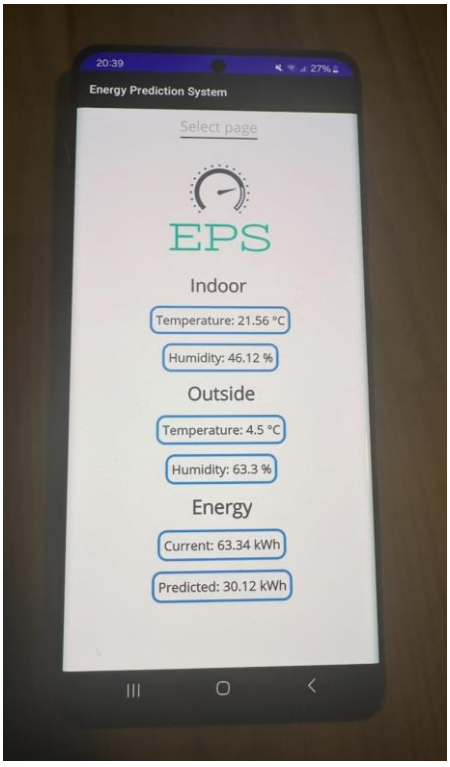
University of South-Eastern Norway

www.usn.no

FM4017 Project 2024

AI driven Development of Open-source, Cross-platform Mobile Applications for Sensor Data Monitoring and Analysis



MP-10-24

The University of South-Eastern Norway takes no responsibility for the results and conclusions in this student report.

University of South-Eastern Norway

www.usn.no

Course: FM4017 Project, 2024

Title: AI-Driven Development of Open-Source, Cross-Platform Mobile Apps for Sensor Data Monitoring and Analysis

Project group: MP-10-24

Group participants: Ørjan Ingebrigtsen
Kim Langvannskås
Aleksander Wad Holthe

Supervisors: Hans-Petter Halvorsen, Saba Mylvaganam, USN, Dept. EE, IT and Cybernetics

External partners:

Summary:

Today almost everyone has access to a mobile device and uses it to communicate, access information, conduct different tasks, and more. Artificial intelligence (AI) is increasingly used as a tool to automate processes and analyze data effectively. To combine mobile application and AI to analyze sensor data is therefore an important topic.

The object of this project was to investigate cross-platform development tool for mobile application development and AI models to enable the development of a mobile application for monitoring sensors in a building, retrieve weather data, and make energy prediction. To be able to do this, energy simulations, parameters used for simulation, and online sources for weather data also had to be investigated and building data collected.

In the report the development of the mobile application is described together with the development of the database and AI model web APIs.

In the results chapter the first version mobile application user interface is presented. It is there shown the functionality of the mobile application.

A mobile application using AI can significantly enhance the user experience and more complex or demanding tasks can be solved. By using cloud services and APIs, the computing requirements of the mobile device is reduced.

Preface

The idea behind this project is the increasing use of mobile devices and mobile applications in almost all elements of life. Today we use the mobile to communicate by phone or e-mail, social media, work and education, and for many other applications and tasks. Mobile devices have replaced or are replacing many older technologies, and new uses for them are continuously developing. We also see that artificial intelligence (AI) is now more commonly used to automate processes, analyze data, recognize patterns, and other tasks. If you read a news article today it frequently states that it is using AI to generate or to help write the article.

During the first part of the project a project description [1] was made by the project supervisor in collaboration with the project team. We there focused the scope and tasks of the project. The core of the project is the development of a mobile application that uses sensors in a building, weather data and forecasts, and AI models to predict the energy consumption of the building for the next period. To enable these studies of technologies and methods is needed. The pace of the project and the limitations in time have limited the time available to develop the mobile application, so further development is needed to have a final application ready to be launched commercially.

During the project, the project supervisors Hans-Petter Halvorsen and Saba Mylvaganam have been an appreciated resource and support. They have given feedback and helped guide us in our work on the project. We would like to thank them for their support.

Porsgrunn, 17.11.2024

Table of Contents

Nomenclature

AI	Artificial Intelligence
API	Application programming interface
ARIMA	Auto-regressive integrated moving average
AutoML	Automated machine learning
CNN	Convolutional neural networks
CSV	Comma-separated values
EPB	Energy performance of buildings
IDE	Integrated development environment
iOS	iPhone operating system
ISO	International Organization for Standardization
JSON	JavaScript object notation
LRM	Linear regression models
LSTM	Long short-term memory networks
MAUI	Multi-platform (cross-platform) application user interface
MET (Norway)	Norwegian Meteorological Institute
ML	Machine learning
NLP	Natural language processing
NS	Norwegian standard
NSPEK	Norwegian specification
OS	Operating system
RNN	Recurrent neural networks
SDK	Software development kit
SQL	Structured query language
UI	User interface
UP	Unified software development process
URL	Uniform resource locator
WinUI	Windows user interface
XAML	Extensible application markup language

1 Introduction to mobile application development

In this project the development of an app for mobile phone that presents current and historical data for indoor environmental data, energy consumption, weather data, and prediction of future energy consumption is studied. The project includes selection of tools and data sources. It also includes the use of Artificial Intelligence (AI) to predict energy consumption in the coming period.

1.1 Background for mobile application development project

As described in the project description [1] the background for this project is the pivotal role of mobile phones and apps in modern technology.

"In today's world, mobile phones and apps are at the center of technological advancements. The two dominant platforms for smartphones are Apple's iOS and Google's Android. Numerous tools are available for developing mobile applications, but one of the most transformative is AI. How can AI be utilized and integrated into the development of contemporary mobile applications?

AI can enhance mobile app development in several ways:

- Personalization: AI algorithms can analyze user behavior to provide personalized experiences, making apps more engaging and user-friendly.
- Automation: AI can automate repetitive tasks, improving efficiency and reducing development time.
- Predictive Analytics: By analyzing data, AI can predict user needs and preferences, allowing developers to create more intuitive and responsive apps.
- Enhanced Security: AI can detect and respond to security threats in real-time, ensuring user data is protected.
- Natural Language Processing (NLP): Integrating NLP allows for advanced features like voice recognition and chatbots, enhancing user interaction.

By leveraging AI, developers can create smarter, more efficient, and highly personalized mobile applications that meet the evolving needs of users."

Above is cited directly from project description by Halvorsen, Mylvaganam, Holte, Langvannskås, and Ingebrigtsen [1].

1.2 Objectives for mobile application development project

The objective of this project is to develop a mobile application for mobile phones that can present current and historical indoor and outdoor environmental data, energy consumption, and prediction of energy consumption in the coming period based on these data with the help of AI. The mobile application will present data from a cloud-based test database, online weather data source, and the AI prediction model. The mobile application should be available

both for iPhone operating system (iOS) and Android mobile operating system (OS). In figure 1.1 you see a system sketch of the energy prediction system.

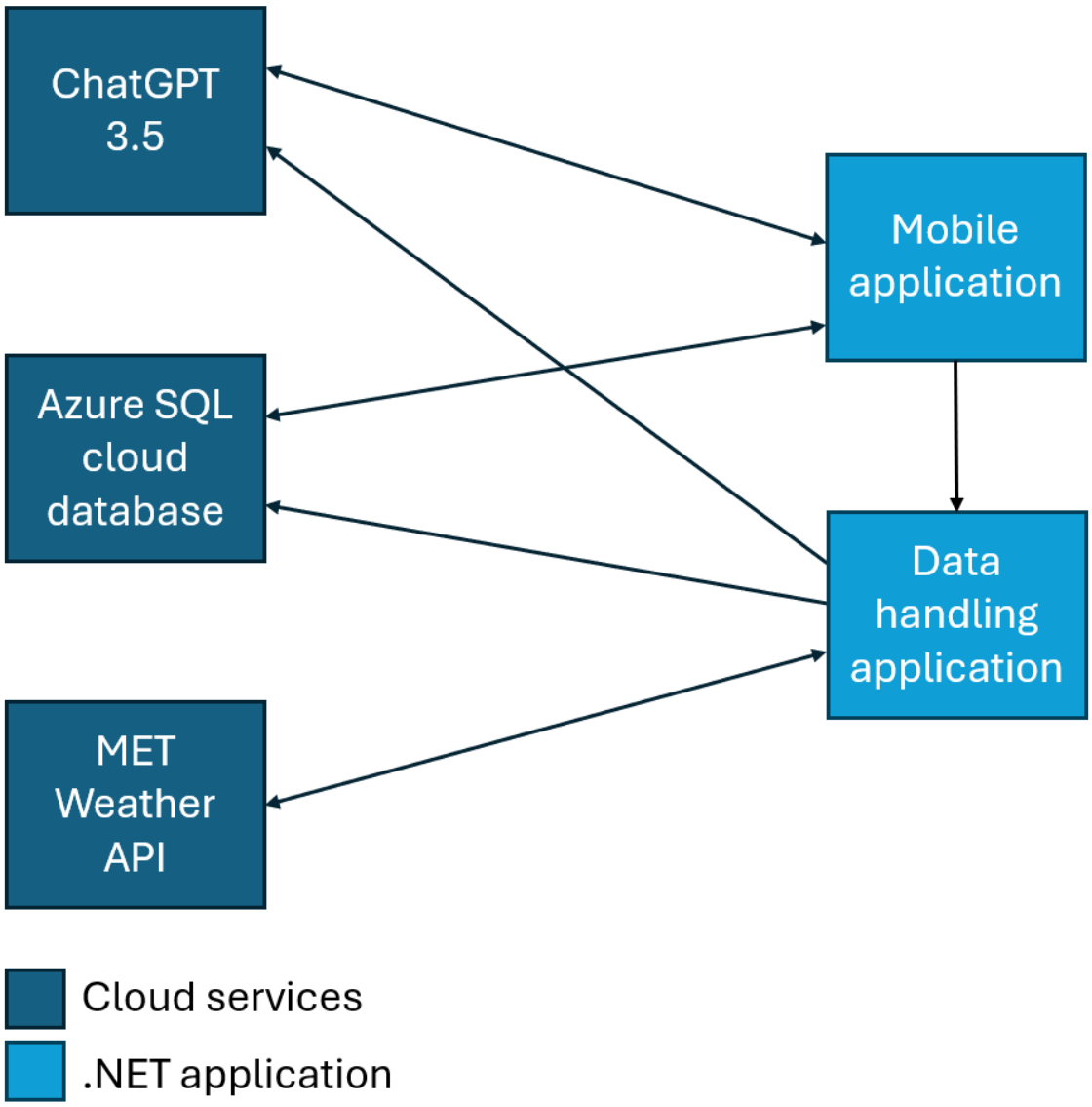


Figure 1.1: System sketch of energy prediction mobile application and its integration with cloud database, ChatGPT 3.5 AI, and weather API

On the right side of the system sketch in figure 1.1 you see the mobile application and the data handling application. In the first version the data handling application will be integrated into the mobile application, but in the future, this will be a separate application hosted on a server or as a web application. On the left side of the system sketch in figure 1.1 you see the other services that are used or developed as part of the project. The MET weather application programming interface (API) [2] is an online service that provides weather data and forecast through their API. The Azure structured query language (SQL) cloud database is a service

developed as part of the project using Microsoft Visual Studio and the Azure cloud service. The ChatGPT 3.5 [3] AI model is used to predict energy consumption. It is made available through an API developed as part of the project using Microsoft Visual Studio and the Azure cloud service.

1.3 Methods used in mobile application development project

In this project studies using open-source materials will be conducted on tools for app development, AI tools suitable for predictive models, parameters used in energy calculations of buildings, and sources for weather data.

Using tools later selected, a prototype app will be developed. This app will source test data from a cloud-based test database containing necessary indoor environmental data and energy consumption data. Weather data will be sourced from a later selected online source. Using AI, an energy consumption prediction model will be created, and predictions displayed in the app. The app will be developed using the Unified Software Development Process (UP).

A software test environment will be set up to test the prototype app.

GitHub will be used to document software development.

1.4 Scope of mobile application development project

The scope of this application development project, as described in the project description [1], are summarized below:

- Selection of a mobile application development tool for developing the application. This will involve a study of possible development tools and discussion of their suitability for this project.
- Investigation of AI tools that can be used for prediction of energy consumption and the selection of the tool to be used in this project.
- Selection of source for weather data that is needed to do the prediction of energy consumption. A study of available sources and requirements will also need to be conducted.
- In energy simulation certain parameters are being used. As part of the project the energy simulation methods will be studied and parameters used will be identified.
- Selection of building as a case study for the prototype mobile application needs to be selected based on available data for the project team.
- Using the AI model the mobile application should present an energy prediction based on the available data for the application.
- The mobile application should have an intuitive and user-friendly user interface (UI) that presents building data, weather data, building energy consumption, and energy prediction.

The results of the project and the development will be documented in this report.

2 Mobile application development tools study and selection

Several development environments exist for Android and iOS mobile application development. The primary and most widely used environments are platform-specific, meaning different codebases and programming languages are required to build the same application for Android and iOS. However, some environments support cross-platform development, allowing developers to save time, reduce maintenance, and lower costs by creating applications for both platforms with a single codebase. This chapter will explore the advantages and disadvantages of various development environments for mobile applications, focusing on a comparison between Android and iOS.

Table 2.1: Comparison of iPhone vs Android Statistics based on web page by Backlinko Team [4]

Aspect	**Android (Google Play)**	**iOS (App Store)**
Global market share	Over 70%	Around 30%
Number of Applications (2023)	2.44 million	1.8 million
User Spending (2023)	$51 billion	$124 billion

Both Android and iOS are found on a range of devices, including phones, watches, TVs, and tablets. Android is an open-source operating system led by Google, whereas iOS is proprietary and owned by Apple.

2.1 Android Studio

According to the web page “Meet Android Studio” by Android Developers [5]:

“Android Studio is the official Integrated Development Environment (IDE) for Android app development. Based on the powerful code editor and developer tools from IntelliJ IDEA, Android Studio offers even more features that enhance your productivity when building Android apps”

Android Studio is available for Windows, Linux, macOS, and ChromeOS. It primarily supports Kotlin as the preferred programming language, though Java and C++ are also supported. Android Studio offers the Android Software Development Kit (SDK), which includes debugging tools, libraries, sample code, tutorials, and an emulator, all essential for Android development.

Some limitations include its lack of cross-platform development support and high system resource demands, which may lead to slow performance on less powerful machines.

2.2 Xcode for iOS

Xcode is Apple's IDE for developing software for macOS, iOS, iPadOS, watchOS, and tvOS [6]. It is a proprietary software developed by Apple and is available only on macOS. It includes the iOS SDK, which is a collection of tools for the creation of applications for Apple’s iOS [7]. Swift is the primary programming language, though Xcode also supports Objective-C, C, C++, Python, Ruby, and others.

A drawback of Xcode is its macOS exclusivity, making it accessible only to Mac users. Additionally, it may perform slowly on complex projects, particularly on older Macs.

2.3 Cross platform

Cross-platform development tools allow a single codebase to be deployed on both Android and iOS, offering time and cost efficiency. However, most cross-platform tools require macOS to build and test iOS applications due to restrictions on the iOS SDK

Multiple solutions exist for cross platform development, and this chapter provides an overview of some of them.

2.3.1 React Native

React Native, developed by Meta, enables developers to build native Android and iOS applications using JavaScript [8]. React Native compiles to native code, delivering high performance, and allows for the integration of third-party libraries through a package manager, supporting customizable UI components. Development can be done on Windows, macOS, and Linux, though macOS is required to build for iOS.

2.3.2 Xamarin

Xamarin, based on the .NET framework, enables cross-platform development for iOS, Android, and Windows using C# as the primary language. Its UI is developed using XAML. Although Xamarin was officially retired in 2024 and replaced by .NET MAUI, it remains notable for its role in cross-platform development. The development environment is Visual Studio, and while development is primarily supported on Windows, macOS is required to build for iOS.

2.3.3 NET MAUI

.NET Multi-platform App UI (MAUI) is the latest framework from Microsoft, replacing Xamarin as the official .NET solution for cross-platform development. Built on .NET, .NET MAUI supports development for Android, iOS, Windows, and macOS using a single codebase [9]. It uses C# for logic and XAML for UI design, aligning with the .NET ecosystem for a cohesive experience.

2.3.4 Flutter

Flutter, developed by Google, enables cross-platform development for Android, iOS, Windows, Linux, and macOS. It uses the Dart programming language, and its Hot Reload feature allows developers to see changes instantly, speeding up debugging and development time. Flutter is known for its comprehensive widget catalog, enabling developers to create visually appealing UI components [10].

2.3.5 SOLAR2D

SOLAR2D, formerly Corona SDK, is an open-source cross-platform framework used to rapidly create apps and games for mobile devices and desktop systems [11]. It supports Windows, macOS, iOS, and Android and offers rapid deployment and fast performance.

2.4 Development tool comparison

Table 2.2 compares some of the key features of each development tool for mobile application development. The table is based on the sources mentioned in the chapters of each development tool.

Table 2.2: Mobile application development tool comparison

Development tool	Platform support	Primary language	Strengths	Limitations
Android Studio	Android	Kotlin	Official Android support, extensive SDK, profiling tools	High system demands, Android-only
Xcode	iOS, macOS, watchOS, tvOS	Swift	Integrated Apple SDK, UI design tools, regular updates	macOS-only, can be slow with complex projects
React Native	Android, iOS	JavaScript	Cross-platform, native code compilation, large UI component library	macOS required for iOS builds, limited access to some native features

Development tool	Platform support	Primary language	Strengths	Limitations
Xamarin	Android, iOS, Windows	C#	Part of .NET, compiles to native, supports Windows development	macOS required for iOS, officially replaced by .NET MAUI
.NET MAUI	Android, iOS, Windows	C#	Single codebase, cross-platform, modern UI components	Newer technology, macOS required for iOS builds, possible platform adaptation needed
Flutter	Android, iOS, Windows, macOS, Linux	Dart	Fast hot reload, large widget catalog, Google integration	Less mature ecosystem, macOS required for iOS builds
SOLAR2D	Android, iOS, Windows, macOS	Lua	Optimized for 2D game dev, fast deployment	Lua is less widely used

2.5 Selection of mobile application development tool for this project

After studying the mentioned mobile application development tools, .NET MAUI was selected as the tool to be used in this project. The main factors for making this choice were:

- .NET MAUI allows us to develop an application that uses one codebase that can be deployed to Android, iOS and WinUI.
- When making changes to the code, it only needs to be maintained in one place. For some special instances, platform specific adaptation might be needed.
- .NET MAUI is integrated in the .NET framework, giving us access to a large ecosystem with libraries, resources, and tools.
- .NET MAUI has Microsoft support and updates as part of .NET.

3 Study of AI tools for prediction and forecasting, and selection of tool

Traditional statistical methods have long been used for energy forecasting, providing a solid foundation for predicting future consumption. However, AI has introduced more advanced techniques, offering improved accuracy, enhanced capabilities, and simpler implementation.

3.1 Traditional methods for energy predictions

Before exploring AI tools, it's essential to recognize the traditional statistical approaches commonly applied for energy prediction:

- Auto-Regressive Integrated Moving Average (ARIMA): A linear model that predicts future values by analyzing historical time series data, based on the assumption that underlying patterns like seasonality and trend remain consistent [12].
- Exponential Smoothing: A technique used for short-term predictions, emphasizing recent data more than older data, making it responsive to immediate changes [13].
- Linear Regression Models (LRM): Simple regression models that use variables like temperature, time of day, and historical consumption to predict future usage.

While effective in some cases, these traditional models often struggle with non-linear data and complex relationships between energy consumption and external factors like weather, time, and consumer behavior.

3.2 AI Methods for energy predictions

AI-based forecasting has several advantages over traditional methods, particularly in handling large datasets and uncovering hidden patterns. In this chapter some key AI approaches are described.

3.2.1 Time Series Forecasting Models

Long Short-Term Memory Networks (LSTM) is a specialized form of recurrent neural networks (RNN) designed for sequential data, such as time series. LSTMs are particularly effective for energy consumption prediction since they can learn from long-term dependencies [14].

3.2.2 Machine Learning Techniques

Random Forest & Decision Trees are tree-based models that are effective for regression tasks involving multiple factors (such as weather data and energy consumption). Random forests, an ensemble of decision trees, can improve accuracy.

3.2.3 Deep Learning Models

Combining Convolutional Neural Networks (CNN) and LSTM are in some cases beneficial. CNNs are efficient at recognizing patterns, while LSTMs handle time dependencies. This combination works particularly well when you need to capture both spatial (e.g., regional weather patterns) and temporal (e.g., historical consumption) dependencies [15].

3.3 Pre-built AI Services

Energy forecasting for the application being made will use a pre-built AI service as an API, pre-built AI services allow you to leverage sophisticated forecasting capabilities without complex setup. In table 3.1 some of the most frequently used AI services are listed with a comparison of capabilities.

