



Western Norway  
University of  
Applied Sciences

BACHELOR THESIS

# **A Mobile Application for Fire Risk Notification based on Edge Computing**

*Authors:*

Emilie Hinna Fisketjøn, Abu Tallaha Hussain, Thorbjørn Svendal

*Supervisors:*

Prof. Lars Michael Kristensen, Ph.D. candidate Ruben Dobler Strand

Department of Computer science, Electrical engineering, and Mathematical sciences

Western Norway University of Applied Sciences

May 23, 2022

## TITTELSIDE FOR HOVEDPROSJEKT

<i>Rapportens tittel</i>  A Mobile Application for Fire Risk Notification based on Edge Computing	<i>Dato:</i>  23.05.2022
<i>Forfatter(e):</i>  Emilie H. Fisketjøn, Abu Tallaha Hussain Thorbjørn Svendal	<i>Antall sider u/vedlegg: 89</i>
<i>Studieretning:</i>  Dataingeniør, Informasjonsteknologi	<i>Antall sider vedlegg: 7</i>
<i>Kontaktperson ved studieretning:</i>  Lars Michael Kristensen	<i>Gradering:</i>  Ingen
<i>Merknader:</i>	

<i>Oppdragsgiver:</i>  DYNAMIC	<i>Oppdragsgivers referanse:</i>
<i>Oppdragsgivers kontaktperson:</i>  Ruben Dobler Strand	<i>Telefon:</i>  48 35 00 63

*Stikkord:*

Fire Risk	Edge Computing	DYNAMIC
-----------	----------------	---------

Høgskulen på Vestlandet, Fakultet for ingeniør- og natuvitskap

Postadresse: Postboks 7030, 5020 BERGEN

Besøksadresse: Inndalsveien 28, Bergen

Tlf. 55 58 75 00

Fax 55 58 77 90

E-post: [post@hvl.no](mailto:post@hvl.no)

Hjemmeside: <http://www.hvl.no>

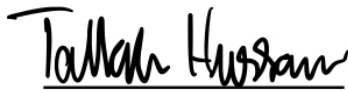
# Preface

This bachelor project is a thesis administrated by The Department of Computer science, Electrical engineering, and Mathematical sciences at Western Norwegian University of Applied Sciences. This project would allow us to contribute to the DYNAMIC Research Project which works with reducing fire disaster risk through dynamic risk assessment and management. We chose this project because we thought it was socially beneficial and at the same time would provide a good opportunity for us to acquire new skills in development.

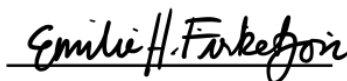
We would like to thank the project supervisors Professor Lars Michael Kristensen and Ph.D. candidate Ruben Dobler Strand for their excellent guidance and constructive feedback throughout the project.

We would also like to express our gratitude to:

- Torgrim Log for developing the Fire Risk Model and his involvement in the project for his input on the design and evaluating the application.
- Maria-Monika Metallinou Log for evaluating the application and the facilitation of the user testing in Haugesund.
- Everyone else involved in user testing, such as representatives from the Fire Safety Engineering Department at Western Norwegian University of Applied Sciences, Haugesund Fire Brigade, Haugaland Brann og Redning, Bergen Fire Brigade, and finally friends and family



Abu Tallaha Hussain



Emilie Hinna Fisketjøn



Thorbjørn Svendal

Bergen, May 23, 2022

# Abstract

Authorities and other organizations may find it difficult to determine which areas are more probable of fire. There are technologies that can predict the fire risk for specific areas, such as MET's forest fire index, however there is no technology available to the general population that provides risk assessment for fire in wooden structures.

We have therefore developed a mobile application by using a cross-platform framework to make it available for mobile phones running on different operative systems. The application's main purpose is to monitor locations for fire risk by requesting weather data from MET and derive fire risk by performing calculations using the weather data. The application will provide users an overview of fire risks in wooden structures for locations they are interested in. The application has been evaluated through user testing and performance assessments; this has helped developing an application with a high degree of usability that meets the users' expectations.

This thesis centres around implementing and extending the accessibility of the Fire Risk Model which calculates fire risk based on weather conditions and is developed by the DYNAMIC Research Project.

# Table of Content

- Chapter 1: Introduction..... 1**
  - 1.1 The DYNAMIC Research Project..... 1
  - 1.2 Fire Risk Prediction Model ..... 2
  - 1.3 Motivation ..... 2
  - 1.4 Problem statement and objective..... 3
  - 1.5 Research Questions ..... 3
  - 1.6 Thesis structure ..... 5
  
- Chapter 2: Project Description ..... 6**
  - 2.1 Project owner..... 6
  - 2.2 Related work ..... 6
  - 2.3 Initial requirements specification ..... 7
  - 2.4 Initial solution idea..... 9
  - 2.5 Resources ..... 9
  
- Chapter 3: Project Approach..... 11**
  - 3.1 Possible cross-platform mobile frameworks ..... 11
    - 3.1.1 Flutter ..... 11
    - 3.1.2 Xamarin..... 11
    - 3.1.3 .NET Multi-Platform App User Interface (MAUI)..... 12
  - 3.2 Assessment of Technology Choice ..... 12
  - 3.3 Development method ..... 14
  - 3.4 Project plan..... 14
    - 3.4.1 Pre-Project Phase..... 15
    - 3.4.2 Development Phase..... 15
    - 3.4.3 Testing..... 15
    - 3.4.4 Finishing Phase ..... 15
  - 3.5 Risk management ..... 16
  - 3.6 Evaluation plan..... 16
  
- Chapter 4: Detailed design ..... 18**
  - 4.1 Use Case Diagram..... 18

4.1.1	Use Case Descriptions.....	19
4.2	Domain Model.....	20
4.3	Application Design Pattern .....	21
4.4	High-Level Application Software Architecture .....	22
4.5	Fire Risk .....	24
4.5.1	Fire risk benchmarking.....	24
4.6	Weather Data Sources .....	25
4.6.1	The Norwegian Meteorological Institute .....	25
4.6.2	The Weather API Location Forecast Service .....	26
4.6.3	The Frost API .....	28
4.6.4	Authorization.....	32
4.7	Data Models .....	32
4.8	Service Clients.....	34
4.8.1	RestSharp .....	36
4.8.2	Exception Handling.....	37
4.9	Application Flow .....	40
4.10	SQLite Database.....	41
4.11	Fire Risk Model.....	44
4.12	Graphical User Interface (Views) .....	48
4.12.1	Pages .....	48
4.12.2	Layouts.....	48
4.12.3	Views .....	49
4.12.4	Cells .....	49
4.13	Graphical User Interface Structure.....	49
4.13.1	Tab navigation .....	49
4.13.2	Items Source.....	50
4.13.3	Data Template.....	50
4.13.4	Expander .....	51
4.13.5	Data Binding and Data Type.....	51
4.13.6	Fire Risk and Wind .....	52
4.13.7	Building the User Interface.....	52
4.14	Data Updates and Notifications.....	54
<b>Chapter 5: Evaluation.....</b>		<b>56</b>
5.1	Evaluation method.....	56
5.1.1	Fire Safety Group.....	57

5.1.2	Laypeople Group.....	58
5.1.3	Fire Brigade Group.....	58
5.2	User testing results .....	58
5.2.1	Fire Safety Group .....	59
5.2.2	Laypeople Group.....	59
5.2.3	Fire Brigade Group.....	59
5.2.4	Post evaluation .....	59
5.3	Performance evaluation .....	60
5.3.1	Data Usage .....	60
5.3.2	Storage.....	62
5.3.3	CPU and memory .....	62
5.3.4	Battery .....	65
<b>Chapter 6: Discussion .....</b>		<b>67</b>
<b>Chapter 7: Conclusion .....</b>		<b>72</b>
7.1	Research Questions Revisited .....	72
7.2	Conclusion.....	73
7.3	Future Work .....	74
<b>Figures.....</b>		<b>75</b>
<b>Listings .....</b>		<b>76</b>
<b>Tables.....</b>		<b>77</b>
<b>Bibliography .....</b>		<b>78</b>
<b>Appendices .....</b>		<b>82</b>
9.1	Gantt Chart .....	83
9.2	Risk Management.....	84
9.3	Risk Matrix.....	85
9.4	User Testing Guide.....	86
9.5	SUS results .....	87
9.6	Wireframe.....	88

# Terminology

---

DYNAMIC	Reducing fire disaster risk through dynamic risk assessment and management is an interdisciplinary research project focusing on fire risk predictions related to wooden homes and the wildland urban interface.
FMC	Fuel Moisture Content
IFD	Indoor Fire Development
TTF	Time To Flashover

---



# Chapter 1: Introduction

Forest fires and other catastrophic fires are now typical news headlines. Many models have been created that can predict – and so provide warnings about the risk of fires. Most of these models cover only wildfire risk, they are primarily used by municipalities and fire brigades to assess wildfire danger. There are approximately 300 000 annual fire-related deaths worldwide [1], with residential fires accounting for the majority of these deaths. This demonstrates the necessity for technologies that can notify users of fire risks in their homes, which is what this bachelor project seeks to implement.

## 1.1 The DYNAMIC Research Project

This project has been undertaken in the context of the research project, Reducing fire disaster risk through *dynamic risk assessment and management (DYNAMIC)*. DYNAMIC is an interdisciplinary research project at Western Norwegian University of Applied Sciences (HVL) focusing on reducing conflagration risk, through dynamic risk predictions and management [2]. The project emphasizes wooden home fire risk and fires in the wildland-urban interface, herein the cultural heathland along the Norwegian west coast.

After a devastating winter fire blaze in Lærdalsøyri (Norway), on 18-19 January 2014, it was brought to attention that cold climate fire risk in wooden homes was under-examined. Therefore, it was seen necessary to conduct further research on this topic [3]. It was suggested that indoor humidity could be a fire risk indicator [4]. A cold climate structural fire danger rating system [1] was propounded and a mathematical model for predicting indoor relative humidity and wooden fuel moisture content (FMC) was developed [5].

## **1.2 Fire Risk Prediction Model**

A wooden home fire risk model was developed by Log [5]. The model predicts indoor relative humidity and wood fuel moisture content (FMC) of indoor wooden coverings, based on measures and predictions of outdoor relative humidity and temperature. The modelled FMC is correlated with the time to flashover (TTF) [6]. TTF is the transition time between the growth period to the fully developed stage in fire development, which indicates untenable conditions [7]. TTF is typically in the order of minutes and decreases with decreasing FMC.

## **1.3 Motivation**

The motivation underlying this bachelor project is to raise awareness of fire and mitigate the consequences of a probable fire. This will be accomplished by notifying users for fire risk when appropriate so that they can take preventative measures. Example of organizations that could be interesting in having real-time access to fire risk are: The fire brigade, municipalities, and volunteer organizations. Knowing ahead of time which places are at risk allows the appropriate authorities to devote more resources to these places.

A major motivator for this project is the lack of consumer-grade tools for assessing fire risk. As of today, there is no technology or product that provides real-time fire risk predictions for wooden structures. The smoke detector is as of now the most utilized fire risk detection tool for homes. The disadvantage of relying exclusively on this technology is that the user will have little time to react, as the smoke detector will only react seconds before disaster.

## **1.4 Problem statement and objective**

As is evident from the DYNAMIC project, there is a clearly defined societal need for access to fire risk assessments. Several candidate fire risk models and software technology solutions have emerged that may potentially be able to bridge the current gap, but they need to be systematically assessed and evaluated in the context of available weather data sources (weather measurement and forecasts).

The objective of the bachelor project is to investigate how a fire risk notification service may be provided to users via a mobile application using an approach based on edge computing, i.e., where the acquisition of weather data and the fire risk computation are performed without relying on cloud services.

## **1.5 Research Questions**

The project is centred around the three following research questions and will be discussed in more detail below.

**RQ1: Can a user interface for a mobile application be designed capable of showing fire risks for multiple locations and be suited for users with different background and use cases?**

**RQ2: What is the impact of the fire risk computation and weather data acquisition on the performance of the mobile device?**

**RQ3: What are the advantages and disadvantages of an approach based on edge computing on mobile devices in comparison to a cloud-based solution?**

## **RQ1**

The desired outcome of this project is to create a mobile application that can be used as a personal tool by the general public. This decision was made by the project owner to increase the accessibility of the modelling results. The idea is that the application will be used daily by users of different backgrounds, thus it must be developed in a way that appeals to the majority. It needs to be taken into consideration how to visually present both the current fire risk and changes to the fire risk. The final solution should be simple to use and comprehend. To achieve this, evaluations will be conducted in which users will be asked questions about their experience with the user interface.

## **RQ2**

In terms of computational capability and specs, the average mobile device today is quite advanced. When developing a mobile application such as this, one of the main concerns is how it will affect the end user's mobile device. As a result, it will be necessary to investigate the implications of computing fire risk directly on a mobile device. Both storage and battery capacity need to be addressed, as an application is expected to not consume too many resources. To address this, the technologies that will be utilized to develop the application will be thoroughly examined, and the application's performance will be assessed on both an iOS and an Android device.

## **RQ3**

Both approaches have advantages and disadvantages; edge computing may be better suited for a certain task where cloud computing is ineffective, and vice versa. The advantages and disadvantages will be investigated in relation to the application to research whether edge computing is a practical approach.

## 1.6 Thesis structure

It is assumed that the reader has a technological background and some understanding of the terms and technologies covered in this thesis.

The bachelor thesis is divided into the following chapters:

**Chapter 2** delves into the project owner's background, previous work, and other resources. This chapter will explain how these resources will benefit the project. The reader will learn about the project's initial solution idea, constraints, and resources.

**Chapter 3** covers the plan for carrying out the project. The plans for project design, evaluation, and technology will be delved into. A technology evaluation will be conducted to determine which technology is best suited for the project's requirements.

**Chapter 4** provide the reader with a detailed description of the systems requirements, structure, and behaviour. The mobile application's solution and design choices will be highlighted. The reader should have a good understanding of how the application's backend and frontend systems work after reading this chapter.

**Chapter 5** delves into the evaluation. This chapter will cover the evaluation methods, a description of the test panel, and an analysis of the evaluation results.

**Chapter 6** discusses the result of the implementation and the obstacles that were encountered.

**Chapter 7** concludes and summarizes the work carried out in this bachelor thesis. The reader will learn about future work and other tasks that must be completed to ensure successful implementation.

# Chapter 2: Project Description

When starting this project, initial work had already been undertaken. An implementation of the model of Log [5] has been done earlier in the context of a research affiliated with the DYNAMIC research project [8] [9].

## 2.1 Project owner

The project owner is Ph.D. candidate Ruben Dobler Strand under the auspices of the DYNAMIC research project. DYNAMIC is the largest ongoing research project within the *Fire Disasters* [10] research group at HVL. The project is supported by the Norwegian Research Council and other partners.

The *Software Engineering* research group [11] at HVL is also a partner in the DYNAMIC project. Their research focus is in the areas of model-driven software engineering, concurrent and distributed systems, software verification and testing, software quality and security, code refactoring, and software architecture.

## 2.2 Related work

The DYNAMIC research project conducted a research in 2019 which led to the development of the model of Log [5]. The model focuses on the first house catching fire, potentially becoming a larger conflagration. The model calculates risk for fire based on the indoor relative humidity and transient drying of wooden wall panels.

Stokkenes investigated whether the model of Log [5] could be used in real-world situations to predict risk of conflagration using both historical weather data and forecast weather data [8]. This was the first implementation and evaluation of the model of Log [5]. A combination of historical weather data and forecast weather data was used to achieve a more accurate model

which can give precise predictions. The model was evaluated using data for four different locations: Bergen, Haugesund, Gjøvik, and Lærdal. This research concluded that the model of Log [5] can be used to give accurate indications for fire risk using a concept such as TTF [8].

A web-based software system was developed by Halderaker and Evjenth in 2021 [9]. This is a Java-based Cloud- and Microservice-based Software system for fire risk prediction. It contains five components that together constitute the system. The Fire Risk Model (FRM), Fire Risk Service (FRS), Data Harvesting Service (DHS), a front-end web application, and a middleware enabling communication between the FRS and the front-end.

The web service was evaluated in terms of performance and accuracy. Halderaker and Evjenth conducted an experiment that monitored the resource usage of the service. The required CPU usage needed to calculate fire risks proved to be negligible deeming this application to be efficient in terms of CPU performance [9]. The project also proved that weather forecast for up to three days ahead can be reliably used. This was done by comparing fire risks using weather forecasts and observations [9].

The main difference between this bachelor project and Halderaker and Evjenth's web-based software system [9] is that the application developed in this project will be explicitly developed for mobile devices. The idea of the project is to increase the availability to the fire risk prediction technology. It is anticipated that this project would result in the first implementation of the model of Log [5] available to the general public.

## **2.3 Initial requirements specification**

The initial requirement specification for the bachelor project was to create a mobile application that implements the fire risk model based on edge computing. Edge computing is a distributed computing architecture that brings applications closer to data sources. The distributed system offers an alternative way of processing data, and all decisions and computing occur on the mobile device itself. Thus, edge computing provides immediate data analysis [12]. In this setting, the mobile device is responsible for retrieving outdoor temperature, relative humidity, peak wind speed, and computing fire risks.

Besides edge computing, cloud computing is another architecture. There is a distinction between edge computing and cloud computing as these concepts endeavor to solve different network infrastructure issues. Cloud computing provides an on-demand availability of computing and storage resources [13]. The architecture provides minimum management, good data backup, and massive storage. It will suit businesses that need a scalable solution for storing and analyzing large amounts of data. The downside of this architecture is that it can result in business downtime because it requires a stable internet connection and bandwidth, it can therefore be perceived as less accessible.

The advantage of using edge computing is that the latency will be significantly reduced. The mobile device will not have to send the data to a cloud service, but will instead perform the calculations locally. This will lead to a faster response time and improve the customer's overall experience [14].

In addition to edge computing, one can consider using a proxy server that acts as a gateway between the client application and a real server [15]. A proxy server can request and communicate with different services on behalf of users by acting as a middleman. The proxy can improve the performance by introducing a concept called proxy caching [16]. Proxy caching stores content on the proxy server itself, making it possible to share resources between users. It handles all communication with servers and caches all the requested resources.

When a user tries to access a resource, the proxy will see if a recent copy is available. It will either deliver the data that has already been cached or it will request the data from the source, then cache it, and finally deliver it. The main advantage of proxy caching is that it can reduce bandwidth use. Another advantage is that it can improve website speed and reliability by providing a point of presence for several end users [17].

Another requirement was to develop a mobile application that is cross-platform compatible. The cross-platform concept stems from the growing necessity to generate mobile applications that reach out to as many followers as possible. A cross-platform application will cover a wide number of operating systems e.g., iOS, Android, and Windows. Applications written using cross-platform technology will usually share the same codebase. This significantly reduces both development costs and time, which allows for a quicker release.



## 2.4 Initial solution idea

Instead of employing native-platform technology, the application will be developed using cross-platform technology. This decision was taken after evaluating project efficiency and time. The goal is to create an application that allows users to monitor several locations for fire risk. These locations will be treated as objects that will store information about fire risk to the locations specified by the user.

It was suggested to display the fire risk by use of a colour bar as illustrated in Figure 1, with the colours computed by the TTF values. The lower the TTF value, the darker the colour. The colours in the colour bar correspond to the Norwegian forest fire index's colour system [18]. Consistency is achieved between the fire risk models by using the same colour system. It is an attempt towards avoiding confusion and misconception when people utilize both the forest fire index and the application.



Figure 1. Color scheme for displaying forest fire risk

## 2.5 Resources

The project is reliant on a collection of resources that will work together to realize the initial solution idea. Access to the RESTful MET API is necessary to obtain weather data [19].

As mentioned in Section 2.1.2 (Previous work), a Java-based Cloud- and Microservice-based Software system for Fire Risk Prediction was developed [9]. The software system used the model of Log [5] which has also been fully developed and implemented using Java. The same model altered to suit the requirement specifications will be used in this project.

The project will have access to Halderaker and Evjenth's [9] findings through the associated research paper. This has aided the project's progress for a variety of reasons. This project faces

many of the same challenges as Halderaker and Evjenth's project. This is because many of the challenges Halderaker and Evjenth encountered will be relevant for this project, as the systems to be developed are like the web-based software system.

