Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Serhii Konar**
Studium:	I1900796
Studijní program:	B0688A140002 Informační management
Studijní obor:	Information Management
Název bakalářské práce:	**Personalized assistant in emergency situations**
Název bakalářské práce AJ:	Personalized assistant in emergency situations

Cíl, metody, literatura, předpoklady:

The goal of the thesis is to conduct a research on navigating systems and develop a mobile application to assist in emergency situations caused by natural hazard.

A-GPS vs GPS. (n.d.). Diffen. Retrieved August 19, 2021 API. (n.d.). Mapbox. Retrieved August 19, 2021 Google Maps Platform Documentation |. (n.d.). Google Developers. Retrieved August 19, 2021 Hildenbrand, J. (2020, November 10). How does GPS work on my phone? Android Central TomTom Developer Portal | Maps APIs and SDKs for Location Applications. (n.d.). TomTom Developer Portal. Retrieved August 19, 2021

Zadávající pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	doc. Ing. Pavel Čech, Ph.D.
Oponent:	Ing. Martina Husáková, Ph.D.
Datum zadání závěrečné práce:	9.9.2021

University of Hradec Králové

Faculty of Informatics and Management

Personalized assistant in emergency situations

Bachelor's/Master's Thesis

Author: Serhii Konar

Branch of Study: Information Management

Supervisor: Pavel Čech

Hradec Kralove August, 2022

Declaration:

I declare I wrote the Bachelor's/Master's thesis myself, using only the listed bibliography.

In Hradec Králové, August 2022 Signature
Serhii Konar

Annotation

English: With the advent of new technologies, it is necessary to solve problems in modern ways. The bachelor thesis deals with implementation and design of an Emergency application for Android operating systems. The final version of the app is developed using common as well as modern development technologies and practices. The application notifies users if the tsunami occured and helps to find the nearest shelters. The thesis contains a review of existing applications, methodologies, design and app architecture and usability testing. Also, the ideas of future development are covered at the end of the thesis.

Czech: S nástupem nových technologií je potřeba řešit problémy moderními způsoby. Bakalářská práce se zabývá implementací a návrhem nouzové aplikace pro operační systémy Android. Finální verze aplikace je vyvíjena pomocí běžných i moderních vývojových technologií a postupů. Aplikace upozorní uživatele, zda došlo k tsunami, a pomůže najít nejbližší úkryty. Práce obsahuje přehled existujících aplikací, metodik, designu a architektury aplikací a testování použitelnosti. V závěru práce jsou také uvedeny představy budoucího vývoje.

Acknowledgements

I would like to thank my supervisor of my Bachelor thesis Pavel Čech for help and assistance. Also I am glad that another student wished to help to develop the backend part of the app.

Content

1. Introduction

Humans live in a modern society where new technologies are being developed every day. That is why scientists and engineers need to find out modern approaches for different problems. Every day thousands of people die due to natural disasters, man-made disasters or health related reasons. We can not reduce the number of deaths to zero yet, but at least we can reduce the number of deaths caused by natural disasters. For this reason some developers have already developed mobile applications to help people in emergency situations.

Lack of information in emergency situations

Many people do not know how to react in emergency situations. One of the reasons is lack of information or disinformation. In the past 10 years there were tsunamis that killed a lot of civilians. The tsunami in Japan in 2011 killed about 18,500, although other estimates gave a final toll of at least 20,000. (*Japan Earthquake and Tsunami of 2011 - Aftermath of the Disaster*, 2011) Sometimes it is not enough to warn about tsunamis using sirens on the streets. Moreover, it will lead people to panic, but the app could send more specific instructions about what to do, people will have enough information about nearest shelters.

First thing that comes to mind is to notify the users about an imminent disaster. Once the users were notified they could decide what to do next. Also, developing such apps is important because it will lead us to better or innovative solutions in future. There are a lot of dangerous areas where people live: near volcanoes, the ocean, etc. This group of people live in risky zones and one day the hazzard can occur. To minimize the number of victims it is necessary to follow the instructions that the emergency app gives you.

Similar apps

There are a lot of similar apps. One of them is **Earthquake & Tsunami Alerts (iOS only)**. The application's main screen is a live map that shows earthquakes. It is designed following design principles with great user experience. The application receives data about earthquakes from various sources. In case of an approaching tsunami or earthquake all the users will receive a notification and a loud siren will play. The app saves quake data such as location, magnitude, depth. Also, it could be a great educational tool.

(*Earthquake & Tsunami Alerts*, 2022)

Hazards - Red Cross is another emergency app developed by Red Cross organization in New Zealand. It has almost the same features as **Earthquake & Tsunami Alerts.** However, the last one is supported by both mobile platforms (Android/iOS). And also, it send the notification according to your current location.

(*Hazards - Red Cross - Apps on Google Play*, 2021)

Air alert is another app that could be kind of an emergency app. Unlike the previous examples, this one notifies the user about an air attack in their region. In case of danger the user will hear the siren as in other apps.
(*Air alert*, 2021)

The apps do not support navigation to a shelter.

Goal
Keeping in mind those examples, we are going to create our own app, conducting features from similar apps. The idea is to create an Android application and connect it to a server to work with useful data related to tsunamis. The application must notify the user in case of an approaching tsunami. Using web-application we are able to create a simulation of the tsunami by setting up the origin point and danger area and then send emergency notification to a user.

2. Methodology

Programming language

To build an Android application the programming language should be selected. We have the possibility to select between Kotlin and Java. I decided to code in Kotlin, because it has a lot of new features over Java. Moreover, Kotlin works on JVM(Java Virtual Machine). Coding in Kotlin will make our experience awesome, because we can use such features as Data classes, extension functions or at least write less lines of code. The app is built and supported for Android 8 and latest versions.

Asynchronous requests

During the development we are calling different HTTP (Hypertext Transfer Protocol) requests in asynchronous way and then we push the received data to our UI (User-Interface) thread. That makes our app more responsive in terms of using the app.

Tools

Developers do not like to code one thing many times. That is why in our work we use different libraries. We used Retrofit to call HTTP requests. Since the main feature of the map is navigation, we need to integrate a map to our map. There are a lot of different solutions, but I have decided to use Google Maps for the app, because it has a big popularity over the Android community. Also, it has a lot of useful features that would definitely help us to build our emergency app.

Enterprise Architect

This program will be used to create an UML diagram according to rules of OOP (Object-oriented programming). It will be much more convenient to build the architecture in the application if we think about it before coding. Also it is very helpful, because the developer could see the relations between classes and properties on the scheme in easy way.

Firebase

Firebase is a tool developed by Google. It combines cloud functions and cloud technologies for your application. Firebase can be used as a backend of the app. It also supports Cloud Messaging, so we are able to send notifications to the user. Firebase plays the role of a backend that could provide us with some data.

Architecture

Creating an Android app we could select any architecture. Some of the architectures might be more suitable for different purposes. Mostly it depends on the developer's view. Our app is designed according to MVVM (Model-View-ViewModel) architecture to separate business logic. Also, the principles of Clean Architecture are strictly followed. Comparing MVVM to other patterns it seems to be the most testable due to following single-responsibility principle.

Pattern	Dependency on Android API	XML Complexity	Unit Testability	Follow Modular and single responsibility principle
MVC	High	Low	Difficult	No
MVP	Low	Low	Good	Yes
MVVM	Low or No dependency	Medium to High	Best	Yes

(Architecture comparison, 2022)

Literature resources
To build the app the developer should read a lot of different articles related not even to programming, but also we need to know more about tsunamis and other hazards. One of the resources that was used is ScienceDirect. It is a very popular web-site where “smart people” could publish interesting articles related to science or research. It helped to find similar works and compare them. Also, the most common resource of information is official Android documentation by Google where it is possible to find out anything you need for your application. Reading the documentation, the developer will delve into Android deeper.

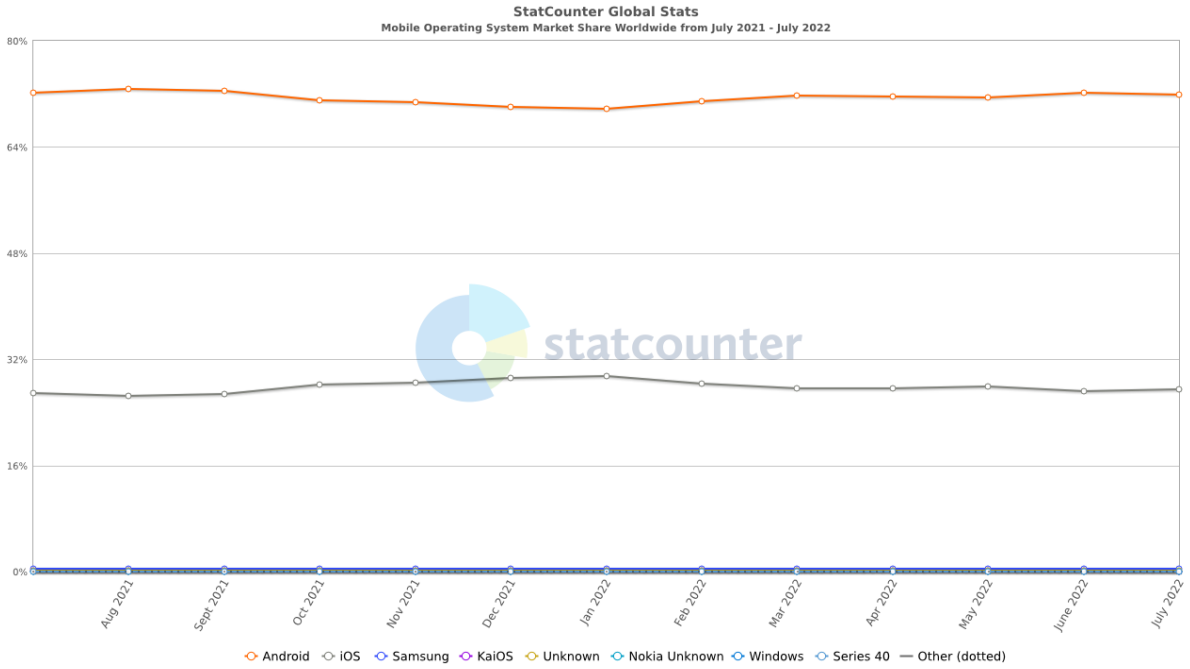
Cooperation with others
To release a better product developers usually need to have other developers or even a team. That is why I cooperated with another student, who helped me to build a server part.

1.1 Basic definitions and classifications of mobile apps

A **mobile application** is a program created to run on a mobile device such as a phone, tablet, or watch. Mobile applications often differ from desktop applications which are designed to run on desktop computers, and web applications which run in mobile web browsers rather than directly on the mobile device. (Techopedia, 2020)

Smartphones usually have an operating system (OS). The most popular OS for mobile devices are: iOS, Android, Windows Phone. The chart below show the market share of operating systems for smartphones.Currently 72 percent of devices use Android as operating

system. The devices could be smartphones, tablets, smart TV, smart watches. De



(*Mobile Operating System Market Share Worldwide*, 2022)

Developing mobile apps we should keep in mind people mostly use Android but 27% of people prefer iOS. The best applications from popular developers are supported on both platforms. The best applications are more accessible. That is why developers should also think about the minority that prefers other OS.

There are some basic types of mobile app if we categorize by approach used to code the app:

1. **Native apps** are created for one specific OS
2. **Web-apps** are responsive web sites which can work on any mobile device using browser
3. **Cross platform apps** are created for a variety of platforms in one specific environment
4. **Hybrid apps** are combinations of native and web app. It is wrapped within a native app. It gives the ability to make an app being downloaded from PlayMarket or AppStore.

Native apps (Valdellon, 2020)
They are built for a specific platform. Thus, we can not run Android native apps on iOS devices. For specific platforms, programmers must code using native language. **Android** native apps are coded in **Kotlin** or **Java, iOS** native apps are coded in **Swift** or **Objective-C**

Native apps have the advantage of being faster and more reliable in terms of performance due to their exceptional focus. They tend to use device resources more efficiently than other types of mobile apps. Native user interface is used in native apps and provides users with a more streamlined customer experience.

Native apps connect with the device's hardware directly. That means native apps have access to a wide range of device features such as Bluetooth, phonebook contacts or NFC. On the other hand, it is impossible to run the native Android app on another platform. In this case

you can hire another developer to create the same app for another platform. This drives up costs. Also you need to maintain and update the codebase for each version.

So, writing native apps can give you the possibility to use all the advantages of Android SDK on different devices. Android is an open source project that means different Android Manufacturers may use its own firmware for Android. Sometimes you need to think about how to implement one solution on different devices. For example, you can think about localizations to create different translations for different regions. Writing code in native environment supports such localizations and even more, you can create different layouts or UI for devices with different screen sizes.

