Implementation and Evaluation of a Fire Risk Indication Model

Sindre Stokkenes

Master's Thesis

Department of Computing
Western Norway University of Applied Sciences
Norway

June 3, 2019

Supervisor
Lars Michael Kristensen

Abstract

During the winter seasons there will often be a higher risk house fires due to the dry climate as the result of cold weather. The fire department, and other clients, would therefore have benefit of a fire risk indication system that can help alert them regarding higher risk of fire.

The thesis questions the use of a mathematical model to indicate the fire risk, as well as determining a suitable software architecture that can support both deployment and evaluation.

The mathematical model gave accurate fire risk readings, indicating that it can be used in real world situation to show the increase and decrease in fire risk with the use of multiple data sources. With regards to the software architecture it was feasible to find a software architecture that could both be used in the evaluation of the model, but also as a basis for the application when put into production.

Acknowledgements

First, and foremost, i want to thank my supervisor, Lars Michael Kristensen, for his continued support and guidance. The advice he has given me have been helped tremendously throughout stages of this thesis.

I would also like to thank Torgrim Log for helping understanding the aspect of the fire risk indication model that makes the foundation of the thesis.

Contents

Chapter 1

Introduction

There has been a very long tradition in Norway of building houses using wood, given the large amount available from forests. These houses can be extremely susceptible to fire under certain weather conditions. These weather conditions usually occur during the winter time when there are long periods of dry and cold weather. When the air gets colder and drier, the water concentration in the wood decreases, meaning there is a higher chance for the wood to catch fire. As identified by Log[1], there is a period during December - January when the weather is cold that have the highest fire frequency in Norway. In addition, this is the time of the year where people light candles, and often use their fire place. Combining the cold and dry weather conditions with the fact that people light fires in their houses, contributes to many of these fires occurring.

1.1 A Fire Risk Indication System

In this project, we address the question of whether there is a way to reduce the impact of fires by means of an early warning system that can for example warn the local fire brigade that there will be a high fire risk in the coming days. This would enable them to stay alert and be better prepared if a fire incident occurs. In this thesis, an application has been implemented aimed at helping in this regard. Our work relies on the research of Log[1] regarding the use of weather stations in combination with a predictive mathematical model[18] to find out how the relative humidity in wood changes when the weather gets colder. The application provides a service that can give fire risk indications that clients can use for their needs. The system uses a mathematical model[18] developed based on the research from Log[1], which is able to estimate the relative humidity in wooden constructions by using weather data from local weather stations. The model relies on a number of mathematical equations that enables us to model the indoor climate of a house which can then be used to find the changes of relative humidity in wood. This process can be divided in three stages.

The first stage involves finding and retrieving the relevant data that must be provided to the model. As of now, the data supplied to the model consists of the two weather elements of outdoor air temperature and outdoor relative humidity. These weather elements may come from historically recorded weather data, or they may come from predictive forecasts of the near future weather. The reason for using temperature and humidity, and not looking at wind for spread of fire, is that the model is still in its early stage of development and focuses on the fire risk of a single wooden structure. The weather elements are then used in equations for calculating certain outdoor weather conditions. In the second stage of the model, the weather conditions from the first stage is used to model the indoor climate based on the outdoor climate. The last stage of the process involves determining whether the wood inside the house, takes humidity from the surrounding area or releases humidity to the surrounding area, and then presents the fire risk by indicating how long it would take until the wood is in complete flash over in case it catches fire. This means that the fire risk is quantified as the time to flash over which is normally in the order of minutes.

When it comes to who the clients of the system are, the most relevant is the fire department. A close second is anyone that deals with protecting older wooden structures such as the old wooden churches and historical buildings in Norway protected by "Riksantikvaren". It is also possible for any ordinary person to use the system if requested.

1.2 Research Questions

This master project focuses on designing and implementing a cloud-based and distributed prototype application that can indicate the fire risk of wooden structures. This system will use a mathematical model[18] to calculate the fire risk by using weather data as input. From the weather data, the application shall produce data that can be used to create an indication as to whether there is increased fire risk or not, and thereby quantify the fire risk.

There are numerous ways of creating distributed applications. Part of the thesis is also to determine a suitable software architecture that can be used to evaluate the mathematical model, but can also serve as a basis for the applications when put into production.

This thesis is concerned with the following research questions.

- **R1:** Can the Fire Risk prediction model of Log[18] be used to give a useful indication of fire risk? This research question has the following sub-research questions:
  - **R1-A:** Measured weather data versus weather forecast data. How big is the difference in the computed fire indication when using measured weather data versus forecast data, and can the two be used in combination?
  - **R2-B:** With regard to measured weather data, how big is the difference between using weather data collected by the Norwegian Meteorological Institute versus weather data collected from private weather stations?
- **R2:** What is a suitable software architecture for collecting sensor data

and provide a fire risk indication service? This research question has the following sub-research questions:

- **R2-A:** Is there a software architecture that can support both deployment and validation?
- **R2-B:** Can we create a software architecture that is both efficient with regard to minimal storage of weather data and running time for fire risk indications computations?

R1 and R2 are two primary research questions explored in this thesis.

The first research question, R1, aims to investigate whether the predictive model of Log[18] can be used in real world situations and accurately predict the increase in fire risk of wooden house structures. That can be if the result from calculating fire risk gives indications at a reasonable level which can be attributed to the weather at the time. The fire risk is given in the amount of minutes it takes for wood to be in complete flash over (Tf0), on a scale from 1-14 minutes. There are two sub-questions, R1-A and R1-B, derived from this research question. These sub-question deals with the weather data supplied to the mathematical model and how it is combined. The data can come from different sources that focus on different ways of representing weather data. This can be from historically recorded weather data and forecast weather data. By using data from different sources, R1-A, aims to investigate whether it is possible to use weather forecasts in order to indicate the fire risk or not. The intention of R1-B is to investigate if it is possible to use consumer-grade weather stations, bought from local stores, to indicate the fire risk and how much it deviates from using professional data sources such as the Norwegian Meteorological Institute weather stations.

The second research question, R2, has the purpose of finding a software architecture suitable for providing weather data and elements to the mathematical model, as well as creating a service that clients can use for their needs. This again is split into sub- research questions R2-A and R2-B. R2-A has the main responsibility of answering whether the software architecture can be used for validation, with regard to this master thesis, and deployment of a production-level application. R2-B is there to answer whether the software architecture leads to an application that is both efficient with regard to running time when calculating fire risk indications, as well as storage efficiency of the collected weather data. It should not take to long when calculating fire risk, as that may have repercussions for the fire risk indication. Storing weather data may potentially accumulate to large amounts of data as time goes by. It is therefore important to store as little weather data as possible but enough to give accurate indications.

1.3 Research Method

We aim to solve the two research questions by creating a prototype application that will implement a mathematical model that can be used for calculating the fire risk.

In order to answer the questions from R1, the prototype application will collect weather data in the winter period 2018/2019 in order to have an overview of the fire risk during this period. Part of this collection process is to place a few Netatmo stations as well so that it is possible to compare the fire risk calculated from the Netatmo stations to the fire risk calculated from the Norwegian Meteorological institute sources (weather station). In addition to collecting weather data in the winter period 2018-2019, there are several historical fires that are taken into consideration to see if the model indicate high or low fire risk at the time of fire.

We investigate R2 by looking at the different solutions that can be used to create a distributed application. It is expected to have a prototype application that implements the mathematical model and can be used for deployment and validation purposes. The application should be able to supply the mathematical model with weather data, from different sources, which is needed by the model to estimate the indoor climate in order to indicate the fire risk of older wooden structures.

Based on the outputs produced by the mathematical model for the different scenarios, it is possible to validate and conclude whether the model is sufficiently accurate to work in real world situations. It is also be possible to look at the differences in the fire risk indication by using different types of weather data, such as historical weather data and forecast data as well as a combination of the two.

The system contains an application whose main functionality is to fetch weather data from various sources and run it through the mathematical model. On top of the application there is a web service that clients can use for their own needs with regard to obtaining fire risk indications.

1.4 Software Technologies

Different technologies and architectural styles were used to create the Fire Risk application (FR). The FRA application is a prototype that will end up having some incomplete features. An important factor was therefore to create the application in such a way that it would be easy to expand the different features without having to significantly rewrite the code base. To achieve this, the architectural style of micro-services was used.

By designing the application as several micro-services[9], the application will be split into several components that each have one specific functionality to take care of. This way, there will not be too much cohesion between the components, and it will be easier to add or change the functionality of the component. Each micro-service must then communicate with the other micro-services if needed. This was achieved by use of REST[17]. Each of the components can be seen as a single small application that runs on its own process. This means that if one component fails or stops working, the other components still work and can provide functionality.

The FR application also uses external web services. These external web services uses the RESTful architectural style, that relies on the standard HTTP methods, such as GET, POST, PUT and DELETE to access the services. The RESTful architectural style is a way of implementing web services that offers platform independence. Since it uses HTTP methods to communicate, it does not matter how the web services is implemented since the data transmitted by the service will be in the form of XML or Json data. As a result of this, any RESTful web services are language independent and it does not matter which programming language was used to implement it.

For evaluation purposes, a small web application was implemented as a means of visualizing several scenarios related to the calculation of the fire risk indication. This web application was implemented using JSF(Java Server Faces)[2] which is a server-side component framework for building web applications with Java.

The Spark Java framework[19] was used to create the web service on top of the application that handles communiation with the clients. This is a framework that reduces the complexity of setting up RESTful web services.

One of the components that makes up the application uses Enterprise Java Beans[2] (EJB) for creating schedulers in order to have continuous harvesting of weather data. The EJBs also provide transaction control when dealing with data storage for SQL databases.

1.5 External Services and Data Sources

Part of the FR application has the task of supplying weather data to the mathematical model for predicting the increase of fire risk. The FR application uses three services to supply weather data from several sources. Two of the sources provide historical weather data while the third one provides forecast weather data to predict the fire risk in the near future.

1.5.1 Frost

Frost is a service[3] that supplies historical weather data recorded by the Norwegian Meteorological Institute (MET). MET is a government agency, under the Climate and Environment Department, and has a number of weather stations placed around Norway that records and measures weather data. This weather data can be freely used by the public. Users of the service must provide the locations of where it shall retrieve weather data from. This can be done by providing the identity of the source (station), or by giving the longitude and latitude of a position and the service will then find the nearest station. Then the user can retrieve weather elements of their choice.

1.5.2 Netatmo

The Netatmo service[6] deals with the same type of weather data as the Frost service, but it relies on consumer grade weather stations that can be bought in stores. The consumers then publish their weather data into a cloud-based server. Through this cloud-base server, it is possible to retrieve the measured weather data, that can then be used in the FR application. The Netatmo stations come with a sensor fore measuring temperature and humidity. Wind and rain gauge sensors must be bought separately if required.

1.5.3 MET API

The MET API[4] deals with current and predictive analysis of weather. It offers resources that estimate how the weather will be in the near future, as well as current weather data such as lowest and highest temperatures over a certain period. The service is able to return the weather data for predictions of the weather for a nine day period into the future. The first three and a half days are provided as hourly measures. The next five and a half days are provided at six hour intervals.

1.5.4 Fire Risk Indication Service

On top of the implemented application, there is as fire risk indication service that users can interact with in order to communicate with the application and its features. The fire risk service includes several features such as, the ability to add locations, with longitude and latitude coordinates, where the application should do continuous fire risk indications. Another feature is the ability to get the fire risk for a specific time and place that is provided by the user. This feature has four options as to how it can be used.

Option 1: See how a fire risk indication would have been between two dates in the past.

Option 2: Look at the fire risk from a time in the past to the present time.

Option 3: Possibility of looking at the fire risk indication from the present time and nine days into the future, by using weather forecast.

Option 4: Use a combination of the two previous options, by using a fire risk indication from a date in the past to the present time and then adding the fire risk indication based on the weather forecast.

This Fire Risk Indication Service also supports the validation of the mathematical model by setting up different test cases that reads data from a database and visualizes in the web application. These test cases includes looking at the difference of using the Meteorological Institutes weather stations, compared to Netatmo stations. It also considers the difference between a forecast fire risk indication, and the corresponding historical fire risk indication. As the model requires a few days of data to calibrate itself, the latter test case explores how many days in the past must be included to get more accurate indications.

1.6 Weather Station Calibration

The consumer grade weather stations from Netatmo in use by the FR application had to be calibrated in order to be able to use the measures of humidity in a reliable way. The process of calibrating the humidity sensors involved creating a controlled environment inside a plastic box. Previous experiments have calculated the humidity that different salts releases when dissolved in water. These salts were dissolved in small paper plates inside the plastic box to create an environment with the humidity that the salts release. This was then left for several hours to create an equilibrium inside the box, and the value of the measure was then recorded in order to find how much it deviated from the known humidity of the salt. This was then used to create a correction curve that could be used to correct the measured value in order for it to be more accurate.

1.7 Summary of Results

The result of comparing the calculated fire risk of measured weather data versus forecast weather data indicates that the difference is minimal and it would be possible to use the forecast weather data as a basis for fire risk calculations.

Concerning the use of Netatmo stations when calculating the fire risk it would also indicated that the difference are within reasonable margin even when using non calibrated stations.

The use of micro-services as an architectural style, combined with REST as a communication form it made it possible to split the functionality two ways with regard to the evaluation of the mathematical as well as the resources that can be used by external clients.

In summary, the result regarding the run-time efficiency of the application indicates that the bottleneck lies with the external services that are in use. The amount of storage also shows that when working with few locations, the amount of storage needed is minimal. There is the possibility that when there a many locations that it can start to become an issue.

1.8 Thesis Organization

This thesis is organized into the following chapters:

Chapter 2: Software Architecture for Distributed Application gives background information about several technologies that may be used to implement distributed system. It discusses what the technology is, and presents the main benefits of using that technology.

Chapter 3: Predictive Modelling of Fire Danger outlines the elements that make up the mathematical model. It goes through the motivation for creating the model and for what purpose it will be used. The first section of the chapter introduces the elements of the model as well as going into detail about the most important equations that make up the model at the different stages. Finally it goes into detail about where the required data comes from and the calibration process of the Netatmo stations.

Chapter 4: System Requirements and Design underlines an analysis of the requirements for the prototype application. From the analysis, it discusses what a possible software architecture for the prototype application may be, as well as the different components and parts that is needed to make up the application.

Chapter 5: Implementation and Deployment goes into detail about the external services that are used in the application and details how to use them. It discusses the use cases and functionality of the application itself, and describes the components that make up the application by providing class diagrams. The Program flow is also presented via time sequence diagrams that shows how the components interact with each other over time.

Chapter 6: Evaluation presents the evaluation of the mathematical model by providing results that show the differences between calculating the fire risk using different sources of weather, as well as combining the sources for creating fire risk indication. It will also demonstrate how the model would have predicted the fire risk for historical fires that occured in the past. Finally it gives an overview of the efficiency of the FR application regarding storage and run-time.

Chapter 7: Conclusion and Future Work draws the conclusion from the result presented in the evaluation by setting it up against the research questions presented in the introduction. It also discusses what can be improved on the application and presents what was completed and which parts requires further work.

The source code of the application can be found in the following GitHub repository `https://github.com/sinstok/master-project`

Chapter 2

Software Architecture for Distributed Applications

There are certain requirements for the FR application that needs to be fulfilled. The first and foremost is to have a prototype application that can validate the use of the proposed mathematical model for calculating fire risk. The approach to do this is to focus on a few locations around Norway, and observe how the model performs on these locations. The prototype should be made in a way such that it can easily be extended with additional locations and deployed in mass scale without having to refactor large portions of the application.

The prototype application will consist of several parts, and the proposed system architecture is to focus on dividing the parts into micro-services and structuring it around a message-driven architecture. The FR application will also access external services that are provided via REST APIs in order to retrieve weather data used in certain parts of the FR application.

The vision is that the FR application is to be hosted on a virtual machine in the cloud that will run continuously, gathering data and calculating the fire risk. Users will then be able to view the fire risk of the selected location, through a web service, over the observation period. There will also be a few predefined locations of previous known fires. These will serve as examples of what the model concludes, based on the weather data from the previous days and the day of the fire in these locations.

2.1 Representational States Transfer (REST)

REST[17] is a main approach that allow software applications to offer web services over the Internet. It is an architectural style of creating web services that relies on the use of standard HTTP methods, most notably, GET, POST, PUT and DELETE to specify operations on resources. REST is simple and easy to use, and is very efficient given the scale of the Internet today. The HTTP methods are used as follows:

- The GET method is used when requesting data from a server hosting a RESTful service.

- The POST method is used for creating new resources.
- The PUT method is used for updating existing resources.
- The DELETE method is used for deleting specified resources.

REST involves using URIs (Uniform Resource Identifier) to identify the resources on which operations are to be performed. One can think of this as an URL which can be used in a web browser, to identify the resource that one wants to access and manipulate. As mentioned earlier, REST uses standard HTTP methods for determining what the web service will do when someone makes a request to the web service. This means that if a person wants to get some data from a web service, the HTTP method GET would be used. If for example a new resource will be created, the HTTP method POST would be used. From this, a normal REST request to get data could look like this:

```
https://api.met.no/forecast/longitude=60.5934&latitude=5.3546
```

which relies on MET having a web service where one can look up a forecast by providing longitude and latitude as query parameters. Using the HTTP method GET, and longitude and latitude coordinates, the service will then fetch the forecast from this location and return it back to the requester. Since REST supports different formats for the response data, the requester can specify in the request whether the service should return for instance Json or XML representation of the resource. If the web service has been implemented in such a way that only one of them is possible, it will give a response where the resources is represented with the default format.

Web services are based on platform independent technology such as HTTP, XML, and Json. These technologies are supported by all major and smaller platforms, and gives everyone the ability use the service. At the same time, a web service should not be language specific, meaning that a web service written in C++ should work with a web service client written in Java.

2.2 Micro-services

Micro-services is an architectural approach to building applications. It relies on the idea that each component of the application should be its own autonomous service[9], or in some cases many small web applications. The communication between these services can be over the network using standard HTTP, it can be through a RESTful API, or it can use Remote Procedural Call (RPC).

When working with traditional monolithic applications, the code base will grow and grow as new features are added. As the application grows, it can be quite hard to make changes as they can have an affect on the whole system. There have been attempts at fixing this issue for quite some time, such as Service Oriented Architecture (SOA)[9]. Micro-services is a modern inspiration of SOA[20] and takes ideas from many different communities in an effort to find a better way of partitioning a very large application into smaller domains, each with their own functionality[8].

Some of the key benefits of using micro-services to build applications is that, as stated above, each functionality have its own domain. When working with a monolithic application that runs as a single process, the fault can cascade through the application and everything will stop working. With micro-services, since each functionality has its own domain and process to run on, a failure will not cascade through the rest of the application and the other parts of the application can remain operations. Another benefit of using a micro-service architecture is that the different services do not have to be implemented with the same technology. One can be implemented in Java, another in C++ and another with C# each of which can use its own database system of choice[8].

Building applications with micro-services in mind also makes it easier to test and maintain. Since each micro-service runs its own domain, it can easily extended without affecting other domains in the application. Deploying the micro-services becomes easier, as it will not be necessary to re-deploy the whole application when there is a one-line change in one of the micro-services. There is also the case for re-usability since the functionality is split into different services and can be used with different applications.

There are several tools that can be used for creating a micro-service architecture. This involves any frame work that can create web services. An example is the JAX-RS API[2] that Java provides for making REST APIs.

2.3 Message-Driven Communication

Message-driven architecture is akin to micro services, but it uses messages to connect different components or applications, with so-called MOM (Message Oriented Middle-ware)[12]. There are different ways to achieve this goal, e.g. using a publish-subscribe model, or a message queuing system[11]. Given the scale of distributed applications, a synchronous communication model makes for a rigid and static application[10], where the application has to wait before it can communicate with a different application or component. An asynchronous communication scheme is therefore better suited for the job as it does not rely on timing, and can reach more applications and components faster[10].

In the case of a publish-subscribe model, it uses a broker that handles communication between the publisher, which is the sender of the messages, and the subscriber, which is the one receiving the messages. The broker has several topics where the publishers can publish their messages. The subscribers are then able to subscribe to their topic of choice and whenever new messages are published to the broker, it sends the message to subscribers of that particular topic.

This type of communication removes the synchronous communication from the publisher[10], as it only has to publish to the broker, which in turn can make it available for all the subscribers that want the message. With the publish-subscribe model, each subscriber can subscribe to as many topics as they want.

A message queuing model is often intertwined with the publish-subscribe model, as it solves the same issue as publish-subscribe but in a different way. It uses message queues, often implemented as FIFO (First in first out), instead of having a broker between the publisher and subscriber. The message is addressed to a specific queue, and then the consumer can fetch the message from the queue that was established to hold the message[10]. With the use of messages queues, it can guarantee acknowledgement that the message was in fact received by the consumer[11].

2.4 Communication Model

We have presented two possible communication models in this chapter, REST and message-driven system. These are two different styles of communication that has their own benefits and disadvantages. A choice had to be made in regard to which communication model should be used for the FR application. As stated, a message-driven communication model offers asynchronous communication that does not block operations by waiting for actions to finish. Using a REST communication is the opposite as the application has to wait for operations to finish before it can move on.

Since this is a simple prototype application that interacts with very few clients and is expected to handle small amounts of data transfer, the benefit of having a asynchronous communication model is limited. It was therefore chosen to use a synchronous communication model through REST as that would be less problematic to set up and because most of the external services that is to be used by the application rely on REST.

2.5 External Data services

The weather data for the FR application will be obtained from several external services:

Frost is a web service[3] hosted by the Norwegian Meteorological Institute. The Frost web service offers free, historically recorded, weather data. The Norwegian Meteorological Institute has a large number of weather stations placed at different locations around Norway that periodically measures and store weather data.

The Met API[4] provides weather forecasts for the next nine and a half days. This includes hourly weather predictions for the first three and a half days, and then six hour weather predictions for the next six days.

Netatmo provides data collected from consumer grade weather stations that are easily accessible. They also offer an API which lets the users retrieve weather information from their own privately owned Netatmo station.

We will provide more technical detail, on these services later in the thesis.

2.6 Data Storage

The FR application interacts with three external services that provide weather data, including Frost, Netatmo API and the Met API. Part of the evaluation process is to compare the fire risk computed using weather data from different sources. In order to have consistent data throughout the evaluation period, the weather data from all external services is stored in a database. This allows the model to use a consistent data-set if testing occurs at a later date, when the model has been updated.

As data is collected and stored from the different services in order to have the data available at a later stage, a choice was made to use the original format as it was retrieved from the external services. This was done in order to achieve a form of consistency in the data. Since most of the services sends data in a Json representation, a traditional relational database did not suit this goal as the data would have to be converted to a format suitable for relational databases. It was therefor decided to investigate whether a noSQL database would be better suited for the job. There exits several types of noSQL database including: Key-Value Store Databases, Column-Oriented Databases, Document Store Databases, Graph Databases and Object Oriented Databases.

Key-Value Store Databases[14][15] relies on assigning the stored data to a key, which can then be used when retrieving the data. The data is stored similarly to that of an object in an object oriented programming language. Column-Oriented Databases[14][15] represents the data as tables, where each table contains several attributes. Document Store Databases[14][15] store the data in the format of Json, XML or BSON (Binary Json) documents. These documents includes the same attributes and their values as found in the corresponding formats. Graph Databases[15] represents the stored data as a graph. The objects stored in the database correspond to that of a node in a graph and the and an edge will be assigned as the relation between the nodes. There are also properties related to the different nodes. In an Object-Oriented[15] Database the data is represented as objects, similarly to what is found in object oriented programming languages.

