BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

COMMUNICATION SYSTEM FOR THE VOLUNTEER FIRE DEPARTMENT

SVOLÁVACÍ SYSTÉM PRO DOBROVOLNÝ HASIČSKÝ SBOR

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. PETER ČAJKA**
AUTOR PRÁCE

SUPERVISOR **Mgr. KAMIL MALINKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

152629

Institut:	Department of Intelligent Systems (DITS)
Student:	**Čajka Peter, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Application Development
Title:	**Communication System for the Volunteer Fire Department**
Category:	Mobile applications
Academic year:	2023/24

Assignment:

1. Study thoroughly the principle of creating mobile applications for the Android operation system.
2. Analyze and compare similar existing solutions of communication systems for volunteer fire departments.
3. Design a communication system (consisting of a mobile application, a database, and an application server), that reflects current fire department requirements. Pay special attention to the reliability of the designed system.
4. Implement the designed system. The final version of the system should be able to send notifications to multiple mobile devices in real-time and subsequently process the reaction of the users of the given device to the received notification.
5. Test the created software solution with members of the volunteer fire department. Process their comments and opinions in an appropriate way. Also, test the reliability of the entire solution.

Literature:

- MEIER, Andreas a Michael KAUFMANN. SQL & NoSQL databases: models, languages, consistency options and architectures for Big data management. Wiesbaden: Springer, [2019]. ISBN 978-3-658-24548-1.
- GRIFFITHS, Dawn a David GRIFFITHS. Head first Android development.2 nd edition. Beijing: O'Reilly, [2017]. ISBN 9781491974056.

Requirements for the semestral defence:
Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Malinka Kamil, Mgr., Ph.D.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	17.5.2024
Approval date:	6.11.2023

Abstract

The process of gathering volunteer firefighters in case of an emergency is often not optimal. In critical situations, every second counts. The goal of this thesis is to create a mobile application that simplifies the process of informing volunteer firefighters about an emergency event. The resulting product will be a mobile application for the Android operating system, capable of rapidly sending SMS messages to all members of the volunteer fire department in case of an emergency and subsequently receiving their responses.

Abstrakt

Proces zhromažďovania dobrovoľných hasičov pri vzniknutej pohotovosti je častokrát neoptimálny. V kritických situáciach je každá sekunda dôležitá. Cieľom tejto diplomovej práce je vytvoriť mobilnú aplikáciu, ktorá zjednoduší dobrovoľným hasičom proces informovania o nežiaducej udalosti. Výsledný produkt bude mobilná aplikácia pre operačný systém Android, ktorá v prípade pohotovosti dokáže urýchlene rozoslať SMS správy všetkým členom v zbore a v zápätí obdržať ich odpoveď.

Keywords

volunteer firefighters, mobile application, location tracking, SMS, Firebase

Kľúčové slová

dobrovoľní hasiči, mobilná aplikácia, sledovanie polohy, SMS, Firebase

Reference

ČAJKA, Peter. *Communication System for the Volunteer Fire Department.* Brno, 2024. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Kamil Malinka, Ph.D.

Rozšírený abstrakt

Dobrovoľní hasiči nie sú neustále k dispozícii na hasičskej zbrojnici. V prípade poplachu sa musia veľmi rýchlo zmobilizovať a dostaviť na zbrojnicu. V niektorých hasičských zboroch je proces informovania členov o nežiaducej udalosti príliš neefektívny. Keďže je v kritických situáciach každá sekunda veľmi dôležitá, je dôležite proces mobilizácie zefektívniť.

Takmer každý človek dnes vlastní mobilný telefón. Dostupnosť mobilných telefónov umožňuje ich zapojenie aj do zvolávacieho procesu hasičských zborov počas pohotovosti.

FireReadyGo je mobilná aplikácia, umožňujúca jednoduchý proces informovania členov zboru o nežiadúcej udalosti prostredníctvom SMS. Veliteľ jednotky má môžnosť len pomocou pár klikov odoslať SMS správy v predpísanom formáte všetkým zvoleným členom zboru. Každému členovi sa po obdržaní SMS správy na pár sekúnd spustí hlasitý alarm sirény a vytvorí sa notifikácia. Vytvorená notifikácia obsahuje dve možnosti, pričom očakáva interakciu používateľa. Zvolením jednej z možností sa odošle spätná SMS veliteľovi s informáciou o účasti/neúčasti daného člena. Veliteľ má možnosť priebežne sledovať počet členov, ktorí sa zúčastnia hasičského výjazdu. Okrem účasti členov zboru sa veliteľovi zobrazuje aj čas od spustenia poplachu, keďže hasičské zbory musia uskutočniť hasičský výjazd do stanoveného limitu.

FireReadyGo taktiež využíva Firebase cloudové služby od spoločnosti Google. Aplikácia je prepojená s NoSQL databázou. Databáza uchováva informácie o členoch, udalostiach a hasičskom zbore. Velitelia majú možnosť vytvárať aj neurgentné hasičské výjazdy, alebo nejaké plánované udalosti, ktoré si nevyžadujú okamžitú odpoveď členov hasičského zboru. Údaje sú synchronizované v reálnom čase, takže aj malá zmena vykonaná jednym členom sa ihneď prejaví všetkým členom v aplikácii s dostupným internetovým pripojením.

Velitelia môžu vytvárať skupiny členov a manažovať vozový park hasičského zboru. Pri vytváraní zasahových vozidiel majú používatelia možnosť zadávať vlastné fotografie.

V rámci udalosti/hasičského výjazdu je každému členovi automaticky pridelené zásahové vozidlo a nejaká rola - veliteľ, strojník, hasič alebo zdravotník. Pridelenie vozidla a roly môže byť za istých podmienok zmenené. Po spustení udalosti/hasičského výjazdu sa používateľom sprístupní mapa, kde môžu vidieť svoju aktuálnu polohu a taktiež polohu svojích kolegov. Kliknutím na polohu iného člena zboru sa vypočíta približná vzdialenosť od aktuálnej polohy. Po skončení udalosti/výjazdu sa každému zučastnenému členovi vylepšia štatistiky, ktoré si môže prehliadať v profile vo forme grafov.

Prístup do aplikácie je umožnený len overeným a registrovaným členom. Overovanie prebieha na báze verifikácie telefónného čísla. Každému používateľovi je zároveň pridelený istý autorizačný level, ktorý mu sprístupňuje rôzne časti aplikácie.

Aplikácia FireReadyGo bola priebežne testovaná členmi dobrovoľného hasičského zboru a ich pripomienky, komentáre a nápady na zlepšenie boli zapracované do vývoja.

Communication System for the Volunteer Fire Department

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mgr. Kamil Malinka Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Peter Čajka
May 14, 2024

Acknowledgements

I would like to thank my supervisor Mgr. Kamil Malinka Ph.D. for all valuable advice during the creation of this thesis. Additionally, I would like to express my gratitude to all members of the volunteer fire department Demjata, who participated in the testing phases during the development of the application.

Contents

List of Figures

Chapter 1

Introduction

On a daily basis, in television, newspapers or on social networks, we come across news about events that negatively affect many people. Whether it is a car accident, a fire or some natural disaster, they all have one thing in common - people who need urgent help. Primary responders for these kinds of rescue operations should be firefighters. However, firefighters may not always be available, for example in the case of a very large number of emergencies or their on-site arrival time is high because of a long distance from the fire station. Some countries, including the Slovak Republic, have addressed these issues by establishing volunteer fire departments.

Volunteer fire departments are typically not on standby and do not have access to the latest technologies. During emergency, the process of calling individual members sometimes takes a significant amount of time. In crisis situations, every second is crucial.

I decided to write this thesis because I have been an active member of the volunteer fire department for nearly 9 years. During my active service in the volunteer fire department, I observed that communication among members, during both urgent and non-urgent dispatches to incidents, is not ideal. I have been thinking for a long time about how I could improve this process. Best idea was to create a custom smartphone based communication system which should be more effective in mobilizing members than making multiple phone calls one by one. More efficient in this case means faster response of members, continuous availability and ease of use.

The content of this thesis will involve analyzing existing systems for convening members of volunteer fire departments. The subsequent step will be the proposal of a custom communication system for volunteer fire departments called FireReadyGo. The proposed system will be entirely free of charge and should reflect the needs of the members. The resulting software will be primarily intended for smaller volunteer fire departments in the Slovak Republic. The design of the solution and the testing of the software took place continuously in cooperation with the members of the volunteer fire department Demjata.

Chapter 2

Analysis/Required information

This chapter covers some existing solutions of communication systems for (volunteer) fire departments, analysis and comparison of existing tools, programming languages, platforms and other software components. Outcome of this overview should be a list of the most suitable software and technologies which will be used during development of the custom communication system for the volunteer fire department.

2.1 Existing solutions

In this section, several existing systems used for convening volunteer firefighters are compared. Each system has its advantages and disadvantages, which are clearly shown in the summary table at the end of each subsection.

Hasičům.CZ

Paid software product Hasičům.CZ[1] is used for convening volunteer firefighters and also to support them during the whole intervention. Whole software consists of two main parts/applications:

- Desktop application
- Mobile application

Desktop application should be installed on computer which is located at the fire station. Mobile application is available for multiple users via Google play. Communication between devices is SMS-based, but some features requires also internet connection. Hasičům.CZ offers many features[2] like:

- Printing of the deployment command including map
- Sound notifications
- Sending of the GPS coordinates
- Control of the switching module
- Display of equipment intended for deployment

[1] https://www.hasicum.cz/
[2] Overview of features: https://www.hasicum.cz/inpage/co-budete-potrebovat/

- Alarming multiple members via SMS
- and many others

Hasičům.CZ also created a video[3] where are some features showed and explained. For potential customers they offer two plans[4]

- Start21 plan for 1500 CZK per year
- Top21 plan for 3600 CZK per year

This system looks like suitable option for bigger volunteer fire departments, or fire departments with frequent interventions. Hasičům.CZ contains many features, but in order to use its full potential multiple connected devices and controllers are required.
Pros

- many features
- sms based communication
- customer support

Cons

- no free plan
- for full functionality multiple controllers and devices are required
- pricing plan[5] only for Czech Republic

VIPTel

Another existing solution is the call-based system from VIPTel[6]. This system was designed in cooperation with volunteer fire department Beluša. According to VIPTel's website information, their system improved response time of Beluša's volunteer fire department members by 19%. In case of emergency, anyone can call numbers intended for reporting required interventions (notify dispatchers). Dispatchers may choose the type of an intervention and VIPtel's system will notify all members via phone call. Members may or may not join the intervention by answering or not answering incoming call. After 5 minutes the dispatcher will receive a list of joined members for the intervention. Setting up the service currently (2.12.2023) costs 0€ but monthly fee for usage of this system is 15€.
Pros

- no internet connection required
- high availability
- simple usage

Cons

- only a basic functionality
- paid system
- response list after 5 min

[3] https://www.youtube.com/watch?v=TXDfdDnM5pI
[4] Detailed description of plans: https://www.hasicum.cz/inpage/varianty-a-ceny-rozcestnik/
[5] https://www.hasicum.cz/inpage/zadost-o-cenovou-nabidku/
[6] https://www.viptel.sk/zvolavaci-system-pre-dobrovolnych-hasicov

FIREPORT

Another system, well known among the volunteer fire departments of Slovak and Czech republic, is FIREPORT[7], which consists of 4 parts:

- FIREPORT SMS
- FIREPORT PRINT
- FIREPORT OZS
- FIREPORT PAGING

Customers may use the whole system, or just some of its parts, depending on their needs. Core functionality of convening the members is SMS and call based. Extended functionality requires the internet connection and additional hardware located in the fire station.
Pros

- many features
- high availability
- credit based pricing

Cons

- no free plan
- for full functionality multiple devices are required

Summary of existing systems

Most of the currently existing convening systems for volunteer fire departments work on a similar principle. Multiple members are notified at the same time either by an SMS or by a phone call. Minimal required hardware for most of the above mentioned systems is a smartphone with a unique and valid phone number for each member. Since communication is provided by the mobile operator, high availability should be ensured. However, all of the aforementioned systems somehow include third party into the communication process, mostly by making calls/sending SMS and processing responses.

2.2 Smartphones and mobile apps

Smartphone plays the key part in most of the already existing convening systems for volunteer fire departments, as described in section 2.1. High availability of smartphones is probably the main reason why most of the existing convening systems for volunteer fire departments are smartphone based. Nowadays almost every person on the Earth owns at least one working smartphone [37]. According to research provided by Counterpoint [7], operating system Android covers 81% of global smartphone sales. Second in the ranking is naturally iOS with 16%. Sum of percentages for Android and iOS is almost 100%, therefore other operating systems are irrelevant for purposes of this thesis. Every mobile application developer must think carefully about what operating systems his potential customers will use. Applications developed for one specific platform are called native mobile applications. Cross-platform mobile applications are developed for multiple operating systems.

[7] https://www.fireport.cz/

Native vs Cross-platform development

Native mobile applications do not run in web browser like websites or web applications. In order to run mobile applications, they have to be downloaded and installed from platform specific application store. Android's store is called Google play store and iOS's store is called Apple's app store. Native mobile applications may access built-in features of smartphones like camera, microphone or GPS by default [25].

Performance and design of graphical user interface is also one of the strong sides of native applications, however they may be more complicated to develop than hybrid applications. Since native applications are created and optimized for specific platform without the need of multiple technologies, better security should be ensured and final product is more likely to be bug-free [21].

In the past was widely used programming language, for Android native development, Java. Nowadays is Java being replaced by Kotlin programming language. Applications for iOS may be developed either by using Swift programming language or Objective-C.

Cross-platform mobile application share the same development code for multiple platforms. Companies or individuals may save a lot of money and time by developing application for both Android and iOS just once.

According to Farooq et al. [11], there are multiple types of cross-platform development:

1. Web Approach - HTML, CSS and JS are core elements of every application
2. Hybrid Approach
    - Web applications are combined with native application features
    - Ionic framework
3. Interpreted Approach
    - Common language (like JavaScript) is being used.
    - The native APIs are accessed through an abstraction layer that is able to convert code between platforms
    - React Native
4. Cross-Compiled Approach
    - Common programming language
    - Transformation by using the cross compiler for every native platform
    - Applications may use native UI elements
    - Flutter

Max Savonin describes multiple advantages and disadvantages of cross-platform development in his blog [30]. According to his opinion, these are the advantages of cross-platform development:

- Cost Efficiency - create one application instead of two
- Faster Time to Market - faster development cycles
- Code Reusability - most of the production code may be used on multiple platforms

- Consistent User Experience - application should look and behave the same on different platforms
- Easier Maintenance - update and maintain just one codebase

Max also describes some disadvantages of cross-platform application development:

- Performance Limitations - cross-platform applications rely on a layer of abstraction to interact with the underlying platform, which may cause slower performance
- Limited Access to Native APIs - access to all native APIs and functionalities may not be provided
- Platform-Specific Bugs - even in shared codebase, platform specific bugs may occur and needs to be fixed separately

Performance

Our goal is not to reduce development costs, but to create a reliable mobile application with a simple graphical user interface and good performance. Because of that, in this subsection we will examine performance differences between native and cross-platform mobile applications.

According to Dorfer et al. [9] mobile applications created by cross-platform development consumed between 6% to 8% more energy. This study compared the same mobile application developed by React Native, which is popular language for cross-platform development, and by native Android application development. Use cases of this application during study included network queries and GPS tracking, which are quite common and widely used features in mobile applications.

Alex Sullivan in conclusion of his own performance testing of native and cross-platform mobile applications says: *„I feel confident in saying that a native Android app will perform better than either a React Native app or a Flutter app. Unfortunately, I do not feel confident in saying that a React Native app will out perform a Flutter app or vice versa. Much more testing will need to be done to figure out if Flutter can actually offer a real world performance improvement over React Native.“* [34].

2.3 Architecture

Most of the modern distributed applications, which allows interaction of multiple users, or use some network requests, are divided into 2 parts:

- Front end
- Back end

Dionysia Lemonaki describes front end as: *„The frontend is anything and everything visual that a user comes in contact with. It's all the parts with which they directly interact. It's all the content and styles. It's the buttons and the different hover effects before a user clicks on a link. It's the contact forms with various input fields, the search boxes and the dropdown menus. The layouts, text, and colors. It's the images and videos. However, it is not just the styles. It is also how fast the website loads, how easy it is to navigate through it, and how accessible it is to people with disabilities. It's how usable and responsive it is on a*

variety of different devices and browsers. Essentially, the frontend is all the parts of a web application that create the look and feel of it.“ [22].

Dionysia describes backend as: *„Backend development deals with the technologies responsible for storing and securely manipulating user data. It is the part associated with all the hidden logic that powers the applications users interact with. Backend is considered the server-side part of an application. Backend is all the hidden inner workings and the behind-the-scenes processes in a web application. It refers to everything going on underneath the hood and all the necessary components that make the front-end function properly and smoothly. It makes sure everything is working optimally. Essentially, the backend is what the users don't have direct access to or don't directly interact with and are most likely unaware of when using an application.“* [22].

From an architectural point of view, Codeacademy team describes back-end as: *„The back-end is all of the technology required to process the incoming request and generate and send the response to the client. This typically includes three major parts:*

- *The server. This is the computer that receives requests.*
- *The app. This is the application running on the server that listens for requests, retrieves information from the database, and sends a response.*
- *The database. Databases are used to organize and persist data.“* [35].

Back-end

There are several ways to create a back-end for mobile applications. Quite complicated and expensive solution is to buy a server and a data storage. This approach would require a custom server application implementation and a lot of setting up the mentioned components. A more popular solution is to use some existing hosting. For a relatively small fee, users can rent a server and data storage. However, in this approach is also custom server application implementation required. Used technologies are mostly independent on each other, therefore developers have multiple options. Some of the most popular programming languages and frameworks for server applications are [12]:

- .NET - ASP.NET Core
- Python - Django
- PHP - Laravel
- Ruby - Rails/ Ruby on Rails
- JavaScript - Express.js

Data may be stored in SQL or NoSQL databases. Quite simple and intuitive way, to store data, is to use a table. Tables are able to store multiple records, but each record in a table has to be of the same type. Structured query language (SQL) is in SQL databases used for operations with data and tables, like selecting some of the records from a table, inserting new records to the specific table and many others [23].

