Credibility Model of Local Weather Prediction

Bc. Filip Findura

Master's thesis
2023

Tomas Bata University in Zlín
Faculty of Applied Informatics

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Filip Findura**
Osobní číslo:	**A21493**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Model věrohodnosti lokální předpovědi počasí**
Téma práce anglicky:	**Credibility Model of Local Weather Prediction**

Zásady pro vypracování

1. Proveďte rešerši veřejně dostupných API s daty o počasí (aktuální i předpověď).
2. Doporučte zdroje dat pro aktuální počasí i předpověď počasí.
3. Implementujte systém sběru dat o počasí z daných zdrojů a jejich uložení v databázi.
4. Implementujte model závislosti aktuálního počasí a předpovědí a určete odhadovanou spolehlivost předpovědi.
5. Implementujte rozhraní pro získání modelových dat.
6. Zpracujte dokumentaci k výslednému softwaru.
7. Věnujte pozornost zabezpečení aplikace.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. BRÖCKER, J. Testing the reliability of forecasting systems. Journal of Applied Statistics [online]. 1-25. ISSN 0266-4763. Dostupné z: doi:10.1080/02664763.2021.1981833.
2. THORNES, John E.; STEPHENSON, David B. How to judge the quality and value of weather forecast products. Meteorological Applications, 2001, 8.3: 307-314.
3. STEWART, Thomas R. Improving Reliability of Judgmental Forecasts. In: ARMSTRONG, J. Scott, ed. Principles of Forecasting [online]. Boston, MA: Springer US, 2001, 2001, s. 81-106. International Series in Operations Research & Management Science. ISBN 978-0-7923-7401-5. Dostupné z: doi:10.1007/978-0-306-47630-3_5.
4. FENTE, Dires Negash a Dheeraj Kumar SINGH (2018). Weather Forecasting Using Artificial Neural Network. In: International Conference on Inventive Communication and Computational Technologies (ICICCT). Coimbatore, INDIA. Dostupné z: https://www.webofscience.com/wos/woscc/full-record/WOS:000456251700349.
5. MAQSOOD, Imran, Muhammad Riaz KHAN a Ajith ABRAHAM. An ensemble of neural networks for weather forecasting. Neural Computing and Applications [online]. 2004, 13(2). ISSN 0941-0643. Dostupné z: doi:10.1007/s00521-004-0413-4.
6. SHRIVASTAVA, Gyanesh, et al. Application of artificial neural networks in weather forecasting: a comprehensive literature review. International Journal of Computer Applications, 2012, 51.18.
7. ABISTADO, Klent Gomez, Catherine N. ARELLANO a Elmer A. MARAVILLAS. Weather Forecasting Using Artificial Neural Network and Bayesian Network. Journal of Advanced Computational Intelligence and Intelligent Informatics [online]. 2014, 18(5), 812-817. ISSN 1883-8014. Dostupné z: doi:10.20965/jaciii.2014.p0812.
8. KUBO, Yûki. Why do some probabilistic forecasts lack reliability?. Journal of Space Weather and Space Climate [online]. 2019, 9. ISSN 2115-7251. Dostupné z: doi:10.1051/swsc/2019016.

Vedoucí diplomové práce: **Ing. Adam Viktorin, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **2. prosince 2022**
Termín odevzdání diplomové práce: **26. května 2023**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.
1

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

ABSTRACT

This work discusses the implementation of software for weather data scraping, storage and evaluation, including the methodology and technology used, the security considerations and the project's sustainability. It offers an overview of available weather data sources, then a neural network is used to develop a model of relations between weather forecasts and measured weather data, and this model's credibility is compared against that of the weather forecast data sources.

Keywords: weather forecast, predictive modelling, neural network

ABSTRAKT

Tato práce se zabývá implementací softwaru pro získávání, uschovávání a vyhodnocení dat o počasí, včetně popisu použité metodologie, technologií, aplikační bezpečnosti a udržitelnosti projektu. Taktéž zpracovává přehled dostupných zdrojů dat o počasí. Následně popisuje využití umělé neuronové sítě pro vytvoření modelu vztahů mezi předpovědí počasí a meřenými daty o počasí, a vyhodnocuje tento model proti dostupným zdrojům dat o počasí.

Klíčová slova: předpověd počasí, model předpovědi, neuronová síť

TABLE OF CONTENTS

INTRODUCTION

Weather has an immense impact on human life and work, as all manner of industries and activities either depend on or are limited by favourable weather conditions. Even in places where weather is of no immediate concern, an indirect influence of weather can be relevant, as unfavourable weather may have sever consequences for manufacturing, transportation, services and human life in general. Therefore, knowing in advance whether weather will be adversarial or nor is a valuable information, making weather forecasting a highly sought-after service.

This of course means that multiple such services exist and can be used. Each service then utilizes a different technique or solution for forecasting, arriving at a slightly different conclusion as to what weather to expect - there is always a factor of uncertainty in predicting the future [1]. As such, a potential client has to decide which weather forecast service provider to trust.

Due to the current push for both economical and ecological optimization in resource usage, a number of solutions is being developed for fine-tuning the resource management of buildings, machines and whole production facilities. Where there is demand for exact temperature management and optimization, weather forecasting plays a significant role, and while there are rarely vast differences in the predicted weather between the available services, these differences still have an impact on the final results of any optimization systems.

In addition, many weather forecasting services are global in nature, using large-scale models and gathering current weather data from public weather stations. Many facilities which would use these predictions, however, have their own equipment for gathering weather data specific to their location. Therefore, it would be beneficial to better understand the differences between weather forecasting services, to be able to determine the credibility of each given weather forecasting service for a specific location, and to increase the accuracy of weather forecasts by applying the knowledge of the local weather data as observed at each location.

I. THEORETICAL PART

1 MOTIVATION

In this master's thesis (henceforth referred to as "the Project"), we shall examine the various publicly available sources of weather forecast data and compare them. Then, an application (henceforth referred to as "the Software") shall be developed for weather data acquisition and storage, which will make it possible for the user to easily access and contrast multiple weather forecasts. In addition, a weather model based on the relations between weather forecasts and the weather data observed at location[1] shall be prepared and a credibility analysis will be conducted to contrast the modelled data and the forecast data. Finally, a documentation for the Software will be provided.

1.1 Thesis aims

This thesis encompasses the following goals:

1. Get acquainted with the publicly available weather data sources[2].
2. Recommend a source for historical, current and future weather data[3].
3. Implement a system of weather data collection from multiple sources and their saving to a database[4].
4. Implement a dependency model of the current weather and weather forecast, and find the credibility of forecast[4].
5. Implement an API for acquiring data from the model[5].
6. Provide a documentation of the software[6].
7. Analyse the security of the software[7].

This Project focuses on weather data acquisition, processing and prediction.

[1] Also referred to as "current weather data", "measured weather data" or "real weather data".
[2] See chapter 6.1, where multiple weather data sources are discussed in detail.
[3] See chapter 6.2, where the APIs for the Software are chosen.
[4] See chapter 10, where the implementation of the Software is discussed.
[5] See chapter 10.4.1, where the REST API is further detailed.
[6] See chapter 10, and also the documentation included in the Software's code repository, as found in appendix A1.
[7] See chapter 8.3 for security analysis and recommendations.

In the Theoretical Part, an overview of weather forecasting and predictive modelling is provided, along with the Project's methodology and an examination of containerization architecture that will be applied in developing the Software. In addition, the technologies used in the Software are discussed.

In the Analytical Part, a comparison of the currently available weather data sources is presented and compared against the Project's requirements and concerns. The available neural network configurations are tested and the best combination of configurations is chosen, then an application security analysis is performed and security recommendations for the Software are postulated. Finally, the Project's benefits, risks and sustainability are examined.

In the Technical Report, the individual components of the Software are more closely described and documentation is provided for them. Additionally, a credibility evaluation of weather data sources is performed.

1.2 Project contribution

The Project aims to offer an easy comparison of multiple weather data sources. The Software provides a quick statistical analysis of forecast data versus real weather data for each data source, including modelled weather data based on the forecasts and real weather data from multiple sources. The user can therefore make an informed decision on which source of weather data to use for their purposes.

The Software can also serve as storage for weather data, both forecast and real, allowing access to historical weather data (since the deployment of the Software) for any user that does not wish to use a paid weather service that would provide historical weather data. As the Software includes a REST API for retrieving any of the weather data, it can serve as a weather data hub for other applications.

While only four weather data sources will be included in this Project, the Software is designed with multiple data sources in mind, to let further data sources be smoothly added at a later date. Each additional data source added will also promote the primary benefits of the Software mentioned above, providing supplementary data points for modelling and comparison.

Importantly, the Software will be of particular interest to customers who already possess a way of measuring weather data at their location, as the Software is able to calibrate the weather model on local real weather data, both providing more accurate predictions and revealing the most accurate source of forecasts for this specific location.

The Software delivered along with this Project will be a proof of concept that demonstrates the feasibility of modelling the relations between weather forecasts and real weather data, and will provide the basis for further development and refinement.

2 PROJECT METHODOLOGY

The Project shall use the Waterfall methodology to ensure a successful completion of individual phases of the Project given the time constraints for finalizing the whole Software. The Waterfall methodology is a project management approach based around a one-way progression through several phases of a project, with each phase being finished completely before the following phase begins [2]. The common phases of a waterfall-managed project can be seen in figure 2.1:

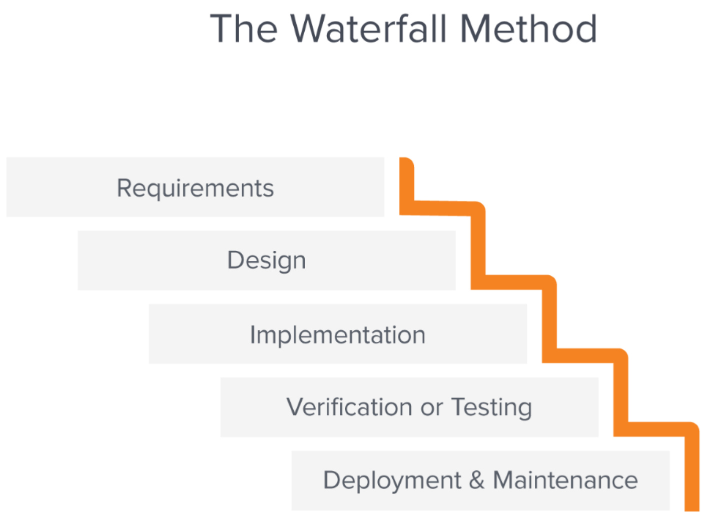


Figure 2.1 Example of common project phases in Waterfall management [3]

1. **Requirements** need to be established, to plan the parts of the software that will meet them.
2. **Design** of the whole structure of the software is performed.
3. **Implementation** of the software is realized.
4. **Verification** with the user and/or **testing** of the functionalities is effected.
5. **Deployment** to production is planned and further **maintenance** arranged.