Meetings that concerned all the project's participants were held on Zoom. In addition to being used as a digital meeting space, Microsoft Teams functioned also as a messaging and file sharing platform for the project group, supervisor, and project owner. Meetings were also held physically; however, due to the uncertainty regarding the Covid-19 pandemic, many meetings were held completely digital.

A private GitHub repository was set up by the project supervisor. It was used along with GitHub Desktop to work concurrently on the code for the application. GitHub provided version control; this is particularly useful to manage the source code and keep track of code modification. In the scenario of a faulty version, previous versions can quickly be accessed. This will prevent time wasted on debugging unnecessary errors.

In the scenario of technological or other programming related problems, the project's supervisor Lars Michael Kristensen was contacted for guidance. The project owner Ruben Dobler Strand was contacted when any problems occurred concerning the model of Log [5], or theoretical understanding of fire risk calculation.

# Chapter 3: Project Approach

There are many candidate technologies that could be used to develop a cross-platform mobile application. An assessment had to be made regarding which technology suited the project's needs best.

## 3.1 Possible cross-platform mobile frameworks

An initial requirement set by the project owner was that the application must be compatible with the two major mobile operating systems: Android and iOS. This criterion can be met with a variety of technologies. However, the group opted to dig deeper into the following frameworks: Flutter, Xamarin and .NET Multi-Platform App User Interface.

### 3.1.1 Flutter

Flutter is Google's open-source user interface framework. It is used for developing a modern and natively compiled mobile applications with only one codebase [20]. This implies that you can build cross-platform applications using one programming language.

Flutter uses Dart as its programming language, which is an object-oriented language. The syntax in Flutter is C-styled, implying it follows many of the same rules as other C-styled languages. Flutter offers a massive collection of built-in widgets. It also provides a set of animation designs, allowing developers to design an interactive application for their customers [21].

### 3.1.2 Xamarin

Xamarin is an open-source platform for developing modern and responsive cross-platform mobile applications. It is built on top of .NET, which automatically manages tasks such as memory allocation, garbage collection and interoperability with underlying platforms [22].

Xamarin uses eXtensible Markup Language (XAML), a declarative markup language developed by Microsoft. XAML is the visual presentation language for an application created using Xamarin and related technologies. Xamarin's programming language is C#, and offers direct invocation of Objective-C, C, C++ and Java libraries, allowing developers to use a wide range of third-party code [22].

### **3.1.3 .NET Multi-Platform App User Interface (MAUI)**

MAUI is an open-source framework used for developing modern and responsive cross-platform mobile applications. MAUI is a part of the .NET platform which gives the developer access to a wide array of features. MAUI uses XAML as its markup language. It is the language behind the visual presentation of an application built using MAUI and associated technologies.

Microsoft describes MAUI as the evolution of Xamarin because it adds desktop app support to the traditional iOS- and Android-focused development framework along with other enhancements [23].

## **3.2 Assessment of Technology Choice**

The cross-platform mobile application development frameworks introduced above have their benefits and drawbacks, the choice boils down to preference. Flutter would be preferred by developers who are more comfortable with Google technologies and frameworks, whilst Xamarin or MAUI would be preferred by developers who are more familiar with Microsoft technology.

An advantage of using Flutter would be the immediate access to the built-in widget system [24]. The main advantage of using Xamarin or MAUI is the access to .NET technologies. It provides the easy-to-use IDE Visual Studio. Visual Studio provides features such as a consistent programming model and advanced profiling tools to measure system performance. It also simplifies application deployment and maintenance [25]. Another advantage of using .NET and associated technologies is the fact that they are easy for newcomers to adapt to, especially if they are familiar with concepts within Object-Oriented programming.

The group members were also participating in a course which covers .NET and associated technologies, this led to the narrowing of choice to either Xamarin or MAUI. These frameworks have many similarities, such as the fact that both are based on object-oriented languages which use C-styled syntax. This implies that many of the features using either one of them are also available using the other.

A simple distinction between MAUI and Xamarin is that with MAUI the developer will be working on a single project that can target Android, iOS, macOS, and Windows, whereas Xamarin has different projects for each of the respective operating systems. Figures 2 and 3 display how a project for mobile application development would be set up using Xamarin or MAUI.

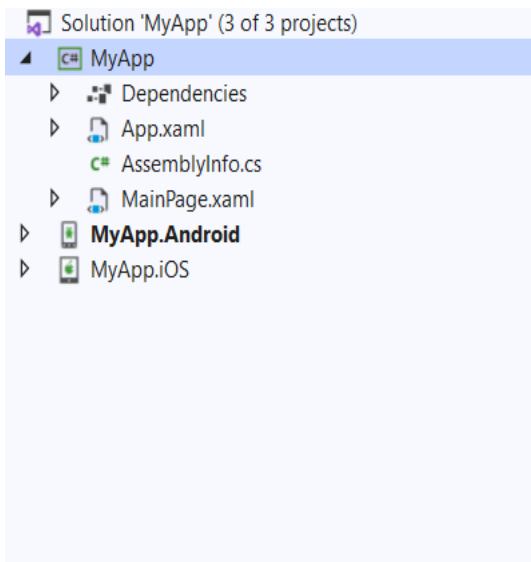


Figure 2. Xamarin project overview

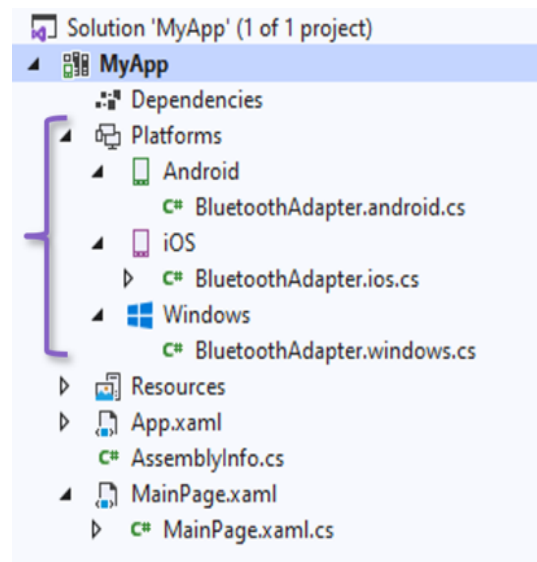


Figure 3. MAUI project overview

MAUI is not the most optimal technology to use at this date. The main reason is that MAUI is a newly released technology and there are many risks associated with using newly released technologies. An example of this is encountering a problem that cannot be solved at the given moment due to the lack of documentation. Aside from the lack of documentation, community support for a newly released technology may be inferior to that of well-established technologies like Xamarin and Flutter. These two issues are less critical for Xamarin because it has been well tested and many of the problems can be resolved with a quick Internet search.

It was decided that the mobile application will be developed using Xamarin and associated technologies. This decision was taken after assessing the project's two most significant constraints: time and experience.

A transition from Xamarin to MAUI must eventually be made, because Microsoft is planning to end updates to the Xamarin mobile application development platform in November 2022 [26]. Concerning the fact that MAUI will over time replace Xamarin, the probability of having to rewrite the Xamarin application to meet MAUI standards is seen as most improbable. The process of migrating the application is expected to be small amounts of code changes [27].

### **3.3 Development method**

Concepts from Agile Methodology have been used for project management. An Agile Methodology is a way to manage a project by breaking it up into several phases. It allows constant collaboration with stakeholders and continuous improvement for every phase since there will be a dialogue between the developers and stakeholders for each iteration.

The Agile technique Scrum was utilized for this project. Scrum allowed continuous communication and collaboration among the project members. Scrum describes a set of tools and ideas that helps structure the work of the project. Scrum was well-suited for this project because the technique laid the foundations for frequent dialogue with the project owner. Having frequent contact was beneficial in many ways. The project owner was able to provide feedback for each iteration and provide new ideas and other requirements if any of the current were to change. Daily scrums are another Scrum philosophy that was used. This allowed group members to catch up, reflect on their previous work, and plan the next iteration's work.

### **3.4 Project plan**

The project was planned and visualized using a Gantt Chart, which may be seen in Appendix 9.1. (Gantt Chart). A Gantt Chart is a bar chart that illustrates the progress of a project and its activities over time. The project plan is not final because it takes into account the practice of

adaptability. The plan can be changed during the project implementation depending on the circumstances and new requirements from the project owner.

The plan was developed in accordance with the Scrum philosophy, which divides a project into several phases. There were four phases to the project: (1) Pre-Project Phase; (2) Development Phase; (3) Testing Phase; (4) Finishing Phase.

### **3.4.1 Pre-Project Phase**

The Pre-Project Phase is where all the initial work and planning were done. All the tasks concerning the pre-project such as developing a strategic plan, defining the initial requirements specification, and discussing optional technologies for the initial solution idea.

### **3.4.2 Development Phase**

The project was mainly focused on application development during this phase. This was done to meet the defined goals. This phase included developing the application, preparing the horizontal prototype to be presented in the Testing Phase which the project will enter simultaneously as this phase is ongoing.

It was expected that the Development Phase would be the most time-consuming and challenging. Therefore, the largest amount of time available for disposition was allocated to this phase.

### **3.4.3 Testing**

It was planned that a prototype will be ready for exhibit in this phase. Unit Testing was done synchronously with developing the basic functions in the application, the principals of test-driven development was central.

### **3.4.4 Finishing Phase**

Adding final touches to the project, submitting a reflection paper and presentation of the work were part of this phase.

### **3.5 Risk management**

Various risks and countermeasures associated with the project can be seen in the Appendix 9.2 (Risk Management) and Appendix 9.3 (Risk Matrix). The risk assessments were placed in a risk matrix to visualise and analyse the risks. The matrix is used to assess the severity of the risks associated with this project, based on the probability of an unwanted event occurring and its consequences. After mapping the risks, it was decided to plan countermeasures and strategies to reduce the probability and severity. By planning how the different risks will be handled, it was concluded that the risks were manageable. The risk plan is highly adaptable and in continuous development. The plan will adapt based on the given circumstances so that new risks will be addressed right away.

### **3.6 Evaluation plan**

With the support of the project owner, who coordinated meetings with project stakeholders, an evaluation plan was created. Different stakeholders will evaluate the application at different stages, with the aim of securing as much varied feedback as possible.

The application was evaluated by:

- Members of the DYNAMIC Research Project, including the project owner. These individuals have extensive knowledge of fire risk and can provide input on the application's usability.
- A group of users with varying technological backgrounds and no expertise in the fields of fire and risk management.
- Members of the fire brigade, ideally firefighters and fire engineers. One of the motivations for this project as mentioned in Chapter 1 is to relieve appropriate organizations so that they can manage their resources better. The application is expected to be of strategic value to the fire brigade, they may therefore be interested in testing it early. Their feedback on the application's flow and user interface will be essential.



The results from the test panel will be evaluated using the System Usability Scale (SUS) which was created by James Brooke in 1995 [28]. It provides a survey scale that is quick and easy for measuring the usability of a wide variety of products or services. The test panel will answer ten questions where they can choose between five response options in a range from strongly agree to strongly disagree.

It is planned to perform an evaluation concerning the application's performance closer to the end of the Testing Phase. This assessment will cover memory usage of the application, battery usage and data usage of the application. The initial idea is to perform this evaluation by using functions from the different operative systems to monitor resource usage of the application.

# Chapter 4: Detailed design

The chosen design pattern will be discussed in this chapter, as well as all the components in the High-Level Application Software Architecture. All diagrams in this chapter adhere to the Unified Modelling Language (UML) convention and aid in the comprehension of the application's functions.

## 4.1 Use Case Diagram

A Use Case Diagram is an efficient way of presenting and summarizing the possible actions associated actors can perform. The primary actor that uses the application is simply described as the *User*. The *Weather Data Source* and *Local Database* are identified as supporting actors since they provide a service to the application. The Use Case Diagram for this project can be seen in Figure 4 and presents the different actors and their roles in the application.

This section will include a brief description of each actor and their objectives.

- **User:** The *User* initializes the application and has the possibility to set up a profile, view-, add-, and delete locations. A notification will be sent out should the fire risk increase for any of the locations.
- **Weather Data Source:** The *Weather Data Source* is responsible for providing weather data, that will be passed into the fire risk model. The only time the *User* will be in direct contact with this actor is when initializing the application.
- **Local Database:** The *Local Database* offers a fully implemented data storage unit directly on the *User's* mobile device.

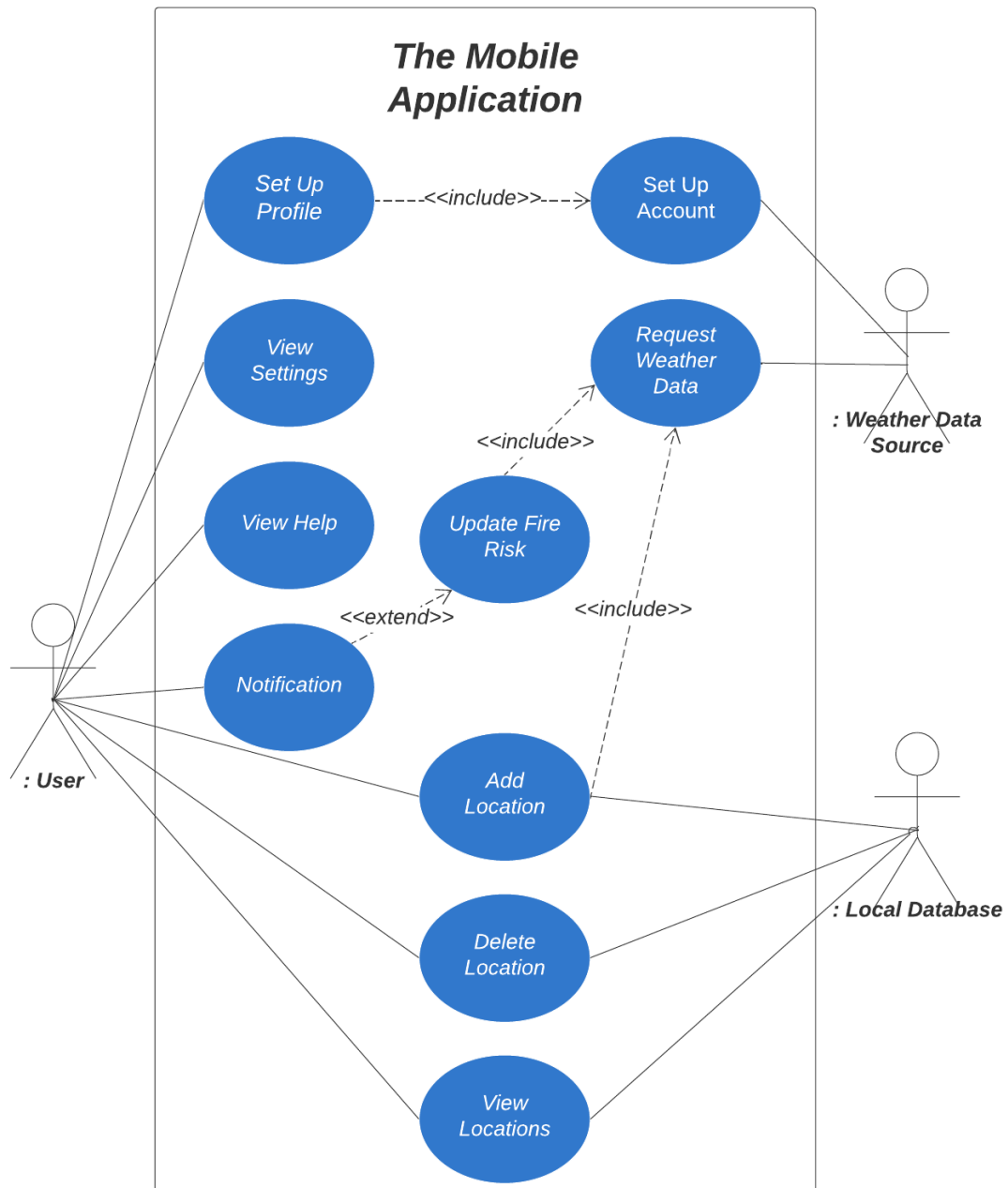


Figure 4. UML Use Case Diagram

#### 4.1.1 Use Case Descriptions

There are three primary use cases for this application:

- **Set Up Profile:** The *User* initializes the application by setting up a profile. This implies retrieving credentials from the *Weather Data Source*'s website. The *User* will launch the application and be redirected to the website.
- **Add Location:** The *User* adds new locations by specifying coordinates.
- **Notification:** The application will notify the *User* if fire risks for a location increase.

## 4.2 Domain Model

A Domain Model is a diagram that depicts the relationships between entities and concepts in a problem/application domain. The diagram assists domain specialists in gaining a solid understanding and agreement on the system [29]. Figure 5 shows the various concepts found within the mobile application.

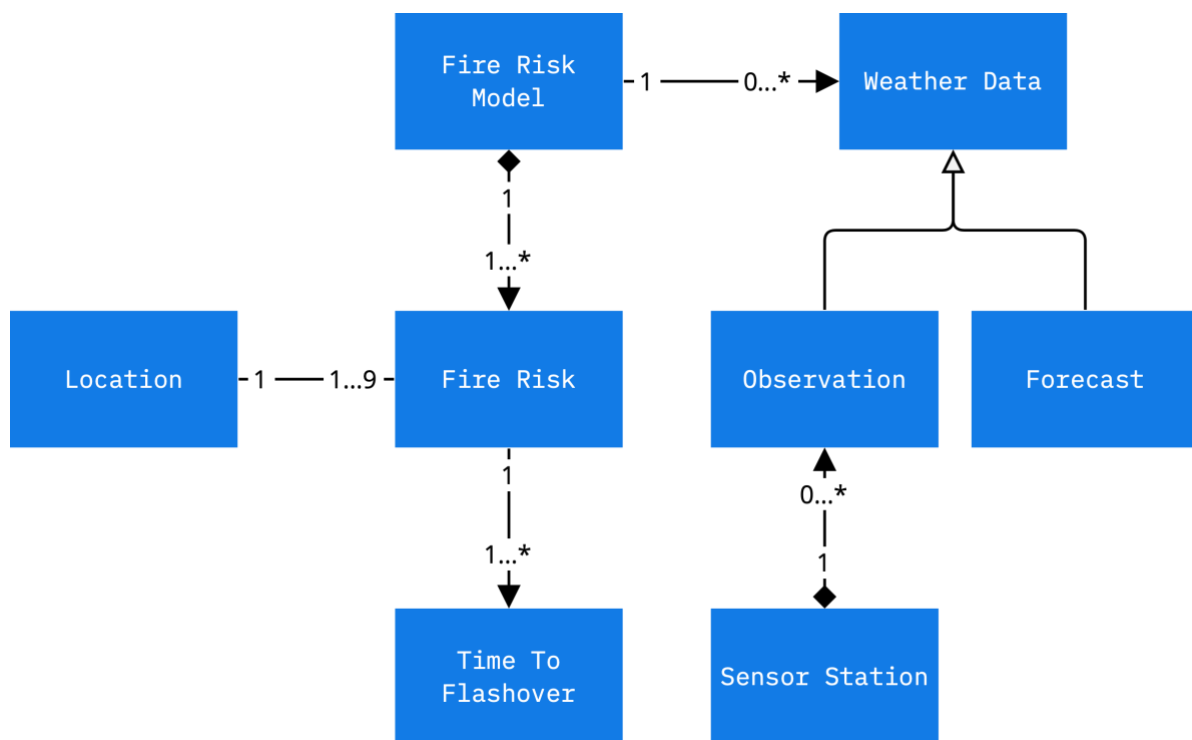


Figure 5. Domain model of the application

The Domain Model has two compositions [30], one from *Fire Risk Model* to *Fire Risk* and the other from *Sensor Station* to *Observation*. A *Fire Risk*, for example, cannot exist without a *Fire Risk Model*. Depending on the type of weather data sought, *Weather Data* can be either an *Observation* or a *Forecast*. The relationship between *Fire Risk* and *Location* states that a *Location* expects *Fire Risks* for the next nine days.

### 4.3 Application Design Pattern

A good practice when writing application code is to follow an application design pattern. The most popular design patterns associated with Xamarin development are the Model-View-Controller (MVC) and Model-View-ViewModel (MVVM) patterns.

The MVVM [31] design pattern was adopted for this project. The justification for this is that the design pattern assists in guiding the code's structure and design. It will also help to separate the various problems. The decoupling made it possible for the project group to work on the application in parallel because the application logic and the user interface were clearly separated.