Table 3.1: Overview and comparison of pre-built AI services with their task capabilities based on sources for each service

AI Service	Text Processing	AutoML	Time Series Forecasting	Code Completion	General AI Tasks
ChatGPT (OpenAI) [16]	✓	✕	✕	✓	✓
GitHub Copilot [17]	✕	✕	✕	✓	✕
Amazon Forecast [18]	✕	✓	✓	✕	✕
Google AutoML [19]	✕	✓	✓	✕	✓
Microsoft Azure AI [20]	✓	✓	✓	✕	✓

As shown in table 3.1 the landscape of AI tools and services that could be used for energy forecasting is diverse, ranging from traditional statistical models to advanced machine learning frameworks, and fully managed AI services. Depending on the complexity of the project, goals, and technical expertise, there are solutions available that require little to no manual model building.

3.4 Selection of AI prediction tool for the project

In this project the selection of the right AI tool is key to achieve accurate energy predictions. The ideal tool should provide good prediction capabilities, ease of integration, scalability, and robust support. After evaluating several AI tools and services, ChatGPT hosted on Microsoft Azure emerged as the most suitable choice. The justification for the selection were:

- Hosting ChatGPT on Azure allows seamless integration with other Azure services. The integration with Azure enables streamlined workflows and reduces the complexity of setting up and managing the infrastructure.
- Using ChatGPT as a managed service offers financial and operational benefits. Azure's flexible pricing models allow us to scale usage on needs and budget constraints.
- Microsoft Azure follows a set of compliance certifications and security standards.
- With ChatGPT hosted on Azure the need for model training and configuration is reduced. The model comes pre-training and can be quickly configured to meet our requirements.

3.5 AI as a tool for programming

The use of AI in programming is growing, and it is also changing how programmers write code [21]. It can support programmers in writing code, debug code, execute projects, and help in documenting the code. There is also a growth in the available tools based on AI that are there to support programmers. In this project we are using GitHub, and one example of the available tools is the GitHub Copilot [22].

For this project AI has been used in the development of the mobile application to help understand error messages, structure the code, and suggest improvement. This helps reduce the development time and allows the programmers to focus on integration rather than coding challenges. The AI tool used in this project is ChatGPT [16].

3.6 AI as a tool for report writing

According to the article "How to Use AI to Write Reports: A Guide" by Insight7 [23]:

"Traditional report writing often involves extensive data collection, detailed analysis, and hours spent drafting, revising, and formatting. For professionals across fields, this process can be tedious and time-consuming."

According to the same article, AI tools can help us do these tasks. You can get help from AI, based on your data, help outline the report, generating drafts, refining content, and finalizing and formatting the report.

For this report we have had help from AI in helping refining content, finalizing the report, and checking for errors. The AI tool used in this project is ChatGPT [16].

4 Online sources for weather forecast data, and selection of source

As described in the project description [1], part of the project is to study sources for online weather forecast data. There are many sources of data available online, but for this project one weather API needs to be selected.

4.1 Examples of weather forecast data APIs

In the table 4.1 below you can see a selection of weather forecast data APIs available to access global weather data according to weatherstack article by Chauahn [24].

Table 4.1: Overview of weather forecast APIs to access global weather data according to weatherstack article by Chauhan [24]

Name	**Features**	**Pricing**
Weatherstack [25]	• Global coverage • Multiple languages • Reliable and easy-to-use solution Based on weatherstack article [24]	Free and paid plans [26]
OpenWeatherMap [27]	• Present weather conditions • 5-day and 16-day long-term forecasts • Trustworthy source of weather data Based on weatherstack article [24]	Free and paid plans [28]
Weatherbit [29]	• Unparalleled forecast accuracy • Hyperlocal weather • 99,9% uptime Based on weatherstack article [24]	Free and paid plans [30]
AccuWeather [31]	• Global coverage • Current conditions • Hourly and daily forecasts	Free and paid plans [32]

	Based on weatherstack article [24]	
Name	**Features**	**Pricing**
Tomorrow.io [33]	• Fast, reliable and accurate • Interface Based on weatherstack article [24]	Free and paid plans [34]
Visual Crossing [35]	• Single API call • 50+ years of data • Sub-hourly, hourly and daily data. Based on weatherstack article [24]	Free and paid plans [36]
Weather company [37]	• Real-time weather data • Comma-separated values (CSV) and JavaScript Object Notation (JSON) results • Geocoding Based on weatherstack article [24]	Paid plans [37]

4.2 Selection of online source of weather data to use in project

In this project we are focusing on the Norwegian regulations for energy calculations and parameters used there.

According to the Norwegian Specification SN-NSPEK 3031-2023 appendix A.8.1 [38], climate data could be sourced from The Norwegian Meteorological Institute or other renowned source that follows the rules in World Meteorological Organization: Guide to meteorological instruments and methods of observation [39].

Based on this we are for this project selecting to use The Norwegian Meteorological Institute [40].

The Norwegian Meteorological Institute offers different APIs for weather data [2]. For the project we need current data, forecasts for the coming period, and access to historical data. The Norwegian Meteorological Institute weather API *locationforecast* [41] provides a full weather forecast for one location, that is, a forecast with several parameters for a nine-day period. Historical data are not available via an API, but The Norwegian Meteorological Institute has archive data that can be downloaded [42].

5 Parameters used for energy simulations in buildings

In this chapter energy simulations and parameters used are studied. The study focuses on Norwegian regulations for energy calculations and parameters used there. Based on the study several suggested parameters for climate data and indoor environmental data is suggested to be used as the input for the energy estimation model.

5.1 TEK17

TEK17 [43] is the current regulations on technical requirements for construction works (Norwegian: "Forskrift om tekniske krav til byggverk") for Norway. The regulation is published in Norwegian, but there exists an unofficial translation released by the Norwegian Building Authority [44].

According to the regulations section 1-1 [43], the purpose of it is (cited from translation [44]):

"The Regulation is intended to ensure that projects are planned, designed and executed on the basis of good visual aesthetics, universal design, and in a manner that ensures that the project complies with the technical standards for safety, the environment health and energy."

Chapter 13 [45] gives requirements for indoor climate and health and chapter 14 [46] gives requirements for Energy.

According to section 14-2 (4) [44] [47], the buildings energy requirements and heat loss figures shall be calculated in accordance with Norwegian Standard NS 3031:2014. Also, section 14-2 (5) [44] [47] states that an energy budget must be calculated for non-residential buildings using actual figures. The regulation also gives minimum requirements for energy efficiency in section 14-3 [44] [48]. These are given as minimum U-values for building parts (outer walls, roofs, floors, windows, and doors) and leakage figures for the building.

5.2 Norwegian Standard NS 3031:2014

The Norwegian Standard NS 3031:2014 [49] covers "Calculation of energy performance of buildings – Method and data".

NS 3031:2014 is referenced in TEK17 section 14-2 (4) [47] and 14-4 (2) [50] but is no longer a valid standard [51]. The standard was withdrawn February 1st, 2018. According to Standard Norge, the reason for this is that the standard conflicts with new European standards. This is linked to NS-EN ISO 52000-1 being published in 2017.

According to Standard Norge's article [51] the Norwegian Standard NS 3031 [49] is under revision to adapt to the content of NS-EN ISO 52000-1 [52] and other European standards. As stated in the article [51] the Norwegian Standard NS 3031:2014 [49] can still be used.

5.3 Norwegian Specification 3031:2023

The Norwegian Specification SN-NSPEK 3031-2023 [38] covers “Energy performance of buildings – Calculation of energy needs and energy supply”. This specification is only available in Norwegian language.

According to the specification [38] it is referencing the European standards that are part of what is referenced as the Energy Performance of Buildings (EPB) or “set of EPB standards” as it is called in NS-EN ISO 52000-1:2017 [52]. In addition, the specification [38] states that it also in addition points to alternative ways to calculate the energy requirements for buildings. The specification [38] is also adapted to the use of dynamic calculation programs.

5.4 Climatic data

According to the Norwegian Standard NS-EN ISO 15927-4 chapter [53] and the Norwegian Specification SN-NSPEK 3031-2023 [38], the reference year used for energy calculations should at least contain the following metrological parameters:

- Air temperature, in °C.
- Relative humidity, in %.
- Wind speed, in m/s.
- Direct normal solar irradiance, in W/m².
- Diffuse solar irradiance on a horizontal surface, in W/m².

According to the specification [38] the energy consumption can be conducted using a standard reference climate or local climate. For Norway, the standard reference climate is defined as the Oslo-area.

Based on this the above measured parameters can be used either as local measured data or taken from online sources.

5.5 Indoor environmental data

According to the Norwegian Specification SN-NSPEK 3031-2023 A.7 [38] the main environmental parameter for the indoor thermal environment is indoor air temperature in °C.

This is for the energy calculation used as a set temperature for the building according to building type with the possibility to adjust for operational hours.

In the specification [38] the air quality is described as minimum air flow through the ventilation in $m^3/(h \cdot m^2)$. This could be measured in the ventilation system.

An alternative to use the volume of air, could be to use the Norwegian Standard NS-EN 16798-1:2019 [54]. This standard [54] split the input parameters for design and assessment of energy performance of buildings into the following sub chapters:

- Thermal environment.
- Design for Indoor air quality.
- Humidity.
- Lighting.

- Noise.

Noise is mostly handled in the design process of the building, and not that relevant for this project.

Based on the standards appendix A [54], the following indoor environmental parameters could be used for the basis of energy calculations in addition or as an alternative to air flow rates; CO_2 concentration, in ppm, and relative humidity, in %.

For lighting the specification [38] is saying that for larger rooms there should be a system with occupancy detection or zone control. The specification [38] also references the Norwegian Standard NS-EN 15193-1:2017+A1 [55] which covers "Energy performance of buildings – Energy requirements for lighting". This standard [55] describes two methods to calculate the lighting energy and one to measure it. The measured method requires to measure the energy consumption on all lighting in the building on separate meters. In the Norwegian Standard NS-EN 16798-1:2019 appendix A [54] parameters for lighting are illuminance, in lx.

In addition, for design the standard [54] uses a measure for how even the lightning are in the room, but this is normally used for calculations in the design phase and during quality control. To measure this the illuminance must be measured on multiple points in the room.

Based on this, the following indoor environmental parameters could be suggested to be used as possible inputs for the energy estimation:

- Indoor air temperature, in °C.
- Air flow, in m^3/s.
- CO_2 concentration, in ppm.
- Relative humidity, in %.
- Occupancy of room, Boolean (true or false).
- Illuminance, in lx.

Based on the size and complexity of the building the number and type of sensors will vary. Also, not all sensors might be present for the building, and then will not be available as data for the estimation.

6 Azure SQL cloud database with web API

To connect the application to the cloud database, a web API is needed. In this chapter the implementation of the web API on Azure and the Azure SQL database is described.

6.1 Azure SQL database

An empty Azure SQL database was created on the Azure portal. The name of the database is *database*. The process of creating a database in Azure is shown on the website "Create a single database - Azure SQL Database" by Assaf [56]

6.2 Web API using Microsoft Entity Framework

To create the web API the tutorial "Create a web API with ASP.NET Core" by Microsoft [57] were partly followed. The tutorial shows how to create a controller-based web API with ASP.NET Core. The tutorial shows how to create a web API for a database running in the memory of the unit running the program. In this project we are using an in-memory database for testing and a Azure SQL database for production, so this was changed in the code using information from the tutorial "Connect to and query Azure SQL Database using .NET and the *Microsoft.Data.SqlClient* library" by Microsoft [58].

The changes done to the ASP.NET core web API template is the following:

- Added model classes for all tables
- Added database context for all tables
- Registered the database context for all tables
- Scaffolded a controller for each table
- Created migrations for the tables and migrated them to the database

An example of the content of a class for a database table, *BuildingTemperatureItem.cs,* is shown in figure 6.1. Also, the content of the class for the database context, *DatabaseContext.cs,* is shown in figure 6.2. For each database table a class as shown in figure 6.1 is created.

The connection string for the database is found in the Azure portal for the database and stored as a connection string in the Azure web app for web API. In the visual studio code, the connection string is empty in the *appsettings.json* file. In figure 6.3 the *appsettings.json* is shown.

The *EPPlus* part of *appsettings.json* sets the non-commercial license for this package which allows reading the excel file defined in *DataFilePath*. This file contains data that is used to fill the in-memory database with data during the launch of the program when using an in-memory database.

The selection of if you are using the in-memory or the SQL database is based on the connection string which is only present when running the program on the Azure app service. The code for the selection is shown in figure 6.4.

```
using System.ComponentModel.DataAnnotations;

namespace DatabaseWebAPI.Models
{
    8 references | Oerjan Ingebrigtsen, 17 hours ago | 1 author, 3 changes
    public class BuildingTemperatureItem
    {
        [Key]
        3 references | Oerjan Ingebrigtsen, 19 days ago | 1 author, 1 change
        public long Id { get; set; }
        [Required]
        1 reference | Oerjan Ingebrigtsen, 19 days ago | 1 author, 1 change
        public float Temp1 { get; set; }
        [Required]
        1 reference | Oerjan Ingebrigtsen, 19 days ago | 1 author, 1 change
        public float Temp2 { get; set; }
        [Required]
        1 reference | Oerjan Ingebrigtsen, 19 days ago | 1 author, 1 change
        public float Temp3 { get; set; }
        [Required]
        1 reference | Oerjan Ingebrigtsen, 19 days ago | 1 author, 1 change
        public float Temp4 { get; set; }
        [Required]
        1 reference | Oerjan Ingebrigtsen, 19 days ago | 1 author, 1 change
        public float Temp5 { get; set; }
        [Required]
        [MaxLength(10)]
        1 reference | Oerjan Ingebrigtsen, 10 days ago | 1 author, 2 changes
        public string? TempUoM { get; set; } = "°C";
        [Required]
        2 references | Oerjan Ingebrigtsen, 17 hours ago | 1 author, 2 changes
        public DateTime? TempDateTime { get; set; } = DateTime.UtcNow;
    }
}
```

Figure 6.1: Example of content of class for *BuildingTemperatureItem.cs*

```
using Microsoft.EntityFrameworkCore;

namespace DatabaseWebAPI.Models
{
    23 references | Oerjan Ingebrigtsen, 10 days ago | 1 author, 2 changes
    public class DatabaseContext : DbContext
    {
        0 references | Oerjan Ingebrigtsen, 19 days ago | 1 author, 1 change
        public DatabaseContext(DbContextOptions<DatabaseContext> options)
            : base(options)
        {
        }

        8 references | Oerjan Ingebrigtsen, 19 days ago | 1 author, 1 change
        public DbSet<BuildingTemperatureItem> BUILDING_TEMP { get; set; } = null!;
        8 references | Oerjan Ingebrigtsen, 19 days ago | 1 author, 1 change
        public DbSet<BuildingRelativeHumidityItem> BUILDING_REL_HUMIDITY { get; set; } = null!;
        8 references | Oerjan Ingebrigtsen, 19 days ago | 1 author, 1 change
        public DbSet<BuildingEnergyMeterItem> BUILDING_ENERGY_METER { get; set; } = null!;
        8 references | Oerjan Ingebrigtsen, 10 days ago | 1 author, 1 change
        public DbSet<EnergyPredictionItem> ENERGY_PREDICTION { get; set; } = null!;
        9 references | Oerjan Ingebrigtsen, 10 days ago | 1 author, 1 change
        public DbSet<WeatherForecastItem> WEATHER_FORECAST { get; set; } = null!;
        8 references | Oerjan Ingebrigtsen, 10 days ago | 1 author, 1 change
        public DbSet<WeatherForecastUoMItem> WEATHER_FORECAST_UOM { get; set; } = null!;
    }
}
```

Figure 6.2: Example of content of the database context class *DatabaseContext.cs*

```json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "EPPlus": {
    "ExcelPackage": {
      "LicenseContext": "NonCommercial"
    }
  },
  "ConnectionStrings": {
    "ProductionConnection": ""
  },
  "DataFilePath": "./Data.xlsx"
}
```

Figure 6.3: Content of the *appsettings.json* file

```
// Connection string for production database
var prodConnectionString = builder.Configuration.GetConnectionString("ProductionConnection");

// Check connection string and select database
if (prodConnectionString != "")
{
    // Attempt to use Azure SQL Server configuration first
    builder.Services.AddDbContext<DatabaseContext>(options =>
        options.UseSqlServer(prodConnectionString, sqlOptions =>
            sqlOptions.EnableRetryOnFailure()));
}
else
{
    // Configure DbContext with in-memory database for development/testing
    builder.Services.AddDbContext<DatabaseContext>(options =>
        options.UseInMemoryDatabase("BuildingDatabase"));
}
```

Figure 6.4: The selection of SQL or in-memory database based in connection sting in *program.cs*

When using the in-memory database we start a special service and pipeline to load the data from the excel file *data.xlsx* to the in-memory database and start Swagger [59]. Swagger is used to create a user interface to test the web API, when running on local host, in a web browser.

```
// Check if we're using an in-memory database and configure services accordingly
if (InMemoryDatabaseSetup.IsInMemoryDatabase(builder))
{
    InMemoryDatabaseSetup.ConfigureInMemoryServices(builder);
}

var app = builder.Build();

// Configure the pipeline if using an in-memory database
if (InMemoryDatabaseSetup.IsInMemoryDatabase(builder))
{
    await InMemoryDatabaseSetup.ConfigureInMemoryPipelineAsync(app);
}
```

Figure 6.5: Code to start the necessary service and pipeline to load data from *data.xlsx* to the in-memory database and start Swagger

We will not go into detail of the configuration of the un-memory database services or pipeline since this mainly is for testing purposes during the development stage.

To have the API working, in addition to the program file, classes for the tables, and database context we need a controller for each table. The controllers are created by making scaffolded items. The process is described in the tutorial “Create a web API with ASP.NET Core” by Microsoft [57]. This process creates a controller with actions without the need to code anything in Visual Studio.

For the classes *BuildingEnergyMeterItem*, *BuildingRelativeHumidityItem*, *BuildingTemperatureItem*, *EnergyPredictionItem*, and *WeatherForecastItem*, we had to add one action. This action retrieves the latest entry or entries into the database table. For *BuildingEnergyMeterItem*, *BuildingRelativeHumidityItem*, and *BuildingTemperatureItem* the last entry into the table is retrieved based on the *DateTime* column. As an example, the code for this action from the *BuildingTemperatureItemsController* is shown in figure.

```
107         // GET: api/BuildingTemperatureItems/latest
108         [HttpGet("latest")]
            0 references | Oerjan Ingebrigtsen, 19 days ago | 1 author, 1 change
109         public async Task<ActionResult<BuildingTemperatureItem>> GetLatestBuildingTemperatureItem()
110         {
111             var latestTemp = await _context.BUILDING_TEMP.OrderByDescending(t => t.TempDateTime).FirstAsync();
112
113             if (latestTemp == null)
114             {
115                 return NotFound();
116             }
117
118             return latestTemp;
119         }
```

Figure 6.6: Example of latest action from the *BuildingTemperatureItemsController*

For the *EnergyPredictionItem*, and *WeatherForecastItem* classes we had to have an action that retrieved several rows with the last *ExecuteTime* and *DateTime* respectively. In figure 6.7, the latest action code from the *WeatherForecastItemsController* is shown.

```
107         // GET: api/WeatherForecastItems/latest
108         [HttpGet("latest")]
            0 references | Oerjan Ingebrigtsen, 10 days ago | 1 author, 1 change
109         public async Task<ActionResult<IEnumerable<WeatherForecastItem>>> GetLatestWeatherForecastItems()
110         {
111             // Find the latest DateTime value in the table
112             var latestDateTime = await _context.WEATHER_FORECAST.MaxAsync(w => w.DateTime);
113
114             // Retrieve all rows with the latest DateTime value
115             var latestForecastItems = await _context.WEATHER_FORECAST
116                 .Where(w => w.DateTime == latestDateTime)
117                 .ToListAsync();
118
119             // If no items are found, return a 404 NotFound response
120             if (!latestForecastItems.Any())
121             {
122                 return NotFound("No weather forecast items found for the latest DateTime.");
123             }
124
125             return Ok(latestForecastItems);
126         }
```

Figure 6.7: Example of latest action from the *WeatherForecastItemsController*

For the *WeatherForecastUoMItem* we had to create an action to retrieve the UoM for one attribute. The code for this action from the *WeatherForecastUoMItemsController* is shown in figure 6.8.

```
107         // GET: api/WeatherForecastUoM/uom/{attribute}
108         [HttpGet("uom/{attribute}")]
            0 references | Oerjan Ingebrigtsen, 8 hours ago | 1 author, 2 changes
109         public async Task<ActionResult<string>> GetUoMByAttribute(string attribute)
110         {
111             // Find the UoM entry based on the attribute name (case-insensitive search)
112             var uomItem = await _context.WEATHER_FORECAST_UOM
113                 .FirstOrDefaultAsync(u => u.Attribute.ToLower() == attribute.ToLower());
114
115             // If the attribute doesn't exist, return a 404 NotFound response
116             if (uomItem == null)
117             {
118                 return NotFound($"No UoM found for attribute '{attribute}'.");
119             }
120
121             // Return the UoM as a plain string
122             return Ok(uomItem.UoM);
123         }
```

Figure 6.8: Example of code to retrieve the UoM for a specified attribute in the *WeatherForecastItemsController*

6.3 Azure SQL database

To create the database table the *Add-migration* tool was used in Visual Studio. How to use this tool is described on the web page Migrations Overview by Mono [60]. The migrations also must be applied to the database. How to do this is explained on the web page Applying Migrations by Mono [61].