After exploring different NoSQL database types[14], it was decided that a document database[14] seemed most appropriate for storing the Json data[14].

After establishing which database management system to use, a suitable document database system had to be found. There are several document databases available, such as MongoDB, Amazon DynamoDB and CouceDB[14]. As the FR application runs on a cloud platform, this was taken into consideration when choosing the right document database. In the end the choice was between MongoDB and Amazoon DynamoDB, as they both offers cloud deployment. MongoDB ended up as the document database of choice as they offer a free database[13] for use below 512 mb of storage in the cloud. The estimated amount of storage needed by FR application did not exceed the amount of storage they offered for free.

Chapter 3

Predictive Modelling of Fire Risk

This chapter introduces the mathematical model[18] that underlines the fire risk prediction application. The presentation is based on Log[1], supplemented with the high level model[18] for fire risk, developed as part of this thesis.

3.1 Fire Risk Prediction

In Norway there are a lot of houses made of wood and similar materials. This can lead to a high fire risk in certain periods of the year when the weather is dry. Log[1] investigated the use of mathematical modelling[1] to accurately predict the relative humidity indoors.

The results showed that the air inside the older buildings are drier than that of the newer houses by around $5 - 10\%$. But when the weather gets colder and drier, a decay period of around 6 - 7 days were identified for the relative humidity in the wood to stabilize in the older houses. At the same time, Log[1] found that for some of the newer houses this decay period was around 3 - 4 days.

Based on the research of Log[1] a mathematical model[18] was devised, aimed at predicting the indoor climate, in wooden houses, derived from the outdoor climate. Knowing the indoor climate makes it possible to determine the concentration of water in the wood, which in turn makes it possible to ascertain how long it would take before reaching complete flash over in case of fire. Figure 3.1 illustrates the equilibrium moisture content of wood which is a period where the wood is neither loosing or gaining moisture. This happens when the wood is exposed to the same humidity over longer periods of time and it reaches equilibrium. Whenever the air gets dry, the humidity of the surrounding area decreases. During this process moisture from the wood is released. When the process goes the other way the wood attracts moisture from the air, thus reducing the fire risk. Being able to predict the indoor climate is of huge benefit as it is not always possible to have tools that can measure the necessary requirements for indicating fire risk. Currently there are restrictions as to which houses the model are able to predict the indoor climate for, and it can only be used for indicating fire risk for single structure buildings.

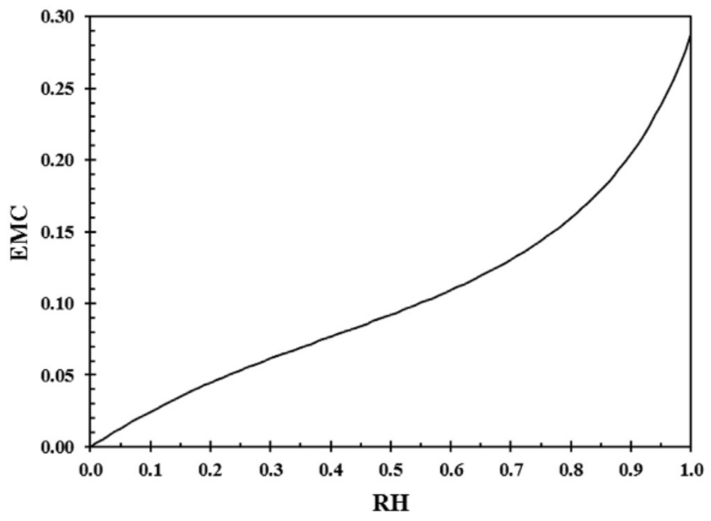


Figure 3.1: Wood Equilibrium Moisture Content (EMC)

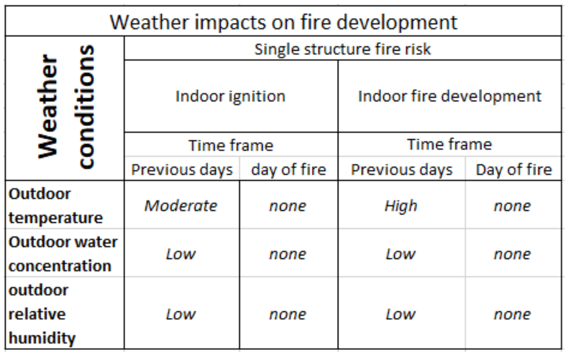
In the process of modelling the indoor climate, certain outdoor weather elements are needed. Currently the weather elements needed are the outdoor air temperature, outdoor relative humidity and outdoor water concentration in the air. The outdoor air temperature and humidity are obtainable by weather sensors, whereas the outdoor water concentration can be acquired from a mathematical equation that takes the temperature and humidity as parameters. In the future the model may be revised to include other parameters that can help produce a better fire risk prediction, with regard to fire conflagration and other house types. The most notable would be the wind speed and the wind direction.

3.2 High-level Prediction Model

With this information in mind, weather elements impact the fire danger at different stages. Some of them are more crucial in the days before a fire happens, and others are more crucial at the day of a fire. Figure 3.2 illustrates this process by showing how crucial and at what stage the weather elements impact the fire.

When it comes to fire risk of a single building where the fire ignites indoors, we can see that outdoor temperature has a greater impact than that of the ambient water concentration and relative humidity in the previous days. On the day of fire, these weather elements do not increase the risk of fire ignition inside the house. When it comes to the indoor fire development, we see that the outdoor temperature has a much greater role than when it comes to the indoor ignition, and there are no weather elements that increase the risk on the day of the fire.

From the three weather elements that are mentioned above, makes it is possible to estimate the indoor climate, which can be used to find the concentration



Weather impacts on fire development				
Weather conditions	Single structure fire risk			
	Indoor ignition		Indoor fire development	
	Time frame		Time frame	
	Previous days	day of fire	Previous days	Day of fire
Outdoor temperature	*Moderate*	*none*	*High*	*none*
Outdoor water concentration	*Low*	*none*	*Low*	*none*
outdoor relative humidity	*Low*	*none*	*Low*	*none*

Figure 3.2: Weather impacts on fire development

of water in the wood. This concentration is then used by the model to create a quantifiable fire risk, known as the time to flash over, that can be used to indicate the fire risk.

3.3 Outdoor and Indoor Climate

As mentioned above, outdoor air temperature, outdoor relative humidity and outdoor water concentration of the air is required in order to model the indoor climate.

3.3.1 Outdoor Climate

One of the necessary weather elements from the outdoors is the outdoor water concentration, which unfortunately can not be measured directly, however it can be calculated. In order to estimate the water concentration it is necessary to know the water saturation vapour pressure. This can be obtained with the following formulae:

$$P_{sat} = 610.78 * e^{\frac{17.2694*T_c}{T_c*237.3}} \tag{3.1}$$

The only parameter needed to calculate the water saturation vapour pressure is the outdoor air temperature. When P_{sat} has been calculated, the outdoor water concentration is obtainable with the following formulae:

$$C_{wa} = RH_{out} * (\frac{P_{sat}(T_c) * M_w}{R * (T_c + K})) \tag{3.2}$$

where RH_{out} is the outdoor relative humidity, P_{sat} which is the same as equation 3.1 and takes T_c (outdoor temperature) as a parameter. $M_w(0.01801528)(kg/mol)$ is the molecular mass of water, $R(8.314J/kmol)$ is the molar gas constant, and K is the absolute temperature 273.15. Now that all the outdoor weather elements are available, it is possible to begin modeling the indoor climate.

3.3.2 Indoor Climate

There are a few aspects that needs to be addressed when calculating indoor climate. One is that at any given time there may be plants, people, animals that releases humidity in a room, which provides a moisture supply. There has been a lot of research on the average moisture supply from external sources in houses[16], but we can not accurately come up with an average for a house. Log[1] found that the moisture supply for the test house had a mean of $1.29 \pm 0.75 g/m^3$, and an average recorded moisture supply that varied from 0.3 to $2.64 g/m^3$. In Log[18] an assumption is made that around $1kg$ of moisture is released inside a house daily.

Another aspect to consider is the air change rate(ACR) of the house, which is the rate at which inside air is replaced with outdoor air. In newer houses the ACR usually follows the countries rules and regulations which puts the ACH(air change rate per hour) at 0.5[1] for normal houses. The ACH may vary from the construction of the house, as there are some construction methods that have lower ACH than average. Older houses do not follow the same rules and regulations, and may have not have taken the ACR into consideration when constructed. As a result of this the indoor climate will be much dryer in a newer house. Older houses also take longer time to react to changes in the weather than newer houses. An estimate for what the ACH of older houses may be, is half of what is considered balanced air today[18]. The ACH of older houses may also vary depending on the temperature, and can then be calculated by using the following equation:

$$ACH = \gamma * \sqrt{(1/T_a - 1/T_{in})/T_a} \tag{3.3}$$

This is a simplified formula that uses a known ventilation rate γ instead of calculating the ventilation rate itself. As the model at this stage focuses on older buildings, this rate is set to $300h^{-1}$ to reflect an ACH of 0.25. The ventilation rate can be adjusted depending on the size of the house, and if there is prior knowledge of the ACH. Newer houses that use forced ventilation, usually keeps the same ACH at all times and can be used in the model if required. T_a is the ambient temperature (outdoor temperature), and T_{in} is the indoor temperature. It is also necessary to look at the changes in the concentration of air, air dilution, when calculating the indoor climate. This can be done with knowledge of the ACH and the following equation.

$$\beta = 1 - e^{-ACH*\Delta t/3600s} \tag{3.4}$$

ACH is the same as equation 3.3, and Δt is the time interval in which the calculation takes place. In our case, this will be every 720 seconds (12 minutes). It is set to this interval in order to model the changes between every hour.

Now that the ACH and moisture supply have been taken into consideration, the concentration of water in the air inside a house can be calculated by doing one of the following. If this is the first calculation, then the following equation will be used:

$$C_{inside} = RH_{inside} * C_{sat,in} \tag{3.5}$$

where RH_{inside} is a base relative humidity set to 40% as a starting point, and $C_{sat,in}$ is the same as equation 3.2 without RH_{out}, and T_c is 22 degrees Celsius. If there are prior calculations the following formula will be used:

$$C_{in} = ((1 - \beta) * C_{in-1} + \beta * C_{wa} * (\frac{T_{out}}{T_{in}}))) + m_{wall,loss}/v + ms * \frac{\Delta t}{v} \tag{3.6}$$

where β is equation 3.4, C_{in-1} is the previous calculation of C_{inside}, C_{wa} is equation 3.2 where T_c is the outdoor temperature, T_{out} and T_{in} is the absolute outdoor and indoor temperature, m_{wall} which is the sum loss of water concentration in the wall, *volume* is the volume of the room, ms is the moisture supply which is $\frac{1}{24*3600}$ and Δt is the calculation interval at 720 seconds.

With the indoors concentration of water calculated, it is possible to find the relative humidity using the following equation:

$$RH_{inside} = \frac{C_{inside}}{C_{sat,in}} \tag{3.7}$$

where C_{inside} is found using equation 3.5 if it is the first calculation, or equation 3.6 if there are prior calculations. $C_{sat,in}$ is equation 3.2 without RH and T_c is set to $22 \deg c$.

3.4 Humidity transportation in wood

The final stage of the calculations is to find the concentration of water in the wood and from there find the fuel moisture content (FMC). The FMC can be used to calculate the time to flash over which is used as a measure of the fire risk .

The process of finding the concentration in the wood is based on the assumption that the wood panels have a thickness of 0.012 m[1][18]. These panels are then divided into 10 layers, each with a thickness $\Delta x = \frac{0.012}{10}$, in order to model the moisture transportation through the wood. These layers are organized as follows. The first layer, the middle layers (2 - 9), and the innermost layer (10). The process of finding the concentration through the layers requires three separate equations. For the first layer, the concentration is obtained by using the following equation:

$$C_{1(t+1)} = C_{1(t)} + \frac{\Delta t}{A * \Delta x}\{\frac{D_{w,a}}{\delta}(RH_{(t)} - RH_{1(t)})C_{sat,22 \deg c} + \frac{D_{w,s}}{\Delta x}(C_{2(t)} - C_{1(t)})\} \tag{3.8}$$

where $C_{1(t)}$ is the concentration of the previous calculated concentration of the first layer. A is the surface area that is drying, Δx is the thickness of the layers, $D_{w,a}$ is the diffusion coefficient of water in the air, $C_{sat,22 \deg c}$ is the water concentration at $22 \deg c$, $D_{w,s}$ is the diffusion coefficient of water in solid material, and $C_{2(t)}$ is the previous calculated concentration in layer 2.

For the layers n = 2 through n = N - 1 the following equation was used:

$$C_{1(t+1)} = C_{1(t)} + Fo * (C_{n-1(t)} - 2 * C(n(t)) + C_{n+1(t)}) \tag{3.9}$$

where Fo is given with the following equation:

$$Fo = \frac{D_{w,s} * \Delta t}{\Delta x^2} \tag{3.10}$$

and $D_{w,s}$ is the diffusion constant of water in wood, which is set to $3.0E - 10m^2s^{-1}$.

For the layer N (innermost layer), it was assumed there is only moisture exchange with layer N - 1[18]:

$$C_{N(t+1)} = C_{N(t)} + Fo * (C_{n-1(t)} - C_{n(t)}) \tag{3.11}$$

Now that the water concentration in the different layers have been calculated, the next step will be to find the fuel moisture content that is used in the final equation for finding the time to flash over. It follows the same procedure as finding the water concentration, and it is divided into the same layers.

$$FMC = 100 * (\frac{C_{surface}}{RH_0}) \tag{3.12}$$

$$FMC = 100 * \frac{\frac{\sum_{n=1}^{9} C_n}{N-1}}{RH_0} \tag{3.13}$$

$$FMC = 100 * \frac{C_N}{RH_0} \tag{3.14}$$

Where $C_{surface}$ is the water concentration at the first layer of the wood, C_N is the water concentration of the inner most layer, and $\frac{\sum_{n=1}^{9} C_n}{N-1}$ is the average water concentration of the layers between the surface and the inner layer. RH_0 is the base humidity inside the house which is set to $500\frac{kg}{m^3}$. Now that the fuel moisture content is obtained, the time to flash over can be found by the following formula.

$$t_{FO} = 2 * e^{0.16*FMC} \tag{3.15}$$

Where FMC is the fuel moisture content which can be obtained by using equation 3.12, equation 3.13, or equation 3.14. By using the weather elements mentioned previously and running all the calculation, the following graph can be constructed which informs of the time to flash over from the different layers over some time. The y-axis shows the time to flash over T_f0 of wood and the x-axis is the time interval the Tf0 was calculated within.

3.5 Weather Stations

There are several means of supplying weather data to the model. Part of Log's[1] research was to investigate whether consumer grade weather stations were comparable to stations operated by the Norwegian Meteorological Institute. Log

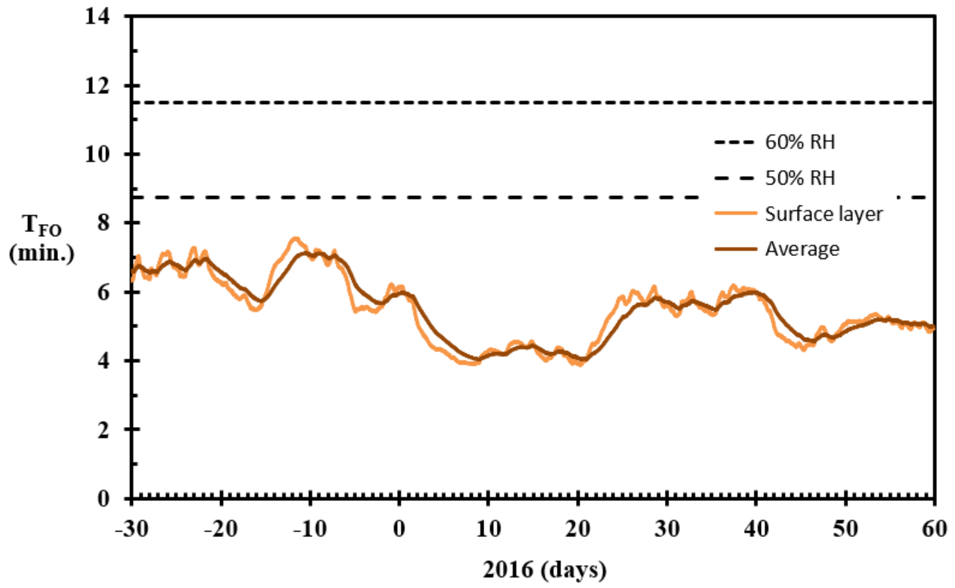


Figure 3.3: Time to flash over[1]

used the weather station from Netatmo[1]. When looking at the specification of the weather station, it claimed that it could read temperature at ± 0.5°C and the relative humidity at $\pm 3\% RH$.

In order to verify the information from the producer, an independent experiment was conducted where the stations were tested against known humidity from different salts dissolved in water. If the stations deviated from the correct humidity from the salts, a calibration curve could be calculated and used to adjust the measurements from the stations. From this experiment, Log[1] found that a few stations gave accurate readings within ± 0.5°C and $\pm 3\% RH$, but some of them had readings well above $10\% RH$ then the humidity from the salts.

Some of the stations used in this project are the same that Log[1] used in his research, but some of them are new and have not been calibrated. To be thorough, the stations from Logs research and the new ones used in this project were calibrated. This way we could be sure that the correct calibration curve is used when adjusting the measurements of Logs stations and the new ones.

The calibration process was the same as Logs[1] research, where a plastic container with a lid was used to create a controlled atmosphere for the calibration. Figure 3.4 show the set up for calibrating the weather station. Most of the equipment used, was the same as Log used in his research. In order to be sure the sensors could handle different temperature, the sensors where first calibrated at room temperature and then at outdoor temperature later. Following the process of Log[1], four sensors were placed inside the container, shown below, at a time. The container was left to achieve equilibrium conditions and the temperature and humidity measurements of the weather stations was noted.

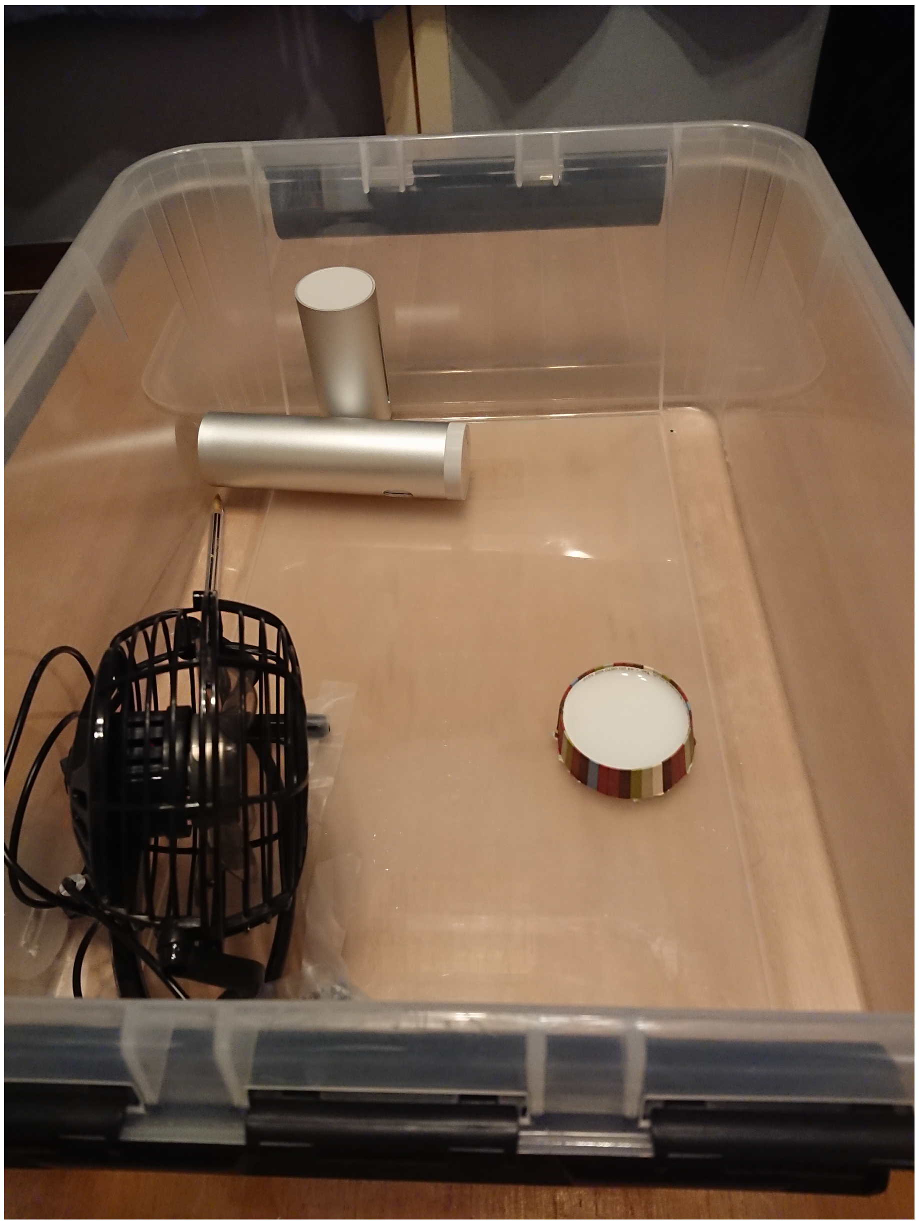


Figure 3.4: Plastic container used to create the controlled environment

It was advised to use salts that gave humidity below 50% as well as salts that gave humidity above 50%. The salts used was $LiCl$, $MgCl_(2)$ and $NaCl$. These would give humidity equilibrium at 11.31 ± 0.31, 33.07 ± 0.18 and 75.47 ± 0.14 % RH.

After the calibration process was finished, we got the same results as Log, where some of the sensors gave accurate readings at certain humidity levels. But at the same time could give readings with an error of 10%. From the measurements of the weather station a linear correlation curve was created that could be used to adjust the measurements. Using the correction curve from the linear correlation, the stations gave measurements within the stated deviation from the producer at $\pm 3\% RH$.

Chapter 4

System Requirements and Design

This chapter analyses the requirements for the prototype application and presents the design of the components that constitute the FR application developed with the aim of validating the Fire Risk Indication model (FRI).

4.1 Analysis

There are several requirements that need to be fulfilled in order to have a working application that can be used in regard to evaluating the FRI model. As mentioned earlier, the FRI model uses weather data to predict the indoor climate based on the outdoor climate. The weather data can come from many sources, such as private weather sensors or more professional sensors operated by larger institutes. Because of this, the application has to be able to support weather data from several types of data sources. In addition, this can be historical recorded weather data as well as forecast weather data. Historical weather data is data that has been recorded in the past based on measurements whereas forecast weather data is a prediction as to how the weather will be some days in the future.

The data may also come from different external services that do not necessarily use the same protocol or data format. This has to be taken into consideration when designing the application. For the sake of consistency of data when testing, it has to be stored such that it can be assured that the data has not been modified if validation needs to be performed again at a later date. Another reason for storing the data is that if for any reason there are changes to the FRI model later, it can be assured that it is applied and re-evaluated on the same data-set that was used previously.