MVVM consists of three components as showed in Figure 6:

- **View:** Responsible for the layout and appearance. It does not contain any business logic besides defining the user interface. The view is typically defined using XAML for applications that use Xamarin.
- **ViewModel:** The ViewModel implements properties which are responsible for interacting with the View component, it does so by connecting to the binding elements in the view component. The ViewModel is also responsible for handling communication and passing all data between the Model and the View.
- **Model:** The Model represents the application's domain and defines the logic of the program. It is typically used in conjunction with services and separate projects.

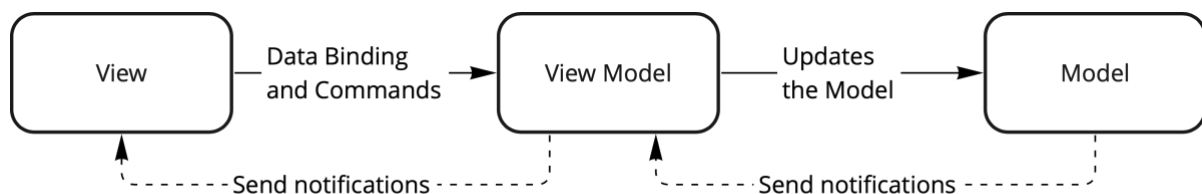


Figure 6. Structure of Model-View-View Model

## 4.4 High-Level Application Software Architecture

Figure 7 shows a High-Level Application Software Architecture of the mobile application. It shows the main components that form the application and how they interact with each other.

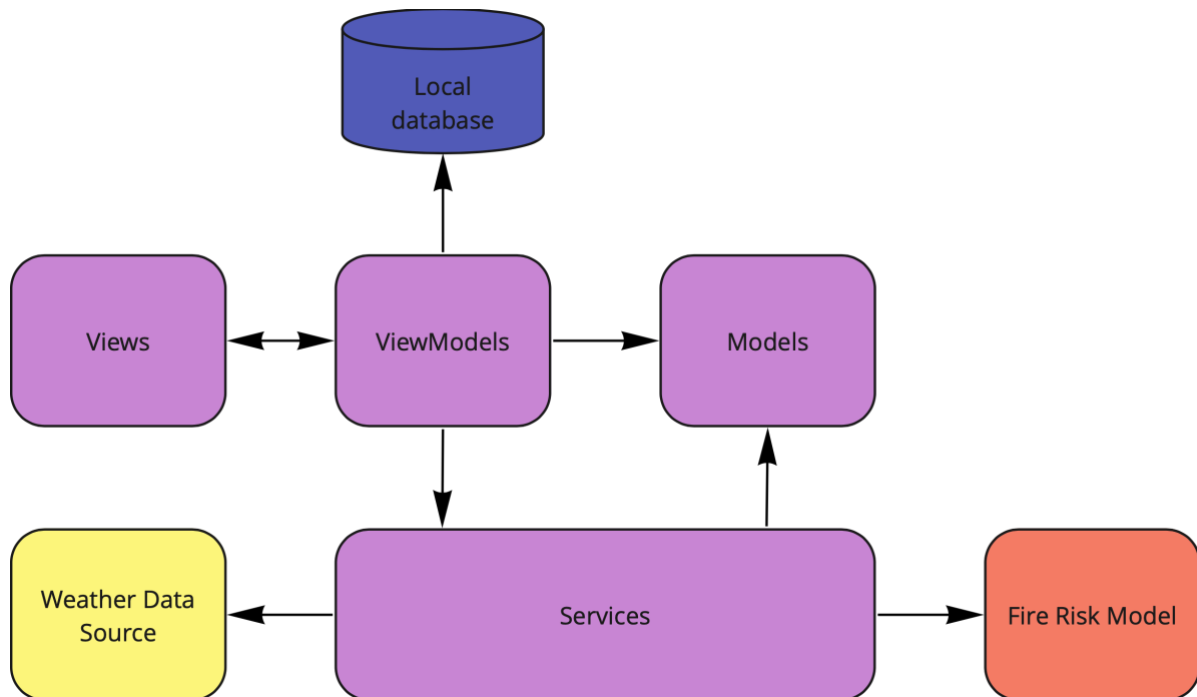


Figure 7. High-Level Application Software Architecture

Purple is used to represent the application's core components, which match the MVVM pattern's naming convention. Views and ViewModels are coupled, as the Views trigger commands in ViewModels. The ViewModels can store and edit content in the Local database, and more information is provided in Section 4.10. (SQLite database). The ViewModels uses Services to retrieve weather data by sending API queries to the Weather Data Source. Models are used by Services to format weather data responses into objects that comply with .NET standards. Services passes the data into the Fire Risk Model and retrieves the fire risk calculations after formatting the responses.

Sequence diagrams for the use cases *Add location* and *Notification* can be seen in Figures 8 and 9. These two act as representative use cases, because the other use cases operate similarly.

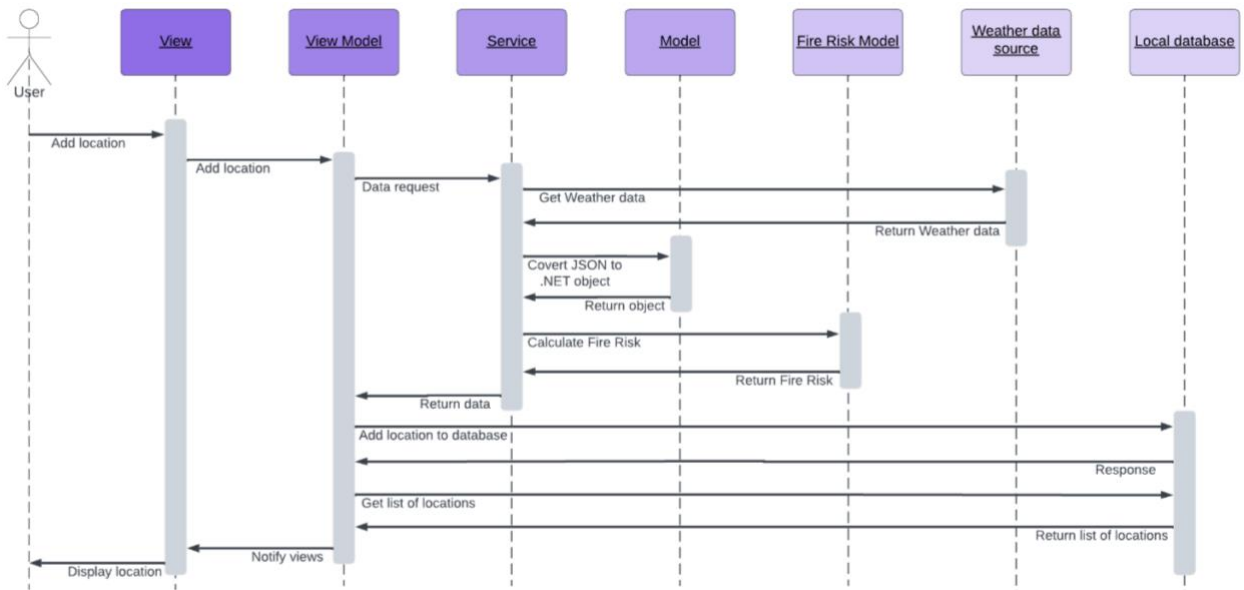


Figure 8. Sequence diagram when a user adds a new location

The *Add Location* sequence diagram illustrates how the components interact when a new location is added. The *ViewModel* triggers *Service*, which is responsible of storing the newly created object in the database. *Service* oversees obtaining and translating JSON responses to .NET objects, as well as calculating fire risks for the new location.

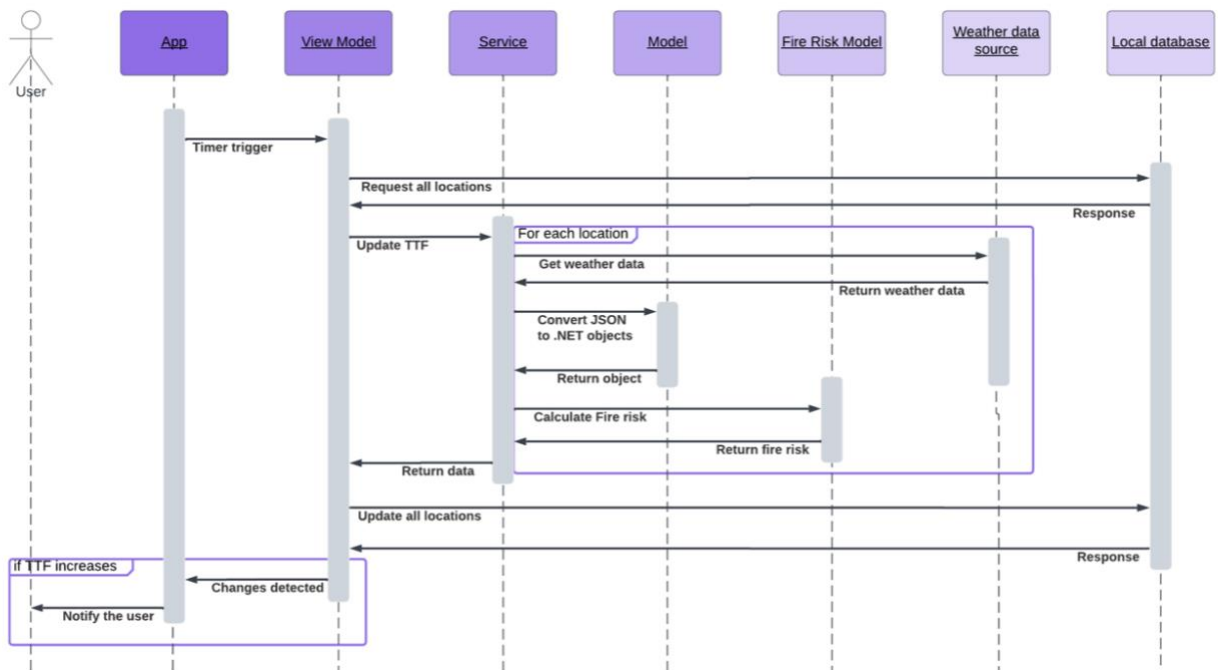


Figure 9. Sequence diagram for Push notification

The *Notification* use case is expected to be triggered by a timer constant, that decides when to update fire risks for all locations. `ViewModel` updates the fire risks using an instance from `Service`. If the fire risk increases, the mobile application will inform the user.

## 4.5 Fire Risk

TTF as previously mentioned is the transition time between the growth period to the fully developed stage in fire development. It's used to present fire risk for locations, as each location contains a list of fire risk values for the next four days, starting with today.

### 4.5.1 Fire risk benchmarking

The project owner created a scale to compare TTF values, as shown in Figure 10. This system assigns a value to the different colour-coded bars, which is highly useful for comparing TTF representations.

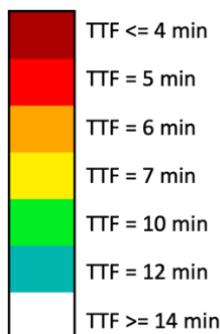


Figure 10. Scaling of the TTF values

The threshold for no fire risk was set at 14 minutes or higher. The white bar will be highlighted if the TTF for a certain location is 14 or greater for the specific day. The backend logic behind the representation of the system to present the fire risk is seen in Listing 1.



```

1  public List<int> FindFireRisk(List<double> ttfList)
2  {
3      List<int> fireRisk = new List<int>();
4
5      foreach(var ttf in ttfList)
6      {
7          if (ttf >= 14)
8              fireRisk.Add(0);
9          if (ttf >= 11 && ttf < 14)
10             fireRisk.Add(1);
11         if (ttf >= 8.5 && ttf < 11)
12             fireRisk.Add(2);
13         if (ttf >= 7 && ttf < 6)
14             fireRisk.Add(3);
15         if (ttf == 6)
16             fireRisk.Add(4);
17         if (ttf == 5)
18             fireRisk.Add(5);
19         if (ttf <= 4)
20             fireRisk.Add(6);
21     }
22     return fireRisk;
23 }

```

*Listing 1. FindFireRisk determines fire risks based on the TTF value.*

## 4.6 Weather Data Sources

The model Halderaker and Evjenth [9] implemented uses outside temperature, wind speed, and relative humidity as input at specific time intervals. The application uses the same model and therefore must retrieve the same data through its API requests.

### 4.6.1 The Norwegian Meteorological Institute

MET is Norway's national meteorological institute [32] and provides weather forecasts for civilians and conducts meteorology, oceanography, and climatology research. In collaboration with the Norwegian Broadcasting Corporation (NRK), they host the site Yr.no [33], an online weather service that offers detailed weather forecasting. In addition to Yr.no, they provide the service MET Norway Weather API [19]. The Weather API is an interface to a selection of data produced and made available by MET through the URL `https://api.met.no`.

The application will use the Location forecast service [34] for a specified area and Frost [35], a REST API for meteorological observation data.

#### 4.6.2 The Weather API Location Forecast Service

The Weather API Location forecast service provides weather forecasts for the next nine days for the specified geographic position based on coordinates. The forecast is based on numerous data sources and is updated several times a day. The forecast provides hourly updates for the first two-three days, after the third day the update frequency is changed to an interval of six hours [36].

The data comes in three endpoints: *classic*, *complete*, and *compact*. *classic* is the old XML format and is now considered legacy. *complete* and *compact* are JSON objects, where *complete* consists of all values and *compact* is a shorter version consisting of the most used parameters. The endpoint *complete* included all the required data and was therefore used [34].

An example of an HTTP GET request to the *locationforecast complete* endpoint:

```
https://api.met.no/weatherapi/locationforecast/2.0/complete.js  
on?lat=60.3691&lon=5.3505
```

The *lat* and *lon* parameters represent the geographic position's latitude and longitude. MET recommends not to use more than four decimals, the reason for this is to avoid blocking which will provide effective caching. The parameters that are typically included in an HTTP GET request are the following: altitude, latitude, and longitude. Longitude and latitude are the only parameters that must be included for every request whereas altitude is optional.

Listing 2 illustrates a JSON response body for the HTTP GET request. The mobile application will retrieve the following parameters from the JSON response body:

*air\_temperature*, *relative\_humidity*, *wind\_speed*, *wind\_speed\_of\_gust*, and *wind\_from\_direction*.

```

{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [
      5.3505,
      60.3691,
      27
    ]
  },
  "properties": {
    "meta": {
      "updated_at": "2022-04-17T06:40:56Z",
      "units": {
        "air_pressure_at_sea_level": "hPa",
        "air_temperature": "celsius",
        "air_temperature_max": "celsius",
        "air_temperature_min": "celsius",
        "air_temperature_percentile_10": "celsius",
        "air_temperature_percentile_90": "celsius",
        "cloud_area_fraction": "%",
        "cloud_area_fraction_high": "%",
        "cloud_area_fraction_low": "%",
        "cloud_area_fraction_medium": "%",
        "dew_point_temperature": "celsius",
        "fog_area_fraction": "%",
        "precipitation_amount": "mm",
        "precipitation_amount_max": "mm",
        "precipitation_amount_min": "mm",
        "probability_of_precipitation": "%",
        "probability_of_thunder": "%",
        "relative_humidity": "%",
        "ultraviolet_index_clear_sky": "1",
        "wind_from_direction": "degrees",
        "wind_speed": "m/s",
        "wind_speed_of_gust": "m/s",
        "wind_speed_percentile_10": "m/s",
        "wind_speed_percentile_90": "m/s"
      }
    },
    "timeseries": [
      {
        "time": "2022-04-17T07:00:00Z",
        "data": {
          "instant": {
            "details": {
              "air_pressure_at_sea_level": 1029.5,
              "air_temperature": 7.9,
              "air_temperature_percentile_10": 7.2,
              "air_temperature_percentile_90": 8.6,
              "cloud_area_fraction": 97.7,
              "cloud_area_fraction_high": 97.6,
              "cloud_area_fraction_low": 1.9,
              "cloud_area_fraction_medium": 0.0,
              "dew_point_temperature": 2.1,
              "fog_area_fraction": 0.0,
              "relative_humidity": 73.5,
              "ultraviolet_index_clear_sky": 0.7,
              "wind_from_direction": 153.7,
              "wind_speed": 4.8,
              "wind_speed_of_gust": 9.4,
              "wind_speed_percentile_10": 3.8,
              "wind_speed_percentile_90": 4.8
            }
          }
        }
      }
    ]
  }
}

```

Listing 2. A JSON response body from the complete endpoint

### 4.6.3 The Frost API

Frost [35] is a RESTful API that provides free access to MET Norway's historical weather and climate data archive. The data includes quality-controlled temperature, precipitation, and wind data measurements. It also offers information such as metadata about the weather stations. The API is primarily used by developers who wish to access MET's historical data archive.

The model of Log [5] uses historical weather data to calibrate, this is done so that model can achieve and provide more accurate predictions. To achieve the most accurate and relevant results, the application must retrieve data from the closest weather station that stores the relevant data. Frost has various API reference endpoints [37] which deliver different types of data. For this project, *sources* and *observations* were used.

The *sources* reference [37] provides the application with the closest station. There are several parameters to define when making a request. Relevant parameters for this application are *types*, *elements*, and *geometry*.

An example of an HTTP GET request to the *sources* endpoint:

```
https://frost.met.no/sources/v0.jsonld?types=SensorSystem&elements=air_temperature,relative_humidity,wind_speed&geometry=nearest(POINT(5.3327 60.383))
```

The *types* parameter specifies the station type, whereas *SensorSystems* is a station with measuring sensors. It is also possible to choose between *InterpolatedDataset* and *RegionDataset*. The API can exclude stations that do not meet the requirements by specifying the *elements* parameter with: *air\_temperature*, *relative\_humidity*, *wind\_speed*.

The *geometry* parameter specifies the geometry of a station. The geographical location is expressed in terms of either a single point or a polygon area [38]. The syntax `nearest(POINT (lon lat))` refers to the item closest to these coordinates. If the `nearest` method is used, the response will include the distance in kilometres from the reference point.

Listing 3 illustrates a JSON response body for the HTTP GET request.

```
{
  "@context" : "https://frost.met.no/schema",
  "@type" : "SourceResponse",
  "apiVersion" : "v0",
  "license" : "https://creativecommons.org/licenses/by/3.0/no/",
  "createdAt" : "2022-04-14T19:16:25Z",
  "queryTime" : 3.498,
  "currentItemCount" : 1,
  "itemsPerPage" : 1,
  "offset" : 0,
  "totalItemCount" : 1,
  "currentLink" :
  "https://frost.met.no/sources/v0.jsonld?types=SensorSystem&elements=air_temperature%2C
  relative_humidity%2Cwind_speed&geometry=nearest(POINT(5.3199%2060.3915))&nearestmaxcou
  nt=1",
  "data" : [ {
    "@type" : "SensorSystem",
    "id" : "SN50540",
    "name" : "BERGEN - FLORIDA",
    "shortName" : "Bergen",
    "country" : "Norge",
    "countryCode" : "NO",
    "wmoId" : 1317,
    "geometry" : {
      "@type" : "Point",
      "coordinates" : [ 5.3327, 60.383 ],
      "nearest" : false
    },
    "distance" : 1.17811577289,
    "masl" : 12,
    "validFrom" : "1949-11-28T00:00:00.000Z",
    "county" : "VESTLAND",
    "countyId" : 46,
    "municipality" : "BERGEN",
    "municipalityId" : 4601,
    "stationHolders" : [ "MET.NO" ],
    "externalIds" : [ "0-20000-0-01317", "10.249.0.159" ],
    "wigosId" : "0-20000-0-01317"
  } ]
}
```

Listing 3. A JSON response body from the sources endpoint

The *observations* reference [37] provides actual observations data from MET Norway's data storage system. It only handles restricted data by using query parameters. The parameters needed to make a request are *sources*, *referencetime*, and *elements*.

An example of an HTTP GET request to the *observations* endpoint:

```
https://frost.met.no/observations/v0.jsonld?sources=SN50540&referencetime=2022-04-01/2022-04-02&elements=air_temperature,relative_humidity,wind_speed
```

*Sources* specify which station to get observations from, and *referencetime* implies for which time range. The time specification [38] is based on UTC and ISO-8601.