Web-apps

Web apps can behave as native apps, but the main difference is that web-apps are accessed via web browser. Usually it is a responsive web-site that adapt its user interface to the user's device. To create such application developers should use HTML, CSS, JavaScript, TypeScript, React. In this approach, developers create a web-site instead of coding many native apps. This cuts down costs. Also, to run the app you do not need to download files, so the app will not use your device memory.

However, to use the app you need to maintain Internet connection. The web app can't work in offline mode. Also, web apps are dependent on web browsers which are installed on smartphones. Some features are not supported in all browsers. That is why some functionality may differ from browsers, possibly giving users varying experiences.

The worst thing is that using such stack you can not directly access device memory or use all features of the device.

Cross platform apps

That kind of app gives you an opportunity to write code once and run on different platforms.
They are usually coded in JavaScript, React, Flutter. Programmers do not need to create different apps for platforms. Thus, the costs of development can be reduced.

Although, the performance of cross platform apps are lower than native apps. In some specific cases you can not use all the features of the device in a cross platform environment.

Comparing native apps with others, it is clear that they are useful for the majority of cases. But sometimes it could be better to create a web application. That is applicable for small projects or web apps where you need to adapt UI for different screen sizes and save functionality.

Hybrid apps

Are considered to be web apps that look like native apps. They also have an app icon, responsive design, fast performance, or even be able to function offline, but they're really web apps made to look native. To create hybrid apps programmers use Objective C, Swift, Kotlin, Java, HTML5, etc.
Building a hybrid app is considered to be faster and more economical than a native app
A hybrid app can be the minimum viable product. In some situations it is a way to prove the viability of building a native app. They also load quickly. It is ideal for use in countries with slower internet connection, and provide users with a consistent user experience. There is much less code to maintain due to using single code base

However, Hybrid apps might lack in power or speed.

In most cases the approach of development may be selected by the customer. In 2022, there are customers that are looking for developers to develop only native apps or only web-apps adapted for different devices. There several reasons why people choose between native app or web app. One of them is, different price for development and amount of time spent for development. Developing two native apps for two platforms may take much more time than writing a web-app. In our case developing a native app is mandatory and only one possible solution, because we want to work with geolocation and decode it.

1.2 Android development features

Developing on Android, the developer should know about specifics and features of different devices. They may differ from manufacturer. This brings some complexity in development.

Variety of Android devices

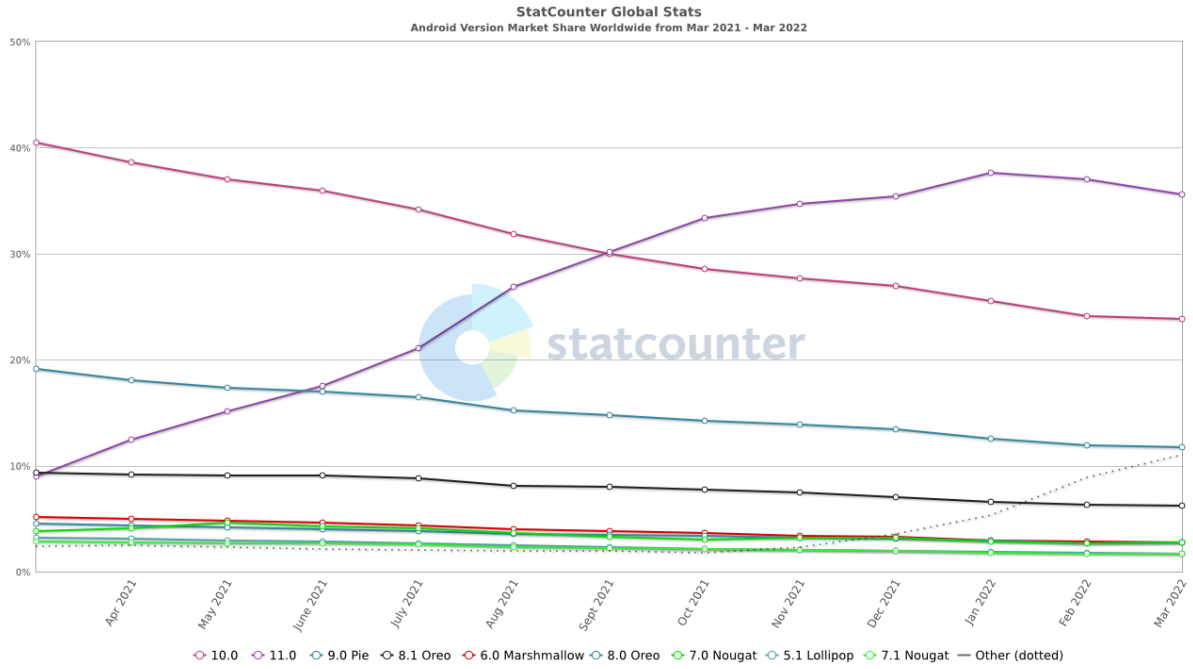
Developing a mobile application for android is not an easy task, because it is important to know a few things. Firstly, this is a large fragmentation of devices. This is an advantage for users: they can choose a phone for every taste and for any technical requirements. Front camera may not be installed in smartphones. Sim Cards can be as many as you like. Two screens may be present, for example, an additional one on the back
Imagine a situation, you need to use a camera in the App. Firstly, the programmer should verify if the device has a camera or not. If the user does not have a camera, they are notified that this feature is not available without a camera.

Also developers should consider different screen sizes. The app should be optimized for small screens and tablets.

Different versions of Android

There are a variety of Android versions. This make developers to think about some important points during the development:

- During the development process, it is necessary to take into account the specificity of displaying the user interface on different versions of Android. For example, system controls may differ from Android versions and skins of the same Android version;
- Different versions have different features and support different logic. For example, before version of Android 6.0 (Marshmallow), applications did not have to request permission separately (access to the camera, microphone, and so on), they were listed in Google Play and it was supposed that the user was familiar with them before downloading. After Android 6.0, each permission must be requested separately at the time the application is running. Accordingly, if you do not implement logic options for different Android versions when developing an application, it will not work properly in all versions;
- Libraries and software methods are updating: some of them are considered deprecated and need to be replaced with newer lines of code. So, developers have a choice: support only latest Android versions or to allow as many users as possible to install a mobile application;



(*Android Versions Market Share*, 2022)

1.3 Architecture of the app

You need to take into account the architecture and the structure of the app. Unlike iOS, where applications are architecturally a single whole, in Android they are assembled from logically independent and separate parts - activities and fragments.

This approach was created precisely in order to ensure the operation of applications on absolutely any mobile, including those with a very small amount of RAM and very weak processors. If the parts of the application are independent, any of them can be thrown out of memory at the right time and not spend precious resources on maintaining its life cycle.

There are a lot of architectural patterns that developers may apply to build an architecture.

Google's recommended programming language for building android apps is currently **Kotlin**, not Java. The difference between them is significantly smaller than between Objective-C and Swift for iOS, but they are still slightly different approaches to development.

1.4 Specifics of Android app development

As well as other platforms Android has its own specifics for development. Creating mobile applications for Android has the following specifics:

- Android is the most popular operating system in the world. As a consequence, the diversification of the devices it manages is huge. In the process of creating an application, you need to make sure that the application supports the vast majority of gadgets of the target audience.
- At the development stage, pay attention that UI / UX should take into account not only the different sizes of devices, but also the work in multi-window mode, and the pixel density of screens: thin fonts on low-quality displays will be distorted or disappear altogether.

- The number of current versions of Android in use is many. When creating applications, you need to consider all of those that your target audience uses.
- During the development process, when designing the interface, it is necessary to be guided by the concept of Material Design (see https://material.io/design).
- Google's recommended programming language for Android is Kotlin.

1.5 How does the smartphone know your location? (Ionescu, 2010)

Receiver in the smartphone provides location and time information to a software, using radio waves between satellites. The user does not have to send any data to the satellite for maintaining GPS tracking. The user is only for receiving and processing the data from four or more satellites that are dedicated for geolocation use.

GPS is precise, but it's slow and uses a lot of power on both ends. Thus, when user uses navigator you need a bit more energy to maintain the smartphone’s work.
Atomic clock is installed in each satellite and the satellite sends a time-coded signal on a specific frequency. Receiver on smartphone starts gathering data from the satellites with the strongest signals after determining visible satellites. GPS data is slow, because satellites run on rechargeable batteries, and sending a fast signal thousands of miles would require a lot of energy — so it can take up to several seconds to get precise location.

To triangulate where the user is located and what time it is phone's GPS receiver uses the data from satellite signals. Triangulation means that four satellite are are required for GPS to work. To determine altitude, the smartphone receives the fourth signal. So it is enough to receive three signals to get the user's location on the map.

2. Technologies and tool for Android development

There are a lot of different ways developers could build the app. Thousands of tools, APIs (Application programming interface), libraries or even languages could be used for native or cross-platform development. For native android apps a developer should choose between Kotlin and Java.

2.1 Selecting the programming language

Programmers can code native Android applications using Kotlin or Java. During the last years Kotlin gained its popularity. Google has listed Kotlin as the primary programming language for Android devices since 2017. However, older applications were usually written in Java and still work on the latest versions of Android.

2.2 Features of Kotlin

Kotlin is a general purpose, free, open source, statically typed "pragmatic" programming language initially designed for the JVM (Java Virtual Machine) and Android that combines object-oriented and functional programming features.

Kotlin Supports Full Java Interoperability

One of the major as well as the best features of Android Kotlin is its deep interoperability with Java. This feature has attracted many Java developers as well as Android app developers to coding in Kotlin.

Developers from JetBrains made the code to run on JVM (Java Virtual Machine). That is why Kotlin supports Java libraries as well as Kotlin libraries , providing full Java interoperability.

Both the languages co-exist, so this makes it easier for developers to be productive. Developers can easily compile one Android project in both languages (Java and Kotlin).

This will allow the developer to switch the programming language instead of changing the codes. So, if you need access to a Kotlin method from a Java class or vice versa, you can do it easily.

Data classes in Kotlin

Typically, a data class in Java contains lots of boilerplate code which developers have to skip in order to find out the real use of that class.

But now in Kotlin, Android developers can write the equivalent of the same Java code in a simple manner, and with lesser code. Therefore, the data classes in Kotlin are also known to be one of the useful features. The data class in kotlin generates all the code under the hood during the compilation.

Extension function

Extensions of this language are very useful because they allow developers to add methods to classes without making changes to source code of the class

Null safety

This feature of Kotlin is one of the strongest because this language is relatively safe. Moreover, programmers can avoid errors such as “NullPointException” using the **Null Safe function**.

Clean and Compact Syntax

With the help of Kotlin, you get things done with only a few lines of code comparing to Java

In addition, clean syntax and code offer benefits, such as easy maintenance, easy-to-read code, and easy

Standard Library Functions

The benefit of using Kotlin is that it offers standard library functions. This is the most important factor for Android developers.
You can easily make the implementations of higher-order functions with the help of idiomatic patterns, such as let, apply, use, and others, with much ease.

Moreover, it also has multiple utilities to work with char and string sequences. Also, Kotlin’s JDK classes can be useful in working with Input and Output operations, files, and threading with more convenience.

Coroutines

Asynchronous or non-blocking programming is an important part of the development process. When building server, desktop, or mobile applications, it's important to provide a user experience that's not only flexible from a user's point of view, but also scalable as needed.

Kotlin solves this problem in a flexible way. It provides coroutine support at the language level.

In addition to asynchronous programming, coroutines also provide a variety of other possibilities, such as concurrency and actors.

https://www.spaceotechnologies.com/blog/kotlin-features/

Features of Kotlin that Java do not supports:

- Lambda expressions and inline functions (i.e., performant custom control structures)
- Separate interfaces for read-only and mutable collections
- Declaration-site variance & Type projections
- Range expressions
- Type inference for variable and property types
- Coroutines
- Extension functions
- Null-safety
- Primary constructors
- First-class delegation
- Operator overloading
- Companion objects
- Smart casts
- Data classes

- String templates
- Singletons

Java is still supported on Android. Android runs over JVM. That means we are able to run Java apps on Android. Java was the official language of Android until Kotlin came along. Google has supported this language by making it official for the platform. We are choosing Kotlin since it has more features over Java and it is more modern approach to develop for Android.

Why is Kotlin good for Android app development?