NoSQL, as the name implies, does not use SQL. Most of the NosQL databases are distributed, open source and they scale horizontally. There are many types of NoSQL databases and each type is suitable for different data. Well known NoSQL database types are [38]:

- Wide Column Store - data storage is organized into flexible columns. Multi-dimensional mapping is used for data referencing by column, row and timestamp. Names and format of the columns may be different across rows. Columns can be loaded and searched quickly. Biggest advantages of Wide column databases are speed of querying, scalability and a flexible data model [31].
  Examples of the Wide column store database management systems [38]:
  - Hadoop / HBase
  - Cassandra
  - Scylla
- Document Store - instead of fixed rows and columns, document databases use flexible documents. One record in a document database is called document. Documents usually stores information about one object in multiple key-value pairs. Multiple document formats are supported, like JSON, BSON and XML. Advantages of document databases are flexible document schema and geospatial data storage support [24].
  Examples of the document database management systems [38]:
  - Elastic
  - MongoDB
  - Cloud Datastore (Google)
- Graph Database - are able to handle very large sets of structured, semi-structured and unstructured data. Relationships between data play an important role. Graph databases make content management and personalization easier and they are widely used in social media platforms [26].
  Examples of the graph database management systems [38]:
  - Neo4J
  - Infinite Graph
  - GraphBase

Special case of a backend for mobile applications is Firebase[8]. Firebase is an application development platform, created by Google. Firebase's data store belongs to the document database category and data are usually stored in JSON format. Some of the Firebase's features widely used by developers are:

- Authentication - password, phone number, Google account...
- Realtime database - data synchronization across all clients
- Notifications - directly from the Firebase web GUI, or generated by an application
- Multiple pricing plans - including Free plan

[8] https://firebase.google.com/

2.4 Communication

Multiple existing convening systems for volunteer fire departments, described in section 2.1, use phone calls as mobilization process, which requires a dispatcher. Simple illustration of such system is shown in figure 2.1. Nowadays multiple CPaaS (Communication Platform as a service) exists, which are able to make multiple calls, such as:

- Vonage [9]
- Twilio [10]
- Plivo [11]

However, most of the available CPaaS offer only paid plans, moreover our goal is to create an independent mobile application with a full functionality, therefore final product should not depend on any third parties. Because of these requirements, SMS-based convening system seems like a better alternative. Multiple messages may be sent within a short period of time. Recipients should receive SMS basically at the same time. Simplified illustration of a SMS-based convening is shown in figure 2.2. Transfer time, reliability and availability depend on several factors, but mobile providers play a key role during whole multi-user communication. Biggest disadvantage of a SMS-based convening, from the user's point of view is probably individual billing. Each recipient, which decides to respond to an urgent event via SMS, will be charged based on active pricing plan associated with used SIM card/phone number.

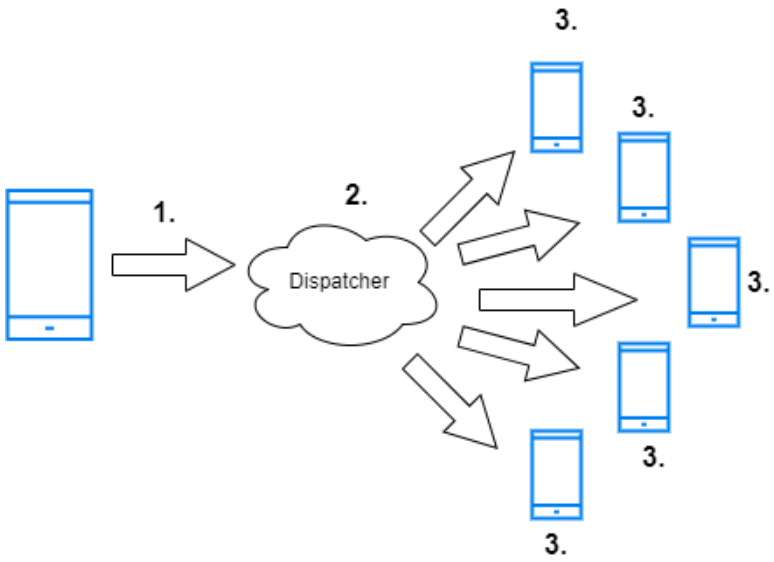


Figure 2.1: Calling system with dispatcher.

[9] https://www.vonage.com/about-us/
[10] https://www.twilio.com/en-us
[11] https://www.plivo.com/

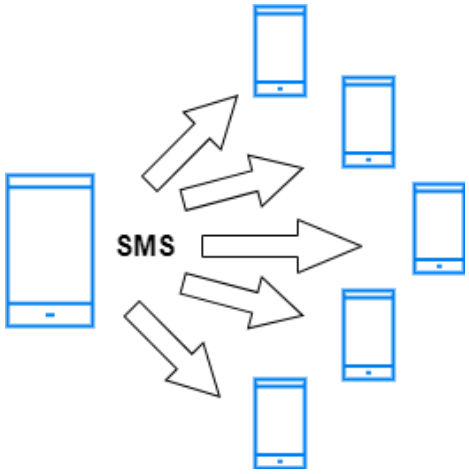


Figure 2.2: SMS-based convening.

Mobile providers in Slovakia

This thesis focus primarily on needs of volunteer fire departments located in Slovak Republic, therefore we assume that potential users of FireReadyGo will use services of a Slovak mobile providers. There are currently 4 major mobile providers operating in Slovakia:

- 4ka
- O2
- Orange
- Slovak Telekom

Opensignal conducted a study [2], in which it evaluated the quality of all 4 mobile providers. This study evaluates various aspects, comparing user experience in different areas such as download and upload speed, video playback, online gaming, availability of 5G, and much more. FireReadyGo requires only basic services, therefore, we focus solely on the results in the areas of availability, consistency and download/upload speed. Naturally, each user may utilize services from a different mobile provider, so average results will play a crucial role. Table 2.3 contains calculated average, minimum and maximum values in multiple categories. All numeric values used for calculation of average values were extracted from the Opensignal's study [2].

Statistics of mobile providers in Slovakia			
Type of service	**Average**	Minimum	Maximum
Availability (% of time)	**94.075**	86.9	97.7
Core Consistency (% of tests)	**91.725**	83.8	95.2
Download speed (Mbps)	**37.625**	18.4	52.5
Upload speed (Mbps)	**11.475**	9.2	14.0

Figure 2.3: Statistics of major mobile providers in Slovak Republic.

While the measured results of availability and consistency may not create ideal conditions, they should be more than sufficient for the occasional mobilization of members in volunteer

fire departments. Download and upload speeds are more informational than key values, as we do not anticipate heavy network utilization by the FireReadyGo mobile application.

Global Positioning System (GPS)

One of the most important and commonly used components of modern smartphones is the GPS. According to Amy He [20], approximately two-thirds of users utilize smartphones as a means of navigation. GPS has high potential and allows mobile application developers to incorporate enticing features. However, before the actual development, a thorough understanding of the global positioning system technology is necessary.

Global positioning system is owned by the USA and offers users services like positioning, navigation and timing. Whole system consists of three segments [16]:

1. Space segment - The united states's commitment is to maintain the availability of at least 24 operational GPS satellites 95% of the time. As of now, the U.S. Space Force operates 31 satellites [17]. The minimum number of satellites is not a random number, precisely 24 satellites have their justification. The official U.S government's website about the GPS and related topics explains it as follows:
   „The satellites in the GPS constellation are arranged into six equally-spaced orbital planes surrounding the Earth. Each plane contains four "slots„ occupied by baseline satellites. This 24-slot arrangement ensures users can view at least four satellites from virtually any point on the planet. The Space Force normally flies more than 24 GPS satellites to maintain coverage whenever the baseline satellites are serviced or decommissioned. The extra satellites may increase GPS performance but are not considered part of the core constellation." [17].

2. Control segment - The GPS control segment is a worldwide network of ground facilities that track, monitor, analyze, and communicate with the GPS satellites. This segment also sends commands and data to the constellation [15].

3. User segment - The GPS user segment is the only segment that is not developed, maintained and operated by the U.S Space Force. The GPS receiver equipment, which receives the signal from the satellites, is core part of the user segment. Received signal is used for calculation of three-dimensional position and time [16]. GPS receivers may be located in almost every modern equipment like smartphone, wristwatch, car, ATM and many others.

Technology improves every day and GPS has also became reliable service for monitoring of movement. While various specialized GPS receivers exist, the built-in receivers in smartphones can compete with them effectively. According to Tierney et al. [36], reliability of smartphone's tracking system is really high, even during variable motion and different speeds. This fact encourages us to incorporate GPS in as many features of the FireReadyGo mobile application as possible. Utilizing the receiver is free and basically unrestricted, but typically requires user consent on their mobile device. However, excessive use of the GPS receiver may result in higher energy consumption, since location service belongs to the most power-consuming services of a smartphone [28].

Network communication

An alternative mode of communication between users is facilitated through the internet. As described in Section 2.3, the establishment of a distributed application, comprising

multiple clients and a server, necessitates the assurance of seamless communication among them. Numerous communication protocols exist for facilitating data exchange between devices, each tailored to specific use cases and functionalities. Gordon H., in his article [19], explained and compared four popular solutions:

1. Hypertext Transfer Protocol (HTTP)
   - HTTP/1 - Operates on the basis of exchanging messages, with requests initiated on the client side and corresponding responses handled on the server side. Communication is unidirectional, therefore server lacks the capability to initiate communication with client. HTTP/1 is not optimal for real-time bidirectional communication between the client and the server due to its inherent limitations.
   - HTTP/2 - Addresses many of the performance limitations present in HTTP/1. It introduces various improvements such as multiplexing, header compression, and server push. HTTP/2 also enables bidirectional data streaming.
2. WebSocket - WebSockets were developed to facilitate full-duplex communication between a client and server, enabling bidirectional data transmission through a single open connection in real-time. WebSocket does not adhere to a specific format. It accommodates the transmission of various data types, including text and bytes. This adaptability contributes to the popularity of WebSockets. WebSockets excel in applications requiring real-time bidirectional communication, especially when there's a necessity to rapidly transmit small data chunks like chat applications or multiplayer games.
3. gRPC - An open-source Remote Procedure Call (RPC) system, originally developed at Google. gRPC generates cross-platform client and server bindings for multiple programming languages. This capability enables a client application to invoke a method on a server application located on a different machine, mimicking the simplicity of calling a method on a local object. Leveraging the foundation of HTTP/2, gRPC incorporates features like bidirectional streaming and integrated Transport Layer Security (TLS).
4. WebRTC - Free and open-source project that leverages real-time communication (RTC) capabilities, making it an excellent solution for applications requiring low-latency audio, video, or screen sharing.

2.5 Data protection

Since May 2018, the General Data Protection Regulation (GDPR) has served as a significant legal framework governing data and privacy matters. The primary objective of the GDPR is to enhance privacy safeguards. Should an application manage personal user data, it is imperative to adhere to GDPR compliance. Nevertheless, the GDPR articulates overarching principles rather than offering explicit, detailed instructions on the procedural steps for crafting an application that meets the stipulated requirements [10].

The FireReadyGo application will undoubtedly handle personal data, such as phone numbers and names. Consequently, it is imperative to thoroughly apprise users of the rationale behind the collection of this data. The GDPR gives end users following rights [3]:

- The Rights to be Informed - User should know what data is the application collecting and for what purpose.
- The Right to Access - Users are entitled to access their data upon request, and the company or the application owner is obligated to furnish the requested data within a 30-day timeframe.
- The Right of Rectification - A user possesses the right to rectify inaccurate or incomplete information pertaining to them.
- The Right to Erasure - Users can request the app owner to delete their personal information if the original purpose for data collection or processing no longer exists. Furthermore, users retain the ability to withdraw their consent for the collection or processing of information at any point if they discern unlawful processing or usage of their data.
- The Right to Restrict Processing - Users have the right to restrict the processing of their data. Application owner is obligated to promptly comply with the restriction.
- The Right to Data Portability - Data portability entails that a user can transfer their data from one device to another without interference from the application owner. Additionally, user has the right to request the application owner to transfer their data to any third party within the confines of the law.
- The Right to Object - A user possesses the right to request the application owner to cease processing their data. It is imperative for an application owner to communicate to the user both at the outset and within the privacy policy that they hold the right to object to the processing of their data.
- Rights in Relation to Automated Decision Making and Profiling - Application owners are obligated to obtain the user's explicit and informed consent before engaging in the automation and profiling of their information.

Encryption

Utilizing encryption stands as one of the most effective methods to ensure data security and compliance with GDPR standards. It's essential that all data gathered by a mobile application is stored securely, with encrypted backups as an added layer of protection. External communications should utilize SSL or HTTPS to ensure secure transmission [32]. To safeguard data in mobile applications, developers should consider the following tips [33]:

- Encrypt source code - Obfuscation renders the source code unreadable, significantly reducing its utility for potential cyber attackers.
- Manage keys securely - Follow best practices by NIST (SP 800-57 Part 1 Rev. 5) [12]
- Use file-level and database encryption - Ensuring the encryption of unstructured data stored in local file systems and/or databases on mobile devices is crucial for mitigating vulnerabilities. File-level and database encryption serve the dual purpose of

[12] https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final

safeguarding the data and diminishing the attractiveness of the database to cybercriminals, as the encrypted data renders itself meaningless without the appropriate decryption keys.

- Use the latest cryptography techniques - Developers should stay informed about current trends in cryptography and regularly conduct penetration testing and threat modeling. This proactive approach is crucial for ensuring the effectiveness of encryption and other protective measures.
- Multi-factor authentication - Adhering to the zero-trust model of „never trust, always verify,“ it is advisable to implement at least two factors of authentication for users. Ideally, each app should provide users with multiple authentication options, allowing them to choose their preferred, robust method.

There is a wide array of cryptographic algorithms and methods, but the two fundamental types are symmetric and asymmetric encryption. Symmetric encryption algorithms utilize the same key for both encrypting and decrypting data. This approach is relatively fast and efficient and is still commonly used, for example, in some messaging applications for securing communication. Asymmetric encryption is based on a key pair - a private key and a public key. Asymmetric encryption plays a crucial role, among other things, in SSL/TLS key exchange process [27].

Examples of symmetric encryption algorithms [8]:

- Data Encryption Standard (DES)
- Triple Data Encryption Standard (Triple DES)
- Advanced Encryption Standard (AES)
- International Data Encryption Algorithm (IDEA)

Examples of asymmetric encryption algorithms [8]:

- Rivest Shamir Adleman (RSA)
- Digital Signature Algorithm (DSA)
- Elliptical Curve Cryptography (ECC)

2.6 Android development

Within the scope of this thesis, we conducted an analysis of mobile application development possibilities and have arrived at the conclusion that a native Android application stands out as the optimal choice for the FireReadyGo system. The development of applications for mobile devices slightly differs from web and desktop development. Mobile applications typically rely on various components such as a camera, GPS receiver, Bluetooth, and others. A developer must possess not only knowledge of utilizing these components but also effectively communicate to the user the purpose for which the application requires access to such components. Subsequently, the developer must solicit and appropriately process these user notifications within the application, whether the user decides to grant or deny access permissions. In the case of mobile applications requiring prolonged activity, whether

in the background or foreground, it is imperative to optimize the application appropriately. Inadequate optimization may result in excessive power consumption, leading to a faster depletion of the smartphone battery. It is noteworthy that while a functional application may perform its intended tasks, it may not necessarily be optimized, resource-efficient, or user-friendly. In this section, we will delve into the fundamentals of developing mobile applications for the Android operating system.

Layout

Common native Android mobile applications typically consist of multiple distinct screens. The appearance of individual screens can be defined using layouts, usually specified in XML. GUI components visible to the user are referred to as „views“ and represent objects taking up space on the screen. All layouts are essentially a special type of view called a „view group“ [18].

There are various types of layouts, such as [6]:

- LinearLayout - A straightforward layout where elements are arranged either vertically or horizontally, based on the selected orientation.
- RelativeLayout - A highly complicated layout where elements are arranged based on their mutual relationships and associations with the parent element.
- AbsoluteLayout - Allows specifying the precise position of elements on the screen.
- TableLayout - Organizes elements into rows and columns, creating a tabular structure.
- FrameLayout - Essentially reserves space on the screen for a specific single view.
- ScrollView - A special type of FrameLayout that enables the display of a list of items extending beyond the dimensions of the screen.
- ListView - Displays a list of items.
- GridView - Displays items in a two-dimensional grid.

Activities and Fragments

The layout itself lacks any functionality, it merely displays GUI elements on the screen. To enable user interaction with the displayed components, such as pressing a button, the layout must be linked to code using a class known as an „Activity.“ An activity represents a specific task that a user can perform on the displayed screen. More complex applications typically consist of multiple activities that can communicate with each other using „intents“. Activities can exist in various states, which are defined by the activity lifecycle [18].

The composition of the activity lifecycle in Android is as follows [39]:

1. onCreate() - Initializes activity’s UI. This method is called when activity is created.
2. onStart() - This method is invoked when the activity’s UI is visible to the user.
3. onResume() - Relevant for the interaction with user. This method is executed when the activity starts running in the foreground.
4. onPause() - Relevant for the interaction with user. This method is executed when the activity stops running in the foreground.

5. onStop() - This method is invoked when the activity's UI is no longer displayed to the user.

6. onDestroy() - Typically used for resource cleanup. This method is called just before the activity is destroyed

A special method is onRestart(), which is called when the activity is stopped and we want to restart it. All methods of the Android activity's lifecycle may be overridden, however each method should have different, appropriately divided responsibilities. The entire lifecycle of activities, including relationships between individual states, is depicted in Figure 2.4.

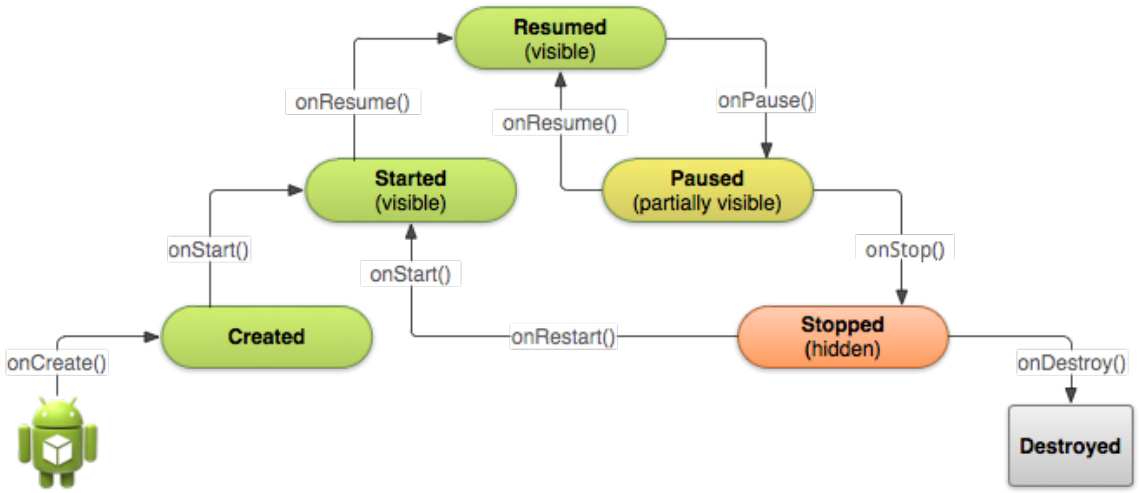


Figure 2.4: Visualization of the activity lifecycle [13]

As smartphones sometimes have limited screen size, certain tasks need to be divided across multiple screens. In the case of devices with larger displays, such as tablets, it becomes possible to utilize a single screen for the same tasks. By utilizing fragments, the application's appearance and behavior can vary based on the device on which the application is running. Fragments represents modular code components that can be reused by multiple different activities. Similar to activities, fragments contain layouts and the corresponding code associated with them [18].