The format the application uses to cite dates is YYYY-MM-DD. For instance, one can define a time range by writing: *2022-04-01/2022-04-02* .

It will retrieve observations that are expressed according to the ISO-8601 format [39]. Starting from *2022-04-01T00:00:00:000Z* to *2022-04-02T23:00:00:000Z*.

Listing 4 illustrates the first observation, and the elements are defined by an *elementId*. Other variables to take notice of are the *timeOffset* and *timeResolution*. The variable *timeOffset* is the observation time relative to midnight. For this observation, the *timeOffset* is PT0H, meaning that the daily value has an observation time of 00:00 UTC. *timeResolution* specifies the period between each data value. For this instance, the value is PT1H, which gives data values hourly.

```

{
  "@context" : "https://frost.met.no/schema",
  "@type" : "ObservationResponse",
  "apiVersion" : "v0",
  "license" : "https://creativecommons.org/licenses/by/3.0/no/",
  "createdAt" : "2022-04-15T19:06:27Z",
  "queryTime" : 0.852,
  "currentItemCount" : 1,
  "itemsPerPage" : 1,
  "offset" : 0,
  "totalItemCount" : 1,
  "currentLink" :
  "https://frost.met.no/observations/v0.jsonld?sources=SN50540&referencetime=2022-04-01/2022-04-01&elements=air_temperature,relative_humidity,wind_speed",
  "data" : [ {
    "sourceId" : "SN50540:0",
    "referenceTime" : "2022-04-01T00:00:00.000Z",
    "observations" : [
      {
        "elementId" : "air_temperature",
        "value" : 0,
        "unit" : "degC",
        "level" : {
          "levelType" : "height_above_ground",
          "unit" : "m",
          "value" : 2
        }
      },
      {
        "timeOffset" : "PT0H",
        "timeResolution" : "PT1H",
        "timeSeriesId" : 0,
        "performanceCategory" : "C",
        "exposureCategory" : "2",
        "qualityCode" : 0
      }
    ],
    {
      "elementId" : "relative_humidity",
      "value" : 58,
      "unit" : "percent",
      "level" : {
        ...
      },
      ...
    },
    {
      "elementId" : "wind_speed",
      "value" : 1.4,
      "unit" : "m/s",
      "level" : {
        ...
      },
      ...
    }
  ]
}

```

Listing 4. A JSON response body from the observations endpoint

#### 4.6.4 Authorization

To gain access to the services, the user must first create an account [40] by visiting `https://frost.met.no/auth/requestCredentials.html` to request credentials. To register, the user will need to submit an email address and will be given a new client ID and client secret. The emails will be saved in a database managed by MET [41]. For free service and fast and stable performance, the client ID is required [42]. The client secret is only necessary for access to information that is not publicly available.

Basic authentication [43] and OAuth2 [44] are alternative options that can be used for authorization. OAuth2 must be used when accessing confidential data, but it will also function for publicly available data. The request relevant for this application only requires the client id, and will hereby be referred to as MET ID.

The applications authorization mechanism relies on Basic Authentication, considering the client's secret was unnecessary. If this changes, the application can easily be modified. The Basic Authentication scheme is considered secure when the web client and server connection are secure [43]. Since MET encourages its customers to use basic authentication, it is safe to assume that these criteria apply to this service. Before providing the credential into an authentication header, it is transformed to base-64 encoding, with the format:

*Authorization: Basic <client ID>:<client secret>.*

## 4.7 Data Models

The `Models` components act as non-visual classes that encapsulate the data and represent the application's domain, including business and validation logic [31]. This chapter will show the different model objects and how they are utilized.

An overview of the different classes listed in the `Models` component can be seen in Figure 11, `TheFrostStation`, `FrostWeatherData`, and `MetForecast` were auto generated classes using quicktype [45]. They were transformed from JSON objects to .NET classes.



## Models

- | FrostStation.cs
- | FrostWeatherData.cs
- | Location.cs
- | MetForecast.cs
- | MetID.cs
- | WindData.cs

Figure 11. A representation of classes in the Models component

The Fire Risk Model requires that the weather data is retrieved from the JSON response bodies into standard C# code before it can be processed. Observation is a data model that exists within the Fire Risk Model project. The Observation objects parameters DateTime, Temperature, Humidity, and WindSpeed can be seen in Listing 5.

```
1  public class Observation
2  {
3      private DateTime referenceTime;
4      private Temperature temperature;
5      private Humidity humidity;
6      private WindSpeed? windSpeed;
7
8
9      public Observation(DateTime referenceTime, Temperature temperature,
10                          Humidity humidity, WindSpeed? windSpeed)
11      {
12          this.referenceTime = referenceTime;
13          this.temperature = temperature;
14          this.humidity = humidity;
15          this.windSpeed = windSpeed;
16      }
17      ...
18 }
```

Listing 5. The Observation object from the Fire Risk Model project

The Location object consists of three lists for handling wind data: *List<double> WindSpeed*, *List<double> WindGustSpeed* and *List<double> WindDirection* as seen in Listing 6.

```

1  public class Location
2  {
3      [PrimaryKey, AutoIncrement]
4      public Guid Id { get; set; }
5      public string Name { get; set; }
6      public string Latitude { get; set; }
7      public string Longitude { get; set; }
8      public string WeatherStationID { get; set; }
9      [TextBlob("WindSpeedsBlobbed")]
10     public List<double> WindSpeed { get; set; }
11     public string WindSpeedsBlobbed { get; set; }
12     [TextBlob("WindGustSpeedsBlobbed")]
13     public List<double> WindGustSpeed { get; set; }
14     public string WindGustSpeedsBlobbed { get; set; }
15     [TextBlob("WindDirectionsBlobbed")]
16     public List<double> WindDirection { get; set; }
17     public string WindDirectionsBlobbed { get; set; }
18     [TextBlob("FireRisksBlobbed")]
19     public List<int> FireRisk { get; set; }
20     public string FireRisksBlobbed { get; set; }
21     public int PeakRisk { get; set; }
22 }

```

*Listing 6. Representation of the Location object*

## 4.8 Service Clients

The following chapter introduces the Services components that implement the protocols defined by the Business Layer and handle all HTTP requests to MET and Frost API [46].

Services consist of two interfaces, `IRestServiceClient` and `IWeatherClient` and classes `RestServiceClient` and `WeatherClient` seen in Figure 12, which implements these interfaces.

### Services

```

|   IRestServiceClient.cs
|   IWeatherClient.cs
|   RestServiceClient.cs
|   WeatherClient.cs

```

*Figure 12. A representation of classes in the Services component*

All API handling is controlled by the `IRestServiceClient` shown in Listing 7. `GetNearestStation` retrieves the nearest station that stores air temperature, relative humidity, and wind speed based on coordinates. The `GetWeatherObservations` uses the station information, MET ID, and a period to request observations from Frost. `GetWeatherForecasts` is responsible for retrieving all weather forecasts that the Fire Risk Model uses to predict the actual fire risk that the application displays.

```
1 public interface IRestServiceClient
2 {
3     Task<MetForecast> GetWeatherForecasts(string longitude, string latitude,
4                                         string MET_ID);
5
6     Task<FrostStation> GetNearestStation(string longitude, string latitude,
7                                         string MET_ID);
8
9     Task<FrostWeatherData> GetWeatherObservations(string station_id, string MET_ID,
10                                                    string period);
11 }
```

*Listing 7. IRestServiceClient's methods*

`IWeatherClient` seen in Listing 8 handles all communication between the `ViewModel`, `IRestServiceClient`, and the Fire Risk Model. The decision behind introducing a second interface was to uphold the principle of loose coupling. `GetStation`, `GetForecasts`, and `GetObservations` all use an instance of the `IRestServiceClient` for retrieving weather data.

```
1 public interface IWeatherClient
2 {
3     Task<string> GetStation(string longitude, string latitude, string MET_ID);
4
5     Task<Tuple<List<Observation>, List<double>, List<double>, List<double>>>
6         GetForecasts(string longitude, string latitude, string MET_ID);
7
8     Task<List<Observation>> GetObservations(string station_id, string MET_ID,
9                                           string period);
10
11    Task<Location> UpdateFireRisk(Location location, string MET_ID, string period);
12
13    List<int> GetFireRiskForNewLocation(List<Observation> observations,
14                                       List<Observation> forecasts);
15 }
```

*Listing 8. IWeatherClient's methods*

### 4.8.1 RestSharp

RestSharp [47] is a C# specific HTTP client library with synchronous and asynchronous methods for calling remote resources over HTTP. The library handles the deserialization and serialization of responses and construction of request URI using several arguments, such as query, path, form, or body.

RestSharp includes authenticators for basic HTTP Authorization headers [48]. Listing 9 (line 7) reveals the `HttpBasicAuthenticator` that takes a *MET ID* and an empty string (as the password). The Authenticator auto generates a base64 encoded string and passes it into the request's header.

```
1 public async Task<FrostWeatherData> GetWeatherObservations(string station_id,
2 string MET_ID, string period)
3 {
4     RestResponse response;
5
6     var client = new RestClient(frost_url_base);
7     client.Authenticator = new HttpBasicAuthenticator(MET_ID, "");
8     var request = new RestRequest("observations/v0.jsonld")
9         .AddParameter("sources", station_id)
10        .AddParameter("referencetime", period)
11        .AddParameter("elements",
12            "air_temperature,relative_humidity,wind_speed");
13
14     response = await client.ExecuteGetAsync<FrostWeatherData>(request);
15
16     if (!response.IsSuccessful)
17         throw new Exception(((int)response.StatusCode).ToString());
18     else
19         return JsonConvert.DeserializeObject<FrostWeatherData>(response.Content);
20 }
```

Listing 9. *GetWeatherObservations* method from the *RestServiceClient* class

A request instance with the resource path is declared in Listing 9 (line 8). *Sources*, *referencetime*, and *elements* are added to the request URI before the client sends requests to the API, using the `ExecuteGetAsync<FrostWeatherData>` method. If the response is successful, the method will return a `FrostWeatherData` object.

## 4.8.2 Exception Handling

Exception handling is an efficient way to pass error messages to the user and alert them when something goes wrong. It is crucial to handle all exceptions properly, as they might cause problems if left unhandled.

To properly handle an exception, one will organize the code in the form of try-catch blocks. The class that initially asks for weather data is responsible for handling all exceptions. All these classes are representatives of the `ViewModel`. Listing 10 (lines 5-13) shows how a try-catch block is used while requesting weather observations to handle possible exceptions. The class uses an instance from `IWeatherClient`. The `WeatherClient` also uses try-catch blocks, as seen in Listing 11, and will only throw the exception further up the chain to the initial class (line 12).

```
1  private async void OnSave()
2  {
3      ...
4
5      try
6      {
7          observations = await client.GetObservations(stationId, App.MetID, period);
8      }
9      catch(Exception ex)
10     {
11         await Application.Current.MainPage.DisplayAlert("Error",
12             "Something went wrong: " + ErrorMessage(ex.Message), "ok");
13         return;
14     }
15     ...
16 }
```

*Listing 10. NewLocationViewModel's method OnSave handles an exception.*

```

1  public async Task<List<Observation>> GetObservations(string station_id,
2                                                    string MET_ID, string period)
3  {
4      FrostWeatherData result;
5
6      try
7      {
8          result = await client.GetWeatherObservations(station_id, MET_ID, period);
9      }
10     catch (Exception ex)
11     {
12         throw;
13     }
14     return ConvertObservations(result.data);
15 }

```

Listing 11. WeatherClient's method GetObsrvations

The `IRestService` client has the sole responsibility of throwing new Exceptions, as seen in Listing 9 (line 16). Should the `IsSuccessful` property be false, the request will be unsuccessful. The `Statuscode` property provides the generated error code from the API [49] and is included when the exception is thrown to the `WeatherClient`. The method `ExecuteGetAsync<FrostWeatherData>` in Listing 11 (lines 14-17) is an instance of `RestResponse<T>`, which offers valid information properties to retrieve the error response from the API [50].

By following these principles, the program will handle all exceptions properly. Hence, the `ViewModel` can notify the user if any errors occur. Should the user provide an invalid `MET ID`, the application will not be able to retrieve weather data from the API. The API will block the request and respond with the error status code 401 Unauthorized. The `ErrorMessage` method determines what type of error was thrown and alerts the user with an explanation, as seen in Listing 12 and Figure 13.

```

1 private string ErrorMessage(string statusCode)
2 {
3     if (statusCode.Equals("400"))
4         return "Invalid coordinates";
5     if (statusCode.Equals("401"))
6         return "Invalid MET ID";
7     if (statusCode.Equals("403"))
8         return "Forbidden, is your MET ID correct?";
9     if (statusCode.Equals("429"))
10        return "Slow down, too many requests! This can result in blacklisting";
11    if (statusCode.Equals("500"))
12        return "Internal server error. Try again later";
13    if (statusCode.Equals("502"))
14        return "Server error. Try again later";
15    if (statusCode.Equals("503"))
16        return "Server error. Try again later";
17    if (statusCode.Equals("504"))
18        return "Server error. Try again later";
19
20    return statusCode;
21 }

```

Listing 12. Representation of the ErrorMessage method

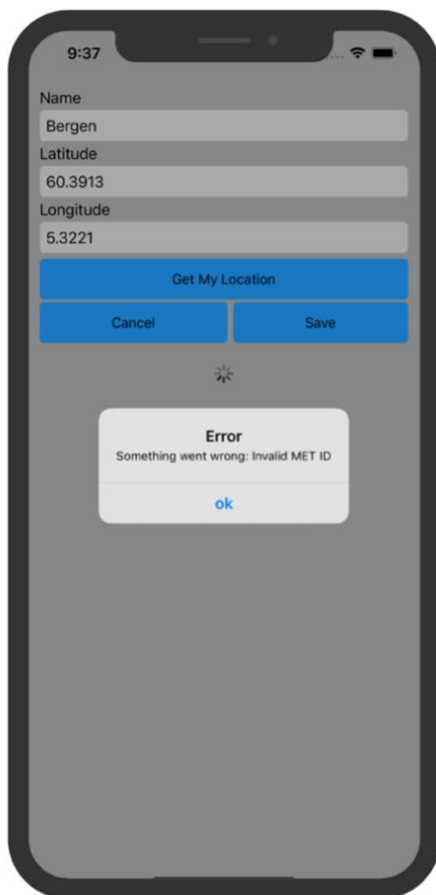


Figure 13. The error message when a user has submitted an invalid MET ID

## 4.9 Application Flow

State machine diagrams are often used to describe and display objects that are state-dependent. It illustrates the execution flow from one state to the next and records the system behavior based on the multiple states the application can be in [51].

As seen in Figure 14, the user will from the initial state at launch land on the *View My Locations* page. The user can from this page navigate to the different pages based on which action they wish to perform. The final state is after the user closes the application.

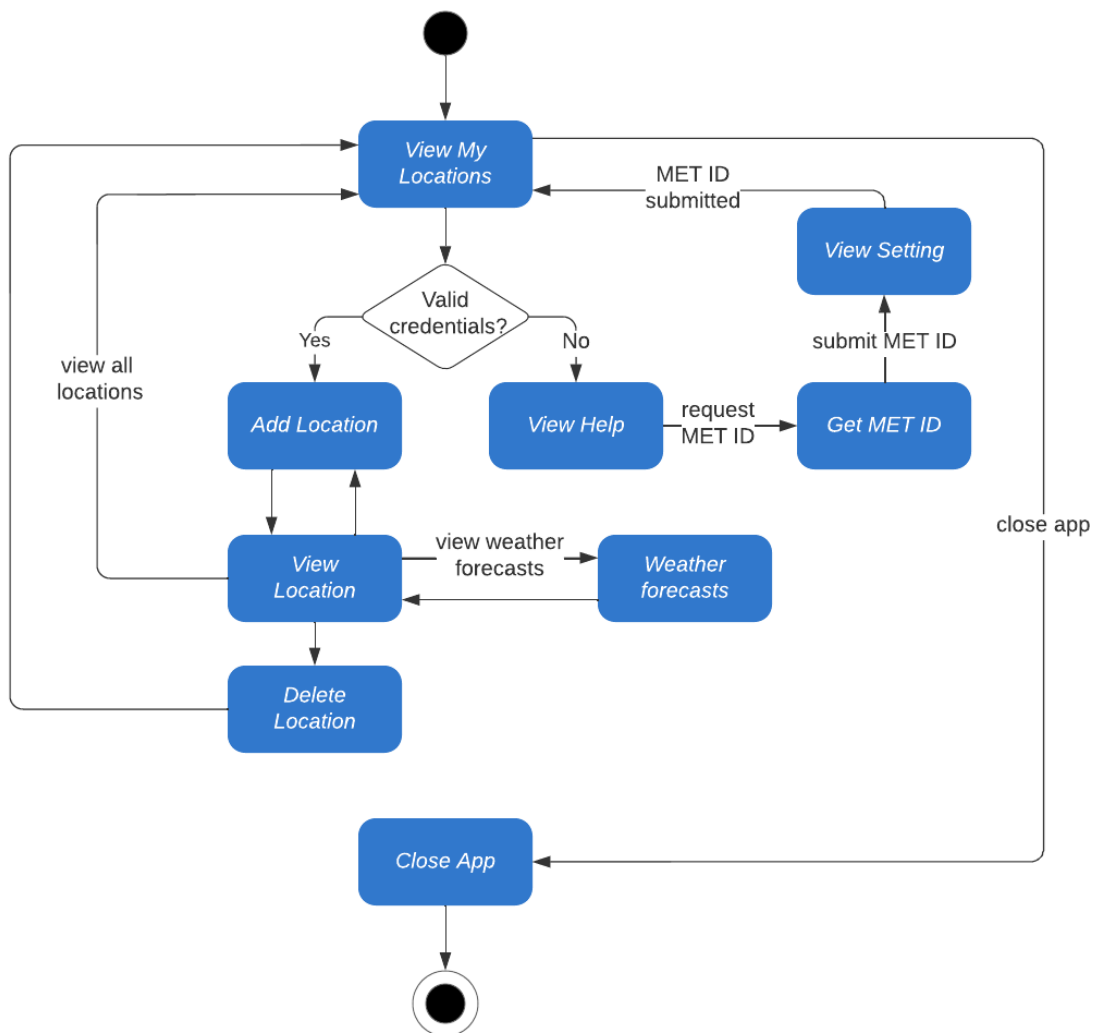


Figure 14. State machine diagram to display the flow.



## 4.10 SQLite Database

A local database was used to store data since it was necessary for a user to be able to track the fire risk of multiple `Locations`. Additionally, the user's unique MET ID is saved for quick access when retrieving data from the APIs.

SQLite [52] is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite reads and writes directly to ordinary files. Furthermore, it offers access to a fully operating database with tables, indices, triggers, and views obtained in a single disk file. SQLite generally runs faster the more memory that is provided. Nevertheless, performance is usually quite good, even in low-memory environments. The file format is cross-platform, thus an excellent option when working with Xamarin as it allows the mobile application to load and save `Locations` in shared code [53].

The SQLite NuGet Façade was firstly installed and then a database wrapper class was created. The `Database` class located in the `Data` component seen in Figure 15 centralizes all query logic and simplifies the management of database initialization. It upholds all database logic from the rest of the application and will make refactoring and expanding simpler if the application grows.

```
Data
|   Database.cs
```

*Figure 15. A representation of the class in the Data folder*

The SQLite .NET library provides a simple Object Relational Map (ORM) that allows storing different objects and retrieving them without writing any SQL statements.

The SQLite-Net extension [54] was used for storing lists in a single column. The extension provides attributes for specifying the relationships. Text blobbed properties can be used to store objects like `Lists`. The property is serialized into a text property when saved and deserialized when retrieved. The only requirement is to declare a string property where the serialized object

is stored. It acts as a JSON-based serializer, and an example of the `Location` object was shown in Listing 6 (lines 9-20).