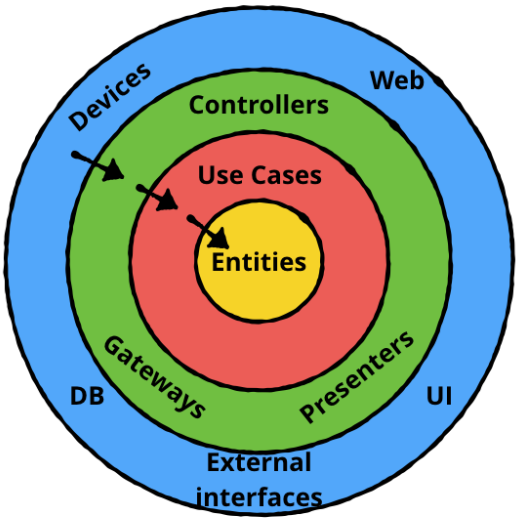
- Kotlin is a quite new programming language that has its own feature that Java does not. And this makes Kotlin the best for developing native Android apps in 2022.
- Google considers Kotlin as the official language of the Android platform.
- Kotlin has a big community. Kotlin was released in 2016 and started to gain popularity.

2.3 Selecting an architecture pattern

There are a lot of different architecture patterns that could be used for building an Android app. All of them exist to reduce boilerplate code, make your code scalable and understandable. It is important to keep code clean and scalable. For this reason architecture patterns exist. So what is clean architecture and how can we apply it in our project?

2.3.1 Clean architecture

This is one the most important software design philosophies. The majority of architectures are built following its principles. Generally, clean architecture separates business logic and UI



(*Clean Architecture*, 2019)

Firstly, we need to understand the basic definitions to apply such approach:

- **Entities** - Encapsulate the most important rules for the functioning of the enterprise level. Entities can be an object with methods or a collection of data structures and functions.
- **Use Cases** - organize the flow of data to and from objects.

- **Controllers, Gateways, Presenters** - it's all a set of adapters that most efficiently convert data from **Use Cases** and objects to pass to the top layer (usually the user interface).
- **UI, External Interfaces, DB, Web, Devices** - the outer layer of the architecture, usually consisting of elements such as user interfaces, front-ends, databases, and web frameworks.

Understanding those definitions can help us to structure the project

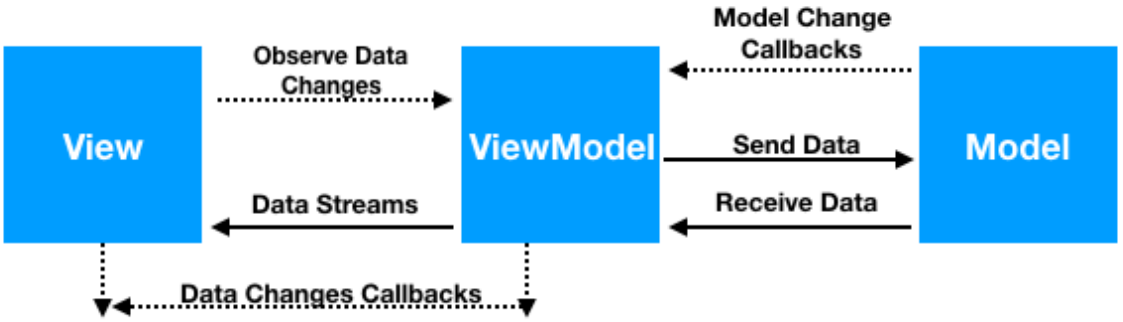
How to implement the structure?

A typical Android project usually requires a separation of concepts between user interface, operation logic, and data model. We need to create three different packages or modules:

- **Domain** - contains definitions of application logic, server data model, abstract definition of repositories, and definition of use cases. This is a simple, pure kotlin module (android independent).
- **Data** - contains an implementation of the abstract definitions of the domain layer. Can be reused by any application without modification. It contains data source repositories and implementations, a database definition and its DAO, network API definitions, some conversion tools to convert network API models to database models and vice versa.
- **App or Presenter layer** - it depends on Android and contains fragments, view models, adapters, activities and UI parts.

2.3.2 MVVM architecture (Guide to App Architecture |, 2020)

Google has its own code-libraries to help Android developers to reduce the boilerplate code and for better structuring and separation of business logic.
MVVM stands for Model-View-ViewModel, that is why this pattern consists of three parts. Applying the pattern we should consider the next parts of logic: Model, View and ViewModel. **Model** is the business data layer and is not associated with any particular graphical representation. In Android, according to the "clean" architecture, the model can contain a database, a repository, and a business logic class. The picture below describes the interaction between different components. **View** contains a structural definition of what users will get on their screens. You can put static and dynamic content here. There may not be any application logic here. For our case, the view can have an activity or a fragment. Only UI (user interface) can be here. **ViewModel** binds the model and the view. Responsible for managing data links and possible conversions. This is where the binding comes in. In Android, we don't worry about this because we can use the AndroidViewModel or ViewModel class directly. (*ViewModel Overview* |, 2020)

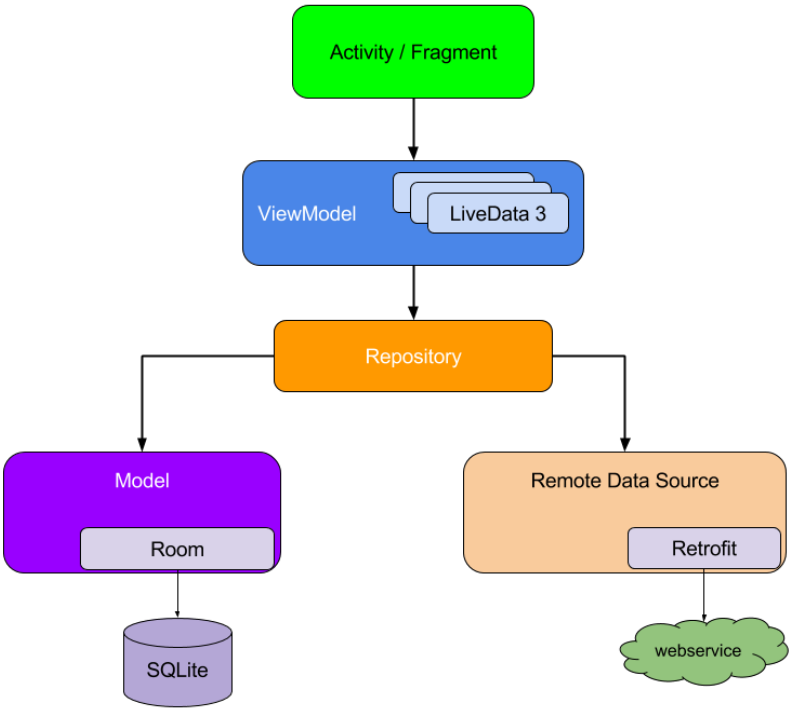


(*MVVM Architecture*, 2022)

How to implement the MVVM pattern (MVVM Architecture - Android Tutorial for Beginners - Step by Step Guide, 2020)

To implement the MVVM pattern, it is important to start with components that need another component to work.

We can see the class diagram below that represents the **MVVM architecture**. And this means that the View (Activity/Fragment) knows about the View-Model, and not vice versa, and the View Model knows about the Model, and not vice versa. That is, the view will have a relationship with the ViewModel, and the ViewModel will have a relationship with the Model. Strictly in that order, nothing else. This architecture makes the application easy to maintain and test.



(*App Architecture*, 2020)

In addition, we can fetch data using databases and API-calls in a more convenient way from Repository.

Advantages of MVVM Pattern:

- A lot of extra effort is minimized as it allows reviewing the complete setup at the compilation stage.
- In this model, the user is allowed to write custom XML attributes by using Binding adapters.
- Helps to separate the code

MVP pattern

With the development of the visual programming environment and the introduction of widgets that encapsulate the rendering and processing of user input, there is no need to create a separate controller class. But developers still need to separate logic from presentation, only now at a higher level of abstraction. Because it turns out that if you create a form with multiple custom elements, it also contains both interface and data logic. The MVP pattern describes how to separate the UI from interface logic (what happens when interacting with widgets) and from data (what data to display on the screen).

- **Model**
  This is your application's data, the logic behind getting it and saving it. It is often based on a database or results from web services. In some cases, it will need to be adapted, changed, or extended before being used in the View.
- **View**
  Usually it is a form with widgets. The user can interact with its elements, but when some widget event affects the logic of the interface, the View will send it to the presenter.

- **Presenter**

The Presenter contains all the UI logic and is responsible for keeping the Model and View in sync. When the view notifies the presenter that the user has done something (such as pressing a button), the presenter decides to update the model and synchronizes all changes between the model and the view.
Speaking about different architectural patterns we can not say that MVP is definitely better than MVVM or vise-versa. Both of them have their advantages and disadvantages. Developers select that pattern which is the most suitable to solve concrete problems.

Creating a user interface without using templates

How would you build a user interface (UI) without using the above patterns? They would take layout , add widgets to it, and write logic in code. Such code, which describes the logic of the View, is tightly coupled to the user interface, since it directly interacts with the elements on the screen. This is a good but straight forward approach. It is applicable only for very simple interfaces. When the logic gets more complex, maintaining such a UI can become a nightmare!

The root of the problem is that building the UI in this way violates the Single Responsibility Principle, which says, "A class should have only one reason to change." If a UI component contains code for display, logic, and data, then it has several reasons to change. For example, if you want to change the type of a custom element that is used to display data, then the change should not affect the logic. However, since the logic is so tightly coupled to the controls, it will also need to be changed. If you will not use architecture patterns, you will definitely face complexity **of support.** Changes in the UI, logic or data will most likely lead to changes in other parts. Therefore, it is much more difficult to make changes, which makes it difficult to maintain. In addition, you will suffer from hard testability. Application logic and data can be written in such a way that each component can be tested separately. However, UI-related code does not lend itself well to unit testing because it often requires user interaction to run logic in the UI. In addition, any visualization often requires human evaluation that everything “looks right ". Note that there are solutions for automating user interface testing. However, they only simulate user interaction. They are generally more difficult to set up and maintain than unit tests and are most often used in integration testing as they require the entire application to run. If your UI code is mixed with logic and data code, then it becomes much more difficult to reuse it. So, the code and relationships become 0.

Choosing a good architecture pattern and following main principles of Clean Architecture is so important as writing good code. Because applying pattern and following architecture styles will make your code more readable.

In real projects programmers usually combine MVVM architecture pattern and Clean architecture principles. It is a useful practice.

Programmers usually want to make their code cleaner and without bugs. If we want to find out possible bugs we usually write Unit tests. Separating business logic and UI representation makes your code testable, so you can test a code when you consider to find and avoid a bug. Also, programmers can write Unit tests to test a behavior before writing a code. Thus, you have a test-covered code that improves your application and efficiency many times.

In my opinion, the main advantage is that the MVVM pattern follows the principle of separation of concerns. That makes your code structured.

On the other hand, let's imagine that we want to build an application without applying any architecture and patterns. What problems may it cause? First idea that comes in my mind, the code is not that flexible and scalable. You will suffer from hard coded code which ıı closely related to your UI. Every time you implement a new feature it is much harder to maintain. Also, this is the reason why the development of the app may be delayed or costly to develop. Also, we will not have all the advantages that appear applying a pattern that means our code is less testable and possibly has errors. However, in specific situations you can omit architecture patterns, especially in small projects.

Comparing MVVM and MVP patterns we can firstly highlight that MVVM pattern is more popular and supported by Google. Of course, under the hood they are implemented differently. To set up our MVVM architecture we need to create a **LiveData object** and update it once we receive the data from the API. Also we create some public methods to observe LiveData from the UI layer and update it once we receive new data. Following the principles of Clean Architecture, I decided to use asynchronous threads to prevent freezing the UI thread when we call it from our **ViewModel**

The best practice in 2022 is to create a ViewModel with functions that get data from API or Database asynchronously. Kotlin supports coroutines that can be very useful if we want to make our app more responsive. In comparison with Java, one does not have coroutines, only standard Java libraries for Threading are supported. Running code in different threads makes the app more efficient, because you are not overloading the UI thread (also known as Main thread). We can handle different states receiving the data. For example, data may have states such as **Loading, Error, Data.** Each state may have its own implementation in the UI layer. For example, we can show progress bars until data is not loaded completely. That is why we combine coroutines with our ViewModel which is responsible for updating our UI.

More structuring

Once we have selected an architecture type we need to implement it. For small projects it is usually enough to select an architecture and just follow its principles. But for big projects or startups it is very important to organize your code well. In this case, we should consider how to make your code as simple as possible. Senior Android Developers suggest using Clean Architecture to divide our project into several modules. For better implementation of Clean Architecture we can use **Hilt** or **Dagger2,** the frameworks that could help us to inject dependencies in our code. Moreover it is important to use patterns to simplify your code and make it more readable. It looks very ugly when a big project has some basic classes for UI and these classes contain a lot of different methods to do what a programmer wants to do. Such an approach violates the principle of separation of concerns, it becomes harder to explain what's happening, because your file contains many lines. Definitely it is not a good way of coding.