There are a lot of different weather elements that are not considered as the FRI model focuses on single building fires in older wooden buildings. Eventually when the application is expanded, these weather elements may be needed when comparing to older fire risk computations. Because of this, even if the data is not needed for now it will still be stored as a precaution. As of now, the application only has a need for outdoor temperature and outdoor relative humidity. This may later be expanded such that wind speed and also wind direction is required in order to model the risk of the fire spreading.

The FRI model can be used with additional parameters to get better fire risk for different situations such as, larger buildings, modern buildings and spread of fire between buildings. Because of this, the application must be implemented in such a way that it can easily be expanded or modified. This goes together with the requirement that weather elements that are not necessarily used in this application may be used later if and when the application is expanded with more sophisticated FRI models.

Apart from different types of weather data and sensor sources, users have to be able to interact with the Fire Risk application (FR) to specify what operations they want to use. It can be that the user wants to know the fire risk for a specific location at a specific time, or want to know the fire risk from now on and some days in the future. Therefore there needs to be a front-end that the users can interact with where they can view results and specify fire risk locations.

Since the weather data comes from different sources it should be possible to combine it when calculating the fire risk. This means that it should be possible to combine historically weather data with forecast weather data to create a fire risk indication. Note that it should also be possible to only use one of them for fire risk indication. This means that it should be possible to only use historically data to look at the accurate measurements to create a fire risk indication, and also only use forecast weather data to create an indication.

Users should also be able to specify a location for where the FR application should do continuous fire risk calculation. The idea of this requirement is that a user may want to monitor a location continuously and observe the fire risk overtime and not have to type date and time every time the user wants to look at it. The FR application should then use a timer that regularly fetches the weather data for that location and runs it through the FRI model to create the fire risk indication.

Both the front-end and the back-end should be deployed to a cloud platform for easy distribution and availability to the end-users. By this requirement, the FR application should be accessible from anywhere and whoever want to use it should be able to.

Weather data can quickly accumulate to large amounts of data. It is therefore important to limit the amount of data as much as possible by avoiding to store any non-essential data values. Another aspect to have in mind is to have as little waiting time as possible when calculating the fire risk for a given location.

4.2 Requirements

Below we list the individual requirements derived from the above analysis. In later sections, we will then refer back to these requirements, when presenting the implementation of the FR application.

Requirement R1: Weather data sources.

- This requirement states that the FR application should be able to fetch weather data from different sources provided via web services.

Requirement R2: Common data format.

- This requirement state that the data from the external services that provides weather data should be converted to a common standard format that can be used for the FRI model. Also, if there are any changes to the FRI model at a later date the data should be stored in order to be able to compare it with older versions of the FRI model.

Requirement R3: FRI model refinement.

- The FRI model that the FR application uses, focuses on single structure fires for older wooden buildings and only relies on two weather elements. The requirement states that the FR application should store all weather data, even the ones that are not currently in use. This requirement is in place in case the FRI model is expanded and as a consequence has use for additional weather elements.

Requirement R4: Expendable Software.

- This requirement states that the components should be easy to expand when adding new functionality.

Requirement R5: User interaction and service.

- Users must be able to interact with the FR application. This can be through a user interface, or as a client with the use of a REST API that provides fire risk indication data.

Requirement R6: Hybrid fire risk calculations.

- This requirement goes together with R2. Since the weather data is converted to a common format it should be possible to combine historically weather data with forecast data when calculating fire risk.

Requirement R7: Continuous data harvesting.

- If the user wants to monitor fire risk for a location, the FR application should continuously collect weather data and calculate the fire risk.

Requirement R8: Cloud deployment.

- The FR application should have a software architecture that enables cloud deployment in order to make the application widely accessible for different types of clients.

Requirement R9: Storage and run-time efficiency.

- This requirement states that the application should have limited data storage needs, as well as run efficiently in order to avoid slowing down calculations.

4.3 Architecture Software Design

The prototype application is limited when it comes to calculating the fire risk. According to R4 and R3, the FR application has to be implemented in a modular way that can support the expansion of the application as well as take in use different weather elements that are currently not used in the FR application.

Figure 4.3 shows the proposed software architecture which divides the functionality into several components, which fulfils R4, each within their own processing. The functionality has been split into three main components, where the Fire Risk Prediction Model (FRP) component deals with the FRI model, and the data harvesting and collection component (DHC) deals with collecting weather data, both forecast and historical weather data. The third component will act as a controller service that handles communication from the end-user to the other two components. In this way all the functionality related to calculating the fire risk will be in one component and all functionality related to collecting and converting weather data is its own component. This way of structuring the functionality gives a lot of flexibility and the two components will not rely on each other. If there are any changes in one component, then the other components should not be affected in a way that would require a rewrite of the component. This way of structuring applications relies on the architectural style of micro services, as discussed in section 2.2. As stated in R8, each of the components will be hosted on a cloud platform for easy accessibility when using the FR application.

Since the functionality is split in several components, the communication differs somewhat from a traditional monolithic application. There are a few different ways to achieve this. Two possible approaches would be to rely on REST or message-based communication. Message-based communication offers asynchronous communication where the application will not block while waiting for an action to finish. Using REST to communicate may block the application when working with large amounts of data, as it has to wait for each part to finish before it can do another action. The FR application will use REST as communication. It is therefore important to abide by R9 to avoid too much blocking. Each component in the FR application will be implemented as a RESTful service, each with their own resource.

Below we provide a more detailed discussion of the components that make up the application.

4.3.1 Fire Risk Web Service

As stated by R5, the user should have to be able to interact with the FR application. The fire risk web service will act as the front-end of the FR application that the user will interact with. This web service contains all the available actions that the user may perform. Depending on the action, the service will communicate with the corresponding component that will execute the action and send it back to the service, which in turn sends it back to the user.

There will also be a web application that will be used for evaluation purposes. It is not intended for general use. This application will have a few

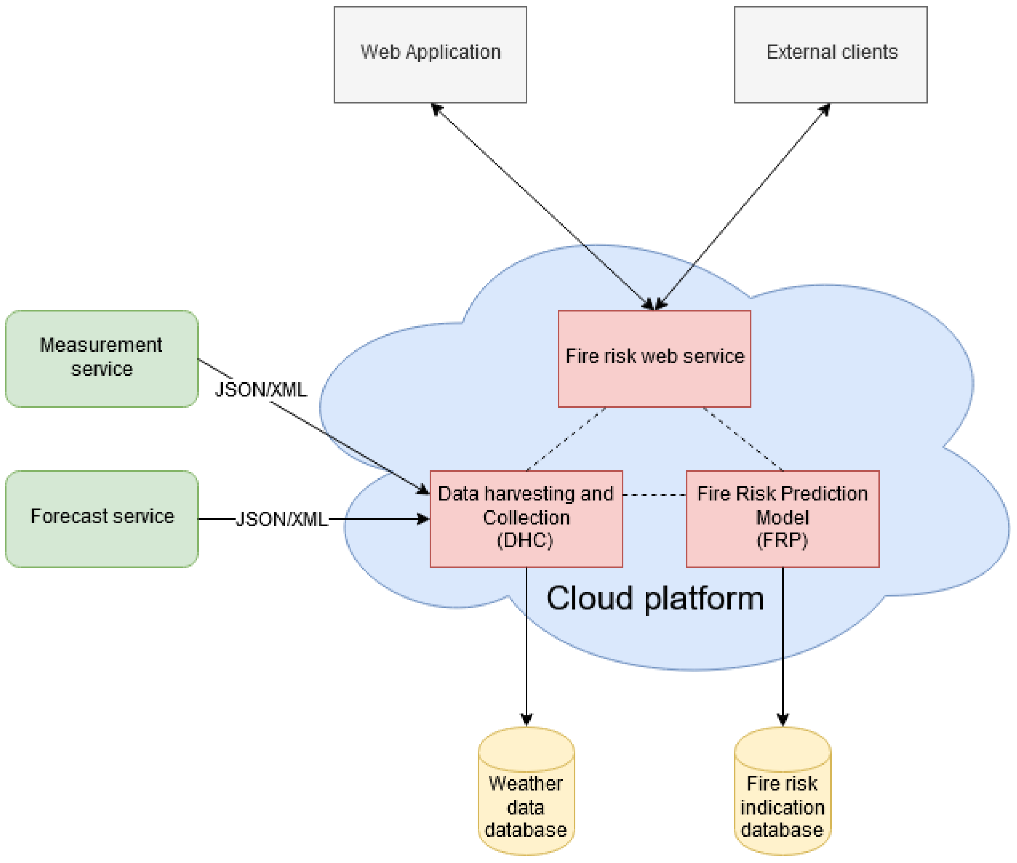


Figure 4.1: Proposed software architecture for the FR application

operations that will request weather data from different sources and display it in graphs for better overview and analysis. This can for instance be how accurate the forecast was versus the historical data recorded for that period.

4.3.2 Data harvesting and collection

As the name data harvesting and collection component (DHC) suggests, the two primary tasks is to harvest weather data and collect it if requested. The term weather data means both forecast and historical weather. The collection part will for the most part be necessary for evaluation of the FRI model, and may not be applicable when the FR application is put into production. However, harvesting of weather data will be required for both. From R1, it is this component that will communicate with the external services that provides forecast, and historical weather data. This data can then be used by the Fire Risk Predictive modelling component. When the data is collected it will be stored in a database, in accordance with R2. Whenever it fetches data, the component will store all available weather elements that may be used, even if unnecessary at the time being, in order to comply with R3. The component will also implement R7, where it will continuously harvest data for a location and run predictions on it by the help of a periodic timer that will run very 24 hour.

To achieve R9, the component will remove non-essential data that is part of the return data from the external services in order to limit the amount of data stored, and transferred in the FR application.

The component will have a resource available for the actions that a user may perform, such as starting a new continuous harvesting of data for a location and fetch weather data for a location in a given time span. It will then communicate with the predictive modelling component to compute the fire risk. There is also a few resources that will be available for evaluation purposes that fetches data that is stored in the database.

4.3.3 Predictive Modelling

The main purpose of the FRP component is to get weather data from the DHC and run it through the FRI model. Communication with this component will happen in two distinct cases. It will either be from the DHC after it has fetched data, or if a user wants to know about already computed fire risks. Whenever it is given new weather data to compute a fire risk, the result will be stored in the database. If a user wants fire risk indication for a specific location, it will be possible to check the database to see if it has already been computed before making a new request. This way it will be possible to decrease the number of requests that will be made to the external services, which conforms to R9.

The component will also convert the weather data from the sources to a common format to be used by the FRI model. The DHC will remove unnecessary data values, but it will still be in the same format as it was retrieved. The reason for converting it to the same format is to comply with R6, where it should be possible to make a fire risk by using both forecast and historical weather data.

4.3.4 Measurements Services

The measurements service offers historical weather data that can be used by the predictive modelling component. The FR application will make use of two services that offers historical data. One of them is from a reliable source that have strict rules when measuring data. The other one uses weather data from consumer grade weather stations which may be less accurate.

Part off the evaluation is to see how much the fire risk indication differs depending on which data source is used for the calculations.

4.3.5 Forecast Services

Forecast services offers predictions about the weather in the future. The FR application makes use of one such services that offers forecast data for around into nine days into the future.

4.4 Summary

Table 4.1 shows a summary of the requirements and where they are implemented in the FR application.

Requirement and Name	Implementation
R1: Weather data source	DHC communicates with several sources
R2: Common Data format	Converts weather data gathered from DHC in FRP
R3: FRI model refinement	All weather data stored in a database in DHC
R4: Expandable Software	Structuring the application as micro-services
R5: User interaction and service	Fire Risk Web Service API
R6: Hybrid Fire Risk Calculations	Converted weather data in FRP can be used together
R7: Continuous Data Harvesting	Timer in DHC harvest weather data
R8: Cloud Deployment	DHC, FRP and FR web service hosted on a virtual machine
R9: Storage and Run-time efficiency	Store as little as possible amount of data in DHC and FRP

Table 4.1: Summary of requirements and their implementation

Chapter 5

Implementation and Deployment

This chapter provides details on how the software architecture derived in the previous chapter has been implemented, including the components that implements the core functionality of the application.

5.1 External Services

As explained in Chapter 4, the application makes use of three different services. These are the services that provides the weather data needed for the FRI model in order to calculate fire risk. These services are provided as RESTful web service. As mentioned in chapter 2, REST is an architectural way of structuring web services that offers a lot of flexibility where one is not constrained by specific languages when creating the web service.

Two of the external web services used is from the Norwegian Meteorological Institute (MET), which is part of the Ministry of Climate and Environment of the Norwegian state. MET offers free weather data from Norway.

The two services focuses on two distinct types of weather data. The Frost web service[3] has a primary focus on weather data from the past. This service gives access to all the stored data that the meteorological institute has recorded since they started. It can offer data about climate normals and extreme records, but also regular every day weather measurements such as temperature, wind, and humidity. Then there is the MET API[4], which focuses on current weather and predicting the future weather. It is mostly used for getting access to weather forecast, but also offers lighting forecasts, where the hottest and coldest places are, ocean forecasts, and aviation forecasts.

The Frost web service is more modern and updated from an old SOAP-based web service and offers weather data represented as Json and XML format. The MET API is not as new as the Frost web service, but offers weather forecasts represented as XML and Json.

The most notable difference is that the Frost web service offers filtering of the response data when making requests. The filtering mechanism means that if provided, only the data that is specified will be returned. With the MET

API web service, such a filter is not available. This means that the return data contains unnecessary information that is not related to the weather data the FRI model needs. One of the requirements, R3, of the application was to store the return data from the web service in the database. The return data from the Frost web service can be stored right away, as it only contains values for the relevant weather elements. The return data from the MET API web service contains a lot of information that will not be used now or in the future. The return data from the MET API web service will be filtered in the component that handles the weather data collection and then stored in the database.

The third external web service is from a private organization, Netatmo[5], that sells consumer grade weather stations.

5.1.1 Frost Web Service

The Frost Web Service offers free access to METs historical weather and climate data. The data includes quality controlled daily, monthly and yearly measurements of temperature, precipitation, and wind data [3]. Metadata for the weather stations are also available through the web service.

The Frost API is a RESTful api that gives access to a lot of different resources about weather and climate data in Norway. In order to use the API, proper authentication is needed. This is done by creating an account on `frost.met.no`. By creating an account, a client identification key will be provided to the user, which again will be used when making request to the API to make sure it knows who connects to the API.

After the account has been created and, a client identification has been provided, the user can use the API freely. There are, however, certain restrictions on using the API. For instance, if a user has a large request this will have to be split into smaller request in order to prevent a time out from the API. It is also preferred to cache the data to avoid making unnecessarily large request more than once.

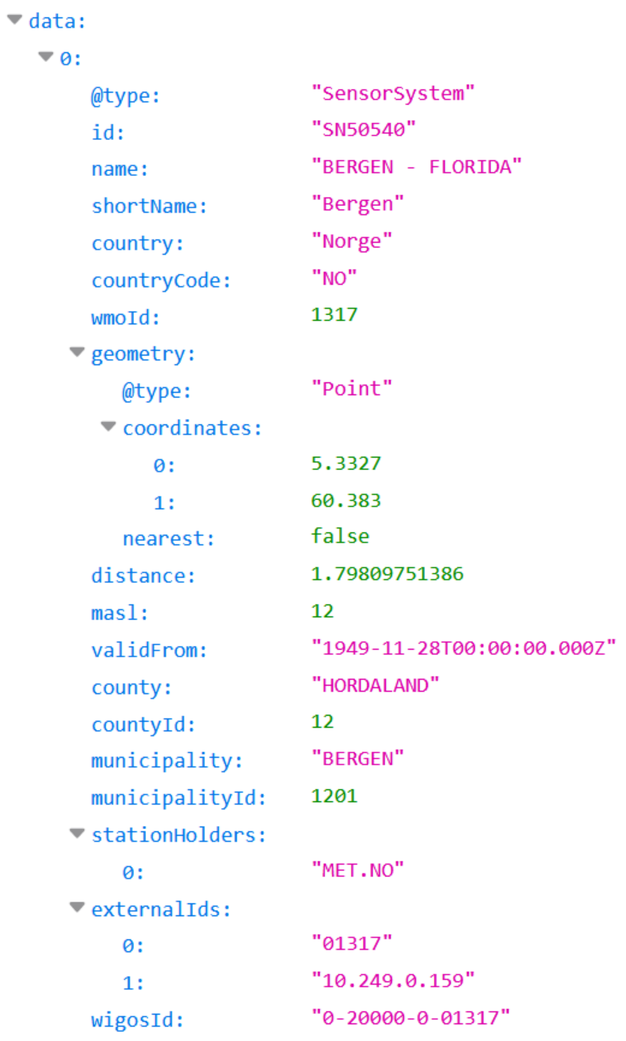
The Frost API gives access to resources about locations, weather records, observations, lighting, sources (weather station metadata), elements (weather elements), climate normals, and frequencies. Not all of these resources will be used by the FR application as they do not provide data required by the FRI model. The ones that are used are, location, observation, and meta data about the stations.

The location resources returns information about a location, such as name, coordinates of the location, and gives a brief description of the location. This information is obtainable by either providing a name of the place, or by providing longitude and latitude coordinates to find the nearest location. This can be used to give a description to the user when selecting a station for fire risk indication.

Sources are the entities that measures data. There are three different types of sources: Sensor Systems (SN), Region Data-set (TR, GR, NR), and Interpolated Data-set. Of the three types of sources, Sensor Systems is the relevant one in this case. This particular resource gives the metadata of the sources (weather stations). The metadata contains which type of source it is, its location in longitude and latitude, its name, information about the place it is located, its altitude, and from when it is valid and to when. To be able to fetch weather data, the identification of a source must be provided. With the help of this resource, it is possible to find the identification of the correct source for the location that a user wants to find the fire risk. There are a few different ways of finding the identification of the station. Most of them requires some previous knowledge as to how and where they are placed. There is, however, one way of finding a source identification by only knowing the latitude and longitude coordinates. There is a parameter in the resource that is called geometry. This parameter accepts a command of *nearest(POINT(longitude latitude)).* This command will find the nearest station to the given coordinates. However, this will give all registered stations in the area and the nearest one may not provide the necessary weather data. A work around to avoid this problem is to use a source that have been assigned a WMO (World Meteorological Organization) ID. These sources will measure the relevant weather elements, and will provide updateed measures every hour. It is then possible to specify that it should only return sources that have a WMO identification. By using the above command with the parameter that specifies which stations it should look for, the nearest station should contain the relevant weather elements. It is also possible to choose how many stations should be returned if that is necessary in the future. An example of a URL for finding a source is as follows:

```
https://frost.met.no/sources/v0.jsonId?geometry=
nearest(POINT(5.3501755 60.3693299))&wmoid=1*
```

An example of return data from this URL can be seen in Figure 5.1. In this particular example, a user wants to find the nearest source with the latitude and longitude coordinates of 60.3693299 and 5.3501755, respectively. The identification of the nearest source can be found in the field labelled id. Since this is a sensor system it has a prefix of SN followed by a number. Information about the location is also present, as well as the distance from the station to the requested coordinates.



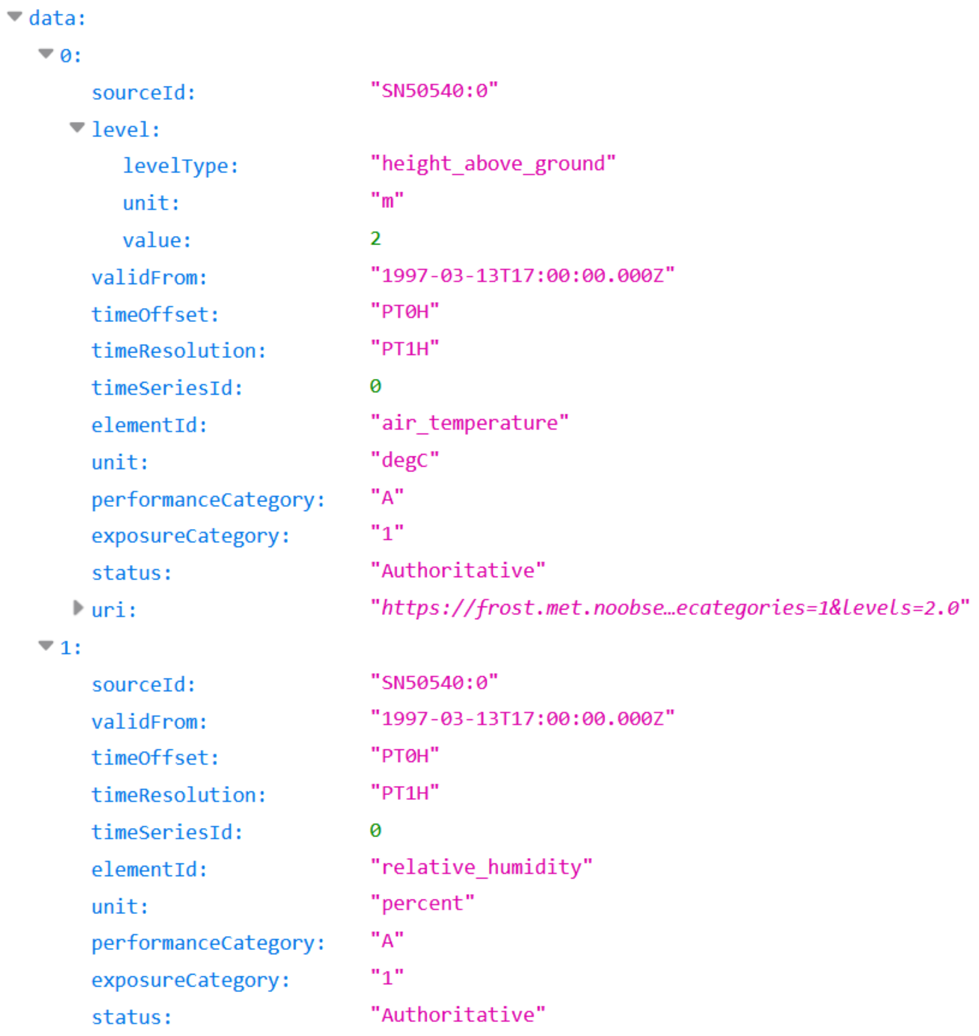
```
▼ data:
  ▼ 0:
      @type:              "SensorSystem"
      id:                 "SN50540"
      name:               "BERGEN - FLORIDA"
      shortName:          "Bergen"
      country:            "Norge"
      countryCode:        "NO"
      wmoId:              1317
    ▼ geometry:
        @type:            "Point"
      ▼ coordinates:
          0:              5.3327
          1:              60.383
        nearest:          false
      distance:           1.79809751386
      masl:               12
      validFrom:          "1949-11-28T00:00:00.000Z"
      county:             "HORDALAND"
      countyId:           12
      municipality:       "BERGEN"
      municipalityId:     1201
    ▼ stationHolders:
        0:                "MET.NO"
    ▼ externalIds:
        0:                "01317"
        1:                "10.249.0.159"
      wigosId:            "0-20000-0-01317"
```

Figure 5.1: Return data for finding the nearest source

The observations resource, which is a part of the Frost API, is where the FR application gets access to the weather data needed for the FRI model. The observation resources are divided into five different sub resources, which will be explained in greater detail below.

- **Available Time Series** gives information about which weather elements it measures as well as how often it measures specific weather elements.
- **Observation** provides weather data from the presented weather elements the users requests.
- **Meta data update count** is not yet implemented
- **Available Quality Codes** gives a brief description about the quality codes from the observation.
- **Quality** provides a more detailed information about the quality codes. It is not properly implemented yet.

