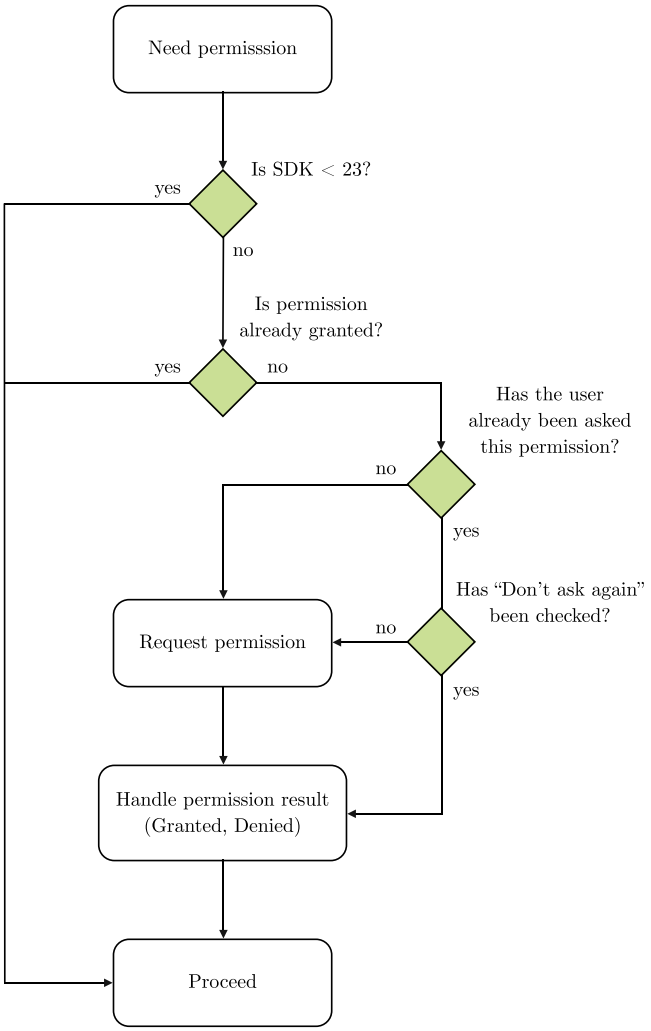

Figure 6.2: The permission flow from Android 6.0 [40]

6.5.2 SMS Management

With `SmsManager`, sending and receiving SMS messages is very simple. This feature is tightly connected with the alarm button. If the appropriate permissions are granted, an SMS is sent to all user's emergency contacts with not only the current location address but also a percentage status of the device's battery. As the content's length dynamically vary, the message content is split into multiple messages if the text is too long. An enhancement that would allow users to choose a time interval in which the SMS would be sent repeatedly can be discussed in the future.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
    if (ContextCompat.checkSelfPermission(it, CAMERA) ==
    ↪  PERMISSION_DENIED || ContextCompat.checkSelfPermission(it,
    ↪  WRITE_EXTERNAL_STORAGE) == PERMISSION_DENIED) {

        // Permissions not granted, show Permission Request dialog
        val permissions = arrayOf(CAMERA, WRITE_EXTERNAL_STORAGE)
        requestPermissions(permissions, CAMERA_PERMISSION_CODE)
    }
}

// Called upon ALLOW or DENY press from Permission Request dialog
override fun onRequestPermissionsResult(requestCode: Int, permissions:
↪  Array<out String>, results: IntArray) {
    if (requestCode == CAMERA_PERMISSION_CODE) {
        if (results.isNotEmpty() &&
            results[0] == PERMISSION_GRANTED) {
            // Permission granted, capture a photo
        }
    }
}
```

Listing 6.6: Camera permission request example

```
val client: FusedLocationProviderClient? =
↪  LocationServices.getFusedLocationProviderClient(context)

// Request location updates and get current position of the user.
override fun startTrackingLocation(callback: LocationCallback) {
    client?.requestLocationUpdates(getLocationRequest(), callback,
    ↪  null)
}

// Stop location updates and clear the user's current location.
override fun stopTrackingLocation(callback: LocationCallback) {
    client?.removeLocationUpdates(callback)
}