A single database connection is used, this is demonstrated in Listing 13 (line 7) which shows the instantiating of a new `SQLiteAsyncConnection`. This object provides a singleton instance which ensures a single database connection. This offers better performance and is more stable compared to opening and closing multiple connections throughout an application session. The connection utilizes the tables with the `CreateTableAsync` method if they do not already exist (lines 8-9).

```
1  public class Database
2  {
3      readonly SQLiteAsyncConnection connection;
4
5      public Database(string dbPath)
6      {
7          connection = new SQLiteAsyncConnection(dbPath);
8          connection.CreateTableAsync<Location>().Wait();
9          connection.CreateTableAsync<MetID>().Wait();
10     }
11
12     public async Task<List<Location>> GetLocationsAsync()
13     {
14         return await ReadOperations.GetAllWithChildrenAsync<Location>(connection);
15     }
16
17     public async Task<Location> GetLocationAsync(Guid id)
18     {
19         return await ReadOperations.GetWithChildrenAsync<Location>(connection, id);
20     }
21
22     public async Task SaveLocationAsync(Location newLocation)
23     {
24         await WriteOperations.InsertWithChildrenAsync(connection, newLocation);
25     }
26
27     public async Task DeleteLocationAsync(Location location)
28     {
29         await WriteOperations.DeleteAsync(connection, location, false);
30     }
31
32     public Task<int> SetMetID(MetID newID)
33     {
34         return connection.InsertOrReplaceAsync(newID);
35     }
36
37     public async Task<List<MetID>> GetMetID()
38     {
39         return await connection.Table<MetID>().ToListAsync();
40     }
41 }
```

Listing 13. The Database class.

The `ReadOperations` and `WriteOperations` are part of the `SQLite-Net` extension library and do not modify or override any method behaviour of `SQLite.Net`. It provides methods that the connection can use to handle the specified relationships. The extension will determine how each element will be handled and stored in the database tables, for reference, see Table 1.

location		met id	
id	Guid [PK]	id	Int [PK]
name	Varchar	client_id	Varchar
longitude	Varchar		
latitude	Varchar		
weather station id	Varchar		
wind speed	List<Double>		
wind gust	List<Double>		
wind direction	List<Double>		
fire risk	List<Int>		
peak risk	Int		

*Table 1. Representation of the location and MET ID tables*

The `View Models` classes are responsible for handling all database logic. Listing 14 shows how the `NewLocationViewModel` uses the database connection (line 19) and provides the `SaveLocationAsync` with a `Location` before storing it in the database. This can be seen in the async method `OnSave`, which is triggered when a user adds a new `Location`.

```

1  private async void OnSave()
2  {
3      ...
4
5      Location newlocation = new Location()
6      {
7          Id = Guid.NewGuid(),
8          Name = Name,
9          Longitude = Longitude,
10         Latitude = Latitude,
11         WeatherStationID = stationId,
12         WindDirection = windDirection,
13         WindSpeed = windSpeed,
14         WindGustSpeed = windGust,
15         FireRisk = fireRisk,
16         PeakRisk = peakFire
17     };
18
19     await App.Database.SaveLocationAsync(newlocation);
20
21     ...
22 }

```

*Listing 14. Method to save a location to the database.*

## 4.11 Fire Risk Model

A translation of the model of Log [5] was done from Java to C# by the group's supervisor Lars Michael Kristensen. This was done so that the model can be used along with the other components which have been developed using C#.

According to Stokkenes' research [8], using historical data to correctly calibrate the model of Log [5] gave the most accurate fire risk predictions. It was also concluded that the model predicted inaccurate fire risks when weather forecasts were provided in an interval of six hours. Therefore, the application will only use the fire risk for the first four days.

The mobile application has a dependency on the Fire Risk Model and will therefore be able to utilize specific type classes, methods, and the interface `IFireRisk` seen in Listing 15. The `fireRiskFactorsFrom`, `fireRiskTtf`, and `interpolateObservations` are methods that were already accessible from the interface.

```

1  public interface IFireRisk
2  {
3      List<FireRiskFactor> fireRiskFactorsFrom(List<Observation> observations);
4
5      FireRiskResult fireRiskTtf(List<Observation> observations);
6
7      List<int> FireRiskTtf(List<Observation> observations);
8
9      List<Observation> interpolateObservations(List<Observation> observations);
10 }

```

Listing 15. IFireRisk's methods

To ease the transition when coupling the mobile application and Fire Risk Model, the method `FireRiskTtf` was created and can be seen in Listing 16. `FireRiskTtf` computes the fire risk based on a list of `Observation` objects spaced an hour apart and the method `computeFireRiskResult` returns a `FireRiskResult` which includes the variables: `RelativeHumidityModelStep[]`, `TimeToFlashover[]`, and an `Observation`.

```

1  public List<int> FireRiskTtf(List<Observation> observations)
2  {
3      FireRiskResult fireRiskResult = computeFireRiskResult(observations, null);
4      List<TimeToFlashover> ttfList = fireRiskResult.TimeToFlashovers.ToList();
5
6      List<double> minTtfList = SortByHighestTtf(ttfList);
7      return FindFireRisk(minTtfList);
8  }

```

Listing 16. IFireRisk's method `FireRiskTtf`

The `SortByHighestTtf` method in Listing 17, is used to sort the list of `TimeToFlashover` objects and can be seen in Listing 18. `SortByHighestTtf` groups all measurements based on date and filter out the *highest timeToFlashoverInMinutes* for each day.

```

1  public List<double> SortByHighestTtf(List<TimeToFlashover> ttfList)
2  {
3      List<double> highestTtfList = new List<double>();
4
5      var groupby = ttfList.GroupBy(date => date.Timestamp.Date).ToList();
6
7      foreach (var list in groupby)
8      {
9          var highestTtf = list.Min(element => element.TimeToFlashoverInMinutes);
10
11         highestTtfList.Add(highestTtf);
12     }
13     highestTtfList.RemoveRange(0, 5);
14
15     return highestTtfList;
16 }

```

*Listing 17. The method SortByHighestTtf*

```

1  public class TimeToFlashover
2  {
3      private double timeToFlashoverInMinutes;
4      private DateTime timestamp;
5
6      public TimeToFlashover(double timeToFlashoverInMinutes, DateTime timestamp)
7      {
8          this.timeToFlashoverInMinutes = timeToFlashoverInMinutes;
9          this.timestamp = timestamp;
10     }
11     ...
12 }

```

*Listing 18. The GetFireRisk method from the WeatherClient class*

The `GetFireRisk` method shown in Listing 19 is responsible of retrieving the fire risk. This is done by using an instance of the Interface `IFireRisk` from the `Fire Risk Model`. `GetFireRiskForNewLocation` and `UpdateFireRisk` seen in Listings 20 and 21 are part of the `IWeatherClient`. These methods are tightly coupled to the `Fire Risk Model` and uses the `GetFireRisk` method to fetch fire risk.

```

1  public List<int> GetFireRisk(List<Observation> observations)
2  {
3      IFireRisk fr = new FireRisk();
4      List<int> results = fr.FireRiskTtf(observations);
5
6      return results;
7  }

```

*Listing 19. The GetFireRisk method from the WeatherClient class*

```

1 public List<int> GetFireRiskForNewLocation(List<Observation> observations,
2 List<Observation> forecasts)
3 {
4     return GetFireRisk(observations.Concat(
5         RemoveNotHourlyForecasts (forecasts)).ToList());
6 }

```

Listing 20. WeatherClient's method GetFireRiskForNewLocation

```

1 public async Task<Location> UpdateFireRisk(Location location, string MET_ID,
2 string period)
3 {
4     List<Observation> observations = new List<Observation>();
5     List<Observation> forecasts = new List<Observation>();
6
7     try
8     {
9         observations = await GetObservations(location.WeatherStationID,
10 MET_ID, period);
11     }
12     catch(Exception ex)
13     {
14         throw;
15     }
16
17     try
18     {
19         var result = await GetForecasts(location.Longitude,
20 location.Latitude, MET_ID);
21         forecasts = result.Item1;
22
23         location.WindSpeed = result.Item2;
24         location.WindGustSpeed = result.Item3;
25         location.WindDirection = result.Item4;
26     }
27     catch(Exception ex)
28     {
29         throw;
30     }
31
32     location.FireRisk =
33         GetFireRisk(observations.Concat(RemoveNotHourlyForecasts(forecasts)).ToList());
34
35     int peak = location.FireRisk.Max(x => x);
36     location.PeakRisk = peak;
37
38     return location;
39 }

```

Listing 21. WeatherClient's method UpdateFireRisk

## 4.12 Graphical User Interface (Views)

A well-designed user interface is a vital part of all computer systems [55]. The user interface must be easy to use and understand to provide a good user experience. There are many uses for the application; for example, the fire brigade or municipality can use it to manage their resources. To provide this service, the application must be practical and user-friendly to ensure higher productivity and efficiency.

The UI consists of Xamarin controls. The controls are divided into four groups: *Pages*, *Layouts*, *Views*, and *Cells*. A UI for a single *Page* can consist of one *Page* with different combinations of *Layouts*, *Views*, and *Cells*.

### 4.12.1 Pages

*Pages* represent the device's screen and hold all the content that will be displayed. *Pages* can be divided into various types for different purposes. A *Content Page* is used in the application since it is a simple control that has a single *Layout* and its child controls. A Xamarin Forms application is often made up of multiple *Pages* that can be navigated between, and each represents different content.

### 4.12.2 Layouts

*Layouts* work as a container for *Views* and occasionally other *Layouts*. They're mostly used to size and position the child elements. Some *Layouts* only allow for one child control, while others allow for several child controls as well as positioning options [56]. There are several *Layouts* in the application, but the most popular are *Stack Layout*, *Scroll View*, *Frame*, and *Grid*.

The *Stack Layout* stacks the child controls in the desired order (horizontal or vertical). A *Scroll View* holds one child and allows scrolling. The *Frame Layout* holds one child and surrounds it with a border. A *Grid Layout* positions children in defined cells and rows.



### 4.12.3 Views

*Views* are UI elements such as buttons, labels, and sliders. Depending on their intended usage, the controls can be divided into several categories. *Views for presentation*, *Views that initiate commands*, and *Views for setting values* are some examples. The *Views for presentation* are responsible for the visual presentation of objects, like text, figures, and images. *Views that initiate commands* are typically buttons or search bars and when these are clicked, they can be assigned to tasks and execute them. *Views for setting values* are check boxes, sliders, and entry fields. These controls are responsible of collecting data from the user and sends it to the backend of the application.

### 4.12.4 Cells

*Cells* are specialized elements used for items in a table that describes how each item in a list should be rendered. The cell functions as a template for creating visual elements. They are used exclusively in *List Views* and *Table Views* [57].

## 4.13 Graphical User Interface Structure

It was decided that the application will display the peak Indoor Fire Development (IFD) and peak wind for all `Locations`. IFD is a measurement of how quickly a fire spreads within a wooden structure, based on relative humidity. Combining the IFD with the wind conditions is thought to give the user a complete picture of the fire risk.

### 4.13.1 Tab navigation

Navigating in the application happens through a menu that is always visible on the bottom of the screen. The mobile application's menu has one button for each of the available pages and indicates which page the user is currently on by being highlighted with a different colour. In Figure 16, the navigation menu is visible, displaying the available pages and the currently viewed page.



Figure 16. MyLocationsPage with closed and open expander(s)

#### 4.13.2 Items Source

The application presents a list of user-defined `Locations`. Since the size and values of these lists can vary depending on the user's input, the application must construct visual UI objects for each `Location` dynamically. The application accomplishes this by binding the list to the `Layout`. Using the `ItemsSource` property of the `Layout` control, as the `Layout` is responsible of displaying. The `Layout` holds a `Data Template` that defines how a `Location` should appear on the screen, and the application constructs an object for each `Location` in the list.

#### 4.13.3 Data Template

A `Data Template` is commonly used when displaying data from a list of objects [58]. The `Data Template` offer a way to write code that will be applied to all the objects in the list, to handle the case where the number of items in the list can vary.

#### 4.13.4 Expander

The purpose of the application is to give the users an overview of the fire risk for the upcoming days. To satisfy this purpose, the *Expander Control* from Xamarin Community Toolkit is used. The *Expander* consists of a header and a body where the header is always visible, and the body is displayed or hidden by tapping the header [59]. As seen in Figure 16 (image 1), the header contains the name of the `Location`, the highest measured fire risk, and wind values for the coming days. The body contains more detailed information about individual fire risk and wind values for each day. Figure 16 (image 2) shows what the *Expander* looks like when the user has tapped the header to reveal the body.

#### 4.13.5 Data Binding and Data Type

Xamarin uses *Data Type* and *Data Binding* to connect the values of the `Location` objects to the UI objects. The *Data Type* specifies the UI object's type, which determines values the child controls can bind to. The *Data Type* of the *CollectionView* (line 1) in Listing 22 is set to a `Location`. The *Data Type* binding allows the *Label*'s `Text` property to be bound to the `Location`'s `Name` property, which becomes the visible text in the UI (line 4).

```
1  <CollectionView x:DataType="model:Location"
2      x:Name="Expander_Holder">
3      ...
4      <Label Text="{Binding Name}"
5          LineBreakMode="NoWrap"
6          Style="{DynamicResource ListItemTextStyle}"
7          VerticalOptions="Center"
8          FontSize="16"
9          FontAttributes="Bold"
10         Grid.RowSpan="2"
11         HorizontalOptions="Start"
12         TextColor="Black"/>
13      ...
14 </CollectionView>
```

Listing 22. *MyLocationsPage.xaml*

### 4.13.6 Fire Risk and Wind

The fire risk is displayed using four columns containing seven colours, as seen in Figure 16 (image 2). The columns are built using rectangles and the fire risk is displayed by adding a *Frame* to the corresponding rectangle with the use of *Data Binding*. The information is displayed within the *Frame*. The Wind is represented in a similar way, with four columns containing text indicating the date, an arrow that rotates in the direction of the wind, and text describing the wind speed and wind gust.

### 4.13.7 Building the User Interface

The design is built using a *Content Page* that holds a *Scroll View*. There is a *Collection View* inside the *Scroll View* which binds to the list of saved `Locations`. One graphical object is generated for each `Location` using a *Data Template*. The graphical objects are built using an *Expander* that holds several *Views*. Figure 17 shows an overview of how the different *Controls* are combined to create the graphical objects and Listing 23 shows a simplified section of the code behind the graphical objects.



Figure 17. The UI and description of the different Controls

```

1  <ContentPage>
2    <RefreshView x:DataType= "local:MyLocationsViewModel">
3      <CollectionView ItemsSource = "{Binding Locations}">
4        ...
5        <CollectionView.ItemTemplate>
6          <DataTemplate>
7            <CollectionView x:DataType="model:Location">
8              <Frame>
9                <CollectionView>
10               <xct:Expander>
11                 <xct:Expander.Header>
12                   <Grid>
13                     <Label Text="{Binding Name}"/>
14                     <!--Triangle Start--!>
15                       <Polyline/>
16                       <Line/>
17                       <Ellipse/>
18                       <Label/>
19                     <!--Triangle end--!>
20                     <!--WindArrow Start--!>
21                       <Label/>
22                       <Polyline/>
23                       <Label/>
24                     <!--WindArrow End--!>
25                   </Grid>
26                 </xct:Expander.Header>
27                 <xct:Expander.Content>
28                   <Grid>
29                     ... (Content inside expanded Expander)
30                   </Grid>
31                 </xct:Expander.Content>
32               </xct:Expander>
33             </CollectionView>
34           </Frame>
35         </CollectionView>
36       </DataTemplate>
37     </CollectionView.ItemTemplate>
38   </CollectionView>
39   ...
40 </RefreshView>
41 </ContentPage>

```

Listing 23. A simplified code section from MyLocationPage

The Triangle that represents the Peak IFD is created using a *Polyline*, an *Ellipse* and a *Line* (lines 15 – 20). The *WindArrow* (lines 21 – 25) is made with a *Polyline* that rotates using a *Data Binding* to the `Location`'s measured wind. The *Labels* in *WindArrow* are the text displaying wind speeds and *Peak Wind*. The *View (Image)* in Figure 17 rotates to illustrate that the graphical object is expendable.

## 4.14 Data Updates and Notifications

The mobile application is entirely responsible for keeping the fire risks up to date. This approach reduces the traffic towards the MET servers and avoids unnecessary requests as the system is automated. Per the findings of Halderaker and Evjenth [9], fire risk should be updated every sixth hour. This pattern will serve as the foundation for the application's notification system. The mobile application will have new fire risk data available every six hours as long as the application is opened. This will ensure that the user always has the newest information about the fire risk. To accomplish this, the `onResume` method in `App.xaml.cs` will run a check whenever the application is reopened after being backgrounded. Listing 24 demonstrates the use of the `onResume` which checks whether the fire risk has been updated in the last six hours, if not, the fire risk will be updated.

```
1  protected override async void OnResume()
2  {
3      ...
4      if (Current.Properties.ContainsKey("LastUpdated"))
5      {
6          if(Current.Properties["LastUpdated"] != null)
7          {
8              LastUpdated =
9                  DateTime.Parse((string)Current.Properties["LastUpdated"]);
10         }
11         else
12         {
13             LastUpdated = null;
14         }
15     }
16     if (LastUpdated.HasValue)
17     {
18         if (DateTime.Now > LastUpdated.Value.AddHours(6) && MetID != null)
19         {
20             ... (update logic)
21         }
22     }
23     else
24     {
25         ...
26         if(!MetID.Equals("null"))
27         {
28             ... (update logic)
29         }
30     }
31     if (numupdated != 0)
32     {
33         await Current.MainPage.DisplayAlert("Data Update", "Higher risk detected
34             for " + numupdated + " locations!", "ok");
35     }
36 }
```

Listing 24 simplified representation of the `OnResume` method from `App.xaml.cs`

The application holds a `DateTime` object which stores information about when the data was last updated, this will be checked by `OnResume`. The application will store information in `DateTime` using the `onSleep` method seen in Listing 25. Should there be need for data update, the application will request `Locations` from the database and weather data from the MET servers to calculate the updated fire risks. After having updated the data, the `DateTime` object will be updated to hold information about the date and time when it was last updated. Figure 18 shows a notification after the fire risk has been updated, where two `Locations` have had an increase in fire risk.

```
1  protected override void OnSleep()
2  {
3      if (LastUpdated.HasValue)
4      {
5          Current.Properties["LastUpdated"] = LastUpdated.Value.ToString();
6      }
7  }
```

Listing 25 simplified representation of the `OnSleep` method from `App.xaml.cs`

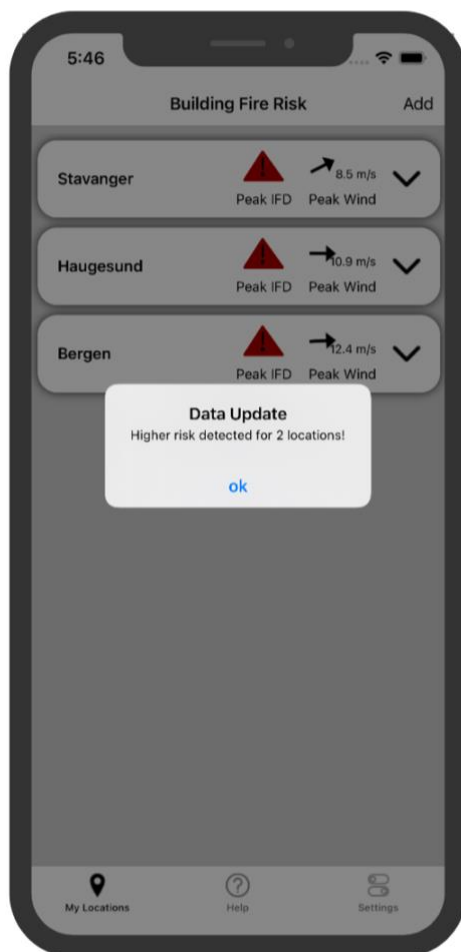


Figure 18. Notification of increased fire risks for two locations

# Chapter 5: Evaluation

The performance and usability of the mobile application were assessed through testing done by different groups of users. The user testing took place in Bergen and Haugesund since members of the user groups lived there. The SUS scale was used to further assess the evaluation results.

## 5.1 Evaluation method

Prior to the evaluation, a plan was created that described the schedule and guidelines of how the application is going to be tested. Many of the essential functions, such as retrieving weather data and doing actual fire risk predictions were missing from the application. Instead, dummy data was utilized to simulate how the application would look with the functions.

During the evaluation, each user group was given a case in which they were instructed to add three new locations: Bergen, Stavanger, and Haugesund. This can be seen in Appendix 9.4 (User Testing Guide). Users were provided geographical coordinates for the cities and were required to utilize the *Get My Location* function to acquire coordinates for the city they were in.