The Waterfall methodology was chosen as thanks to its front-loaded strategy, where all features of the resulting project are to be accounted for in the planning phase, allows for a more reliable time estimates for individual features of the to-be-developed software, and thus offers a more predictable release date. As this Project must be finished within a strict time frame, the Waterfall methodology can help ensure that the deadline is successfully met.

As a downside, with all the planning done upfront and all features designed from the get-go, a later change in such a project can prove problematic. However, as this Project has a fixed set of requirements that it aspires to fulfill, it is unlikely that the need for a new feature will be discovered partway through the development process, barring a grave error in the planning phase.

3 FORECASTING

Forecasting in general is the estimation of future data (values, events, or anything else we might be interested in) from past and current data, which might refer both to the more formal mathematical and statistical methods of predicting the future, which depend on availability of historical data, but offer a high degree of precision once a good model of the system is established, and the less formal judgemental methods, which are less reliable and subjective, but worth exploring where usable data is not available [1]. In this Project, we will be discussing the former approach of modelling a predictive system.

3.1 Predictive modelling

In machine learning, predictive modelling is a technique that uses existing data to train a model of the given system which can then predict future data. It is a statistical technique, as it searches for patterns in the given data set to deduce the chances of a future event occurring. Therefore, with a good sample of historical data, it is possible to create a reliable system for prediction of future events. Such techniques are used in a wide array of jobs - from detection of unusual activities in banking or cybersecurity to optimization of stock in both factories and shops - as it allows the business to enhance current products or services, better understand developments that are coming and their impact on production, or to mitigate risks and prepare for problems before they arrive.[1]

Predictive modelling requires a set of current and historical data, which are then used to train a predictive model. Then, the model must be evaluated as to its effectiveness against real data and other models. Therefore, the following six steps are usually preformed in a predictive modelling project [4]. The first three steps encompass data preparation, the next three steps model development.

1. **Data collection:** Acquiring a set of current and historical data. This data is then used to create a training and testing data set.
2. **Data cleaning:** Data preparation for model training usage. It is necessary to make sure that the data sets do no contain incorrect data, or to fix any missing, redundant, or otherwise corrupt data that might interfere with model creation.
3. **Data exploration:** The final step in data preparation, where the data set is

analysed in respect to associations in data points, to decide which kinds of data will be used as inputs in training of the model. Incorrectly chosen inputs will lead to worse results of the model.

4. **Model training:** The implementation of machine learning algorithms and their training, resulting in a testable model.
5. **Model testing:** A series of tests that should determine the success rate of given model. This step is crucial to assess the model's credibility and to compare it against other models and approaches.
6. **Model deployment:** Using the model on a real-world data set. It is necessary to check the performance of the model outside of the training and testing data sets, and then keep checking the results against the actual data we will collect in the future.

Data collection and data cleaning will then be repeated continuously after model deployment, to provide new inputs for predictions.

Various types of predictive models exist, such as classification, clustering or outlier models [1], but this Project will focus on forecasting models which predict a future value in a data set, often for multiple parameters at one time. Of course, there are also numerous algorithms that are used in such a forecasting model's implementation.

3.2 Forecasting algorithms

According to the so-called "no free lunch" theorem, there is no machine learning algorithm that would work the best for every problem [5]. Therefore, we have to choose whichever algorithm would likely be well-suited to the task at hand, as there are always many factors at play and each algorithm responds better to a different type of problems.

For the Software, a LSTM network was selected as the most suitable tool, given its good results with predicting time series data. [6][7]

3.2.1 LSTM networks

A Long Short-Term Memory network, or a LSTM network, is a type of an artificial neural network, or specifically a recurrent neural network, which includes both feedforward and feedback connections. This makes LSTM networks very powerful when it comes to predicting data in a sequence or time series. [7]

Standard RNNs have a "long-term memory" and a "short-term memory", as the weights are adjusted per epoch and per single step. LSTM networks were developed as a response to the so-called "vanishing gradient problem" [8] of standard RNNs, where information may be lost before entering the long-term memory due to finite precision of the floating point variables used for weights. LSTM networks extend the short-term memory to a much larger number of steps, which is achieved by including memory cells in the network. These memory cells store previously encountered information and selectively inject it into the current set of data. Thus, LSTM networks are often used in forecasting of time series data, as there can be dependencies between distant points of the time series, or "lag" between cause and effect, that would "vanish" before being discovered if the data were processed by a standard RNN. Effectively, a LSTM network learns which information may be useful later and retains it.

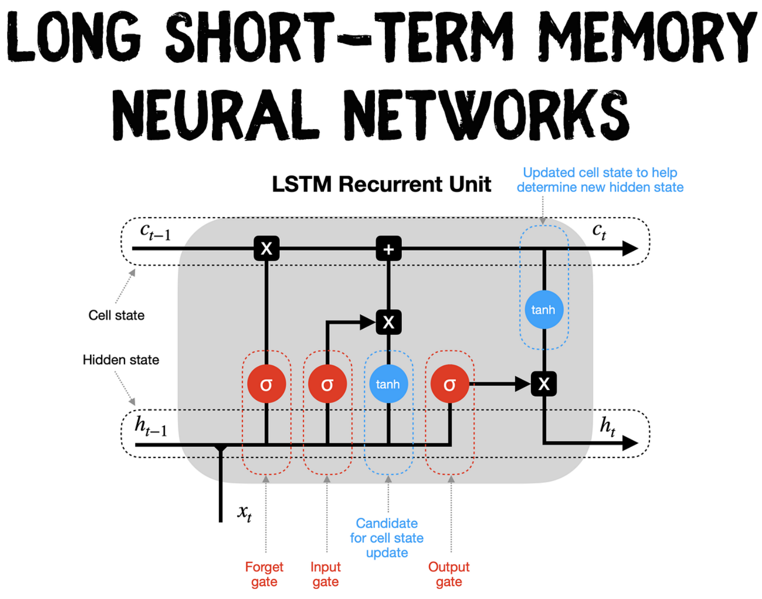

LONG SHORT-TERM MEMORY NEURAL NETWORKS
LSTM Recurrent Unit
Updated cell state to help determine new hidden state
c_{t-1}
c_t
Cell state
Hidden state
h_{t-1}
h_t
x_t
X
+
X
X
tanh
σ
σ
tanh
σ
Candidate for cell state update
Forget gate
Input gate
Output gate

Figure 3.1 Memory cell in a LSTM network [9]

The memory cell of a LSTM network[1)] has four main components: the forget gate, the input gate, the output gate, and the cell state. The forget gate controls how much information is retained and for how long. The input gate controls which information is added to the memory cell. The output gate controls how much is the current step supplemented with the remembered information. And finally the cell state is the memory itself, a vector of data points from previous training steps. [9]

1) See figure 3.1.

LSTM networks thus offer some advantages over standard RNNs:

1. They handle long-term dependencies well, as they remember information and thus lessen the impact of the vanishing gradient problem.
2. They handle complex sequential data well, as the increased capacity for finding long-term dependencies increases the precision of overall results on complex time series.

However, they also suffer from certain drawbacks:

1. They require more data for good results than a standard RNN.
2. Their training is also more computationally expensive, owning to their higher complexity.
3. They are not very applicable to problems which do not use sequential data.
4. They do not handle data with a lot of noise well.

In addition, as with all neural networks, it is necessary to select a suitable optimization algorithm and a loss function.

3.2.2 Optimization algorithm and loss function

Neural networks are trained by forming weighted, probability-based relations between the input and output sets of data, using an optimization algorithm to adjust the weights after each step of training. The optimization algorithm attempts to adjust the weights in such a way that the error of prediction is minimized, using a loss function to find the error of the current step of training. In other words, the loss function (also known as an objective function, error function, cost function or criterion [10]) sums the differences between the training inputs and outputs into a single number which can then be minimized (or maximized) using the optimization algorithm.

Depending on the problem to be solved by the neural network, the suitability of different optimization algorithms and loss functions varies. Therefore, it is useful to test which would perform best on our specific set of data. This test is discussed in chapter 7, where the optimization algorithms and loss functions offered by the machine learning platform[2)] are compared.

[2)]See chapter 5.

4 CONTAINERIZATION

Containerization is a form of virtualization on the application level which ensures that software may run in an isolated space, the so-called *container*. Containers are fully functional and portable self-contained environments enveloping the run time of the application, letting it run separately from other containerized environments and applications running on the same hardware. The code for each container is packaged into an *image*, which is a static file that includes all necessary configuration files, libraries and other dependencies to run the application, making it independent on the base environment or infrastructure, which in turn makes it easy to move the application between platforms. [11]

On the other hand, unlike full virtualization, all containers on the same platform share the host machine's kernel, which makes them more light-weight when compared to virtual machines. Where a virtual machine would measure in gigabytes in size and include its own operating system, CPU, memory, et cetera, a container generally encompasses only a single application and its dependencies, making it much faster and manageable on smaller platforms, including personal computers. [12]

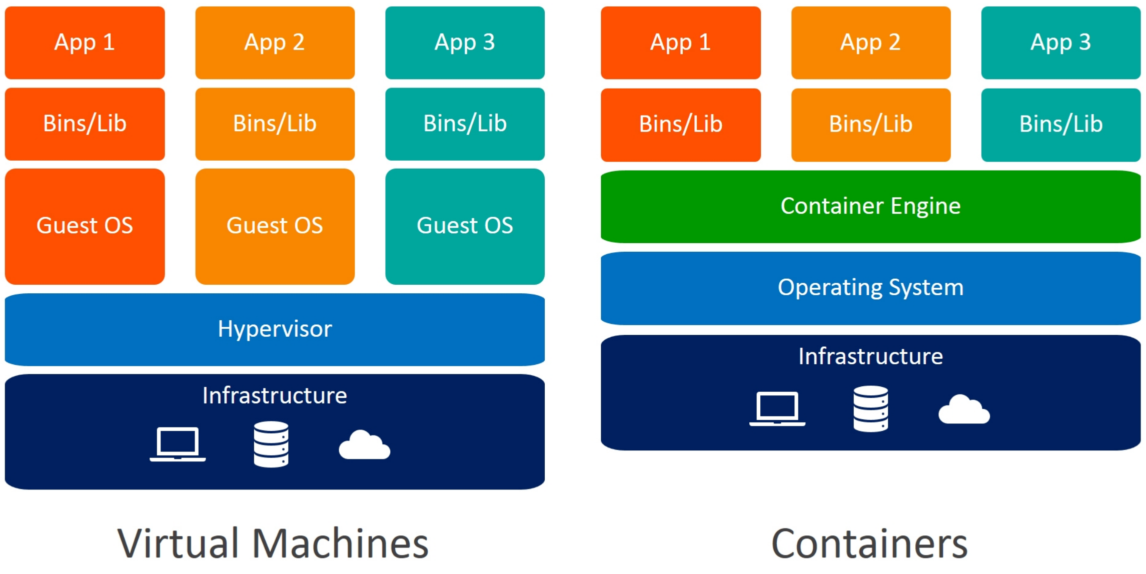


Figure 4.1 The differences in architecture of virtual machines and containers [13]

Even then, containers are rarely used to wrap large applications into a single package. Instead, each container performs a single task, breaking up the functionality of a larger application into a set of specialized microservices. Microservices and containers fit rather well together, as containers offer lightweight portability, compatibility, and scalability, while microservices are best designed as small, standalone, mostly independent features.