The tables added to the Azure database using the *Add-Migration* and *Update-Database* functionality are shown in figure 6.9.

Historical data was added to the production database running the code used for the in-memory database once. The historical data in the *Data.xlsx* file was loaded to the database.

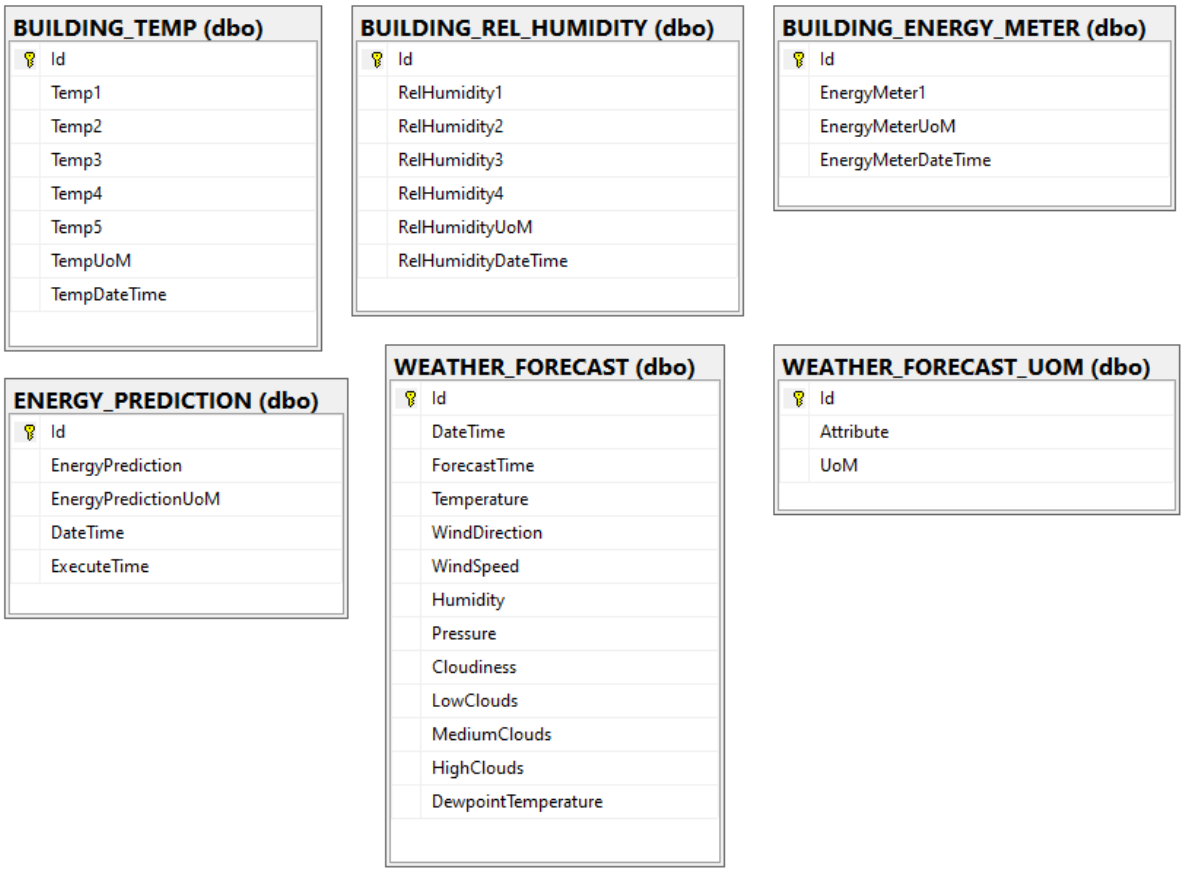


Figure 6.9: Table structure of tables added to database from Visual Studio using *Add-Migration* and *Update-Database* functionality

6.4 Azure app service for web API

When the code for the *DatabaseWebAPI* was finished, it was published to Azure using Visual Studio. How to deploy an application to Azure app service is described on the web page "Quickstart: Deploy an ASP.NET web app" by Microsoft [62]. Since we were publishing a web API some more options were available, and we selected to skip the API management option.

6.5 Testing of web API

Testing of the web API on Azure was done using Postman [63] and the mobile application. All the API requests used in the application were tested from both.

7 AI prediction model using ChatGPT 3.5 and Azure web API

This chapter describes how the AI prediction model is implemented, using ChatGPT 3.5 with Azure AI Studio to forecast energy consumption based on weather forecast and building data from the Azure SQL database via its API.

7.1 Overview of the prediction model

The task of the AI prediction model is to forecast the energy consumption of the customer's building for the next week. This is done by utilizing historical weather and energy consumption data to train the model for the specific customer. The model therefore requires some historical data to be able to predict. The more data the more accurate the prediction will be. To limit over usage the request will only be sent if similar data does not already exist in the database. For each request the model will return the prediction for the following week, based on weather forecast. The predictions are then stored back into the Azure SQL Database via the API.

7.2 Using Azure AI Studio

As described in chapter 3, Azure AI Studio was selected to be used with the ChatGPT 3.5 model. To do this an instance of the ChatGPT 3.5 model was created in the Azure AI Studio. To secure the AI prediction we are using API keys and authentication settings to allow secure connection between the API and the application. The model is then ready to be used with a train-as-you-go approach. Meaning the model will not be pre trained, but it will be supplied with training data for each request. This is possible because of the speed this large language model delivers.

Azure AI Studio provides monitoring tools to track error rates or API usage as shown in Figure 7.1. This gives insight into the frequency the app is being used by customers. Integration with other Azure services, like the Azure SQL Database ensures a smooth data flow between the model and the rest of the application.

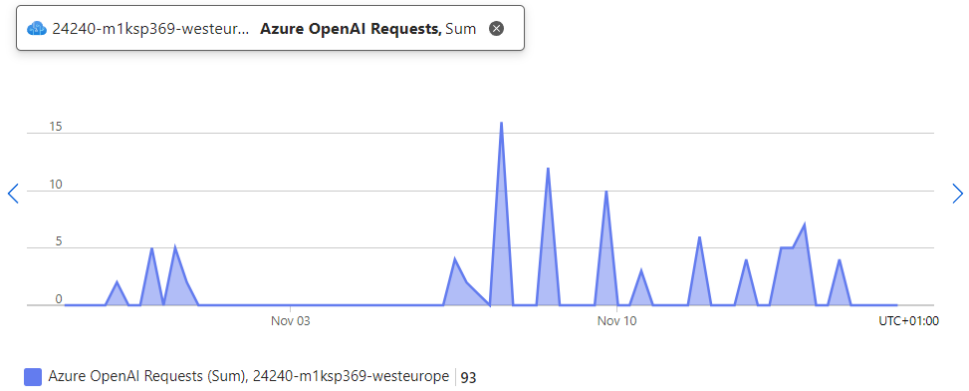


Figure 7.1 Prediction API Usage for 14 days

In addition to error and usage overview Azure AI Studio also displays accumulated as well as per day cost. The displayed cost in Figure 7.2 represents the 93 first request used in the development stage.

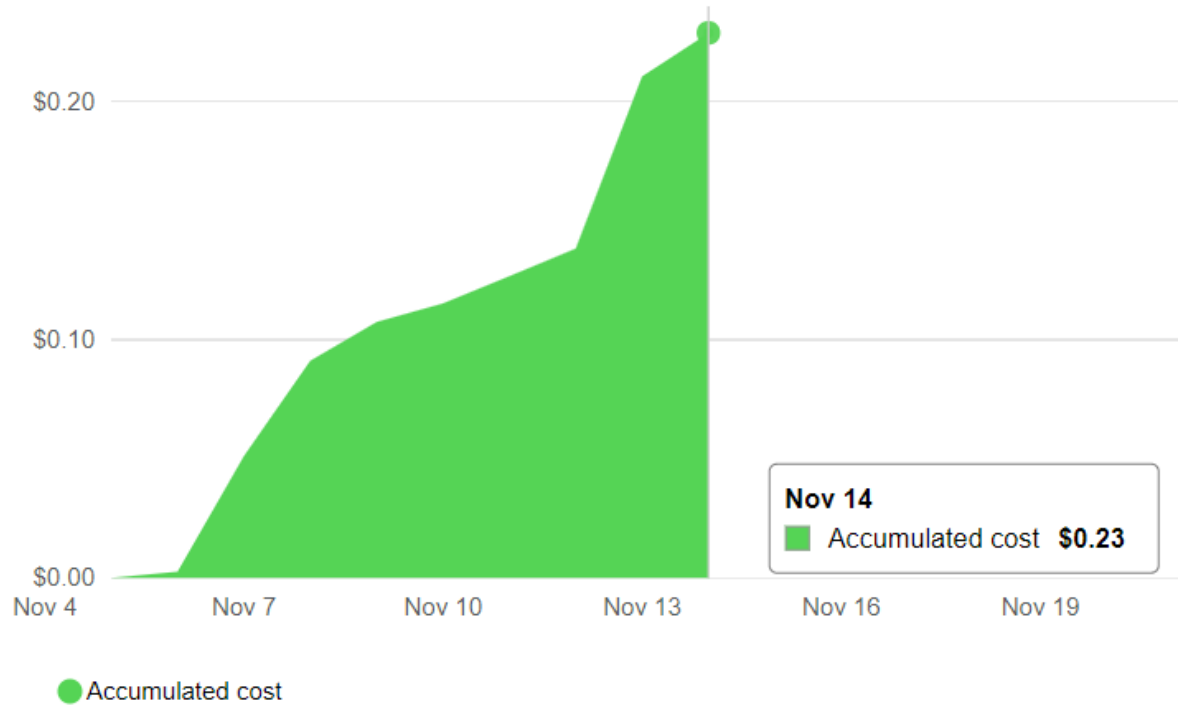


Figure 7.2 Accumulated cost of prediction model for 93 requests

7.3 Generating predictions with ChatGPT 3.5

The process of generating predictions with ChatGPT 3.5 involves multiple steps, from data retrieval to API request formatting, and output handling.

1. Data Preparation
    - Data is extracted from the SQL database. All relevant data could be included but in the development stage only historic outside temperature and energy consumption is used.
    - Data is extracted from the weather forecast API and used as inputs for the trained model.
2. API Request input
   The API request consists of a message and some parameters. The message includes a system and a user role. The system role provides context or guidelines for the model's behavior. In Figure 7.3 it is shown that the system role is specifically designed to only return values in a JSON format.

```
messages = new[]
{
    new { role = "system", content = "You are an AI model that strictly provides JSON-formatted predictions " +
    "based on provided data, with no additional text or explanations." },
    new
    {
        role = "user",
        content = $@"
            Predict the daily energy consumption for the following days based on the following conditions:
            {predictData}

            Based on the historical training data:
            {promptData}

            Respond only in JSON format with an array of predictions for each day,
            where each entry includes the date and the predicted energy consumption.
            Use the following exact format and do not include any other text:

            {{
                ""predictions"": [
                    {{ ""date"": ""YYYY-MM-DD"", ""consumption"": predicted_value }},
                    {{ ""date"": ""YYYY-MM-DD"", ""consumption"": predicted_value }},
                    ...
                ]
            }}"
    }
},
```

Figure 7.3 Prediction API input message

The data included in the variable *promptData* from Figure 7.3 will be all the data the model will use to train. In the development stage this includes all dates were both outside temperature and energy consumption is stored in the database. The data is given in this format:

Date: 2024-09-01, Temperature: 13,7°C, Energy Consumption: 24,96 kWh

Date: 2024-09-02, Temperature: 11°C, Energy Consumption: 24,96 kWh

Date: 2024-09-03, Temperature: 13,6°C, Energy Consumption: 24,96 kWh

…

Date: 2024-11-11, Temperature: 1°C, Energy Consumption: 78,36 kWh

Date: 2024-11-12, Temperature: 0,3°C, Energy Consumption: 87,33 kWh

The variable *predictionData* includes the days and data the model will use as in inputs for the predictions. This is given in this format:

Date: 2024-11-15, Average Temperature: 8,10°C

Date: 2024-11-16, Average Temperature: 6,46°C

Date: 2024-11-17, Average Temperature: 4,40°C

Date: 2024-11-18, Average Temperature: 0,00°C

Date: 2024-11-19, Average Temperature: -2,22°C

Date: 2024-11-20, Average Temperature: -0,48°C

Date: 2024-11-21, Average Temperature: -2,03°C

3. Handling prediction output
   Once ChatGPT 3.5 returns a response, the model's output would be like:

```
"predictions":
[
   { "date": "2024-11-15", "consumption": 62.43 },
   { "date": "2024-11-16", "consumption": 64.85 },
   { "date": "2024-11-17", "consumption": 71.83 },
   { "date": "2024-11-18", "consumption": 76.35 },
   { "date": "2024-11-19", "consumption": 82.38 },
   { "date": "2024-11-20", "consumption": 78.11 },
   { "date": "2024-11-21", "consumption": 80.21 },
]
```

The response is parsed, and the predicted values are stored in the database. Each row includes a timestamp for the prediction, making it easy to track and retrieve specific forecasts.

8 Development of the mobile application

In this chapter the development of the mobile application is described. The mobile application will present the current and historical building data and weather data and allow the user to request a prediction for the future energy consumption.

8.1 Detailed system sketch over energy prediction system

In figure 8.1 a detailed system sketch over the energy prediction system is shown.

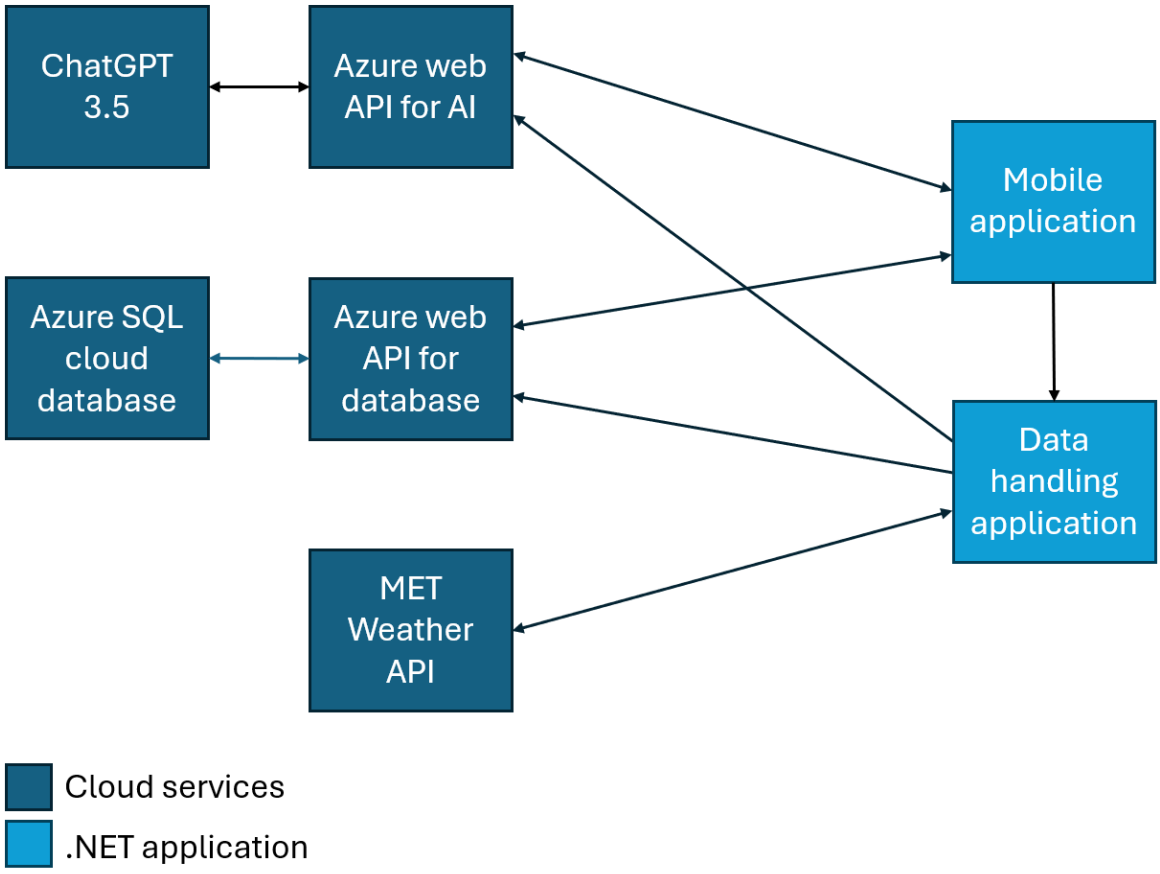


Figure 8.1: Detailed system sketch of the energy prediction system with applications and services including necessary APIs

In this first version of the mobile application the data handling application is included in the application. In later versions the recommendation is to develop this as a separate application to be hosted on a server or a cloud service.

In this more detailed system sketch figure 8.1, compared to figure 1.1, the APIs for the ChatGPT 3.5 AI model and the Azure SQL cloud databases are shown. The mobile

application and the data handling application is not communicating directly to the AI model or the database. All communication goes through the designated API.

This is the same for the MET Weather API [2], but this is developed by the Norwegian Metrological Institute.

8.2 FURPS+ for mobile application

FURPS+ has been used to collect requirements for the application. FURPS+ is described in chapter 6.2.3 of the book Lecture notes for object-oriented analysis, design, and programming using UML and C# [64].

1) F – Functional requirements
   i) The current data (latest entry into database) should be displayed in the application as a dashboard.
   ii) For each data element a historical log of the data should be presented in the application when selected.
   iii) An prediction of the energy consumption of the next period is to be shown in the application using an artificial intelligence generated estimation model.
   iv) The data should be stored in an online database that could be updated from an external building management system (updating the database is not part of this project).
2) U – Usability
   i) The data displayed in the dashboard should be updated regularly.
   ii) If multiple sensors exist for one data element, the data should be displayed as an average or sum depending on the data element in the dashboard.
   iii) If multiple sensors exist for one data element, the data elements should be displayed individually when clicking on the data element.
   iv) When selecting one data element the historical log should be presented in the application as a graph.
   v) The date and time of the estimation of the energy consumption and the period should be shown together with the estimation in the application.
   vi) The period for the estimation of the energy consumption is normally the coming seven days.
3) R – Reliability
   i) The database should be able to be written and read at any time outside of announced maintenance windows.
   ii) The prediction of the energy consumption can only be updated once per 24 hours without payment.
   iii) The access to weather data is only available when the weather data source is available.
4) P – Performance
   i) The application should be able to load the current data within 30 seconds.
5) S – supportability
   i) The app should be available for both Android and iOS mobile units.
6) + Design challenges/limitations

8.3 Use case diagram for mobile application

The use case diagram in figure 8.2 illustrates the primary interactions between the mobile application, the user and external systems. The mobile application allows the user to view weather and energy consumption both historical and future predictions. Each of these data sources are retrieved from external sources.

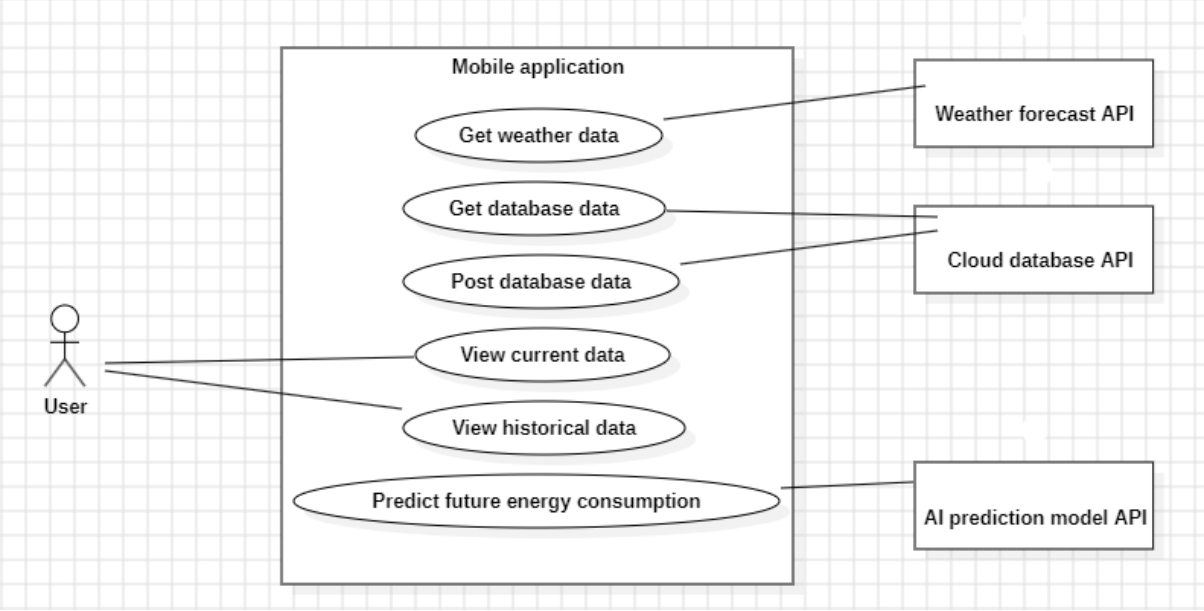


Figure 8.2: Use case diagram for mobile application

8.4 Use case: Get weather data

The goal of the use case “Get weather data” is to fetch future weather forecasts from the Norwegian Meteorological Institute [2] through an API. This use case is essential for applications like energy consumption predictions to supply necessary data. The use case is described in the “Fully Dressed Use Case Document” in Table 8.1 and the system sequence diagram in Figure 8.3.