// Get the location updates with following options:
//    interval -> the update interval in ms
//    fastestInterval -> the rate of updates
//    priority -> the balance between power consumption and accuracy
private fun getLocationRequest(): LocationRequest {
    return LocationRequest().apply {
        interval = 10000
        fastestInterval = 5000
        priority = LocationRequest.PRIORITY_HIGH_ACCURACY
    }
}
```

Listing 6.7: Location request example

CHAPTER 7

Testing

In this chapter, individual test categories and third-party libraries that were used to ensure the code correctness and functional behavior are provided. Among the applied frameworks, there is also Firebase platform. Its close integration with Android Studio helped connect the application with several valuable test tools such as Analytics, Crashlytics, and Performance Monitoring. Lastly, usability tests are performed on a group of people—two of them being police officers. Along with supplementary questions and results, they are stated at the end of this chapter.

7.1 Test Categories

Android Studio acquires built-in testing functionality to ease the process in full measure. Depending on the test location in source code directories and the environment in which the test will be running, Android Studio identifies two types of tests—Unit and Instrumented tests. [41]

7.1.1 Unit Tests

Unit tests, also called local tests, run entirely in machine's local Java Virtual Machine (JVM), meaning no emulator or a real device is required. Generally, they test the internal logic where the Android framework dependencies are not necessary (see the example in Listing 7.1). Thus, leading to faster performance. On the other hand, they do not consider the conditions of the physical device. Hence, they do not reflect the real world. Unit tests are written with JUnit, a standard unit testing framework for Java.

```
@RunWith(RobolectricTestRunner::class)
class ReportViewModelTest {

    @Test
    fun testOnCrimeCategorySelected() {
        val categoryAbuse = CategoryView(getApplicationContext())
        val categoryTheft = CategoryView(getApplicationContext())

        reportViewModel.apply {
            // At the beginning nothing is selected
            assert(checkedCategory == null)

            // 1. Select category Abuse - it should be checked
            onCrimeCategorySelected(categoryAbuse)
            assertThat(checkedCategory, `is`(categoryAbuse))
            assertThat(checkedCategory?.isChecked, `is`(true))
            assertThat(categoryAbuse.isChecked, `is`(true))
            assertThat(categoryTheft.isChecked, `is`(false))

            // 2. Select category Theft - it is the only checked
            // 3. Select category Theft again - nothing is changed
        }
    }
}
```

Listing 7.1: Unit test with Hamcrest example

7.1.2 Instrumented Tests

Instrumented tests, often called as Android tests, which may be confusing, require a hardware device or an emulator. Interactions regarding the test operations are visually drawn on the screen. In the vast majority of cases, they are written to test the application integrity, the correctness of the UI functionality, and to automate user interactions. In the application, most of the instrumentation tests are written with Espresso library (see more in Section 7.2.1).

7.2 Used Libraries

Android platform supports several different kinds of testing frameworks and libraries. In this manner, application test capabilities are further extended with frameworks such as Espresso, Robolectric, and Hamcrest, which converts the test code into a more readable language acting more as documentation. All of the mentioned libraries are industry-standards in the Android world. With the recommendation by Google [42], they are an obvious choice.

```
@RunWith(AndroidJUnit4ClassRunner::class)
class ReportDialogFragmentTest : KoinTest {

    @Before
    fun setUp() {
        launchFragmentInContainer<ReportDialogFragment>(null,
        ↪  R.style.AppTheme)
    }

    @Test
    fun testPrioritySliderWorks() {
        onView(withId(R.id.vPrioritySeekbar))
            .check(matches(isDisplayed()))
            .check(matches(withSeekbarProgress(0)))
            .perform(setProgress(2))