4.1 Microservices

Microservices are an approach to building a larger application from a set of distributed, self-contained components with a specialized function. Each microservice is independent of the larger application, to a degree, meaning that the parts can work or fail without affecting the whole. Each microservice serves a single core function and can be independently developed and changed without changing the rest of the overall application at the same time. [14]

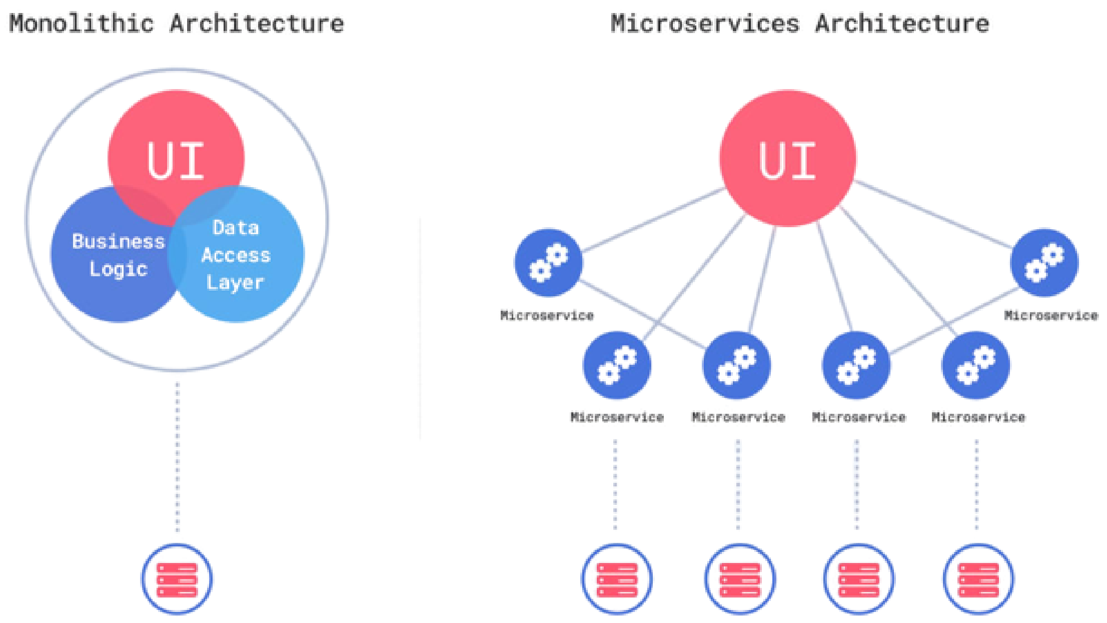


Figure 4.2 The difference in architecture between monolithic and microservice applications [15]

To allow a full inter-independence of microservices, the application architecture must ensure a way of communication between the services that doesn't break when one part of the application changes. Such is the task of an application programming interface.

4.2 API

An *application programming interface*, usually shortened to API, is a way of passing data between services or applications in a pre-defined, formalized way. It is a service, returning a standard response when a request is received, or exposing only those functions of a software that an outside actor may need to interact with. Unlike a user interface, an API is not intended to be used by a human, but rather by another piece of software.

The main advantage of an API is the standardization of data exchange, which hides the internals of the service. As such, the service behind the API may change as much and as often as needed, but other systems are not affected as long as the API remains untouched. This is especially useful for microservices, which are already built as independent, and therefore the systems underlying various parts of an application may be developed and deployed on their own, without affecting the overall application.

REST API is then an interface following a specific set of guidelines developed for exchanging information over the Internet. [16]

5 TECHNOLOGIES USED

In developing the Software, two following technologies were used. When choosing the technologies, open-source solutions were preferred to keep the operating costs of the Software low, as mentioned in chapter 9.

5.1 Python

The component microservices of the Software, further discussed in chapter 10, were written in Python, which was chosen due to its wide support for machine learning. While multiple Python packages were used in minor manner, the following packages are the most relevant to the Software's functionality (in alphabetical order):

- **ephem** [17] is an astronomy library used to compute the position of astronomical bodies around the Earth. While the weather data gathered from weather APIs do not include irradiance1), it is a useful metric that can be calculated from the available weather data and the position of the Sun, using this library.
- **Flask** [19] is a web interface framework, used to run the Software's API.
- **keras** [10] is a deep learning library working with TensorFlow (see below) to make machine learning code more concise, maintainable and also faster.
- **NumPy** [20], **pandas** [21] and **scikit-learn** [22] are libraries supporting data manipulation, processing, analysis and statistical evaluation.
- **SQLAlchemy** [23] is an interface package for SQL databases, which lets the Software be easily switched to a different database, should the need arise.
- **TensorFlow** [24] is a machine learning and artificial intelligence platform focusing mainly on deep learning.

5.2 Docker

Containerization in Docker was already discussed in chapter 4. For the purposes of developing the Software, Docker Desktop [25] was used, a fast way to set up containerized

1)Irradiance is the radiant power received by a surface area, per unit. [18]

applications on a personal computer. As mentioned in chapter 9, in production deployment a full version of Docker running on a server is recommended, or a containerization platform such a Kubernetes or OpenShift.

As the Software is a multi-container application with cross-container dependencies, Docker Compose [26] was used to configure the whole application. Docker Compose is a tool for configuring larger Docker applications to make their deployment simple and standardized. On Kubernetes or OpenShift, Helm [27] could be used for the same purpose, packaging the configuration of the whole application into a single easy-to-use setup chart.

5.3 PostreSQL

PostgreSQL was chosen as the database for the Software, as it is the highest ranking open-source relational database [28] that also allows an easy setup in a containerized environment. However, any other relational database could be paired with the Software with minimal changes required, as the SQLAlchemy library mentioned above offers a set of drivers for most SQL databases.

5.4 Grafana

Grafana [29] is an open-source data analysis web application. It provides visualization dashboards that can directly interface with a database, and is most often used for graphing time series data, or for monitoring. It can be set up as a containerized service, and while visualization is not the main goal of the Software, it is useful as an aid in quickly assessing the weather data gathered and processed. In addition, Grafana allows the user to implement their own visualization by adding new dashboards with graphs and other components of their choice, making personal adjustments and modifications to the visualization very easy.

II. ANALYTICAL PART

6 WEATHER DATA SOURCE

To meet the requirements of this Project, we need one or more online services that will serve as the sources of weather data, both forecast and current, for the data gathering component of the Software. In this chapter, we will discuss first the services available as weather data sources, then the viability of individual services for the Project.

6.1 Available data sources

An inquiry was made into the available online weather data services offering an API that could be connected to via the data gathering component of the Software. The following paragraphs summarize, in alphabetical order, the explored options. The listing does not offer an exhaustive enumeration of all the options and plans available at the mentioned service providers; only the plans that would be most applicable for the Project were included. Other sources of weather data were also considered, but rejected as they do not provide an API which the Software could use.

6.1.1 AccuWeather

AccuWeather has an API with a **Free** plan limited to fifty calls per day, but it offers no historical data and an hourly forecast for only five days. [30]

6.1.2 Azure Maps Weather

Azure offers a thousand free calls of their weather API to any new account, or a 200$ sign-up bonus [31]. Their weather API then has current weather data, an hourly forecast for up to ten days, and historical data for up to forty years back, all priced per call [32].

6.1.3 ČHMÚ

ČHMÚ (Český hydrometeorologický ústav, Czech Hydrometeorological Institute) has an archive of weather data in the Czech Republic, access to which is, however, a paid service at a negotiable price. Rather than an API, ČHMÚ offers a one-time bulk download of historical data. [33]

6.1.4 in-Počasí

in-Počasí offers a free archive of daily measured data, with downloads available in CSV, which unfortunately makes it unsuitable as a data source for the Software. [34]

6.1.5 Meteoblue

Meteoblue offers **Weather API+** service, with the cheapest plan starting at 72.60€ (1 721 Kč) per year. Meteoblue uses a proprietary weather model for nowcasting global and local weather in high-resolution, for four days in the past and up to fourteen days in the future. This includes specialized data for agriculture, solar or wind power industry, and more. [35]

6.1.6 Meteocentrum

Meteocentrum has a freely accessible archive of measured data in the Czech Republic since 2018, manually downloadable as CSV. While this could serve as a source of test data, it is not possible to easily automate for regular use. [36]

6.1.7 Meteomatics

Meteomatics offers a weather API with a **Free Basic** package which does not allow commercial usage, is limited to five hundred calls per day, and includes a ten day forecast and one day of historical data. The **Business** package, on the other hand, includes a fifteen day forecast and historical data since 1979, covering both measured and historical forecast data. [37]

6.1.8 MeteoSource

MeteoSource has a **Free** plan with four hundred calls per day, including the current weather and hourly forecast for one day. Historical data is however only available from the **Startup** plan onward. The most useful for gathering historical data would be the **Flexi** plan, though, priced at 5 $ (109 Kč) per month, with just a hundred free calls per day, but offering hourly forecast for seven days and eight years of historical data.[38]

6.1.9 Meteostat

Meteostat offers weather data over Rapid API. Their **Basic** plan is for free, limited to five hundred calls per month. The **Pro** plan, priced at 9.99 $ (218 Kč) per month, grants ten thousand calls per month. Meteostat provides current weather and historical measured weather data since at least 2000. [39]

6.1.10 Norwegian Meteorological Institute

Norwegian Meteorological Institute (NMI) offers a free **Location Forecast API** covering weather forecast for one location and a nine day period [40], at no more than twenty requests per second per application [41]. However, no current weather data are available.