After the evaluation, the users replied to a SUS questionnaire consisting of ten questions related to the application's usability. Each question had five response options where the user picked a suitable response option, based on their experience with the application. The response options related to the application can be seen in Figure 19 ranging from one to five, where 1 indicates Strongly disagree and 5 Strongly agree.





Figure 19. SUS sample statement

The results from the survey will be assessed using the curved grading scale [60], shown in Table 2. The grading scale is used to grade the score of the users, which indicates A for superior performance and F for failing performance [61]. The complete results can be seen in Appendix 9.5 (SUS results).

Grade	SUS score
A+	84.1 – 100
A	80.8 – 84.0
A-	78.9 – 80.7
B+	77.2 – 78.8
B	74.1 – 77.1
B-	72.6 – 74.0
C+	71.1 – 72.5
C	65.0 – 71.0
C-	62.7 – 64.9
D	51.7 – 62.6
F	0 – 51.6

Table 2. Curved Grading Scale for SUS

### 5.1.1 Fire Safety Group

As part of the evaluation-plan, the application was evaluated in Haugesund where it took place at the HVL campus on the 19<sup>th</sup> of April 2022. The application was evaluated here by a selected group of people with expertise in the areas of fire safety and contingency management. Some of them are also members of the DYNAMIC research group.

### **5.1.2 Laypeople Group**

The application was tested in Bergen by a selected group of people with no experience in the fields of fire safety and risk management.

### **5.1.3 Fire Brigade Group**

The application was tested by members of Fire Brigades in Bergen and Haugaland. It was tested by two different Fire Brigades that operate in the Haugaland region, the local Fire Brigade in Haugesund and Haugaland Brann og Redning (HBRE) which operates intermunicipal. The application was evaluated by people with varied backgrounds in the Fire Brigades, including fire fighters, fire safety engineers, and fire analysts.

## **5.2 User testing results**

The post-evaluation feedback differed depending on which group was testing the application. There were numerous proposals for improvements and new ideas for the application. The average impression of the application was good, but there were still some suggestions for improvement and other feedback:

- Change the landing page to the help page at first launch
- Clear instructions on the first page
- Add a homepage with the name of the application
- Give the user a validation when data is submitted
- Change the direction of the wind symbol to match peak wind of the associated day
- Make the graphs for the risk display 10% wider and 10 % higher
- Provide an explanation of the graphs and the colours on the help page
- Change the colours on the tabs, they were difficult to see
- Make it clearer which day is today in the risk display
- Add more information on the help page
- Add a guide on how to share location for the “get my location” function, in the scenario of someone cancelling at first and wanting to undo
- Add yesterday’s graph at the risk display, so that the user can compare

- Some preferred a website over an application
- Remove the need for a MET ID.
- The use of TTF was unclear
- Add pre-defined locations only by specifying the name of a location (e.g., Bergen)

### **5.2.1 Fire Safety Group**

The Fire Safety Group had a test panel of five people and an average score of 84 which maps to the grade C+. The person with the highest score had a score of 95 which maps to the grade A+ and the lowest was 70 which maps to the grade C.

### **5.2.2 Laypeople Group**

The Laypeople Group had a test panel of 11 people and an average score of 65,7 which maps to the grade C. The person with the highest score had a score of 80 which maps to the grade A- and the lowest was 50 which maps to the grade F.

### **5.2.3 Fire Brigade Group**

The Fire Brigade Group had a test panel of six people and an average score of 64,2 which maps to the grade C-. The person with the highest score had a score of 85 which maps to the grade A+ and the lowest was 42,5 which maps to the grade F.

### **5.2.4 Post evaluation**

Several users had difficulties getting started with the application. They had troubles knowing what to do at certain times, and what was happening because of the lack of response from the application. A function to alert the user when actions were submitted was missing. This impacted the overall user experience because many users wasted time navigating back and forth trying to resubmit the MET ID in the hopes of receiving a response from the application.

Questions about the need for a MET ID were frequently asked by the user groups. Many users expressed concerns that the use of MET ID makes the application more difficult to use. They complained that navigating back and forth to MET's website to get the MET ID is cumbersome, and that as a result, less people will use the application.

Several users complained that TTF's meaning was difficult to understand, that it was complex, and that it was hard to relate TTF to the coloured bar charts. IFD was introduced because of this.

## 5.3 Performance evaluation

Different performance tests were performed on the application to assess Data Usage, Storage, CPU, Memory, and Battery. All the evaluations for iOS have been conducted on an iPhone XS running iOS 15.4.1 and the evaluations for Android have been conducted on a Samsung Galaxy S10 running Android 12.

### 5.3.1 Data Usage

An experiment was done to evaluate the application's Data Usage. The purpose of this experiment was to see if data usage increases in a linear relationship with the locations. The research started with creating a location, the data usage for updating the fire risks was measured on both devices. The results showed that the difference between the two operating systems was minimal:

- The application when run on the iPhone consumed 165 KB of data
- The application when run on the Samsung Galaxy consumed 177 KB of data

It was decided to use an average of 171 for the further assessment of the data usage. This was decided because the difference in data used per request was minimal between an iPhone and a Samsung Galaxy. Further research regarding requests were conducted using the iPhone since the difference concerning data usage was minimal. A function was made using the average data usage per request as seen below. This function finds out how much mobile data the application uses to perform four updates for n number of locations. Four updates were chosen for this experiment, this is because the data will be updated four times a day if the application is used as intended.

$$f(n) = 4 \times n(\text{average data usage})$$

The purpose of this function is to compare its output with experimental results to evaluate whether the data usage increases linearly or not. The experimental results will be derived from a mobile device running the application.

Four updates on the mobile device for one location took 666 KB as seen through point A. Point B shows 2 MB data consumed for three locations, point C shows 3,1 MB data consumed for five locations, point D shows 4,4 MB data consumed for seven locations and point E shows 6,7 MB of data consumed for ten locations. The plotted points can be seen in Figure 20. The difference between the plotted points and the function is minimal. This means that the function can be used as a representative function for the data usage, as it shows that the data usage increases linearly.

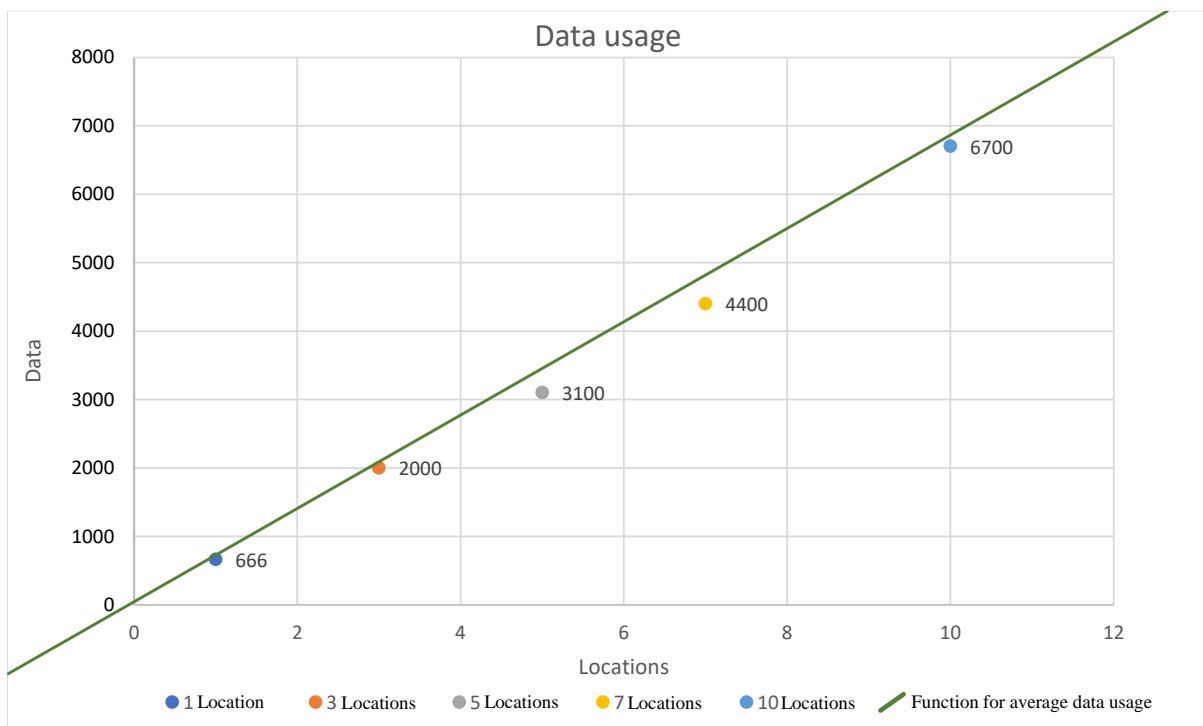


Figure 20. The expected and actual data usage plotted in a graph

### 5.3.2 Storage

The size of the application on the iPhone was measured to be 44.6 MB, the *Documents & Data* is measured to be around 254 kB. The *Documents & Data* alternated around 70-80 kB when new Locations were added or deleted.

The application was measured to consume 51.7 MB on the Samsung Galaxy. The data was 123 kB and increased by 8kB whenever a new location was added.

In comparison to similar apps for various operating systems, the application size, including data, is larger [62]. While many of the applications are designed specifically for one operating systems, this application is cross-platform. Extra libraries and other extensions will be included in the application's file packages to ensure cross-platform compatibility. Another factor is the use of edge computing, which could mean that there will be more code because the mobile device will be doing all the computational work.

### 5.3.3 CPU and memory

The application's CPU and memory usage on the iPhone was measured using the iOS debugger in Apple's developer tool; Xcode. The application was launched and Xcode was then attached to the application's process. The debugger showed statistics of the phone's CPU and memory usage while the process ran.

As seen in Figure 21, the application reached a peak of 50% CPU usage during the profiling. The number may seem high, but 50% implies that the application was using half of the processing power of one core, out of six possible cores. This essentially means the application used 1/12th of the CPU's processing power at its peak. When the screenshot was taken, the application used 32% of the CPU, other processes used 131%, and 437% of the CPU's processing power was left free. When no actions were made in the app, the CPU usage was at 0%.

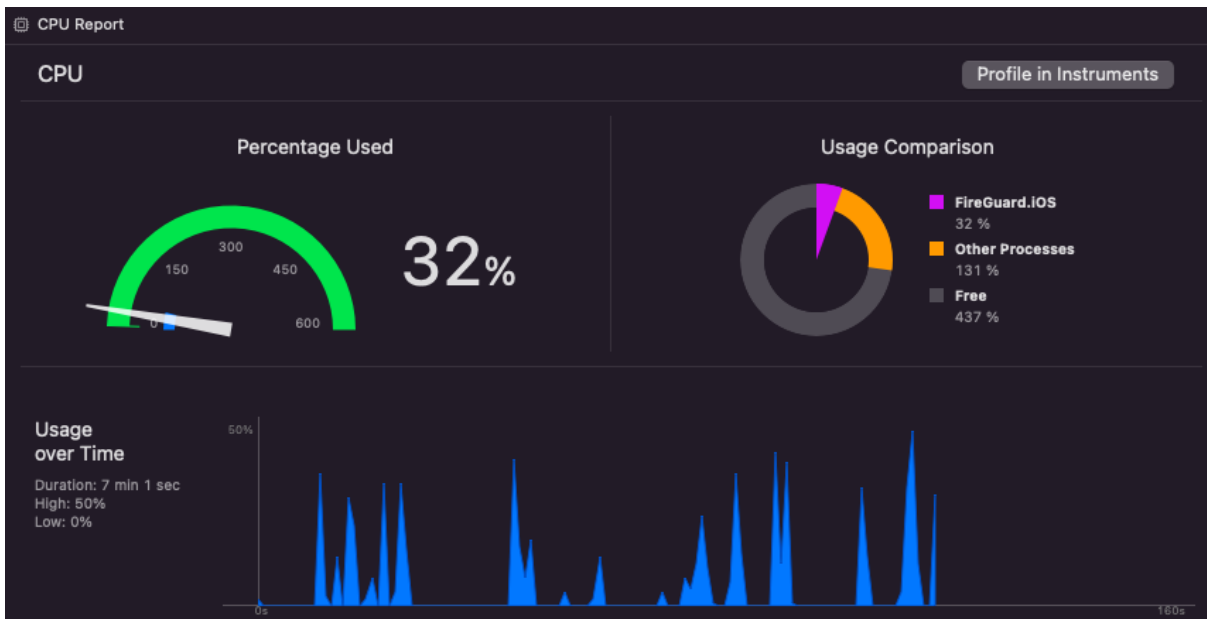


Figure 21. The application's CPU usage on an iPhone XS

The application's memory usage is displayed in Figure 22. The memory usage hovered around 129MB, with a peak at 131.3MB which was about 3.4% of the available memory on the iPhone. Profiling can only be done on applications that allow profiling. New released application usually has this setting off. This led to using a source for further profiling which has documented statistics about the average RAM usage on iOS and Android devices [63]. The mean RAM usage for an iOS application was 146.6MB, according to the source [63]. This shows that the mobile application's RAM usage is right below average.

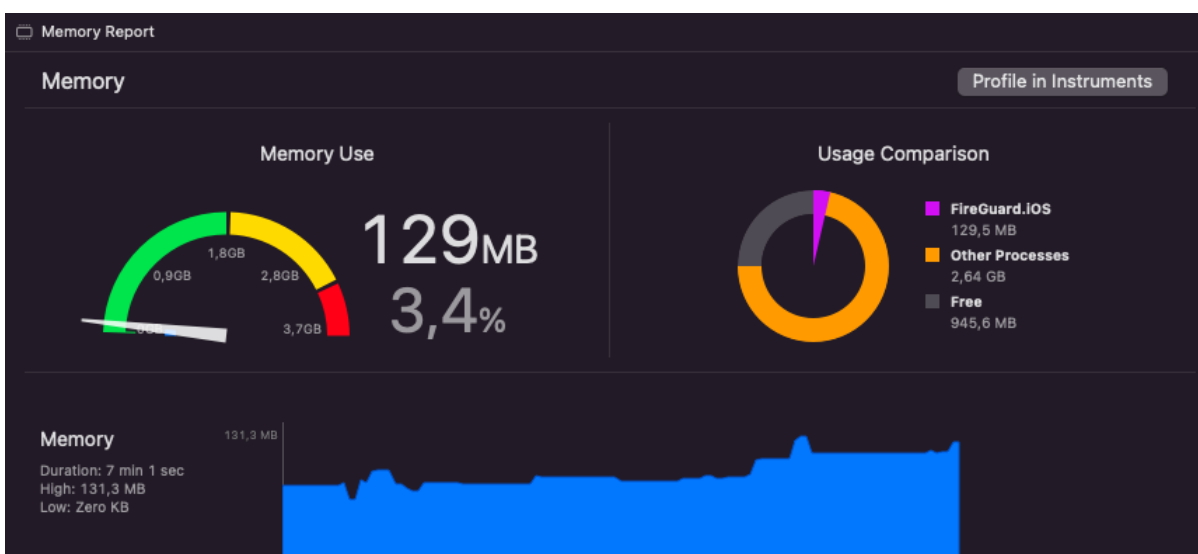


Figure 22. The application's memory usage on an iPhone XS

The application's RAM and CPU usage for the Samsung Galaxy was measured using Android Studio's profiling tool. The Android Profiler was used to assess the signed apk file generated by Visual Studio.

The CPU usage for the Samsung Galaxy was around 40% on launch as seen in Figure 23. The usage was at 0% when the application was idle. Performing actions caused the CPU usage to spike for a couple of seconds while it made changes and re-rendered. The CPU usage was measured out of 100%, meaning the application used 2/5 of the total CPU processing power at its peak. This still leaves 3/5 of the CPU available for other unrelated tasks.

The RAM usage on the Samsung Galaxy was slightly higher than the RAM usage the iPhone, but as seen in the article, higher RAM usage was expected [63]. The results from the article showed that the mean RAM usage of an Android application was 302.7MB [63]. As seen in Figure 23 the peak RAM usage on the Samsung Galaxy was around 320MB which is slightly above average.

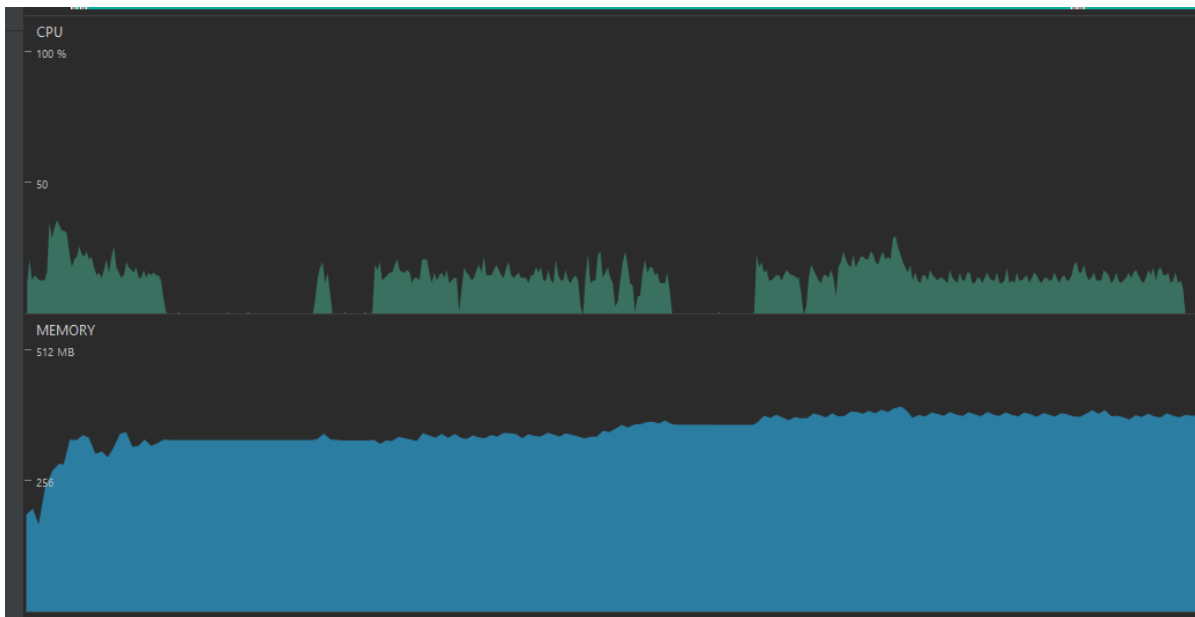


Figure 23. Android CPU and RAM profiling results on a Samsung Galaxy s10



### 5.3.4 Battery

The battery usage of the application had to be evaluated to as a part of the performance evaluation. Three locations were monitored for ten days, and the data was updated four times per day. This test could not be performed on an Android device because the group could not get access to one for ten days. The iPhone's battery use was 2%, with screentime of one hour and 11 minutes.

Xcode was used to track the application's energy usage on the iPhone. Energy usage evaluation includes an assessment of Network, CPU, GPU, Location services, and Background services [64]. The application's energy usage proved to be low. The energy usage increased barely when a new location was added. As no background tasks were implemented, the Background energy usage was zero. This corresponds with the fact that no activities were implemented in the application that would run in the background. An overview of the application's energy usage can be seen in Figure 24.