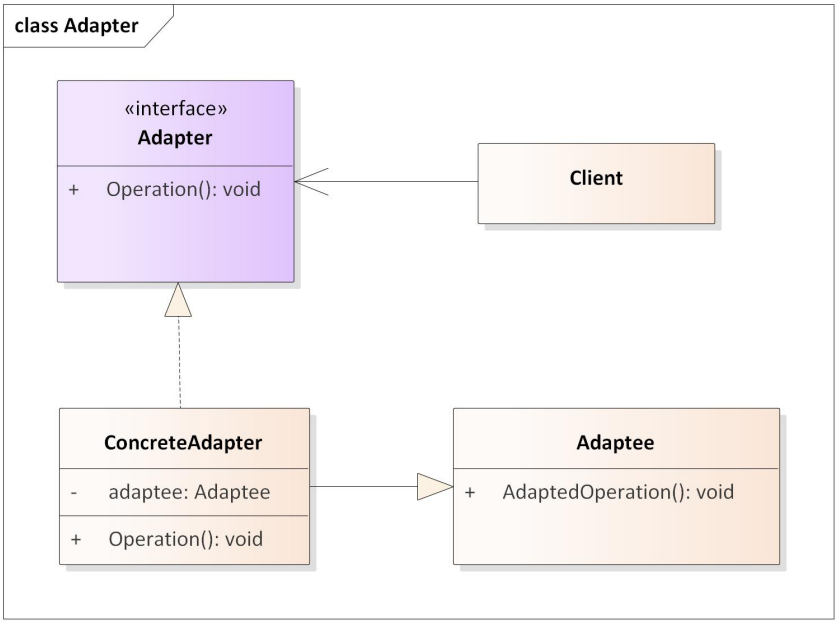
Let's think about which programming patterns we can use additionally to write pretty Kotlin code. Each pattern exists to solve concrete problems. There are many themes. Sometimes people. Patterns are often confused with algorithms, because both concepts describe typical solutions to some well-known problems. But if an algorithm is a clear set of actions, then a

pattern is a high-level description of a solution, the implementation of which may differ in two different programs.

So why do we need to use patterns? It is a proven solution. That means we spend much less time using complete solutions than reinventing the wheel. Patterns make code more standard. So, you don't need to explain what hierarchy you implemented, just say the name of the pattern and we can understand the behavior of classes. The need for patterns appears when people choose a programming language with an insufficient level of abstraction for their project. In this case, patterns are the crutch that gives this language its superpowers.

In my opinion the most interesting pattern is Adapter. This pattern makes it possible to collaborate with two objects of incompatible interfaces. In terms of Android programming it is recommended to use this pattern when you are creating a **RecyclerView**. RecyclerView is just an UI component which is responsible for displaying a list of items. And we need somehow to bind data to our item in the list. Usually the item in the list has the same views and layout, but only the different data is displayed. In this case Adapter pattern is good to apply, because we want to inflate our RecyclerView with meaningful data. So, applying this pattern helps to collaborate with data in our list.

We made our code better.



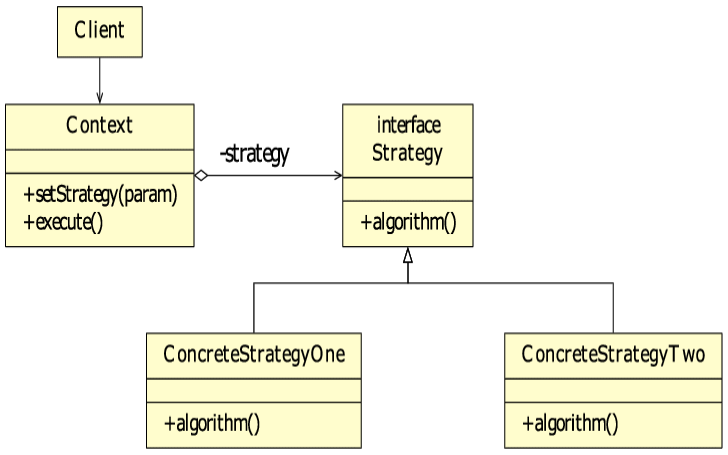
(*Adapter Pattern*, 2022)

I applied this pattern to operate the data set in RecyclerView that is responsible for displaying a list of shelters. The list of shelters will be updated with our objects after the successful completion of the GET request. Then, using the Adapter class we can do valuable changes in our RecyclerView. Adapter class can notify the RecyclerView that one item or item set was changed. After that the RecyclerView will redraw itself to apply changes in UI.

Also I wanted to highlight a pattern which is no less useful. It is **Strategy.** It is a behavioral design pattern that defines a family of similar algorithms and places each of them in its own class, after which the algorithms can be interchanged right at runtime. Imagine we have three

different situations: flood, air alert or earthquake. For these three different situations we need to implement a bit different logic because each shelter has its own purpose.
So we define the interface of Strategy and implement this interface in FloodStrategy, AirAlertStrategy and EarthquakeStartegy. Context holds a reference to one of the strategies and we can execute the strategy in runtime.

Applying the pattern helped us to make different calculations for each situation. Dividing each strategy that realizes an algorithm makes our code readable as well.



(*Strategy Pattern*, 2022)

In project I decided to apply the pattern to divide different algorithms. For different emergency situations we need to operate with data in different ways. For example, if meteorologists warns about flood or tsunami we need to draw a route to shelter which is located high enough, so the tsunami will not reach the point.
The second algorithm is responsible for giving instructions during the air attack. You will be able to find the nearest bomb shelter. And in case of an earthquake need to stay away from buildings so as not to suffer from a damaged building. As we see, different situation requires another algorithm.

3. How a beta-version of Emergency App was created? MVVM structure.

For that reason I divided my app into three different modules: app, data and domain. App is responsible for UI and presentation of data. Data - calls API, do our business logic here and contains package “di” (which stands for Dependency Injection)
Domain - is a more abstract module which contains models, interfaces and use cases.

To create a new module in Android Studio we go to the File -> New -> New Module. From the following templates we should select the Android Library to generate gradle files and be able to work with Android environment.

Once the modules were successfully generated, we can import these modules. In app module we should implement domain and data modules, in data we should implement only domain module and domain module implements nothing. To implement modules we need to modify

our gradle files for each module. Adding this line will implement domain module in our app module:

```
implementation project(path: ':domain')
```

Also, in gradle files we will add some useful libraries to be able to use the in our project. For dependency injection I will use **Hilt.** Also, I need to work with network requests and responses. For this purpose I will use **Retrofit.** To handle JSON responses I want to use the Gson library. For better structuring I created new packages in every module. Each package is responsible for separation of classes according to their functionalities or purpose.

3.1 Dependency Injection

It is a widely used method in programming not only in Android. The dependencies are provided to a class instead of creating the instances of classes itself. Good application architecture could be built by following the principles of dependency injection

The new **Hilt** library defines a standard way to do DI in your application. Hilt provides containers for different Android classes in your project and mages their lifecycles under the hood.
Hilt is built on top of Dagger which is also popular for library dependency injection. Since many Android framework classes are instantiated by the OS itself, there's an associated boilerplate when using Dagger in Android apps. Unlike Dagger, Hilt can be easily integrated with Jetpack libraries and removes the boilerplate code, so you can focus on more relevant things. Hilt automatically generates and provides:

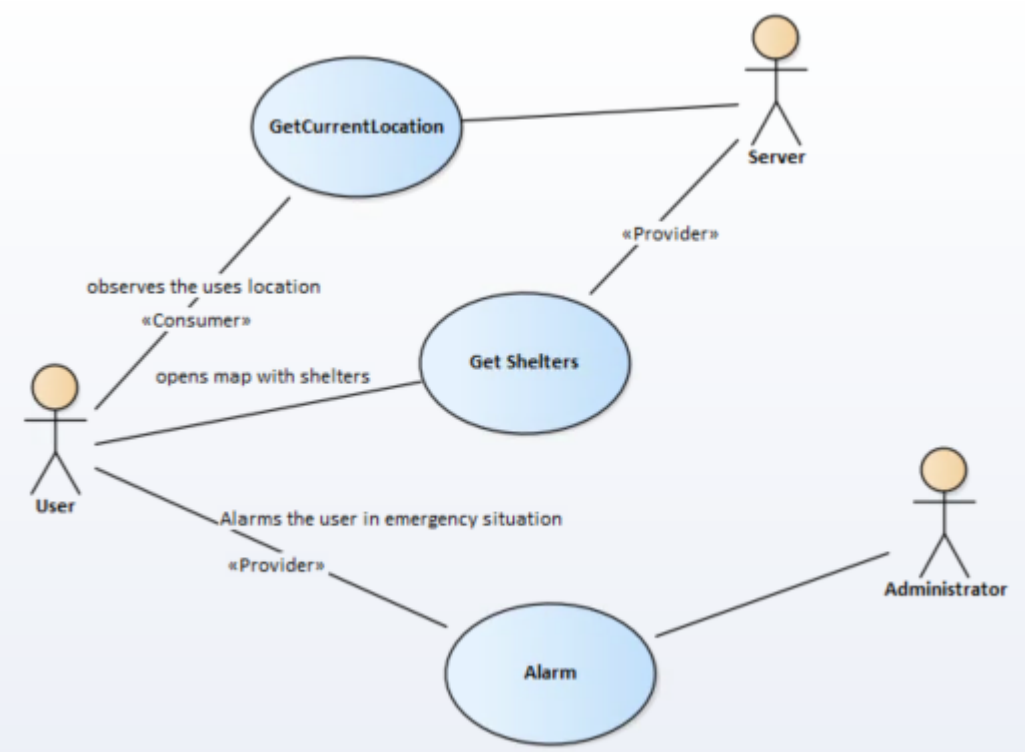
- Components for integrating Android framework classes
- Bindings and qualifiers.
- Scope annotations for the components

Also, Dagger and Hilt can co-exist.
So, I decided to use Hilt in the app, because this library is written specially for Android and it supports all the necessary things.

3.2 Use Cases

To create a scalable application we need to specify our use cases for better understanding of the logic and separation of concerns. Use cases are considered to be as a part of Clean Architecture. Each **usecase** do the certain work. For example, I created a **usecase** for getting the locations of shelters.

Graphical representation of use cases



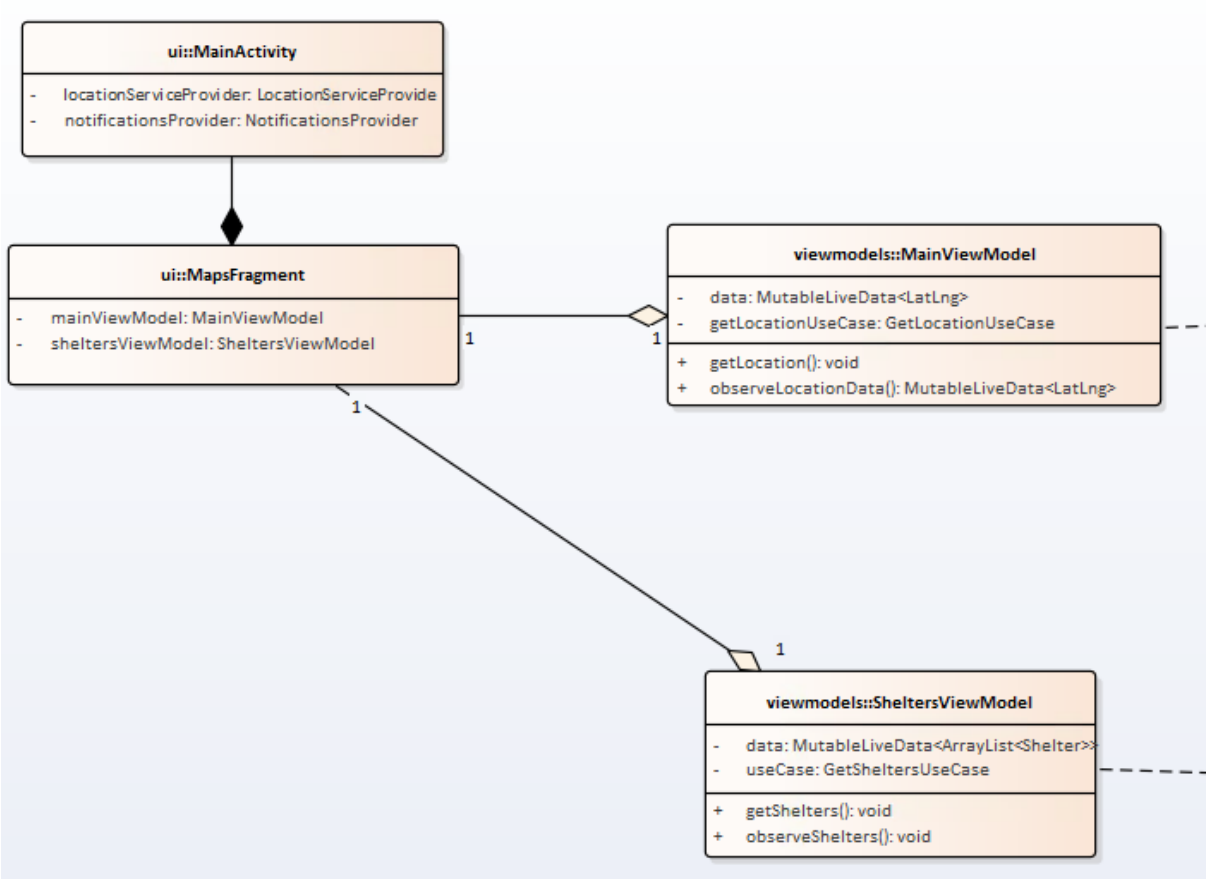
(*Use Cases*, 2022)

For better understanding of the action flow, I created a use case diagram in Enterprise Architect
So, I specified 2 use cases for users. GetCurrentLocationUseCase - this use case obtains the longitude and latitude of user's location. Each UseCase performs specific actions. We assume that **GetCurrentLocation** has some useful functionality. Using this use case, we check if the user has granted permission to access restricted data, in our case its GPS location of user. If the permission was granted, we can work with location. In other cases, we ask users to grant permission. **GetSheltersUseCase** - is used for getting a list of shelters with coordinates. The usecase performs a simple GET request to a server to get data about shelters which was inserted in the DataBase by the Administrator.
Administrator has only **Alarm** use case, which is used for notifying the users about hazzard. Understanding the UseCase diagram gives us a clear vision of our functional part.