        onView(withId(R.id.vPrioritySeekbar))
            .check(matches(isDisplayed()))
            .check(matches(withSeekbarProgress(2)))
    }
}
```

Listing 7.2: Espresso test of the slider component

7.2.1 Espresso

As an emulator or a hardware device is required to perform Espresso tests, they are instrumented by nature. The tests are written based on what a user might do while interacting with the application, including clicking buttons, sliding a bar, or even entering data. [43]

Espresso tests can be compared to a robot that listens to a script of actions. First, it must find a given view on the screen. Second, perform a task which will often trigger a response. Finally, check the corresponding result or content. If it is unable to finish any of the steps properly, or the results do not meet the expectations, the test fails. The example of individual steps can be seen in Listing 7.2.

7.2.2 Robolectric

When particular Android dependencies are needed while testing, a simulated Android environment, rather than the emulator or real device, can be preferred as it runs faster. Robolectric library creates such an environment for both unit and instrumented tests and is usually practiced with integration tests, where application context is required. [44]

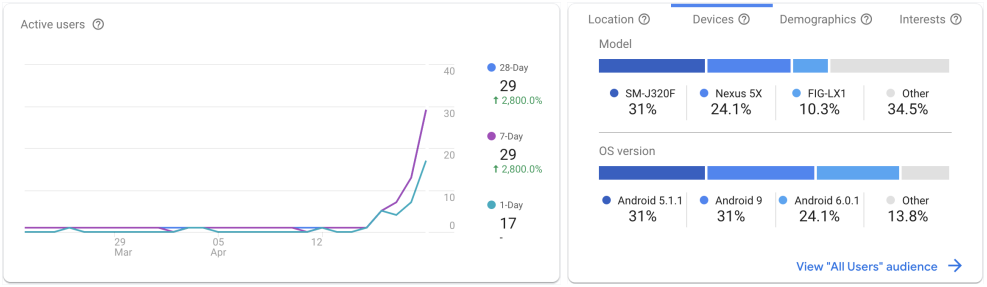

Figure 7.1: Analytics screenshots from Firebase console

7.3 Firebase

It is no surprise that even the testing category falls into Firebase responsibilities. With its close cooperation with Android Studio, it is possible to integrate Firebase testing products directly through the development environment simply. Associate logs are visible in the IDE as well as the Firebase console. The testing products that were used to ensure the application's quality are described in the following sections.

7.3.1 Google Analytics

With Google Analytics for Firebase, it is possible to log application events in order to understand appealing content and choose user properties to identify the characteristics that distinguish the behavior of different groups of users [45]. With all this information, Analytics allows sending targeted notifications or functionalities regarding optimizations. By default, Analytics logs several key user properties such as language, location, device model, and OS version (see Figure 7.1). Custom properties are not adopted yet as the only information the prototype requires is a user's phone number. However, in the future, if the user is willing to provide demographic information, statistics on the most vulnerable segment of users could be obtained.

7.3.2 Crashlytics

With Crashlytics, it is easy to detect common issues that impact a larger amount of users (see Figure 7.2). Furthermore, it collects many device characteristics such as a battery level or network connection in order to maximize the chance of fixing the problem. [46] Crashlytics are generally used for a production code, where there is no possibility to debug the device locally. Custom logs were integrated into the main application features regarding reports, alarm, authentication, and SMS sending to further understand the context of the error.

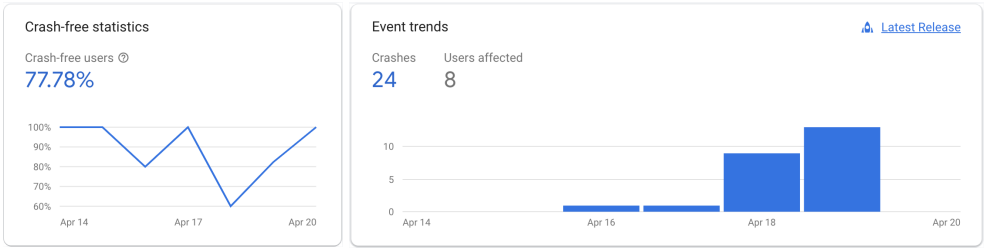

Figure 7.2: Crashlytics screenshots from Firebase console

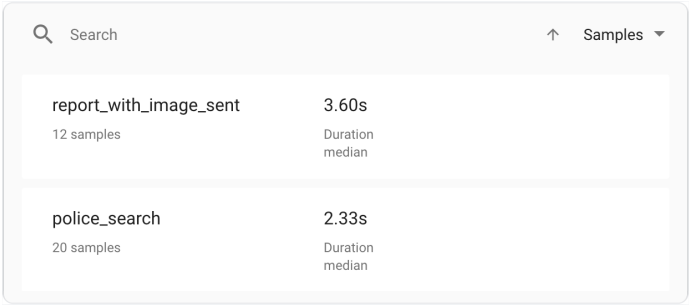

Figure 7.3: Performance Monitoring screenshot from Firebase console

7.3.3 Test Lab

As mentioned before, instrumented tests written with Espresso are primarily to test the correctness of the UI. However, programmers rarely have the resources to run tests on multiple devices with various sizes and different versions of OS. Furthermore, working with emulators can often be time-consuming. With a few configurations in Android Studio, Test Lab provides a direct connection to a wide range of devices that are operated right from Google's cloud [47]. Test results can be reviewed in the Firebase console along with performance and even a record, including screenshots.

7.3.4 Performance Monitoring

Firebase Performance Monitoring provides an insight into how long it takes the application to complete a specific task. By default, it automatically measures attributes regarding the application lifecycle, such as the time period between when the application is opened and when it becomes responsive. [48] The application traces the performance of specific parts of the code associated with the police search and reports sending that includes an image. It is assumed, these two operations will take the longest time as they both execute heavy network tasks. Their median duration can be seen in Figure 7.3.

7.4 Usability Testing

A small group of people was chosen to provide feedback on the application. Each of them had a different level of technical knowledge. The participants consisted of experienced programmers, students, a married couple, and two members of the police force. They were divided into smaller groups and tested simultaneously as it is assumed, the application will be used at least in pairs. The whole communication was carried out online through video conferencing tools, where the testers' devices were mirrored onto the screen[12].