Table 8.1: FDUCD for use case: Get weather data

1	Use case name	Get weather data
2	Scope	Retrieve Weather Data from Third-Party API
3	Level	User-goal
4	Primary actor	Application user
5	Stakeholders	Application user: Wants weather data
		Third-party API: Provides data
		System administrator: Ensures API access
		Developer: Integrates API
6	Precondition	Application is online and running
		Weather API service is functioning
		Valid API key is provided
7	Success Guarantee	Weather data is retrieved for the correct location
8	Main Success Scenario	1. Location is provided by user
		2. System sends request to API
		3. API returns weather data
		4. System processes and displays the data
9	Extensions	2a. No internet connection: System shows "No internet connection"
		3a. Invalid API key: System shows "Invalid API key"
		3b. API Limit exceeded: System shows "API limit exceeded"
		3c. Location not found: System informs user that location is invalid
10	Special Requirements	Weather data updated frequently
		API key stored in securely in seperate file
11	Technology List	Third-party API: Meteorologisk institutt; api.met.no "Locationforecast"
12	Frequency	Intervals: 24 hour
13	Misc	User-friendly error messages when data retrieval fails

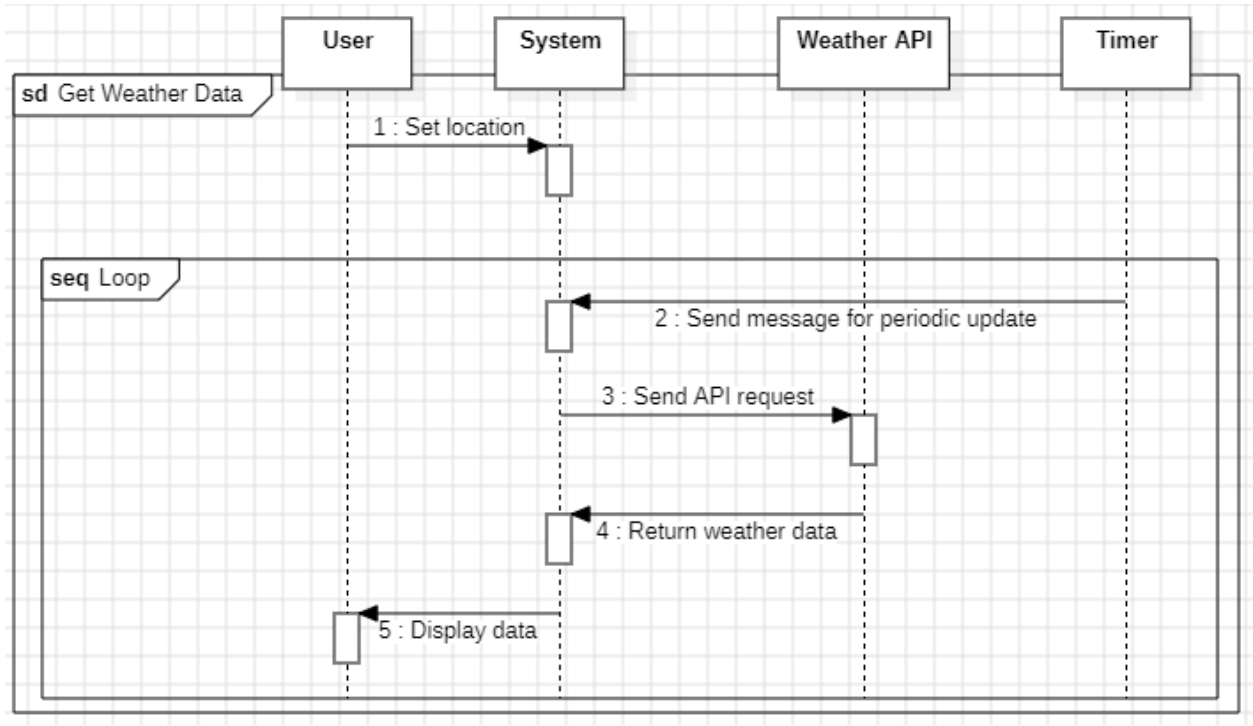


Figure 8.3: System sequence diagram for use case: Get weather data

For the use case a class called *WeatherService.cs* is created to request data from the API. The API requires a name of the application and an email address. This information is added to the header of the client in the constructor of the class as shown in Figure 8.4. Later a method called *GetWeatherDataListAsync* shown in Figure 8.5, that takes longitude, latitude and altitude as inputs, sends a request to the API. The returned information is parsed into elements of the class *WeatherForecastItem.cs* shown in Figure 8.6.

```
public WeatherService()
{
    // Add the User-Agent header required by the API
    client.DefaultRequestHeaders.Add("User-Agent", "EnergyPredictionSystem/1.0 (242402@usn.no)");
    _databaseWebAPIServices = new DatabaseWebAPIServices();
}
```

Figure 8.4: Class *WeatherService* constructor in *WeatherService.cs*

```
public async Task<List<WeatherForecastItem>> GetWeatherDataListAsync(double lat, double lon, double altitude)
{
    string url = $"https://api.met.no/weatherapi/locationforecast/2.0/classic?lat={lat.ToString(CultureInfo.InvariantCulture)}&" +
        $"lon={lon.ToString(CultureInfo.InvariantCulture)}&altitude={altitude.ToString(CultureInfo.InvariantCulture)}";

    try
    {
        Debug.WriteLine($"Requesting: {url}");
        var response = await client.GetAsync(url);
```

Figure 8.5: Method *GetWeatherDataListAsync* in *WeatherService.cs*

```
public class WeatherForecastItem
{
    1 reference
    public int Id { get; set; }
    3 references
    public string DateTime { get; set; }
    5 references
    public string ForecastTime { get; set; }
    5 references
    public float Temperature { get; set; }
    3 references
    public float WindDirection { get; set; }
    3 references
    public float WindSpeed { get; set; }
    3 references
    public float Humidity { get; set; }
    3 references
    public float Pressure { get; set; }
    3 references
    public float Cloudiness { get; set; }
    3 references
    public float LowClouds { get; set; }
    3 references
    public float MediumClouds { get; set; }
    3 references
    public float HighClouds { get; set; }
    3 references
    public float DewpointTemperature { get; set; }
}
```

Figure 8.6: Class *WeatherForecastItem* in classes

8.5 Use case: Get database data

The goal of this use case is to retrieve data stored in cloud database. To make the Azure SLQ database available a web API is being used.

In this use case, the *DatabaseWebAPIServices* class serves as the primary service between the client application and the database API. By encapsulating API requests, it allows the client application to retrieve multiple data types from a database via the API without direct database interaction.

In figure 8.7 the sequence diagram of the request is shown.

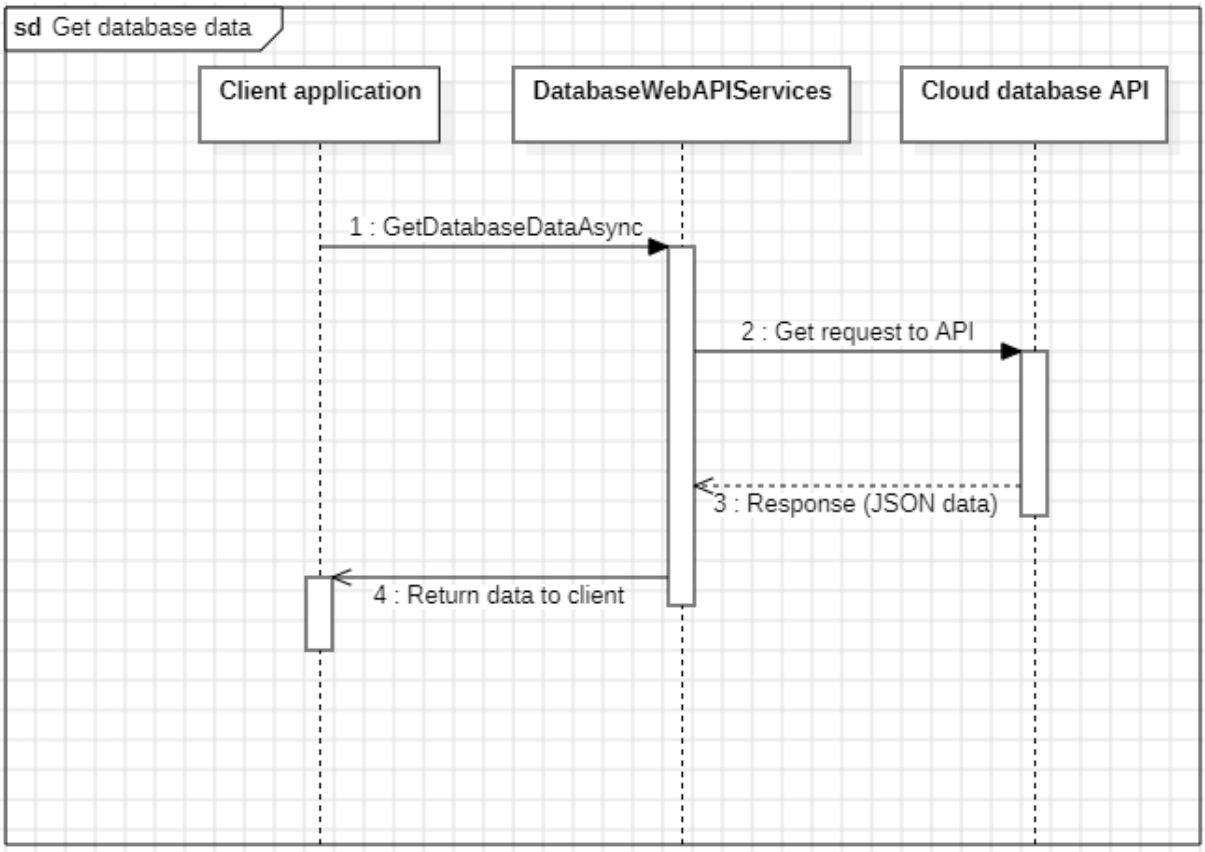


Figure 8.7: Sequence diagram for use case: Get database data

In table 8.2 the use case document for the get database data is shown.

Table 8.2: FDUCD for use case: Get database data

1	Use case name	Get database data
2	Scope	To retrieve specific types of data from the database using a REST API call, based on the specified data type
3	Level	User-goal
4	Primary actor	Application user
5	Stakeholders	Application user: Want to retrive data from database
		Cloud database API: Provides data from database
		System administrator: Ensures API access
		Developer: Integrates API
6	Precondition	The cloud database and web API are operational and accessible
		The specified URL endpoint provides valid JSON data for the requested data type
		The client has the necessary permissions to access the API endpoint
		The database web API services class is correctly instantiated in the application and configured to use the correct endpoint URLs for each data type
7	Success Guarantee	The requested data is successfully retrieved and returned to the client application
8	Main Success Scenario	1. User/Client application initiates get request
		2. Service executes request
		3. Data retrieval and seserialization
		4. Return data to client
		5. Client uses data
9	Extensions	1a. If the server returns a non-success status code , the GetAsync<T> method throws an HttpRequestException
		1b. The exception is handled by the client application, and an appropriate error message is displayed or logged for troubleshooting
		2a. If there is a network or connection issue, the HTTP GET request fails
		2b. An exception is thrown, which is caught and logged by the client application for user notification or retry logic
10	Special Requirements	Data is retrived frequently
		API key stored in separate file
11	Technology List	Azure cloud database and web API
12	Frequency	Continous application usage
13	Misc	User friendly error messages when data retrival fails

There is used a general framework for all the get requests except one. The general framework code is shown in figure 8.8.

```
47      private async Task<T> GetAsync<T>(string url)
48      {
49          var fullUrl = $"{_baseApiAddress}{url}"; // Kombiner baseadresse med URL
50          try
51          {
52              var response = await _httpClient.GetAsync(fullUrl);
53
54              if (response.IsSuccessStatusCode)
55              {
56                  var jsonResult = await response.Content.ReadAsStringAsync();
57                  var result = JsonConvert.DeserializeObject<T>(jsonResult);
58                  return result;
59              }
60              else
61              {
62                  // Log response details if not successful
63                  var errorContent = await response.Content.ReadAsStringAsync();
64                  throw new HttpRequestException($"Unable to retrieve data: {response.StatusCode}. Details: {errorContent}");
65              }
66          }
67          catch (Exception ex)
68          {
69              // Log detailed exception information
70              Console.WriteLine($"Exception caught while making GET request to {fullUrl}: {ex.Message}");
71              throw;
72          }
73      }
```

Figure 8.8: Snip of code for the framework for getting data from the database via the database API

In this code a uniform resource locator (URL) is defined as parameter for the *GetAsync* method. This URL is the request URL for the database API. The response is in JSON-format and converted using the *JsonConvert* class. The result is returned or if no response an error is thrown to the user interface.

In figure 8.9 a snip of one of the methods used to request specific data from the database API using the general *GetAsync* method.

```
105     public Task<List<BuildingTemperatureItem>> GetBuildingTempsAsync(string url) => GetAsync<List<BuildingTemperatureItem>>(url);
```

Figure 8.9: Snip of code showing the *GetBuildingTempsAsync* using the *GetAsync* method to request data form the database API

The URL received as a parameter from the *GetBuildingTempsAsync* is used as a parameter for the general *GetAsync* method.

The methods, in other parts of the application, used to call the methods in the *DatabaseWebAPIServices* are varying since the requests and responses are varying. In figure 8.10 an example of the code that could be used to call the *GetBuildingTempsAsync* method is shown.

```
string apiUrl = "/api/BuildingTemperatureItems";
try
{
    var temperatureItems = await _databaseWebAPIServices.GetBuildingTempsAsync(apiUrl);
    TemperatureListView.ItemsSource = temperatureItems;
}
catch (Exception ex)
{
    await HandleError(ex, "Failed to fetch temperature data");
}
```

Figure 8.10: Snip of code showing the call of the *GetBuildingTempsAsync* method

In this code you can see the generation of the URL using the *BaseApiUrl* and the extension needed to request the building temperature items defined in the database API. It also shows how the converted JSON data is shown in a list view and the content of the error message if the method failed.

The method for getting the unit of measure (UoM) for one attribute in the table containing UoMs for the weather forecast is a little different than the others. The reason for this is that in addition to the URL the name of the attribute is included as a parameter for the method. The code for the method is shown in figure 8.11.

```
118     public async Task<string> GetUoMForAttributeAsync(string url, string attribute)
119     {
120         var fullUrl = $"{_baseApiAddress}{url}{Uri.EscapeDataString(attribute)}";
121         try
122         {
123             var response = await _httpClient.GetAsync(fullUrl);
124
125             if (response.IsSuccessStatusCode)
126             {
127                 return await response.Content.ReadAsStringAsync();
128             }
129             else
130             {
131                 // Log error details
132                 var errorContent = await response.Content.ReadAsStringAsync();
133                 throw new HttpRequestException($"Unable to retrieve UoM: {response.StatusCode}. Details: {errorContent}");
134             }
135         }
136         catch (Exception ex)
137         {
138             // Log detailed exception information
139             Console.WriteLine($"Exception caught while making UoM request to {fullUrl}: {ex.Message}");
140             throw;
141         }
142     }
```

Figure 8.11: Snip of code for the *GetUoMForAttributeAsync* method

As we can see from the code the method has two parameters. This is the URL and the attribute parameters. These are combined and used to create the request URL for database API. The response UoM is returned as a text string.

8.6 Use case: Post database data

The goal of this use case is to store data in the cloud database. To make the Azure SQL database available a web API is being used.

In this use case, the *DatabaseWebAPIServices* class serves as the primary service between the client application and the database API. By encapsulating API requests, it allows the client application to post multiple data types from a database via the API without direct database interaction.

In figure 8.12 the system sequence diagram for the post database use case is shown.

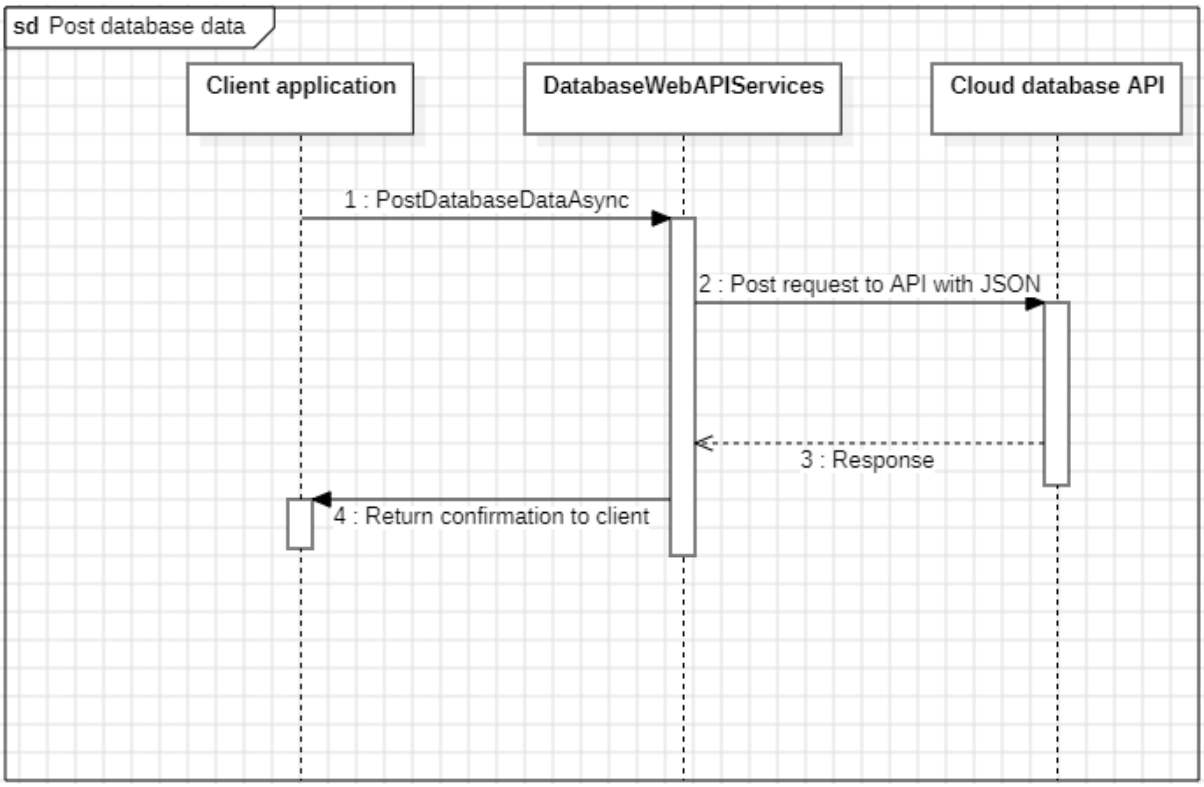


Figure 8.12: System sequence diagram for the use case: Post database data

In table 8.3 the use case document for the post database data is shown.

Table 8.3: FDUCD for use case: Post database data

1	Use case name	Post database data
2	Scope	To submit specific types of data to the database using a REST API call.
3	Level	User-goal
4	Primary actor	Application user
5	Stakeholders	Application user: Want to post data to database
		Cloud database API: Store data from client
		System administrator: Ensures API access
		Developer: Integrates API
6	Precondition	The cloud database and web API are operational and accessible
		The client application has valid data ready to be sent in JSON format
		The client has the necessary permissions to access the API endpoint for posting
		The database web API services class is correctly instantiated in the application and configured to use the correct endpoint URLs for each data type
7	Success Guarantee	The data is successfully posted to the database, and a confirmation message or response is received from the server
8	Main Success Scenario	1. User/Client application initiates post request
		2. Service executes post request
		3. Data posting and response
		4. Return confirmation to client
		5. Client recives confirmation
9	Extensions	1a. If the server returns a non-success status code , the method throws an exception
		1b. The exception is handled by the client application, and an appropriate error message is displayed or logged for troubleshooting
		2a. If there is a network or connection issue, the HTTP post request fails
		2b. An exception is thrown, which is caught and logged by the client application for user notification or retry logic
10	Special Requirements	Data is posted frequently
		API key stored in separate file
11	Technology List	Azure cloud database and web API
12	Frequency	Continous application usage
13	Misc	User friendly error messages when data retrival fails

There is used a general framework for all the post request methods. The general framework is shown in figure 8.13.

```
76      private async Task<string> PostAsync<T>(string apiUrl, T data)
77      {
78          var fullUrl = $"{_baseApiAddress}{apiUrl}";
79          try
80          {
81              var jsonData = JsonConvert.SerializeObject(data);
82              var content = new StringContent(jsonData, Encoding.UTF8, "application/json");
83              var response = await _httpClient.PostAsync(fullUrl, content);
84
85              if (response.IsSuccessStatusCode)
86              {
87                  return await response.Content.ReadAsStringAsync(); // Return API confirmation
88              }
89              else
90              {
91                  // Log error details from the response
92                  var errorContent = await response.Content.ReadAsStringAsync();
93                  throw new HttpRequestException($"Failed to post data: {response.StatusCode}. Details: {errorContent}");
94              }
95          }
96          catch (Exception ex)
97          {
98              // Log detailed exception information
99              Console.WriteLine($"Exception caught while making POST request to {fullUrl}: {ex.Message}");
100             throw;
101         }
102     }
```

Figure 8.13: Snip of code for the framework for posting data to the database via database API

In this code the *PostAsync* method is getting the request URL and data as parameters. These are converted to JSON data using the *JsonConvert* method. If the post request is successful, the response is returned or if not, an error is thrown.