UI components and data representation

If we speak about Android app it is crucial to create the correct relationships between Activities, Fragments and other Android components. Using the diagram below we build a simple UI . Our MainActivity holds MapsFragment which shows the user the map and other UI components such as buttons or texts. Using **viewModels** in MapsFragment helps us observe the data and once we fetched the data, we update the UI. For observing the data we have methods **observeShelters()** and **observeLocationData().**

The diagram below represents the UI module of the app, which must be separated from business logic.



(*Emergency App Architecture*, 2022)

Data models

To represent some data, programmers usually create classes. In Kotlin we can use Data Class. Our model consists of pieces of information that can be shown in UI. Each Data class must have at least one attribute.

Example of Data class in Kotlin:

```
data class Shelter(var latLng: LatLng, type: ShelterType) { }
```

The same logic in Java

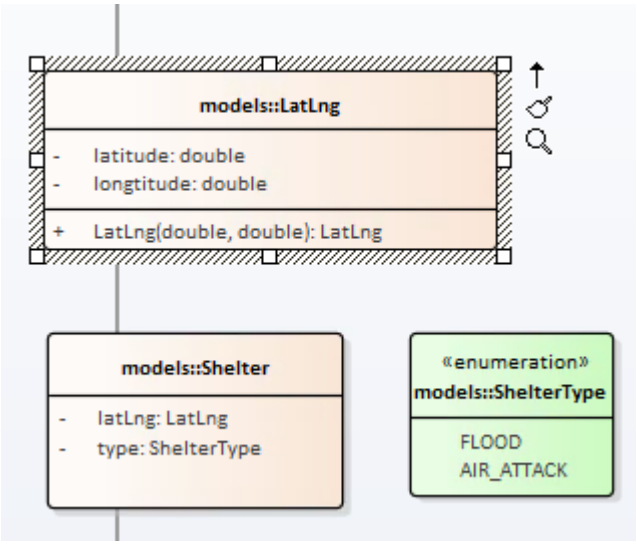
```
class Shelter {
        private LatLng latLng;
        private ShelterType shelterType;

        public Shelter(LatLng latLng, ShelterType shelterType){
                this.latLng = latLng;
                this.shelterType = shelterType
        }
        // Also we should define getters and setters
}
```

We can see that in Kotlin we write much less code than in Java to do the same. That is really awesome. In future we will be mapping our data which was received from the server to our data classes.

In terms of Clean Architecture we should define our models in the **model package** in the **domain module** which is the abstract layer of our application.

In my app I have specified some models:

- LatLng - represents a latitude a longitude
- Shelter - has a LatLng attribute, so it has its own location. Also shelters have a specific type. We can have shelters to rescue from **floods** or air **attacks.**

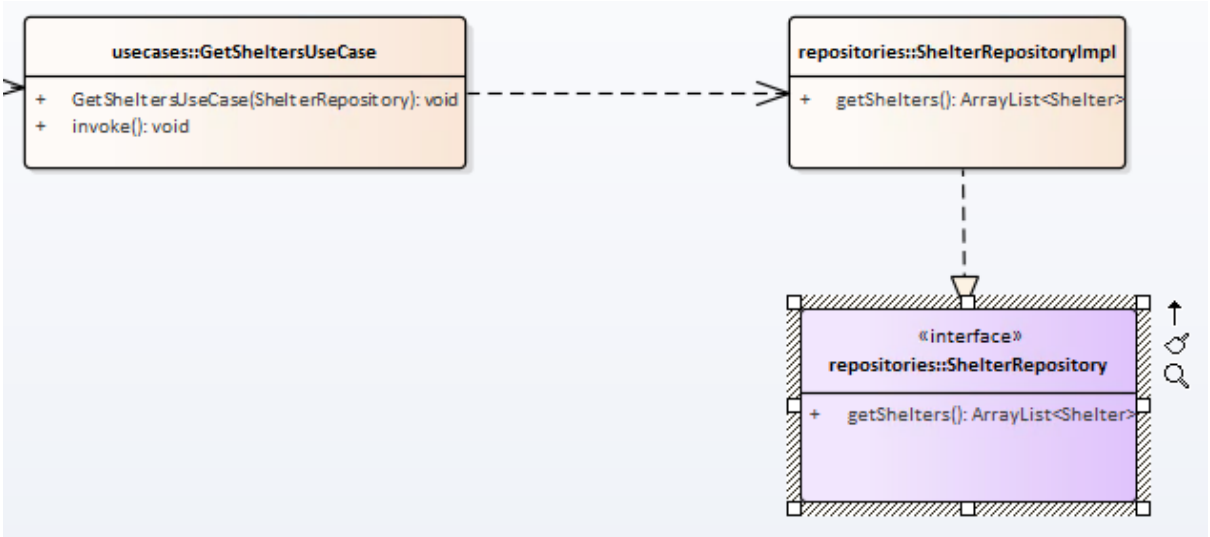


(*Domain Layer*, 2022)

Repositories

According to Clean Architecture, a repository is a kind of abstraction that allows access to entities by using an interface. We create a ShelterRepository in the domain module. Also, we must create the realization of the interface in the data module. We create the realization of the interface in ShelterRepositoryImpl. It must be in the data module because we want to access our server to fetch data using Retrofit

For this reason a created a package which is called **di** (Dependency Injection). In this package we declare classes using **Hilt.** The **di** package consists of classes which are responsible for network requests and RetrofitClient building. So the reason why we implement the interface in the **data** module is that this module gives us access to our backend that provides us useful information.



Services

Service is one of the application's components. They are used to perform long-running tasks in the background thread. In addition, the developer can bind some primitive UI to control behavior of the service. In Android there are three types of services.

Foreground service

Is a type of service that performs an operation which is noticeable for the user. For example, audio streaming will require playing sound via foreground service. Very important thing is that to start foreground service we need to show a notification to the user. The notification shows what is currently running in the foreground. The service does not require user interface, but in cases it could be useful to customize the notification, for example add buttons in notification. This approach can help us to send some commands to the service and the service can handle them.

Modeling

The architecture of the app or it is scratch can be modeled in Enterprise Architect. That way of modeling the architecture is very good. Because you visualize what type of relations and classes you need. Modeling strong scratch and is important for programmers. Of course it is better when the programmer can see the relation between parts of the program. It gives a clear vision what components, libraries and classes programmer should use. I believe that modeling the architecture before writing the code is better approach in software development than building an architecture on the go. Visualizing the architecture is good because Software Engineers could prevent duplication of classes, bugs. Also, the model usually represents the final product. So it is much better to create classes and components which were described in model at first time when coding than modify the file and functionality many times.

3.3 Applying pattern to handle different HTTP responses

Every time when when we call an API to get a list of shelters we expect to receive a result. Of course you expect to fetch result successfully, but the could be the case when Retrofit or other networking library produce an exception due to some reasons, for example device is not connected to the Internet. In this case the user will not receive expected result and the program did not handle an error. As we want to improve user's experience, we can handle the exception and then notify user about Internet connection using UI.

To handle different results and states there is a **sealed class** which is an advantage of Kotlin. Sealed classes represent class hierarchy that gives more interaction over inheritance. All direct subclasses are generated at compile time. That means each instance has limited set of types that is known at compile time. With the help of this feature, we can handle three different states when fetching data: Loading, Success and Error. Each state is an instance of our sealed class, let's name it Result.

When we do a call to api we need to think about doing it asynchronously and handle different states. Our first state is Loading that we return oncle we requested HTTP . And using **try/catch** construction we can handle an error. If the error is occured we return Resource.Error class and provide exception in the constructor. HTTP request is called asynchronously and we do not overload our UI. Once the data fetched successfully we can get that data from UI accessing to class Result.Data<T>.

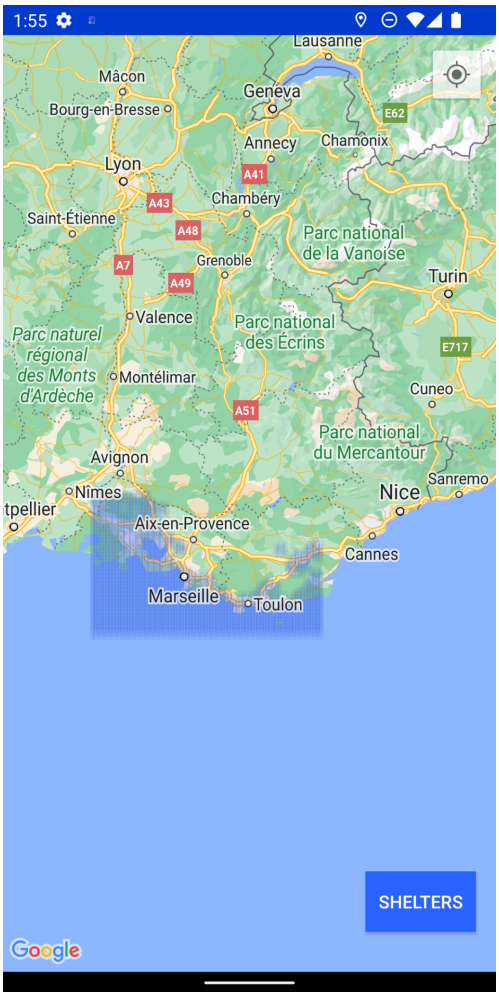
```
sealed class Resource<out T>{
    object Loading: Resource<Nothing>()
    class Error(var exception: Exception): Resource<Exception>()
    class Data<T>(var data: T): Resource<T>()
}
```

(*Response Handling*, 2022)

We can use Kotlin Coroutines to achieve asynchronous work on main thread.

3.4 Integration of Google Maps to our app

The main idea is to draw danger areas on the map using Google Maps API. Maps API provides various methods and features to implement our awesome ideas in the app. To work with Maps API we also should import Maps SDK to access those methods. Also, to set up our environment we should provide an API key. We can receive it from Google Cloud Console. Google Cloud Console is a default tool for developers where they can configure different APIs, not only for Maps, but even for artificial intelligence. But we will focus on Maps API. So what features could be useful for our app? Firstly, we need to add a Google map in our app and then we can configure it. In terms of programming we should create a **GoogleMap** object. As we want to display on the map shelters that means we should put a **marker** on the map. Marker is a pointer that has a specific position on the map. That is why we should provide a latitude and longitude of the shelter. Latitude and longitude are decimal numbers, so in code it should be of type **float**. We can provide our marker with some custom attributes such as title or even change the default marker icon.



(*Map Fragment*, 2022)

As we want to highlight a danger area we need to draw a polygon. It is not much more complicated than adding a marker. Only the difference is that we should provide at least 3 points. In a real situation there could be more points. Those points represent the bounds of the polygon. That is why we also should configure the polygon using latitudes and longitudes.

Also, we can specify some actions when the user clicks on the polygone or marker. That could help us to develop more useful features in our app.