6.1.11 Stormglass.io

Stormglass.io provides current weather, ten days of hourly forecast and historical data since 2017. In addition to basic weather data, the service also includes agricultural data such as soil moisture and soil temperature. The two plans of note are the **Free** plan, limited to ten calls per day, plus no commercial use and no support, and the **Large** plan with a thousand requests per day and limited support priced at 19 € (455 Kč) per month. [42]

6.1.12 Tomorrow.io

Tommorow.io offers three plans of note. The **Free** plan includes five days of hourly forecasts and six hours of historical data. The **Enterprise** plan includes fourteen days of hourly forecasts and various premium data, such as air quality, pollen, lightning occurrence, and more. The **Historical** plan includes hourly historical data since 1997. Both the Enterprise and the Historical plan are priced per individual negotiation. [43]

6.1.13 OpenWeatherMap

OpenWeatherMap API provides global weather data with multiple subscription plans to choose from, including current weather data, forecasts and historical weather data for any geographical location. [44]

For the purposes of this Project, two offers are especially interesting. **History Bulk** plan features measured historical weather data for over forty years (since 1st January 1979) at any chosen location or coordinate, priced at 9 € (222 Kč) per location. **History Forecast Bulk** plan features historical weather forecast data1) since 7th October 2017, from any chosen location or coordinate, priced at 40 € (950 Kč) per location. In both cases, the data is available with a one hour step.

OpenWeatherMap also offers six months of free **Student Initiative** plan with access to current weather and forecasts, and one year of historical data, including statistical data.

Alternately, OpenWeatherMap also offers a **One Call API** which grants access to the current weather, an hourly forecast for forty eight hours, a daily forecast for eight days and historical data. The plan includes a thousand free calls per day, with each call over the limit costing 0.0014 € (0.03 Kč).

6.1.14 Weather API

Weather API offers a **Free** plan which includes up to a million calls per month with data covering, in addition to the current weather, hourly forecast for three days and historical data for seven days. The most popular paid plan, **Pro**+ includes up to five million calls per month, hourly forecast for fourteen days and historical data for one year at the cost of 35 $ (780 Kč) per month. The **Business** plan offers ten million calls per month, and in addition to all the already mentioned features also historical data since 2010 at the cost of 65 $ (1450 Kč) per month. [45]

6.1.15 WeatherKit

WeatherKit is a platform-specific weather API for iOS, iPadOS, macOS, tvOS and watchOS applications. Therefore, it is unsuitable for this Project. [46]

6.1.16 Weather Stack

Weather Stack has a **Free** plan limited to two hundred fifty calls per month and current weather only. More useful is the **Standard** plan with fifty thousand calls per month and historical data since 2008, priced at 9.99 $ (222 Kč) per month. Forecast is

1) The data were taken from an original sixteen day forecast.

only included from the **Professional** plan onward, with three hundred thousand calls per month and seven days of forecast, priced at 49.99 $ (1 114 Kč) per month. The **Business** plan with a million calls per month then includes forecast for fourteen days, priced at 99.99 $ (2 228 Kč) per month. [47]

6.1.17 World Weather Online

World Weather Online offers a customisable **Pro** plan where the price depends on the features requested. For current weather, fourteen day forecast and historical data with up to five hundred calls per day, the price is 60 $ (1 336 Kč) per month. [48]

6.2 Project requirements and concerns

The Project aims to implement a weather data collection system that could draw from multiple sources of current, historical and future weather data. From the above mentioned APIs, the following four were chosen:

- **Norwegian Weather Institute** offers a completely free weather forecast, though no current weather.
- **OpenWeatherMaps** offers both current weather and weather forecast for free. In addition, it would be possible to purchase the highly useful Bulk plans for both forecasts and historical measured data of a single location.
- **Weather API** has a very generous Free plan with both current and forecast weather.
- **Weather Stack** is used for its free current weather, to complement the free forecast from NWI.

While more APIs could be added, time restrictions necessitated a limit to the inputs of the Software, given the differing specifications of each API.

7 NEURAL NETWORK CONFIGURATION

When developing the model using the technologies described in chapter 5, a need has been formulated to configure the neural network with an optimization algorithm and a loss function. As a neural network model with multiple inputs and outputs obscures the relations between data and the impact of individual parameters, making informed choice between configurations difficult, a test has been performed instead to compare the performance of different optimization algorithms and loss functions offered by the machine learning platform used in the Software against the weather data gathered by the Software. A static set of weather data was prepared, using one month of forecasts and real weather data[1)], then a model was trained for each combination of an optimization algorithm and a loss function. The results were then evaluated using RMSE[2)]. For each combination, this was repeated a hundred times and the average performance was calculated. Table 7.1 summarizes the findings of this test.

The lowest RMSE and thus the best performance was achieved by the root mean squared propagation optimization algorithm combined with the Huber loss function. Therefore, this combination was used to configure the model ran by the Software.[3)]

[1)]Weather data gathered throughout April 2023 were used.

[2)]See chapter 12 for more information.

[3)]See chapter 10.5, which discusses how to configure the Software with a different optimizer or loss function, should that be required.

Table 7.1 Comparison of optimizer algorithms and loss functions

Optimization algorithm	Loss function	Average RMSE
Adam	mean absolute error	5.23
	mean squared error	5.43
	mean absolute percentage error	19.94
	mean squared logarithmic error	7.04
	consine similarity	17.26
	logarithmic hyperbolic cosine loss	5.42
	Huber loss	5.43
AdaGrad	mean absolute error	9.74
	mean squared error	9.39
	mean absolute percentage error	19.94
	mean squared logarithmic error	12.28
	consine similarity	17.54
	logarithmic hyperbolic cosine loss	11.51
	Huber loss	11.24
AdaDelta	mean absolute error	17.87
	mean squared error	17.64
	mean absolute percentage error	20.12
	mean squared logarithmic error	18.45
	consine similarity	18.25
	logarithmic hyperbolic cosine loss	18.05
	Huber loss	17.98
Root mean squared propagation	mean absolute error	5.09
	mean squared error	5.04
	mean absolute percentage error	19.94
	mean squared logarithmic error	6.49
	consine similarity	15.11
	logarithmic hyperbolic cosine loss	5.05
	Huber loss	5.03
Stochastic gradient descent	mean absolute error	8.26
	mean squared error	8.33
	mean absolute percentage error	13.45
	mean squared logarithmic error	10.05
	consine similarity	17.75
	logarithmic hyperbolic cosine loss	8.66
	Huber loss	8.64

8 APPLICATION SECURITY

Containerized applications carry a somewhat higher security risk, owning to the lightweight and flexible nature of containers along with multiple microservice components exposed on the network. Therefore, proper authentication and authorization, or issues such as misconfiguration and more must be addressed for all individual parts of the application. [49]

In this chapter, we will first categorize the security issues faced by a containerized application, then offer solutions for these issues, and finally elaborate on the security recommendations for the Software's deployment.

8.1 Security issues

Sorting by which part of the system is affected, we can recognize the following categories of risks and possible solutions for them. [50]

8.1.1 Image risks

This category includes vulnerabilities present in images themselves, such as outdated libraries with since then known security vulnerabilities, or embedded malware. Another form of risk may be posed by authentication keys or certificates packaged unencoded in the image, with will compromise the rest of the system if one container is successfully attacked, or bad configuration left over from development, often involving exposed insecure ports unnecessary for the application. Image build monitoring and testing, such as unit tests and static application security testing (SAST), will reduce the risk of building and deploying a vulnerable image.

8.1.2 Infrastructure risks

Insecure connection to an image registry or an authentication and authorization vulnerability may lead to deploying an infected or fake image. Encrypted or otherwise secured connection between the registry and the platform should thus be provided and only trusted sources of images used. Proper authentication and authorization when uploading and downloading images is also prudent.

Similar risks and solutions are also applicable to the CI/CD part of infrastructure.

8.1.3 Platform risks

Insecure intra-platform traffic, unauthorized administrative access or insufficient separation of platform networks may help compromise the whole application, or all applications on the platform. Once again, proper authentication and authorization will prevent unauthorized access. In addition, discrete virtual networks should be created on the platform to separate the outward-facing parts of the system from the internal parts working with sensitive information.

8.1.4 Host risks

Unmonitored attack surfaces of the host machine, operating system vulnerabilities, or direct file system tampering are among the hardest to detect if present. As the host system lies at the lowest level of the architecture, it is the most critical target for security threats. Therefore, the physical host should not be used to run other systems but the containerization platform, reducing the area of exposure, and should always be kept up to date with security patches. In addition, it should be properly configured as far as limiting necessary network access goes.

8.1.5 Runtime risks

The previous risks allow a runtime risk to emerge when a compromised container is deployed to an insecure platform and may start affecting the whole system. Monitoring and alerting may help identify any network anomalies, which in turn may help remove infections and fix vulnerabilities.

8.2 Security solutions

Containerized systems are usually highly dynamic, thus their security is a continuous process which requires a high level of automation. Therefore, it is necessary to focus our security efforts towards minimizing the attack surface of both the application and the platform. [51]

8.2.1 Limited access

A key security principle for container applications, which can be split into two best practices.

Limited resources: Containerization platforms allow setting maximum usage of the CPU and memory resources for each container. Utilizing these quotas not only makes the application as a whole more efficient by rationing the available resources between the more and less important services, but also minimizes the impact of an infected container, which cannot leverage the full computational resources of the platform for the attacker's purposes.

Limited privileges: The default settings of Docker containers prevent running any process within the container as the root user. This should be respected if at all possible, as it severely limits the options of an attacker, hard countering any attempts at privilege escalation inside of the container.

8.2.2 Secrets

Sensitive data, such as credentials and API keys, should never be stored within the container image or its configuration, where they might be accessible. Instead, a special resource called Secret [52] is normally used, which limits the accessibility of the sensitive information. A dedicated software may also be used to manage Secrets, such as Vault[53].

8.2.3 Monitoring

Due to the aforementioned dynamic nature of containerized applications, monitoring the whole process from image creation to its runtime is essential in mitigating security risks. SAST scans were already mentioned, and other analysis tools may be used on the image itself. Careful logging and alerting on the platform itself will then help ensure that any vulnerabilities are fixed as soon as possible.

8.3 Security recommendations

The Software shall mitigate the image risks as defined in chapter 8.1.1 by using up-to-date libraries and including no passwords, keys, certificates or other security information within the container image itself. In addition, a production versus development switch will be included in the code that can only be changed by re-building the image, with a warning when deploying a development image. Thus the risk of unknowingly deploying a development image can be diminished.

Additionally, the practices from chapter 8.2 shall be followed, especially by using only containers with limited privileges and storing all credentials in a secured database table rather than a plain configuration file.

However, as outlined in the previous two chapters, a containerized application faces multiple security risks, only some of which can be solved in the application itself. As containers are very lightweight and shift a lot of operational load onto the platform where they run, security of a containerized application must follow suit and focus on the security of the platform, too. It is not within the scope of the Project to provide a properly set-up and secure platform and CI/CD infrastructure along with the Software, therefore only recommendations for setting up the platform shall be discussed.