In figure 8.14 a snip of one of the methods used to request specific data from the database API using the general *PostAsync* method.

```
145     public Task<string> PostBuildingTemperatureAsync(string apiUrl, BuildingTemperatureItem item) => PostAsync(apiUrl, item);
```

Figure 8.14: Snip of code showing the *PostBuildingTemperatureAsync* using the *PostAsync* method to request data form the database API

The URL received as a parameter from the *PostBuildingTemperatureAsync* is used as a parameter for the general *PostAsync* method.

8.7 Use case: View current data

The purpose of this use case is to display the most recent data on the GUI. The application should provide users with latest sampled data, showing the latest readings for temperature, humidity, energy consumption, and the predicted energy consumption.

Table 8.4: FDUCD for use case: view current data

1	Use case name	View current data
2	Scope	View current values on GUI
3	Level	User-goal
4	Primary actor	Application user
5	Stakeholders	Application user: Wants to see latest sampled sensor data, energy consumption data data, and latest energy prediction
		Wheater API: Provides live wheater data
		Cloud database API: Provides live sensor data
6	Precondition	Application running
		Internet connection
		Database operational
		Database WEB API operational
7	Success Guarantee	User of application is successfully able to view latest sampled sensor data
8	Main Success Scenario	1: Application running with internet
		2: User select live data pages
		3: Application sends request to database API for latest data
		4: Application receives latest data
		6: Application presents latest data on GUI
9	Extensions	1a: No internet connection: System message: "No internet connection".
		3a: If the request fails the application throws an exeption
		3b: System message: Unable to fetch data from WEB API.
10	Special Requirements	The data should be requested every minute.
11	Technology List	Mobile phone application, Third party database API.
12	Frequency	Continious application usage
13	Misc	Feedback to application user when connection to internet or WEB API fails.

In figure 8.15 the system sequence diagram for the view current data use case is shown.

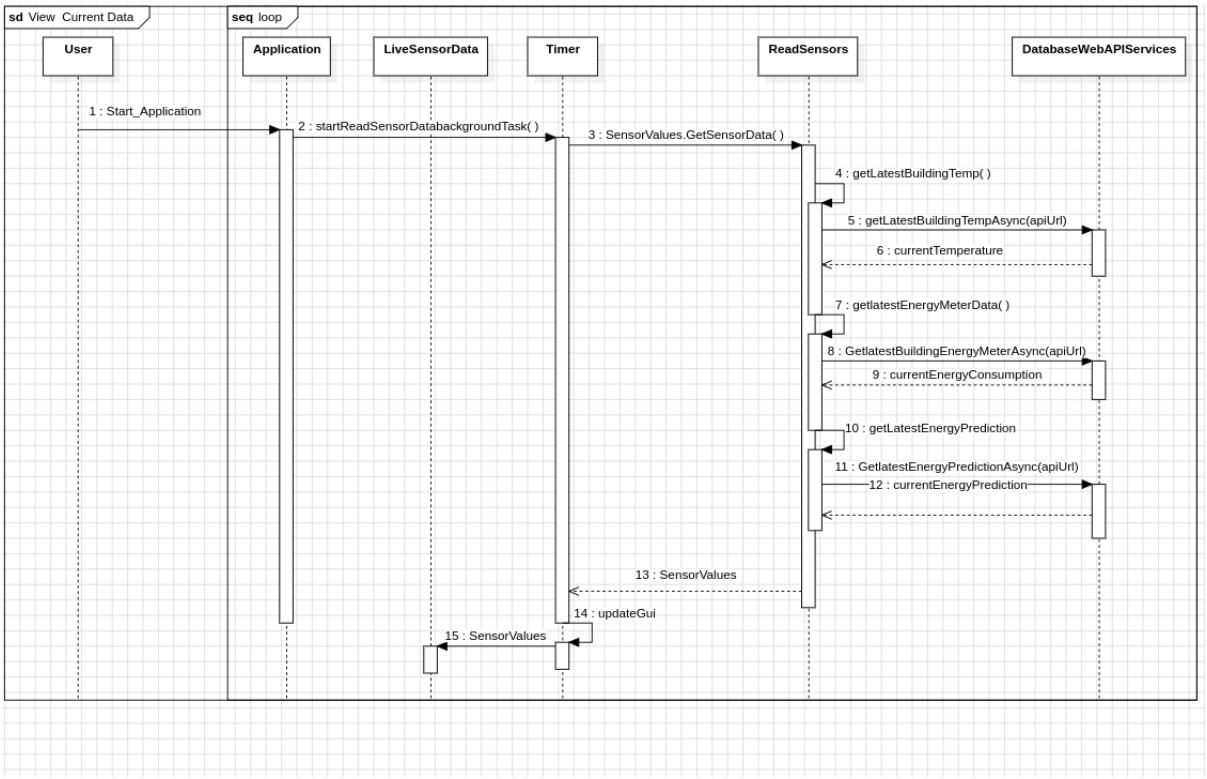


Figure 8.15: System sequence diagram view current data

When the application is started, a background task named *startReadSensorDataBackgroundTask* is initiated. This task is executed once every minute and ensures cyclic request of current sensor data.

This background task is called from the *GetSensorData* method, which is part of the *ReadSensors* class. This method interacts with the *DatabaseWebAPIServices* class to fetch the latest sensor data. The retrieved data is used by the *LiveSensorData* class which is responsible for presenting the data on the GUI.

```
47      private void StartReadSensorDataBackgroundTask()
48      {
49          Task.Run(async () =>
50          {
51              while (_isRunning)
52              {
53                  SensorValues.getSensorData();
54                  await Task.Delay(TimeSpan.FromSeconds(60));
55              }
56          });
57      }
```

Figure 8.16: Background task polling current sensor data

Figure 8.16 shows the implementation for the background task that polls sensor data every minute. This method is defined within the *App.xaml.cs* class, which is the class that initializes the application and is the entry point for the application logic.

```
14        public static ReadSensors SensorValues { get; set; } = new ReadSensors();
```

Figure 8.17: Public static variable storing current sensor data

As seen in figure 8.17, the sensor values are stored in *SensorValues* object, which is an instance of the *ReadSensors* class. The object is declared public, so that other classes can access the sensor data.

The *getSensorData* method is seen in figure 8.18. This method is cyclically called from the *startReadSensorDataBackgroundTask*. This method also calls other methods that is fetching historical data.

```
69        public void getSensorData()
70        {
71            GetLatestBuildingTemp();
72            GetLatestBuildingRelHumidity();
73            GetLatestEnergyMeterData();
74            GetAllBuildingTemp();
75            GetAllBuildingRelHum();
76            GetAllBuildingKwh();
77            GetAllWeatherForecasts();
78            GetLatestEnergyPrediction();
79        }
```

Figure 8.18: getSensorData from ReadSensors class

Figure 8.19 shows the *GetLatestBuildingTemp* method which retrieves the most recent temperature data. It sends a request to the *datebaseWebAPIServices* class using the *GetLatestBuildingTempAsync* method.

```
191       private async void GetLatestBuildingTemp()
192       {
193           string apiUrl = "/api/BuildingTemperatureItems/latest";
194           try
195           {
196               var latestTemp = await _databaseWebAPIServices.GetLatestBuildingTempAsync(apiUrl);
197               sensor.TT01 = latestTemp.Temp1;
198               sensor.TT02 = latestTemp.Temp2;
199               sensor.TT03 = latestTemp.Temp3;
200               sensor.TT04 = latestTemp.Temp4;
201               sensor.TT05 = latestTemp.Temp5;
202               sensor.TT_AVG = (latestTemp.Temp1 + latestTemp.Temp2 + latestTemp.Temp3 + latestTemp.Temp4 + latestTemp.Temp5) / 5.0;
203               sensor.TT_DT = latestTemp.TempDateTime;
204           }
205           catch (Exception ex)
206           {
207               Debug.WriteLine(ex + " Failed to fetch temperature data");
208               //await HandleError(ex, "Failed to fetch temperature data");
209           }
210       }
```

Figure 8.19*: getLatestBuildingTemp* method from ReadSensors Class

The fetched data is stored in the following class defined from within the *ReadSensors* class.

```
22      public Livesensors sensor { get; set; } = new Livesensors();
23      public class Livesensors
24      {
25          public double TT01 { get; set; }
26          public double TT02 { get; set; }
27          public double TT03 { get; set; }
28          public double TT04 { get; set; }
29          public double TT05 { get; set; }
30          public double TT_AVG { get; set; }
31          public DateTime? TT_DT { get; set; }
32          public double RHT01 { get; set; }
33          public double RHT02 { get; set; }
34          public double RHT03 { get; set; }
35          public double RHT04 { get; set; }
36          public double RHT_AVG { get; set; }
37          public DateTime? RHT_DT { get; set; }
38          public double KWH01 { get; set; }
39          public DateTime? KWH_DT { get; set; }
40          public float Current_Prediciton { get; set; }
41      }
```

Figure 8.20: *Livesensors* class, used for storing latest sensor data

8.8 Use case: View historical data

The purpose of this use case is to display historical data. The application should provide users with historical sampled data, including historical temperature, humidity, energy consumption, and predicted energy consumption.

Table 8.5 FDUCD for use case: View historical data

1	Use case name	View historical data
2	Scope	View historical data on GUI
3	Level	User-goal
4	Primary actor	Application user
5	Stakeholders	Application user: Wants to see historical trend of relevant data
		Cloud database API: Provider of historical data from database
6	Precondition	Application running
		Internet connection
		Cloud database operational
		Web API operational
7	Success Guarantee	User of application is successfully able to view historical sensor data
8	Main Success Scenario	1: Application running connected to internet
		2: User selects sensor to retrive historical data from
		4: Application sends request to database API for historical data
		5: Histoical data is returned from database API
		6: Historical data is presented in graph.
9	Extensions	4a: If the server returns a non-success status code, the GetAsync<T> method throws an HttpRequestException
		4b: The application displays an appropiate error message
10	Special Requirements	n.a
11	Technology List	Azure cloud database, web API
12	Frequency	Continious application usage
13	Misc	Feedback to user when fetching data from WEB API fails.

Figure 8.21 shows the system sequence diagram for the view historical data use case.

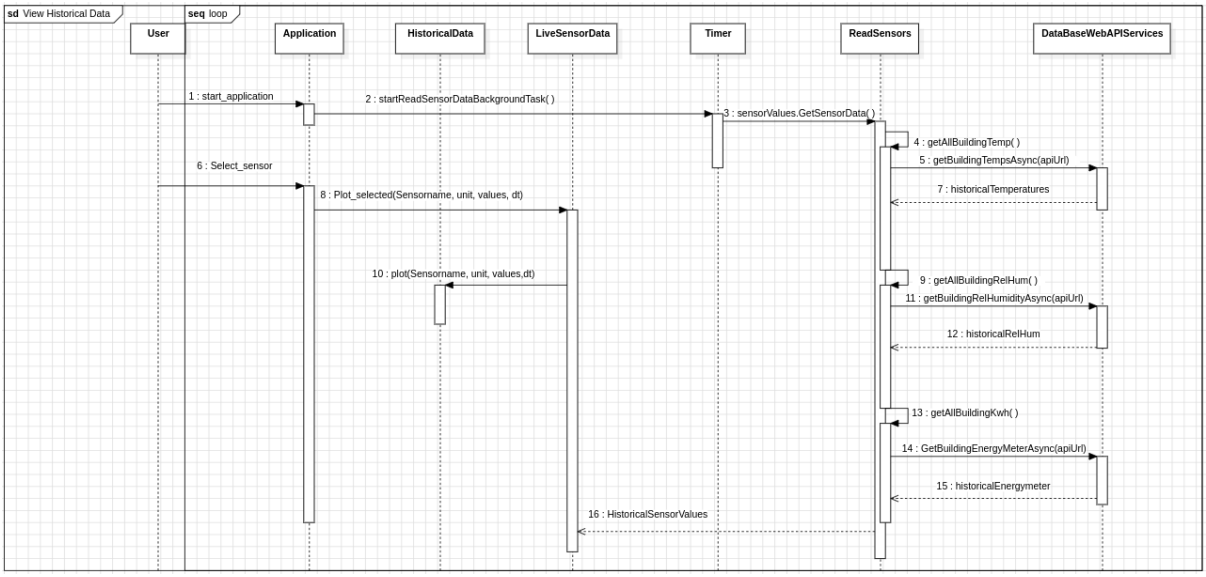


Figure 8.21: System sequence diagram view historical data

When the application is started, a background task named *startReadSensorDataBackgroundTask* is initiated. This task is executed once every minute and ensures cyclic request of historical data. This background task uses the *GetSensorData* method, which is part of the *ReadSensors* class. This method interacts with the *DatabaseWebAPIServices* class to fetch the historical data.

The retrieved data is used by the class *LiveSensorData*, which initiates plotting of user selected sensor by using the class named *HistoricalData* which is responsible for presenting the data on the GUI. The plot function uses the *syncfusion Maui charts* library to plot.

The *startReadSensorDataBackgroundTask* and *GetSensorData* methods were shown in detail for the use case view current data.

Figure 8.19 shows the method named *GetAllBuildingRelHum* which is called from the *LiveSensorData* class. It makes an API request of humidity data to the *databaseWebAPIServices* class.

```
154     private async void GetAllBuildingRelHum()
155     {
156         string apiUrl = "/api/BuildingRelativeHumidityItems";
157         try
158         {
159             List<BuildingRelativeHumidityItem> relHumItems = await _databaseWebAPIServices.GetBuildingRelHumidityAsync(apiUrl);
160             // Last 14 days
161             List<BuildingRelativeHumidityItem> last14Items = relHumItems.OrderByDescending(item => item.Id).Take(14).ToList();
162             historical.RHT01 = last14Items.Select(item => item.RelHumidity1).ToArray();
163             historical.RHT02 = last14Items.Select(item => item.RelHumidity1).ToArray();
164             historical.RHT03 = last14Items.Select(item => item.RelHumidity1).ToArray();
165             historical.RHT04 = last14Items.Select(item => item.RelHumidity1).ToArray();
166         }
167         catch (Exception ex)
168         {
169             Debug.WriteLine(ex + " Failed to fetch rel hum data");
170         }
171     }
```

Figure 8.22: *Livesensors* class, used for fetching and storing historical humidity data

The retrieved data is stored in the following class defined from within the *ReadSensors* class.

```
42      public HistoricData historical { get; set; } = new HistoricData();
43      public class HistoricData
44      {
45          public float[] TT01 { get; set; }
46          public float[] TT02 { get; set; }
47          public float[] TT03 { get; set; }
48          public float[] TT04 { get; set; }
49          public float[] TT05 { get; set; }
50          public float[] RHT01 { get; set; }
51          public float[] RHT02 { get; set; }
52          public float[] RHT03 { get; set; }
53          public float[] RHT04 { get; set; }
54          public float[] KWH01 { get; set; }
55      }
```

Figure 8.23: *HistoricData* class, storing arrays historical data

Figure 8.24 shows the function that is responsible for initiating plotting of one of the temperature sensors. The function is defined within the *LiveSensorData* class and is called when the user touches the label. Touching the label creates a instance of the *HistoricalData* class, with corresponding sensor values, timestamps, name and unit as input.

```
93      private async void OnLabelTapped_TT01(object sender, EventArgs e) => await Navigation.PushAsync(
94          new HistoricalData_1("First floor", "Temperature[°C]", App.SensorValues.historical.TT01, App.SensorValues.sensor.TT_DT));
```

Figure 8.24: Plotting sensor when tapping label on GUI

Figure 8.25 shows the method for plotting the graph on the GUI.

```
20      public HistoricalData_1(string sensorName, string unit, float[] values, DateTime? dt)
21      {
22          InitializeComponent();
23          SensorName = sensorName;
24          AxisTitle.Text = unit;
25
26          string[] TimeStamps = new string[values.Length];
27          Data = [];
28          if (dt != null)
29          {
30              for (int i = 0; i < values.Length; i++)
31              {
32                  TimeStamps[i] = dt.Value.AddDays(-i).ToString("dd.MM.yyyy");
33                  Data.Add(new ChartModel
34                  {
35                      Time = TimeStamps[i],
36                      Value = values[i]
37                  });
38              }
39              BindingContext = this;
40          }
41      }
```

Figure 8.25: *HistoricData* class, storing arrays historical data

8.9 Use case: Predict future energy consumption

The purpose of the use case “Predict future energy consumption” is to fetch predictions of the energy consumption in the given residence based on previous historical data as well as future weather forecasts. This is done through an API-request to a GPT3.5 model running on azure AI services. The returned data is pushed to the database, pulled to the mobile application and displayed in the energy page. Details of the use case can be found in the FDUCD in Table 8.6 and the sequence diagram in figure 8.23.

Table 8.6 FDUCD for use case: Predict future energy consumption

1	Use case name	Predict Furure Energy Consumption
2	Scope	Predict future energy consumption based on historical and future data of consumption, outdoor and indoor climate
3	Level	User-goal
4	Primary actor	Application user
5	Stakeholders	Application User: Wants to view predicted energy consumption
		Cloud Prediction API: Provides AI model for predictions
		System administrator: Ensures API access
		Developer: Integrates API
6	Precondition	Application is online and running
		Cloud prediction API service is functioning
		Valid API key is provided
		Training data and weather forecast available
7	Success Guarantee	A predicted value for energy consumption is returned for the correct location based on the input data
8	Main Success Scenario	1. Application is running and connected to the internet
		2. Predicted values are not already fetched
		3. Predicted energy consumption page loaded
		4. Training data and weather forecast is collected
		5. Application sends a request to the Prediction API with the provided data
		6. Prediction API returns the predicted energy consumption
		7. Predicted energy consumption is uploaded to database
		8. The predicted energy consumption is displayed to the user
9	Extensions	1a. No internet connection: System shows "No internet connection"
		2a. Values already fetched: Skip to avoid unnecessary request
		4a. Missing data: Error message shown.
		5a. Invalid API key: System shows "Invalid API key"
		5b. API Limit exceeded: System shows "API limit exceeded"
10	Special Requirements	API key stored in securely in seperate file
11	Technology List	Third-party API: Meteorologisk institutt; api.met.no "Locationforecast"
		Database with previous energy consumption, indoor and outdoor climate
12	Frequency	Intervals: 24 hour
13	Misc	User-friendly error messages when data retrieval fails

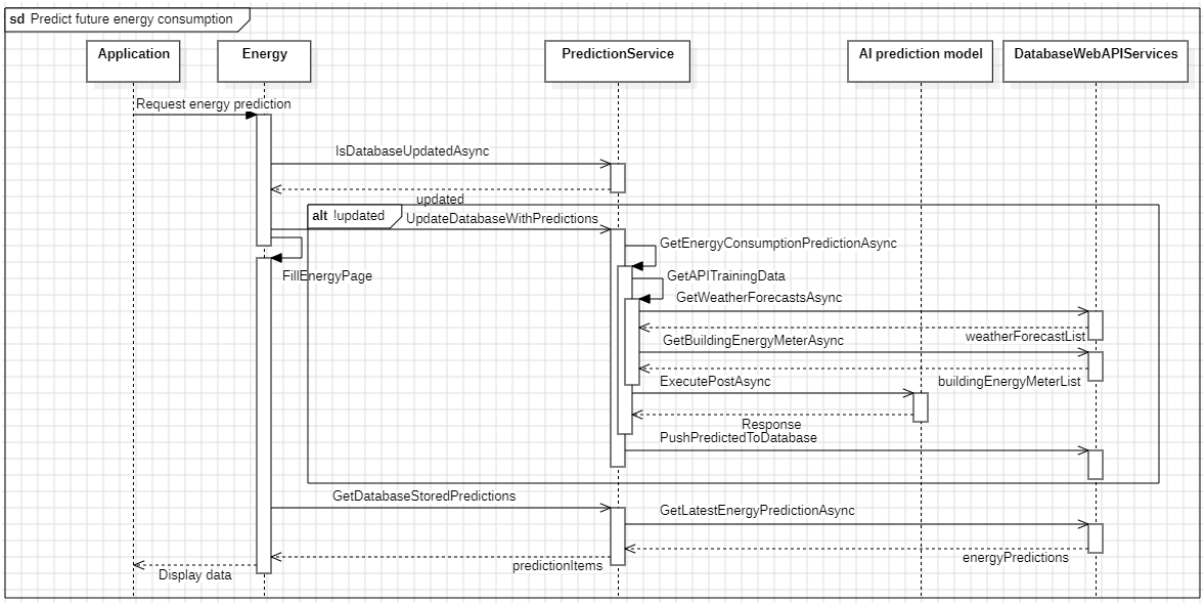


Figure 8.23: Sequence diagram for the predict future energy consumption

The use case is based on the method *GetEnergyConsumptionPredictionAsync* that collects training data from the database and weather forecast based on dates given as an input. It then constructs this into a large input prompt for the ai model and returns the content of the reply. The first few lines of the method where data is collected is shown in figure 8.24. This method will collect prediction for multiple days ahead in one request, and it will only run if the requested data is not already present in the database. This is to minimize unnecessary requests and reduce costs.

```
public async Task<string> GetEnergyConsumptionPredictionAsync(List<DateTime> days)
{
    /*
        Method that returns predicted energy consumption for a given input day
        Day must be within range of api (typically 10 days ahead)
     */


    string promptData = await GetAPITrainingData();
    StringBuilder predictData = new StringBuilder();
    foreach (var day in days) { predictData.AppendLine(await GetInputDataForPrediction(day)); }
```

Figure 8.24 Method GetEnergyConsumptionPredictionAsync

9 Results of mobile application development

In this chapter the results of the project will be presented. This will focus on the presentation of information in the mobile application. The database API with the SQL database and the AI energy prediction model has no interface directly to the user, but data received from these services are presented in the mobile application.