Available Time Series is a resource that can be used to find information or metadata about what and how often a weather station measures. This will be which weather element it measures, how often it is measured as well as the identification of the weather element, the unit of measure of the weather element, and a lot of additional information that describes the weather element. This identification will be used later in order to find the measurements from the stations. Some of the available time series for the source in Figure 5.1 can be seen in Figure 5.2. The most important information from this data is the weather element id, and the time resolution. The time resolution specifies how often the data is measured, which in the case of the FR application has to be every hour. In figure 5.2, the weather element air_temperature has a time resolution of "PT1H", which means that it will measure air temperature every hour of the day.



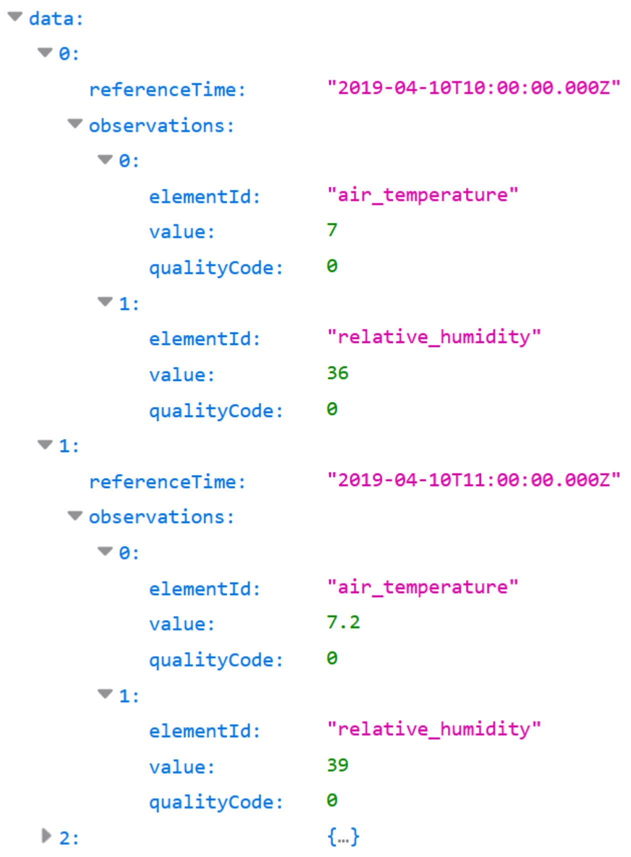
```
▼ data:
  ▼ 0:
      sourceId:               "SN50540:0"
    ▼ level:
        levelType:            "height_above_ground"
        unit:                 "m"
        value:                2
      validFrom:              "1997-03-13T17:00:00.000Z"
      timeOffset:             "PT0H"
      timeResolution:         "PT1H"
      timeSeriesId:           0
      elementId:              "air_temperature"
      unit:                   "degC"
      performanceCategory:    "A"
      exposureCategory:       "1"
      status:                 "Authoritative"
    ▶ uri:                    "https://frost.met.noobse…ecategories=1&levels=2.0"
  ▼ 1:
      sourceId:               "SN50540:0"
      validFrom:              "1997-03-13T17:00:00.000Z"
      timeOffset:             "PT0H"
      timeResolution:         "PT1H"
      timeSeriesId:           0
      elementId:              "relative_humidity"
      unit:                   "percent"
      performanceCategory:    "A"
      exposureCategory:       "1"
      status:                 "Authoritative"
```

Figure 5.2: Available time series for source SN50540

Observations returns the observed measures from a source. It is from this sub resource that the weather data can be found. There are a few parameters that are required in order to retrieve the data. The first and most important is the source identification, which can be found as was described above. Then, a time interval must be specified. The time interval can be specified in two ways. By specifying one date, it will retrieve data from the specified date to present date of the request. The other way is to specify from one date in the past to another date in the past. The last required parameter is the identification of the weather elements that should be retrieved. If the source is known, it is possible to use the available time series to find which weather elements are available. However, in the case of the FR application, these parameters are hard coded in the application, as it is unnecessary to check every time once it is known which sources are used. There are some sources that measures weather data at different time resolutions. Some of them do every hour, some do every 30 minutes as well as every hour. As the FR application is only concerned with data every hour, it can specify a parameter that only data that is measured every hour should be returned. There is also the case of unnecessary return data that will not be used. The Frost API has a parameter called *field*, where the requester can specify which return values should be sent back. To retrieve weather data for a location for a specific time, the following URL may be used:

```
https://frost.met.no/observations/v0.jsonld?sources=[SOURCE_ID]
&referencetime=[FROM_TIME/TO_TIME]
&elements=air_temperature,relative_humidity
&timeresolutions=PT1H
&fields=elementId,value,referenceTime,qualityCode
```

If the same user as above wants to find weather data using the source, *SN50540*, that the user identified previously, it must be placed where *SOURCE_ID* is. Then the user would have to pick a time, in this case it could be *2019-04-20T10 00/2019-04-10T11 00*. This will find all recorded weather that the source *SN50540* have recorded in the time span of April 20th at 10 am to April 21th at 10 am. The return data of this request can be found in figure 5.3.



```
▼ data:
  ▼ 0:
      referenceTime:      "2019-04-10T10:00:00.000Z"
    ▼ observations:
      ▼ 0:
          elementId:      "air_temperature"
          value:          7
          qualityCode:    0
      ▼ 1:
          elementId:      "relative_humidity"
          value:          36
          qualityCode:    0
  ▼ 1:
      referenceTime:      "2019-04-10T11:00:00.000Z"
    ▼ observations:
      ▼ 0:
          elementId:      "air_temperature"
          value:          7.2
          qualityCode:    0
      ▼ 1:
          elementId:      "relative_humidity"
          value:          39
          qualityCode:    0
  ▶ 2:                    {…}
```

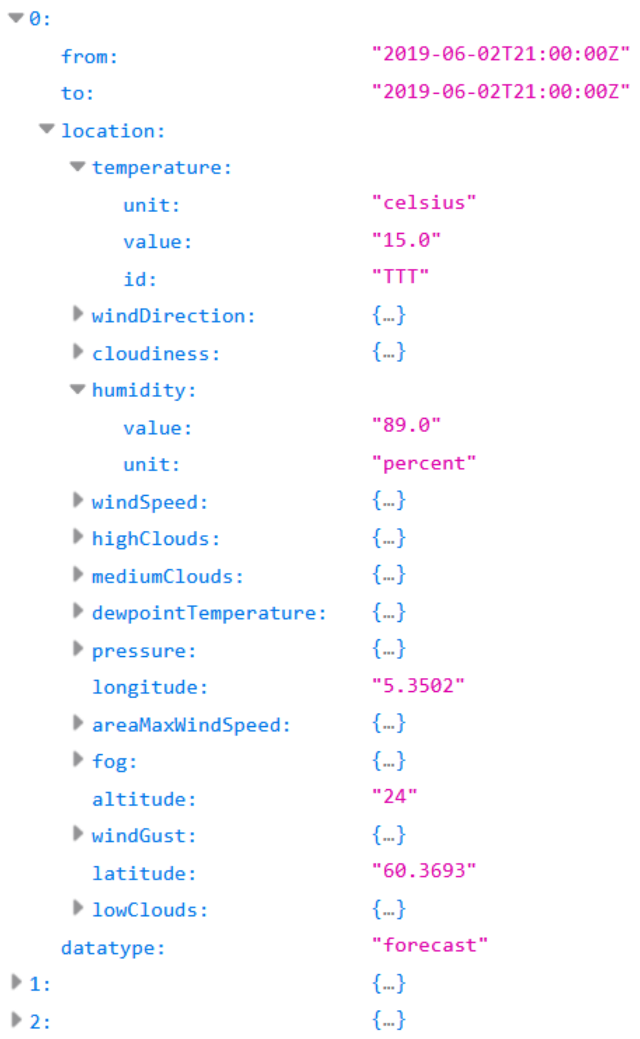
Figure 5.3: Return filtered data from observation request

Available Quality Codes returns a list containing definitions for the different quality codes. The current quality codes used by the Frost service ranges from 0-7 where 0 is an accurate measure and 7 is an unreliable measure. The quality code informs about the quality of the observation and can be used to inform about uncertainties in the data.

5.1.2 Met Web Service

As with the Frost web service, the MET web service offers free access to a range of different weather resources, such as air quality, aviation forecast, lighting map, UV forecast, and location forecast. Most notable in this case is the location forecast service. The location forecast service provides access to the weather forecast for a specified location. Unlike the Frost web service which required authentication, the location forecast does not. It has only one resource available and that is to find the forecast of a location. This location must be specified on the longitude and latitude format. The returned forecast will be the weather every hour for the first tree and a half days, and every 6 hour up to the 9th day. The location forecast can be obtained by the following URL: `https://api.met.no/weatherapi/locationforecast/1.9/.json?lat=LAT&lon=LON`.

An example of what will be returned if a user wants the forecast for a location with the latitude coordinate 60.3693299 and longitude coordinate 5.3501755 can be seen in figure 5.4. For each of the fields in the returned data, it will specify the return type, value and unit type.



```
▼ 0:
    from:                       "2019-06-02T21:00:00Z"
    to:                         "2019-06-02T21:00:00Z"
  ▼ location:
    ▼ temperature:
        unit:                   "celsius"
        value:                  "15.0"
        id:                     "TTT"
    ▶ windDirection:            {…}
    ▶ cloudiness:               {…}
    ▼ humidity:
        value:                  "89.0"
        unit:                   "percent"
    ▶ windSpeed:                {…}
    ▶ highClouds:               {…}
    ▶ mediumClouds:             {…}
    ▶ dewpointTemperature:      {…}
    ▶ pressure:                 {…}
      longitude:                "5.3502"
    ▶ areaMaxWindSpeed:         {…}
    ▶ fog:                      {…}
      altitude:                 "24"
    ▶ windGust:                 {…}
      latitude:                 "60.3693"
    ▶ lowClouds:                {…}
    datatype:                   "forecast"
▶ 1:                            {…}
▶ 2:                            {…}
```

Figure 5.4: Return data for a weather forecast

5.1.3 Netatmo Web Service

The Netatmo web services uses privately owned weather stations for retrieving historical weather data. These must be bought at a local store or at a web store. The base package of the Netatmo weather station comes with an indoor module, and an outdoor module. The indoor module measures temperature, humidity, $C0_2$ and noise levels. The outdoor module measures temperature, humidity, and air pressure. The indoor module requires an Internet connection in order for it to send measured data to the Netatmo back-end cloud service. The indoor module acts as a gateway controller for the outdoor module, where the outdoor module sends measured data to the indoor module which in turn sends it to the back-end server.

In order to use the Netatmo weather station, an account must be made, either through their website (`www.netatmo.com`) or on the mobile phone application available on both Android and iOS. In the mobile application or the web site, there are instructions for how to install new weather stations and setting up the network connection.

The Netatmo API offer different services based on the types of Netatmo product the user has available. There are security products, energy products, and the weather products. In the case of the FR application, only the weather products are used. The base package for the weather products measures temperature and humidity, but it is possible to buy extension modules that measures wind speed and direction, and rain levels. For the weather products there are three services available, but not all of them will be used. There is a service for getting public data from all stations in an area, and this may be used in the future, but for now it is not in use. The two services are split in a control service, that handles authentication, and a data service, that enables the user to fetch measurements and meta data of the Netatmo stations. The metadata for the stations contains the identification of the indoor and outdoor module, as well as general information about the stations such as battery life, up time, location, and which modules are connected and what they measure. The identifications are used by the data service to retrieve measured weather data from the outdoor module.

As with the Frost Web service, authentication is required for making requests to their weather API. For authentication purposes, the control service is used for creating access tokens that can be used to gain access to the data service. This access token can be obtained by using a client identity and a client secret. The client secret and identity can be found by navigating to the developer page of Netatmo[5]. At this page, there is an option to create an app and by creating this app, the user will be assigned a client identity and secret. When the client secret and identity is obtained, it will be possible acquire the access token required for retrieving measurements and station metadata. To get the access token the following HTTP POST must be made to the Netatmo API

```
https://api.netatmo.com/oauth2/token
```

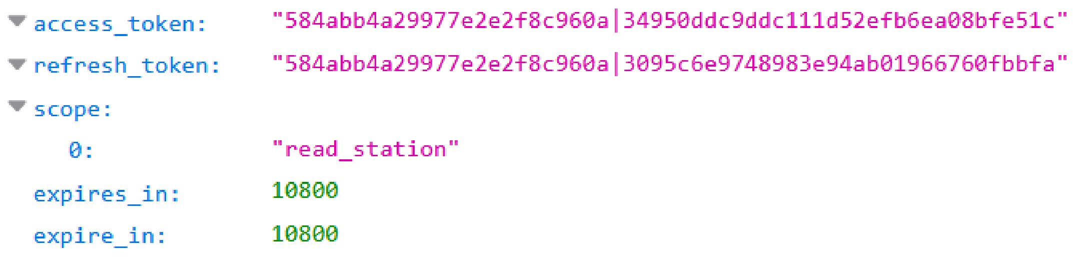
with these url parameters:

```
grant_type=password&clientId=YOUR_CLIENT_ID
&cleint_secret=YOUR_CLIENT_SECRET
&username=USERNAME&password=PASSWROD
```

This will retrieve the access token, as well as a refresh token. This refresh token can be used to acquire a new access token later by replacing the user name and password from the URL parameters with the following URL parameters:

```
grant_type=refresh_token&refresh_token=REFRESH_TOKEN
&client_ID=YOUR_CLIENT_ID
&client_secret=YOUR_CLIENT_SECRET
```

The return data for getting the access and refresh token can be seen in Figure 5.5. The access token has a limited duration before a new token must be assigned. In Figure 5.5 there is a field named *expires in* that provides information on how long the token can be utilized before a new one has to be issued. The refresh token value in the return data can be used for retrieving a new token instead of the user name and the password.



```
▼ access_token:   "584abb4a29977e2e2f8c960a|34950ddc9ddc111d52efb6ea08bfe51c"
▼ refresh_token:  "584abb4a29977e2e2f8c960a|3095c6e9748983e94ab01966760fbbfa"
▼ scope:
     0:           "read_station"
  expires_in:     10800
  expire_in:      10800
```

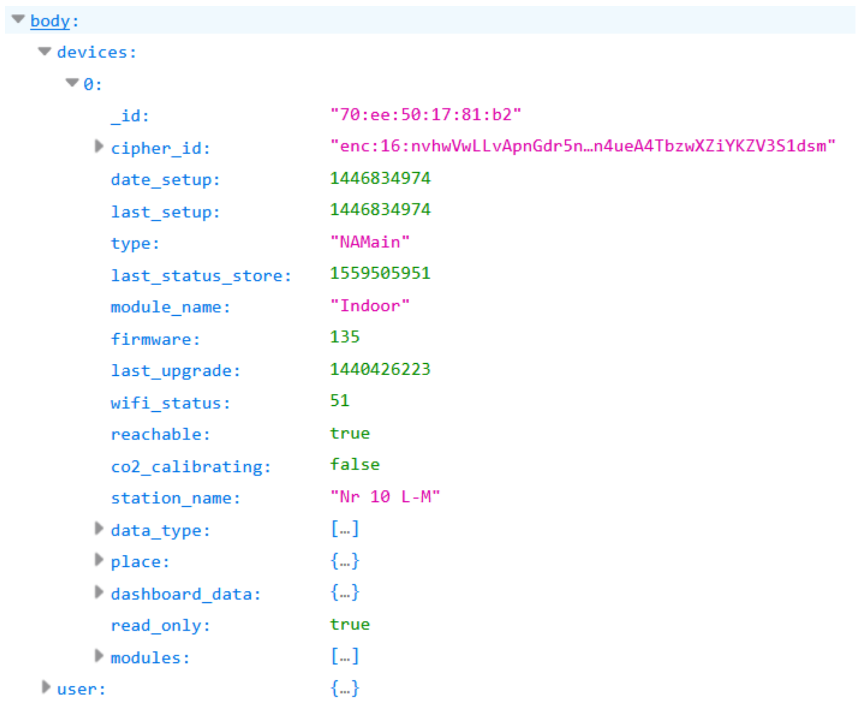
Figure 5.5: access and refresh token

With the new acquired access token, the two services can be accessed. As mentioned, the identification of the outdoor module is essential to find the measures recorded by that stations. This is done by the service that retrieves the metadata for the station. To do this, the MAC address of the indoor module must be known. The address can be found in either the mobile application or the web application. On the mobile application it will be under settings and found by clicking the station name. On the web application, the station must be selected and then click *Manage my station*. The station metadata can then be obtained via the following URL:

```
https://api.netatmo.com/api/getstationsdata
?access_token=ACCESS_TOKEN
&device_id=MAC_ADDRESS
```

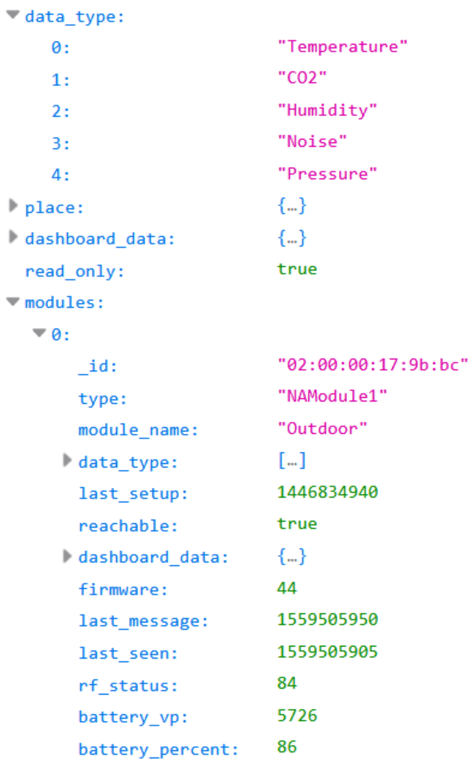
An example of the return data from the service is shown in Figure 5.6. As can be seen, there is a lot of information about the status of the weather station itself. The most relevant information, in the case for the FR application, can be found under the data type and modules field. The information inside these fields can be seen in figure 5.7.

Inside the data type field, one can find information about what weather elements the station can provide measures for. Note that some of the weather elements may be limited to one or more modules. For instance the indoor module will not measure pressure and the outdoor module does not measure CO_2 levels. The commonality between the modules is the temperature and the humidity. In the modules field, all the available modules will be listed such as outdoor modules, rain gauges, and wind turbines. In this case only one outdoor module is connected to the indoor module. It is possible to have more than one outdoor module if that is necessary. As with the indoor module, the identification field is the MAC address of the outdoor module that will be used for identifying the correct outdoor module when fetching the measurements.



```
▼ body:
  ▼ devices:
    ▼ 0:
        _id:                 "70:ee:50:17:81:b2"
      ▶ cipher_id:           "enc:16:nvhwVwLLvApnGdr5n…n4ueA4TbzwXZiYKZV3S1dsm"
        date_setup:          1446834974
        last_setup:          1446834974
        type:                "NAMain"
        last_status_store:   1559505951
        module_name:         "Indoor"
        firmware:            135
        last_upgrade:        1440426223
        wifi_status:         51
        reachable:           true
        co2_calibrating:     false
        station_name:        "Nr 10 L-M"
      ▶ data_type:           […]
      ▶ place:               {…}
      ▶ dashboard_data:      {…}
        read_only:           true
      ▶ modules:             […]
  ▶ user:                    {…}
```

Figure 5.6: Metadata for the weather station



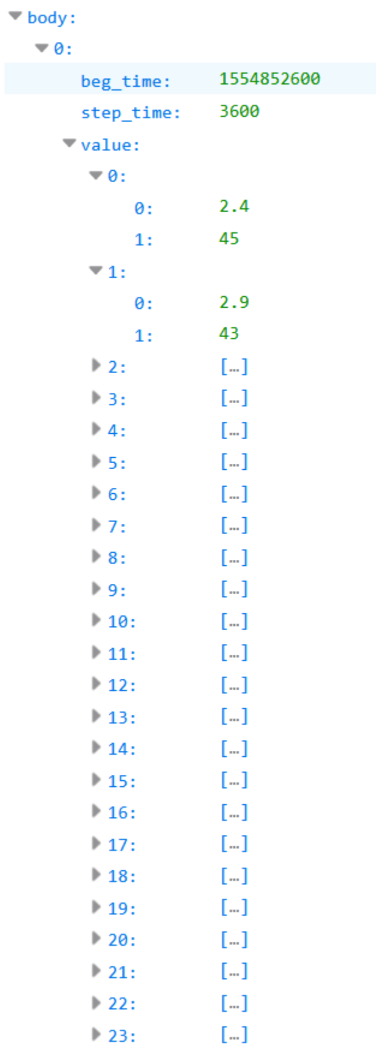
```
▼ data_type:
    0:                   "Temperature"
    1:                   "CO2"
    2:                   "Humidity"
    3:                   "Noise"
    4:                   "Pressure"
▶ place:                 {…}
▶ dashboard_data:        {…}
  read_only:             true
▼ modules:
  ▼ 0:
      _id:               "02:00:00:17:9b:bc"
      type:              "NAModule1"
      module_name:       "Outdoor"
    ▶ data_type:         […]
      last_setup:        1446834940
      reachable:         true
    ▶ dashboard_data:    {…}
      firmware:          44
      last_message:      1559505950
      last_seen:         1559505905
      rf_status:         84
      battery_vp:        5726
      battery_percent:   86
```

Figure 5.7: Data types and modules

The process of getting measurements is much the same as with the observation service from the Frost web service. First and foremost, the access token must be provided, so that the API can ensure that the correct person accesses the station data. The identification of both the outdoor and indoor module is to be provided. As with the Frost web service, the weather data is updated around every hour, and it is possible to specify the time interval for the measurements that should be returned. In this case it will be every hour. It is also possible to specify dates. If none is provided it will return every recorded measurements starting from the install date, with a maximum of 1024 measurements. There is also a type field where it is possible to choose what weather elements should be returned. Only temperature and humidity will be returned, as that is what the base package of the Netatmo stations that are in use offers. The URL for requesting the measurements is the following:

```
https://api.netatmo.com/api/getmeasure?access_token=[YOUR_TOKEN]
&device_id=[YOUR_DEVICE_ID]&modeule_id=[YOUR_MODULE_ID]
&scale=1hour
&date_being[START_DATE]&date_end[END_DATE]
&type=temperature,humidity
```

As a note, when choosing start date and end date it must be provided in seconds from 1 Jan 1970 to the selected date. Data from this request can be seen in figure 5.8. The data states at what time and date the first measurement is from, in seconds from 1 Jan 1970, as well as the step time between each measurement in seconds. Then the measurement values are given as a list of measurements for every hour from the start date until the end date as specified in the request. The values themselves are ordered in the same way as specified in the request URL.