All best practices discussed in chapters 8.1 and 8.2 should be adhered to, but we shall highlight several security features which should definitely be present to allow for smooth and secure functioning of the Software:

- A strict role-based access control (RBAC) should be employed, both for the infrastructure and the platform, thus limiting the chances of the attacker gaining increased privileges.
- An automated CI/CD orchestration should be built, which reduces the risk of human error and makes automated testing of every step of preparing the application possible.
- Automated security scans (SAST, DAST) should be employed both during the development and building phase, and on the platform after deployment.
- The platform should be set up with demilitarized zone (DMZ) and firewalls, to limit its attack surface and prevent access over unexpected channels.
- The Software should be deployed within a separated namespace or subnet of the platform, to prevent cross-contamination should an attack be successful against a part of the platform.

To prevent improper deployment of the Software, only authorized personnel shall have the rights to deploy it, or change its deployment, using RBAC on the deployment infrastructure. Similarly, only authorized personnel shall have access to the configuration tables of the Software, to prevent misconfiguration and wrongful access to the stored credentials. As these are stored in PostreSQL tables, this can be handled by the database's access control.

The Software offers two other ways to access its data, a REST API and a visualization platform. However, only Grafana, the visualization platform, shall be an external-facing service, and shall once again employ RBAC to prevent unauthorized access. The REST API shall not be externally exposed, as it is strictly intended to provide data for other applications on the platform, which is assumed to be secure and only contain intentionally deployed software. Therefore, the REST API shall not require authentication, as is a common practice for internal services on containerized platforms.

9 PROJECT SUSTAINABILITY

In this chapter, we shall discuss the Project's benefits versus risks, and its long-term costs and sustainability.

9.1 Benefits and risks

The Project offers weather data acquirement and storage, forecasting modelling and statistical evaluation of the available data. It is easy to integrate with and be used by other systems of the customer, thanks to the REST API for retrieving the gathered weather data. As such, any project which requires accurate weather data can benefit from the Software; for example smart farming software in agriculture, energy savings software for buildings, solar or wind power plant administration software, planning software in the transport and shipping industry, or in water reservoir regulation.

The Software will be of even greater benefit to any customer that possesses a weather measuring equipment of their own, such as greenhouses or certain industrial complexes, as it can use this real weather data measured directly on site and calibrate its model against it, rather than generic real weather data from nearby meteorological stations, thus both increasing the precision of the predictions, and discovering which of the configured forecasts provides the most accurate data for the particular location.

As far as risks go, the Software requires a containerization platform provided by the customer[1)] and unless the customer purchases a weather data source subscription with historical weather data[2)], it will offer only limited benefits until a sufficient archive of data can be accrued. Additionally, should the Software be marketed for a monthly fee, it is possible that customers will use it only temporarily, to weigh the accuracy of multiple weather forecast services, and then discard the Software once a satisfactory weather forecast service is discovered thanks to the Software's analysis.

9.2 Price

The purchasing price of the Software will not be suggested here, but we will consider the two continuous costs of the Software, the cost of the platform and the costs of the weather data source subscriptions.

[1)] See chapters 5.2 and 9.2.
[2)] Discussed in chapter 6.2.

The Software is designed as a containerized microservice application, any thus has to be deployed on a containerization platform. Docker, Kubernetes or OpenShift were already mentioned as potential platforms, with the best practices and security recommendations for the platform discussed in chapter 8.3. With the more and more frequent usage of PaaS (Platform as a Service), it is relatively easy to get a completely new platform. For example, on Digital Ocean, a virtual machine with Docker can be obtained from 4 $ (85 Kč) per month or a Kubernetes cluster from 12 $ (256 Kč) per month [54], with the setup taking less than an hour. In the best case scenario, the Software can be deployed right away, should the customer already possess a containerization platform.

As for the the weather data source subscriptions, the Software was primarily developed with using free subscriptions in mind. The Software's secondary use is as a weather data storage, to offer a greater range of weather data time series than provided by the weather data services at the free tier, in addition to its primary use for the modelled data. However, it is possible to buy any of the paid subscriptions examined in chapter 6.1 and thus broaden the Software's functionality. In addition, a one-time purchase of historical weather data pack when deploying the Software, such as the ones reviewed in chapter 6.1.13, will boost the immediate benefits of the Software, as it will not need to start gathering data from scratch.

In conclusion, the Software offers immediate integration with existing containerized applications that the customer possesses and its continuous costs can be mostly decided upon by the customer, too. Its immediate benefits can also be increased should the customer choose to do so.

III. TECHNICAL REPORT

10 SOFTWARE COMPONENTS

In this chapter, the components of the Software are examined, including documentation of their functions. A more extensive developer documentation can be found in the Software's code repository[1)].

Based on their function, five main components of the Software can be identified, which are then split into seven microservice containers, as shown in figure 10.1:

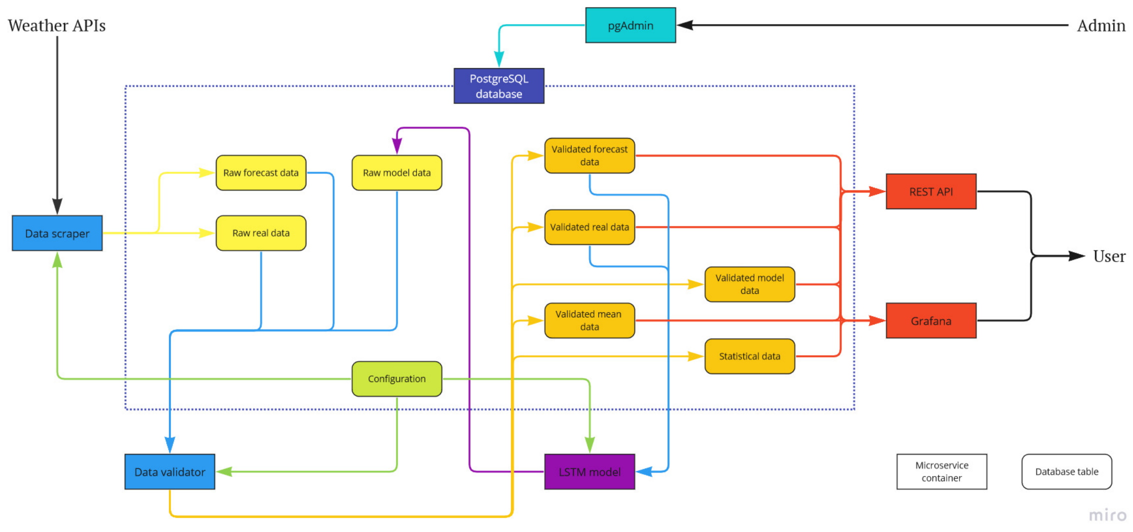


Figure 10.1 Diagram of data flow within the Software[2)]

The five components are:

1. **Data acquisition**, which includes data scraping from weather data services and validation of the acquired data. Shown in blue in figure 10.1.
2. **Data storage** in an SQL database. In figure 10.1, this container is shown in indigo with a dotted expansion encompassing all database tables used to store and process weather data. The three kinds of data (shown in yellow, orange and green) are described below.
3. **Prediction model** - a LSTM neural network. Shown in purple in figure 10.1.
4. **User access** both for other applications via a REST API and for human users via a visualization software. Shown in red in figure 10.1.

1)See appendix A1.
2)Diagram was created using Miro. [55]

5. **Administrator access** to the database. Shown in teal in figure 10.1. This access is also used to manage the configuration of the Software.

Three kinds of data can then be found in the database:

1. **Raw data**, either scraped from external sources or a direct output of the model. Shown in yellow in figure 10.1.
2. **Validated data**, which is used both for modelling and as an output for REST API and visualization. Shown in gold in figure 10.1.
3. **Configuration** of the Software. Shown in green in figure 10.1.

These components will be now discussed in further detail.

10.1 Data acquisition

Weather data for the Software are first scraped from individual weather APIs and then standardized by validating the inputs for later processing.

10.1.1 Data scraping

As mentioned previously[2)], four weather APIs were chosen as sources of weather data for the Software. A free-tier subscription was secured where necessary, providing an API key for each data source. In addition, a location was chosen and configured for the Software - the city of Zlín. Further information on configuring the location for which the Software will gather weather data can be found in chapter 10.5.

The weather data scraper microservice makes an hourly request to the configured weather data sources to retrieve weather data from each data source, and then enters the data into a table of raw weather data. Each data source has its own data scraping method for processing of the data, owning to the differences between the API responses of each data source, however the handling of the scraped data is standardized and thus adding a new data source is as simple as adding a new data scraping method.

For an example of how to configure and add a new weather data source, the developer documentation in the Software's code repository should be consulted.

[2)]See chapter 6.2.

10.1.2 Data validation

It is almost always a good practice to standardize the data before modelling it using a neural network model, as neural networks can easily run into problems when dealing with inconsistent data - missing values or erroneous, outlying values will affect the learning process of the network and may have a serious negative impact on the network's results. In addition, as we are processing data from multiple sources, inconsistencies between the sources are to be expected and should be rectified before we enter the modelling stage. [4]

Just as data scraping, data validation runs hourly, when all forecast, real weather and modelled data are processed. Forecast and real weather data are mainly standardized. Missing data are replaced with a mean of values across all (other) sources at the given time, using a zero value if no other values with the given timestamp exist. Real weather data also has its timestamps rounded to the hour, as an hourly forecast is used in the Software.

Additionally, irradiance is calculated from the scraped values, using cloudiness, the configured location and the timestamp of the given data sample. The calculation uses astronomical computations based on the ephem library. [17]

Validation also provides mean weather forecast data, including the minimum and maximum temperature across data sources, and irradiation calculated from the mean weather data. Mean weather forecast data will be used as one of the points of comparison against the modelled data.

Modelled data are also validated, as raw modelled data can include data points in excess of the viable range of individual metrics, such as cloud fraction higher than 100% or negative humidity, which need to be limited to the appropriate range of values.

Finally, statistical data for each weather forecast data source are processed as a part of data validation, calculating their difference against real weather data. Four statistical metrics are thus prepared: root mean squared error, root mean squared percentage error, R^2 and adjusted R^2. These statistical metrics will be used to compare the credibility of the weather forecast data sources in chapter 12.

10.2 Data storage

All data are stored in a PostreSQL database. The Software components use a SQL-Alchemy interface to access the data, but an outside access via visualization or a REST API is also possible[3]. In addition, a direct access to the database for development or administrative purposes is provided using pgAdmin [56], a PostreSQL administration tool.

10.3 Neural network model

A LSTM network was prepared for training a model of relations between weather forecast and real weather at a location, using keras interface [10] and TensorFlow platform [24] to power the neural network. In an hourly cycle, the model is re-trained on a newly acquired data set (the volume of stored validated data will be steadily increasing as the Software is left running, thus expanding the training set of the model and therefore its accuracy) and a prediction is made for all available forecast data.