7.4.1 Basic Operations

To cover most of the application's functionality during the testing process, a list of most significant user interactions and flows was prepared in advance. First, a brief description of the application was given to put testers into context. Then, after successful signup, they had a minute to get quickly acquainted with the application design and navigation. Following is the list of all the basic operations testers had to complete:

1. Sign up with your own or a test phone number.
2. You see a man hiding a bomb under a jacket. Create and send a report.
3. Display this report on a map.
4. Sort the reports by the distance from your current location.
5. Find the nearest police stations in your area.
6. Choose members of the group and add them to your contact list.
7. Change and delete some of your contacts.
8. Call for help when others have the application in the foreground.
9. Call for help when others have closed the application.
10. Find the caller on the map and display directions to him/her.

7.4.2 Usability Results

The very first problem that occurred with the first group of testers was that they were unable to sign up with their own phone numbers. It was due to missing release key in Firebase configurations. This issue was fixed and did not arise later on with the following testers.

[12]Usability tests were not performed in person due to government orders during the COVID-19 pandemic [49].

Another problem that appeared was that on devices with higher resolution, the UI elements inside the report form had incorrect sizes, although they worked properly. Operations regarding the reports, such as creating one, sorting, and displaying a specific report on the map, were performed without difficulties. Testers knew what to do immediately as well as there were no issues from the technical point of view.

The hardest task appeared to be finding the directions on the map to a person calling for help. As the person's location was already in focus with all additional information displayed, it was not intuitive for the testers to also click on the place marker to show the native Google Map functions such as directions. One of the testers had even uninstalled the Google Map application, so this functionality was disabled to him.

The rest of the operations were successfully completed. However, some of them were not carried out straightforwardly, and a number of uncertainties were discovered. Thus, leaving room for improvements. Followings are issues that were unclear:

- At first sight, testers did not know what individual map tabs represent.
- At first, they were uncertain of what exactly the alarm button does.
- They were not sure what is the difference between the emergency contact and a ward until the alarm was triggered.
- Even though a text hint was provided, testers were not sure whether to type the phone number in a full format, including the country calling code.
- If a person was calling for help and the emergency contact had the application in the foreground. It was not clear that the caller can be found on the map.

7.4.3 Supplementary Questions

After all the functionalities were tested, a couple of supplementary questions were asked. Most of them were answered positively. Testers had a variety of constructive suggestions and ideas for future improvements. Followings are the questions of the survey:

- Is the application's design clean and understandable?
- What functionalities are you missing?
- Did you find any issues with the performance of the application?
- Would you consider using the application for personal needs? If not, please explain why.

All of the testers agreed on the UI design being clear as they did not find any unnecessary elements. However, some of the testers would appreciate additional labels on tabs or more information on the alarm. This requirement of a clear design is a crucial part of the thesis as it is the main condition for the elderly to be able to use the application. Except for times when uploading a report that included an image was rather lengthy, there were no issues with the performance. All testers would consider using the application, assuming that only the reports that are relevant to them are displayed, and they are correctly validated. Followings are some of the suggested functionalities that were found especially valuable and are taken into consideration:

- Putting an application link directly inside the alarm SMS message. This would primarily benefit emergency contacts that do not have the application installed yet.
- At the moment, all emergency contacts are notified in a situation where the user is in danger. Testers suggested notifying only selected people of the user's choice.