Permissions and Android manifest

To ensure the privacy of users is appropriately safeguarded, an application must request permissions if it needs access to certain components. Applications that targets Android software development kit (SDK) version 23 or lower, requires all permissions during installation. Since SDK 23 (Android 6.0) is permission granting process divided into install-time and run-time levels [1].

Every Android application contains an XML file called the manifest. All permissions and application components are defined within the manifest. Improper configuration of this file can pose serious security risks [4].

Development and build tools

Android studio[13] is the recommended and official Integrated Development Environment (IDE) for Android application development. Android Studio was created and is currently maintained by Google. Connecting this IDE to other Google services is straightforward. Android Studio includes extensions, for example, for Firebase, making it easier for developers to integrate their applications with the cloud. For testing purposes, Android Studio provides the option of an emulator that can simulate the behavior of smartphones of various sizes, manufacturers, and with different versions of Android. The emulator even allows control over certain components, such as toggling internet connectivity or simulating received SMS messages.

The majority of Android applications are built using the Gradle[14] build tool. Among the main tasks of Gradle are searching for and downloading dependencies from various repositories, compiling code, running tests, and more. Typically, Android projects have two build.gradle files, where one is located at the project level and usually contains minimal information such as the Gradle version and a list of repositories. The second one is at the application level and provides more detailed information, such as the target and minimum supported Android versions, a list of libraries and their versions, and so on [18].

From a security perspective, Gradle allows code obfuscation, significantly hindering analysis of code and subsequent attacks on the application.

[13] https://developer.android.com/studio
[14] https://gradle.org/

Chapter 3

Proposal of the solution

In chapter 2 we analyzed all the necessary information and compared some possible approaches. In this chapter, solution will be proposed. The expected end product should be a native mobile application for the Android operating system and a backend connected to a database. The choice of a native mobile application is crucial, as we anticipate that the application may run for extended periods and during time-critical moments. Therefore, we need to ensure high performance and efficiency. The most suitable solution for the backend in conjunction with Android applications is Firebase by Google. Its easy integration, numerous supporting features, and a free pricing plan create ideal conditions for smaller projects.

The application should minimize dependencies on third parties. While the final form of the FireReadyGo system should include advanced features, the system must be capable of functioning correctly in a limited mode without an internet connection and access to the current location, provided it is appropriately configured. Therefore, all main use cases will be directed towards the usability of the application in an „offline“ mode. At the same time, proper interaction between multiple applications in offline and online modes must be ensured.

Within the application, firefighting dispatches will be frequently referred to as events. There are two types of events distinguished:

- Urgent - requires an immediate response from members of the volunteer fire department and is available even in offline mode. An example of an urgent event could be a fire or a car accident.
- Non-urgent - available only in online mode. This type of event represents a planned firefighting dispatch, or another future activity related to volunteer fire department, with an approximately chosen date and time. An example of an non-urgent event could be a planned water supply or training.

Each member in the volunteer fire department holds a different position. Similarly, within the FireReadyGo system, each member of the firefighting unit will have different permissions. Users with lower-level permissions will only be able to access a designated part of the application. Some features will be available only to users with higher-level permissions to prevent misuse of the application and false alarms.

Before the actual development, it is crucial to create a list of functional and non-functional requirements for the system.

3.1 Functional and non-functional requirements

Pavel Gorbachenko defines functional requirements in his blog as *„Something the system must do. If the system does not meet a functional requirement it will fail. This is because it will not be able to achieve something it must do to operate properly.“* [14].

Table 3.1 describes the functional requirements for the mobile application of the FireReadyGo system.

Functional requirements of the mobile application	
Category	**Requirement**
Connection	The application should be able to work in at least two modes: 1. No internet connection with limited functionality 2. Mode with full functionality The user should have the option to switch between available modes at any time, and the application should promptly respond and adapt to the change.
Events	The application should distinguish between two types of events: urgent and non-urgent. Events should exist in the following states: created, ongoing, and completed. In the case of creating an urgent event, it should be immediately considered ongoing.
Events	The application should be able to display multiple created and ongoing events to the user, with the possibility to view event's details.
Events	A user with standard permissions as a member of the volunteer fire department should have the ability to create non-urgent events. The creation of urgent events should be accessible only to members with high-level permissions, such as the commander.
Events	The application should include uniform categories of events. For specific events that do not fall into any predefined category, a default category named „Other“ will be created.
Events	In the case of applications running in full functionality mode, users should see an overview of registered members of the volunteer fire department for a particular event and their current locations during the ongoing event.
Mobilization	The creation of an urgent event should trigger a service that sends multiple SMS messages from the device of the member who created the event. The list of recipients for SMS messages must be accessible within the application and may be edited.

Mobilization	Users with insufficient permissions should not have the ability to create urgent events. Instead, when attempting to create an urgent event, the application should display contact information for authorized individuals or emergency services to these users.
Mobilization	Users should have the ability to participate in urgent events even without an internet connection.
Mobilization	Users should have the option to join a non-urgent event using an internet connection.
Notifications	The application should display notifications about urgent events with a possibility of a quick binary choice response (yes/no).
Notifications	The application should be able to display informative notifications about non-urgent events and other activity without the option to respond.
Permissions	The application should be able to detect different user's permissions/ authorization levels and adapt GUI accordingly.
Setup	The application should be partially dynamically configurable. For the proper functionality of the application, it is necessary to pair it with the backend and the database.
Setup	A new user must undergo registration and verification.
Setup	The application should verify users using their mobile phone numbers.
Setup	An existing user should have the option to log in to their account on a new device, meaning that they will not be required to go through the registration process again.
Users	The application should be able to display a list of users and their profiles.
Users	The application should be able to display a list of groups and members of respective groups.
Users	Users with appropriate permissions should be able to create groups and add other members to respective groups.
Vehicles	Members with high permissions/ authorization levels should have the ability to manage the fleet of vehicles, meaning - create, edit, and remove vehicles.

Table 3.1: Functional requirements

While the system can function even without meeting the non-functional requirements, it is not always considered a win. Availability, reliability, user-friendliness, and security are some of the categories influenced by non-functional requirements [29].

Table 3.2 describes the non-functional requirements for the mobile application of the FireReadyGo system.

Non-functional requirements of the mobile application	
Category	**Requirement**
Compatibility	The application should support Android SDK version 28 and higher.
Connection	The application should trigger only the necessary minimum of network requests for its proper functionality in online mode.
Data security	All data should be securely stored and should not be accessible to unauthorized individuals. Attempts to obtain data by attackers should not be feasible with common devices in real-time.
Event	The application should be able to send SMS to all members in the list within 10 seconds of creating an urgent event.
Event	In full functionality mode, the application should send information about the movement of members at least every 30 seconds during an ongoing event. If members stay in the same location for an extended period, this metric can be increased for GPS records measured in the same area.

Table 3.2: Non-functional requirements

3.2 Architecture

In the context of this thesis, we have proposed two architectural solutions for the FireReadyGo system. The difference between the mentioned options lies in the connection between the mobile application and the backend/database.

In Figure 3.1, we can see the first solution, which is based on distributing a common mobile application for all volunteer fire departments (A, B, C). The application will communicate with a shared backend and database that will store information about all clients, meaning details of all volunteer fire departments and their members.

The advantages of this approach are as follows:

- Simple for implementation and setup. This approach does not require custom configuration from customers.
- Widely used architecture
- In the case of a globally managed backend, such as by a state or region, having data from various fire departments available in one place could be a significant advantage.

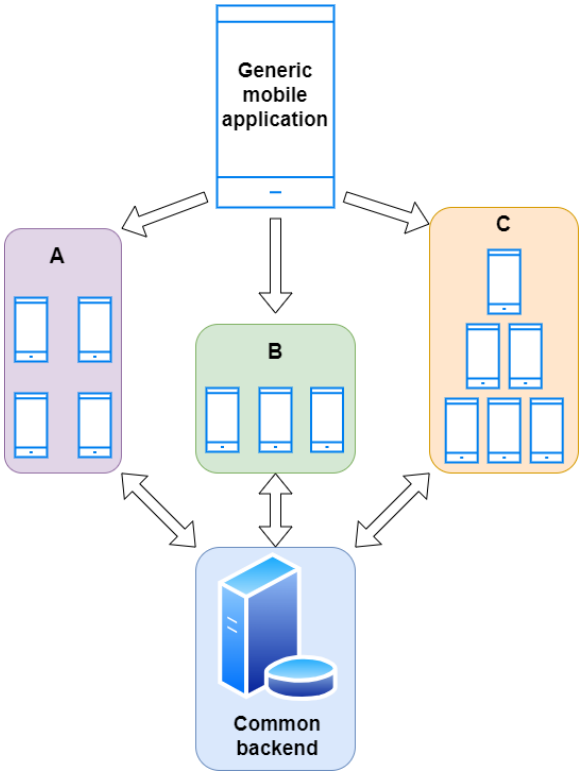


Figure 3.1: Common backend solution.

Disadvantages of this approach are:

- With a large number of clients, it requires a significant amount of computational resources.
- Expensive pricing plan when free limits are exceeded [1]

Figure 3.2 represents the second proposed solution, which is also based on distributing a common mobile application across volunteer fire departments (A, B, C). However, the connection to the backend is different in this case. This architecture considers self-management of the backend by individual volunteer fire departments. In simplicity, each volunteer fire department creates its own backend with a database and connects it to the mobile application. This way, we achieve data isolation across volunteer fire departments and, at the same time, lower backend load or computational resource usage. The approach of client-side backend configuration is unusual, but I am convinced that it has higher potential for practical use than the first architecture.

The advantages of this approach are as follows:

- Data isolation, which can enhance security from a global perspective.
- It allows the utilization of the free pricing plan for the backend from Google even with a high number of volunteer fire departments.
- Depending on the implementation, it may allow customization of certain features. For example, if the backend sends server-push notifications, individual volunteer fire departments can modify them according to their own needs.

[1] https://firebase.google.com/pricing

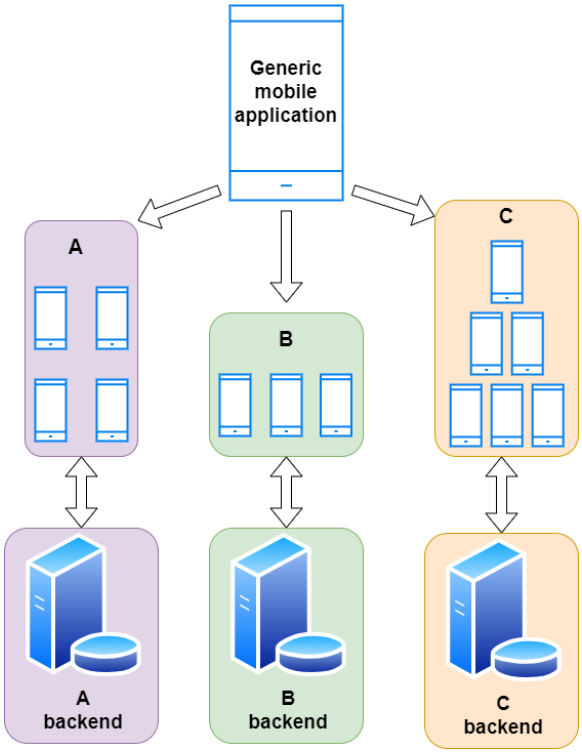


Figure 3.2: Separated backend solution.

Disadvantages of this approach are:

- Additional configuration of the backend by volunteer fire departments. This is a one-time action performed by one member.
- In case of necessity, the management and maintenance of the backend and database.
- Uncertainty of the free pricing plan - the entire advantage of this approach lies in utilizing the free Firebase pricing plan from Google, which may be a time-limited offer.

Reliability

The reliability of the FireReadyGo application will be based on minimizing dependencies on third parties. The only critical aspect of the application is the creation and transmission of SMS messages, which will be provided by the mobile operator (see Subsection 2.4). The proper functionality of the application will be verified through testing on multiple devices (see Chapter 5).

Chapter 4

Implementation

While the FireReadyGo application appears simple to the user, it conceals an extensive architecture and a large number of files in the background. This section will describe the different parts of the application divided into logical units. The order of the sections presented does not represent their priority or interdependence, except for registration, which users must complete in order to use the application.

4.1 Architecture

The original idea to use a distributed/separated backend (see Figure 3.2) could not be realized. Excessive effort was made to explore various options on how the application could connect to Firebase cloud services at runtime. However, for security reasons, this is not feasible. An application utilizing Firebase services cannot be properly built without the necessary configuration file. This file cannot be modified after installation. FireReadyGo is therefore distributed with a configuration file. The application is connected to cloud services immediately after installation, without the need for configuration. Users of the application utilize the following Firebase services:

- Realtime database - NoSQL database which stores all information about events, users and fire department.
- Authentication - Provides user authentication through phone number verification.
- Messaging - Allows to generate and send server push notifications.
- Functions - Javascript code which observes changes in database, and generates server push notifications for relevant users.
- Storage - Storage used for custom pictures of department's vehicle fleet.

The FireReadyGo application is logically divided into multiple packages. All parts of the application are designed according to the popular Model–View–ViewModel[1] pattern. Data is locally stored in models. Viewmodels handle, among other things, retrieving data from the database and transforming it into usable data within the application. Data in viewmodels is accessible through observers, ensuring instant data updates even with minor changes in the database by any member.

[1] https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/

Access to various parts of the application is controlled according to the authorization level. Upon registration, users are automatically assigned authorization level 1. The highest possible authorization level is 10. Authorization level can only be changed by a member with a sufficiently high authorization level directly within the application. However, the authorization level of the first highly authorized member in the volunteer fire department must be manually increased in the database.

4.2 Registration

Each user must go through a verification and registration process before being granted access to the application. The entire process is very straightforward and should not take more than a few minutes.

Verification

If a user has just installed the application, their identity is unknown. The first step after opening the application and accepting the terms is verification using a phone number. The user has access to a simple form in which they must enter their phone number with the correct country code to proceed.

After pressing the button to verify the number, a web browser automatically opens and the user is redirected to the webpage where the verification process takes place. Verification in the web browser may be completed in some devices within a moment, causing the browser to immediately close. In other cases, verification may require CAPTCHA.

After successful web verification, a verification SMS code is sent, which the user must enter into a new form that was not initially visible. The form for entering the verification code appears to the users immediately after pressing the button to verify the phone number. Example of the last step of the verification process is shown at Figure 4.1. If the verification code matches, the user is redirected to the registration screen, otherwise, the process must be repeated.

Verification is handled by Firebase Authentication from Google. With an active Blaze plan [2], up to 10 users can be verified per day. If the daily limit of 10 verification SMS messages is exceeded, it is possible to increase the limit for a fee or wait until the next day for the limit to reset.

[2] https://firebase.google.com/pricing

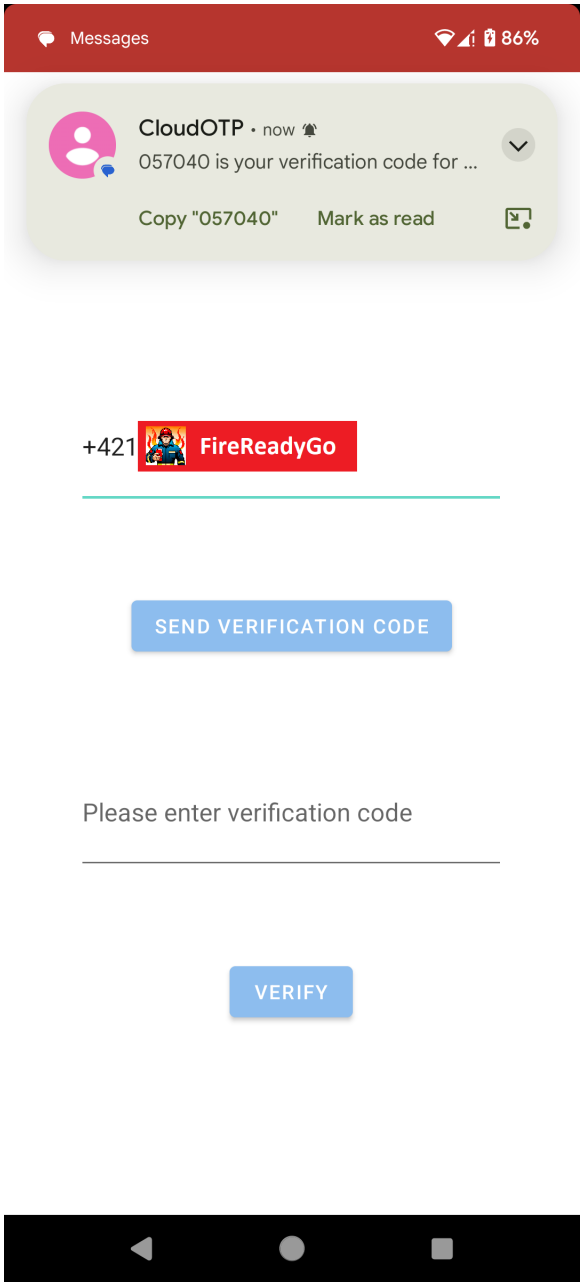


Figure 4.1: SMS code verification process

Figure 4.2 displays the code for creating settings for the verification process. The most important part from the developer's perspective is the setCallbacks() method, which must take correctly implemented callbacks as input parameters. Once verification is completed, one of the callbacks is invoked based on how the verification process went. The developer needs to handle various scenarios and implement both a successful callback and an error callback (see Figure 4.3). A successful callback should include call of the function signIn-WithCredential(credential) on the FirebaseAuth [3] object to finish authentication process.

[3] https://firebase.google.com/docs/reference/android/com/google/firebase/auth/FirebaseAuth

```
sendCodeButton.setOnClickListener { it: View!
    Log.d(TAG, msg: "Requiring verification code")
    phoneNumber = phoneNumberField.text.toString()
    val options = PhoneAuthOptions.newBuilder(auth)
        .setPhoneNumber(phoneNumber!!)
        .setTimeout( timeout: 60L, TimeUnit.SECONDS)
        .setActivity(requireActivity())
        .setCallbacks(callbacks)
        .build()
    PhoneAuthProvider.verifyPhoneNumber(options)

    // Hide the keyboard
    val view = this.activity?.currentFocus
    if (view != null) {
        val inputMethodManager =
            this.activity?.getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, flags: 0)
    }

}
```

Figure 4.2: Code snippet - Phone authentication options

```
callbacks = object : PhoneAuthProvider.OnVerificationStateChangedCallbacks() {

    override fun onVerificationCompleted(credential: PhoneAuthCredential) {...}

    override fun onVerificationFailed(e: FirebaseException) {...}

    override fun onCodeSent(
        verificationId: String,
        token: PhoneAuthProvider.ForceResendingToken,
    ) {...}
}
```

Figure 4.3: Code snippet - Phone authentication callbacks

Registration

If the user has already completed the registration process and therefore has their profile saved in the database, they are automatically redirected to the home screen without the need for re-registration.

Unregistered users must fill out a form with their first and last names. Then, users choose an avatar, which serves as a substitute for a profile picture. After filling out the forms and selecting an avatar, the registration process can be completed by clicking registration the button. Example of the registration screen is displayed at Figure 4.4.