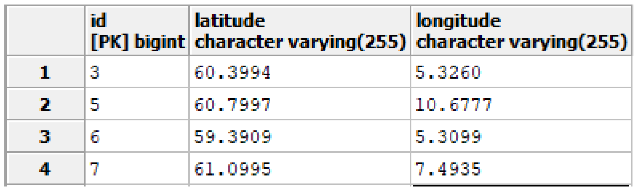
```
▼ body:
  ▼ 0:
      beg_time:    1554852600
      step_time:   3600
    ▼ value:
      ▼ 0:
          0:       2.4
          1:       45
      ▼ 1:
          0:       2.9
          1:       43
      ▶ 2:         […]
      ▶ 3:         […]
      ▶ 4:         […]
      ▶ 5:         […]
      ▶ 6:         […]
      ▶ 7:         […]
      ▶ 8:         […]
      ▶ 9:         […]
      ▶ 10:        […]
      ▶ 11:        […]
      ▶ 12:        […]
      ▶ 13:        […]
      ▶ 14:        […]
      ▶ 15:        […]
      ▶ 16:        […]
      ▶ 17:        […]
      ▶ 18:        […]
      ▶ 19:        […]
      ▶ 20:        […]
      ▶ 21:        […]
      ▶ 22:        […]
      ▶ 23:        […]
```

Figure 5.8: Return data for getting measurements from the outdoor module

5.2 Databases and Storage

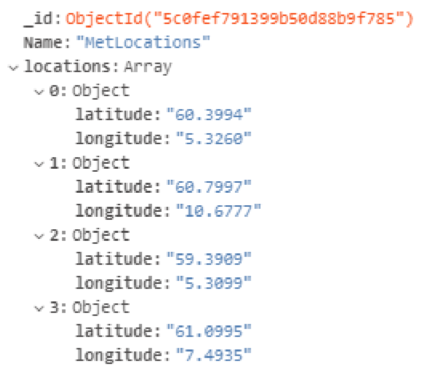
As discussed in section 4.2, two of the components use databases to store weather data and location data. For each component there are two databases, one relational database and one noSQL database, where the latter is a so-called document database.

The data harvesting and collection component uses the database for storing the weather data from the external services. It also stores the location of where it should do continuous data harvesting from. Storing a location can be done in three ways: one is using latitude and longitude coordinates for forecast data and the corresponding Frost source. The Frost source is obtained in the same way as described previously. It is also possible to add a Netatmo station by adding the MAC address of the indoor and outdoor module. In the noSQL database, this information is stored in Json format. While in the relational database it is stored in its own table. An example of how the latitude and longitude is stored in the different databases is shown in figure 5.9 and figure 5.10.



	id [PK] bigint	latitude character varying(255)	longitude character varying(255)
1	3	60.3994	5.3260
2	5	60.7997	10.6777
3	6	59.3909	5.3099
4	7	61.0995	7.4935

Figure 5.9: Table for latitude and longitude location in SQL



```
_id: ObjectId("5c0fef791399b50d88b9f785")
Name: "MetLocations"
⌄ locations: Array
  ⌄ 0: Object
      latitude: "60.3994"
      longitude: "5.3260"
  ⌄ 1: Object
      latitude: "60.7997"
      longitude: "10.6777"
  ⌄ 2: Object
      latitude: "59.3909"
      longitude: "5.3099"
  ⌄ 3: Object
      latitude: "61.0995"
      longitude: "7.4935"
```

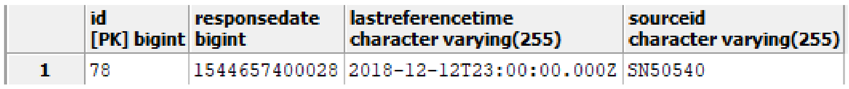
Figure 5.10: Json document in the noSQL database

For the weather data, as long as it is in Json format, it can be stored directly in the noSQL database. To be stored in a relational database, it must be converted to the corresponding entities in the application that will map it to a table in the SQL database. There are libraries available that converts Json format to other programming language. Since the FR application is implemented in Java, Gson[21] is the library that has been used for this purpose. In order to convert from Json to Java, a Java class with the corresponding fields and value types can be created. Gson will then map the fields and their values to the same variables and types in Java Objects. These Java Objects can then be mapped to tables and columns in databases with the help of other libraries, such as the Java Persistence API[2] (JPA). The JPA will then create a table, in the relational database, for that class where each row is a object variable from the Java class. The entity in a Java class can be seen in the code example below, and the corresponding table in the SQL database is seen in figure 5.11. It is also possible to use methods within gson that can represent the Json data without having to map it to a Java Class.

```java
@Entity
Public class FrostResponse implements Serializable {
    @Id
    @GeneratedValue
    private Long id;

    private long responseDate;
    @OneToMany(Cascade = CascadeType.PERSIST)
    private List<FrostData> data;
    private String sourceId;
    private lastReferenceTime;
}
```

Listing 5.1: Entity in Java



	id [PK] bigint	responsedate bigint	lastreferencetime character varying(255)	sourceid character varying(255)
1	78	1544657400028	2018-12-12T23:00:00.000Z	SN50540

Figure 5.11: Table in SQL database

When the fire risk indication is computed, it must be converted to Json in order to store it in the noSQL database. As for the weather data, Gson is used for this action. By having Java objects available it will create Json text from the Objects that can be used to stored in the database.

5.3 Implementation

Will now give a brief overview of the developed prototype application. The prototype consists of a front-end and a back-end. The front-end handles how the user interacts with service and uses the components from the back-end, which in turn deals with calculating and fetching data.

5.3.1 Functionality and Use Cases

In chapter 4, we discussed the different requirements for the FR application. R5 states that a user should be able to use a fire risk web service where they can specify if they want to subscribe to a fire risk for a given location and time. A second requirement was that they should be able to start a new continuous fire risk calculation for a given location. These will be the two primary use cases of the FR application.

The user has access to a web service that contains all the available resources that can be used. These resources have certain parameters that must be filled in by the user in order to retrieve the relevant fire risk indications. This includes the longitude and latitude coordinates of the location for the fire risk indication as well as a time frame (if required). If location has both a consumer grade station and a station from the Norwegian Meteorological Institute, the FR application chooses the nearest one. When starting a new continuous fire risk indication, the type of station must be entered. This can be a Netatmo station, by providing the MAC address for the indoor and outdoor modules. In the case a Frost station is used, longitude and latitude must be provided for FR application to find the one with the nearest location. Note that, if the Netatmo station is not calibrated the measures may be inaccurate.

The use cases rely on different types of weather data. The use case that runs continuously uses historically data, that can be from either the consumer grade weather station, or the Norwegian Meteorological Institute. It will also be possible to add a forecast to the fire risk indication if specified. The use case that provide fire risk indication for a location and a time point can both use historically data, but it can also use weather forecast data. If the time provided is in the past, then it will use historically data. If a time point is not provided and they want to know the fire risk in the coming days, the weather forecast will be used for calculations. However, in order to have more accurate indication when only using forecast data, some historical weather data will be added on top of the forecast which can result in more accurate indications as this will serve to calibrate the FRI model.

The main functionality of the application is to fetch weather data from the external services and then run it though the FRI model for determining fire risk for a given area. This functionality is split in two categories, depending on what the user of the application wants.

- The first category is where the user specifies to get fire risk for a time and location, where the location is on the longitude latitude format. This choice can use both use historically weather data, and also forecast weather data. The user can input the location by typing in the longitude and latitude of the location, and the time period for the weather readings. This time period can be specified in four ways.

  – The time period can be based on two dates in the past.
  – The time period can be from a date in the past to the present date.
  – The time period can be from a date in the past to the present date, as well as specifying that forecast weather should also be added after the present date.
  – The time period is in the future and the data used can only be from the weather forecast.

- The second category is when the user wants to start a new continuous fire risk indication for a location. By using this option, the application will begin collection data from the location provided by the user. This data collection will happen every 24 hours, currently at 00:30 CET every day. The user still needs to specify a location, same as the other category with longitude and latitude. The application will then find the nearest weather data source, as discussed in Section 5.2.3. The user is also given an option to add their own station if they are from Netatmo.

The FR application uses a Java enterprise architectural style which is comprised of a client tier, web tier, business tier, and an enterprise information tier. Each of these tiers take care of their own part of the system. The client tier is usually a normal web browser communicating with the web application. The web tier is the view, or the web pages the application generates to present information on the screen to the user. The business tier is where the business logic of the application is implemented, e.g. for communicating with databases, or different components, or external services.

In the case of the FR application, the components can be placed in the different layers. For instance, the fire risk web service has the responsibility of communicating with the client and would be placed in the web tier and can be considered the front-end part of the FR application.

For the two other components, they would be placed in the business tier as they handle all business logic. The DHC component handles communication with the external services, whereas the FRP component handles the logic responsible for creating the fire risk indication that the users retrieves via the fire risk web service. These two components makes up the back-end of the FR application.

5.3.2 Front-End Implementation

The front-end of the FR application consists of two parts. The controller service that handles communication with the back-end services. It also has a web application that is used for evaluation of the FRI model.

The controller service is implemented with the Spark Framework[19]. Spark Java is an easy to use framework for setting up RESTful application services.

In order to use Spark Java, the Maven repository must be added to the POM-file. Then, all that is required is to create a class with a Main method and add GET, POST, PUT or UPDATE methods, as seen on line 5 in the code example below. Spark Java comes with a built-in web server that will run when the application is started. There exists several configuration options, however that will not be necessary to cover more closely for simple use.

```
1 import statis spark.Spark.*;
2
3 public class HelloWorld() {
4         public static void main(String[] args) {
5                 get("/hello", (req, res) -> "Hello␣World!");
6         }
7 }
```

Listing 5.2: GET method in Spark Java

The web application created for visualisation and analysis of the results, was built using JSF(Java Server Faces) and Facelets. JSF is a server-side component[2] framework for building web applications with Java. JSF gives access to an API[2] for representation of components and their states, handle events, and server-side validation. It also has a tag library[2] to add components to the web page and connecting the components to server-side objects. JSF works with both JSP (Java Server Pages), but works best with the use of Facelets.

Facelets is a lightweight[2], but at the same time powerful, page declaration language that is used to build Java Server Faces using HTML style templates, and for building the component tree. Facelets uses XHTML, which is a combination of XML and HTML structure, to build the web pages.

JSF uses a standard MVC (model, view, controller) architectural pattern, which can be seen in figure 5.12. This type of pattern splits the functionality in three separate parts. The model, the view of the model, and the controller of the view. The model represents the data that is passed through the application or between the view and the controller. The view is the user interface that is represented on the screen and can be displayed in the web browser. The controller acts as an intermediary between the view and the model in order to process the incoming request from the view and then to manipulated the model. When the model has been manipulated, the controller interacts with the view to render ab updated view based on the data that was manipulated.

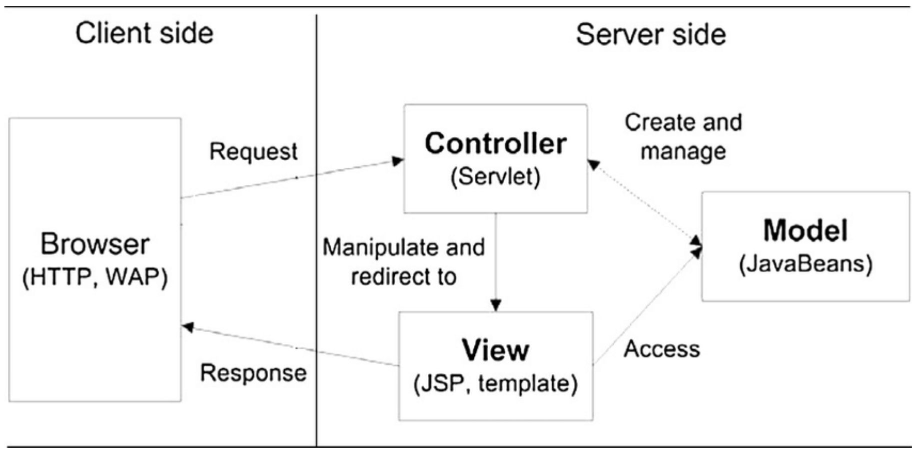


Figure 5.12: Java Server Faces architecture

To visualize data in the form of graphs, a library called PrimeFaces[22] was used. This library has a wide variety of graph types available, but for the most part line graphs will be used.

In the visualization application, the view would be the Facelet pages which creates what is seen on the web page. Then ther is the Faces Servlet which acts as the controller. And lasttly, the managed beans in JSF corresponds to the model in the MVC architectural pattern.

5.3.3 Back-end Implementation

The back-end part of the application consists of two main components that handle the logic for harvesting and collecting data and indicating the fire risk. These components are written in Java and are also implemented as a RESTful web service. This is achieved with the use of Java Enterprise Editions JAX-RS service. The JAX-RS service is an API for Java that makes it possible to create web services on the REST architectural pattern.

The back end also uses EJB (Enterprise Java Beans)[2] which is an API that Java provides for handling business logic in Enterprise Applications. EJB is a web container that provides security, Java servlet lifecycle, transaction processing and timers. The FR application is mainly concerned about using the transaction processing, but the DHCC will use timers as part of the continuous harvesting of weather data.

Transaction processing is needed when working with databases in order to ensure integrity of data. The main issue here is to ensure that consistent data is stored in the databases. A transaction is a series of actions that have to end successfully or else all actions are rolled-back. If transaction control is not used, all changes that are successful, will be committed to the database. This means that if two data-sets, that are dependent on one another, are stored in the database, and one of them fails, the one that is successful will still be added to the database. If the application wants to use the two data-sets later, it will encounter an error because only on of the data-sets is to be found in the database. This is what the transaction control will prevent from happening in the application. Instead of having one of the databases used succeed and the other fail, it will fail both of them. The application will then be able to try again until all actions are successful. The transaction will end in either a commit, where all actions are successful, or in a rollback. If a rollback happens one or more actions was not successful and the data will not be committed to the database.

The timers from the EJBs is a scheduler that makes it possible to specify when certain jobs should be executed. This can be once every 5 hour, every Monday at 10 pm and so on. The code example below showcases a timer that will execute every 24 hour at 00:30 am CST.

```
@Schedule(minute = "30", hour = "00", dayOfWeek = "sun-sat",
persistent = false, info = "collectionTimer", timezone = "Europe/Oslo")
```

Listing 5.3: Timed schedule example

Class Structure

The class structure of the back-end components is shown below. Some of the classes that make up the application are used by multiple components and will only be shown once. These are usually utility classes that have operations regarding conversion of data and setting up database connections, and model classes used for representing the data. Figure 5.13 illustrates the models used for representing the weather data as well as the different sources that can be used for retrieving weather data.

In figure 5.15 the relation between the primary classes that make up the DHC component is shown. This component uses a singleton class that sets up the scheduler used for retrieving weather data periodically. The scheduler class has relations to three facade classes that have the purpose of storing and retrieving weather data. For each set of external services used there is a corresponding facade class that has operations regarding the use of that particular sets of weather data. In each of the facade class listed in figure 5.15 there is a variable MongoDbFacade, which can be seen in figure 5.14 and is common for all. The Mongo Database needed extra configuration in order to be operable. This was done by creating a factory class that implements the java Object Factory, which is then used to create the Mongo Database instance used by several of the classes.

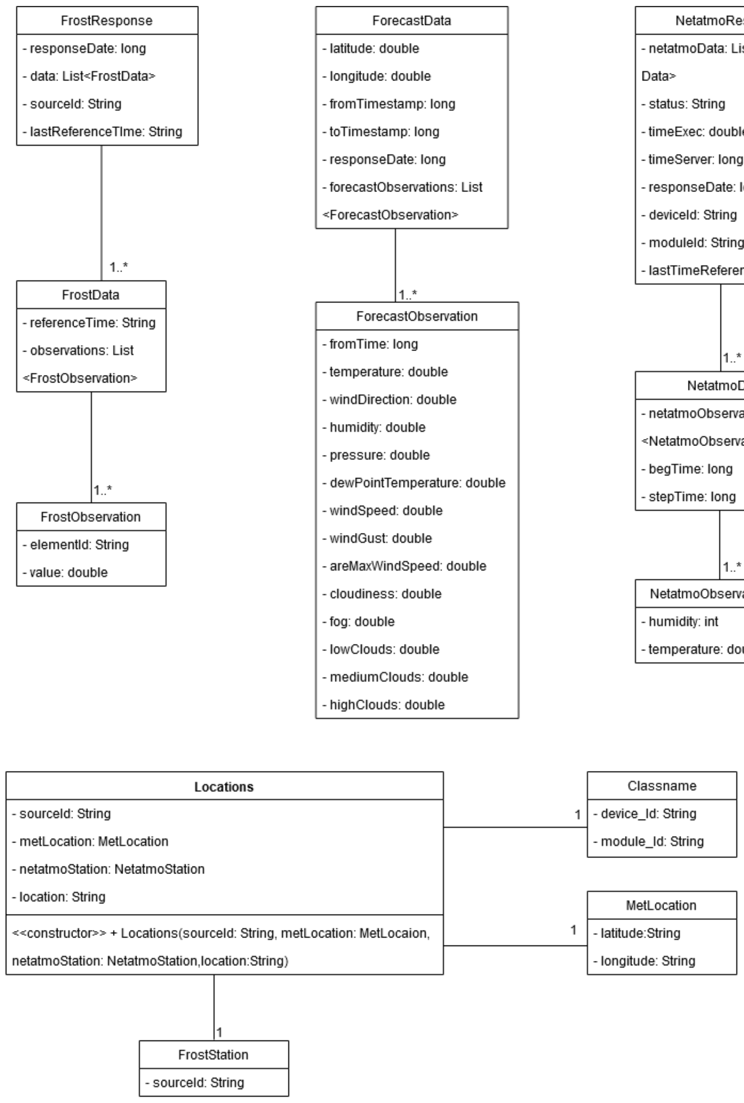


Figure 5.13: Class diagram of the models used by multiple components in the application

Figure 5.14 lists the utilities that are used by the application.

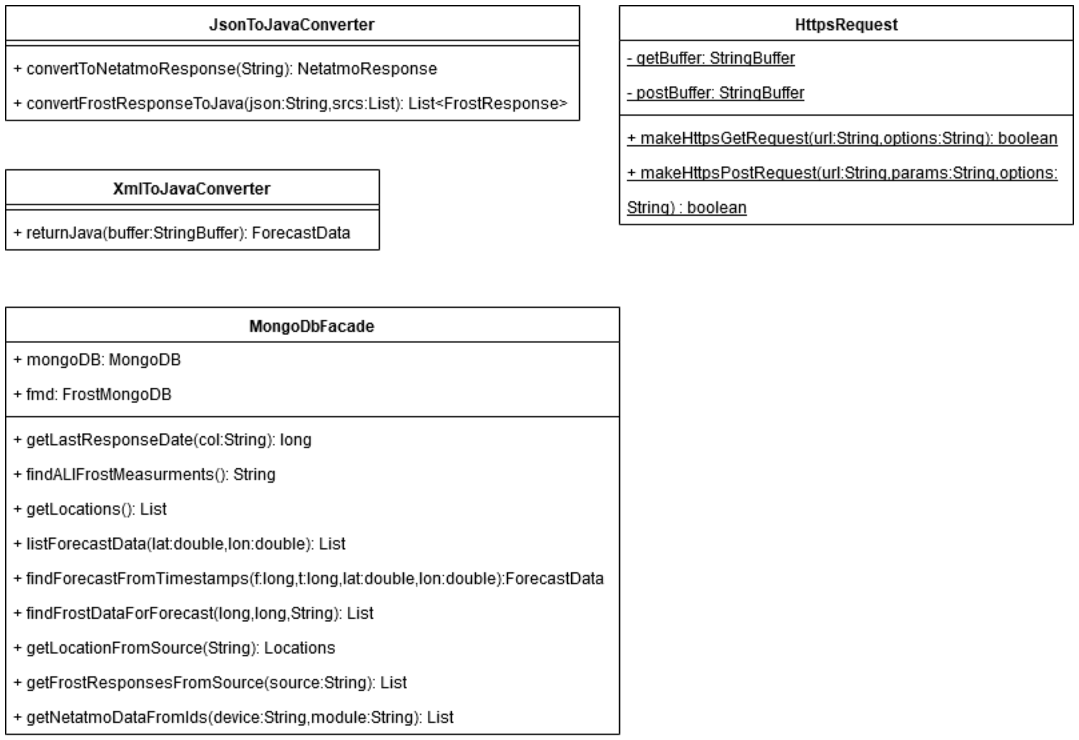
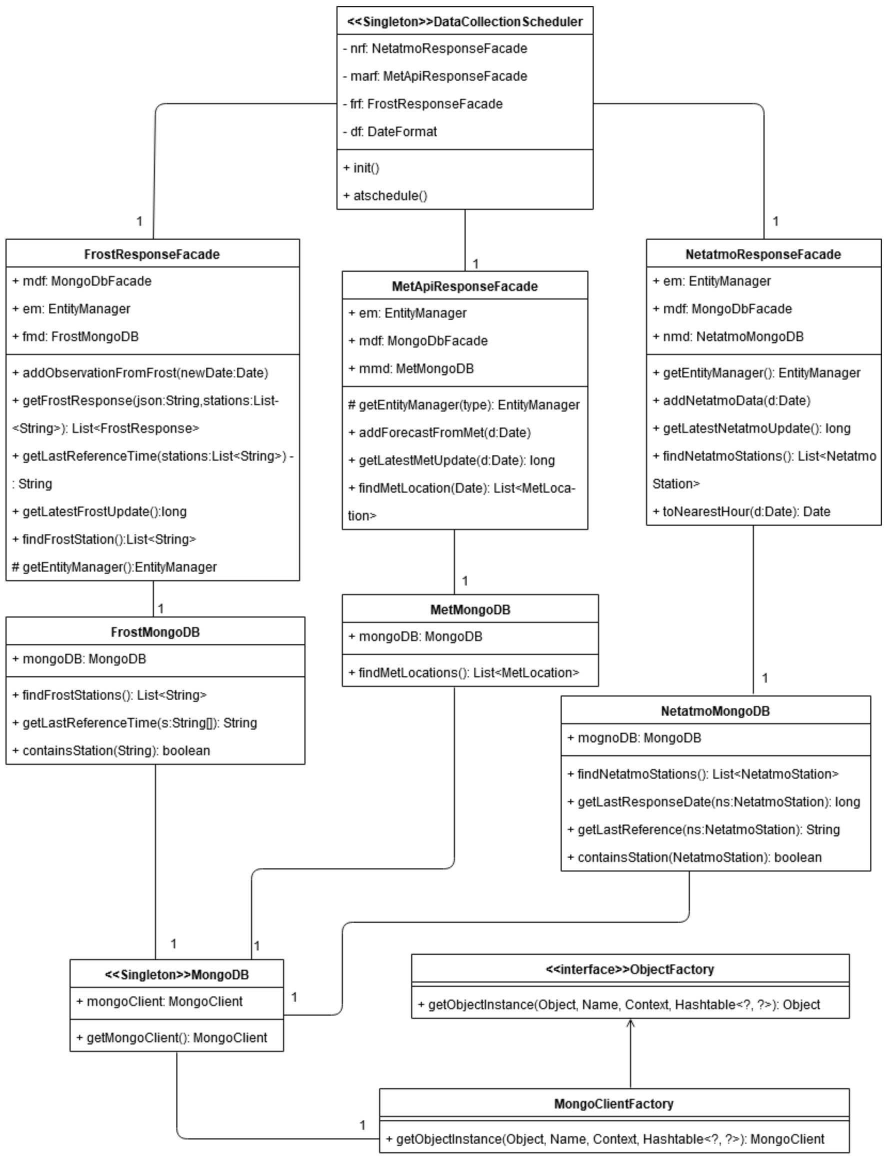

JsonToJavaConverter
+ convertToNetatmoResponse(String): NetatmoResponse
+ convertFrostResponseToJava(json:String,srcs:List): List<FrostResponse>
HttpsRequest
- getBuffer: StringBuffer
- postBuffer: StringBuffer
+ makeHttpsGetRequest(url:String,options:String): boolean
+ makeHttpsPostRequest(url:String,params:String,options: String) : boolean
XmlToJavaConverter
+ returnJava(buffer:StringBuffer): ForecastData
MongoDbFacade
+ mongoDB: MongoDB
+ fmd: FrostMongoDB
+ getLastResponseDate(col:String): long
+ findALlFrostMeasurments(): String
+ getLocations(): List
+ listForecastData(lat:double,lon:double): List
+ findForecastFromTimestamps(f:long,t:long,lat:double,lon:double):ForecastData
+ findFrostDataForForecast(long,long,String): List
+ getLocationFromSource(String): Locations
+ getFrostResponsesFromSource(source:String): List
+ getNetatmoDataFromIds(device:String,module:String): List

Figure 5.14: Class diagram of the utility classes used by the components of the application



<<Singleton>>DataCollectionScheduler
- nrf: NetatmoResponseFacade
- marf: MetApiResponseFacade
- frf: FrostResponseFacade
- df: DateFormat
+ init()
+ atschedule()
FrostResponseFacade
+ mdf: MongoDbFacade
+ em: EntityManager
+ fmd: FrostMongoDB
+ addObservationFromFrost(newDate:Date)
+ getFrostResponse(json:String,stations:List<String>): List<FrostResponse>
+ getLastReferenceTime(stations:List<String>) -: String
+ getLatestFrostUpdate():long
+ findFrostStation():List<String>
# getEntityManager():EntityManager
MetApiResponseFacade
+ em: EntityManager
+ mdf: MongoDbFacade
+ mmd: MetMongoDB
# getEntityManager(type): EntityManager
+ addForecastFromMet(d:Date)
+ getLatestMetUpdate(d:Date): long
+ findMetLocation(Date): List<MetLocation>
NetatmoResponseFacade
+ em: EntityManager
+ mdf: MongoDbFacade
+ nmd: NetatmoMongoDB
+ getEntityManager(): EntityManager
+ addNetatmoData(d:Date)
+ getLatestNetatmoUpdate(): long
+ findNetatmoStations(): List<Netatmo Station>
+ toNearestHour(d:Date): Date
FrostMongoDB
+ mongoDB: MongoDB
+ findFrostStations(): List<String>
+ getLastReferenceTime(s:String[]): String
+ containsStation(String): boolean
MetMongoDB
+ mongoDB: MongoDB
+ findMetLocations(): List<MetLocation>
NetatmoMongoDB
+ mognoDB: MongoDB
+ findNetatmoStations(): List<NetatmoStation>
+ getLastResponseDate(ns:NetatmoStation): long
+ getLastReference(ns:NetatmoStation): String
+ containsStation(NetatmoStation): boolean
<<Singleton>>MongoDB
+ mongoClient: MongoClient
+ getMongoClient(): MongoClient
<<interface>>ObjectFactory
+ getObjectInstance(Object, Name, Context, Hashtable<?, ?>): Object
MongoClientFactory
+ getObjectInstance(Object, Name, Context, Hashtable<?, ?>): MongoClient
1

Figure 5.15: Class structure that makes up the DHC component

Figure 5.16 displays the primary classes of the fire risk prediction component. It is made up of several of the same classes shown in figures 5.13 and 5.14. The most important class of this component is the Fire Risk Result class that has the functionality of calculating the fire risks. It uses the same classes as shown in figure 5.15 to set up the Mongo Database instance. The observation class in figure 5.16 is used to convert the weather data to the same common format, that is supplied to the Fire Risk Result Class.