- Adding a notification that a ward is in danger directly into the application screens.
- Assigning a priority value to individual crime categories to more specify the overall severity.

7.4.4 The Police Review

This thesis obtained an immense opportunity to test two of the current members of the Police of the Czech Republic. One of them working as a patrol that is deployed to the crime scenes by the integrated centers operating on national emergency line 158. The other one being a criminalist that is responsible, among other things, for conducting investigations. As the video call could not be managed due to the officers' busy schedule, a document with application specifications, and all possible interactions was made and sent them for a review. The text is in Czech language and is enclosed in this thesis. Moreover, a few additional questions regarding the police were attached.

- Do you find the application useful? Would you consider using it for personal needs? If not, please state why.
- Does the application have the potential to help the police department assuming that reports are correctly validated, and no duplicates appear?
- What is your opinion on adding functionality that would allow police forces to react on reports' situation or even create an area alarm themselves?

The police response was very comprehensive and helpful. Both stated that the application would be taken into consideration regarding violent crimes, including murders, bodily harms, thefts, and rapes. However, violent crimes take up a tiny percentage of the overall reports. Moreover, according to their statement, 90 percent of the time, they are committed by an acquaintance. The vast majority of crimes composes of offenses against the property where the offender cannot be found on the crime scene. Thus, reports are not substantial in this case. Followings are other reasons why the application would be rather a burden for the police department:

- In the present moment, the police department has no resources to appropriately react to the crime reports that would be acknowledged in the application nor to create its own announcement.
- By law, police officers have to invest and physically check every crime report. There is no other way how to filter the reports, nor can no one be in charge of whether to send a patrol or not. It has to be sent every time.
- Moreover, spread about wrongdoings in the user's surroundings could have an opposite effect. It could lead to civil unrest and panic.

Although the police officers do not see the possibility of the application's integration, they would consider using it for personal needs regarding the alarm button. They believe it is particularly convenient in situations where people want to make sure their close ones, especially children and the elderly, are safe. However, implementing an offline alarm is a must condition for them as people in stress situations rarely have a clear mind, and their manipulation skills are often limited.

7.4.5 Summary

To perform usability tests, a group of testers was selected to complete a series of application's basic operations. Testers had devices with various versions of OS and different screen resolutions. Hence, device coverage was also tested. Supplementary questions were answered at the end of the session.

Technical issues that occurred during the testing, such as the inability to sign up and UI layout being incorrectly displayed, are fixed and are no longer detected in the final prototype. In addition to that, it is ensured that when users delete an emergency contact, their location is no longer tracked and shared with this person. Based on the test results, a few testers' suggestions, such as an introductory tutorial and an automatic recognition of a country calling code, are more described in Chapter 8. The tutorial would answer most of the questions and solve misunderstandings that had occurred.

One of the most valuable parts in this section is the police point of view and their opinion on the application's usage in the Czech Republic. Although integration with the police forces does not seem feasible, there were positive responses to the alarm function of the application. With this insight, future enhancements will be focused on the process of calling for help in emergencies.

The testing process showed that the application works very well under the simultaneous usage of multiple testers. Especially the location tracking as it was observed to be accurate and swift. Furthermore, it was proven that the UI is clean and understandable. Many testers found the application valuable and would consider using it. Overall, the usability test was a success as no significant issues, that were not immediately fixed, were found.

CHAPTER 8

Future Improvements

To satisfy the basic requirements of the final prototype, only core features were implemented. The additional enhancements were not in the scope of the application but are definitely essential for the future release into the public community.