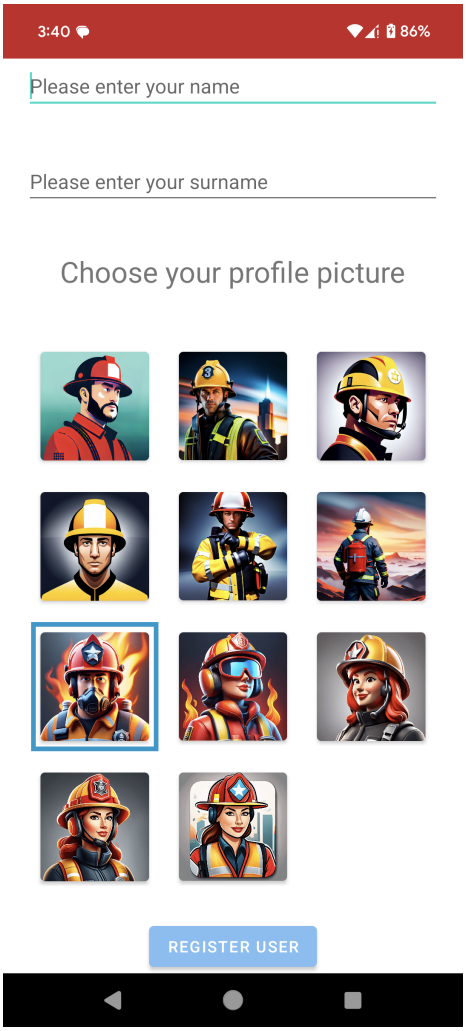


Figure 4.4: Registration screen

```
val newUser =
    User(
        auth.uid!!,
        EncryptionUtils.encrypt(name),
        currentExperience: 0,
        rank: 0,
        privileges: 1,
        profilePictureName,
        mutableListOf(),
        EncryptionUtils.encrypt(phoneNumber)
    )
userViewModel.createUser(newUser)
```

Figure 4.5: Code snippet - Create a new user

Pressing the „REGISTER USER“ button creates a new profile in the database with default data, encrypted name + surname, and phone number (see Figure 4.5). Personal information entered during verification and registration does not leave the device in an unencrypted/plaintext form.

Encryption

The FireReadyGo application uses the AES algorithm in CBC mode for encrypting and decrypting text. The application has an initialization vector (IV) and a secret key stored, which are necessary for running the algorithm. Base64 encoder and decoder are used as helper tools for data transformation. Encryption and decription functions are displayed at Figure 4.6. Data is encrypted before being transferred out of the application, such as when creating a new user and uploading data to the database (see 4.2). When data is retrieved from the database, it is decrypted before it can be used. The process of mapping retrieved data to the objects used in the application includes decryption. During the runtime of the application, data is stored unencrypted.

```
// Function to encrypt data
Peter Čajka *
fun encrypt(data: String): String {
    val cipher = Cipher.getInstance(TRANSFORMATION)
    val secretKeySpec = SecretKeySpec(SECRET_KEY.toByteArray(), ALGORITHM)
    cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, IvParameterSpec(IV.toByteArray()))
    val encryptedBytes = cipher.doFinal(data.toByteArray())
    return Base64.getEncoder().encodeToString(encryptedBytes)
}

// Function to decrypt data
Peter Čajka *
fun decrypt(encryptedData: String): String {
    val cipher = Cipher.getInstance(TRANSFORMATION)
    val secretKeySpec = SecretKeySpec(SECRET_KEY.toByteArray(), ALGORITHM)
    cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, IvParameterSpec(IV.toByteArray()))
    val decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedData))
    return String(decryptedBytes)
}
```

Figure 4.6: Code snippet - Encryption and decryption

Permissions

After a successful registration, the user is redirected to the home screen, where they are asked for all the necessary permissions for the app to function properly. Due to the nature of the app, all permissions are requested at first startup, not when they are needed. FireReadyGo requires the following permissions:

- Location - to track members location during the event
- Notification - to display notification about events and emergencies
- SMS - to parse incoming messages and send messages directly from the app during an emergency

4.3 Events

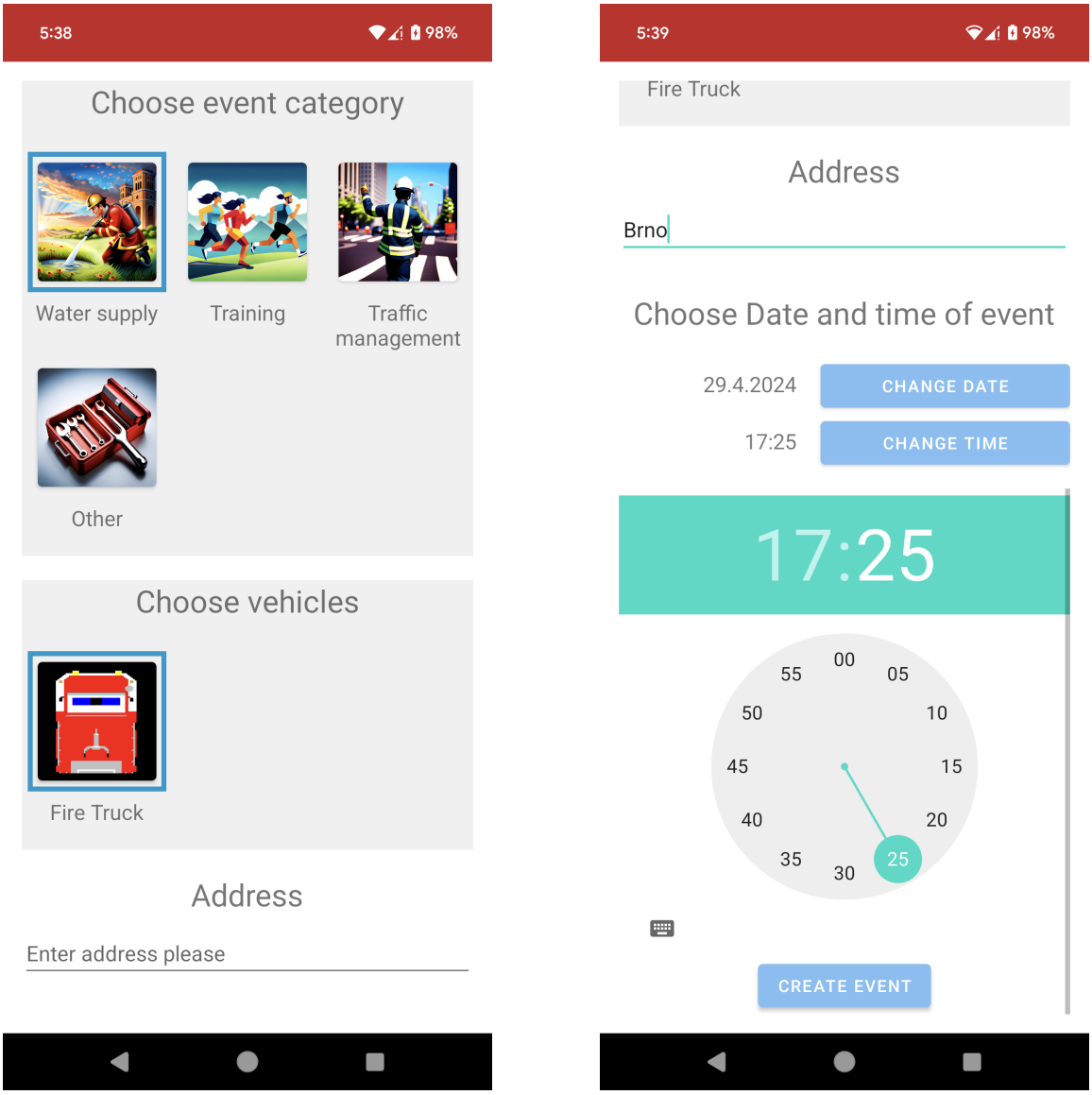
The most important part of the application is events. Creating, modifying, and interacting with events takes place on the home page. All event-related actions require internet connection, however user has local database copy available. Authorized users may interact with events even without the internet connection, but changes in database will not be synchronized with other users until internet connection is available.

Non-urgent event creation

Events scheduled for a specific time in the future and not requiring immediate interaction from members of the volunteer fire department are referred to, in this thesis, as non-urgent. By clicking the „CREATE NEW EVENT“ button (see Figure 4.8), the user is redirected to a new screen where they must select the following parameters:

- Event Category - there are 3 specific event categories, which should cover most common non-urgent activities of volunteer firefighters. To cover also unusual events, special „Others“ category was created.
- Vehicle(s) - user can choose any number of vehicles. The available vehicles depend on the current fleet of the fire department, for which authorized users, such as the chief commander, are responsible for keeping up to date.
- Date of the event
- Time of the event

Another parameter, which is optional, is event address. Example of an non-urgent event creation with all mandatory and optional fields is shown at Figures 4.7a and 4.7b.



(a) Part 1

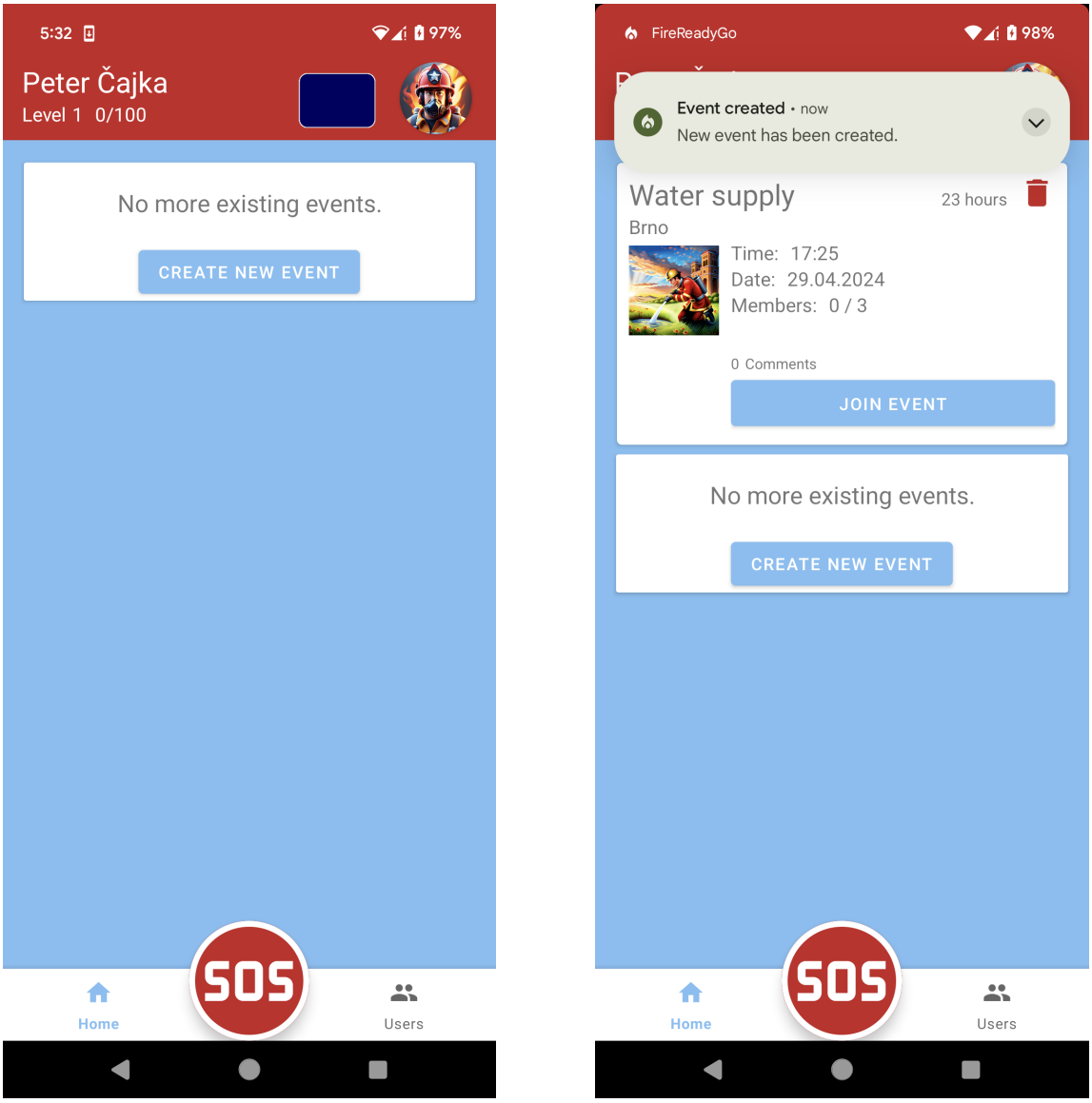
(b) Part 2

Figure 4.7: Creation of a new non-urgent event

Each event can be in one of the three states:

- CREATED - An event is in the „CREATED“ state if it has been created but has not yet been started. In this state, members can interact with the event.
- ONGOING - An event is in the „ONGOING“ state when it has been started. Members can see such event on the home screen, but they can no longer join the event.
- FINISHED - An event is in the „FINISHED“ state when it has been successfully completed. Such event is hidden from users but is archived in the database.

By default, non-urgent events are created with „CREATED“ state. After the event is created, a server push notification is automatically sent to all members, as shows Figure 4.8b



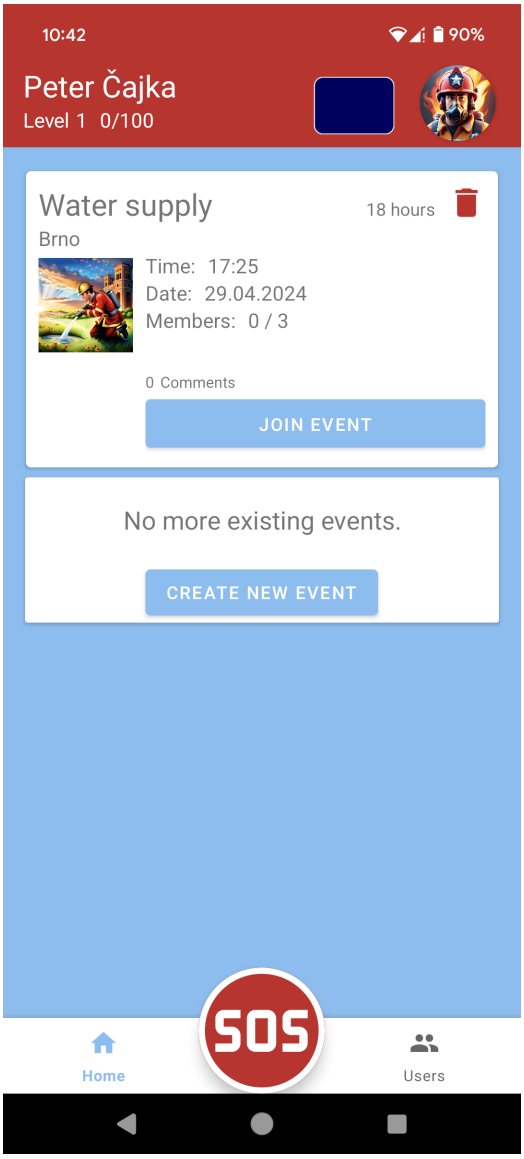
(a) No existing events

(b) New event in CREATED state

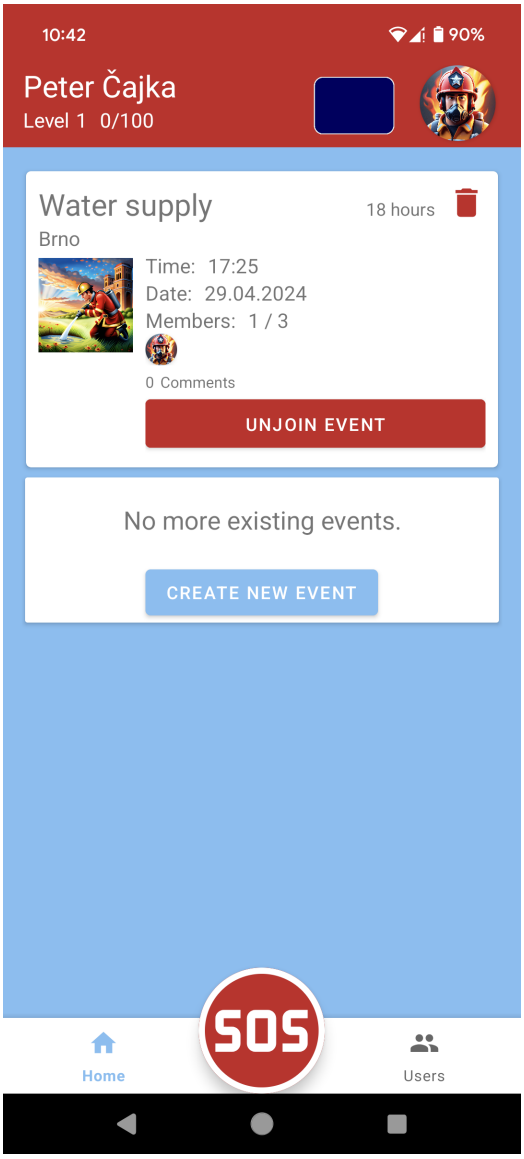
Figure 4.8: Comparison of an empty and non empty non-urgent event list

Non-urgent event interaction

The list of events in the „CREATED“ and „ONGOING“ states is available on the home page. Users can interact only with events that have not yet started, that is, events in the „CREATED“ state. The user can request to join any event. If there is an available spot in the created event, the user is assigned to a vehicle in a specific role. A brief overview of the event, including avatars of joined members, is available directly from the home page, as depicted at Figure 4.9.