The input data are combined into pairs of forecast and real weather data at each time point, using each forecast data source independently to increase the number of training points and to find possible common relations between the data. Non-numeric data (data sources and wind direction descriptions) are encoded as integer flags. Afterwards, the data are normalized to the scale of zero to one. Finally, the data set is split into a training and testing set with a seventy-five to twenty-fire percent ratio between the former and the latter.

A model is defined and fitted with the training set, then validated with the testing set. This model is then used to make prediction for the future forecast data where real weather data are yet unknown. Output data are then inversely scaled to revert the normalization.

The network uses one dense layer, with the optimization algorithm and loss function chosen by comparing the performance of available combinations of these, as discussed in chapter 7.

[3] See chapter 10.4 below.

10.4 User access

Two access points, for regular users and for other applications, were prepared. The latter, however, is the primary use case of the Software - REST API as a source of weather data.

10.4.1 REST API

The REST API allows access to the data gathered and processed by the Software. Several endpoints are prepared:

```
/info
/v1/weather
/health
```

`/info` returns basic information about the application.

`/v1/weather` is the primary endpoint for requesting weather data. The arguments with which to call the endpoint are further described in the developer documentation in the Software's code repository. Also see appendix A2 for an example request and response of this endpoint.

`/health` offers responses for startup, liveness and readiness probes [57]. The API will return HTTP status code 200 if the check is processed with no problems.

10.4.2 Visualization

A dashboard with an overview of collected weather data and their comparison was prepared in Grafana4), which shows the time series of individual metrics as gathered from all data sources, both external (weather APIs) and internal (mean and modelled data), and the associated statistical data. This can help a user in quickly familiarizing themselves with the data available and also with the general trends of the weather metrics over time.

More dashboards can easily be defined by the user as necessary, Grafana being designed for exactly such a use.

4)See chapter 5.4.

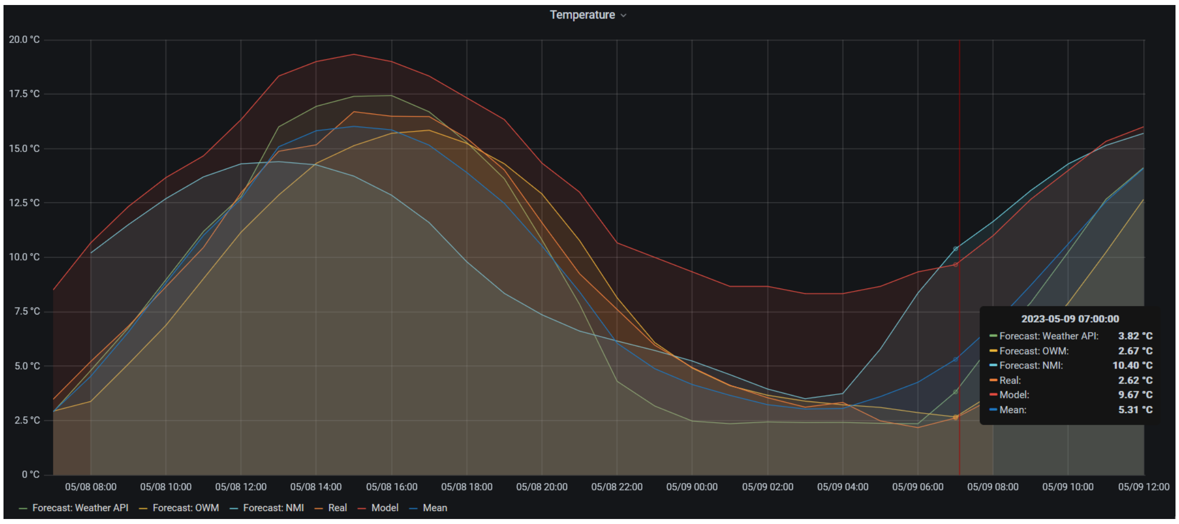


Figure 10.2 Temperature comparison panel from a Grafana dashboard

10.5 Configuration

When setting up a new instance of the Software, configuration is required. First, an environment file needs to be created for the deployment, and then configuration tables in the database need to be filled in.

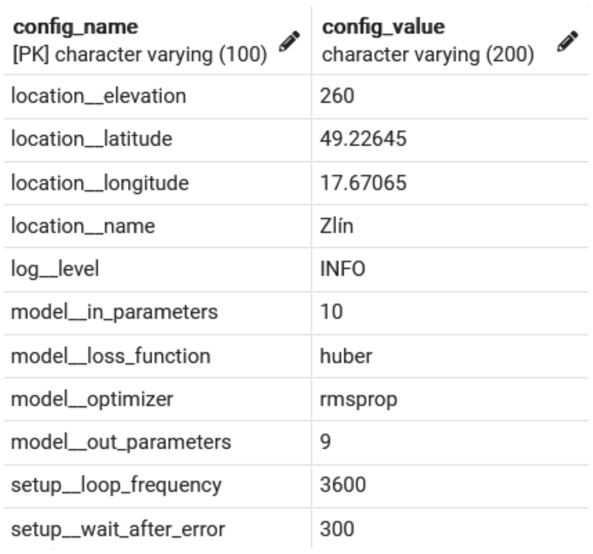
10.5.1 Environment file

An example environment file is provided in appendix A3 and further explanation can be found in the developer documentation in the Software's code repository.

10.5.2 Configuration tables

The Software uses two configuration tables in the database, one for all containers and one for the data scraper, where the weather data sources are specified. The tables can be changed by any user with sufficient database privileges, using the database's RBAC. As these tables both contain Secrets (API keys) and can prevent the application from functioning properly if misconfigured, it is recommended than only administrator users have access to the configuration.

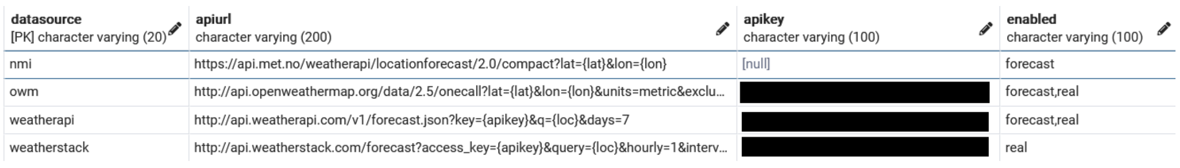
The **application configuration** contains `key value` pairs, which are loaded and parsed when the application is starting. See figure 10.3 for an example of a filled-in configuration table with the location for weather data gathering being set to the city of Zlín. Only a single location is allowed per application instance in the current version



config_name [PK] character varying (100)	config_value character varying (200)
location__elevation	260
location__latitude	49.22645
location__longitude	17.67065
location__name	Zlín
log__level	INFO
model__in_parameters	10
model__loss_function	huber
model__optimizer	rmsprop
model__out_parameters	9
setup__loop_frequency	3600
setup__wait_after_error	300

Figure 10.3 Example application configuration, as found in the database

of the Software, though a single application instance monitoring multiple locations is a planned feature for later development[5)].



datasource [PK] character varying (20)	apiurl character varying (200)	apikey character varying (100)	enabled character varying (100)
nmi	https://api.met.no/weatherapi/locationforecast/2.0/compact?lat={lat}&lon={lon}	[null]	forecast
owm	http://api.openweathermap.org/data/2.5/onecall?lat={lat}&lon={lon}&units=metric&exclu...	[redacted]	forecast,real
weatherapi	http://api.weatherapi.com/v1/forecast.json?key={apikey}&q={loc}&days=7	[redacted]	forecast,real
weatherstack	http://api.weatherstack.com/forecast?access_key={apikey}&query={loc}&hourly=1&interv...	[redacted]	real

Figure 10.4 Example configuration of weather data sources, as found in the database, with the API keys censored for security purposes

The **API configuration** is used by the data scraper only, as it contains the information needed for requesting data from weather data sources. Figure 10.4 shows an example API configuration.

`datasource` is the name of each weather data source.

`apiurl` contains the URL for requesting data from the given weather API.

`apikey` contains the API key for each data source that requires one. The API key is used to authenticate the application to the weather data source server, making it a sensitive information, and thus is is censored in the figure.

`enabled` is used to define which type of data, "forecast" or "real" (current measured data), should be scraped from each data source. As some weather data sources offer only one type of data, it is best to disable the data types not supported by the source

[5)]See chapter 11.

to prevent unnecessary warnings. This setting can also be used to disable all but one source of real weather data, for example when the user has a local weather measuring station against which they wish to calibrate the model and statistical data.

Further explanation and examples of configuration tables can be found in the developer documentation in the Software's code repository.

11 FURTHER DEVELOPMENT

For further development, these features can be recommended:

- **Best weather forecast**, which would retrieve forecast data separately for each meteorological parameter from a different weather data source based on the best fit as determined by the statistical metrics for each parameter. See chapter 12 for more information on the statistical metrics which would be used in implementing this endpoint.
- **Multiple locations**. As previously mentioned, the current version of the Software supports monitoring only one location per application instance. However, making it possible to configure multiple locations on a single application instance would improve user experience, as users might wish to collect data from more than a single location and running a separate instance for each location increases both resource and management overhead.
- **Performance evaluation** of different machine learning techniques, apart from the already implemented LSTM model, could be realized by adding new modelling modules to the Software. That way, a more in-depth study of machine learning techniques in weather forecasting could be prepared.
- **Prometheus metrics**, while a less significant feature, would allow easier integration into containerization infrastructure and monitoring. In the current version of the Software, only health probes are available1), but more detailed metrics as provided by Prometheus would help in diagnosing both possible malfunctions and security threats2) in a deployed application.

These features were planned for the Software, but were not implemented due to the time constraints of the Project.

1)See chapter 10.4.1.
2)See chapter 8.3.

12 CREDIBILITY EVALUATION

In this chapter, the statistical metrics produced by the Software will be examined with an eye towards finding the most reliable weather data source. The statistical metrics are calculated by taking the forecast data for each meteorological parameter and each data source, then comparing them against the real weather data. The following four metrics were chosen:

- **Root mean squared error** (RMSE) is the standard deviation of prediction errors. In other words, it measures the difference between the predicted values and the observed values, with larger RMSE meaning larger mean error of the prediction; RMSE of zero would indicate a perfect fit [58]. Therefore, the aim is to minimize RMSE. See table 12.3 for an overview of RMSE for individual meteorological parameters.
- **Root mean squared percentage error** (RMSPE) is similar to RMSE, except normalized to the percentage scale. Where RMSE of different parameters is incomparable, because RMSE values for each parameter depend on the scale of the parameter, RMSPE can be used for comparisons even across various parameters. Therefore, it is the best metric for easily judging the reliability of a model across all its parameters. See table 12.2 for an overview of RMSPE for individual meteorological parameters.
- $\mathbf{R^2}$, or the coefficient of determination, is a measure of how well a model predicts the observed values, based on the proportion of outcomes explained by the model[59]. The coefficient normally ranges from zero to one, where R^2 of one would mean a perfect fit, but negative values are possible when the mean of the data would provide better results than the modelled function [60]. Therefore, the aim is to maximize R^2. See table 12.4 for an overview of R^2 for individual meteorological parameters.
- **Adjusted $\mathbf{R^2}$** is often used in place of basic R^2 when there are multiple input variables in the model. Where R^2 always increases with added input variables, adjusted R^2 decreases when a new input variable is added that doesn't contribute to the precision of the prediction. It is therefore important in determining the best number of inputs for a model [59]. See table 12.5 for an overview of adjusted R^2 for individual meteorological parameters.