9.1 Main page of mobile application

In the main page of the mobile application a dashboard with current values is displayed. The current values are the latest values that are updated in the database.

For the indoor environment in the building the average temperature and relative humidity for the latest set of measurements are shown. This data would be based on sensors in the building.

For the outdoor environment the latest temperature and relative humidity for the location of the building is shown based on the current values of the latest forecast received. Since there were not outdoor sensors at the selected building, the data presented is based on the forecast received from the weather forecast API.

Under energy, the latest energy value from the energy meter in the building is shown and in addition the energy prediction for the first day of the latest energy prediction.

There is also a drop-down menu on the top of the page where you can select to navigate to the different pages directly.

In the following pages the following pictures and screenshots are shown:

- Figure 9.1 is a picture of Samsung Galaxy S21 Ultra running application displaying the main page.
- Figure 9.2 is a screenshot of the main page of application on a physical Samsung Galaxy S21 Ultra.
- Figure 9.3 is a screenshot of the drop-down list of main page on a physical Samsung Galaxy S21 Ultra
- Figure 9.4 is a screenshot of the main page of application on WinUI running on a laptop.
- Figure 9.5 is a screenshot of the drop-down list of main page on WinUI running on a laptop.
- Figure 9.6 is a screenshot of the main page of application on an iPhone 16 simulator running on a laptop.
- Figure 9.7 is a screenshot of the drop-down list of main page on an iPhone 16 simulator running on a laptop.

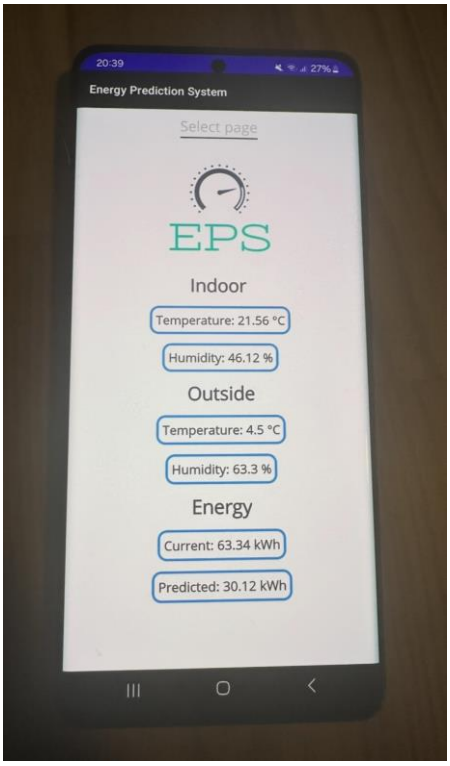


Figure 9.1: Picture of Samsung Galaxy S21 Ultra running application displaying the main page

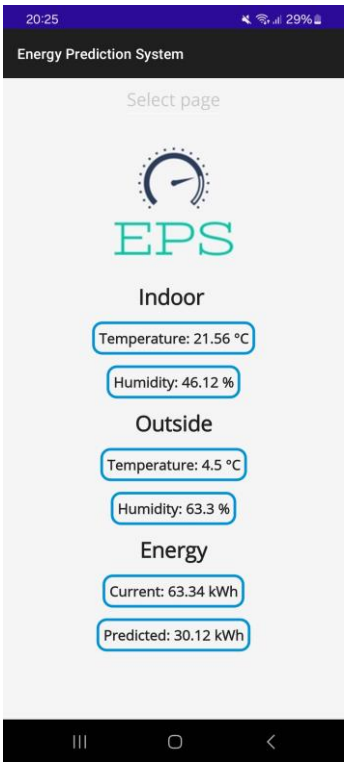


Figure 9.2: Screenshot of the main page of application on a physical Samsung Galaxy S21 Ultra

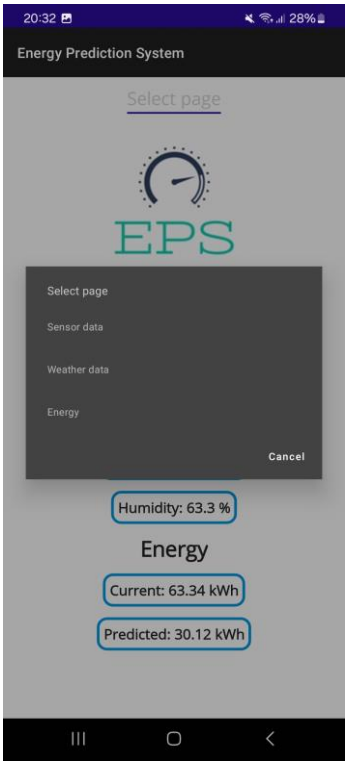


Figure 9.3: Screenshot of the drop-down list of main page on a physical Samsung Galaxy S21 Ultra

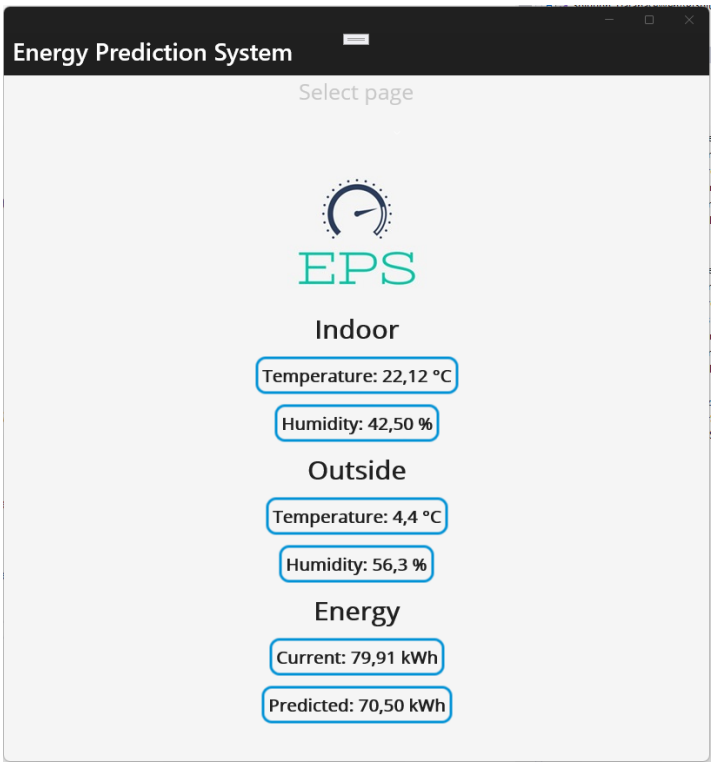


Figure 9.4: Screenshot of the main page of application on WinUI running on a laptop

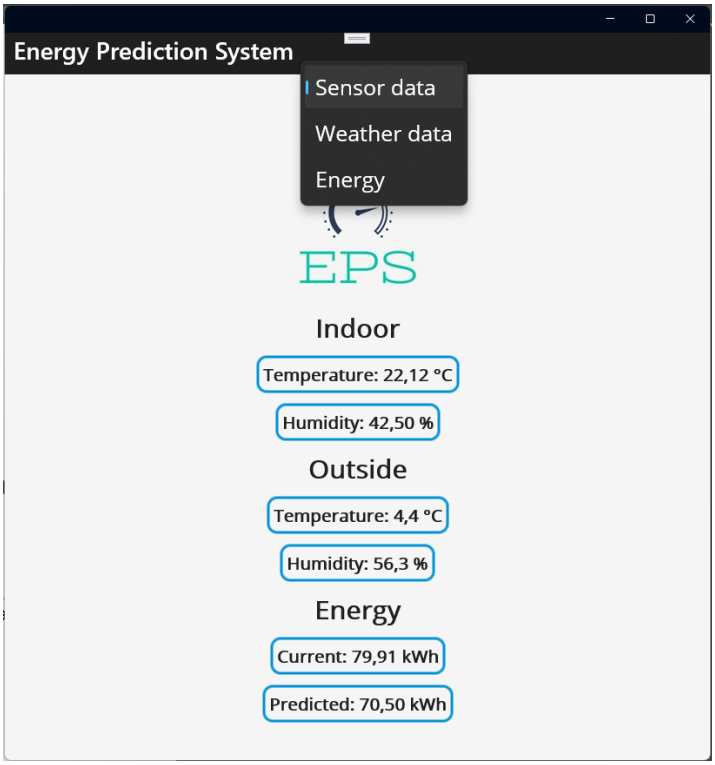


Figure 9.5: Screenshot of the drop-down list of main page on WinUI running on a laptop

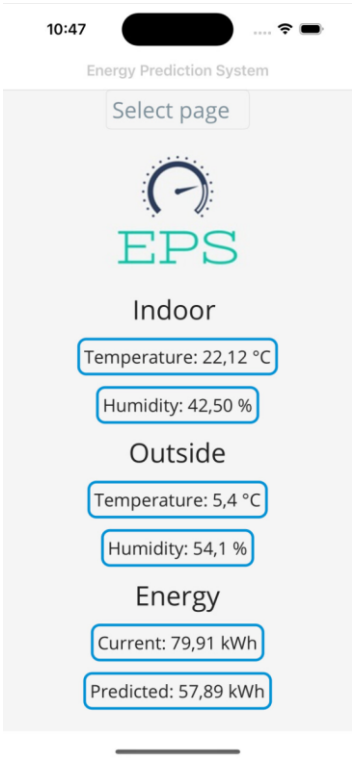


Figure 9.6: Screenshot of the main page of application on an iPhone 16 simulator running on a laptop

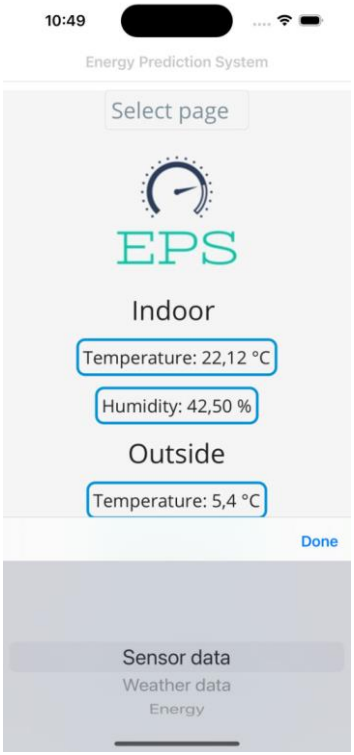


Figure 9.7: Screenshot of the drop-down list of main page on an iPhone 16 simulator running on a laptop

9.2 Page to present current data received from database API

When in the main page click on the indoor temperature or humidity field, or by selecting the sensor data from the drop-down menu, a page showing the current (latest posted) data retrieve from the SQL database.

For the building used in the project it is five temperatures for different rooms, four humidity values for different rooms, and one energy meter for building. In the bottom of each group that date and time of the last posting into the database is shown. At the time of this report the data in the database is not being updated directly from the building control system, but manually entered.

If you in this page click on one of the values, a page with an historical graph will open. This page is described and shown in chapter 9.4 of this report.

In the following pages the following screenshots are shown:

- Figure 9.8 is a screenshot of the sensor data page on physical Samsung Galaxy S21 Ultra.
- Figure 9.9 is a screenshot of the sensor data page of application in WinUI running on a laptop.
- Figure 9.10 is a screenshot of the sensor data page of the application on an iPhone 16 simulator running on a laptop.

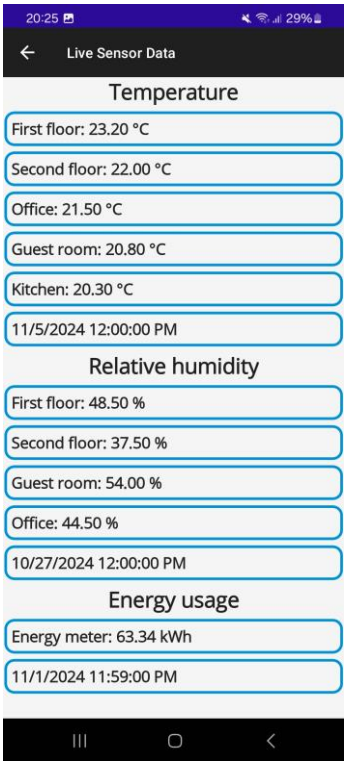


Figure 9.8: Screenshot of the sensor data page on physical Samsung Galaxy S21 Ultra

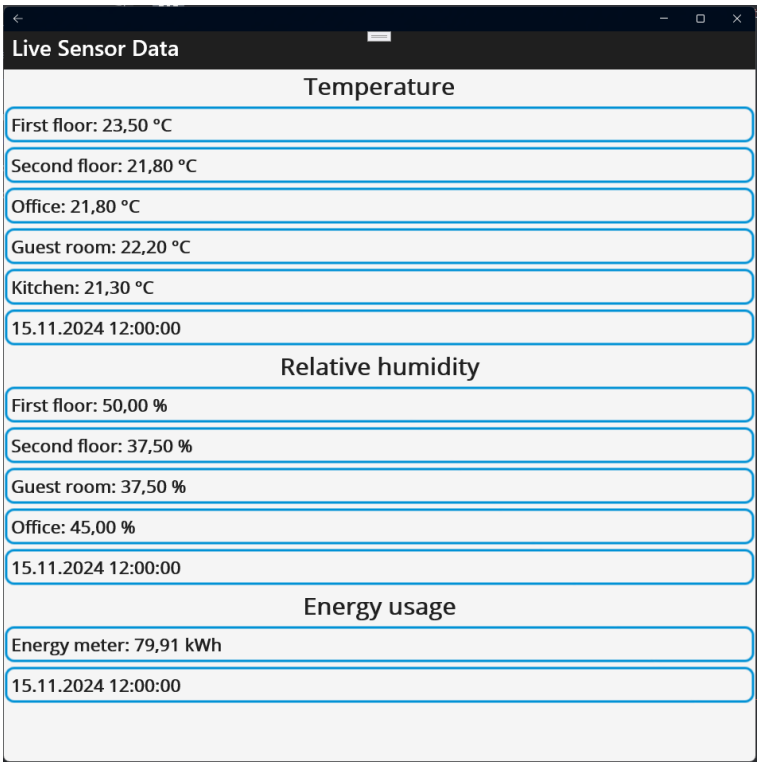


Figure 9.9: Screenshot of the sensor data page of application in WinUI running on a laptop

Figure 9.10: Screenshot of the sensor data page of application on an iPhone 16 simulator running on a laptop

9.3 Page to present current data received from weather forecast API

If you on the main page click on the outdoor temperature or humidity values, or select weather data from the drop-down menu, you will open a page that displays the latest weather data in the database. The application will try to collect weather data when the application is started and every hour when it is running.

The weather data displayed in the page is a selection of the data available in the MET weather API *locationforecast* [41].

If you in this page, click on one of the values, a graph showing the historical values for this parameter will be shown. This page is described in chapter 9.4 of this report.

In the following pages the following screenshots are shown:

- Figure 9.11 is a screenshot of the current weather data page on physical Samsung Galaxy S21 Ultra.
- Figure 9.12 is a screenshot of the current weather data page of application in WinUI running on a laptop.
- Figure 9.13 is a screenshot of the current weather data page of application on an iPhone 16 simulator running on a laptop.

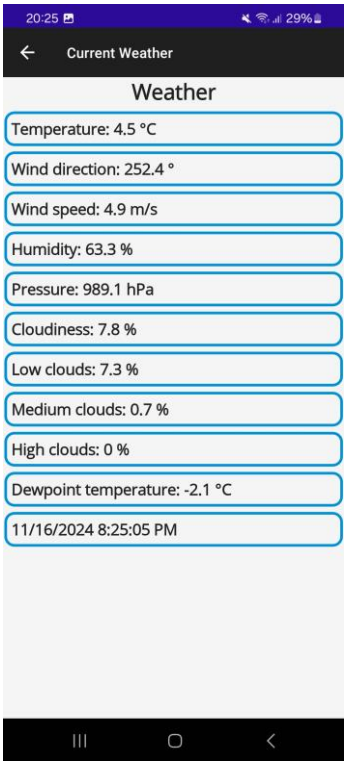


Figure 9.11: Screenshot of the current weather data page on physical Samsung Galaxy S21 Ultra

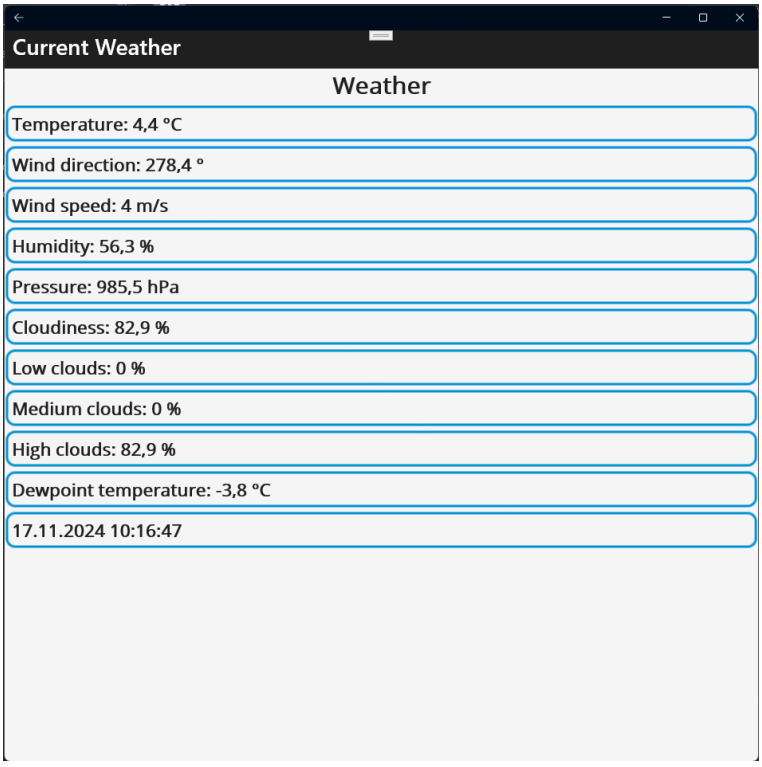


Figure 9.12: Screenshot of the current weather data page of application in WinUI running on a laptop

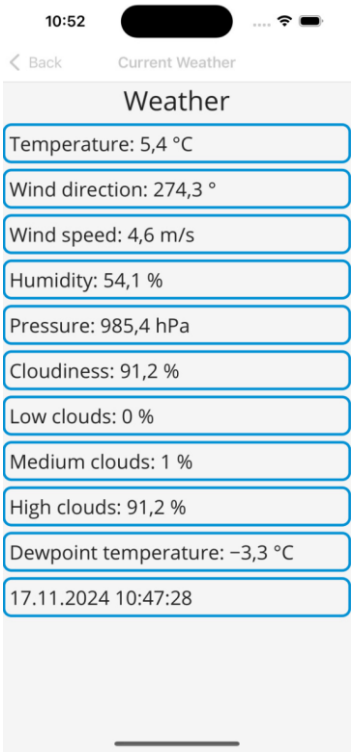


Figure 9.13: Screenshot of the current weather data page of application on an iPhone 16 simulator running on a laptop

9.4 Examples of pages showing graphs of historical data

When clicking on values in the main, sensor data, and current weather pages, a historical graph of these values will be shown. In the version of the application at the time of this report, the graph is for a fixed period.