Introductory tutorial Based on confusions that appeared during the user-testing, an introductory tutorial should be displayed upon application installation. Although onboarding screens provide a simple description of basic functionalities, they did not prevent the misunderstandings that occurred inside the application core. The tutorial would include detailed information on individual map tabs, the difference between an emergency contact and a ward, as well as a specification on what exactly happens when the alarm is triggered.

Offline alarm In the application, the user can call for help with a single tap of a button. To go a little bit further, the time spent interacting with the application before the actual notification of contacts can be shortened up. In the future, the alarm could go off even when the application is not on the foreground. It could be triggered by pressing a combination of system buttons such as a long press on both of the volume buttons, by device shake, or on a key phrase of user's choice.

Crime reports filter The crime reports should be filtered by the locality as the ones that are not in the user's surrounding might be irrelevant. Moreover, an algorithm that would distinguish false or improper reports could be implemented. One of the solutions is to mark these reports by the users themselves or ban the authors from the application. Another enhancement that would improve the application's quality is the ability to identify reports duplicates based on the comparison of key phrases contained in the title, location, or the crime category.

Video and audio recordings To provide users with the maximum information, the audio-visual elements should be more integrated within the application. They could be attached to crime reports or the alarm. This feature would probably lead to communication channels such as chat rooms between the users where the recordings would be uploaded in case of an emergency. Giving the assumption that the application will be most of the time used outside, the slow internet connection should be taken into consideration.

Contacts requests The current state of the application allows users to add anyone as their emergency contact. To avoid the misusage, the contact requests should be implemented for both parties to agree to the location tracking. They will also have the possibility to remove or even block one another without the other's permission.

Auto recognition of country calling codes The application should be able to automatically recognize a phone country code and, by default, suggest it as a first option to the user. This will take the necessity of writing the full phone number from the user's hands. Furthermore, the emergency numbers for medical help and the police could be detected according to the device's location.

More languages support Now the application supports only two languages—English and Czech. In the future, the translation to the top most spoken languages such as Chinese, Spanish, or French should be added, to approach as many people as possible. As the application was designed to adapt to other languages, this feature should be the easiest to include.

IOS version As of today, devices running on iPhone OS take up approximately one-third of the whole mobile operating system market, which is a significant portion [50]. Thus, in the future, it should be ensured that this platform is also supported. A solution, where source code that is already written on the Android side is shared between both of the platforms, should be taken into consideration.

Conclusion

One of the purposes of this thesis was to analyze existing mobile applications dealing with the problem of crime prevention in the Czech Republic and worldwide with a focus on personal safety applications. This part has been fulfilled in Chapter 1. Furthermore, based on the applications' comparison, a detailed analysis of requirements was performed.

Subsequently, application design has been created based on established requirements. What is more, the description of the selected architecture patterns was given. The UI design was composed in a way it follows the Material Design concept and is clean to the greatest extent.

Successful implementation of the functional prototype was carried out using standard mobile development practices and applying well-known modern technologies such as Firebase platform and Google Maps interface. The final application contains an in-app alarm to notify emergency contacts in danger as well as functionality to create and display crime reports. The map also plays a crucial role in the application, as it is the place where the user in danger is tracked in real-time. Moreover, the nearest police stations can be found here. The rest of the requirements were also accomplished.