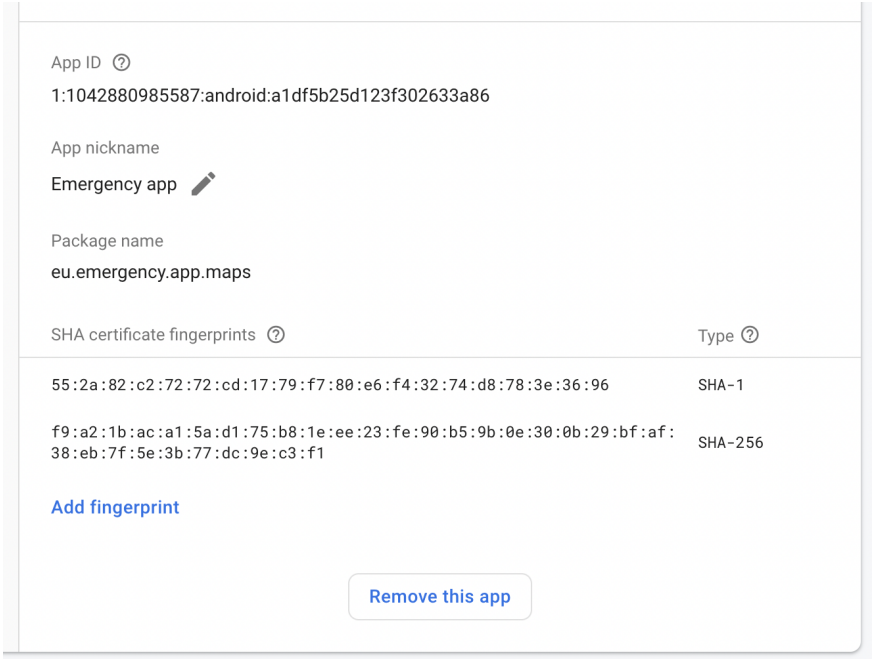
In Conclusion, the main thing on which we should concentrate is our points. They will help us to draw a polygon or put a marker.

Also, we need to use the Directions API which is also developed by Google. The reason is that we want to draw a route and do navigation in our app. This API could help us to configure different routes. For example, evacuating by car and evacuating on foot requires showing different routes. We can easily restrict our API in Google Cloud Console and use Directions API together with Maps API. Requesting the route to a specific direction we should define the destination. And as custom attributes we can specify the transportation mode. There are also other attributes that could be configured, but we will focus only on this because they are more useful.

By requesting the direction, we will receive encoded polylines that need to be decoded. And then we can draw it on our map. Also, we can estimate the time of arrival to the destination.

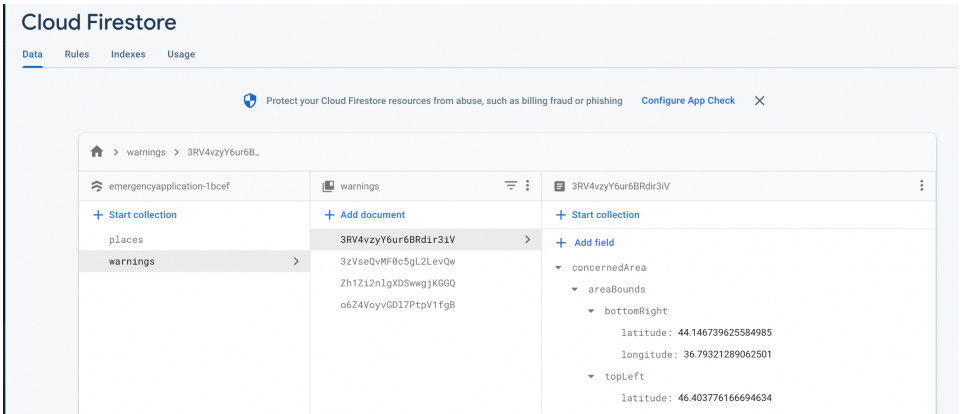
3.5 Connecting to Firebase

To implement the features we want we should connect our app to Firebase. We can easily do this in Firebase Console by simply writing the main package of your app. Also, it is desirable to generate SHA certificate fingerprints and provide it in Firebase. It is possible to do in Android Studio. We need to run a task **signingReport** using Gradle Tasks.



(*Firebase Settings*, 2022)

After setting up the app in Console, we must download a .json file with our configuration and put the file in app's directory. After that we are able to work with Firebase. Firstly, we want to get data about warnings. For this reason we could think about using Cloud Firestore. It will allow us to store data in very structured form. And we are able to read the data easily from Android App. Firestore stores that in form of



collections. We are able to get all documents in the collection. Every document represents a JSON object, so we are able to extract some data from the document stored in Cloud Firestore. To get the data from Cloud Firestore we should firstly initialize Firestore to it by simply providing the instance:

```
FirebaseFirestore.getInstance()
```

Compared with Realtime Database, Cloud Firestore is a bit faster. And it is more suitable when we want to work with a big amount of data. Also, Firestore supports a special type of data which is called **GeoPoint.** In our case it will be very helpful, because we are able to extract latitude and longitude. (*Firebase Cloud Store*, 2022)

```
val docRef = db.collection("cities").document("SF")
docRef.get()
        .addOnSuccessListener { document ->
            if (document != null) {
                Log.d(TAG, "DocumentSnapshot data: ${document.data}")
            } else {
                Log.d(TAG, "No such document")
            }
        }
        .addOnFailureListener { exception ->
            Log.d(TAG, "get failed with ", exception)
        }
```

3.6 Working with a network. Retrofit

Every Android app needs an Internet connection if you want to obtain some data from database. For this reason we can use internal android services. But mostly, developers use Retrofit as a library to work with Internet requests. **Retrofit** is a type-safe HTTP client for Android

Why to use?

Using Retrofit made networking easier in Android apps. As it has many features like easy to add custom headers and request types, file uploads, mocking responses, etc through which we can reduce boilerplate code in our apps and consume the web service easily.

Basic implementation

To work with Internet you need to grant permission in **AndroidMainfest.xml**

```
<uses-permission android:name="android.permission.INTERNET" />
```

Also, If we need to specify endpoints to get, create, update or delete data in the database, we can easily do it with the Retrofit library, using annotations. All the necessary API calls we write in separated interface. The library generates all boilerplate code under the hood.

To work with HTTP requests we need to initialize **Retofit,** using **Retrofit.Builder()** Usually, we parse data with JSON (JavaScript Object Notation). For this reason we should specify **converterFactory** property of RetofitClient. Converter factory needs to be added, just so Retrofit can convert JSON data into **Kotlin or Java** objects model, to use in Android Project. **GsonConverterFactory** is converter factory developed by Google that can help us to convert data from JSON to a model of object.

To work with Retrofit we basically need the following three classes:

1. A model class which is used as a JSON model. In Kotlin it is a simple **Data class.**
2. An interface that defines the **HTTP** operations needs to be performed.
3. **Retrofit.Builde**r class: Instance which uses the interface defined above and the Builder API to allow defining the URL endpoint for the HTTP operations. It also takes the converters we provide to format the Response.

https://square.github.io/retrofit/

```
fun providesRetrofit(): Retrofit = Retrofit.Builder()
    .baseUrl(baseUrl)
    .addConverterFactory(GsonConverterFactory.create())
    .build()
```

(*Retrofit Dependency*, 2022)

Other features of Retrofit:
Interceptors -is a powerful mechanism that can log, and retry calls.

Headers - are components of a network packet sent by a browser or client to the server to request for a specific page or data on the Web server. We can specify them using Retrofit. Also we can specify **headers** for specific requests. It makes sense when we need to check if the user has a permission to see the content

Handle requests - probably the main feature of Retrofit. We can check whether the response was successful or not and receive the body of the response in the form of **Data Class.**

3.7 What data do we expect to receive?

To create a responsive mobile app it is clearly visible that we need to work with HTTP requests and handle data models. Almost every HTTP response should receive or return some data. And we want to work with our data as an object to get some useful data using object properties.

Programmers usually use JSON to build models. But in Kotlin we receive JSON format as a String value. Storing the objects in String values is not that convenient. So that is why we need to convert our JSON format to Kotlin Data Class. For that purpose we should use a converter. Building an instance of **Retrofit** object we should provide a converter factory. For example, for processing JSON we can use GsonConverterFactory. (Retrofit also supports converters for XML responses)

So after successful response Retrofit will return an object as a Kotlin Data class. It gives us a strong advantage because now we can use that data for our business logic. We expect to receive a list of shelters using a GET request. The list must store some data type. In our case it could be **Shelter.** For example we will use the list of shelters to create a RecyclerView using Adapter or put a marker on the map. Understanding these use cases gives us a clear vision of what JSON object we should receive and what exact data we should store in our BackEnd. For example if we know that we want to display our shelter on the map it is clear that we should provide our JSON with latitude and longitude.

Modeling an objects

To work with data in Object oriented programming (OOP) developers could create a class that represents a particular object. Since the app is being coded in Kotlin, it is better to use **data class** for that purpose. A shelter should have a name, elevation above the sea and coordinates in the form of latitude and longitude. Using data class provides us quick access to its fields. We could access each field without generating getters and setters functions. It is already done by Kotlin.

```
data class Shelter(
    var name: String,
    var elevation: String,
    var lat: Double,
    var lng: Double
)
```

3.8 Android permissions

Android permissions might be useful for developers to support user privacy using the app by accessing to:

- **Restricted data**, such as system state and a user's contact information.
- **Restricted actions**, such as connecting to a paired device and recording audio.

Types of permissions

There are three types of permissions in the Android operating system. Each permissions specify the scope of restricted data that your app can access, and the scope of restricted actions that your app can perform, when the system grants your app that permission.

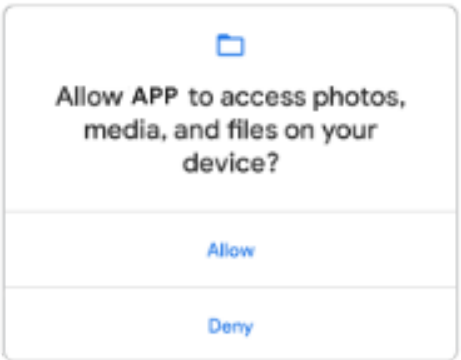
Install-time permissions

This permission type is no longer supported in the latest versions of Android due to lack of privacy. When you declare install-time permissions in your app, the system automatically grants your app the permissions when the user installs your app.

Normal permissions

Restricted data and actions may be accessed by using these permissions. However, the data and actions present very little risk to the user's privacy, and the operation of other apps.

Android assigns the the protection level as “normal” to normal permissions



(*Permissions Request*, 2019)

Examples of normal permissions:
android.permission.**INTERNET** - to access the data over the Internet
android.permission.**USE_BIOMETRIC** - to some biometric data as fingerprint or face within your app

Runtime permissions

Runtime permissions are marked as “dangerous” permissions in a list of Android permissions. They give your app additional access to restricted data. Also, they allow your app to perform restricted actions that more substantially affect the system and other apps.

Because it is a “dangerous” permission we need to request a permission in a runtime before the user access the restricted data or perform a restricted action.

Examples of private user data may include current location, contact information or media files

How to use permissions
If the app offers functionality that might require access to restricted data or actions, determine whether you can get the information or perform the actions without needing to grant permissions. Developers may implement features such as taking photos, pausing media playback
If you find out that your app must access restricted data or perform action, you need permission. Then you should think about which type of permission you need. For example, access to GPS location requires granting permissions in runtime.

Android permissions are used widely in almost any Android app.

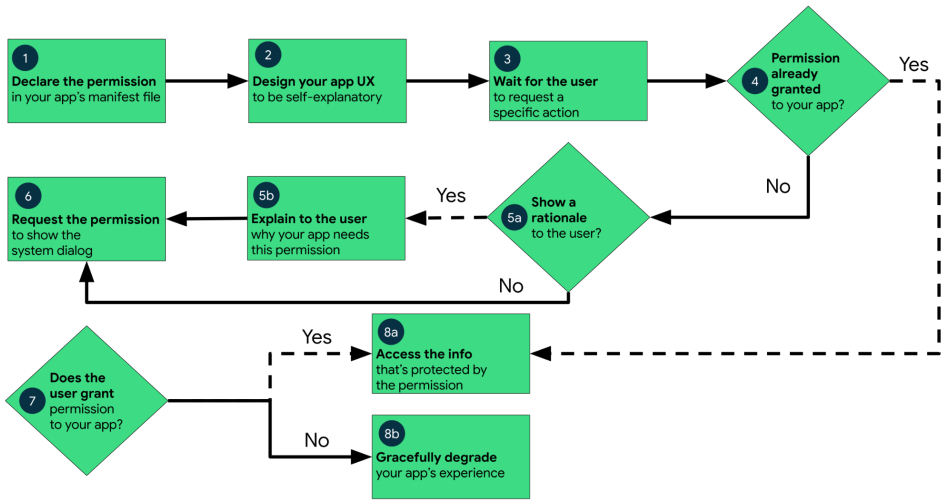
3.9 Configuring permissions in the app

We need to keep in mind what data we want to use to make our app working properly. We need to provide an INTERNET permission to work with our API. Also we need to grant permission at runtime to get the user's location. Observing a user's location is co-called “dangerous permission”. That is why the app is needed to ask the user to grant it manually. Such type of permission also requires a notification to be shown in the notification bar. Thus, the user understands which processes are running.