In the following pages the following screenshots are shown:

- Figure 9.14 is a screenshot of a historical graph of energy consumption displayed on a physical Samsung Galaxy S21 Ultra.
- Figure 9.15 is a screenshot of a historical graph of energy consumption displayed on a physical Samsung Galaxy S21 Ultra while flipped.
- Figure 9.16 is a screenshot of a historical graph of temperature on second floor displayed on a physical Samsung Galaxy S21 Ultra while flipped.
- Figure 9.17 is a screenshot of a historical graph of energy consumption displayed in WinUI running on a laptop.
- Figure 9.18 is a screenshot of a historical graph of temperature on first floor displayed in WinUI running on a laptop.
- Figure 9.19 is a screenshot of a historical graph of relative humidity in the office displayed on an iPhone 16 simulator running on a laptop.
- Figure 9.20 is a screenshot of a historical graph of outside temperature displayed on an iPhone 16 simulator while flipped running on a laptop.

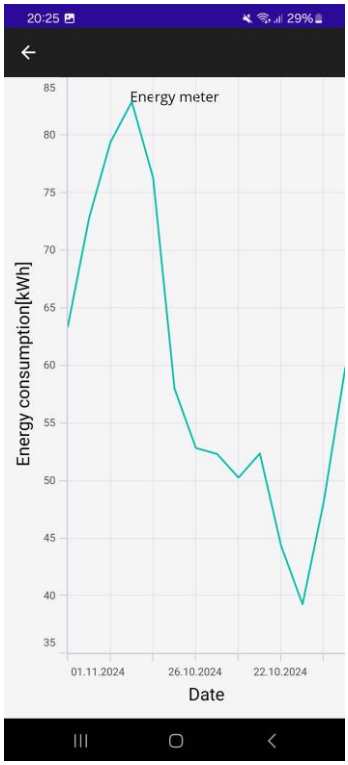


Figure 9.14: Screenshot of a historical graph of energy consumption displayed on a physical Samsung Galaxy S21 Ultra

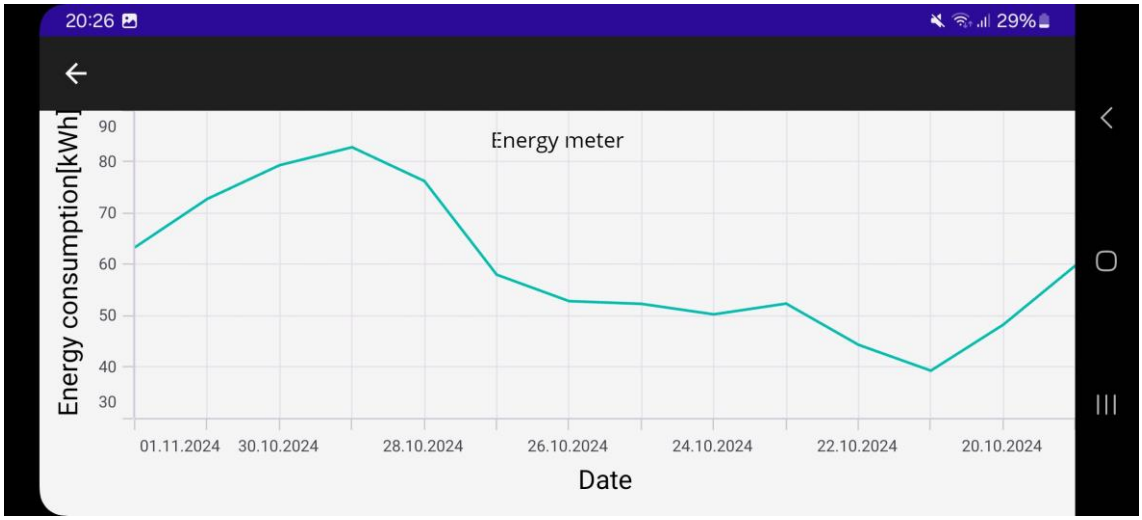


Figure 9.15: Screenshot of a historical graph of energy consumption displayed on a physical Samsung Galaxy S21 Ultra while flipped

Figure 9.16: Screenshot of a historical graph of temperature on second floor displayed on a physical Samsung Galaxy S21 Ultra while flipped

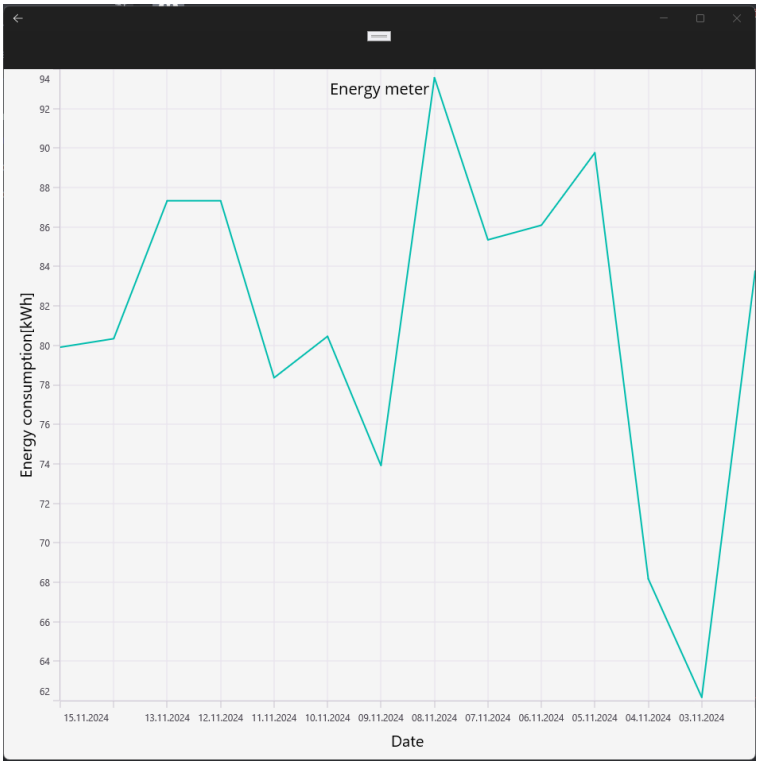


Figure 9.17: Screenshot of a historical graph of energy consumption displayed in WinUI running on a laptop

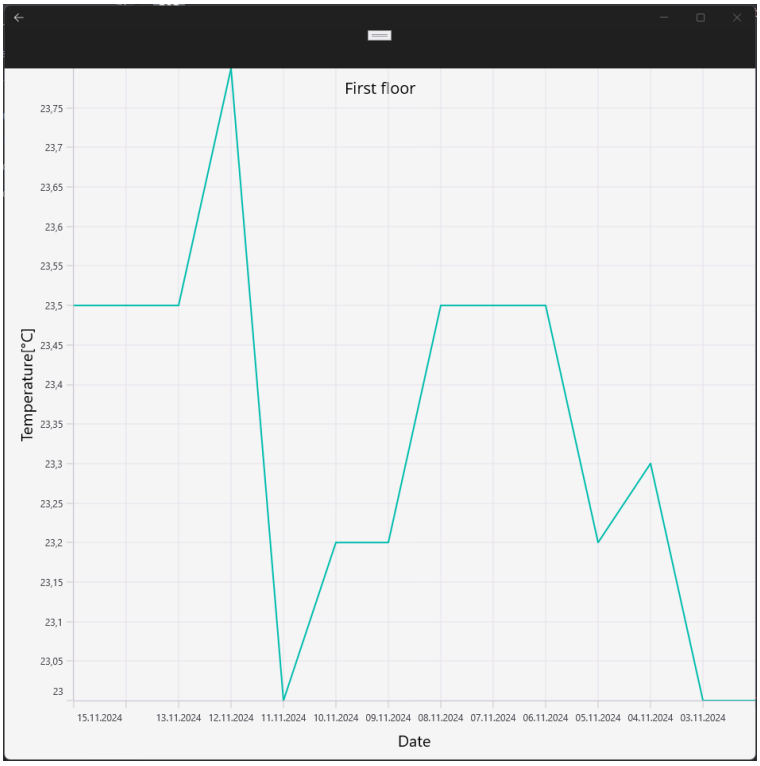


Figure 9.18: Screenshot of a historical graph of temperature on first floor displayed in WinUI running on a laptop

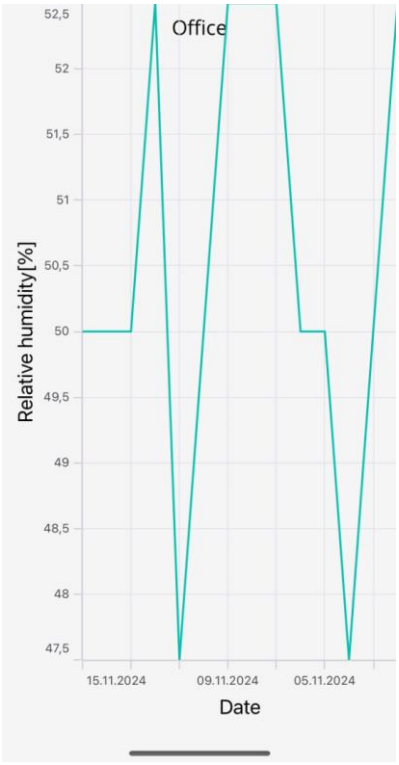


Figure 9.19: Screenshot of a historical graph of relative humidity in the office displayed on an iPhone 16 simulator running on a laptop

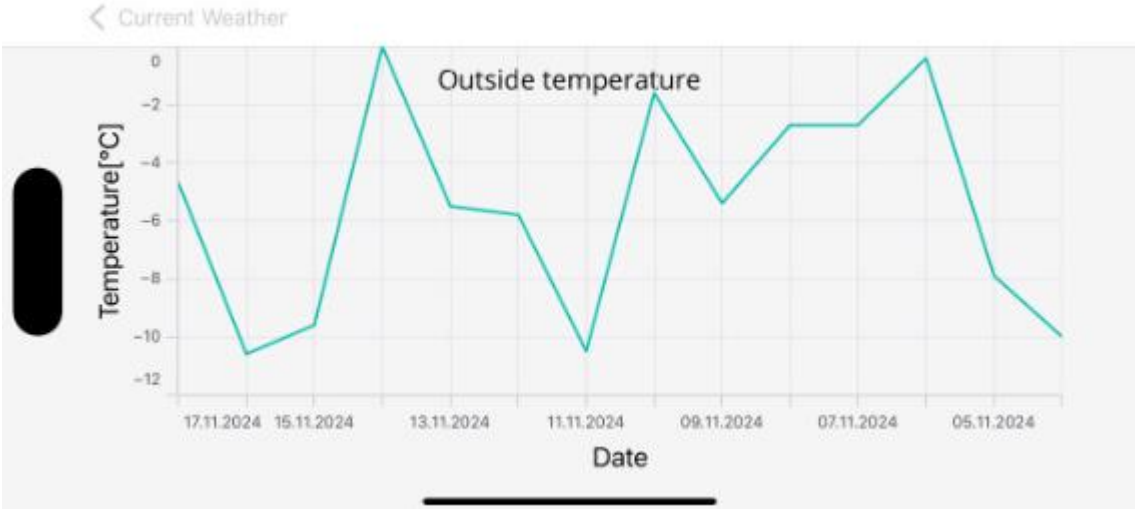


Figure 9.20: Screenshot of a historical graph of outside temperature displayed on an iPhone 16 simulator while flipped running on a laptop

9.5 Page to present latest energy prediction made by the AI model

If you in the main page click on the predicted energy value, or select energy from the drop-down menu, the predicted energy consumption page will be displayed.

This page displays the predicted energy consumption for the building over the next seven days and the time the prediction was run.

In the following pages the following screenshots are shown:

- Figure 9.21 is a screenshot of the predicted energy consumption page on a physical Samsung Galaxy S21 Ultra.
- Figure 9.22 is a screenshot of the predicted energy consumption page displayed in WinUI running on a laptop.
- Figure 9.23 is a screenshot of the predicted energy consumption page of application on an iPhone 16 simulator running on a laptop.

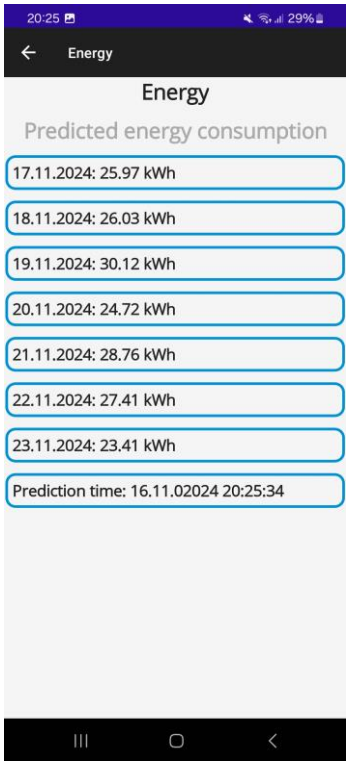


Figure 9.21: Screenshot of the predicted energy consumption page on a physical Samsung Galaxy S21 Ultra

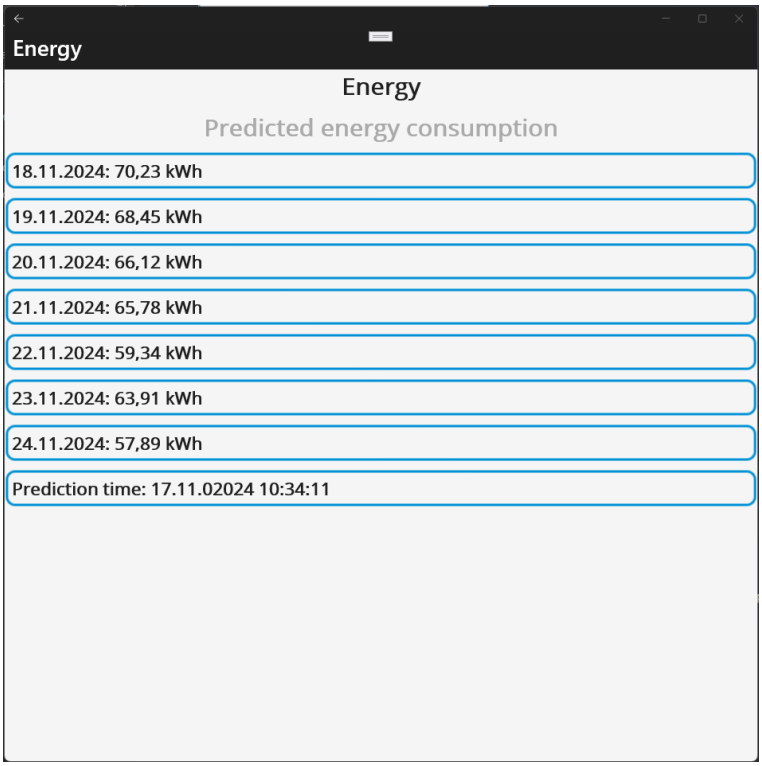


Figure 9.22: Screenshot of the predicted energy consumption page displayed in WinUI running on a laptop

Figure 9.23: Screenshot of the predicted energy consumption page of application on an iPhone 16 simulator running on a laptop

10 Discussion and conclusion of mobile application development project

The target for the project was to do an AI-driven development of open-source, cross-platform mobile application for sensor data monitoring av analysis as described in the project description [1].

The project focused on an application for energy prediction of a building using existing sensors in the building and weather forecasts from online sources. In addition to the prediction of energy consumption, actual data are displayed in the application. There is also the possibility to show historical data in graphs. The control system of the building and the storing of data from the building to the database has not been part of the project.

The first version of the application was successfully created using Visual Studio and the .NET MAUI framework. The resulting application and its GUI is described in this report. In addition to the application, an Azure SQL database with a web API has been created, and a ChatGPT 3.5 AI prediction model has also been made available using Azure. The application is also using the MET weather API's *locationforecast* [41] to collect weather forecast information. The development of these services and the development of the mobile application is described in this report.

At the end of this project, we have a functioning cross-platform that is tested in emulators, for Android, iOS, and WinUI, and on a physical Android device.

The development of the application is not complete and further development is needed. Our suggestion for further development is focused on the following areas.

In the first version the mobile application contains both the GUI layer and application layer. A split of these is suggested in the report.

The user interface could be improved both graphically and on usability of the application. Things like adapting better to different screen sizes is one suggestion from the project team.

The possibility to give the user the ability to customize the information displayed could give a better user experience.

The current application is developed focused on one building. The mobile application and its services could be improved for better scalability so that it quickly can be adapted to different buildings by simple configuration. Also, the infrastructure of the system should be studied with focus to improve scalability.

The connection between the buildings control system and the Azure SQL database was not part of this project and should be further developed.

The application as currently developed relies on cloud services. There should be studied if any offline functionality is needed.

11 References

[1] S. Mylvaganam, "Project Description," University of South-Eastern Norway, Porsgrunn, 2024.

[2] The Norwegian Meterological Institute, "Welcome to the MET Weather API," The Norwegian Meterological Institute, [Online]. Available: https://api.met.no/. [Accessed 06 10 2024].

[3] OpenAI, "Introducing ChatGPT," Open AI, [Online]. Available: https://openai.com/index/chatgpt/. [Accessed 03 11 2024].

[4] Backlinko, "iPhone vs. Android User & Revenue Statistics (2024)," Backlinko, [Online]. Available: https://backlinko.com/iphone-vs-android-statistics. [Accessed 29 09 2024].

[5] Android Developers, "Meet Android Studio," Android Developers, [Online]. Available: https://developer.android.com/studio/intro. [Accessed 11 11 2024].

[6] E. K. Ekren, "What Is Xcode and How to Use It?," Netguru, 25 05 2024. [Online]. Available: https://www.netguru.com/blog/what-is-xcode-and-how-to-use-it. [Accessed 11 11 2024].

[7] R. Sheldon, "What is the iOS software development kit (iOS SDK)?," TechTarget, [Online]. Available: https://www.techtarget.com/searchmobilecomputing/definition/iOS-developer-kit. [Accessed 11 11 2024].

[8] Reactive Native, "React Native · Learn once, write anywhere," Reactive Native, [Online]. Available: https://reactnative.dev/. [Accessed 29 09 2024].

[9] Microsoft, ".NET Multi-platform App UI (.NET MAUI) | .NET," Microsoft, [Online]. Available: https://dotnet.microsoft.com/en-us/apps/maui. [Accessed 29 09 2024].

[10] Stack Overflow, "Why Flutter is the most popular cross-platform mobile SDK," Stack Overflow, [Online]. Available: https://stackoverflow.blog/2022/02/21/why-flutter-is-the-most-popular-cross-platform-mobile-sdk/. [Accessed 29 09 2024].

[11] Solar 2D, "Solar2D Documentation — Developer Guides | Getting Started," Solar 2D, [Online]. Available: https://docs.coronalabs.com/guide/programming/intro/index.html#whatis. [Accessed 29 09 2024].

[12] Wikipedia, "Autoregressive integrated moving average," Wikipedia, 27 05 2024. [Online]. Available:

https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average. [Accessed 14 09 2024].

[13] J. Brownlee, "A Gentle Introduction to Exponential Smoothing for Time Series Forecasting in Python," machinelearningmastery.com, 12 04 2020. [Online]. Available: https://machinelearningmastery.com/exponential-smoothing-for-time-series-forecasting-in-python/. [Accessed 14 09 2024].

[14] W. Yu and J. Gonzalez, "Non-linear system modeling using LSTM neural networks," sciencedirect.com, 31 08 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896318310814. [Accessed 14 09 2024].

[15] M. Rahman, "Different ways to combine CNN and LSTM networks for time series classification tasks," medium.com, 04 12 2022. [Online]. Available: https://medium.com/@mijanr/different-ways-to-combine-cnn-and-lstm-networks-for-time-series-classification-tasks-b03fc37e91b6. [Accessed 14 09 2024].

[16] GPT-4o, "ChatGPT," OpenAI, [Online]. Available: https://chatgpt.com. [Accessed 28 09 2024].

[17] Copilot, "GitHub Copilot," GitHub, [Online]. Available: https://github.com/features/copilot. [Accessed 28 09 2024].

[18] A. Forecast, "Aws Amazon," Amazon, [Online]. Available: https://aws.amazon.com/forecast/. [Accessed 28 09 2024].

[19] AutoML, "Google Cloud," Google, [Online]. Available: https://cloud.google.com/automl. [Accessed 28 09 2024].

[20] A. AI, "Azure AI Studio," Microsoft, [Online]. Available: https://portal.azure.com/#browse/Microsoft.MachineLearningServices%2Faistudio. [Accessed 28 09 2024].

[21] V. Kaushik, "Best AI Tools for Programmers," medium.com, 16 04 2024. [Online]. Available: https://medium.com/@kaushikvikas/various-ai-tools-for-programmers-an-in-depth-analysis-e4ddc1cde88d. [Accessed 17 11 2024].

[22] GitHub, "GitHub Copilot - Your AI pair programmer," GitHub, 2024. [Online]. Available: https://github.com/features/copilot. [Accessed 17 11 2024].

[23] Insight7, "How to Use AI to Write Reports: A Guide," Insight7, 16 07 2024. [Online]. Available: https://insight7.io/how-to-use-ai-to-write-reports-a-guide/. [Accessed 17 11 2024].