Another aim was to perform usability testing, which was carried out with a chosen group of people from various fields. In addition to that, the application was reviewed by two of the members of the Police of the Czech Republic. It was discovered that the integration with the police forces is unlikely as they do not have enough resources to manage the application. However, using the application regarding the alarm was stated promising. Based on testing, future improvements such as an introductory tutorial, offline alarm, or support for iPhone OS are discussed in the last chapter of the thesis.

Bibliography

1. *Writings*. 7th print. New York City: Library of America, [© 2007]. ISBN 978-0-940450-33-2.
2. *V ČR loni vzrostla kriminalita o 3,5 procenta, přibylo i vražd [Last year in the Czech Republic, criminality increased by 3.5 percent, the number of homicides has also increased]* [online]. Česká tisková kancelář, © 2018 [visited on 2020-04-29]. Available from: `https://www.ceskenoviny.cz/zpravy/-v-cesku-loni-vzrostla-kriminalita-o-3-5-procenta-na-199-221-cinu/1842856`.
3. *Lexico* [online]. Oxford University Press, 2019 [visited on 2019-11-17]. Available from: `www.lexico.com`.
4. *Google Play* [online]. Google Inc., © 2019 [visited on 2019-12-02]. Available from: `https://play.google.com/store`.
5. *Záchranka* [online]. Brno: Záchranka s.r.o., 2016 [visited on 2019-12-02]. Available from: `https://www.zachrankaapp.cz/en`.
6. *Citizen* [online]. Sp0n Inc., 2017 [visited on 2019-11-14]. Available from: `https://play.google.com/store/apps/details?id=sp0n.citizen`.
7. *BSafe: Never walk alone* [online]. Mobile Software AS, © 2019 [visited on 2019-11-14]. Available from: `https://getbsafe.com`.
8. *Life360, GPS Tracker* [online]. Life360, 2010 [visited on 2019-12-05]. Available from: `https://play.google.com/store/apps/details?id=com.life360.android.safetymapd%5C&hl=en_US`.
9. *Android Developer Challenge: helpful innovation, powered by On-Device Machine Learning + you!* [online]. Google Inc. [visited on 2020-04-29]. Available from: `https://developer.android.com/dev-challenge`.
10. *Announcement: fbFund* [online]. Facebook Inc., © 2020 [visited on 2020-04-29]. Available from: `https://about.fb.com/news/2007/09/announcement-fbfund/`.
11. *Noonlight* [online]. Noonlight Inc., [2013] [visited on 2019-12-02]. Available from: `https://www.noonlight.com`.

12. *Functional and Nonfunctional Requirements: Specification and Types* [online]. AltexSoft, © 2020 [visited on 2020-04-30]. Available from: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/.

13. *What is Android* [online]. Google Inc. [visited on 2019-11-26]. Available from: https://www.android.com/what-is-android/.

14. *Distribution Dashboard* [online]. Google Inc., 2016 [visited on 2020-04-20]. Available from: https://developer.android.com/about/dashboards.

15. CARTER, Kristen. *How Android App Development Became Kotlin-first?* [online]. Hackernoon, 2015 [visited on 2019-11-26]. Available from: https://hackernoon.com/how-android-app-development-became-kotlin-first-bh28929gu.

16. *Fragments* [online]. Google Inc. [visited on 2020-04-13]. Available from: https://developer.android.com/guide/components/fragments.

17. *Activity* [online]. Google Inc. [visited on 2020-04-13]. Available from: https://developer.android.com/reference/android/app/Activity.

18. RAFEEZADEH, Alireza. *Modularization by Feature and Layer with Android Architecture Components* [online]. A Medium Corporation, 2012 [visited on 2019-11-26]. Available from: https://medium.com/swlh/modularization-by-feature-and-layer-with-android-architecture-components-80bf317d737.

19. *Guide to App Architecture* [online]. Google Inc. [visited on 2020-04-13]. Available from: https://developer.android.com/jetpack/docs/guide.

20. SALEH, Hazem. *MVVM architecture, ViewModel and LiveData (Part 1)* [online] [visited on 2020-04-29]. Available from: https://proandroiddev.com/mvvm-architecture-viewmodel-and-livedata-part-1-604f50cda1.

21. *Material Design* [online]. Google Inc., 2020 [visited on 2020-03-13]. Available from: https://material.io.

22. *Maps SDK for Android* [online]. Google Inc., 2020 [visited on 2020-03-14]. Available from: https://developers.google.com/maps/documentation/android-sdk/intro.

23. *Introduction to animations* [online]. Google Inc., 2020 [visited on 2020-03-13]. Available from: https://developer.android.com/training/animation/overview.

24. *Firebase* [online]. Google Inc. [visited on 2020-03-11]. Available from: https://firebase.google.com.

25. *Get Started with Firebase Authentication on Android* [online]. Google Inc. [visited on 2020-04-06]. Available from: https://firebase.google.com/docs/auth/android/start.

26. *Choose a Database: Cloud Firestore or Realtime Database* [online]. Google Inc. [visited on 2020-04-06]. Available from: https://firebase.google.com/docs/database/rtdb-vs-firestore.

27. *Cloud Storage* [online]. Google Inc. [visited on 2020-04-29]. Available from: https://firebase.google.com/docs/storage.