An average value for each metric was then also determined. See table 12.1 for a comparison across all data sources and statistical metrics, ordered by RMSPE from the lowest deviation to the highest.

Table 12.1 Average statistical metrics

Data source	Average metrics through all parameters			
	RMSPE (%)	RMSE	R^2	R^2 adjusted
Mean	10.8	5.85	0.67	0.67
Weather API	11.3	5.86	0.60	0.59
Model	12.5	5.92	0.57	0.56
OWM	12.8	7.18	0.51	0.49
NMI	13.3	7.83	0.40	0.39

Differences can be observed in the reliability of prediction of different meteorological parameters by different weather data sources, but certain parameters (especially cloud fraction) suffer from a higher prediction error in general. The user might therefore wish to take the best performing source of prediction for each parameter, rather than relying on a single data source for all parameters. Implementing an automated way of retrieving such "best performing" prediction can be recommended for the next step of developing the Software[1)].

As seen in table 12.1, mean weather forecast offers the overall best precision of prediction across all statistical metrics, with Weather API placing second and the neural network weather model only at the third rank. In addition, certain parameters exhibit negative R^2, which means that the forecast values do not fit the real weather data well and it would be better to use mean rather than that model [60].

However, it is necessary to note that the behaviour of the mean weather forecast in response to spikes in the parameter values, as often seen in real weather, was not scrutinized. It is the author's personal worry that the mean of weather data will not work well with short, sudden changes in meteorological parameters. While some data sources might correctly predict the spike, averaging the values from multiple sources will likely prevent rapid changes from manifesting in the mean weather data time series, making the predictions of such changes inaccurate. However, sufficient analysis was not performed to confirm this effect, and thus mean weather data remains as the most reliable source of weather data.

It should also be noted that many parameters have not scored particularly well in the R^2 test. On the other hand, adjusted R^2 does not exhibit larger losses when compared to R^2, meaning that the input data are relevant to the model's output.

1)See chapter 11.

As previously mentioned, the neural network weather model placed third in overall prediction precision, so it cannot be recommended for immediate usage. Interestingly, its performance was unrivaled for cloud fraction, where other data sources suffered from lower accuracy while the neural network weather model achieved a significantly better precision in comparison. In addition, as the model can be retrained with larger data set every time new weather data are received, it can be expected that its accuracy will improve as the Software remains deployed for some time, or if a historical set of forecast and real weather data is purchased[2)].

An additional way of increasing the accuracy of the model and the statistical analysis of weather data sources in general can be easily implemented by a user who has access to weather data gathering equipment at their location. Should the Software work with locally measured weather data rather than generic real weather data, its accuracy for given location would be much improved. While this function could not be tested during development, this was one of the main planned benefits of the Software.

[2)] See for example chapter 6.1.13 for one source of such data.

[3)] Also known as the "felt air temperature" or "apparent temperature", but as all weather data sources used in the Project refer to this parameter as "feels like", this name will be employed.

Table 12.2 Root mean square percentage error (%)

Data source	T	FL	CF	UVI	WS	H	PPT	P
Mean	5.4	6.6	28.4	9.5	13.3	12.1	8.6	2.8
Model	13.1	13.1	23.5	10.5	14.8	11.1	9.2	5.0
NMI	6.8	6.0	38.3	4.9	20.4	15.8	10.7	4.0
OWM	5.6	6.2	37.2	16.3	14.5	13.4	6.4	2.6
Weather API	5.6	7.3	27.9	13.7	13.0	12.5	6.8	3.2

For tables 12.2 to 12.5, the following abbreviations are used:
T: Temperature, FL: Feels like[3)]*, CF: Cloud fraction, UVI: UV index,*
WS: Wind speed, H: Humidity, PPT: Precipitation, P: Pressure

Table 12.3 Root mean square error

Data source	T	FL	CF	UVI	WS	H	PPT	P
Mean	1.93	1.50	28.46	0.49	3.51	9.40	0.36	1.11
Model	4.66	3.09	23.52	0.56	3.95	8.68	0.39	2.06
NMI	2.72	1.14	38.34	0.26	5.37	12.78	0.45	1.61
OWM	2.03	1.41	37.16	0.93	3.85	10.48	0.55	1.05
Weather API	2.00	1.71	27.97	0.72	3.50	9.71	0.44	1.28

Table 12.4 R^2

Data source	T	FL	CF	UVI	WS	H	PPT	P
Mean	0.90	0.93	0.31	0.86	0.58	0.71	0.12	0.96
Model	0.39	0.70	0.53	0.82	0.47	0.75	-0.01	0.87
NMI	0.84	0.94	-0.14	0.95	0.05	0.51	-0.04	0.93
OWM	0.88	0.94	-0.17	0.51	0.50	0.64	-1.05	0.97
Weather API	0.89	0.91	0.34	0.71	0.59	0.69	-0.29	0.95

Table 12.5 R^2 adjusted

Data source	T	FL	CF	UVI	WS	H	PPT	P
Mean	0.89	0.93	0.30	0.86	0.57	0.71	0.10	0.96
Model	0.38	0.70	0.52	0.82	0.46	0.75	-0.03	0.87
NMI	0.84	0.94	-0.18	0.95	0.02	0.50	-0.07	0.93
OWM	0.88	0.94	-0.19	0.50	0.49	0.63	-1.08	0.97
Weather API	0.89	0.91	0.33	0.70	0.58	0.69	-0.31	0.95

CONCLUSION

This master's thesis aimed to implement utility software for weather data acquisition, processing and prediction. An overview of available weather data APIs was prepared[3)] and data sources were recommended for the Software's use[4)]. The Software's architecture[5)] and security[6)] were discussed and possible neural network configurations were tested[7)] to find the best fit for weather data modelling. The Project's methodology[8)] and sustainability[9)] was also touched upon. The Software's implementation was then described[10)] and the statistical metrics it produces were evaluated[11)]. Finally, the future of the Software's development was suggested[12)]

The Software attached to this thesis is a fully functional proof of concept which can be further expanded upon should the user need more or other features. The Software offers easy access to multiple weather data sources, including historical data storage for sources which do not offer such, or only have historical data as a part of a paid plan. In addition, the statistical metrics presented by the Software allow comparison of individual data sources. New weather data sources can be readily added to the Software's configuration, and should the user possess a local meteorological measuring equipment, the Software can calibrate its metrics against real weather data of the location. The Software may facilitate an in-depth statistical analysis of different weather forecasting models, which was unfortunately not within the scope of this Project.

[3)] See chapter 6.1.
[4)] See chapter 6.2.
[5)] See chapter 4.
[6)] See chapter 8.3.
[7)] See chapter 7.
[8)] See chapter 2.
[9)] See chapter 9.
[10)] See chapter 10.
[11)] See chapter 12.
[12)] See chapter 11.

REFERENCES

[1] Hyndman, R. J.; Athanasopoulos, G.: *Forecasting: principles and practice*. Melbourne, Australia: O-texts, third print edition, 2021, ISBN 978-0-9875071-3-6.

[2] McConnell, S.: *Code complete: a practical handbook of software construction*. Redmond, Wash: Microsoft Press, 1993, ISBN 978-1-55615-484-3.

[3] Waterfall Methodology. *Adobe Workfront*, visited: 2023-05-10.
URL `https://business.adobe.com/blog/basics/waterfall`

[4] Data Preparation for Machine Learning Projects: Know It All Here. *ProjectPro*, visited: 2023-05-10.
URL `https://www.projectpro.io/article/data-preparation-for-machine-learning/595`

[5] Wolpert, D. H.: The Supervised Learning No-Free-Lunch Theorems. In *Soft Computing and Industry*, edited by R. Roy; M. Köppen; S. Ovaska; T. Furuhashi; F. Hoffmann, London: Springer London, 2002, ISBN 978-1-4471-1101-6 978-1-4471-0123-9, pp. 25–42, doi:10.1007/978-1-4471-0123-9_3.
URL `http://link.springer.com/10.1007/978-1-4471-0123-9_3`

[6] Fente, D. N.; Singh, D. K.: Weather Forecasting Using Artificial Neural Network. In *Proceedings of the 2018 Second International Conference on Inventive Communication and Computational Technologies (icicct)*, New York: Ieee, 2018, ISBN 978-1-5386-1974-2, pp. 1757–1761, wOS:000456251700349.
URL `https://www.webofscience.com/wos/woscc/full-record/WOS:000456251700349`

[7] Lang, N.: An Introduction to Long Short-Term Memory Networks (LSTM). *Medium*, October 2022, visited: 2023-05-10.
URL `https://towardsdatascience.com/an-introduction-to-long-short-term-memory-networks-lstm-27af36dde85d`

[8] Wang, C.-F.: The Vanishing Gradient Problem. *Medium*, January 2019, visited: 2023-05-10.
URL `https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484`

[9] A Guide to Long Short Term Memory (LSTM) Networks. *Knowledge Hut*, visited: 2023-05-10.
URL `https://www.knowledgehut.com/blog/web-development/long-short-term-memory`

[10] Deep Learning for humans. *Keras*, visited: 2023-05-10.
URL `https://keras.io/`

[11] Containerization Explained. *Amazon Web Services*, visited: 2023-04-04.
URL `https://aws.amazon.com/what-is/containerization/`

[12] What is containerization? *RedHat*, visited: 2023-04-04.
URL `https://www.redhat.com/en/topics/cloud-native-apps/what-is-containerization`

[13] A Practical Guide to Choosing between Docker Containers and VMs. *Weave-Works*, visited: 2023-05-10.
URL `https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms`

[14] What are microservices? *RedHat*, visited: 2023-04-04.
URL `https://www.redhat.com/en/topics/microservices/what-are-microservices`

[15] Microservices vs. Monolith Architecture. *DEV Community*, December 2018, visited: 2023-05-10.
URL `https://dev.to/alex_barashkov/microservices-vs-monolith-architecture-4l1m`

[16] What is REST. *REST API Tutorial*, April 2022, visited: 2023-05-20.
URL `https://restfulapi.net/`

[17] PyEphem.
URL `https://rhodesmill.org/pyephem/`

[18] Carroll, B. W.; Ostlie, D. A.: *An introduction to modern astrophysics*. Cambridge New York Melbourne: Cambridge University Press, second edition edition, 2017, ISBN 978-1-108-42216-1, doi:10.1017./9781108380980.