We are going to observe the user’s location in the foreground service. For this reason we also need to grant FOREGROUND SERVICE permission. This could be provided in the AndroidManifest file and we don’t need to ask the user to grant manually.

Since we are going to access the user's location we need to grant permissions: ACCESS FINE LOCATION, ACCESS COARSE LOCATION and ACCESS BACKGROUND LOCATION. These permissions are supposed to be dangerous. That is why we need users to grant them directly.
ACCESS BACKGROUND LOCATION - gives us the possibility to get the info about the user's location in the background. This feature is supported in Android API higher than 31 (Android 12).
ACCESS FINE LOCATION and ACCESS COARSE LOCATION are needed to be granted to access the user's location. The main and only one difference is that ACCESS FINE LOCATION returns more precise data. In our app we are going to use all of these 3.

The diagram below shows the workflow of granting “dangerous” permissions



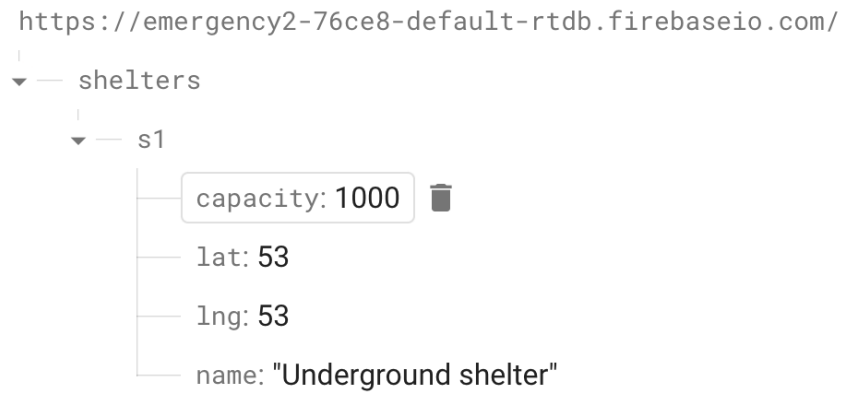
(*Permissions Request*, 2019)

3.10 Firebase

Firebase is a service that provides cloud functions. It could help to create apps with backend while mobile developers focus on the UI of the app. It has functions such as: cloud storage, creating and signing in users, sending messages, working with machine learning and more. With the help of Firebase we will build our backend. We will store the list of shelters in Database and also we will use Cloud Messaging to send notifications to a client.

Also, Firebase provides performance monitoring. By default it automatically attributes the actions related to the lifecycle of the application. Such as the time period between opening until becoming responsive. Moreover, the developer could specify different actions to be measured. For example, we can measure the time after receiving the notification and tapping the notification. This will help us to understand how much time user spend to open the app after receiving the notification.

Both of the databases supported by Firebase are non-relational. The difference between Realtime Database and Cloud Firestore is that Cloud Firestore supports storage of files, images and collections which provide better querying. And Realtime Database also could store data but in one JSON tree (JavaScript Object Notation). I decided to use Realtime DB, because we will not store a big amount of data



(*Firebase Database*, 2022)

Firebase Cloud Messaging

It is a tool which is also included in the definition of Firebase. FCM gives us a possibility to send push-notifications on multiple Android devices. On real projects developers usually develop their own middleware, the so-called layer between server and user.
It is a good tool for small projects. However, Cloud Messaging has very critical cons. Some of them I have experienced on my own.

Firebase Cloud Messaging has a **very strong minus** which is an unrealistic GCM heartbeat interval.This is possibly the most frustrating bug in Google Cloud Messaging. Google Cloud Messaging works by maintaining an idle socket connection from an Android device to Google's servers. This is awesome because it consumes less battery power compared to polling. The device is woken up instantly when the message arrives.

To ensure that the device is still connected, Android will send a heartbeat every 28 minutes if you are connected via mobile data and every 15 minutes if you are using Wifi. The connection could be terminated only when the heartbeat failed, and GCM will try to re-establish it and attempt to deliver any pending push notifications. That means the higher the heartbeat interval, the less battery consumed thus the less times the device has to be woken up after receiving the notification. The heartbeat interval on Android devices may vary from different manufacturers.

However, this comes at a great price: the higher the heartbeat interval, the longer it takes to identify a broken socket connection. So, if we do not want to notify the user’s instantly we can use Firebase Cloud Messaging, but in our case we want to deliver notification as soon as possible. For that reason we will work on a better solution.

3.11 Foreground Location

Some functionality of the app should be running in foreground service. The foreground service can work when even the app is closed. The foreground service can be killed by the system to optimize usage of resources. We are going to get the user's location in the foreground. This will help us to notify users in the area of danger.

This part of the app was designed to make the user experience more responsive.

Since we are going to get the user's location in the foreground service, we need to request permission **FOREGROUND LOCATION.** Firstly, we provide it in the Manifest file and then request the user to grant the permission. Doing such work we can get the user's accurate location any time we want and use that location to draw a polyline which navigates to the nearest shelter. In our case we will use the user's location only in case of hazzard after the notification was sent.

Also, I wanted to stress that getting a precise user's location means we touch privacy. Sending the build in production, we need to specify the reasons in Google Play Console why we need to use the user's location, for what feature. In other cases, the moderator will reject the submission of our application to the Play Market.

Display list of shelters

Since we want to display a list of shelters, we need to provide data about our shelters to the l UI element that represents items in the style of the list. Using **RecyclerView** and it's adapter class could help us to resolve the problem. For this purpose we need to create our own adapter class. Remember that the adapter pattern is created to interact with two objects of incompatible interfaces. In our case it will help us to bind the objects we received from the server to the RecyclerView by attaching an adapter to it.

To give more information to a user about shelters' location it is important to create a graphical representation of the item in the list that stands for shelter. In other words, developers should create an .xml file where the UI for item list is designed. And then, that UI will be used as a template. We could substitute and change the UI according to what data we received from the backend.

Using the adapter class for our RecyclerView we could populate the list with items containing the data from our server. For this reason recyclerview supports an adapter.

To setup an adapter we basically need to assign the value to property:

```
recyclerView.adapter = MyRecyclerViewAdapter(listOfData)
```

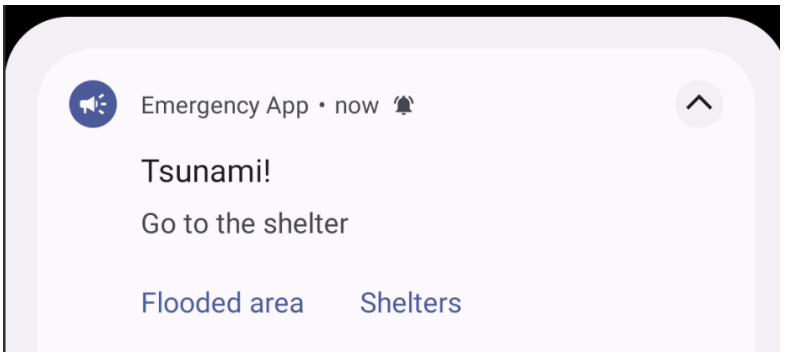
Once the adapter has been connected to the recyclerview, the contents of the list will be displayed.

3.12 Emergency notification

Since we are going to notify the users via notification, we need to build it. We are going to run a service that will show a notification once the message is sent. Since we want to send the notification even if the app is closed we should use **foreground service.** After Android 8 (Oreo) was released, developers needed to create a notification channel. This gives developers more options on how to manage notification broadcasting, because different notifications could be sent on different notification channels. For example, we possibly have two different notification channels. First one is for our service that observes the user's geolocation in the foreground. And the second notification channel is for sending emergency push notifications.

After configuring notification channels, we can show the notification to the user. To build a notification we can user standard Android library. Using class NotificationCompat.Builder, we can configure different parameters for our notification, such as icon, title, text etc. Since

our notifications are tied to time, we need to set the highest priority for the notification and show it from foreground service. Also it is great to open a specific screen of the app after clicking on the notification. For this purpose, we can create a **PendingIntent** and pass it to NotificationCompat.Builder via setPendingIntent(intent). In addition, we could use some prepared by Google styles for notifications even though we could create our custom design for notifications. In our case we just use some standart notification layout. For instance, creating custom views for notification could be useful if you develop a music player and you want to change or start/stop music from notification.



(*Emergency Notification*, 2022)

Android devices are not only smartphones. That could also be a smartwatch that uses Wear OS to work. To make the user's experience more modern we can duplicate our notification on smartwatch if it is connected via Bluetooth.

```
var builder = NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle(textTitle)
        .setContentText(textContent)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
```

4. UX/UI

In the majority of cases the best apps usually have a responsive and cool design.

4.1 Principles of creating UI for Android app

Android devices support two modes for displaying color palette. It is light and dark mode. Thus, developers should think about optimization for different users. To make it easier to set up colors developers just need to create two different themes which specify colors for text, buttons, status bar etc. The best practice is to get the color directly from the theme, because this will save your time in the future. Theme is just an **.xml file** where you could create some styles and then reuse it in your app.

To create UI developers usually needed to specify margins and paddings for different elements of the interface. Many developers suggest using margins and paddings which are numbers divisible by 4. Ideally, is to create a new file where you specify constant values for different types of margins.

Google suggests Android developers to follow Material Design guidelines which combine best design practices for Android development. One of its principles is to specify colors such **colorPrimary**, **colorOnPrimary.** Also there are colors such colorSurface and onSurfaceColor. Primary color is the main color of the color palette, this color is used more often than others. But what is colorOnPrimary? Prefix “On” means that this color is for the text. So, colorOnPrimary will be set up for the texts where the background of the element has colorPrimary. Also we need to keep in mind that the text must be readable, otherwise it does not look good. So, having those colors set up in theme will help us to easily redefine colors of the UI elements for dark and light mode.

One of the main principles of UX is to make the UI as easy as possible, so the user intuitively understands what to click, swipe etc. (*10 Fundamental UI Design Principles You Need to Know | Dribbble*, 2021)
To make our app usable it is not enough to write code that works perfectly. Developers should also consider features of the app in terms of a user. That means our app should be easy to use. For this reason the term UX/UI exists.

The goal of a UI/UX designer is to bring the user to some logical point in the interface. Make sure the user achieves their goal.

UX - is **U**ser E**x**perience. This is what experience / impression the user receives from working with your interface. Does he succeed in achieving the goal and how easy or difficult is it to do?

UI - is **U**ser **I**nterface. It's what the interface looks like and what physical characteristics it acquires. It determines what color your “product” will be, whether it will be convenient for a person to hit the buttons with his finger, whether the text will be readable etc.

So, UX/UI design is a design of any user interface in which usability is as important as appearance.

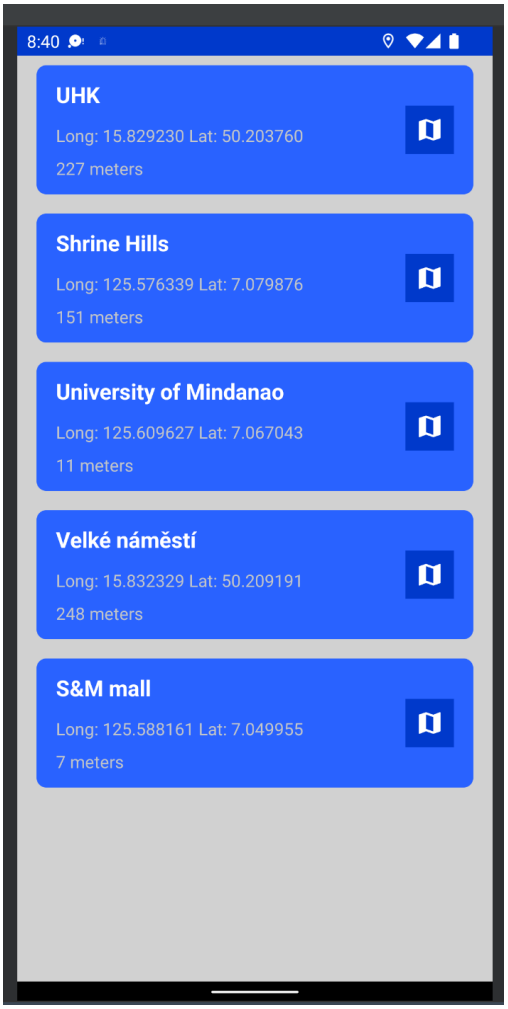
The direct responsibility of a UX/UI designer is, for example, to “sell” a product or service through the interface. It is on the basis of the work of the UX / UI designer that the user makes a decision: "Like or dislike. Buy or not buy."