(a) User is not joined to the event



(b) User is joined to the event

Figure 4.9: A comparison of event overviews

Detailed information about the event is accessible by clicking on the event. Figure 4.10 shows an example event with all the information that was provided when creating the event. At the bottom of the screen, there is a list of vehicle crews. Each vehicle crew contains a vehicle from the fire department's fleet and the crew of that vehicle. The capacity of the crew is determined by the maximum capacity of the vehicle. The crew consists of a total of 4 roles, although not all of them need to be filled:

- Commander
- Driver
- Fireman
- Medic

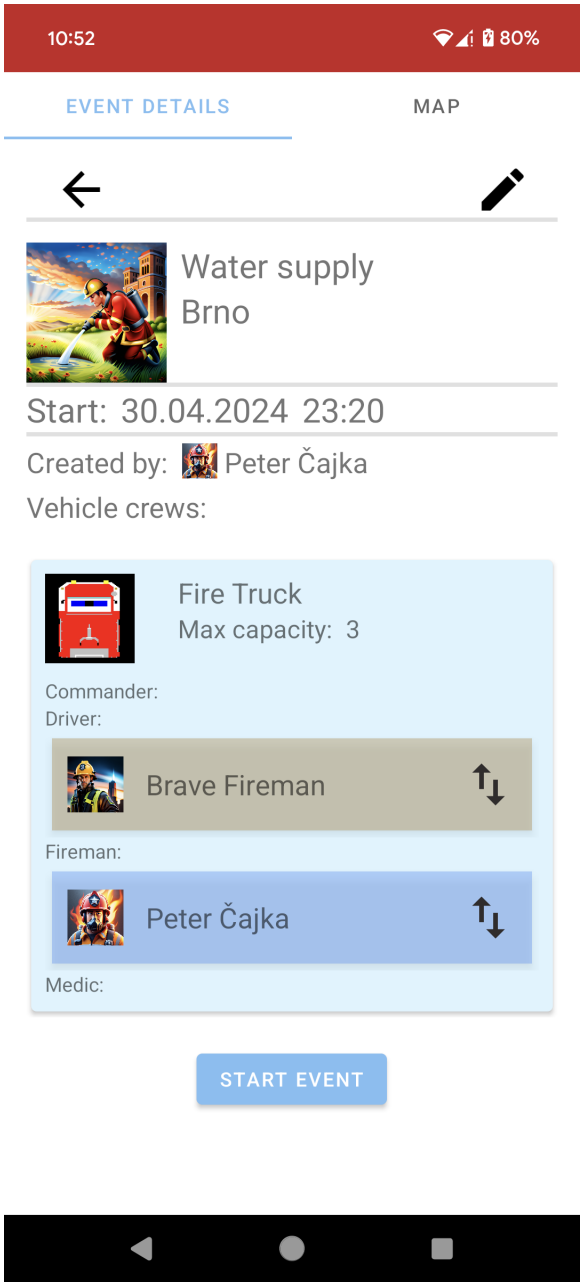


Figure 4.10: Event details

For each new member who signs up for an event, the best role and vehicle crew are automatically calculated. Figure 4.11 illustrates a part of the method that determines the appropriate selection of the role and vehicle crew. The displayed code iterates through all available vehicle crews within the created event, searching for the first vehicle crew that does not yet have a commander position filled.

```
Peter Čajka *
fun getBestAvailableRoleAndVehiclePair(event: Event, user: User): Pair<Int, Role>? {
    // Priority no 1 - check if user can be commander in any vehicle
    if (user.roles.contains(Role.COMMANDER)) {
        // User may be a commander, check if there is any vehicle crew without commander
        for ((index, crew) in event.vehicleCrews.withIndex()) {
            if (crew.commanders.isEmpty()) {
                return Pair(index, Role.COMMANDER)
            }
        }
    }
    // Process other roles...
```

Figure 4.11: Code snippet - vehicle crew and role assignment

If it finds a vehicle crew without a commander, it returns the index of the vehicle crew and the role of commander for the user, then terminates. If it does not find a vehicle crew with a free commander position, it checks also other roles in the similar way, following this priority order:

1. Commander
2. Driver
3. Medic
4. Fireman

The roles of commander and driver must be empty in order to assign a new user to them, because each vehicle, or vehicle crew, can have only one commander and one driver. The roles of fireman and medic have an unlimited capacity, so there is no need to check whether these roles are already filled or not. However, despite the unlimited capacity of the fireman and medic roles, the maximum capacity of the vehicle must not be exceeded. The control of overall occupancy is ensured by the method in Figure 4.12.

```
Peter Čajka
private fun isVehicleCrewFree(vehicleCrew: VehicleCrew): Boolean {
    var members = 0
    var mandatoryMembers = 0
    mandatoryMembers += vehicleCrew.commanders.size
    mandatoryMembers += vehicleCrew.drivers.size
    members += vehicleCrew.firemen.size
    members += vehicleCrew.medics.size
    // Check vehicle capacity -2 to ensure that driver and commander have "reservation"
    // Second part should cover case when multiple commanders or drivers are joined (should not happen)
    return members <= (vehicleCrew.vehicle.maxCapacity -2) && (members + mandatoryMembers) <= vehicleCrew.vehicle.maxCapacity
}
```

Figure 4.12: Code snippet - vehicle crew capacity check

Authorized users can further adjust member assignments within the event. By clicking on the icon of opposing arrows located next to the name of a member (see Figure 4.10), a new dialog opens in which the member's assignment can be changed. Figure 4.13 shows such a dialog. At the top of the dialog is the current assignment of the member. In the bottom left part, there is a list of available vehicles in the event to which the member can

be assigned. In the bottom right part, there is a list of roles to which the member can be assigned. User may click on any combination of the vehicle and role and save changes. Reassigning a member will be immediately updated in real-time.

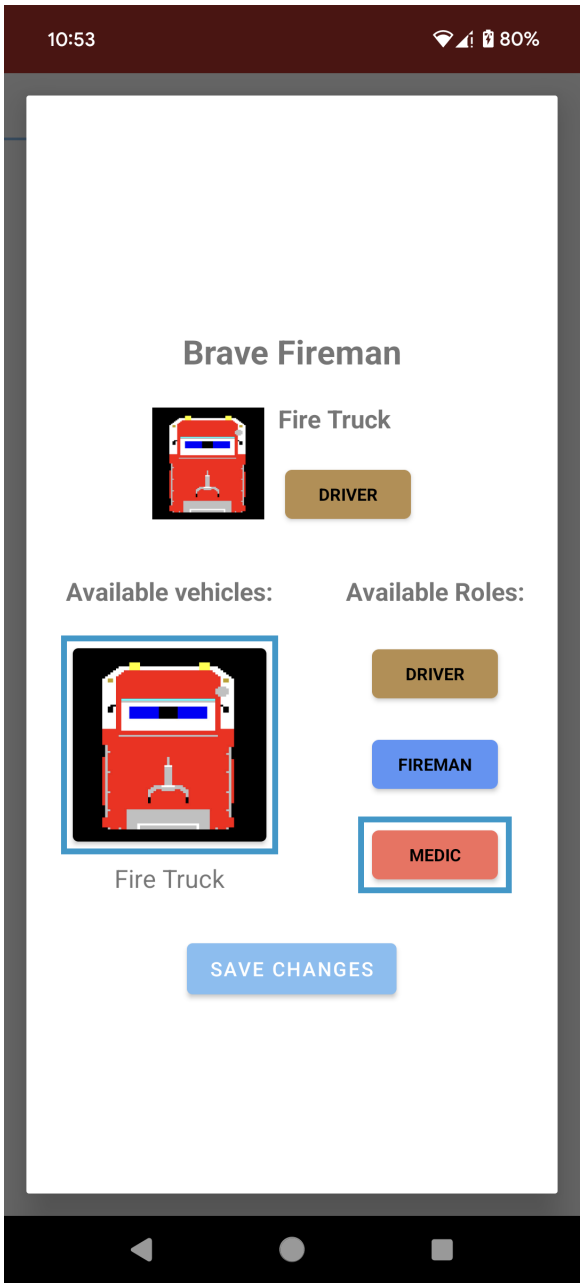


Figure 4.13: Reassignment of a user within the event

In addition to event details, users also have access to a tab with a map, which is not available until the event is started. The content of the map during an ongoing event is described in details in section 4.6.

Event may be started, edited or deleted only by the event creator, or an user with high authorization level. Event deletion is carried out using a soft-delete method, meaning it is only hidden from the user but can be retrieved from the database at a later time. Editing

an event is almost identical to creating a new event, with minor modifications. The event category cannot be changed during editing. Adding a new vehicle to an event does not cause any conflicts, but removing a vehicle that is currently assigned to an event is only possible if no members are assigned to the vehicle (crew).

Finishing the event

After starting the event, the „START EVENT“ button changes to the red „FINISH EVENT“ button. The rest of the screen remains unchanged. Pressing the „FINISH EVENT“ button opens a new dialog with options to adjust the start and end times of the event. Figure 4.14 displays an example dialog with adjusted dates and times. The default start time of the event is identical to the time it was planned. The default end time is the current time. The difference between the selected times is calculated in minutes. The calculated difference is presented to the user as the time spent on the event.

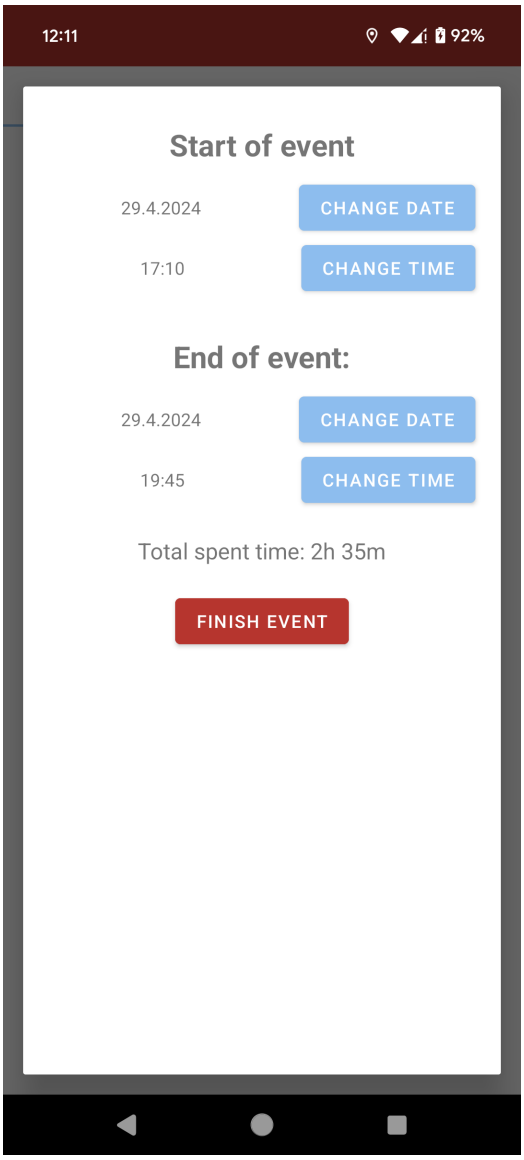


Figure 4.14: Finishing the event

After clicking the „FINISH EVENT“ button, the event is successfully finished and moved to the FINISHED state, where it is no longer visible on the home screen. Finishing the event calculates experience points for each member who participated in the event. Additionally, statistics are incremented for each member for the type of event, the role they participated in the event, and their total time spent in events (see Subsection 4.5).

Urgent event and emergency creation

An urgent event differs from a regular event in its red background color (see Figure 4.15), creation process, and interaction. When creating an urgent event, there is no need to specify the address, date, and time. Each created urgent event is immediately switched to the ONGOING state, preventing users from freely joining the event. Only the creator of the urgent event can add members to the urgent event they created, however this action is done automatically.

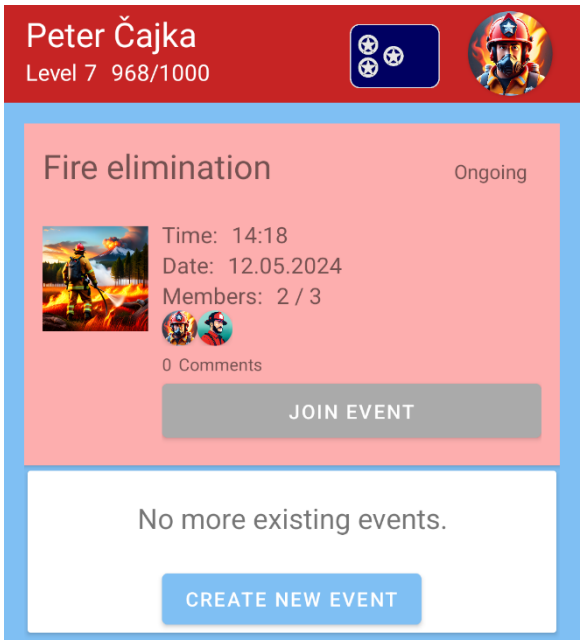


Figure 4.15: Urgent event

An emergency can be declared and ended only by an authorized member, such as a commander. Creating an urgent event is a prerequisite for starting an emergency. Although the FireReadyGo application allows multiple emergency declarations simultaneously by different members, this situation is highly unlikely. We can therefore assume, that under normal operation, there will be a maximum of one emergency and one urgent event.

At the bottom of the home screen, there is a large red button labeled SOS (see Figure 4.16). Pressing this button directs the user to the urgent event creation screen (see Figure 4.17), which resembles the non-urgent event creation screen. However, in this case, the user does not enter the date and time of the event or the address. The only data that must be chosen are the type of event and the vehicles.

Figure 4.16: Bottom menu

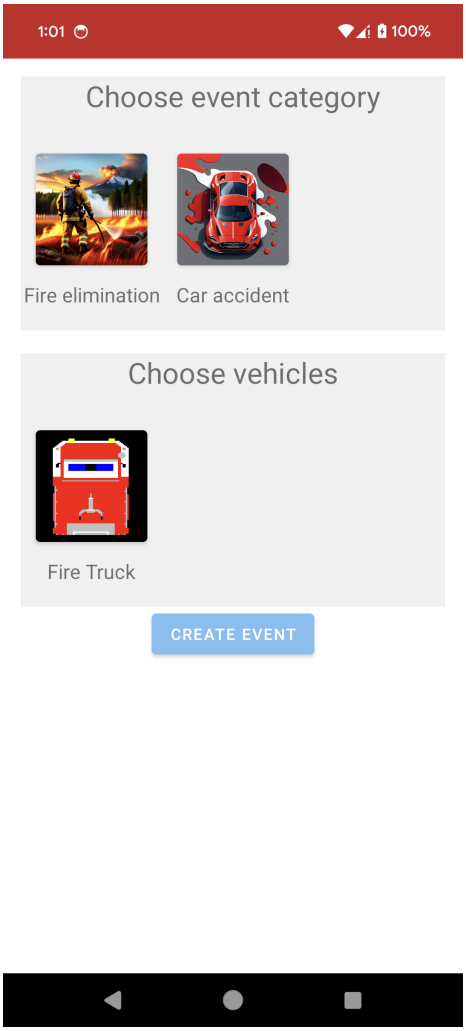
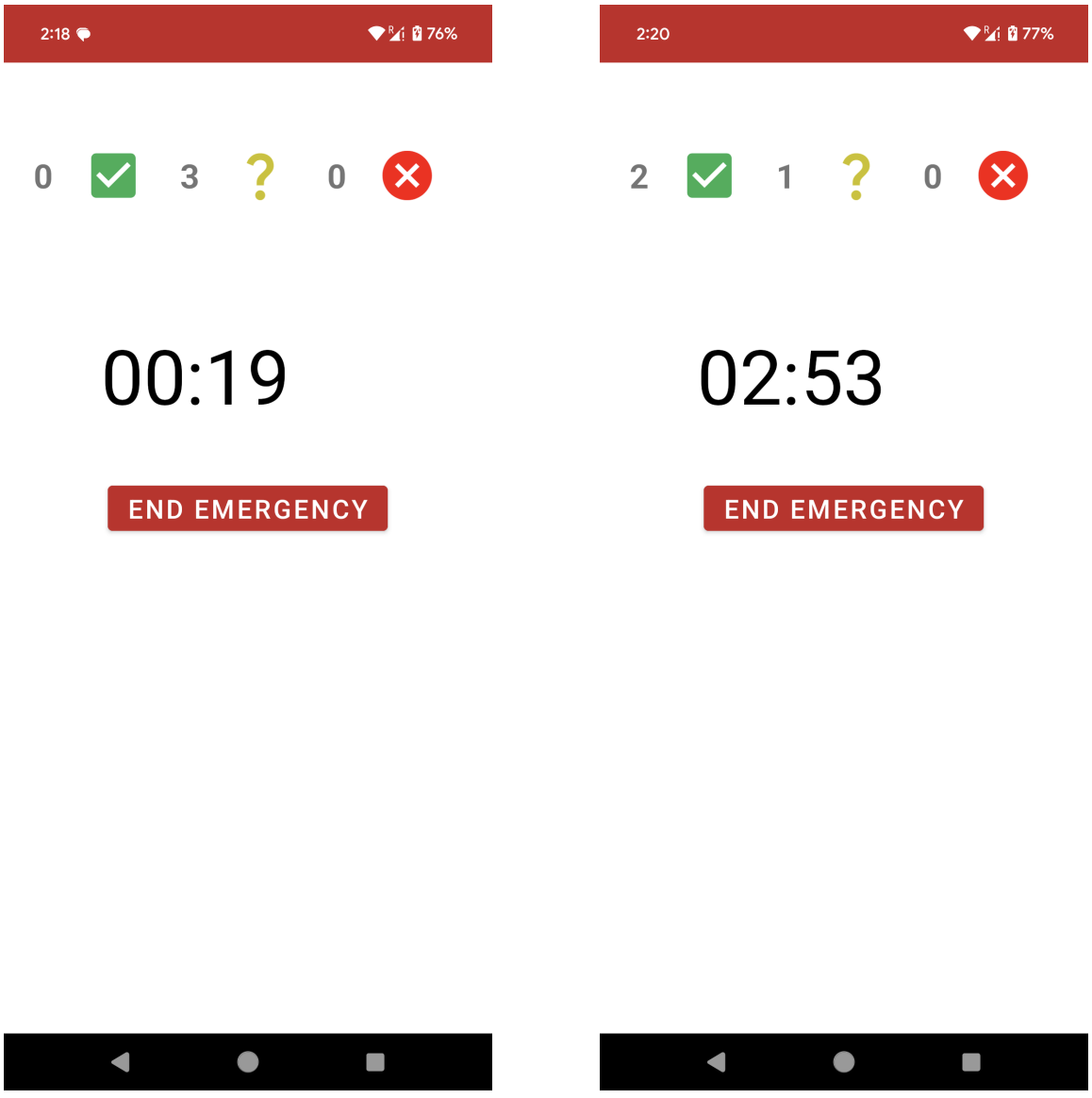


Figure 4.17: Creation of an urgent event

By creating an urgent event, the emergency state is temporarily activated. When the emergency state is initiated, SMS messages with preset text are sent from the phone of the user who activated the emergency state. The user who activated the emergency state is directed to a new screen where they can monitor the status of the fire department members. Initially, all members who received the SMS messages are in a waiting state, indicated on the screen by a yellow question mark. If a user confirms their participation in the urgent response through a generated notification, they are automatically included in the created event and moved to the category with a green check mark in the emergency screen. If users decline participation, they are moved to the category with a red cross.

In addition to monitoring availability via SMS messages, the activation of emergency status also starts a timer. Volunteer fire departments in Slovakia have a set time limit from the declaration of an emergency within which they must conduct a fire response. The time limits vary depending on the classification of the fire department. The activated timer serves only as a visual aid for the chief commander.

The ongoing emergency screen, displayed at Figure 4.18, is visible only to the user who created the emergency. Emergency state related data is securely stored locally on the device, so turning off the app does not result in data loss. Responses to emergency SMS message from members of the fire department are parsed in the background, so the app can also be turned off in this case. New responses are synchronized continuously at 10-second intervals.