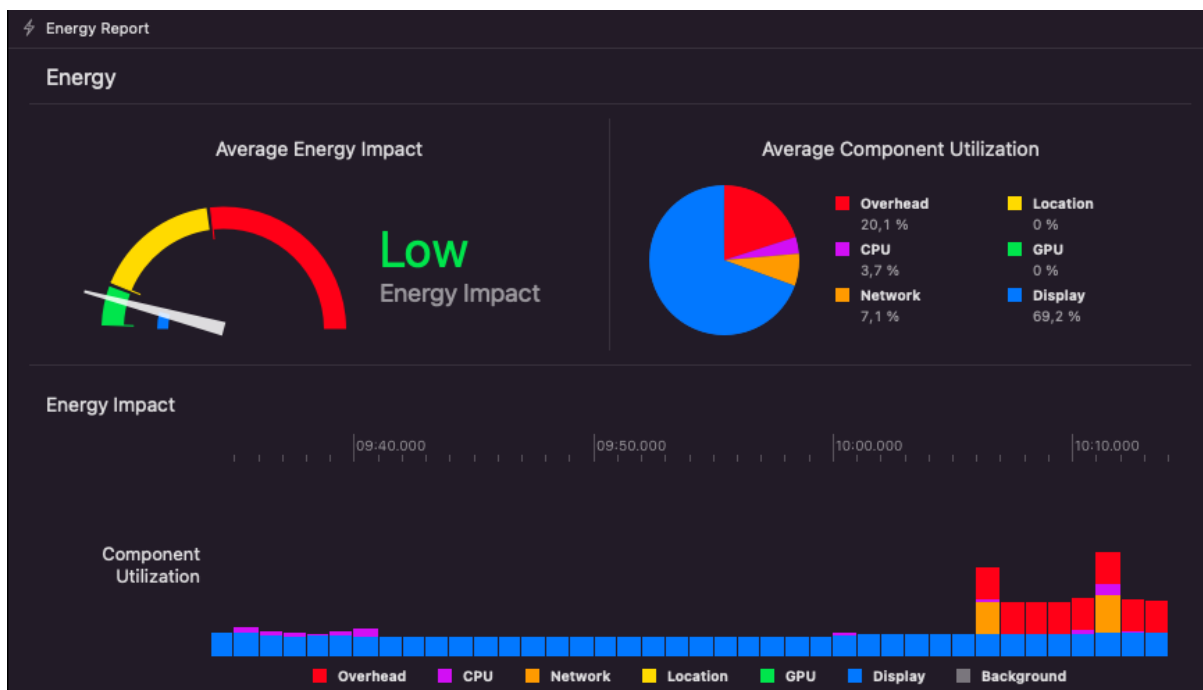


Figure 24. Energy usage of the application running on iPhone XS

An energy usage evaluation was also done for the Samsung Galaxy using Android Studio's application profiling tool. The energy usage can be seen in Figure 25. There are noticeable spikes in energy usage when the locations were created and updated.

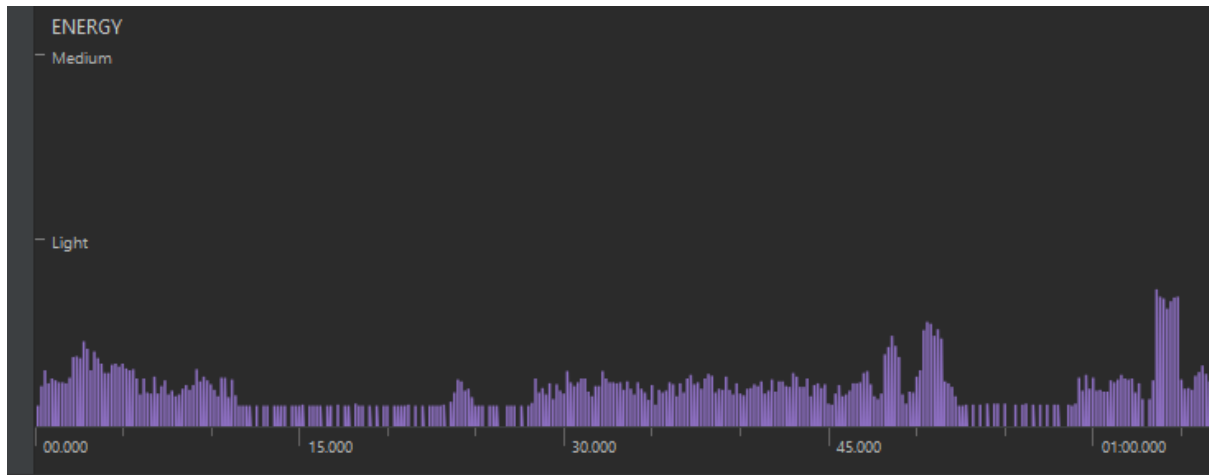


Figure 25. Android energy usage profiling results on a Samsung Galaxy s10

# Chapter 6: Discussion

The project progressed mostly according to plan, however there were some deviations from the plan. The deviations were mostly due to revisions and some of the activities taking longer than anticipated.

The project plan served as an excellent reminder for many of the activities, and to complete the scheduled activities within the deadlines. Due to the project group's lack of experience, determining the duration of the various phases for the project plan was difficult. During the creation of the project plan, it was anticipated that the development phase would be the most difficult and time-consuming. This was the case because many of the problems encountered during this phase were connected to instantiating and configuring the various components.

Many of the deviations happened due to inexperience, and this was the group's first time working on a project of this scale. Another reason for many of the deviations was that several members of the group contracted covid-19 at the same time, which temporarily halted the progress. As a result, many of the scheduled activities had to be postponed. Meetings were held as planned during this time, but accordingly with the covid-19 regulations. This was not a problem because meetings were already partially digital, allowing the project owner to attend from Haugesund.

The project emphasized an approach where continuous dialogue with the project owner was held in addition to using principles from Scrum. Following Scrum's iterative and incremental process principles helped deliver a result that was aligned with the project owner's expectations.

A consequence of using Scrum was that many of the phases and activities' deadlines were difficult to meet. New approaches to many of the activities were discussed during the meetings. Some of them were picked because they proved to be better than the already decided ones.

The project owner was mostly clear when expressing his wishes for the application, but some freedom of choice was also given. Prototypes in the form of drawings were shown to the group.

These prototypes served as a starting point for the application's GUI design. The first prototype was sent by the project owner after the group presented the wireframe which can be seen in Appendix 9.6 (Wireframe). The prototype for the GUI can be seen in Figure 26 (image 1), the prototype for the final iteration can be seen in Figure 26 (images 2 and 3).



Figure 26. Various prototype proposals from the project owner

The group's interpretation of the prototypes is seen in Figure 27. In addition to presenting risk for fire, it was decided that the application will also give the user an overview of the wind speed and wind direction. This is because the wind is crucial information for determining how the fire will evolve. It was decided to include a link to yr.no which directs the user to the weather page for the same coordinates as the location. This will save the users time if they are interested in the weather for the specific location while maintaining the application's status as an application for fire risk predictions for homes.



Figure 27. The evolution of FireGuard

The feedback from the evaluation revealed inconsistencies in the GUI. The original plan was to present fire risks with TTF, but many users misinterpreted this. As a result, the term IFD was introduced. The assumption is that IFD is a more generic phrase since it can be understood by the majority and easier to relate with the coloured bars. This can be seen in Figure 27 (image 4).

Many users complained having a tough time getting started with the app. They were unsure about what to do next after launching the application. This revealed that the application was not as simple as first anticipated. Changes were made in the application to give users feedback and instructions; the intention was to simplify the application. A guide to getting started as seen in Figure 28 (image 1), and a verification message after submitting the MET ID as seen in Figure 28 (image 2) were added.

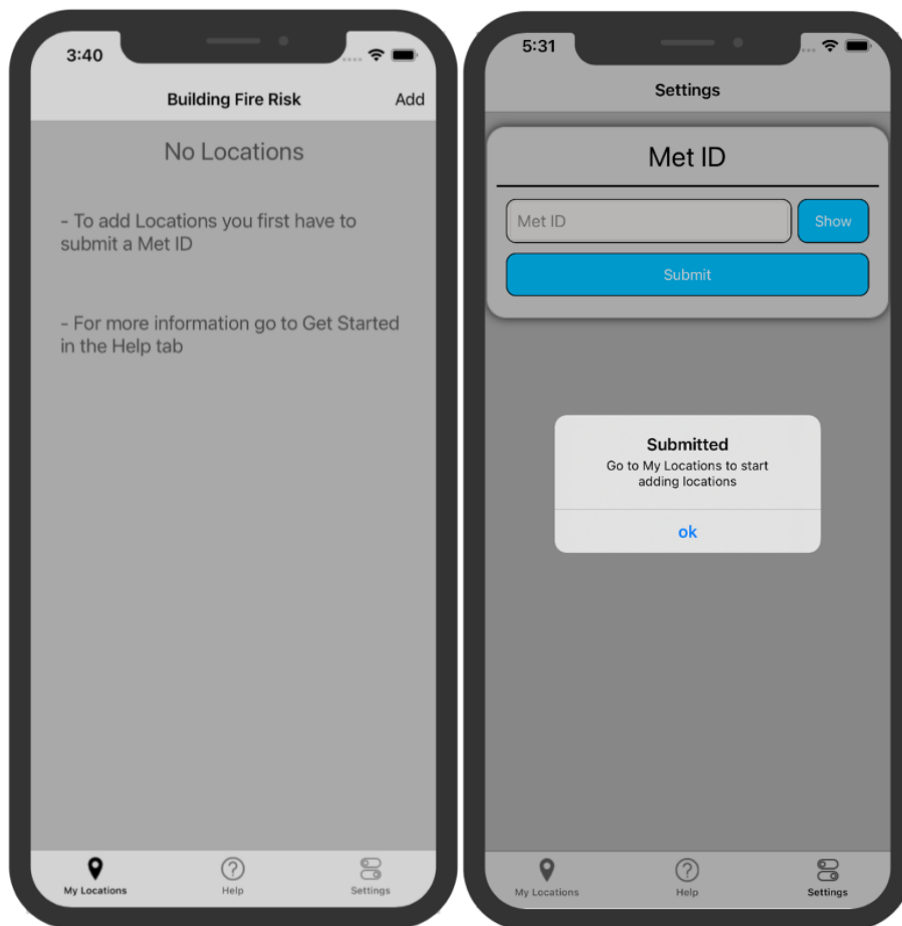


Figure 28. Changes made in the application to give users feedback

Scheduling of data fetches was not implemented as initially planned. This concept's implementation turned out to be more complex and time-consuming than anticipated. The fire risk was supposed to be updated every six hours automatically in the background. Since scheduling background tasks proved to be a much larger and time-consuming task, an alternative approach was implemented. This approach was elaborated in Section 4.14 (Data Updates and Notifications).

The weather data sources MET, and Frost have documents describing the terms of use [65] [66] and state that one must not cause unnecessary traffic when utilizing their services and emphasize the importance of respecting their guidelines. They continue stressing that mobile applications should not directly use client-to-API connections, but instead use a local proxy server that can cache data and add authentication details for requests. Implementing a proxy server combined with caching would handle all communications with the Frost and MET API servers and cache all the requested weather data. This would reduce the requests and stress on

MET's servers. Using a proxy server would also eliminate the requirement for users to obtain their own MET ID.

During the coupling of the model and the mobile application, it was observed that the `Fire Risk Model` was producing inaccurate results. Several attempts were made to debug the issue together with the project owner, since the group members' understanding of the model was limited. It was decided that the debugging will be continued in the `Finishing Phase` together with the project owner and hopefully determine the model's irregular behavior.

Many adjustments would be made if the project was to be re-implemented. The project plan would be developed to be more realistic concerning the uncertainties when the project plan was developed. The group has gotten a more realistic perception of how a project plan should be developed and a better perception of how much time the different phases take.

The coupling of the `Fire Risk Model` with the mobile application would also be done earlier in a re-implementation of the project. Coupling the `Fire Risk Model` with the mobile application could have saved the project group a lot of time as other stakeholders could have helped investigating the problem. There was not enough time for this process when the problem was first discovered. The problems with the `Fire Risk Model` were not expected as it has successfully been implemented to give correct fire risk predictions in a project earlier [9].

# Chapter 7: Conclusion

In this bachelor thesis, it has been investigated how a fire risk notification service may be provided to users via a mobile application using edge computing. The objective was to conduct research into an approach where all acquisitions of weather data and fire risk computation were performed directly on a mobile device.

## 7.1 Research Questions Revisited

In this section, the research questions will be reconsidered and put into context of the results obtained.

**RQ1: Can a user interface for a mobile application be designed capable of showing fire risks for multiple locations and be suited for users with different background and use cases?**

The developed application's user interface has been assessed at different stages through user testing by people with different backgrounds. The comprehensibility of the user interface was the primary focus. The user testing showed that the application's user interface was not as simple as first anticipated. The feedback from the evaluation was used to further develop the application with a user interface capable of showing fire risk notifications for multiple locations that meets the users' standards with a high degree of usability.



**RQ2: What is the impact of the fire risk computation and weather data acquisition on the performance of the mobile device?**

The developed application has low impact on the mobile device's performance. The application used expected amounts of RAM and CPU on both iOS and Android, leaving much of the device's processing power available for other activities. The application has no background processes running; meaning it has no effect on the device's performance while it is closed.

**RQ3: What are the advantages and disadvantages of an approach based on edge computing on mobile devices in comparison to a cloud-based solution?**

An approach based on edge computing has several advantages. The fire risk will be available on the mobile device until the next request without the need of an internet connection. Another advantage is processing speed; accessing the fire risk takes less time because it is stored locally on the device.

The weather data that the application fetches is lost after being processed by the mobile device, which is a disadvantage of edge computing. Another disadvantage is the increased traffic towards MET's servers, because of all data retrieval being done by the mobile devices. A cloud-based solution would have used a cloud service dedicated to retrieving, processing, and storing data available to all devices. Instead of requesting data on their own, other devices would use the cloud service to obtain data.

## **7.2 Conclusion**

Several experiments were conducted to evaluate the application's usability and performance. According to the results of the SUS survey and the overall perception of the test-panel, the application is better suited for some individuals and organizations with a particular interest in the domain of fire risk, such as fire brigades, volunteer organizations and municipalities.

Based on the Objective and Problem Statement, the group concluded that a mobile application utilizing edge computing would be a good way of providing a fire risk assessment service to the general public. The solution would be reliable and fast, and the impact on the device's performance would be minimal. The initial idea was that the mobile device would be responsible of performing weather data acquisition on its own. Through research, it was discovered that MET and Frost APIs terms of use [\[65\]](#) [\[66\]](#) would not allow this as a long term solution. The application cannot be released in its current state; however, a proxy-caching server could be used to solve this problem.

### **7.3 Future Work**

The project's future work will consist of further developing the application. During the evaluation, the project received many helpful suggestions and other feedback that might be used to improve the application. Because of the project's limited time, it was impossible to properly address all the feedback. Suggestions that have not been addressed had a lesser priority than those that were evaluated. The suggestions were prioritized based on the time it would take to implement and relevance to the project. The remaining suggestions can be reassessed and if found useful implemented to the application.

Before the application can be released to the public, a proxy server must be added. This is necessary to provide a smooth user experience. Without a proxy server the application might create unnecessary traffic towards the MET servers and possibly denying them from performing their other services.

The problems of scheduling background data updates that was encountered during the project can be handled to allow notifications without the need of opening the app every six hours. The Nuget package Shiny.Jobs may be a viable option to develop a function that fills this need [\[67\]](#). Shiny provides a cross platform interface that manages scheduling of jobs for Xamarin applications. Although the background tasks cannot be set at specific times, it allows some control over when the data updates by checking when the job was last run.

# Figures

FIGURE 1. COLOR SCHEME FOR DISPLAYING FOREST FIRE RISK .....	9
FIGURE 2. XAMARIN PROJECT OVERVIEW .....	13
FIGURE 3. MAUI PROJECT OVERVIEW.....	13
FIGURE 4. UML USE CASE DIAGRAM.....	19
FIGURE 5. DOMAIN MODEL OF THE APPLICATION .....	20
FIGURE 6. STRUCTURE OF MODEL-VIEW-VIEW MODEL .....	21
FIGURE 7. HIGH-LEVEL APPLICATION SOFTWARE ARCHITECTURE.....	22
FIGURE 8. SEQUENCE DIAGRAM WHEN A USER ADDS A NEW LOCATION .....	23
FIGURE 9. SEQUENCE DIAGRAM FOR PUSH NOTIFICATION.....	23
FIGURE 10. SCALING OF THE TTF VALUES .....	24
FIGURE 11. A REPRESENTATION OF CLASSES IN THE MODELS COMPONENT .....	33
FIGURE 12. A REPRESENTATION OF CLASSES IN THE SERVICES COMPONENT.....	34
FIGURE 13. THE ERROR MESSAGE WHEN A USER HAS SUBMITTED AN INVALID MET ID.....	39
FIGURE 14. STATE MACHINE DIAGRAM TO DISPLAY THE FLOW. ....	40
FIGURE 15. A REPRESENTATION OF THE CLASS IN THE DATA FOLDER.....	41
FIGURE 16. MYLOCATIONS PAGE WITH CLOSED AND OPEN EXPANDER(S).....	50
FIGURE 17. THE UI AND DESCRIPTION OF THE DIFFERENT CONTROLS.....	52
FIGURE 18. NOTIFICATION OF INCREASED FIRE RISKS FOR TWO LOCATIONS.....	55
FIGURE 19. SUS SAMPLE STATEMENT.....	57
FIGURE 20. THE EXPECTED AND ACTUAL DATA USAGE PLOTTED IN A GRAPH.....	61
FIGURE 21. THE APPLICATION'S CPU USAGE ON AN IPHONE XS .....	63
FIGURE 22. THE APPLICATION'S MEMORY USAGE ON AN IPHONE XS.....	63
FIGURE 23. ANDROID CPU AND RAM PROFILING RESULTS ON A SAMSUNG GALAXY S10 .....	64
FIGURE 24. ENERGY USAGE OF THE APPLICATION RUNNING ON IPHONE XS .....	65
FIGURE 25. ANDROID ENERGY USAGE PROFILING RESULTS ON A SAMSUNG GALAXY S10 .....	66
FIGURE 26. VARIOUS PROTOTYPE PROPOSALS FROM THE PROJECT OWNER .....	68
FIGURE 27. THE EVOLUTION OF FIREGUARD .....	69
FIGURE 28. CHANGES MADE IN THE APPLICATION TO GIVE USERS FEEDBACK .....	70

# Listings

LISTING 1. FINDFIRERISK DETERMINES FIRE RISKS BASED ON THE TTF VALUE. ....	25
LISTING 2. A JSON RESPONSE BODY FROM THE COMPLETE ENDPOINT .....	27
LISTING 3. A JSON RESPONSE BODY FROM THE SOURCES ENDPOINT .....	29
LISTING 4. A JSON RESPONSE BODY FROM THE OBSERVATIONS ENDPOINT .....	31
LISTING 5. THE OBSERVATION OBJECT FROM THE FIRE RISK MODEL PROJECT .....	33
LISTING 6. REPRESENTATION OF THE LOCATION OBJECT .....	34
LISTING 7. IRESTSERVICECLIENT'S METHODS .....	35
LISTING 8. IWEATHERCLIENT'S METHODS.....	35
LISTING 9. GETWEATHEROBSERVATIONS METHOD FROM THE RESTSERVICECLIENT CLASS ....	36
LISTING 10. NEWLOCATIONVIEWMODEL'S METHOD ONSAVE HANDLES AN EXCEPTION. ....	37
LISTING 11. WEATHERCLIENT'S METHOD GETOBSRVATIONS .....	38
LISTING 12. REPRESENTATION OF THE ERRORMESSAGE METHOD .....	39
LISTING 13. THE DATABASE CLASS.....	42
LISTING 14. METHOD TO SAVE A LOCATION TO THE DATABASE. ....	44
LISTING 15. IFIRERISK'S METHODS.....	45
LISTING 16. IFIRERISK'S METHOD FIRERISKTTF .....	45
LISTING 17. THE METHOD SORTBYHIGHESTTTF.....	46
LISTING 18. THEGETFIRERISK METHOD FRON THE WEATHERCLIENT CLASS.....	46
LISTING 19. THEGETFIRERISK METHOD FRON THE WEATHERCLIENT CLASS.....	46
LISTING 20. WEATHERCLIENT'S METHOD GETFIRERISKFORNEWLOCATION.....	47
LISTING 21. WEATHERCLIENT'S METHOD UPDATEFIRERISK.....	47
LISTING 22. MYLOCATIONSPAGE.XAML.....	51
LISTING 23. A SIMPLIFIED CODE SECTION FROM MYLOCATIONPAGE.....	53
LISTING 24 SIMPLIFIED REPRESENTATION OF THE ONRESUME METHOD FROM APP.XAML.CS....	54
LISTING 25 SIMPLIFIED REPRESENTATION OF THE ONSLEEP METHOD FROM APP.XAML.CS .....	55

# Tables

TABLE 1. REPRESENTATION OF THE LOCATION AND MET ID TABLES.....	43
TABLE 2. CURVED GRADING SCALE FOR SUS .....	57

# Bibliography

1. Metallinou, M.-M. and T. Log, *Cold Climate Structural Fire Danger Rating System? Challenges*, 2018. **9**(1): p. 12.
2. *Reducing fire disaster risk through dynamic risk assessment and management (DYNAMIC)* 2021 7. february 2021, 15:11; Available from: <https://app.cristin.no/projects/show.jsf?id=2495578>.
3. Log, T., *Cold climate fire risk; A case study of the Lærdalsøyri Fire, January 2014*. Fire technology, 2016. **52**(6): p. 1825-1843.
4. Log, T., *Indoor relative humidity as a fire risk indicator*. Building and Environment, 2017. **111**: p. 238-248.
5. Log, T., *Modeling Indoor Relative Humidity and Wood Moisture Content as a Proxy for Wooden Home Fire Risk*. Sensors, 2019. **19**(22): p. 5050.
6. Kraaijeveld, A., *Burning Rate and Time to Flashover in Wooden ¼ scale Compartments as a Function of Fuel Moisture Content*. 2016: p. 6.
7. Karlsson, B., & Quintiere, J. G, *Enclosure Fire Dynamic 2000*: CRC Press LLC.
8. Stokkenes, S., *Implementation and Evaluation of a Fire Risk Indication Model*. 2019, The University of Bergen.
9. Halderaker, E.D., & Evjenth, A, *Development and Evaluation of a Software System for Fire Risk Prediction*. 2021, Western Norway University of Applied Sciences
10. HVL. *Fire Disasters*. 2021; Available from: <https://www.hvl.no/en/research/group/fire-disasters/>.
11. HVL. *Software Engineering*. 2021; Available from: <https://www.hvl.no/en/research/group/software-engineering/>.
12. Bigelow, S.J., *What is edge computing? Everything you need to know*. techtarget, 2021.
13. Gupta, J. *What are the pros and cons of cloud computing?* 2016; Available from: <https://www.znetlive.com/blog/pros-and-cons-of-cloud-computing/>.
14. IBM. *What is edge computing?* ; Available from: <https://www.ibm.com/cloud/what-is-edge-computing>.
15. Mamalgaha, L. *What is a Proxy Server?* 2019; Available from: <https://medium.com/@lakshanmamalgaha/what-is-a-proxy-server-f0a6f685fba8>.
16. Gibb, R. *What is Proxy Caching?* 2016; Available from: <https://blog.stackpath.com/proxy-caching/>.
17. IBM. *Caching Proxy*. 2022; Available from: <https://www.ibm.com/docs/en/was-nd/9.0.5?topic=overview-caching-proxy>.
18. Meteorologisk-Institutt. *Skogbrannfare*. Available from: <https://skogbrannfare.met.no>.
19. Meteorologisk-Institutt. *WeatherAPI*. Available from: <https://api.met.no>.