[24] V. Angelova, "Top 7 Best Free Weather Forecast APIs to Access Global Weather Data in 2024," Weatherstack, 23 06 2024. [Online]. Available:

https://blog.weatherstack.com/blog/top-7-best-free-weather-forecast-apis-to-access-global-weather-data/. [Accessed 06 10 2024].

[25] Weatherstack, "Weatherstack - Real-Time World Weather REST API," Weatherstack, [Online]. Available: https://weatherstack.com/. [Accessed 06 10 2024].

[26] Weatherstack, "Weatherstack API Pricing Plans," Weatherstack, [Online]. Available: https://weatherstack.com/product. [Accessed 06 10 2024].

[27] OpenWeather, "Current weather and forecast - OpenWeatherMap," OpenWeather, [Online]. Available: https://openweathermap.org/. [Accessed 06 10 2024].

[28] OpenWeather, "Pricing - OpenWeather," OpenWeather, [Online]. Available: https://openweather.co.uk/pricing-corp. [Accessed 06 10 2024].

[29] Weatherbit, "Free Weather API | Weatherbit," Weatherbit, [Online]. Available: https://www.weatherbit.io/. [Accessed 06 10 2024].

[30] Weatherbit, "Weatherbit API - Pricing," Weatherbit, [Online]. Available: https://www.weatherbit.io/pricing. [Accessed 06 10 2024].

[31] AccuWeather, "Local, National, & Global Daily Weather Forecast | AccuWeather," AccuWeather, [Online]. Available: https://www.accuweather.com/. [Accessed 06 10 2024].

[32] AccuWeather, "AccuWeather APIs | Packages," AccuWeather, [Online]. Available: https://developer.accuweather.com/packages. [Accessed 06 10 2024].

[33] Tomorrow.io, "The World’s Weather Intelligence Platform," Tomorrow.io, [Online]. Available: https://www.tomorrow.io/. [Accessed 06 10 2024].

[34] Tomorrow.io, "Weather API," Tomorrow.io, [Online]. Available: https://www.tomorrow.io/weather-api/. [Accessed 06 10 2024].

[35] Visual Crossing, "Weather Data & Weather API | Visual Crossing," Visual Crossing, [Online]. Available: https://www.visualcrossing.com/. [Accessed 06 10 2024].

[36] Visual Crossing, "Weather Data & Weather API Pricing | Visual Crossing," Visual Crossing, [Online]. Available: https://www.visualcrossing.com/weather-data-editions. [Accessed 06 10 2024].

[37] The Weather Company, "The World’s Leading Weather Provider," The Weather Company, [Online]. Available: https://www.weathercompany.com/. [Accessed 06 10 2024].

[38] Standard Norge, "SN-NSPEK 3031:2023," Standard Norge, 07 07 2023. [Online]. Available: https://lese.standard.no/product/2548419?langUI=nb&filePath=76d9c61c-bcea-432f-ba2d-a22969f627a9.pdf&fileType=Pdf. [Accessed 29 09 2024].

[39] World Meteorological Organization, "Guide to Instruments and Methods of Observation (WMO-No. 8) | World Meteorological Organization," World Meteorological Organization, [Online]. Available: https://community.wmo.int/en/activity-areas/imop/wmo-no_8. [Accessed 06 10 2024].

[40] The Norwegian Meteorological Institute, "met.no/en," The Norwegian Meteorological Institute, [Online]. Available: https://www.met.no/en. [Accessed 06 10 2024].

[41] The Norwegian Meteorological Institute, "Locationforecast," The Norwegian Meteorological Institute, [Online]. Available: https://api.met.no/weatherapi/locationforecast/2.0/documentation. [Accessed 06 10 2024].

[42] The Norwegian Meteorological Institute, "THREDDS dataset archive landing page," The Norwegian Meteorological Institute, [Online]. Available: https://api.met.no/product/THREDDS. [Accessed 06 10 2024].

[43] Direktoratet for byggkvalitet, "Byggteknisk forskrift (TEK17) med veiledning," Direktoratet for byggkvalitet, 29 09 2024. [Online]. Available: https://www.dibk.no/regelverk/byggteknisk-forskrift-tek17.

[44] Norwegian Building Authority, "Regulations on technical requirements for construction works," 07 2017. [Online]. Available: https://www.dibk.no/globalassets/byggeregler/regulation-on-technical-requirements-for-construction-works--technical-regulations.pdf. [Accessed 29 09 2024].

[45] Direktoratet for byggkvalitet, "Innledning til kapittel 13 Inneklima og helse," Direktoratet for byggkvalitet, 29 09 2024. [Online]. Available: https://www.dibk.no/regelverk/byggteknisk-forskrift-tek17/13/i/innledning.

[46] Direktoratet for byggkvalitet, "Innledning til kapittel 14 Energi," Direktoratet for byggkvalitet, 29 09 2024. [Online]. Available: https://www.dibk.no/regelverk/byggteknisk-forskrift-tek17/14/innledning-til-kapittel-14-energi.

[47] Direktoratet for byggkvalitet, "§ 14-2. Krav til energieffektivitet," Direktoratet for byggkvalitet, [Online]. Available: https://www.dibk.no/regelverk/byggteknisk-forskrift-tek17/14/14-2. [Accessed 29 09 2024].

[48] Direktoratet for byggkvalitet, "§ 14-3. Minimumsnivå for energieffektivitet," Direktoratet for byggkvalitet, [Online]. Available:

https://www.dibk.no/regelverk/byggteknisk-forskrift-tek17/14/14-3. [Accessed 29 09 2024].

[49] Standard Norge, "NS 3031:2014," Standard Norge, [Online]. Available: https://online.standard.no/nb/ns-3031-2014. [Accessed 29 09 2024].

[50] Direktoratet for byggkvalitet, "§ 14-4. Krav til løsninger for energiforsyning," Direktoratet for byggkvalitet, [Online]. Available: https://www.dibk.no/regelverk/byggteknisk-forskrift-tek17/14/14-4. [Accessed 29 09 2024].

[51] Standard Norge, "NS 3031 Beregning av bygningers energiytelse er trukket tilbake, men vises fortsatt til i byggteknisk forskrift," Standard Norge, 29 09 2021. [Online]. Available: https://standard.no/fagomrader/energi-og-klima-i-bygg/bygningsenergi/ns-3031-beregning-av-bygningers-energiytelse-er-trukket-tilbake-men-vises-fortsatt-til-i-byggteknisk-forskrift/. [Accessed 29 09 2024].

[52] Standard Norge, "NS-EN ISO 52000-1:2017," 01 10 2017. [Online]. Available: https://lese.standard.no/product/2520509?langUI=en&filePath=35443749-c0b8-4990-b452-6f2ec6ed2c1c.pdf&fileType=Pdf. [Accessed 29 09 2024].

[53] Standard Norge, "NS-EN ISO 15927-4:2005," 11 2005. [Online]. Available: https://lese.standard.no/product/2523705/en?langUI=nb. [Accessed 29 09 2024].

[54] Standard Norge, "NS-EN 16798-1:2019," 09 04 2024. [Online]. Available: https://lese.standard.no/product/2535947/en?langUI=nb. [Accessed 29 09 2024].

[55] Standard Norge, "NS-EN 15192-1:2017+A1," 13 07 2021. [Online]. Available: https://lese.standard.no/product/2545777?langUI=nb&filePath=3f0b1b9f-58e1-4720-a7a3-9177a65b760e.pdf&fileType=Pdf. [Accessed 29 09 2024].

[56] WilliamDAssafMSFT, "Create a single database - Azure SQL Database," Microsoft, 18 09 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-sql/database/single-database-create-quickstart?view=azuresql. [Accessed 10 11 2024].

[57] Microsoft, "Tutorial: Create a web API with ASP.NET Core," Microsoft, 23 08 2024. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-8.0. [Accessed 13 10 2024].

[58] Microsoft, "Connect to and query Azure SQL Database using .NET and the Microsoft.Data.SqlClient library - Azure SQL Database," Microsoft, 17 09 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-sql/database/azure-sql-dotnet-quickstart?view=azuresql. [Accessed 13 10 2024].

[59] Swagger, "SwaggerHub | API Design & Documentation Tool," [Online]. Available: https://swagger.io/tools/swaggerhub/?utm_source=aw&utm_medium=ppcg&utm_campaign=SEM_SwaggerHub_PR_EMEA_ENG_EXT_Prospecting_Tier2&utm_term

=swagger&utm_content=665457100535&gad_source=1&gclid=Cj0KCQiA0MG5BhD1ARIsAEcZtwRuzNFZ6YeKVKXvNpCfCza8u1VzJOV3O5gKawI9A. [Accessed 10 11 2024].

[60] SamMonoRT, "Migrations Overview - EF Core," Microsoft, 02 01 2023. [Online]. Available: https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/. [Accessed 10 11 2024].

[61] SamMonoRT, "Applying Migrations - EF Core," Microsoft, 18 02 2023. [Online]. Available: https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/applying. [Accessed 10 11 2024].

[62] Microsoft, "Quickstart: Deploy an ASP.NET web app - Azure App Service | Microsoft Learn," Microsoft, [Online]. Available: https://learn.microsoft.com/en-us/azure/app-service/quickstart-dotnetcore?tabs=net80&pivots=development-environment-vs. [Accessed 10 11 2024].

[63] Postman, "Postman API Platform," Postman, [Online]. Available: https://www.postman.com/. [Accessed 10 11 2024].

[64] N.-O. Skeie, Lecture notes for object-oriented analysis, design, and programming using UML and C#, Porsgrunn: University of South-Eastern Norway, 2023.

[65] Stack Overflow, "2024 Stack Overflow Developer Survey," Stack Overflow, [Online]. Available: https://survey.stackoverflow.co/2024/technology#most-popular-technologies-language. [Accessed 29 09 2024].

[66] Apple, "Apple Developer Documentation," Apple, [Online]. Available: https://developer.apple.com/documentation/xcode. [Accessed 29 09 2024].

[67] BrowserStack, "What is Xcode: Features, Installation, Uses, Advantages and Limitations," BrowserStack, [Online]. Available: https://browserstack.wpengine.com/guide/what-is-xcode/. [Accessed 29 09 2024].

[68] REactive Native, "Set Up Your Environment · React Native," Reactive Native, [Online]. Available: https://reactnative.dev/docs/set-up-your-environment. [Accessed 29 09 2024].

[69] AltexSoft, "What is Xamarin? Xamarin vs Native App Development," AltexSoft, 21 12 2022. [Online]. Available: https://www.altexsoft.com/blog/pros-and-cons-of-xamarin-vs-native/. [Accessed 29 09 2024].

[70] Softude, "Xamarin Cross-Platform Development: Fundamentals & Best Practices," Softude, [Online]. Available: https://www.softude.com/blog/xamarin-the-ultimate-cross-platform-mobile-app-development. [Accessed 29 09 2024].

[71] Flat Rock Technology, "From Xamarin to MAUI, What has changed?," Flat Rock Technology, [Online]. Available: https://flatrocktech.com/blog/xamarin-forms-maui-migration. [Accessed 29 09 2024].

[72] GitHub, "Releases · coronalabs/corona," GitHub, [Online]. Available: https://github.com/coronalabs/corona/releases. [Accessed 29 09 2024].

[73] M. Bellinaso, "Flutter: the good, the bad and the ugly," ASOS Tech Blog, 25 07 2020. [Online]. Available: https://medium.com/asos-techblog/flutter-vs-react-native-for-ios-android-app-development-c41b4e038db9. [Accessed 29 09 2024].

[74] BeeWare, "Write once. Deploy everywhere.," BeeWare, [Online]. Available: https://beeware.org/. [Accessed 29 09 2024].

[75] Kivy, "Installing Kivy," Kivy, [Online]. Available: https://kivy.org/doc/stable/gettingstarted/installation.html. [Accessed 29 09 2024].

[76] Ionic, "Open-Source UI Toolkit to Create Your Own Mobile Apps," Ionic, [Online]. Available: https://ionicframework.com/docs. [Accessed 29 09 2024].

Appendices

Appendix A Project description

Appendix B Gantt-diagram for project

Appendix C GitHub repository

Appendix A Project description

University of South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FM4017 Project

Title: AI-Driven Development of Open-Source, Cross-Platform Mobile Apps for Sensor Data Monitoring and Analysis

USN supervisor: Hans-Petter Halvorsen, Saba Mylvaganam

External partners: Altibox, Dimension Four

Task background:

The pivotal role of mobile phones and apps in modern technology: In today's world, mobile phones and apps are at the center of technological advancements. The two dominant platforms for smartphones are Apple's iOS and Google's Android. Numerous tools are available for developing mobile applications, but one of the most transformative is Artificial Intelligence (AI). How can AI be utilized and integrated into the development of contemporary mobile applications?

AI can enhance mobile app development in several ways:

- Personalization: AI algorithms can analyze user behavior to provide personalized experiences, making apps more engaging and user-friendly.
- Automation: AI can automate repetitive tasks, improving efficiency and reducing development time.
- Predictive Analytics: By analyzing data, AI can predict user needs and preferences, allowing developers to create more intuitive and responsive apps.
- Enhanced Security: AI can detect and respond to security threats in real-time, ensuring user data is protected.
- Natural Language Processing (NLP): Integrating NLP allows for advanced features like voice recognition and chatbots, enhancing user interaction.

By leveraging AI, developers can create smarter, more efficient, and highly personalized mobile applications that meet the evolving needs of users.

Task description:

Main tasks planned for this project are:

- Perform a study on tools for app development and discuss pros and cons of the different tools. Present an overview and select one tool to be used in this project.
- Check open-source materials on AI tools suitable for developing predictive models. Discuss pros and cons of selected different AI tools. Based on your research, select one open-source AI tool for your project.

- Perform thorough research on online sources for weather forecasting including available weather data and discuss pros and cons. Select one source to use in project.
- Select which parameters are normally used for indoor environment and weather to do energy calculations of buildings, based on your research. Discuss the use of the selected parameters.
- Select a building where we can get historical and current data for indoor environment, energy consumption, and weather data (possibly one building at USN campus). Data will be stored in a cloud database for use in project. Updating data from building control system to cloud database will not be part of project.
- Use an open-source AI to make a prediction of energy consumption based on historical data and weather forecast for a selected period.
- Develop a mobile app that will display current and historical data for indoor environment, energy consumption, and weather data. The selected App should also display a forecast of future energy consumption.
- The HW, SW and HW/SW integration for your system should be properly documented using the USN guidelines in the form of a technical report, with software documentation on GitHub and possibly presentation using YouTube.

Student category: ITA - suitable for both for Campus students and Online students

The project is suitable for students not present at the campus (e.g., online students): Yes

Practical arrangements: None

Signatures:
Supervisors:

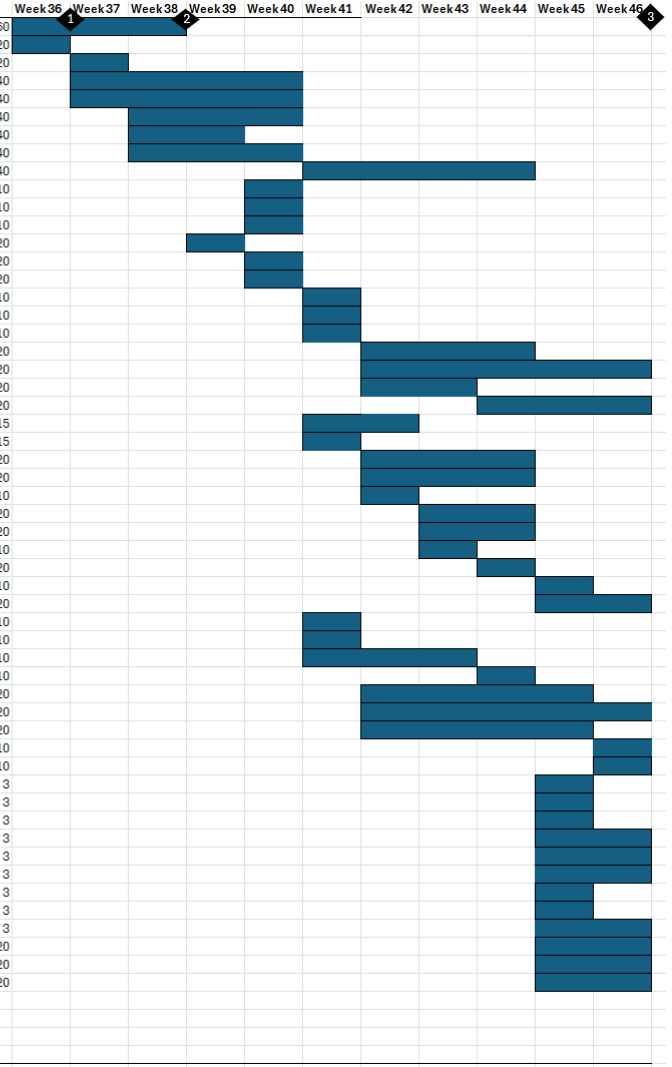
2024.09.29
Hans-Petter Halvorsen
Hans-Petter Halvorsen

Students:

2024.09.29	2024.09.29	2024.09.29
Ørjan Ingebrigtsen	Kim Langvannskås	Aleksander Wad Holthe

Appendix B Gantt-diagram for project

Tasks	Resource	Progress	Hours
Defining and signing the final (adjusted) project topic description	Everyone	100 %	60
Meet supervisors and intial planning	Everyone	100 %	20
Make introduction chapter of project report	Ørjan	100 %	20
Study of app development frameworks	Aleksander	100 %	40
Study of AI for prediction model development	Kim	100 %	40
Research on online sources for weather data and forecasts	Ørjan	100 %	40
Study of energy simulation of buildings and parameters used for environmental data	Ørjan	100 %	40
Select dataset to be used in development of app	Kim	100 %	40
Extract data to build dataset in database	Aleksander	100 %	40
Select development tool/framework for app development	Aleksander	100 %	10
Select AI for prediction model development	Kim	100 %	10
Select cloud database to be used in project	Ørjan	100 %	10
Iteration #1 elaboration phase: Collecting requirements using FURPS+	Ørjan	100 %	20
Iteration #1 elaboration phase: Create use case diagram	Kim	100 %	20
Iteration #1 elaboration phase: Create domain model	Aleksander	100 %	20
Create use case document: View live data	Aleksander	100 %	10
System sequence diagram: View live data	Aleksander	100 %	10
Development use case: View live data	Aleksander	100 %	10
Create use case document: View historical data	Aleksander	100 %	20
System sequence diagram: View historical data	Aleksander	100 %	20
Development use case: View historical data	Aleksander	100 %	20
Improve graphical use interface with dashboard and better layout	Aleksander	100 %	20
Documentation: Web API for Azure database	Ørjan	100 %	15
Development use case: Web API for Azure database	Ørjan	100 %	15
Create use case document: Get database data in application	Ørjan	100 %	20
Create sequence diagram: Get database data in application	Ørjan	100 %	20
Development use case: Get database data in application	Ørjan	100 %	10
Create use case document: Post database data from application	Ørjan	100 %	20
Create sequence diagram: Post database data from application	Ørjan	100 %	20
Development use case: Post database data from application	Ørjan	100 %	10
Extend use cases for get and post to database to include weather data and energy prediction	Ørjan	100 %	20
Bug fix to allow Android emulator to use 10.0.2.2 as localhost	Ørjan	100 %	10
Prepare Azure SQL database and Azure API	Ørjan	100 %	20
Create use case document: Get weather data	Kim	100 %	10
Create sequence diagram: Get weather data	Kim	100 %	10
Development use case: Get weather data	Kim	100 %	10
Make table for weather data received from weather API	Kim	100 %	10
Create use case document: AI prediction model API	Kim	100 %	20
Create sequence diagram: AI prediction model API	Kim	100 %	20
Development use case: AI prediction model API	Kim	100 %	20
Fix bug with API key for AI prediction model in Android app	Kim	100 %	10
Include API base adress for production in energy prediction system	Ørjan	100 %	10
Finish report summary	Ørjan	100 %	3
Finish report preface	Ørjan	100 %	3
Finish report chapter 1 introduction	Ørjan	100 %	3
Finish report chapter 2 Mobile application development tools study and selection	Aleksander	100 %	3
Finish report chapter 3 Study of AI tools for prediction and forecasting, and selection of tool	Kim	100 %	3
Finish report chapter 4 Online sources for weather forecasting data	Kim	100 %	3
Finish report chapter 5 Parameters used for energy simulations in buildings	Ørjan	100 %	3
Finish report chapter 6 Azure SQL cloud database with web API	Ørjan	100 %	3
Finish report chapter 7 AI prediction model using ChatGPT 3.5 and Azure web API	Kim	100 %	3
Finish report chapter 8 Development of the mobile application	Everyone	100 %	20
Finish report chapter 9 Results	Everyone	100 %	20
Finish report chapter 10 Discussion	Ørjan	100 %	20
Milestones:			
1. Project Kick-off meeting September 5th (Week 36)			
2. Deadline signing final Project Description September 22nd (Week 38)			
3. Deadline for submitting report November 17th (Week 46)			
Project status		100 %	927



Appendix C GitHub repository

The GitHub repositories can be found at:

https://github.com/FM4017-Project-2024