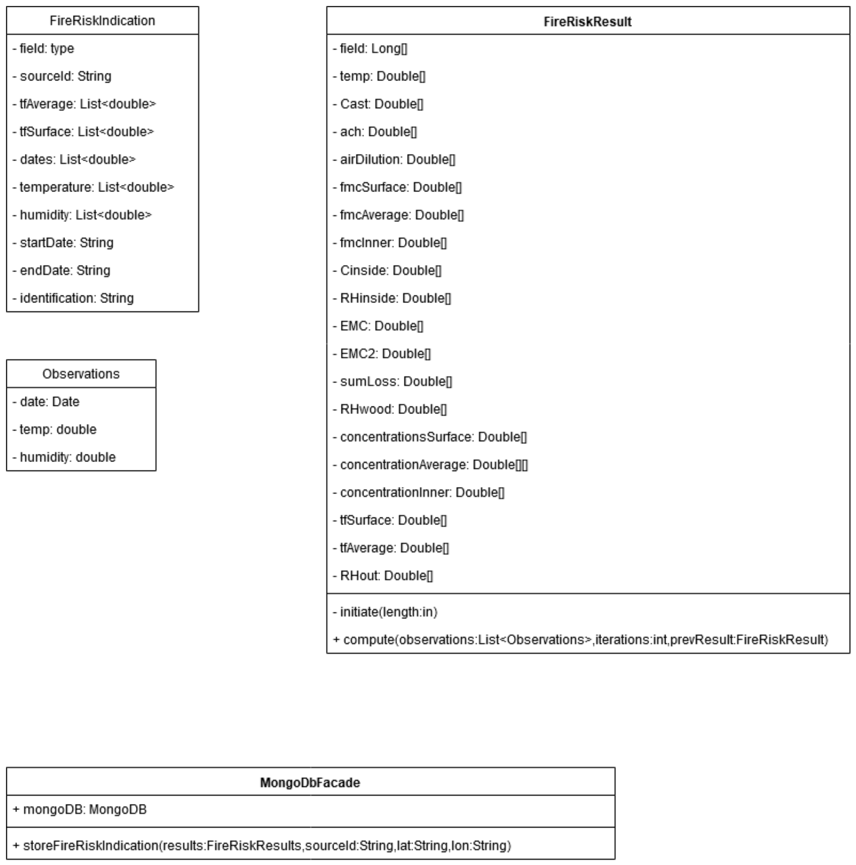


Figure 5.16: Class structure that makes up the fire risk prediction component

5.3.4 Program Flow

The program flow is split in two parts, one that handles interaction with the end users and one that handles validation of the mathematical model.

End User Program Flow

As discussed in the Section 5.3.1 on the use cases, there are several ways to operate the FR application. Figure 5.17 illustrates how a new location, to be used for continuous harvesting, will be added to the location database. It starts with a user that types a location that should be added to the database. The controller service will then communicate with the DHC with a HTTP POST request with a latitude and longitude parameters. The DHC will then make a request to the Frost service and find the nearest source for that location. Then the source identification and the coordinates of the location will be added to the location database, both in the SQl and noSQL database.

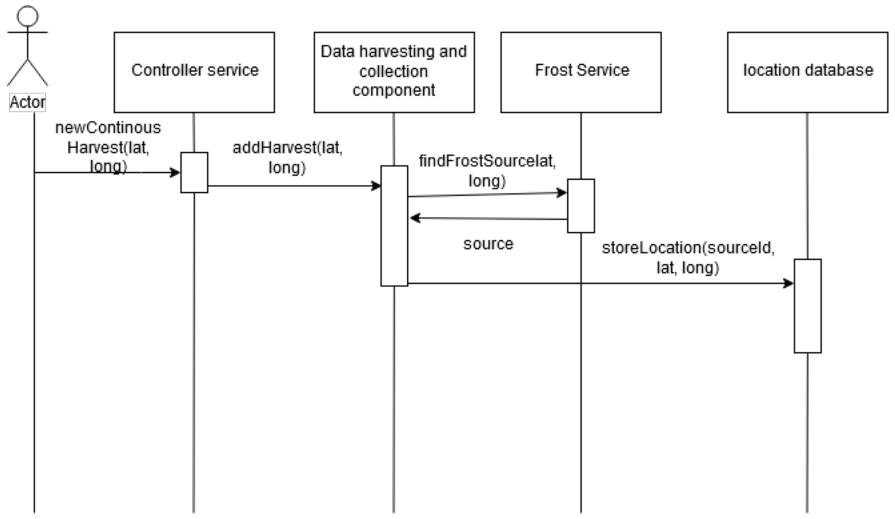

Actor
Controller service
Data harvesting and collection component
Frost Service
location database
newContinous Harvest(lat, long)
addHarvest(lat, long)
findFrostSourcelat, long)
source
storeLocation(sourceId, lat, long)

Figure 5.17: Sequence diagram for adding new continuous harvest

Figure 5.18 illustrates the sequence the FR application goes through for fetching weather data and creating fire risk indication by using only forecast data. The user starts by providing the longitude and latitude coordinates for the location. The fire risk web service will then transmit the request to the FRP component that it wants a fire risk indication using forecast data. Since the FRP does not have any forecast data available, it will ask the DHC component if it can retrieve the data needed. Then the DHC component will make a request to the MET API in order to retrieve forecast data for that location. When the DHC components receives the data, it will convert and filter it and send it back to the FRP component. The FRP component will then create a fire risk indication based on the weather data it received from the DHC component and send it back to the fire risk web service, which in turn gives it back to the client.

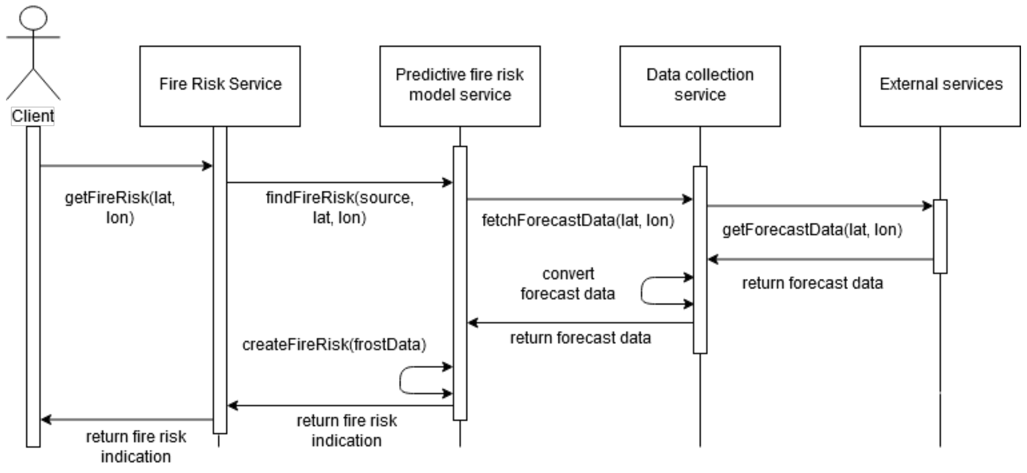

Client
Fire Risk Service
Predictive fire risk model service
Data collection service
External services
getFireRisk(lat, lon)
findFireRisk(source, lat, lon)
fetchForecastData(lat, lon)
getForecastData(lat, lon)
return forecast data
convert forecast data
return forecast data
createFireRisk(frostData)
return fire risk indication
return fire risk indication

Figure 5.18: Sequence diagram for creating fire risk indication with forecast data

Figure 5.19 shows the necessary steps fore creating a fire risk indication with using only historical data. The user must still provide the coordinates for the location of the fire risk indication, but also add the date range. As mentioned previously, the date range in this case can be between two dates in the past or from a date in the past to present date. The fire risk web service passes this information on to the FRP component which in turn ask the DHC component for the relevant weather data. The DHC component must first find the sources located nearest to the coordinates, and then use the closest station to find the weather data within the given time frame. The weather data is then converted and filtered and sent back to the FRP component and the fire risk indication for the weather data is computed. It is then sent back to the fire risk web service, and then passed on to the user.

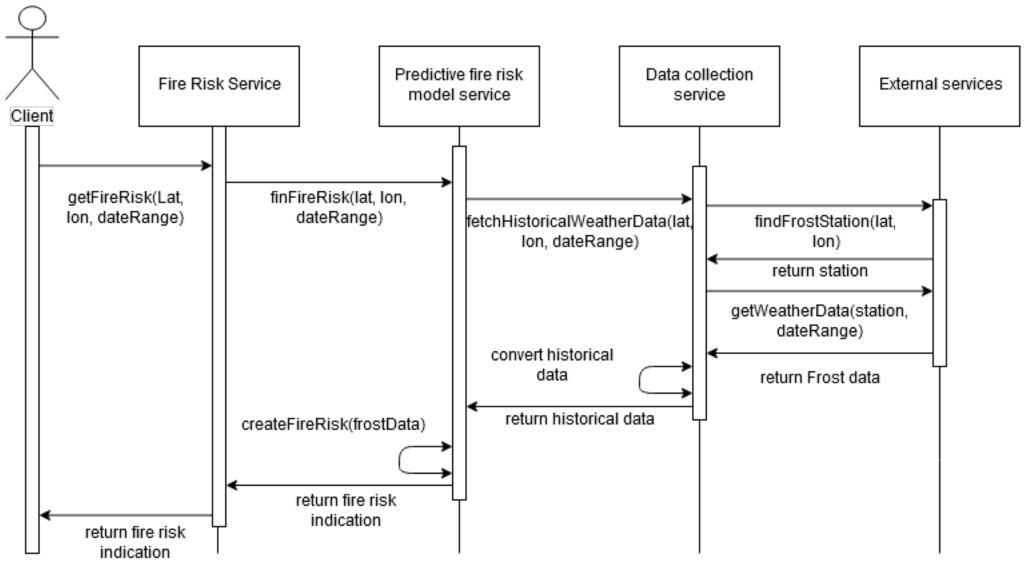

Client
Fire Risk Service
Predictive fire risk model service
Data collection service
External services
getFireRisk(Lat, lon, dateRange)
finFireRisk(lat, lon, dateRange)
fetchHistoricalWeatherData(lat, lon, dateRange)
findFrostStation(lat, lon)
return station
getWeatherData(station, dateRange)
return Frost data
convert historical data
return historical data
createFireRisk(frostData)
return fire risk indication
return fire risk indication

Figure 5.19: Sequence diagram for creating fire risk indications with only historical data

Regarding the sue of both historical and forecast data, the process is mostly the same as above. Figure 5.20 presents the process which shows how it combines the steps from both Figure 5.18 and Figure 5.19.

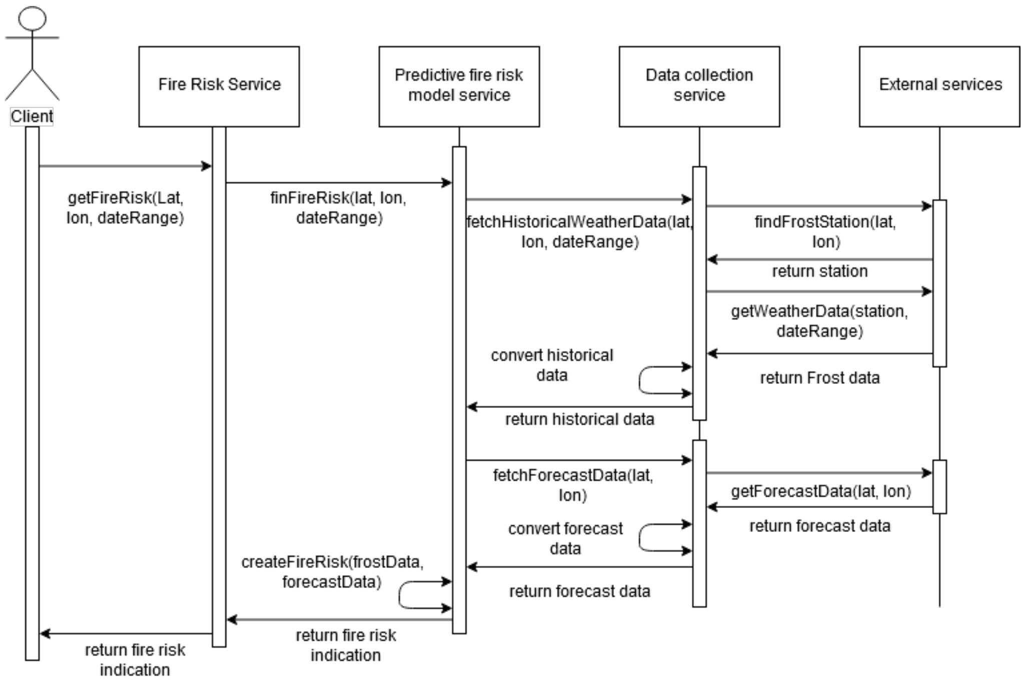

Client
Fire Risk Service
Predictive fire risk model service
Data collection service
External services
getFireRisk(Lat, lon, dateRange)
finFireRisk(lat, lon, dateRange)
fetchHistoricalWeatherData(lat, lon, dateRange)
findFrostStation(lat, lon)
return station
getWeatherData(station, dateRange)
return Frost data
convert historical data
return historical data
fetchForecastData(lat, lon)
getForecastData(lat, lon)
return forecast data
convert forecast data
return forecast data
createFireRisk(frostData, forecastData)
return fire risk indication
return fire risk indication

Figure 5.20: Sequence diagram for creating fire risk indication with historical data and forecast data

Figure 5.21 presents the interaction between the components when the (DHC) component harvest new weather data. The harvesting is scheduled to occur every 24 hours every day of the week. When the schedule is triggered, the data harvesting component communicates with the location database in order to find which locations it should find weather data for. Then it makes a request to the services to find the weather data required. The DHC component can then use a PUT method in the FRP component service in order to push the data. At this point the FRP component service can create fire risk and store it in the fire risk database.

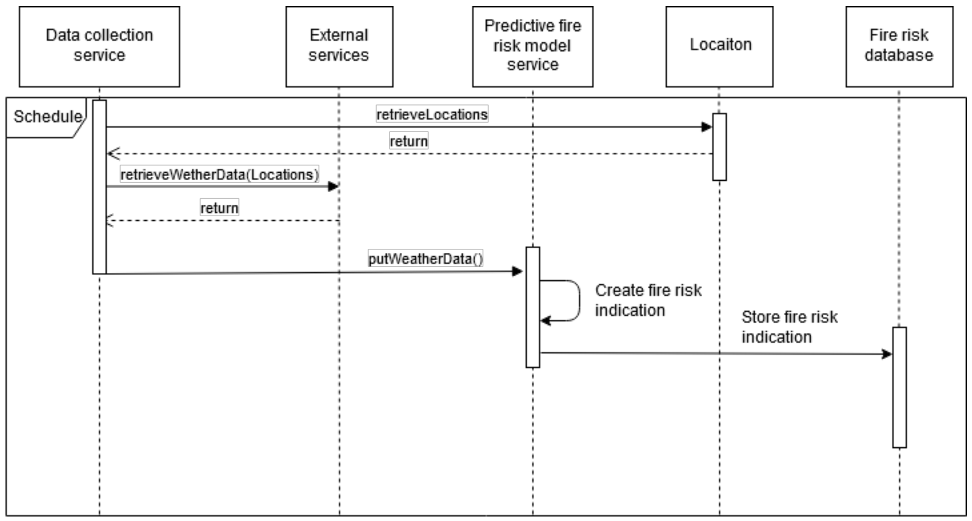


Figure 5.21: Sequence diagram for continuous data harvesting

Validation Program Flow

The program flow for the validation works with the data that is collected in the winter period 2018 - 2019. There are several aspects that are validated such as comparing the fire risk indication of a forecast with the corresponding fire risk indication using historical weather data. It also enables comparison of the Netatmo stations with the stations from MET.

The figures below presents the interaction between the components for retrieving the required weather data stored in the database for comparison. These interactions are nearly identical to that of the sequence diagrams presented above, the only difference is that instead of using the external services as a source for the weather data it uses data stored in the database.

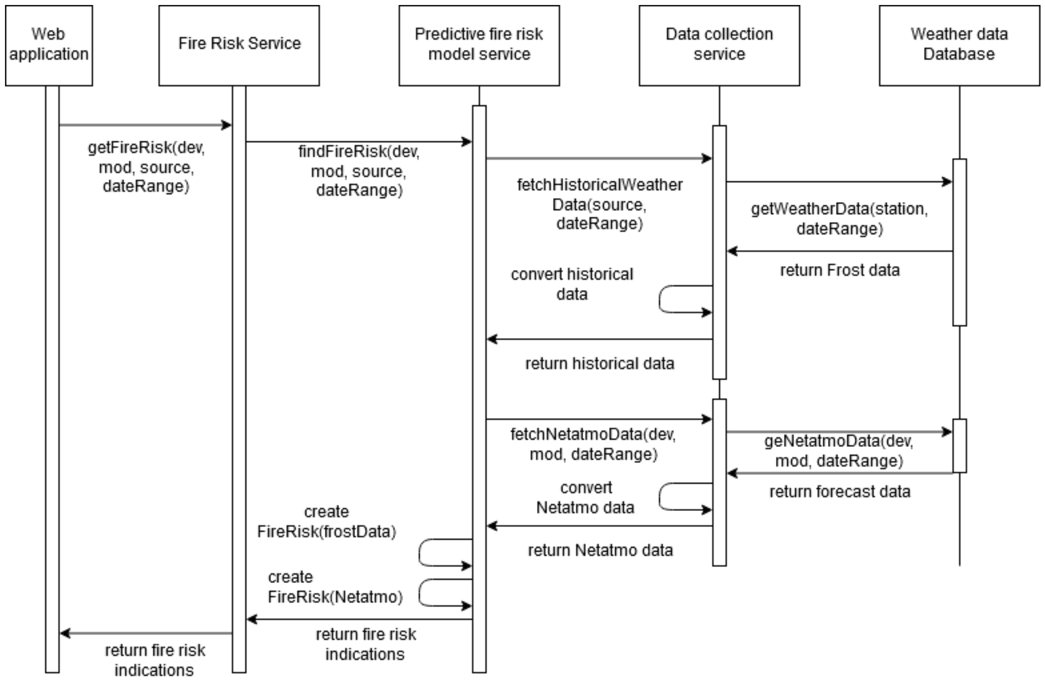

Web application
Fire Risk Service
Predictive fire risk model service
Data collection service
Weather data Database
getFireRisk(dev, mod, source, dateRange)
findFireRisk(dev, mod, source, dateRange)
fetchHistoricalWeather Data(source, dateRange)
getWeatherData(station, dateRange)
return Frost data
convert historical data
return historical data
fetchNetatmoData(dev, mod, dateRange)
geNetatmoData(dev, mod, dateRange)
return forecast data
convert Netatmo data
return Netatmo data
create FireRisk(frostData)
create FireRisk(Netatmo)
return fire risk indications
return fire risk indications

Figure 5.22: Sequence diagram for retrieving and comparing the difference in the fire risk calculations between Met and Netatmo stations

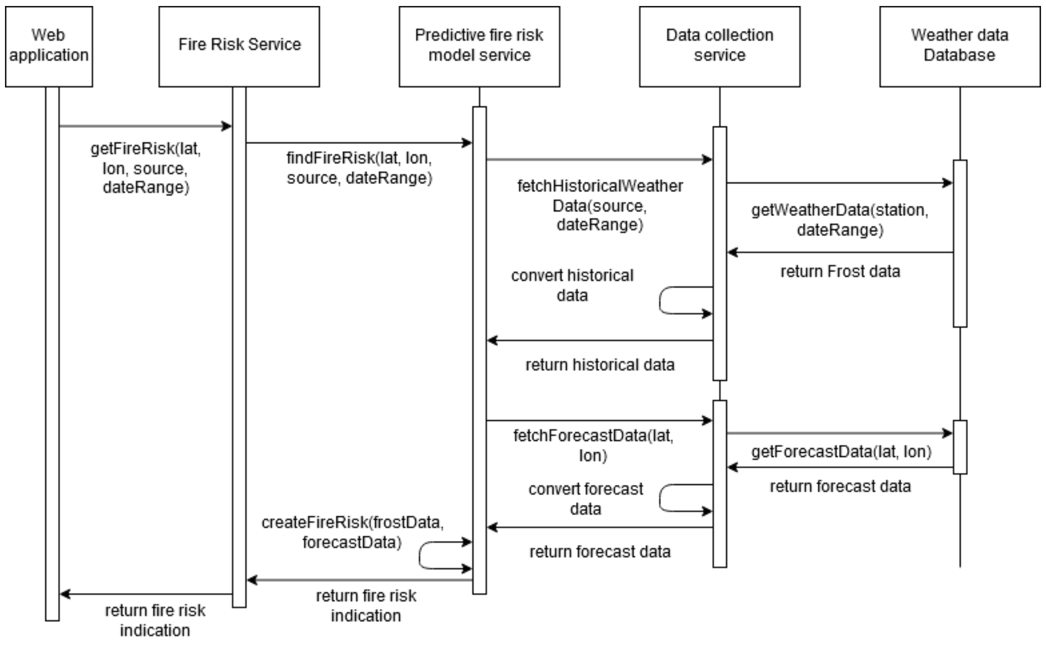

Web application
Fire Risk Service
Predictive fire risk model service
Data collection service
Weather data Database
getFireRisk(lat, lon, source, dateRange)
findFireRisk(lat, lon, source, dateRange)
fetchHistoricalWeatherData(source, dateRange)
getWeatherData(station, dateRange)
return Frost data
convert historical data
return historical data
fetchForecastData(lat, lon)
getForecastData(lat, lon)
return forecast data
convert forecast data
createFireRisk(frostData, forecastData)
return forecast data
return fire risk indication
return fire risk indication

Figure 5.23: Sequence diagram for retrieving and comparing the difference in the fire risk calculations between Met stations and forecast data

5.4 Deployment

The application is deployed on a server hosted on the Amazon web service platform. The virtual machine used for hosting is running the 2018 edition of the Windows Server operating System. Using this virtual machine gives the ability to access the application from anywhere, and we are not constrained by using local machines. It also gives the opportunity to verify that the application works in a deployment environment. The virtual machine runs a TomEE[23] web server, which hosts the different components of the application. The reason for using the TomEE web server and not a light weight web server such as Tomcat is that the component for collecting weather data uses database connections that are not supported by the Tomcat web server.