20. Thomas, G. *What is Flutter and Why You Should Learn it in 2020*. 2019.
21. Clark, J. *Flutter vs Dart - Which is better?* ; Available from: <https://blog.back4app.com/flutter-vs-dart/>.
22. *What is Xamarin?*, in *Microsoft Documentation*. 2021, Microsoft.
23. Ramel, D., *With .NET MAUI Delayed, Xamarin.Forms Remains Mobile Dev Option in .NET 6*, in *Visualstudiomagazine*. 2021.
24. Karasavvas, T. *Why Flutter is the most popular cross-platform mobile SDK*. 2022; Available from: <https://stackoverflow.blog/2022/02/21/why-flutter-is-the-most-popular-cross-platform-mobile-sdk/>.
25. D, C., *.NET Core, .NET Framework, Xamarin – The “WHAT and WHEN to use it”*. 2016, Microsoft: Microsoft DevBlogs.
26. Ewbank, K. *Microsoft Will Replace Xamarin Forms With MAUI .NET 2021* [cited 2022 08.03.2022]; Available from: <https://www.i-programmer.info/news/89-net/14762-microsoft-will-replace-xamarin-forms-with-maui-net.html>.
27. Microsoft. *Migrate your app from Xamarin-Forms*. 2021; Available from: <https://docs.microsoft.com/en-us/dotnet/maui/get-started/migrate>.
28. Brooke, J.B. *SUS: A 'Quick and Dirty' Usability Scale*. 1996.
29. Gibbs, M. *Domain Modeling: Definition & Examples* 2017; Available from: <https://study.com/academy/lesson/domain-modeling-definition-examples.html>.
30. Bandarupalli, K. *Domain Model Using UML*. 2009; Available from: <https://www.techbubbles.com/softwarearchitecture/domain-model-using-uml/>.
31. Britch, D., *Enterprise Application Patterns using Xamarin. Forms*. 2017, Microsoft Press, A Division of Microsoft Corporation, One Microsoft Way ....
32. Meteorologisk-Institutt. *About the Norwegian Meteorological Institute*. 2017 2020; Available from: <https://www.met.no/en/About-us/About-MET-Norway>.
33. Yr.no. *Facts about Yr*. Available from: <https://hjelp.yr.no/hc/en-us/sections/115001514149-About-us>.
34. Meteorologisk-Institutt, *Locationforecast*.
35. Meteorologisk-Institutt. *What is Frost?* . Available from: <https://frost.met.no/index.html>.
36. Yr.no. *Location Forecast*. Available from: <https://developer.yr.no/featured-products/forecast/>.
37. Meteorologisk-Institutt. *API REFERENCE*. Available from: <https://frost.met.no/api.html>.
38. Meteorologisk-Institutt. *API Concepts*. Available from: <https://frost.met.no/concepts2.html>.
39. Wikipedia. *ISO 8601*. 2022; Available from: [https://en.wikipedia.org/wiki/ISO\\_8601](https://en.wikipedia.org/wiki/ISO_8601).
40. Meteorologisk-Institutt, *Request New Client Credentials*.
41. Meteorologisk-Institutt. *MET's Privacy Policy Statement*. 2017 2021; Available from: <https://www.met.no/en/About-us/privacy>.
42. Meteorologisk-Institutt. *Authentication*. Available from: <https://frost.met.no/authentication.html>.

43. IBM. *HTTP basic authentication*. 2022; Available from: <https://www.ibm.com/docs/en/cics-ts/5.4?topic=concepts-http-basic-authentication>.
44. auth0. *What is OAuth 2.0?* ; Available from: <https://auth0.com/intro-to-iam/what-is-oauth-2/>.
45. quicktype. *Convert JSON into gorgeous, typesafe code in any language*. Available from: <https://quicktype.io>.
46. Strizic, M. *MVVM architecture: a step-by-step guide*. 2017; Available from: <https://decode.agency/article/mvvm-architecture-a-step-by-step-guide/>.
47. RestSharp. *Introduction*. 2022; Available from: <https://restsharp.dev/intro.html#introduction>.
48. RestSharp. *Authenticators*. 2022; Available from: <https://restsharp.dev/authenticators.html#basic-authentication>.
49. Meteorologisk-Institutt. *HTTP status codes 2020*; Available from: <https://api.met.no/doc/StatusCodes>.
50. RestSharp. *Error handling*. 2022; Available from: <https://restsharp.dev/error-handling.html>.
51. javaTpoint. *UML State Machine Diagram*. Available from: <https://www.javatpoint.com/uml-state-machine-diagram>.
52. SQLite. *About SQLite* Available from: <https://www.sqlite.org/about.html>.
53. Docs, M. *Xamarin.Forms Local Databases*. 2021; Available from: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/data/databases#feedback>.
54. TwinCoders. *SQLite-Net Extensions*. 2018; Available from: <https://bitbucket.org/twincoders/sqlite-net-extensions/src/master/>.
55. Debbie Stone, C.J., Mark Woodroffe, *User Interface Design and Evaluation*. 2005: Elsevier.
56. David Britch, D.C., Craig Dunn, Justin Johnson, Timo Salomäki, Charles Petzold, *Xamarin.Forms Layouts*. 2021.
57. David Britch, D.C., Justin Johnson, Nick Schonning, Craig Dunn, Charles Petzold, *Xamarin.Forms Cells*, in *Microsoft Documentations*. 2021, Microsoft.
58. David Britch, D.C., Nick Schonning, Craig Dunn, Charles Petzold, *Creating a Xamarin.Forms DataTemplate*, in *Microsoft Documentation*. 2021, Microsoft.
59. Gerald Versluis, A.M., David Britch, *Xamarin Community Toolkit Expander*. 2021.
60. Lewis, J.R. and J. Sauro, *Item benchmarks for the system usability scale*. Journal of Usability Studies, 2018. **13**(3).
61. Sauro, J. *5 Ways to Interpret a SUS Score*. 2018.
62. Shafi, A., *Do you know the average Android and iOS app file size?* 2020.
63. Sims, G. *Apple vs Android RAM management: Who does it better?* 2022; Available from: <https://www.androidauthority.com/apple-vs-android-ram-management-3100032/>.
64. Apple. *Measure Energy Impact with Xcode*. 2016; Available from: <https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/MonitorEnergyWithXcode.html>.



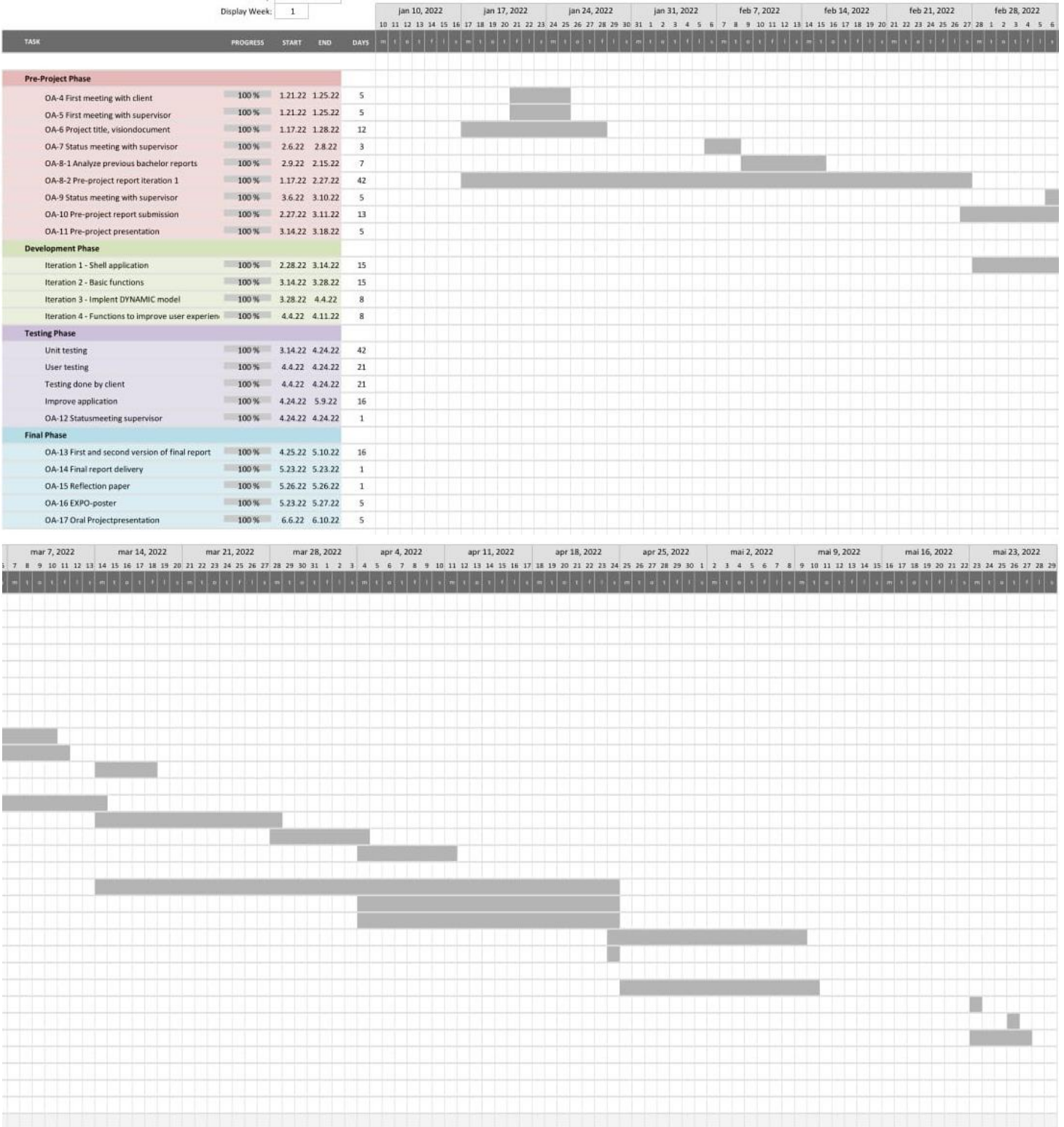
65. Meteorologisk-Institutt. *Terms Of Use* Available from:  
<https://frost.met.no/termsfuse2.html>.
66. Aalberg, G., *Terms of Service* 2020.
67. Ritchie, A., *Background Jobs - Shiny Style*. 2019.

# Appendices

# 9.1 Gantt Chart

## A Mobile Application for Fire Risk Notification based on Edge Computing

Project Start: tir, 1.11.2022  
 Today: søn, 7.24.2022  
 Display Week: 1



View in PDF:

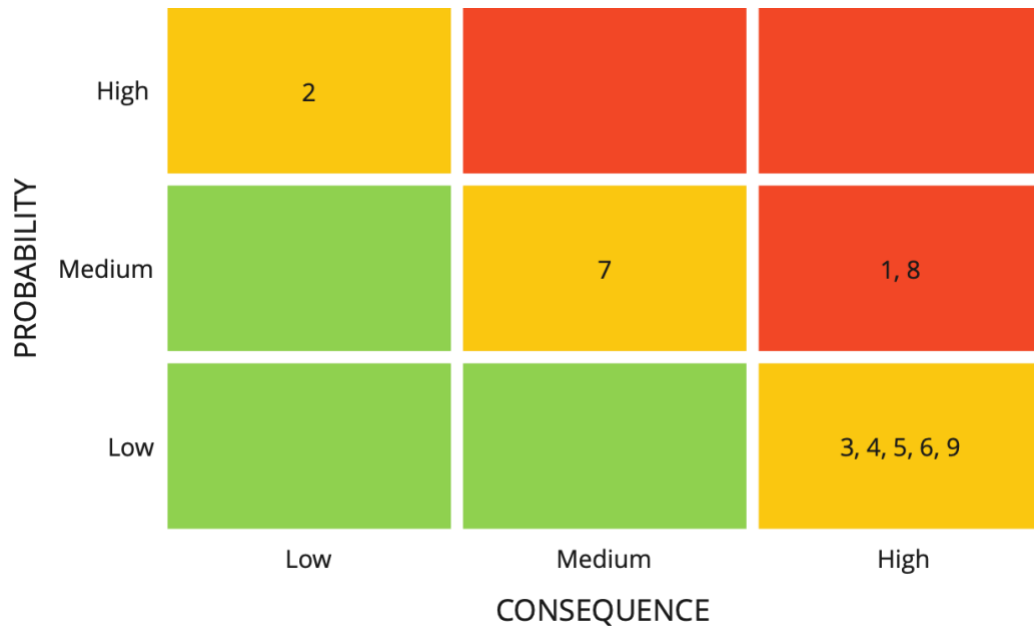
<https://drive.google.com/file/d/1f8rVZuLYKO0Se7RJHEoiEgzE2YHvVO4I/view>

## 9.2 Risk Management

	Risk	Cause	Chance	Conse- quenc	Risk product	Measure
1	<i>Inaccurate fire risk predictions</i>	<i>The Model of Log is implemented incorrectly</i>	3	5	15	<i>Test the model with test data</i>
2	<i>The application will not be used</i>	<i>Not user-friendly and/or resource intensive</i>	4	2	8	<i>Evaluate the performance and usability</i>
3	<i>Not finish developing</i>	<i>Poor planning and time estimation</i>	2	5	10	<i>Good planning and follow-up throughout the project</i>
4	<i>The application does not interact with the Model of Log</i>	<i>Versions not working together or not enough time</i>	2	5	10	<i>Start developing early</i>
5	<i>Unable to fetch weather data</i>	<i>Can't figure out how to retrieve data from API</i>	1	5	5	<i>Set off time studying and testing API calls</i>
6	<i>Hacking attacks</i>	<i>Poor security</i>	1	5	5	<i>Secure communication when retrieving data from API</i>
7	<i>Fails to implement a cross-platform application</i>	<i>The implementation not done correctly</i>	3	3	9	<i>Start developing early</i>
8	<i>Long-term illness</i>	<i>Covid-19 infection or other diseases</i>	3	4	12	<i>Home office or redistribute tasks</i>
9	<i>MET API is removed</i>	<i>Closure, termination of work or other incidents</i>	1	5	5	<i>Find new weather data sources and adapt the application to new input</i>

Scale: Very low (1), Low (2), Medium (3), High(4) and Very high(5)

### 9.3 Risk Matrix



## 9.4 User Testing Guide

The application is a prototype which lacks many functionalities, the purpose of the evaluation is the user-interface and risk presentation.

The test application is made using dummy data.

### **MET ID**

Input “abcdefl”.

### **Data-preset**

The data-presets are just example representations of different risk displays. It is only available in the dummy-version.

### **CASE**

Please add new locations for these cities using a different data preset for each:

- Stavanger: Latitude: 58.96, Longitude: 5.726
- Bergen: Latitude: 60.39, Longitude: 5.32
- Haugesund: Use “Get my location”.

## 9.5 SUS results

1. I think that I would like to use FireGuard frequently.
2. I found FireGuard unnecessarily complex.
3. I thought FireGuard was easy to use.
4. I think that I would need the support of a technical person to be able to use FireGuard
5. I found the various functions in FireGuard were well integrated.
6. I thought there was too much inconsistency in FireGuard.
7. I would imagine that most people would learn to use FireGuard very quickly.
8. I found FireGuard very cumbersome to use.
9. I felt very confident using FireGuard.
10. I needed to learn a lot of things before I could get going with FireGuard.

Questions	1. I thi	2. I fo	3. I th	4. I thi	5. I fou	6. I th	7. I w	8. I fo	9. I fel	10. I ne	Test group	Score	Grade
<b>User Group 1</b>													
User 1	2	2	4	1	3	2	5	1	4	1	1	77,5	B
User 2	4	1	5	1	4	1	5	1	5	4	1	87,5	A+
User 3	3	1	5	1	5	1	5	1	5	1	1	95	A+
User 4	5	1	4	2	4	1	5	1	5	2	1	90	A+
User 5	4	2	3	2	4	2	4	2	3	2	1	70	C
<b>Average</b>	3,6	1,4	4,2	1,4	4	1,4	4,8	1,2	4,4	2		84	A
<b>User Group 2</b>													
User 1	1	2	4	1	5	2	4	1	5	1	2	80	A-
User 2	1	3	5	2	5	1	5	2	5	2	2	77,5	B+
User 3	2	2	4	2	4	3	4	3	4	2	2	65	C
User 4	3	4	3	2	4	3	3	1	4	2	2	62,5	D
User 5	3	3	2	3	3	3	3	3	2	3	2	45	F
User 6	4	2	3	2	3	5	4	3	2	1	2	57,5	D
User 7	4	2	4	1	3	1	5	2	4	2	2	80	A-
User 8	1	1	5	1	4	1	5	2	5	3	2	80	A-
User 9	2	2	5	1	3	3	4	1	4	2	2	72,5	C+
User 10	1	3	3	3	3	4	3	4	3	4	2	37,5	F
User 11	3	2	4	1	3	4	5	4	3	1	2	65	C
<b>Average</b>	2,3	2,4	3,8	1,7	3,6	2,7	4,1	2,4	3,7	2,1		65,7	C
<b>User Group 3</b>													
User 1	3	3	3	3	2	2	3	4	2	1	3	50	F
User 2	4	2	3	2	3	2	4	3	3	2	3	65	C
User 3	4	2	3	2	4	2	4	3	3	2	3	67,5	C
User 4	4	1	4	1	4	1	4	2	4	1	3	85	A+
User 5	5	2	3	1	4	2	5	4	5	3	3	75	B
User 6	3	3	3	4	3	2	2	3	2	4	3	42,5	F
<b>Average</b>	3,8	2,2	3,2	2,2	3,3	1,8	3,7	3,2	3,2	2,2		64,2	C-

## 9.6 Wireframe