(a) Emergency without responses

(b) Emergency with some responses

Figure 4.18: Monitoring of member's responses during emergency

Urgent event and emergency interaction

When users receive an emergency SMS message, the background service checks if sender is valid member of the department and if so, it creates a notification and activates a loud alarm. Part of the code that is responsible for checking the SMS is shown at Figure 4.19. The notification contains a brief description and two options. The user can either accept or reject the notification. Both actions generate an SMS message that is sent as a response to the member who triggered the emergency status. Part of the code that sends response SMS is displayed at Figure 4.20.

```
for (smsMessage in Telephony.Sms.Intents.getMessagesFromIntent(intent)) {
    if (smsMessage.messageBody == SMSMessages.URGENT_MESSAGE) {
        Log.d(TAG, msg: "Checking if SMS is from valid sender")
        val userViewModel =
            ViewModelProvider(
                ViewModelStore(),
                ViewModelProvider.AndroidViewModelFactory.getInstance(context.applicationContext as Application)
            )[UserViewModel::class.java]

        userViewModel.isValidSender( senderNumber: smsMessage.originatingAddress ?: "") { isValid ->
            if (isValid) {
                Log.v(TAG, msg: "Creating SMS urgent event notification!")
                smsMessage.originatingAddress?.let { it: String
                    val serviceIntent = Intent(context, SMSForegroundService::class.java)
                    serviceIntent.putExtra(Arguments.PHONE_NUMBER, smsMessage.originatingAddress)
                    ContextCompat.startForegroundService(context, serviceIntent)
                }
            } else {
                Log.d(TAG, msg: "SMS is not from valid sender!")
            }
        }
    }
```

Figure 4.19: Code snippet - SMS validation

```
    Peter Čajka *
    private fun sendSMS(
        context: Context,
        phoneNumber: String,
        message: String,
        notificationId: Int
    ) {
        if (phoneNumber == "")
            return
        try {
            val smsManager = this.getSystemService(SmsManager::class.java)
            smsManager.sendTextMessage(phoneNumber, scAddress: null, message, sentIntent: null, deliveryIntent: null)
            val notificationManager =
                context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
            notificationManager.cancel(notificationId)
            Toast.makeText(context, context.getString(R.string.sms_sent), Toast.LENGTH_SHORT).show()
        } catch (e: Exception) {
            Toast.makeText(context, context.getString(R.string.sms_was_not_sent), Toast.LENGTH_SHORT).show()
            e.printStackTrace()
        }
    }
```

Figure 4.20: Code snippet - Sending SMS response

A notification should be created even if the smartphone screen is locked. The siren sound lasts approximately 10 seconds at maximum volume. After the siren sound ends, the volume is automatically reset to its original level.

If a member who triggered the emergency receives a response SMS from other members, they will update their emergency screen and simultaneously assign members to the created emergency event. Since the emergency event switches to the ONGOING state immediately after creation, no member will have the option to join the created event through the usual button (see Figure 4.15). Only the creator of the emergency event can add members to the emergency event. Upon receiving the SMS, the phone number is extracted and verified to determine who owns it. The member with the given phone number is then assigned to the emergency event.

4.4 Fire department management

Authorized users may view and modify different screens of the fire department. The management of the fire department is spread across three tabs in total.

Users

The USERS tab contains a list of all members of the volunteer fire department. Each member is displayed with their name, avatar, rank, current level, and roles. Clicking on any user will open their profile. Figure 4.21 shows a list of all existing users in the volunteer fire department.

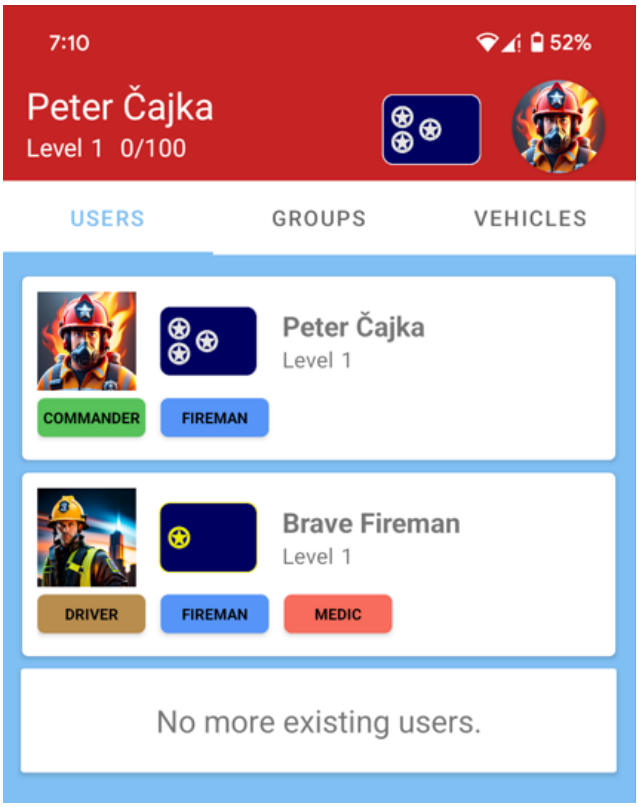
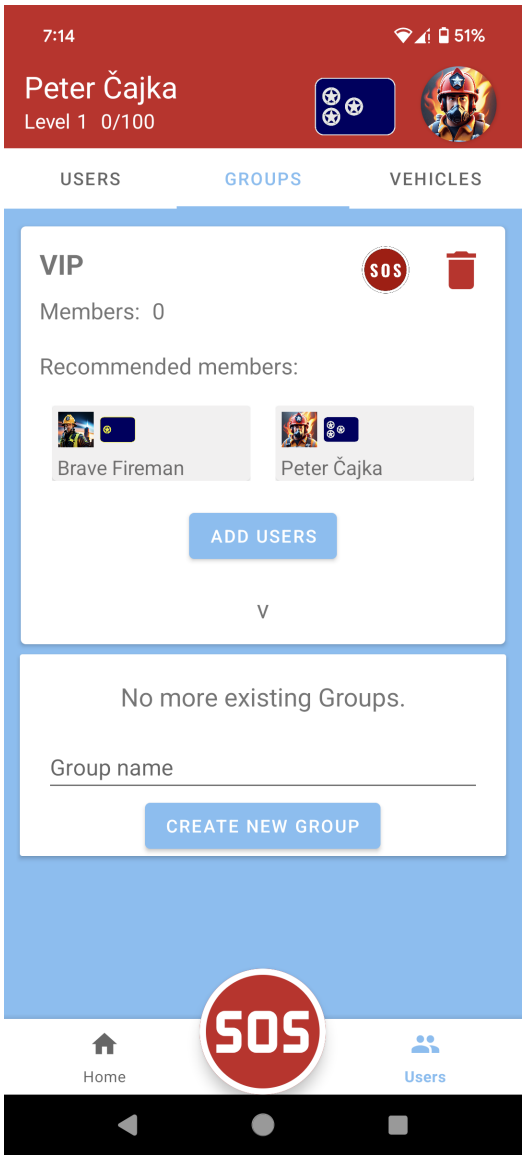


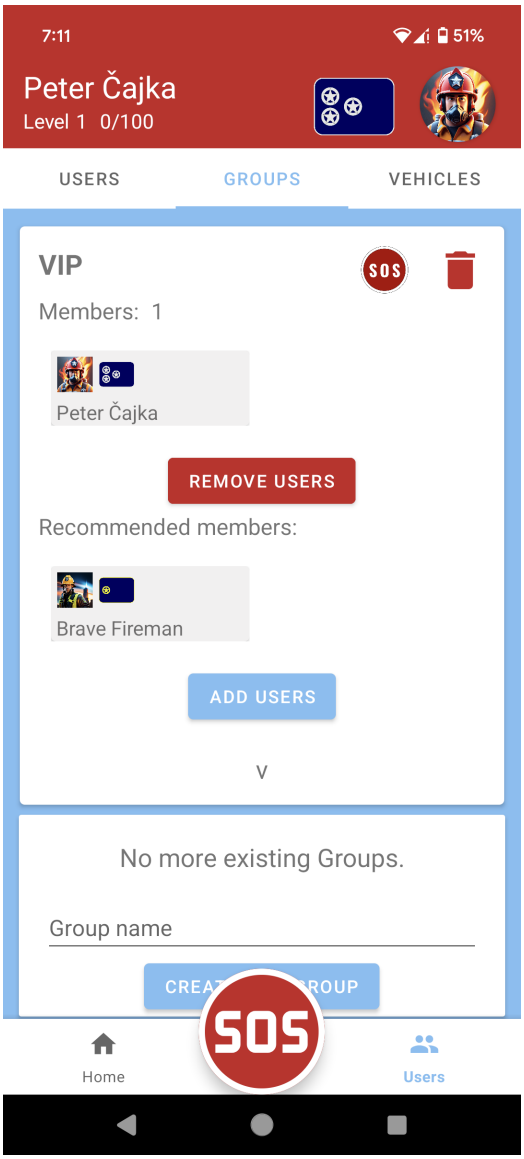
Figure 4.21: All members in volunteer fire department

Groups

The GROUPS tab contains a list of groups. Only authorized users can create and delete groups. Creating a group is very straightforward. The user must enter a group name that is at least 3 characters long and then press the „CREATE NEW GROUP“ button. After creation, a group is empty. Verified users can add members to the group from the list of recommended members (see Figure 4.22a). If a group contains any members, authorized users have the ability to remove members from the group (see Figure 4.22b). By default, a group is not marked as urgent. Only an authorized user can mark a group as urgent by clicking on a small SOS icon. Difference between urgent and non urgent groups is displayed at Figure 4.23. All members in urgent groups will receive emergency SMS message when an emergency state is declared.



(a) Empty group



(b) Group with member(s)

Figure 4.22: A comparison of empty group and group with member(s)

(a) Non-urgent group

(b) Urgent group

Figure 4.23: A comparison of urgent and non-urgent group

Vehicles

Authorized users are able to create new and edit existing vehicles. By clicking the „ADD NEW VEHICLE“ button (see Figure 4.27), the user is directed to a new screen where they must fill in the vehicle's name, maximum capacity of people, and maximum volume of water that can be transported. The last required step is to upload a custom photo of the vehicle from the gallery. The photo is uploaded to internet storage. Before uploading, the photo is adjusted to avoid taking up unnecessarily large amounts of space in the storage. Figure 4.24 shows a code, which transforms loaded image to 1024x1024 pixel size and does the compression.

```
val inputStream = contentResolver.openInputStream(imageUri)
val bitmap = BitmapFactory.decodeStream(inputStream)

// Resize the bitmap to desired dimensions
val resizedBitmap = Bitmap.createScaledBitmap(bitmap, dstWidth: 1024, dstHeight: 1024, filter: false)

// Compress the bitmap to reduce size
val outputStream = ByteArrayOutputStream()
resizedBitmap.compress(Bitmap.CompressFormat.JPEG, quality: 80, outputStream)
```

Figure 4.24: Code snippet - image compression

Additionally, each new image of a vehicle is versioned. The vehicle maintains information about the current version, which increments with each uploaded image. This allows for previous images of the vehicle to also be available in the online storage. This solution was necessary due to the way how caching functions in the Glide library works[4]. Glide is used in various parts of the FireReadyGo application for downloading images. While intelligent caching prevents unnecessary repeated downloading of images from online storage, it also creates issues when replacing an image with another. During the conducted analysis, it was found that caching in the Glide library is linked to the URL of the image. The initial idea for changing the vehicle images was to retain the URL and simply change its content by uploading a new image. However, this approach led to multiple unsuccessful attempts, with different Glide configurations, and the loading of images in the application did not behave as expected. A simple and effective solution was therefore to always use a unique URL for each image. Figure 4.25 illustrates the creation of a unique image URL consisting of the vehicle name, which is unique, and the image version. A minor disadvantage of this

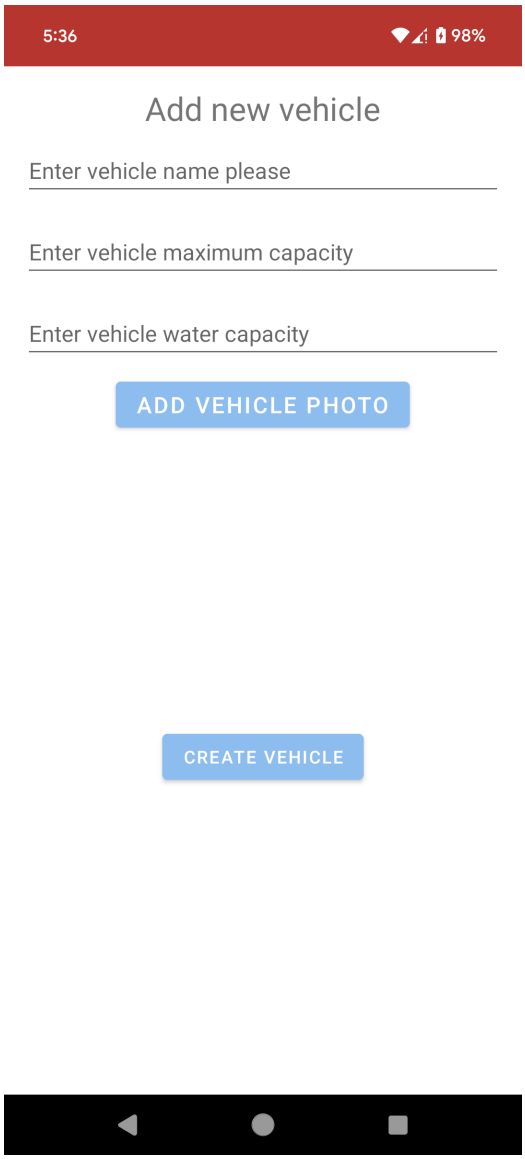
[4] https://github.com/bumptech/glide

approach is the duplicate storage of identical images under different versions. However, the nature of the application does not anticipate frequent changes of images.

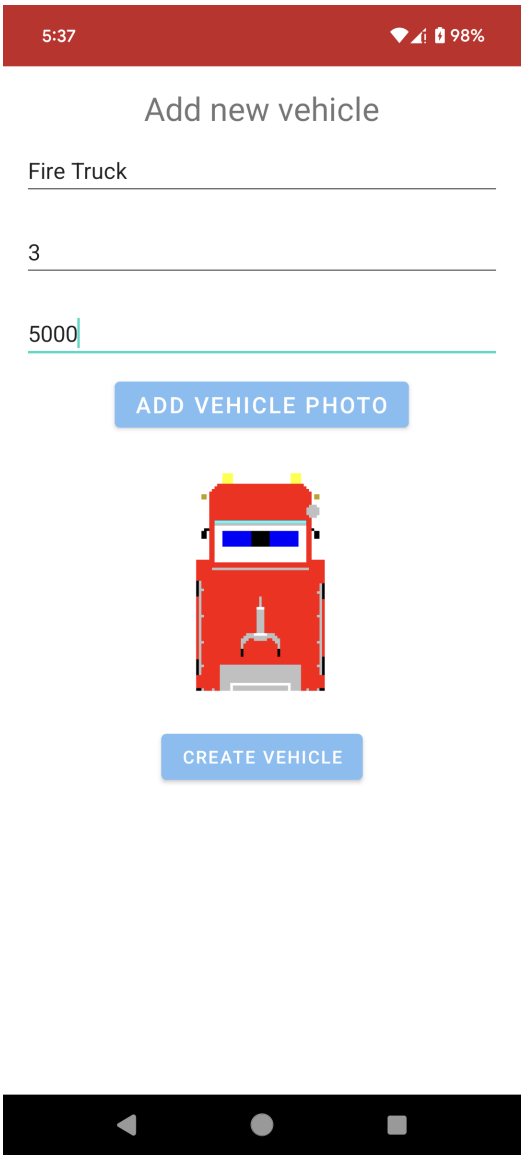
```
Peter Čajka *
private fun uploadImageToFirebase(imageUri: Uri, imageTitle: String, vehicle: Vehicle): Boolean {
    vehicle.version += 1
    val storageRef = FirebaseStorage.getInstance().reference
    val imageRef = storageRef.child( pathString: "images/$imageTitle${vehicle.version}")
    // Rest of the code...
```

Figure 4.25: Code snippet - unique image URL creation

Example of the new vehicle creation process is shown at Figure 4.26.



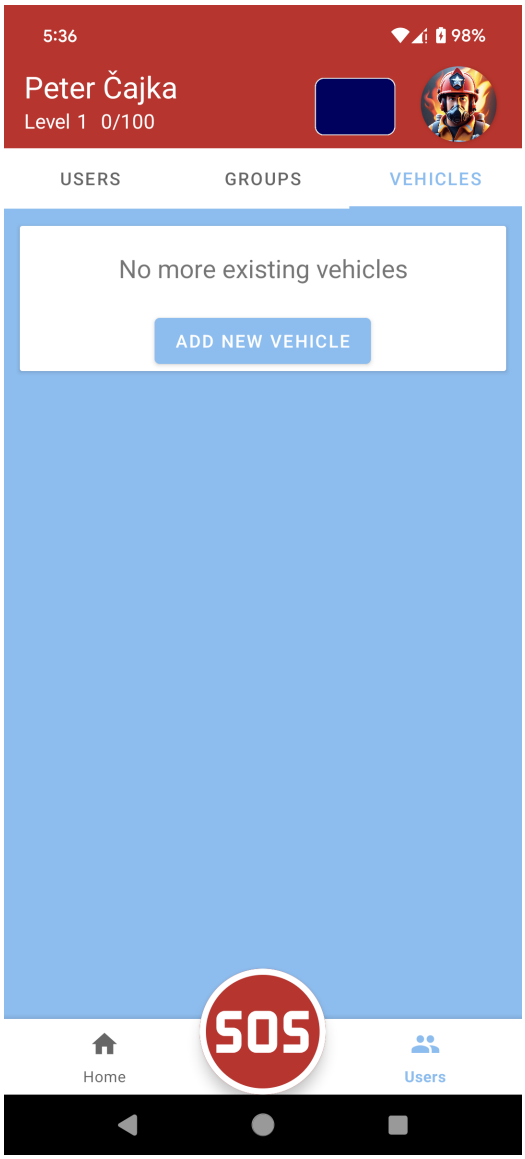
(a) Empty forms



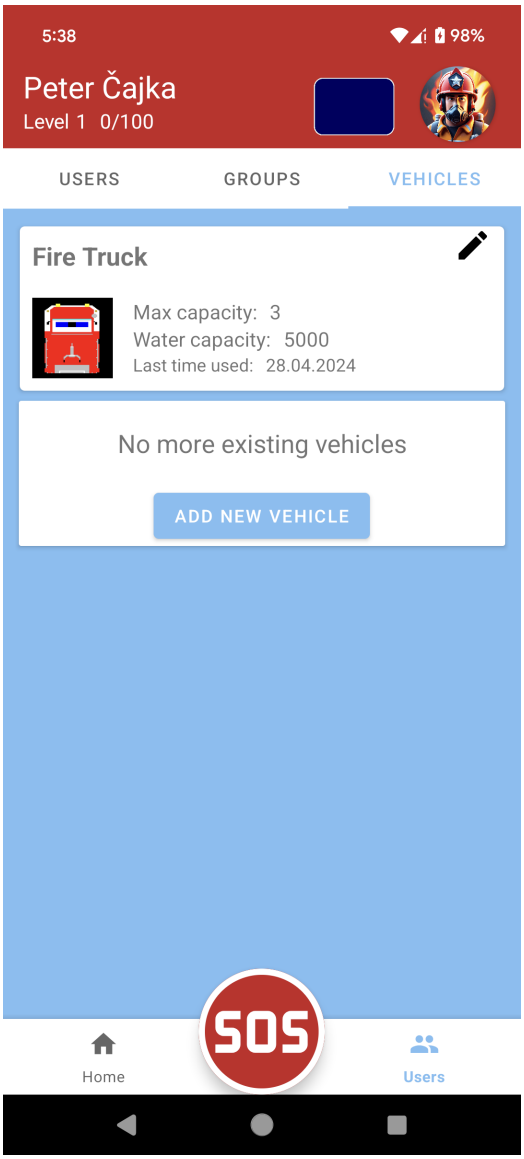
(b) Correctly filled forms

Figure 4.26: Create new vehicle

If all required fields are filled out and the vehicle image is properly uploaded, the created vehicle is saved to the database and automatically added to the list of available vehicles in the department.