The virtual machine needed some setup before it could be used. This had mainly to do with the firewall for the virtual machine, which had to be configured in order for outside communication to work. On the virtual machine, the port 8080 is used for communication. In order for any outside parties to connect to the web server on the virtual machine, this port had to be opened in the firewall. The Amazon web service platform has a secondary firewall outside the virtual machine, which also had to be opened in order for port 8080 to work.

The FR application uses two types of databases, where one of them is a noSQL database and the other is a standard relational database. The relational database is hosted on a server run by the Western University of Applied Science. The noSQL database is provided by MongoDB and is hosted on the Microsoft Azure cloud platform. The databases also had ports that were closed by the virtual machine and these were also opened in both of the firewalls.

The different components of the application also uses Apache Maven[24] as a build automation system. Maven keeps track of the necessary dependencies of third party libraries and the structure of the application. By using Maven, it is possible to make the application independent of what IDE is used when developing as Maven stores the structure and the dependencies in what is called the POM-file. Most IDEs can read this file and then automatically create the work-space and download the necessary dependencies referenced in the POM-file.

Part of the reasoning behind using TomEE server instead of Tomcat server, is that TomEE offers different dependencies that do not require to download them with Maven. However, there are some dependencies that TomEE does not have available, but these are possible to get through Maven. Maven will then automatically add these libraries to the TomEE web server in order to enable the application to work properly. Whenever the web server starts, it reads the POM-file of the application and it analyses which of the libraries are provided with TomEE and which it does not provide. When the web server finds a library it does not provide, the web server will download it from the Maven repository and add it to the library folder of the web server which the application can then make use of.

Chapter 6

Evaluation

During the winter period 2018 - 2019 weather data has been collected at four different locations to get an overview of how the fire risk evolved during this period. The locations are Bergen, Haugesund, Gjøvik, and Lærdal. The collected data includes weather data from Netatmo stations, placed at two locations, MET stations, and weather forecasts. Part of the evaluation has been to investigate how the weather data from the different sources differs from each other and how it impacts the computations of the fire risk indication.

This chapter presents the results from the evaluation of the collected data by giving an overview of how the fire risk evolved at the four location that was focused on. It also includes the results of the difference between calculating the fire risk with Netatmo stations versus MET stations. Another result that is presented is the difference in a fire risk calculated from forecast weather data versus historical weather data collected in the same time period. There is also the case of looking at historical fires to see how the FRI model would have indicated the fire risk at the time of fire, and some days in the past leading up to the fire. Finally the results regarding the run-time and storage efficiency is explored.

6.1 Historical Weather Data

Four places were chosen as a means to measure the fire risk for places around Norway. Two of the locations were at the west coast (Bergen, Haugesund), and the two other were inland locations (Lærdal, Gjøvik). The reason for choosing these locations is due to the varied climate. At the coast it will be a lot more humid in the winter than in some of the inland locations which may also be a lot colder during the winter. Since this is the case, the FRI model should give a higher fire risk for places that are significantly colder than that of the coastal cities that may have a warmer climate.

Location	avg	std.div	max	min
Bergen	5.50	0.67	7.64	4.13
Haugesund	5.70	0.63	7.589	4.32
Gjøvik	4.48	0.90	8.02	3.32
Lærdal	4.77	0.74	7.57	3.56

Table 6.1: Fire risk information from four locations

Figure 6.1 explores this by visualizing the collected weather data in the winter period 2018 - 2019. From the graph it can be seen that the FRI model does precisely what it should do by reporting an expected higher risk of fire because of the colder climate. Recall that a lower time to flash-over is what indicates a higher fire risk. This can also be verified by table 6.1 where it indicates that the coastal cities have a higher average time to flash over which means the overall fire risk is lower than that of the inner cities. The two lines at the top of 6.1, illustrates what the the time to flash over (Tf0) would be at 50 % and 60 % relative humidity. As the weather gets colder, the climate gets drier which leads to wood panels releasing humidity to the environment. This phenomenon was shown by the equilibrium moisture content in wood at certain humidity levels discussed in chapter 3.

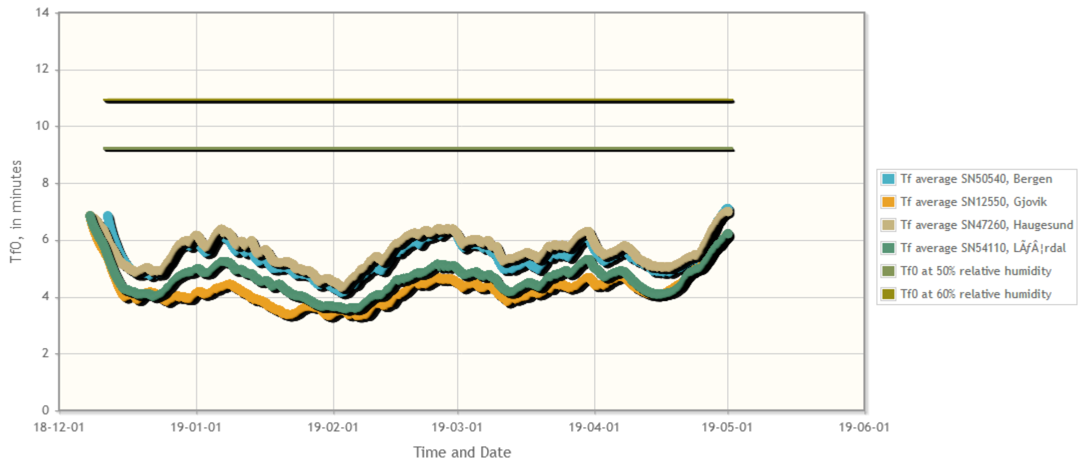


Figure 6.1: Fire risk indication of the four location

6.2 Historical and Forecast Weather Data

Being able to predict the fire risk within the next few days is of huge benefit for anyone involved in protecting against fires. Therefore, one of the validation elements was to see how it would fair against historical weather data. Since there exists no archive of forecast data, an important aspects of the winter 2018 - 2019 weather data collection was to store the forecast. This gives the possibility to validate it with the same measured weather data from reliable sources such as MET. Another aspect to consider is that the FRI model requires a few days of self-calibration before it can accurately begin to indicate the fire risk. If it does not have any previous fire risk indications to work with, the first few days of the forecast data is used for calibrating which may end up give inaccurate readings as it starts at a middle point and works towards the correct fire risk. Therefore we investigated how to use historical weather data and add it onto the forecast data in a way that the historical weather data was used to calibrate the fire risk indication. This means that it should start indicating the fire risk at a more accurate point when it comes to the forecast data. In order to have daily information, forecast data was stored every 24 hour. The reason for storing this is because forecast data is not available for retrieving when a new one has been calculated by METs services. Since the fire risk indication is not stored in a database, the fire risk indication is computed every time it is requested, by fetching the related data from the weather data database.

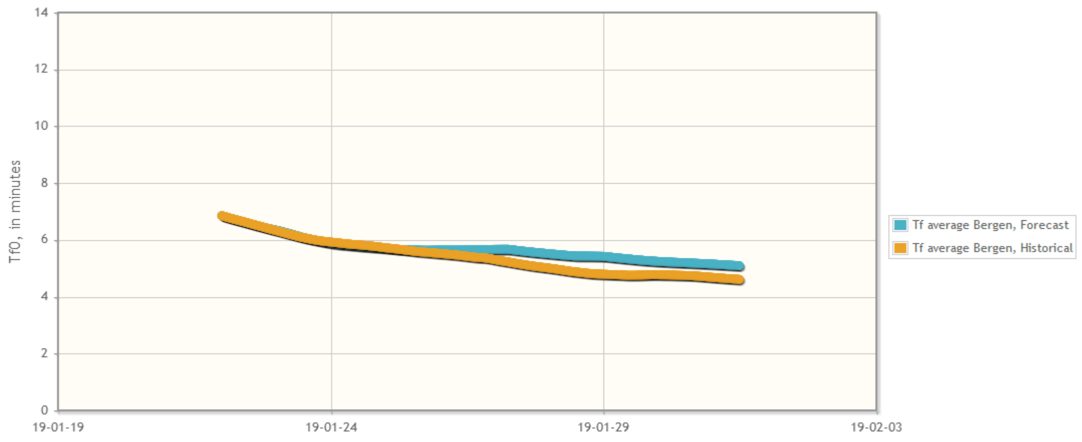


Figure 6.2: Fire risk indication difference between forecast and historical weather data, without calibration

Figure 6.2 indicates this process where historical data has not been added to the forecast data used to create the fire risk indication. The corresponding fire risk indication by only using historical data is also visualized to show the difference. In figure 6.3 historical weather data is added on to the forecast data, which is used for the calibration. As can bee seen from the figures, the fire risk from the forecast follows roughly the same curve as the fire risk indication from historical data in the first three and a half days. In this period of time, the forecast works with weather data at hourly measures. After that period it starts giving measures at a six hour interval. When it starts the six hour interval, the FRI model has less data to work with and it has to model what it predicts the fire risk will be until the next measure in six hours. For the most part it is within reasonably limits to that of the historical fire risk, but there are certain points where it starts varying a little more, but it still follows the same curve as the historical fire risk indication.

Following the information from Table 6.2, the visualization from figure 6.2 and figure 6.3 has been quantified in order to give a better understanding of the differences. As Table 6.2 indicates, the average difference between only historical and forecast fire risk indication when additional data has not been added for self-calibration is estimated to around $\pm 0.23 minutes$. The standard deviation for the difference is at ± 0.21. The maximum difference between forecast and historical when calibrated is 0.58 and when not calibrated is 0.62.

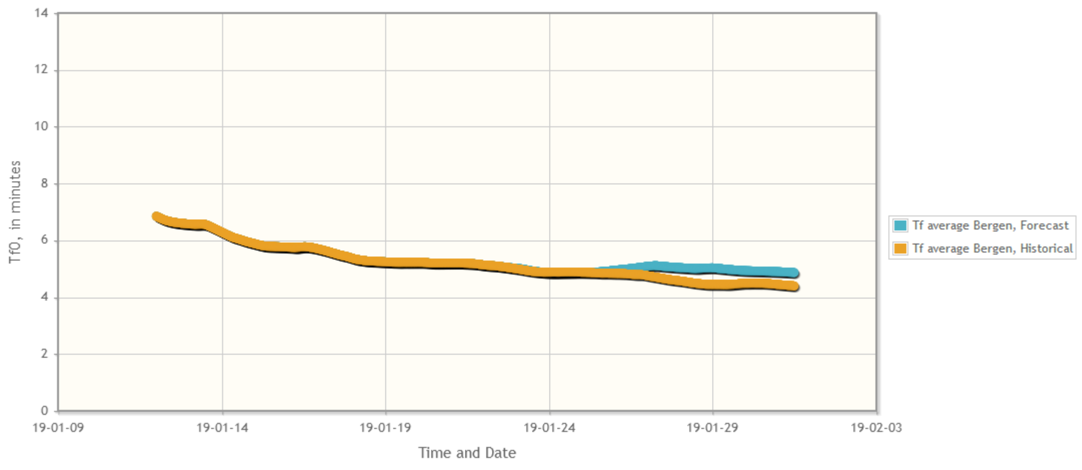


Figure 6.3: Fire risk indication with forecast and historical, with calibration

Location	avg diff	std.div	max diff
Bergen (no calibration)	0.26	0.24	0.63
Bergen (calibration)	0.12	0.18	0.58

Table 6.2: Fire risk information from four locations

6.3 MET stations and Netatmo stations

One goal of this thesis has been to investigate how the use of different sources impacts the indication of the fire risk. The previous sections discussed the result of using stations provided by the Norwegian Meteorological Institute, who operates advanced historical measuring stations. It was therefore important to explore sources that are more readily available and can be placed wherever is needed. The sources in question are the Netatmo stations. These stations are placed closer to the houses than that of the sources from MET. As has been seen in Log's research[1], it was found that with calibration they measured within reasonable margins compared to that of the stations from MET.

Figures 6.4 and 6.5 shows the difference between using a calibrated Netatmo station and MET station for two locations. As can be seen in Figure 6.5, the station Netatmo station in Gjøvik follows almost the exact same curve as the one based on the MET station. This indicates that the Netatmo station is calibrated correctly since it is able to follow the MET stations measures.

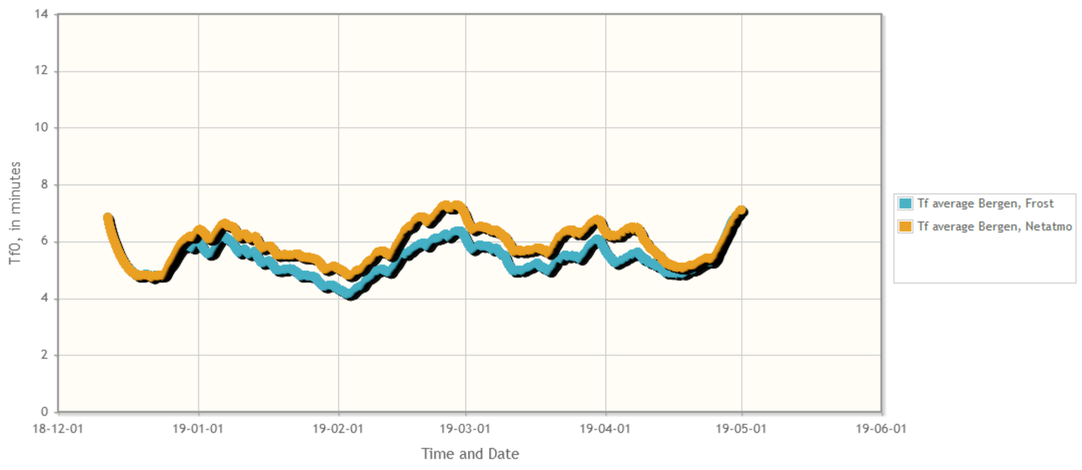


Figure 6.4: Difference between MET station and calibrated Netatmo station in Bergen

In Figure 6.4, the Netatmo fire risk indication from Bergen is not exactly the same as the one from MET. The curve itself follows almost to the point of what the MET fire risk indicates, which tells us that either the Netatmo station is not correctly calibrated and measures temperatures higher then the MET, station or it may be that the Netatmo station is placed at a location that has a different climate than that of the Met station.

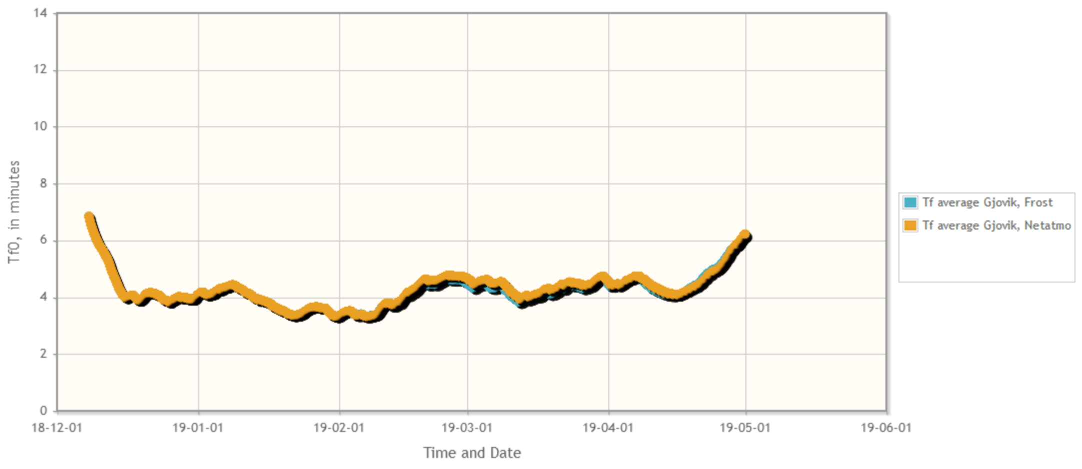


Figure 6.5: Difference between MET station and calibrated Netatmo station in Gjøvik

Table 6.3 shows the numerical differences between the us of Netatmo stations and MET stations when computing the fire risk. At Gjøvik, the difference between the Netatmo stations and the Frost stations is almost negligible, whereas the result from Netatmo and Frost Bergen is more varied for certain periods. Still the overall difference is 0.5 minutes on average, and at most 1 minute.

Location	avg.diff	std.div	Max diff	Max Tf0(F)	Min Tf0	Max Tf0(N)	Min Tf0
Bergen	0.53	0.28	1.06	7.09	4.13	7.28	4.72
Gjøvik	0.07	0.07	019	6.14	3.32	7.25	3.31

Table 6.3: Difference between MET stations an Netatmo stations for selected locations

The figures below illustrates the difference in fire risk indication when using a non calibrated Netatmo station versus a MET station. As can be seen between figures 6.5 and 6.7 it is slightly more inaccurate when using a non calibrated Netatmo station. However, when looking at the difference between figures 6.4 and 6.6 is is almost impossible to notice the difference. By looking at tables 6.3 and 6.4 the difference between a non calibrated and a calibrated Netatmo station in Bergen is minimal, whereas the difference in Gjøvik is a lot more noticeable where the average difference goes from 0.07 to 0.35. The reason for the minimal difference in Bergen is the fact that originally the station measured humidity closely to that of the known humidity of salts during the calibration process. Because of this, the calibration curve does minimal changes to the measured humidity. The measured humidity during the calibration process can be seen in Apendix A. The station from Bergen is the one listed as stasjon 11, and the one in Gjøvik is listed as stasjon 15.

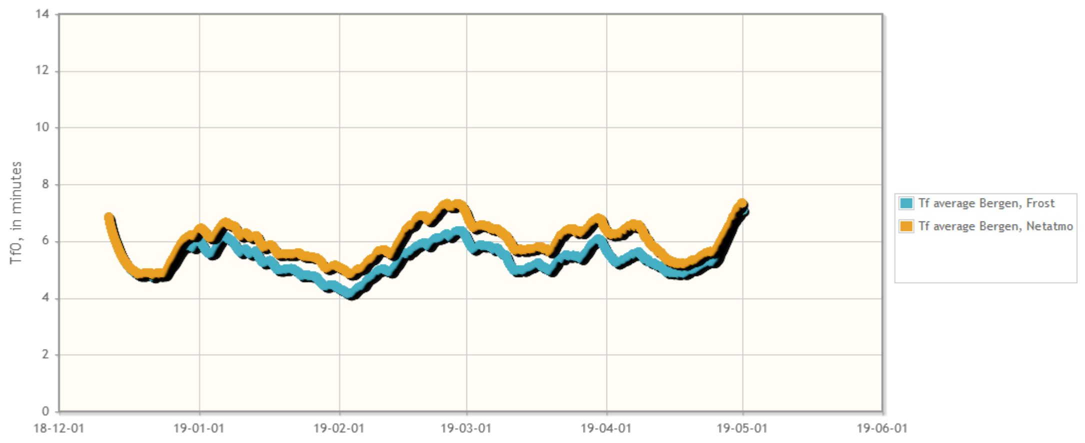


Figure 6.6: Difference between MET station and a non calibrated Netatmo station in Bergen

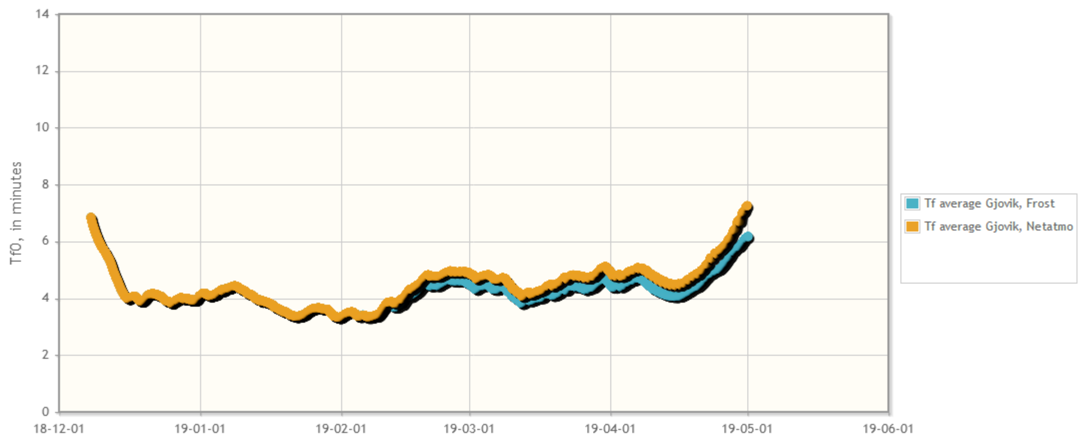


Figure 6.7: Difference between MET station and non calibrated Netatmo station in Gjøvik

Location	avg.diff	std.div	Max diff	Max Tf0(F)	Min Tf0	Max Tf0(N)	Min Tf0
Bergen	0.59	0.26	1.10	7.09	4.13	7.33	4.81
Gjøvik	0.35	0.17	1.10	6.14	3.32	7.25	3.34

Table 6.4: Difference between MET and non calibrated Netatmo stations for selected locations

Another possible reason for the differences may be the placement of the stations. For instance, the Netatmo station in Gjøvik is placed approximately 6.017km north west of the MET stations. The Netamtmo station in Bergen is placed around 1.76km directly north east of the MET station. The MET station in Bergen is located close to the water with an open fields surrounding the station. The Netatmo station is located in a densely populated area with lots of buildings surrounding it. It is also relatively close to the water but not as close as the MET stations. The station in Gjøvik is located at opposite sides of the lake Mjøsa, where the Netatmo station is located at a house in Gjøvik and the MET station is located in Nedre Kise. Based on this information one could come to the conclusion that the stations in Bergen would have the least difference between the fire risk indication. However, as Table 6.3 and Figures 6.5 and 6.4 indicate, this is not the case. A reason that the stations in Gjøvik have a closer result may be that, since they both are located closely to Mjøsa the climate at both sides is closely related to each other. In Bergen, which is a lot more densely populated than Gjøvik, the micro climate may vary to a greater deal at closer distances than what it does in Gjøvik.