The difference between UX and UI is that the UX designer plans how you will interact with the interface and what steps you need to take to get things done. And the UI designer comes up with how each of these steps will look like. As you can see from the examples above, UX and UI are so closely related that sometimes the line between concepts is blurred.

So, we want to make our app as good as possible. That means we need to focus on logic and design of the app. They should be very convenient to use.

One of the main principles of UX is **don't make users think.**
UX designers must follow the design standards developed for products, applications, and web design. For example, you should not place navigation in an unexpected place for users. Make sure people recognize buttons, calls to action, and links right away and find them easily, and these design elements lead potential customers to the right place.

The basis of creativity and innovation is not to create experiences that users have to re-learn, but to **solve problems that competitors have not thought about.**



(*List of Shelters*, 2022)

In the early 1900s, the basic building blocks of design consisted of three key elements: dots, lines, and planes. The most skilled UX professionals understand how to apply them to minimize design complexity, facilitate product navigation, and thereby improve user experience.

Also, it is very important to select a color palette for the application. Following the UI principles, we should select the main color for the palette. In material design it is called **primaryColor.** This color will be the most frequent color in the application. In the Emergency App, blue color is considered to be a primary. Selecting colors is no less important than writing code. Different colors can evoke different emotions and feelings (Cao, 2021). Blue is considered to be a color of trust and is associated with water, bringing a feeling of calmness, reliability, peace. For this reason, blue is the primaryColor. Users should trust our app, because the app will be used in emergency situations.

Since our app is developed for emergency situations, the user should be notified in case of emergency. In our application the main focus is on the map. The direction to a shelter should be drawn there as well as locations of all shelters. Also, in terms of UX principles it is important to send notifications with loud sound, so the user could recognize that Emergency App sent a notification. To make the user's experience better, the notification should have two action buttons. First one to navigate to a shelter, second to show the list of shelters. This feature gives users more options and a clear vision of how the app works.

Concluding, it is enough to click one button in notification to see the dangerous area and be navigated to a shelter. Performing only one click makes the app usability much easier. On the contrary, the notification could be sent without any action buttons and by only clicking on the notification we can navigate to a shelter. But in dangerous situations users might not know if the notification is clickable or not. They want to save lives and be in a safe place as soon as possible. That is why adding action buttons to a notification can make the experience better.

4.2 Users' opinions

To create a better user's experience it is important to keep in touch with users. They could possibly explain what is missed in the app and what could be improved. For that reason the expertes in UX/UI were asked to share thoughts on application. The group of 3 UI/UX designers and 3 users were selected to give a better assessment of the app.

According to their comments we should work around and bring an awesome experience to potential user's.
One of the valuable remarks is that user's should know what is happening in the app at the moment. It was advised to use the progress bar to show that data is loading. It seems to be not very important, but users are already used to seeing the loading of the particular screens. Using progress bars is very often practiced in different applications. It is considered quite successful according to UX principles.

Some users(testers, designers) think that it is not that much convenient to display the coordinates of a shelter using latitude and longitude. People came up with such a thing as an address, a street and postal code. For people it is more readable if the location is expressed as street and building number. However, we still need latitude and longitude to navigate to a shelter. User do not care about exact coordinates because it is too hard to even imagine the approximate location, but Maps API could help to get address and street using the coordinates. So, the location of the shelter should be written using street and building number to make it more understandable.
Since our application needs to maintain an Internet connection, it would be very nice if the program displayed a message after the connection was lost.

Also, testers suggested removing the button "Start" from the main screen which is supposed to start the simulation. It is better to start the simulation automatically since we want to make it useful in real situations and we want to show how it should work.

It would be better if we add the button "Navigate to the shelter" which must be visible only in case of hazzard. It is better because the user could think whether to use navigation or evacuate on his own. It is better because not every user wants to use navigation. For some of them, it is enough to be warned of an emergency. This should bring better user experience for users. It is not a good idea to navigate to a shelter every time the hazard occurs if the user does not want to for many reasons: battery usage, sick internet connection etc.

5. Results

The final result of the work is a complete Android app for emergency situations.

5.1 Key achievements

The UML diagram was designed to create some technical requirements for the app's functionality. The UseCase model was designed to capture requirements.
The Emergency app was developed in Kotlin language. Besides that we created a web-application to send warnings about tsunamis. So, we have a mobile client for ordinary users and a web client for an administrator who could send an emergency notification.
Also, to make the app working, the Android app as well as web app was connected to Firebase to use its features. For now the app is just a simulation of behavior in real hazzard.
The mobile app supports such functionality:

- Observe user's geolocation even if the app is closed
- Automatically start the simulation
- Send emergency notification
- Draw flooded area on the map
- See full list of shelters
- The navigation to a shelter

Also we focused on UX and UI to make it convenient to use in case of real hazzard. No less important is styling for light and dark mode. The app is considered to be more responsive after implementation of dark theme support as a lot of people use it.

The Firebase was used to implement some backend logic. However, there are things that should be improved in future development in terms of backend. Firebase is not that good for sending push notifications that is why we need to use any different service or create our own.

The code of the Android app was pushed to GitHub repository:
https://github.com/sergeykonar/emergency_app

5.2 Future development

It is not the best option to use Firebase for sending notifications. There are a lot of different options better than Firebase Cloud Messaging. One of the possible substitutes is **Pusher** (https://pusher.com/).

Since we want to warn not only in case of tsunami we need to implement more use cases. The idea is to notify in case of earthquakes and air alerts too. To support such functionality more strategies should be implemented both on backend side and mobile client. On the backend part we need to specify the type of emergency situation, so the app could apply a particular strategy in different situations.

Also, it would be good to notify the user's only in a dangerous area. In this version we notify all the users even those who are in a safe area. We have done the work behind, so at the moment we just need to show the notifications only for those who are in the selected area. For this reason we need to interact with our service that provides us with the user's geolocation.

Also in the future, we would like to start an animation once the navigating was started. It will improve the user experience, because everything will be smooth.In addition, it could be more user-friendly if the user could change the sound of the siren in the settings of the app.

Moreover, it is a good idea to create a native app for iOS. There are a lot of ways it could be achieved. By doing that, we could attract more users to our app. As a result more users will be notified in case of real danger and they will have a possibility to save their lives.

5.3 Conclusion

One of the purposes of the thesis was to analyze existing emergency apps, what features they have and then create my own prototype of the Android app which could be used to notify citizens in emergency situations.

Subsequently, application design has been created according to requirements and UX-principles. The description of the selected architecture patterns was given. The UI follows the rules of Material Design.

Functional prototype was developed by successful implementation of mobile development practices and modern technologies such as Firebase and Google Maps. The final application is just the simulation of the behavior in the emergency situation. The map plays a crucial role, because there the user could see the flooded area in case of danger.

Moreover, the nearest shelter could be found there. Another aim was to perform usability testing, which was carried out with a chosen group of people from various fields. Based on the testing, it is expected to introduce future improvements, such as an introductory guide, offline notification.

Bibliography

Air alert. (2021). Play Market. https://play.google.com/store/apps/details?id=com.ukrainealarm&hl=ru&gl=US

10 Fundamental UI Design Principles You Need to Know | Dribbble. (2021, December 6). Dribbble. https://dribbble.com/resources/ui-design-principles

Blog, B. T. (2022, May 16). *Geo-location tracking in Android with Kotlin - B/O Trading Blog*. Medium. https://medium.com/@chris_42047/geo-location-tracking-in-android-with-kotlin-f4ec57743956

Cao, J. (2021, January 26). 12 colours and the emotions they evoke. Creative Bloq. https://www.creativebloq.com/web-design/12-colours-and-emotions-they-evoke-61515112

ConstraintLayout : android developers. Android Developers. (n.d.). Retrieved July 28, 2022, from https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout?hl=fr

Create dynamic lists with RecyclerView : android developers. Android Developers. (n.d.). Retrieved July 28, 2022, from https://developer.android.com/guide/topics/ui/layout/recyclerview

Foreground services. (2020). Android Developers. https://developer.android.com/guide/components/foreground-services

GeeksforGeeks. (2022, June 9). MVVM (Model View ViewModel) Architecture Pattern in Android. https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/

Guide to app architecture |. (2020). Android Developers. https://developer.android.com/topic/architecture?gclid=Cj0KCQjwxb2XBhDBARIsAOjDZ36ZfQczVbSdfztl0mFmTqCuBgBOx3uTQjsp6Hnaw9zS4b-XiDNUWNsaAk7qEALw_wcB&gclsrc=aw.ds

Ionescu, D. (2010, March 30). Geolocation 101: How It Works, the Apps, and Your Privacy. PCWorld. https://www.pcworld.com/article/511772/geolo.html

Manifest.permission : android developers. Android Developers. (n.d.). Retrieved July 28, 2022, from https://developer.android.com/reference/android/Manifest.permission

MVVM Architecture - Android Tutorial for Beginners - Step by Step Guide. (2020, March 4). MindOrks. https://blog.mindorks.com/mvvm-architecture-android-tutorial-for-beginners-step-by-step-guide

Nava, E. (2018, August 13). Google Cloud Messaging is Extremely Unreliable for Push Notifications. Elad Nava. https://eladnava.com/google-cloud-messaging-extremely-unreliable/

Phamová, B. P. (n.d.). *Android Mobile Application for personal safety- CVUT.CZ*. Retrieved May 6, 2020, from https://dspace.cvut.cz/bitstream/handle/10467/88325/F8-BP-2020-Phamova-Bich%20Phuong-thesis.pdf

Japan earthquake and tsunami of 2011 - Aftermath of the disaster. (2011). Encyclopedia Britannica. https://www.britannica.com/event/Japan-earthquake-and-tsunami-of-2011/Aftermath-of-the-disaster

Handling false information in emergency management: A cross-national comparative study of European practices. (2021). Sciencedirect. https://www.sciencedirect.com/science/article/pii/S2212420921001175

Hazards - Red Cross - Apps on Google Play. (2021). Play Market. https://play.google.com/store/apps/details?id=com.cube.gdpc.nzl.hzd

Techopedia. (2020, August 7). Mobile Application (Mobile App). Techopedia.Com. https://www.techopedia.com/definition/2953/mobile-application-mobile-app#:%7E:text=A

pplication%20(Mobile%20App)-,What%20Does%20Mobile%20Application%20(Mobile%20App)%20Mean%3F,to%20those%20accessed%20on%20PCs.

Valdellon, L. (2020, November 2). What Are the Different Types of Mobile Apps? And How Do You Choose? CleverTap. https://clevertap.com/blog/types-of-mobile-apps/

ViewModel Overview |. (2020). Android Developers. https://developer.android.com/topic/libraries/architecture/viewmodel

Image references

Adapter pattern. (2022). [Model]. https://imgur.com/a/GmKLNCs

Architecture comparison [Table]. https://imgur.com/a/YjKuinT

App architecture. (2020). [Model]. https://miro.medium.com/max/1400/0*vnWgSTVC8qoJdusd.png

Clean architecture. (2019). [Model]. https://koenig-media.raywenderlich.com/uploads/2019/06/Clean-Architecture-graph.png

Earthquake & Tsunami Alerts. (2022). [Screenshot]. https://apps.apple.com/us/app/earthquake-tsunami-alerts/id1505203661

Emergency app architecture. (2022). [Model]. https://imgur.com/LzYIPxI

Emergency notification. (2022). [Screenshot]. https://imgur.com/w6cTYoh

Firebase Cloud Store. (2022). [Screenshot]. https://imgur.com/jqlrqE0

Firebase Database. (2022). [Screenshot]. https://imgur.com/arspV2g

List of shelters. (2022). [Screenshot]. https://imgur.com/zChrpSO

Mobile Operating System Market Share Worldwide. (2022). [Diagram]. https://gs.statcounter.com/os-market-share/mobile/worldwide

Map Fragment. (2022). [Screenshot]. https://imgur.com/rsklebw

MVVM Architecture. (2022). [Model]. https://journaldev.nyc3.digitaloceanspaces.com/2018/04/android-mvvm-pattern.png

Permissions request. (2019). [Diagram]. https://developer.android.com/static/images/training/permissions/workflow-overview.svg

Response handling. (2022). [Code]. https://imgur.com/undefined

Retrofit dependency. (2022). [Code]. https://imgur.com/poTtaQv

Strategy pattern. (2022). [Model]. https://imgur.com/D9XCrMd