28. *Cloud Functions for Firebase* [online]. Google Inc. [visited on 2020-04-29]. Available from: https://firebase.google.com/docs/functions.

29. *Firebase Cloud Messaging* [online]. Google Inc. [visited on 2020-04-29]. Available from: https://firebase.google.com/docs/cloud-messaging.

30. *Google Maps Platform* [online]. Google Inc., 2020 [visited on 2020-03-14]. Available from: https://cloud.google.com/maps-platform.

31. *Koin: What is KOIN?* [online] [visited on 2020-04-29]. Available from: https://start.insert-koin.io/%5C#/.

32. *About Glide* [online]. Mountain View: Bump Technologies, 2020 [visited on 2020-03-10]. Available from: https://bumptech.github.io/glide/.

33. *Announcing Architecture Components 1.0 Stable* [online]. Google Inc. [visited on 2020-04-01]. Available from: https://android-developers.googleblog.com/2017/11/announcing-architecture-components-10.html.

34. *Room Persistence Library* [online]. Google Inc. [visited on 2020-04-01]. Available from: https://developer.android.com/topic/libraries/architecture/room.

35. *ConstraintLayout* [online]. Google Inc. [visited on 2020-04-01]. Available from: https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout?hl=fr.

36. *Create a List with RecyclerView* [online]. Google Inc. [visited on 2020-04-29]. Available from: https://developer.android.com/guide/topics/ui/layout/recyclerview.

37. *Kotlin 1.3 Released with Coroutines* [online]. JetBrains s.r.o. [visited on 2020-04-01]. Available from: https://blog.jetbrains.com/kotlin/2018/10/kotlin-1-3/.

38. *Android 6.0 Changes* [online]. Google Inc. [visited on 2020-04-01]. Available from: https://developer.android.com/about/versions/marshmallow/android-6.0-changes.

39. *Manifest permission* [online]. Google Inc. [visited on 2020-04-30]. Available from: https://developer.android.com/reference/android/Manifest.permission.

40. BIRCH, Joe. *Exploring the new Android Permissions Model* [online] [visited on 2020-04-29]. Available from: https://labs.ribot.co.uk/exploring-the-new-android-permissions-model-ba1d5d6c0610.

41. *Test your app* [online]. Google Inc., 2020 [visited on 2020-04-14]. Available from: https://developer.android.com/studio/test.

42. *Fundamentals of Testing* [online]. Google Inc. [visited on 2020-04-24]. Available from: https://developer.android.com/training/testing/fundamentals.

43. *Espresso* [online]. Google Inc. [visited on 2020-04-24]. Available from: https://developer.android.com/training/testing/espresso.

44. *Robolectric* [online]. © 2010–2017 [visited on 2020-04-24]. Available from: http://robolectric.org/.

45. *Google Analytics* [online]. Google Inc. [visited on 2020-04-24]. Available from: https://firebase.google.com/docs/analytics.

46. *Firebase Crashlytics* [online]. Google Inc. [visited on 2020-04-24]. Available from: https://firebase.google.com/docs/crashlytics.

47. *Firebase Test Lab* [online]. Google Inc. [visited on 2020-04-24]. Available from: https://firebase.google.com/docs/test-lab.

48. *Firebase Performance Monitoring* [online]. Google Inc. [visited on 2020-04-24]. Available from: https://firebase.google.com/docs/perf-mon.

49. *Measures adopted by the Czech Government against coronavirus* [online]. Government of the Czech Republic, © 2009–2020 [visited on 2020-04-27]. Available from: https://www.vlada.cz/en/media-centrum/aktualne/measures-adopted-by-the-czech-government-against-coronavirus-180545/.

50. *Mobile Operating System Market Share Worldwide* [online]. Statcounter, © 1999–2020 [visited on 2020-04-24]. Available from: https://gs.statcounter.com/os-market-share/mobile/worldwide.

APPENDIX A

Acronyms

AI Artificial Intelligence

API Application Programming Interface

DAO Data Access Object

DI Dependency Injection

FAB Floating Action Button

GPS Global Positioning System

IDE Integrated Development Environment

IoT Internet of Things

JSON JavaScript Object Notation

JVM Java Virtual Machine

MVC Model View Controller

MVP Model View Presenter

MVVM Model View ViewModel

OS Operating System

POJO Plain Old Java Object

SDK Software Development Kit

SMS Short Message Service

SQL Structured Query Language

UI User Interface

UML Unified Modeling Language

APPENDIX B

Contents of Enclosed SD Card

- `readme.md` ..... the file with SD card contents description in MD format
- `apk` ..... the directory with executable APK file
- `src` ..... the directory of source codes
  - `implementation` ..... the directory of implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the directory of the thesis text
  - `thesis.pdf` ..... the thesis text in PDF format
- `samples` ..... the directory with application samples
  - `screenshots` ..... the directory with application screenshots
  - `sample.mov` ..... the application sample video in MOV format
- `test` ..... the directory of usability tests resources
  - `safety_cs.pdf` ..... the application overview in PDF format