[19] Flask.
URL `https://flask.palletsprojects.com/en/2.3.x/`

[20] NumPy.
URL `https://numpy.org/`

[21] pandas - Python Data Analysis Library.
URL `https://pandas.pydata.org/`

[22] scikit-learn: machine learning in Python.
URL `https://scikit-learn.org/stable/`

[23] SQLAlchemy - The Database Toolkit for Python.
URL `https://www.sqlalchemy.org/`

[24] TensorFlow.
URL `https://www.tensorflow.org/`

[25] Docker Desktop. *Docker*, October 2021, visited: 2023-05-10.
URL `https://www.docker.com/products/docker-desktop/`

[26] Docker Compose overview. *Docker Documentation*, May 2023, visited: 2023-05-10.
URL `https://docs.docker.com/compose/`

[27] Helm charts. *Helm - The Kubernetes Package Manager.*, visited: 2023-05-10.
URL `https://helm.sh/`

[28] DB Ranking. *DB-Engines*, visited: 2023-05-10.
URL `https://db-engines.com/en/ranking`

[29] Grafana: The open observability platform. *Grafana Labs*, visited: 2023-05-10.
URL `https://grafana.com/`

[30] AccuWeather API.
URL `https://developer.accuweather.com/`

[31] Azure Maps. *Microsoft Azure.*
URL `https://azure.microsoft.com/en-gb/pricing/details/azure-maps/`

[32] Julmar, M.: Weather REST API. *Microsoft Azure*, visited: 2023-04-04.
URL `https://learn.microsoft.com/en-us/rest/api/maps/weather`

[33] Portál ČHMÚ.
URL `https://www.chmi.cz/historicka-data/pocasi/`

[34] in-Počasí: archiv.
URL `https://www.in-pocasi.cz/archiv/`

[35] Meteoblue - weather close to you.
URL `https://www.meteoblue.com/en/weather/`

[36] Meteocentrum.cz.
URL `https://www.meteocentrum.cz/archiv-pocasi`

[37] Weather API. *Meteomatics.*
URL `https://www.meteomatics.com/en/weather-api/`

[38] Meteosource.
URL `https://www.meteosource.com/`

[39] Meteostat. *GitHub.*
URL `https://github.com/meteostat`

[40] Location Forecast. *Meteorologisk institutt.*
URL `https://api.met.no/weatherapi/locationforecast/2.0/documentation`

[41] Terms of Service. *Meteorologisk institutt.*
URL `https://api.met.no/doc/TermsOfService`

[42] Stormglass.io.
URL `https://stormglass.io/`

[43] Tomorrow.io.
URL `https://www.tomorrow.io/`

[44] Weather and forecast. *OpenWeatherMap.*
URL `https://openweathermap.org/`

[45] Weather API.
URL `https://www.weatherapi.com/`

[46] WeatherKit. *Apple Developer.*
URL `https://developer.apple.com/weatherkit/`

[47] Real-Time World Weather REST API. *Weatherstack.*
URL `https://weatherstack.com/`

[48] World Weather Online.
URL `https://www.worldweatheronline.com/`

[49] Souppaya, M.; Morello, J.; Scarfone, K.: Application container security guide. Technical report NIST SP 800-190, National Institute of Standards and Technology, Gaithersburg, MD, September 2017, doi:10.6028/NIST.SP.800-190.
URL `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf`

[50] Risks and Best Practices for Container Security. *Veritis*, visited: 2023-04-04.
URL `https://www.veritis.com/blog/container-security-an-overview-of-risks-and-security-best-practices/`

[51] 7 Container Security Best Practices. *CrowdStrike*, visited: 2023-04-04.
URL `https://www.crowdstrike.com/cybersecurity-101/cloud-security/container-security-best-practices/`

[52] Secrets. *Kubernetes*, visited: 2023-04-04.
URL `https://kubernetes.io/docs/concepts/configuration/secret/`

[53] Vault. *HashiCorp Developer*, visited: 2023-04-04.
URL `https://developer.hashicorp.com/vault`

[54] Pricing Overview. *DigitalOcean*, visited: 2023-05-10.
URL `https://www.digitalocean.com/pricing`

[55] The Visual Collaboration Platform for Every Team. *Miro*, visited: 2023-05-10.
URL `https://miro.com/`

[56] pgAdmin - PostgreSQL Tools.
URL `https://www.pgadmin.org/`

[57] Configure Liveness, Readiness and Startup Probes. *Kubernetes*, visited: 2023-05-18.
URL `https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/`

[58] RMSE: Root Mean Square Error. *Statistics How To*, visited: 2023-05-18.
URL `https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/`

[59] R-Squared vs. Adjusted R-Squared: What's the Difference? *Investopedia*, visited: 2023-05-18.
URL `https://www.investopedia.com/ask/answers/012615/whats-difference-between-rsquared-and-adjusted-rsquared.asp`

[60] Colin Cameron, A.; Windmeijer, F. A.: An R-squared measure of goodness of fit for some common nonlinear regression models. *Journal of Econometrics*, volume 77, no. 2, April 1997: pp. 329–342, ISSN 03044076, doi:10.1016/S0304-4076(96)01818-0.
URL `https://linkinghub.elsevier.com/retrieve/pii/S0304407696018180`

LIST OF ABBREVIATIONS

API	Application Programming Interface
CI/CD	Continuous Integration, Continuous Deployment
CSV	Comma-Separated Values file format
DAST	Dynamic Application Security Testing
DMZ	Demilitarized Zone
HTTP	Hypertext Transfer Protocol
LSTM	Long Short-Term Memory neural network model
NWI	Norwegian Weather Institute
OWM	OpenWeatherMap
PaaS	Platform as a Service
R^2	Coefficient of Determination
RBAC	Role-Based Access Control
REST	Representational State Transfer
RMSE	Root Mean Squared Error
RMSPE	Root Mean Squared Percentage Error
RNN	Recurrent Neural Network
SAST	Static Application Security Testing

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

APPENDIX A I. SOFTWARE CODE REPOSITORY

The Software's code repository is attached to this thesis on a portable data medium.

APPENDIX A II. EXAMPLE OF REST API REQUEST AND RESPONSE

Call the API with the following GET request:

```
weather_api:5051/v1/weather?kind=forecast&source=nmi
  &date_from=2022-11-01&date_to=2022-11-01
```

Response on success:

```
{
  "args": {
    "date_from": "2022-11-01",
    "date_to": "2022-11-01",
    "kind": "forecast",
    "source": "nmi"
  },
  "data": {
    "2022-11-01T00:00:00.000Z": {
      "chance_rain": 0,
      "chance_snow": 0,
      "cloud_fraction": 100,
      "datasource": "nmi",
      "feels_like": 0,
      "humidity": 95,
      "location": "Zlín",
      "precipitation": 0,
      "pressure": 1020.7,
      "temperature": 10.7,
      "uv_index": 0,
      "validated_time": "2023-05-14T07:30:16.902Z",
      "wind_dir": "SE",
      "wind_speed": 11.88
    }
  },
  "stats": {
    "r2": {
      "cloud_fraction": -0.1408078122556371,
      "feels_like": 0.9411633833625154,
      "humidity": 0.5132157449092919,
      "precipitation": -0.03573661032437436,
      "pressure": 0.9335580499506693,
      "temperature": 0.8427505396758244,
      "total": 0.5070296355144298,
      "uv_index": 0.9498726626146752,
      "wind_speed": 0.05222112618247399
    },
    "r2adj": {
      "cloud_fraction": -0.17747663479242548,
      "feels_like": 0.9392722063991676,
      "humidity": 0.49756910813851907,
      "precipitation": -0.06902814422765768,
      "pressure": 0.9314224158419407,
      "temperature": 0.837696092736833,
      "total": 0.49118415951310795,
      "uv_index": 0.9482614267701469,
      "wind_speed": 0.02175680523833934
    },
    "rmse": {
      "cloud_fraction": 38.28540920887064,
      "feels_like": 1.1400623261294498,
      "humidity": 12.825030333379253,
      "precipitation": 0.44862521040038594,
      "pressure": 1.6034419857355005,
      "temperature": 2.7264194461659614,
      "total": 7.831809557316093,
      "uv_index": 0.2525413991530252,
      "wind_speed": 5.372946548694527
    },
    "rmspe": {
      "cloud_fraction": 0.38285409208870635,
      "feels_like": 0.05977257913995718,
      "humidity": 0.15833370781949693,
      "precipitation": 0.10681552628580616,
      "pressure": 0.040086049643387425,
      "temperature": 0.06765308799419259,
      "total": 0.13346279852915238,
      "uv_index": 0.048512343415784205,
      "wind_speed": 0.20367500184588805
    }
  },
  "version": {
    "api": "v1",
    "handler": "0.1.2"
  }
}
```

Response on failure:

```
{
  "args": {
    "date_format": "other",
    "date_from": "1999-11-01",
    "date_to": "1999-13-01",
    "kind": "model",
    "source": "nmi"
  },
  "data": {},
  "errors": {
    "0": {
      "message": "Filtering by source not allowed for model data."
    },
    "1": {
      "error": "Both dates must have the same UTC offset",
      "message": "Failed to filter by time."
    },
    "2": {
      "message": "Date format other is not allowed."
    }
  },
  "stats": {},
  "version": {
    "api": "v1",
    "handler": "0.1.2"
  }
}
```

APPENDIX A III. EXAMPLE DOCKER .ENV FILE

This example `.env` file needs to have all "placeholder" entries changed to usernames and passwords defined by the user before deployment.

`.env`

```
# Environment variables file
# ==========================
# Any environment variables declared here will be passed to each pod.

# PostgreSQL setup:
POSTGRES_USER="placeholder"
POSTGRES_PASSWORD="placeholder"
POSTGRES_DB="weather"

# pgAdmin setup:
PGADMIN_DEFAULT_EMAIL="placeholder@pgadmin.cz"
PGADMIN_DEFAULT_PASSWORD="placeholder"

# Timezone correction (Etc/UTC -> Europe/Prague)
TZ="Europe/Prague"

# Grafana:
GF_SECURITY_ADMIN_USER="placeholder"
GF_SECURITY_ADMIN_PASSWORD="placeholder"

# Database connection:
AK8PO_USER="placeholder"
AK8PO_PWD="placeholder"
AK8PO_HOST="weather_postgres"
AK8PO_NAME="weather"
```