6.4 Indicating Fire Risk for Historical Fires

Another aspect in terms of validating the FRI model, is to consider fires in the past. This way it is possible to determine how the fire risk was at the time of the fire, and the period leading up to the fire. Then we can look at the same period in the previous years and the years after to see if this in genera is a period of very high fire risk, or if it was just normal conditions for that specific period.

A recent fire that will serve as an example, is the one in Lærdalsøyri on 18th January 2014[26]. This is a place with many old wooden buildings, and is located in a place that gets very dry during the winter period. The fire risk estimated for that time is visualized in Figure 6.8. During a period of around 12 days before the fire, the temperature started dropping which results in the climate getting drier. In this dry period, the wood inside the houses released humidity to the surrounding area. At the time of the fire at around 22:50 the FRI model indicates that it would take around 3.8 minutes until complete flash over. The fire department learnt about the fire at 22:53pm[26], and the fire fire truck was on scene at 22:59pm[26]. At this time it was reported that the house was in complete flash over. Since the FRI model indicate a complete flash over in 3.8 minutes, the fire department did not have sufficient time to put out the fire. It should also be noted that at the time, there was heavy winds in the area[26] which would also contributes to higher risk of fire conflagration. The FRI model does not take this aspect into consideration when indicating fire risk. In the case of the Lærdal fire, it spread quickly to the surrounding houses because of the wind. And these houses would have had equally high fire risk.

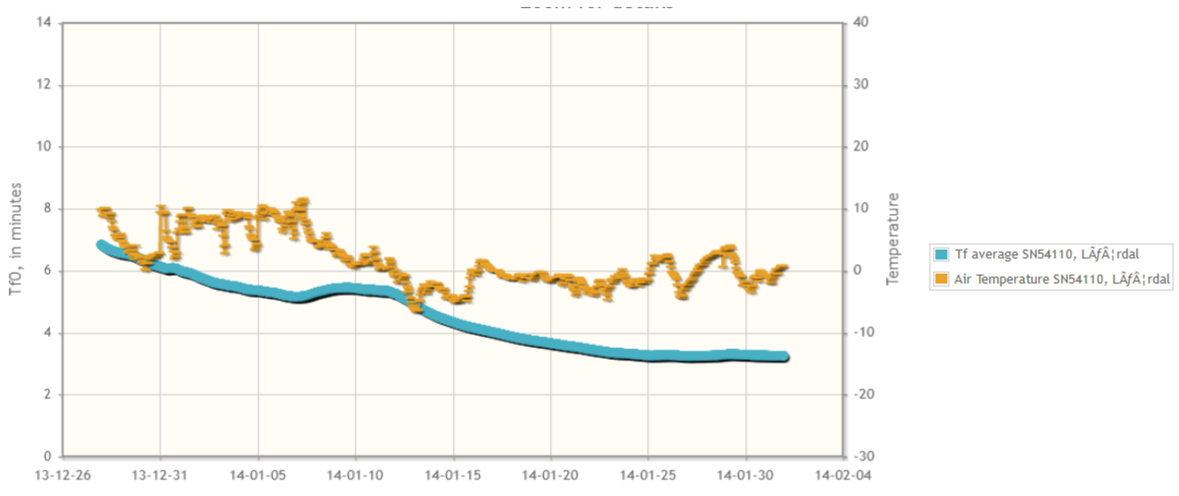


Figure 6.8: Fire risk for the fire in Lærdal 18 January 2014

Another fire to consider is the one in Kongsberg, at 24th December 2017, at a home care center[27]. The fire happened in the night of Christmas eve and resulted in the loss of life. The fire risk indication for this period is visualized in Figure 6.9. During the December month of that year the time to flash over average around 4.4 minutes. The result indicates the same as the fire risk from the fire in Lærdal of 2014, that the time it takes for a complete flash over is considerably lower than that of the required response time from the fire department. Since this is a home care center, the fire department has special laws, section 4-8 of Norwegian law, that states the required response time is to be 10 minutes or less.

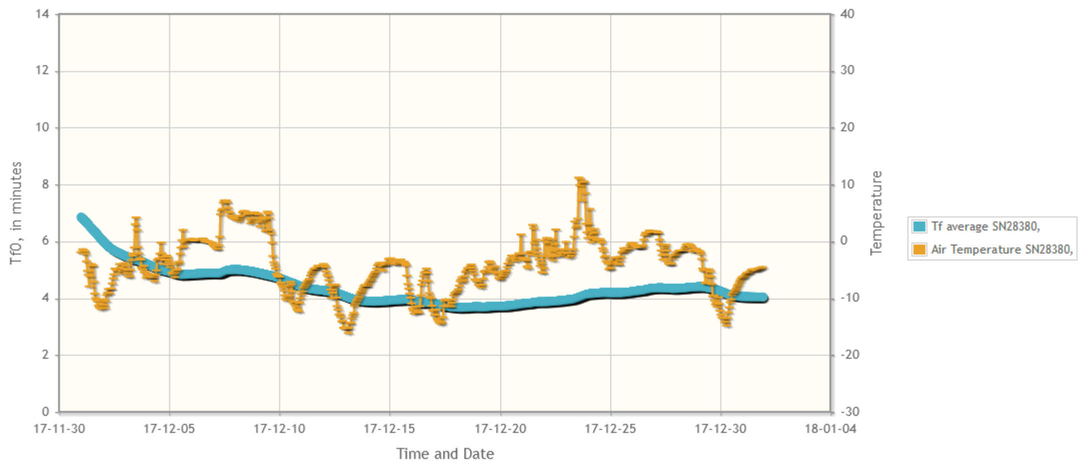


Figure 6.9: Fire at a home care center in Kongsberg 2017

Given these results, the FRI model could have warned the fire department to be readily available at this period in time and it would also seem that the required response time does not take into consideration dry periods. Based on these results, the FRI model could have predicted that the fire department was not ready for a fire that would flash over that quickly. The current minimum response time that the fire department is expected to keep is way too high for the dry periods of winter. Given the outdoor air temperature in these periods of time, the FRI model will always give a low time to flash over in the period from December through May.

6.5 Discussion of Storage Efficiency

As of now, the FR application has collected approximately five moths worth of weather data. This includes forecast data and historical data from four locations. It also includes weather data from two Netatmo stations placed at two of the four places.

Each weather forecast stored in the database has a list of 87 objects containing weather information, such as temperature, humidity. There are currently a total of 504 of these stored in a database. Some days are missing due to issues early in the development of the FR application where it would not store the forecasts correctly. It should be noted that a lot of data has been filtered out so that the storage mostly contains data that can be used. The total amount of storage that these forecasts use, amounts to 12.5Mb, with an average of around

25.4 kb. per forecasts.

The weather data from the Frost stations are stored in 24 hour intervals and contains hourly recorded weather elements, mostly the same types as the forecast. Currently, weather data is stored for each day since the collection started, but some of the stations started later than the others. As with the forecast data, unnecessary data have been filtered and thus minimal data have been stored in the database. The collection in the database that stores historical data contains 634 documents, each of these documents contains 24 hours worth of weather data. The total amount of storage is 5.6 Mb of storage with an average of 9.0 kb per document.

As of now there are only two Netatmo stations in use, where the one in Bergen started collecting weather data in December of 2019. The station in Gjøvik started around February 2019. The measurements from the Netatmo stations are stored in the same way as the Frost stations, where each document contains 24 hours worth of weather data. It takes substantial less storage space than the other two. This is because it contains less information as it only have humidity and temperature, where the forecast and MET stations have a lot more weather elements attached. It is also filtered by the Netatmo web service in order to minimize variables attached to the request when retrieving it. With this in mind, the total number of documents containing weather data from Netatmo totals at 336.7 kb of storage with a average of 1.3 kb per document.

The FR application performs continuous harvest of weather data every 24 hour. This amounts to all the data that is currently stored in the databases of the FR application. This is mainly for evaluation purposes as the data has to be consistent throughout the evaluation of the FRI model. Every 24 hours, the FR application fetches historical weather data for the previous day, from MET and Netatmo, and forecasts for the next nine and half days. These are stored in the database for future use. This will not be the case when the FR application is deployed in production and no longer used for evaluation. At that point, the only necessary data that has to be stored is the fire risk indications, and maybe the forecast data for the location. Whenever the FR application fetches new historical weather data, it will take the previously calculated fire risk indication and create an augmented fire risk indication for the new weather data, and add it to the back of the previous one. By doing it this way, the storage efficiency depends not on the weather data, but only on how many fire risk indications are stored.

A fire risk indication for a 24 hour period at one location uses 61.6 kb of storage. Given this information it is possible to calculate how much storage is needed when doing continuous fire risk calculations for several locations. For instance, if we have continuous fire risk calculations for 10 locations this will amount to 616 kb of fire risk indications every day. For a whole year this will total at 224.84 mb of storage. If there are now 100 locations, each with a separate weather station as source, the total amount of storage for a whole year is 2.24 gb.

The component that handles collection of weather data in the FR application used two databases: one SQL database and one noSQL database. The two databases are widely different in how they store the data. For the SQL databases, the weather data had to be converted from either XML or Json, depending on what the external service offered, to the format of the programming language used. The noSQL document database operates with documents of Json data, meaning that in theory if the data from the external service is provided in Json format it could be stored directly in the database without having to do any conversions. As we wanted as little data stored as possible, this was not the case and it had to be converted to the programming language and then back to JSON. Therefore the benefit of using the noSQL database when storing weather data is lost as it creates more steps for both storing and retrieving data back from the database. Because of this, it would almost be exactly the same as using a SQL database. However, for unknown reasons, the SQL database had problems halfway through the collection period where it stared to throw exceptions and would not store data in the database. Whenever the FR application attempted to communicate with the database the exception was thrown, and the collection component would have to be restarted. In an attempt to not corrupt or delete data from the database, the SQL database was disconnected and only the noSQL database was used for storing data.

Despite this fact, there would not be that much of a difference between using a noSQL database and a SQL database for storing weather data in the FR application as it would have to be converted in some form or another before dispatching it to other components.

6.6 Discussion of Run time Efficiency

In order to find the execution time of the steps it takes to create a fire risk indication, the Java method System.nanoTime() was used. In order to find the execution time, the nano time method is called right before and after the actions are executed to find the elapsed time.

With regard to runtime efficiency, it took 0.07 seconds to compute a fire risk indication for a full year. Note that this excludes the time it takes to retrieve the weather data from the external services and the time for converting the data. If all steps are included for creating a fire risk for a whole year, it takes 4.1 seconds to retrieve the weather data, another 0.2 seconds to convert it. Then it is passed on to the FRI component which add another 0.6 seconds for conversions and it takes 0.07 seconds to compute the fire risk. The total time elapsed for creating a fire risk indication with weather data for a full years amounts to 5 seconds.

If the same was done for half a year, the time is halved to 2.5 seconds. Where 2.36 seconds was used to fetch the data, and 0.04 seconds used for converting and computations. The rest of the time is spent communicating between the components.

It must also be specified that the time it took to create the fire risk indication was only calculated using the external services from MET. The reason for not using the Netatmo service is that it only returns a maximum of 1024 elements of data per request whereas Frost and the Met API returns all data with one request. This means that a total of 8 requests have to be made to the service in order to be able to elapsed time of a fire risk indication for a whole year.

With this in mind, the bottleneck of the FR application would not be local one, but rather related to the external services that supplies the weather data.

Chapter 7

Conclusion and Future Work

In this chapter we revisit the research questions of the thesis by linking the results obtained, and presented in the previous chapter to the research questions. We also outline areas where our work can be improved and provide directions for future development of the FR application.

7.1 Predictive mathematical model for fire risk indication

The primary research question, R1, aimed to answer whether the FRI model of Log[18] is sufficient enough to give a useful fire risk indication.

7.1.1 Conclusion

Based on the result of the fire in Lærdal and Kongsberg, it is possible to conclude that the FRI model can give accurate fire risk indication regarding how long it takes for the fire to be in complete flash over. Information gathered from the fire department in Bergen, stated that they had a minimum requirement of 10 minutes response time to certain buildings. This included hospitals, nursing homes, historical buildings and shopping centres. With the result regarding the Lærdal and Kongsberg fires, many of the fire departments around Norway would not have sufficient time to respond to fire when the time to flash over is low during the winter period.

Based on R1, two sub-questions where asked regarding the use of different types of weather data from different sources. In chapter 6 the result of combining historical weather data and forecast weather data were discussed. The research question were related to investigating whether it would be possible to use only forecaste weather data as a means to create a fire risk indication, or would the result be out of margin compared with the use of historical recorded weather data. As seen in Chapter 6 it is possible to only use forecast weather data, but the best possible option is combine it with the use of historical data to properly calibrate the FRI model and then use the forecast data on top. It should also be noted that the last few days of the forecast can be inaccurate at times, but for the most part it is within realistic margins compared to the historical measured fire risk indications. There are a few instances where the differences in time to flash over reaces 1 minutes, but overal the difference is lower.

In Chapter 6, the result and comparison of the different sources were discussed. In the case of using Netatmo stations versus MET stations, related to R1-B, it can be concluded that they can be used as a source of weather data. As seen in the examples in chapter 6, the Netatmo station in Bergen had humidity measurements within reasonable margins before applying the calibration curve. The station in Gjøvik did not have the same margins, bu the result when not using the calibration curve was still within reasonable limits. Based on this we can conclude that it is possible to use Netatmo stations without a calibration curve, but as seen from the result in appendix A the measurements varies widely from station to station. It would therefore be advised to calibrate the station before using it as a source of fire risk calculations.

Based on the result of indicating the fire risk of the four locations, we can identify a period from the start of January to the middle of February as the period with the highest fire risk. After this period the fire risk reduces steadily with a few periods where it goes up again before it starts to decrease during middle of April. It is at this point the climate gets warmer, but that does not necessarily mean that the fire risk will be lower. The reason for this is because of the dry grass and nature that can easily catch fire. This may also have an impact on the fire risk for later periods of the year.

7.1.2 Future Work

One major issue regarding the use of the FRI model for fire risk indication is that it only works for colder climates. One of the current issues is that whenever the temperature goes above a certain point, it stops working properly. For that, reason there were certain period around April - May where the temperatures were to high. This is noticeable in some of the figures in Chapter 6. This is one area where the the FRI model will have to be refined in order to be applied to warmer periods throughout the year.

Another aspect that may be improved in the future is to involve more parameters when calculating the fire risk. This could be done by including more weather elements, such as wind to get a more accurate result regarding conflagration risk. There is also the possibility of combining the wind with a parameter that may indicate how densely the houses are located.

There is also the possibility of expanding the FRI model to look at other means for fire risk indication for other periods of the year. Mainly the ones dealing with high temperatures and dry nature as pointed out above. The current FRI model may give a low fire risk during a heating period since it only looks at the house. It is possible that during this period there may in fact be a high fire risk due to greater risk of heather fires.

7.2 Software Architecture

Research question R2 asks whether there is a suitable software architecture for collecting sensor data and a Fire risk indication service provider. This question is split into two sub-questions, R2-A and R2-B. Question R2-A asks if there there is a software architecture that can be used both for evaluation and deployment.

7.2.1 Conclusion

As the external services used in the FR application is based on RESTful web service it seemed most appropriate to structure the FR application in a similar manner to avoid complications. Because of this, the fire risk web service offered by the FR application was also implemented as a RESTful web service. The use of a micro-service architecture also complemented the use of REST as the components can also use it to communication between each other. Since there is only one specific form of communication in the FR application, it limits the miscommunication between the components of the FR application as it does not have to combine several forms of communication.

Chapter 5, which details the implementation of the FR application, includes the program flow of the whole FR application. This flow is split in two, where one handles the validation part and the other handles communication with the clients. Based on this, we have shown that it is possible to have a software architecture that can both be used as part of validation, but also as a part of the deployment of the finished FR application. By splitting the FR application into several components, as dictated by the micro-service architecture, and using REST as communication made it easy to set up test functionality and the functionality aimed at the clients.

7.2.2 Future Work

As of now, the communication between the components is synchronous and based on REST. An idea for the future would be to see if it possible to convert to an asynchronous communication FRI model by the use of a messaging-driven system. This can be in the form of a message queuing system or a topic based system. An example can be with the use of JMS[2] or CloudMQTT[25].

It will also be a possibility to further improve the implementation of the FRI model for fire risk indication with the hopes of making it more efficient than it is at the moment. However, the time it takes to compute the fire risk is very low, and for that reason it would be more applicable to spend more time to see if it is possible to reduce the time it takes to retrieve data from external services possibly by using some form of background fetch of the data.

Another aspect to look into is the possibility of different client sides, with regard to mobile phones and other platforms.

7.3 Storage and run-time efficiency

The seconds sub-question, R2-B, focuses on whether the software architecture is efficient with regard to storage and run-time computation of fire risk indications.

7.3.1 conclusion

The total amount of storage for the winter 2018/2019 comes back to 18.42 Mb from various sources of weather data. With regard to storage efficiency, the FR application uses very little storage and we can conclude that the software architecture has adequate storage efficiency. Furthermore it has been shown that it does not accumulate large amount of weather data data. The state of the FR application does not store the fire risk indication, therefore the fire risk indication has to be computed every time it is requested by fetching weather data that is stored in the database.

With regard to the run-time efficiency of creating fire risk indications, most of the time was spent fetching data from the external services. The time it takes to compute a fire risk indication was negligible, even when using weather data for a whole year. This indicates that the implementation of the FRI model is adequate regarding the run-time efficiency. The most time consuming operation of the FR application was the conversion of weather data from the request. As seen in chapter 6, the time it took to convert the data from the request was 0.2 seconds for the whole year which can be considered to be within reasonable time.

7.3.2 Future Work

There are certain aspects of the FR application that may be improved in the future. One of them have to do with how the weather data is stored. As stated, the Netatmo and MET data is stored in bulks spanning 24 hour intervals. It would be of interest to investigate if it would be more efficient and easier to handle by storing them as single entities containing only the weather data? The same question can be asked regarding storing the fire risk indication in an efficient manner.

A possible aspect to consider, would be to look at a better method for converting the weather data from the request to the format of the related programming language used.

Bibliography

[1] T. Log, "Building and Environment," Indoor relative humidity as afire risk indicator, Sciencedirect, pp. 238 - 248, 10 September 2016.

[2] Oracle, Java Enterprise Edition 7, docs.oracle.com/javaEE, https://docs.oracle.com/javaee/7/JEETT.pdf, Septermber 2014

[3] Meterlogisk Institutt, Historical Observer Weather, www.met.no, https://frost.met.no/reference#!, 15 Mars 2018

[4] Meterlogisk Institutt, Weather Forecast, www.met.no, https://api.met.no/weatherapi/locationforecast/1.9/documentation#/, 20 Mai 2014

[5] Netatmo, Weather stations, www.netatmo.com,

[6] Netatmo, Historical Weather data, api.netatmo.com/api/

[7] Tutorialspoint, JSF Architecture, www.tutorialspoint.com, https://www.tutorialspoint.com/jsf/jsf_architecture.html, (Accessed 23 August 2018,

[8] IEEE, Microservices, https://ieeexplore.ieee.org/abstract/document/7030212 Jan. -Feb 2015, ()

[9] Building Microservices, Designing fine-grained systems, S. Newman, 2015

[10] Patrick TH. Eugster, Pascal A. Felber, Rachid Guerraoui, Anne-Marie Kermarrec, The Many Faces of Publish/Subscribe, ACM Computing Surveys volume 35 issue 2, pp 114-131, June 2003

[11] Message-Oriented Middleware, Middleware for communication, E. Curry, 2004

[12] Guruduth Banvar, Tushar Chandra, Robert Strom, Daniel Sturman, A case for Message Oriented Middleware, ACM Symposium on the Principles of Distributed Computing, 1999

[13] MongoDb, Document Database cloud, https://www.mongodb.com/cloud/atlas/pricing, (access 20 November 2018)

[14] Jing Han, Haihong E, Guan Le, Jian Du, Survey on NoSQL database, 2011 6th International Conference on Pervasive Computing and Applications, IEEE, 26-28 October 2011

[15] Ameya Nayak, Anil Poriya, Dikshay Poojary, Typeof NOSQL Databases and its Comparison with Relational Databases, nternational Journal of Applied Information Systems (IJAIS), Volume 5–No.4, March 2013

[16] Jenssen JA, Geving S, and Johnsen R, "Assessments on Indoor air Humidity in Four Different Types of Dwelling Randomly Selected in Trondheim, Norway" In: Proceedings of the 6th Symposium on Building Physics in the Nordic Countries, Trondheim, Norway, 17th-19th June, 2002, pp 729-735

[17] Fielding RT, Architectural styles and the design of network-based software architectures, 2000

[18] T. Log, Modeling Moisture Content and Time To Flashover as a Proxy for Wooden Home Fire Risk in Cold Climates, Work in progress

[19] Spark, Micro framework for creating web application in java with minimal effort, `www.sparkjava.com`

[20] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel MazzaraFabrizio Montesi, Ruslan Mustafin, Larisa Safina, Microservices: yesterday, today, and tomorrow, 20 Apr 2017

[21] Google, `https://github.com/google/gson`, (Accessed November 2018)

[22] Prime Faces, `https://www.primefaces.org`, (Accessed 23 January 2019)

[23] Apache TomEE, `https://tomee.apache.org/`

[24] Apache Maven, `https://maven.apache.org/`

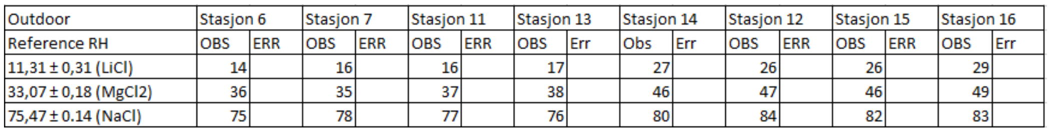
[25] CloudMQTT, Globally Distributed MQTT broker, `https://www.cloudmqtt.com/`, (Accessed 20 may 2019)

[26] Farid Ighoubah, Simon Solheim, (2014), Slik var de første meldingene om Lærdalsbrannen, nrk.no, 20 Januar, (Accessed 5 May 2019)

[27] Carina Hunshamar, Irene Rønold, Gordon Andersen, Martha Holmes, (2017), Brann i Kongsberg: En person funnet død, vg.no, 26 December, (Accessed 7 May 2019)

Appendix A

Result of Calibrating Netatmo Stations



Outdoor	Stasjon 6		Stasjon 7		Stasjon 11		Stasjon 13		Stasjon 14		Stasjon 12		Stasjon 15		Stasjon 16	
Reference RH	OBS	ERR	OBS	ERR	OBS	ERR	OBS	Err	Obs	Err	OBS	ERR	OBS	ERR	OBS	Err
11,31 ± 0,31 (LiCl)	14		16		16		17		27		26		26		29	
33,07 ± 0,18 (MgCl2)	36		35		37		38		46		47		46		49	
75,47 ± 0.14 (NaCl)	75		78		77		76		80		84		82		83	

Figure A.1: The humidity measured by Netatmo stations with known humidty by salts