(a) No existing vehicles



(b) List of all existing vehicles

Figure 4.27: Comparison of an empty and non empty vehicle list

4.5 Profile

Users can access their own profile from almost any part of the application by clicking on the avatar in the right part of the top toolbar (see Figure 4.28).

Figure 4.28: Toolbar

The second way to open their own profile and the profiles of other users is through the USERS tab in the fire department management menu (see 4.21). Only authorized users can view the profiles of other members. Every profile consists of two tabs: Details and Statistics.

Details

In the profile details (see Figure 4.29a) is available a complete summary of the member, including their name, roles, authorization level, rank, and avatar. At the bottom of the screen, there are settings that only the member, to whom the profile belongs, can configure. In the top right corner of the screen, there is a button to edit the profile. Only authorized users can edit any profile. By entering edit mode, the visibility of the screen changes, and new buttons become available. In edit mode (see Figure 4.29b), users can add or remove roles, change the current rank, and authorization level.

Statistics

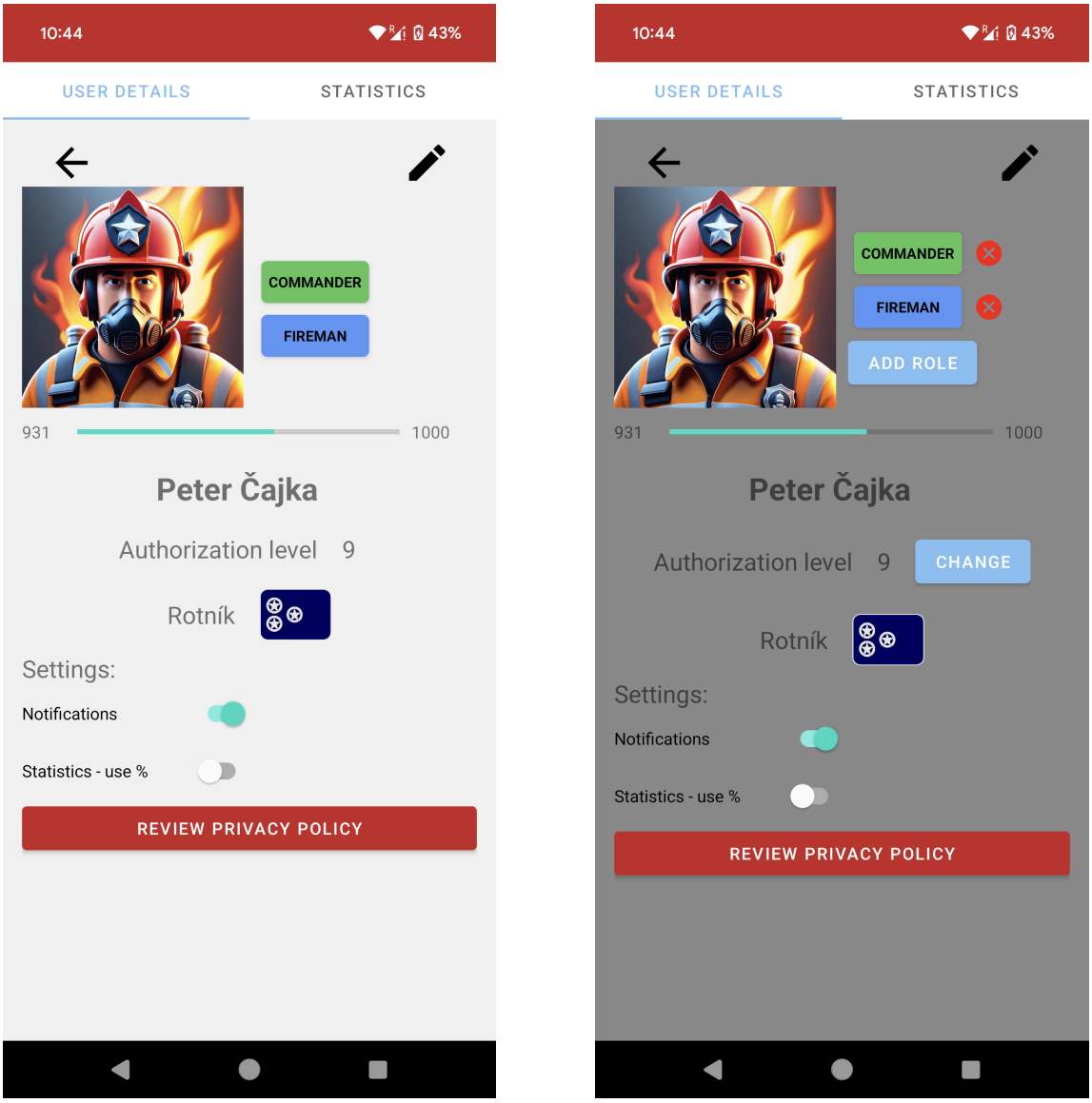
In the statistics tab, there is a summary of the user's overall activity. If the user has participated in any events, they will see pie charts displayed. MPAndroidChart[5] library is being used for pie chart creation and manipulation. This library offers many configuration options so developers may create various pie charts to fulfill their needs. When creating a dataset that forms the foundation of the entire pie chart, choosing the right color template is crucial. An incorrect choice of color template caused an error during development, which was found in the second testing phase (see subsection 5.2). Figure 4.30 shows the method with the correct settings for pie chart datasets to meet the needs of the FireReadyGo application.

After each finished event, user's statistics are improved (see subsection 4.3). In total, there are three pie charts on the screen:

- Distribution of events by role
- Distribution of non-urgent events by category
- Distribution of urgent events by category

At the bottom of the screen, there is text description of the member's registration date, date of the most recent event they joined and their total time spent in all events. Figure 4.31 displays difference between empty and non-empty statistics tabs.

[5] https://github.com/PhilJay/MPAndroidChart



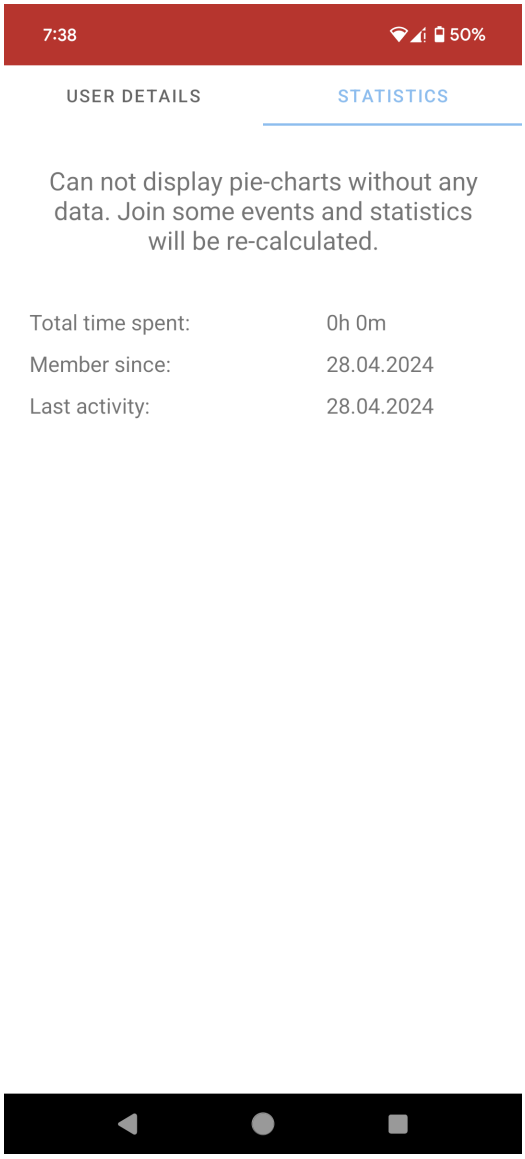
(a) User details

(b) User details in edit mode

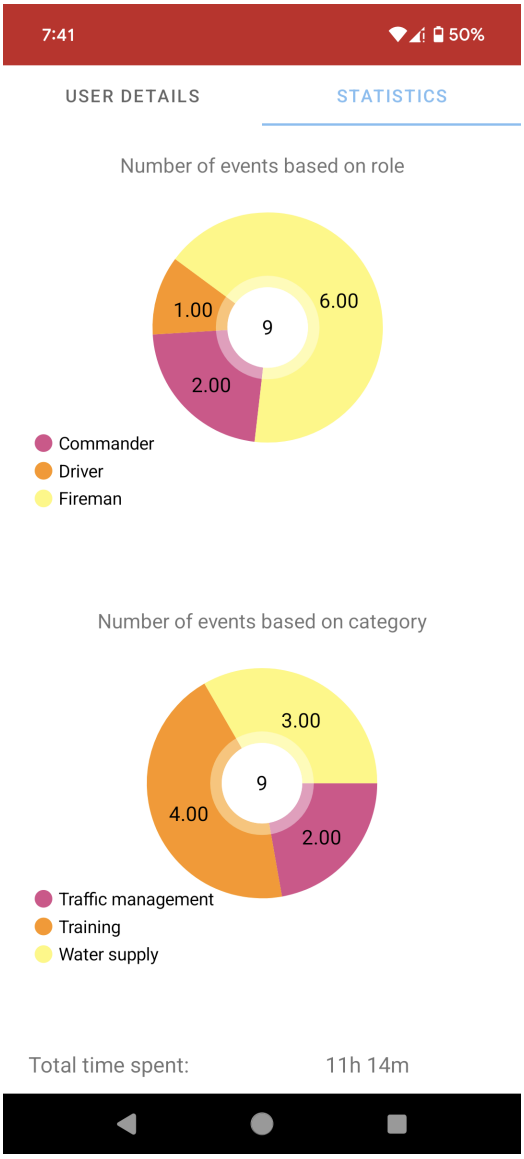
Figure 4.29: Comparison of event details screens

```
Peter Čajka
private fun createDataset(entries: MutableList<PieEntry>): PieDataSet{
    // Create a dataset with the data
    val dataSet = PieDataSet(entries, label: "")
    dataSet.setColors(*ColorTemplate.JOYFUL_COLORS)
    dataSet.valueTextColor = Color.BLACK
    dataSet.valueTextSize = 16f
    return dataSet
}
```

Figure 4.30: Code snippet - configuration of a dataset for pie charts



(a) No data to compute pie charts



(b) Existing pie charts

Figure 4.31: Comparison of an empty statistics screen and screen with pie charts

4.6 Location

Location tracking begins upon receiving a Firebase server push notification about the start of an event, or when the user opens the map of the ongoing event they are part of. A location background service collects location data every 10 seconds, and when distance between consecutive locations is at least 10 meters, (see Figure 4.32) and saves it to the database. Each user is assigned a random color. The current location of members is displayed as a large circle in the generated color with an avatar image in the center. Previous locations are displayed as smaller circles of the same color. The older the location data, the lower the visibility of the displayed circles (see Figure 4.34). Figure 4.33 displays an example of the location tracking service during an event. In this case, location of the two users is being

recorded and observed. One user (with red color circle) remains on the same spot. Second user (with purple color circle) changes his location. Each location is displayed as a marker on the map. Users may interact with markers. After clicking on a marker, a small label with the name of the user whose location was measured will be displayed, along with the approximate distance and time from the user's current location to the selected marker. In the top part of the screen, the user can choose the method of calculating distance and time according to the mode of transportation. The entire calculation of the distance between two points is provided by Google's Directions API [6]. The portion of the code utilizing the Directions API is displayed in Figure 4.35.

The collection of location data is stopped upon receiving a Firebase push notification about the finish of an event. In case of notification failure, the FireReadyGo application is contains additional startup check, which suspend any potentially running location tracking service if the user is not part of any ongoing event.

```
val request = LocationRequest.Builder(interval)
    .setMinUpdateDistanceMeters(10f)
    .setWaitForAccurateLocation(true)
    .setPriority(Priority.PRIORITY_HIGH_ACCURACY)
    .build()
val locationCallback = object : LocationCallback() {
    override fun onLocationResult(result: LocationResult) {
        super.onLocationResult(result)
        result.locations.lastOrNull()?.let { it: Location
            launch { this: CoroutineScope
                send(it)
            }
        }
    }
}
```

Figure 4.32: Code snippet - Getting the current location

[6] https://developers.google.com/maps/documentation/directions/overview

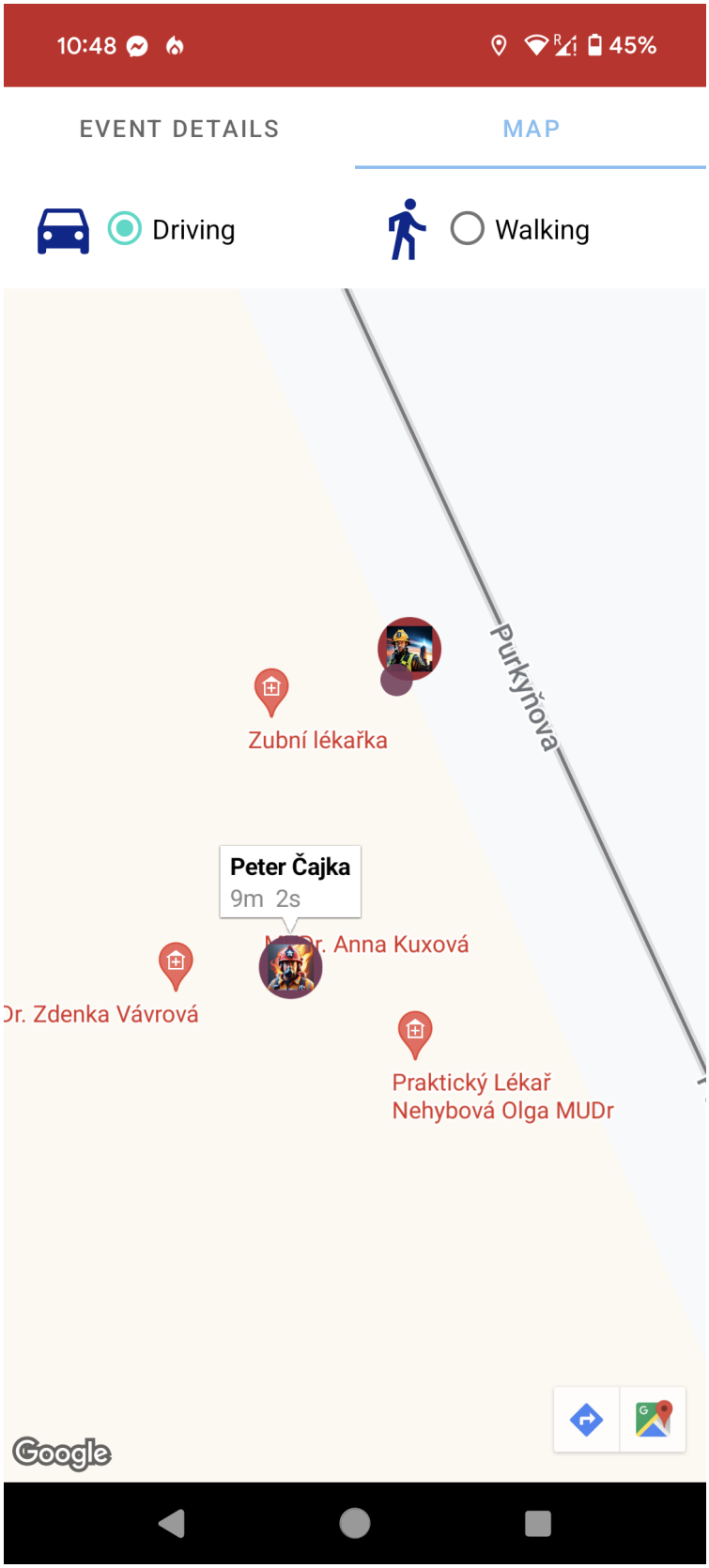


Figure 4.33: Map and location tracking

```
val latLng = LatLng(location.latitude, location.longitude)
val markerOptions = MarkerOptions().position(latLng).title(userViewModel.getUserName(userId))

// Last index should be bigger circle with an icon in the middle
if (index == 0) {
    // If it's the latest location (first in the sorted list), add the icon
    markerOptions.icon(
        BitmapDescriptorFactory.fromBitmap(
            createIconWithCircle(
                Utils.getProfilePictureId( imageTitle: userIconsMap[userId] ?: ""),
                userColorMap[userId]!!,
                 alpha: 250
            )
        )
    )
} else {
    // If it's not the latest location, set color point with varying opacity
    val alpha = (1 - (index.toFloat() / 10)) * 250
    markerOptions.icon(
        BitmapDescriptorFactory.fromBitmap(
            createIconWithCircle(
                 iconResId: null,
                userColorMap[userId]!!,
                alpha.toInt()
            )
        )
    )
}

this.googleMap.addMarker(markerOptions)
```

Figure 4.34: Code snippet - Creation of a location circles

```
Peter Čajka *
private suspend fun computeTravelDetails(
    origin: String,
    destination: String,
    mode: String,
    apiKey: String
): Pair<String, String>? {
    val response = directionsApi.getDirections(origin, destination, mode, apiKey)
    return if (response.routes.isNotEmpty()) {
        val leg = response.routes[0].legs[0]
        val formattedDistance = formatDistance(leg.distance.value)
        val formattedDuration = formatDuration(leg.duration.value)
        Log.d(TAG,  msg: "Computed distance: $formattedDistance, duration: $formattedDuration")
        Pair(formattedDistance, formattedDuration)
    } else {
        Log.d(TAG,  msg: "No route was found from: $origin to $destination")
        null
    }
}
```

Figure 4.35: Code snippet - Computation of a distance between two points

Chapter 5

Testing

Testing is an integral part of software development. Repeated testing helps ensure higher quality software. Although a developer can test most of the software on their own, they will never achieve perfect results. No developer is infallible, and no one can create perfect, flawless software. Great idea is to involve a group of potential users of the final product in the testing process. Users bring a fresh, non-technical perspective to testing and may uncover issues that the developer had not considered [5].

Since the FireReadyGo application was not publicly available on Google Play during development, an alternative distribution method was used during testing. On the day of testing, an installation file with the latest changes was created and uploaded to a specific GitHub[1] repository. A QR code was created from the address where the installation file was uploaded, which the testers scanned. They were then redirected to a website where they downloaded the installation file and installed the application.

5.1 Phase 1

On the 16th of March, 2024, the first phase of testing the FireReadyGo application took place. At that time, the application was still under development and was estimated to meet approximately 60% of the expected use cases. Some important features that were implemented at that time:

- Verification and registration process
- Server push notifications - when event is created, started and finished
- Event creation - users are able to create time-scheduled events
- Event modification - users are able to edit and delete events
- Event interaction - users are able to join some existing events, and change their assigned vehicle and role
- Location tracking - not fully implemented
- Group and Vehicle management
- User profile - with possibility to change user roles

[1] https://github.com/

A total of 5 individuals/members of a volunteer fire department participated in the testing, including the developer. The individuals participated in the testing voluntarily and varied across different categories such as age, gender, and technical knowledge. Out of the total of 5 testers, there were two women and three men. Two respondents were of middle age, while three were young adults.

Each tester used a different phone. During the testing, some deficiencies were identified, primarily related to the design and display of elements on the screen. In the following part of this section, the revealed deficiencies will be described in more details and categorized according to the severity of the issue.

Critical

1. The creation of a new vehicle resulted in an immediate application crash. The error was reproducible with every attempt to create a new vehicle if the user did not insert a photo of the vehicle.
2. Some parts of the application were not visible in dark mode. An example is shown at figure 5.1 The error was reproducible with every application launch in dark mode.

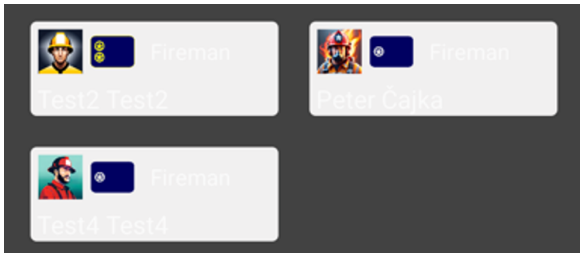


Figure 5.1: Unreadable text in dark mode.

Major

1. The number of participants who can attend an event is calculated based on the characteristics of vehicles assigned to the event. This calculation method caused two issues:
   - It was possible to create an event without a vehicle, which meant that the created event had a maximum allowed number of members of 0. No member could register for this event. An example is shown at figure 5.2. The error was reproducible consistently.
   - During an event that had a maximum allowed number of members, in some cases, it was possible to exceed the specified limit. An example is shown at figure 5.3. The error was not always reproducible since the application partially checked the filled capacity.
2. Differences in screen sizes and various settings across devices caused significant visual discrepancies in certain cases. In some instances, insufficient responsive design of the application led to certain elements being inadequately visible or extending beyond the visible screen area. This complicated the interaction with the application, especially concerning interactive elements such as buttons. The error was reproducible consistently on certain smartphones and with specific system settings.

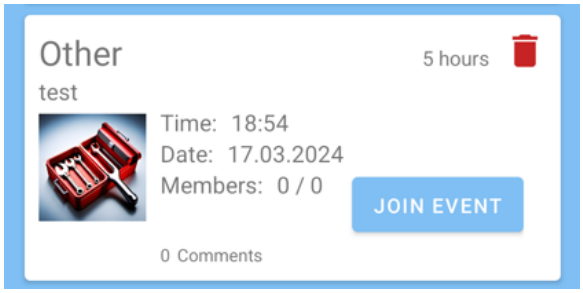


Figure 5.2: Event with maximum number of members 0.

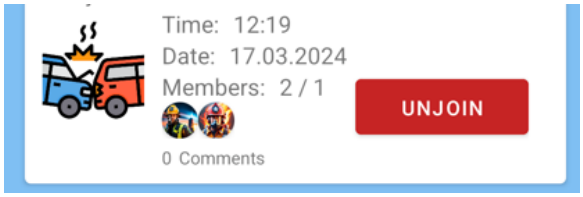


Figure 5.3: Event with exceeded maximum number of members.

Minor

1. Issues related to responsive design, which, however, did not affect the functionality of the application but rather slightly worsened the user experience. An example is shown at figure 5.4. The error was reproducible consistently on certain smartphones and with specific system settings.

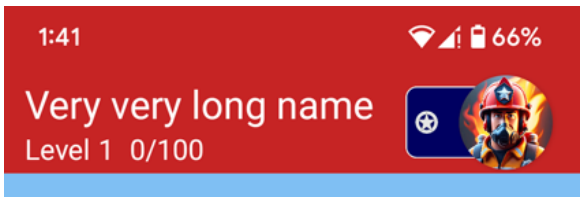


Figure 5.4: Responsive design issue.

2. The error message displayed to the user when attempting to sign up for an event was overly generic. The issue stemmed from the fact that the user did not have the required role for the event. However, the error message did not specify the exact cause of the login failure. The error was reproducible consistently.

3. Several missing translations into the Slovak language. The error was reproducible consistently when using the application in the Slovak language.

Known issues

This subsection describes already known issues and features, that were not implemented, but discussed during first phase of testing. Biggest issue was probably the location tracking service, since it was started and stopped externally for multiple members at the same time. Obtained location was not always accurate.

5.2 Phase 2

On the 20th of April, 2024, the second phase of testing the FireReadyGo application took place. At that time, the application was still under development and was estimated to meet approximately 85% of the expected use cases. Compared to the first testing phase, the following features have been added to the application:

- Urgent groups - groups of members may be set as urgent.
- SMS parsing - service running in the background parses incoming messages and checks if messages were generated by the FireReadyGo during an urgent event.
- SMS notification - notification generated when parsed SMS has correct format.
- Custom sound of the notification - when SMS notification is created, loud siren effect is played for approximately 10 seconds.
- SMS sending - authorized user sends predefined SMS messages to all users in urgent groups.
- Urgent events - event categories were divided into non-urgent and urgent.
- User statistics - each successfully finished event improves user's statistics that are displayed in the second tab of the profile.

Similar to the first testing phase 5.1, the same 5 testers participated in the testing again. Each tester used a different phone. During the testing, some deficiencies were identified. In the following part of this section, the revealed deficiencies will be described in more details and categorized according to the severity of the issue.

Critical

1. The registration screen was repeatedly displayed at the start of the application, even in cases where the registration was successfully completed. Error occurred only on one tested device and could not be reproduced.
2. Creating an event and then joining the event caused a fatal crash of the application. Error was always reproducible on all devices when an event with some joined medics existed.

Major

1. Notification generated by the SMS receiver (see figure 5.5) did not send response SMS message when user pressed accept or decline action button. This error was always reproducible.
2. Two smartphones did not generate an SMS notification with a loud siren sound after receiving a preset SMS message. The problem was reproducible when the phones had the screen locked. After granting additional permissions in the phone settings, the issue was resolved.

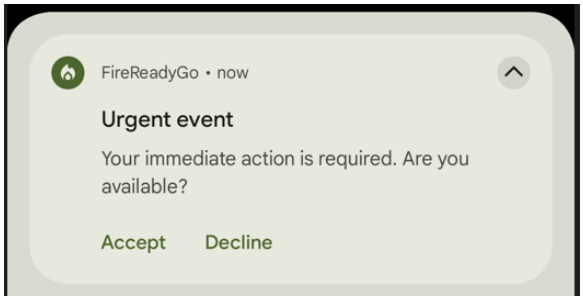
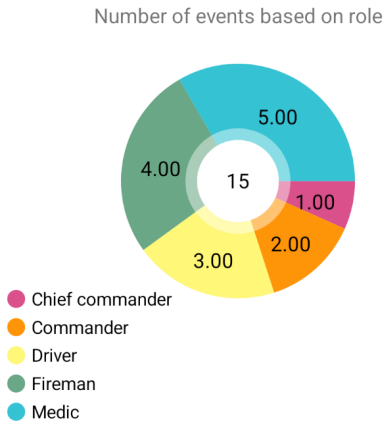


Figure 5.5: Urgent notification response button bug.

Minor

1. Pie charts in user statistics tab were able to display maximum 4 categories. Pie charts have become unclear. Figure 5.6a shows a situation where a volunteer fire department member participated in events in 5 different roles, but only the first 4 roles are displayed in the statistics. Figure 5.6b shows the same situation as Figure 5.6a, but with correct rendering of all categories. This error was always reproducible.

(a) Pie chart defect.



(b) Correct pie chart behavior.

Figure 5.6: Comparison of incorrect and correct pie chart

2. Inaccurate names of some parts of the application in the Slovak language. This error was always reproducible.

Chapter 6

Conclusion

The goals of this thesis have been achieved. FireReadyGo is a fully functional application that has been thoroughly tested on various devices throughout its development. Its greatest advantage lies in real-time communication between devices and instant updates, even for minor changes. Compared to competing products, FireReadyGo offers the following features:

- Non urgent events
- Real-time location tracking during events
- Custom statistics and overview of member's activity

Informing members of the department about an emergency works similarly to competing products. An emergency is triggered upon receiving an SMS. Users are notified through a displayed notification and a loud sound signal of a siren, even if their phone is currently idle. The displayed notification with 2 response options enables users to promptly respond and send information to the commander regarding their participation or absence.

6.1 Future work

While the FireReadyGo application is ready for practical use, it is not perfect. The application will require ongoing development to improve its quality. A major part of the ongoing development will involve enhancing the appearance of certain screens within the application to ensure a higher level of user-friendliness. Another significant aspect, mainly for the developer, will be refactoring the current code and writing unit tests to ensure software maintainability in the long run.

Some parts of the application were not implemented, mainly because their priority during application development significantly decreased. One example is part of the functional requirement *„Users with insufficient permissions should not have the ability to create urgent events. Instead, when attempting to create an urgent event, the application should display contact information for authorized individuals or emergency services to these users.“* (see Figure 3.1).Users with a low authorization level are not allowed to create urgent events. However, they will not be shown any contact information for commanders. Another example of an unimplemented feature would be comment section under created events. Comments are not crucial to the application, but adding this feature would be beneficial. It will make the application more independent and reduce reliance on other communication platforms gradually.

FireReadyGo was not published on the official Android application repository - Google Play Store as part of this thesis. However, it will likely be published in the future, as distributing applications outside of Google Play is very inefficient.

Bibliography

[1] Almomani, I. M. and Khayer, A. A. A Comprehensive Analysis of the Android Permissions System. *IEEE Access*. 2020, vol. 8, p. 216671–216688, [cit. 2024-01-13]. DOI: 10.1109/ACCESS.2020.3041432.

[2] Bapty, R. *Mobile Network Experience Report* [online]. [cit. 2023-12-30]. Available at: https://www.opensignal.com/reports/2023/03/slovakia/mobile-network-experience.

[3] Buck, A. *10 Steps to Ensure Your Mobile App Meets Gdpr Compliance Standards* [online]. [cit. 2023-12-31]. Available at: https://www.mobiloud.com/blog/gdpr-compliant-mobile-app.

[4] Cartier, C. *Field Guide to the Android Manifest File* [online]. [cit. 2024-01-13]. Available at: https://www.blackhillsinfosec.com/field-guide-to-the-android-manifest-file/.

[5] Chopra, R. *Software Testing: A Self-Teaching Introduction.* Mercury Learning and Information, 2018 [cit. 2024-04-22]. ISBN 9781683921677.

[6] Chugh, A. *Android Layout - LinearLayout, RelativeLayout* [online]. [cit. 2024-01-12]. Available at: https://www.digitalocean.com/community/tutorials/android-layout-linearlayout-relativelayout.

[7] Counterpoint, T. *Global Smartphone Sales Share by Operating System* [online]. [cit. 2023-12-03]. Available at: https://www.counterpointresearch.com/insights/global-smartphone-os-market-share/.

[8] Daniel, B. *Symmetric vs. Asymmetric Encryption: What's the Difference?* [online]. [cit. 2024-01-13]. Available at: https://www.trentonsystems.com/blog/symmetric-vs-asymmetric-encryption.

[9] Dorfer, T., Demetz, L. and Huber, S. Impact of mobile cross-platform development on CPU, memory and battery of mobile devices when using common mobile app features. *Procedia Computer Science*. 2020, vol. 175, p. 189–196. DOI: https://doi.org/10.1016/j.procs.2020.07.029. ISSN 1877-0509. The 17th International Conference on Mobile Systems and Pervasive Computing (MobiSPC),The 15th International Conference on Future Networks and Communications (FNC),The 10th International Conference on Sustainable Energy Information Technology. Available at: https://www.sciencedirect.com/science/article/pii/S1877050920317099.

[10] Fan, M., Yu, L., Chen, S., Zhou, H., Luo, X. et al. An Empirical Evaluation of GDPR Compliance Violations in Android mHealth Apps. In: *2020 IEEE 31st*

International Symposium on Software Reliability Engineering (ISSRE). 2020, p. 253–264 [cit. 2023-12-31]. DOI: 10.1109/ISSRE5003.2020.00032.

[11] Farooq, M. S., Riaz, S., Alvi, A., Ali, A. and Rehman, I. U. Cross-Platform Mobile Development Approaches and Frameworks. *VFAST Transactions on Software Engineering*. May 2022, vol. 10, no. 2, p. 79–93. DOI: 10.21015/vtse.v10i2.978. Available at: https://vfast.org/journals/index.php/VTSE/article/view/978.

[12] Gadhavi, M. *Most Popular Backend Frameworks for Web Development in 2024* [online]. [cit. 2023-12-26]. Available at: https://radixweb.com/blog/best-backend-frameworks.

[13] Google. *Android fundamentals 02.2:Activity lifecycle and state* [online]. [cit. 2024-01-13]. Available at: https://developer.android.com/codelabs/android-fundamentals-02-2-activity-lifecycle-and-state#0.

[14] Gorbachenko, P. *What are Functional and Non-Functional Requirements and How to Document These* [online]. [cit. 2024-01-14]. Available at: https://enkonix.com/blog/functional-requirements-vs-non-functional/.

[15] government, U. *Control Segment* [online]. [cit. 2023-12-30]. Available at: https://www.gps.gov/systems/gps/control/.

[16] government, U. *The Global Positioning System* [online]. [cit. 2023-12-30]. Available at: https://www.gps.gov/systems/gps/.

[17] government, U. *Space Segment* [online]. [cit. 2023-12-30]. Available at: https://www.gps.gov/systems/gps/space/.

[18] Griffiths, D. *Head First Android Development 2e*. Sebastopol, CA: O'Reilly Media, september 2017 [cit. 2024-01-12].

[19] H., G. *HTTP, WebSocket, gRPC or WebRTC: Which Communication Protocol is Best For Your App?* [online]. [cit. 2023-12-31]. Available at: https://getstream.io/blog/communication-protocols/.

[20] He, A. *People Continue to Rely on Maps and Navigational Apps* [online]. [cit. 2023-12-30]. Available at: https://www.insiderintelligence.com/content/people-continue-to-rely-on-maps-and-navigational-apps-emarketer-forecasts-show.

[21] Koffer, P. *What is a Native Mobile App Development?* [online]. [cit. 2023-12-03]. Available at: https://mdevelopers.com/blog/what-is-a-native-mobile-app-development.

[22] Lemonaki, D. *Frontend VS Backend – What's the Difference?* [online]. [cit. 2023-12-25]. Available at: https://www.freecodecamp.org/news/frontend-vs-backend-whats-the-difference/.

[23] Meier, A. and Kaufmann, M. *SQL amp; NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*. Springer Fachmedien Wiesbaden, 2019. ISBN 9783658245498. Available at: http://dx.doi.org/10.1007/978-3-658-24549-8.

[24] MongoDB. *What is a Document Database?* [online]. [cit. 2023-12-26]. Available at: https://www.mongodb.com/document-databases.

[25] Monus, A. *The 2023 guide to native app development* [online]. [cit. 2023-12-03]. Available at: https://raygun.com/blog/native-app-development/.

[26] Ontotext. *What is a NoSQL Graph Database?* [online]. [cit. 2023-12-26]. Available at: https://www.ontotext.com/knowledgehub/fundamentals/nosql-graph-database/.

[27] Poggi, N. *Types of Encryption: Symmetric or Asymmetric? RSA or AES?* [online]. [cit. 2024-01-13]. Available at: https://preyproject.com/blog/types-of-encryption-symmetric-or-asymmetric-rsa-or-aes.

[28] Pramanik, P. K. D., Sinhababu, N., Mukherjee, B., Padmanaban, S., Maity, A. et al. Power Consumption Analysis, Measurement, Management, and Issues: A State-of-the-Art Review of Smartphone Battery and Energy Usage. *IEEE Access*. 2019, vol. 7, p. 182113–182172, [cit. 2023-12-30]. DOI: 10.1109/ACCESS.2019.2958684.

[29] Rome, P. *What are Non Functional Requirements — With Examples* [online]. [cit. 2024-01-14]. Available at: https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples.

[30] Savonin, M. *Native vs. Cross-Platform Apps: Analysis 2023* [online]. [cit. 2023-12-10]. Available at: https://keenethics.com/blog/cross-platform-vs-native.

[31] Scylla. *Wide-column Database* [online]. [cit. 2023-12-26]. Available at: https://www.scylladb.com/glossary/wide-column-database/.

[32] Selleo. *How To Meet GDPR Requirements In Mobile And Web Applications* [online]. [cit. 2023-12-31]. Available at: https://selleo.com/blog/how-to-meet-gdpr-requirements-in-mobile-and-web-applications.

[33] Sharma, R. K. *How To Protect Data In Mobile Web Apps Using Encryption* [online]. [cit. 2023-12-31]. Available at: https://www.netsolutions.com/insights/how-to-protect-data-in-mobile-web-apps-using-encryption/#how-to-protect-data-in-mobile-web-apps-using-encryption.

[34] Sullivan, A. *Examining performance differences between Native, Flutter, and React Native mobile development.* [online]. [cit. 2023-12-10]. Available at: https://thoughtbot.com/blog/examining-performance-differences-between-native-flutter-and-react-native-mobile-development.

[35] Team, C. *Back-End Web Architecture* [online]. [cit. 2023-12-26]. Available at: https://www.codecademy.com/article/back-end-architecture.

[36] Tierney, P. and Clarke, N. A Comparison of a Smartphone App with Other GPS Tracking Type Devices Employed in Football. *Exercise Medicine*. Sapientia Publishing Group. april 2019, vol. 3, p. 4, [cit. 2023-12-30]. DOI: 10.26644/em.2019.004. ISSN 2508-9056. Available at: http://dx.doi.org/10.26644/em.2019.004.

[37] Turner, A. *HOW MANY SMARTPHONES ARE IN THE WORLD?* [online]. [cit. 2023-12-03]. Available at: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world.

[38] Williams, A. *NoSQL Database* [online]. [cit. 2023-12-26]. Available at: https://hostingdata.co.uk/nosql-database/.

[39] yadav, R. K. *Android Activity Lifecycle In detail* [online]. [cit. 2024-01-13]. Available at: https://medium.com/@ranjeet123/android-activity-lifecycle-in-detail-eaf2931a1b37.

Appendix A

Contents of the attached media

FireReadyGo Mobile application source code

FireReadyGo.apk Installation file for the FireReadyGo application

FirebaseFunctions Source code of the Firebase functions

README.pdf Description of an application including few basic use cases.

xcajka00.pdf This document

xcajka00 Source code required to build this document