NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
"KYIV POLYTECHNIC INSTITUTE
named after Igor Sikorsky"

Faculty of Informatics and Computer Science

Department of Information Systems and Technologies

Admitted to the defense:

Head of the Department

_____ Oleksandr ROLIK

"___"_____2025 p.

Diploma project(TRANSLATED)

for a bachelor's degree
in the educational and professional program "Information support of robotic systems"
specialty 126 "Information systems and technologies"
On the topic: "Robotic Air Environment Monitoring System with Analysis and Alerting Function"

Completed by:
fourth year student, group IK-11
Roman Babukh _____

Leader:
Assistant of the Department of ICT
Dragan Mykhailo Serhiyovych _____

Reviewer:
Associate Professor of the Department of IPI, Candidate of Technical Sciences, Associate Professor
Lishchuk Kateryna Ihorivna _____

I hereby certify that this thesis project does not contain any borrowings from the works of other authors without appropriate references.
Student _____

Kyiv - 2025

National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute
Faculty of Informatics and Computer Science
Department of Information Systems and Technologies

Level of higher education - first (bachelor's)

Specialty - 126 "Information systems and technologies"

Educational and professional program "Information support of robotic systems"

I APPROVE
Head of the Department
_____ Oleksandr ROLIK
"___"_____2025 p.

TASKS
for a student's graduation project
To Roman Yuriyovych Babukh

1. Project topic "Robotic system for monitoring air environment parameters with the function of analysis and warning", project manager Dragan Mykhailo Sergiyovych, approved by the order of the university dated May 23, 2025 № 1705-s

2. Deadline for student project submission: 09.06.2025.

3. Baseline data for the project: The system should provide real-time collection, storage, and analysis of air quality data, forecasting of future indicators based on the data obtained, and display of information in the form of graphs via a web interface.

4. Contents of the explanatory note: list of abbreviations, introduction, overview of the subject area, analysis of existing solutions, formulation of system requirements, selection of technologies, system development, system testing.

5. List of graphic material: four drawings, a functional diagram, a precedent diagram, a business process diagram, a data flow diagram.

6. Date of issue of the assignment: 25.02.2025.

Calendar plan

No. s/n	The name of the stages of the of the diploma project	The term of execution of project stages	Note
	Approval of the topic and terms of reference	08.03.2025	
	Description of the subject area	18.04.2025	
	Analysis of existing solutions	25.04.2025	
	Formation of system requirements	02.05.2025	
	Software implementation of the system	09.05.2025	
	Integration of machine learning algorithms	16.05.2025	
	Testing the system	28.06.2025	
	Formation of an explanatory note	09.06.2025	

Student Roman Babukh

The head of Mykhailo Dragan

SUMMARY

Robotic Air Environment Monitoring System with Analysis and Alerting Function

The project contains 74 pages, 27 figures, 1 table, links to 32 sources, annexes and 4 design documents.

MONITORING, AIR QUALITY, IoT, ARTIFICIAL INTELLIGENCE, FORECASTING, WEB INTERFACE, MACHINE LEARNING

The object of development is an automated real-time air quality monitoring system.

The purpose of the development is to improve the environmental situation in urban areas, the agricultural sector or for individual use, ensuring timely detection of pollution and maintaining environmental awareness of the population.

The thesis developed a complete architecture of a software monitoring system. The system consists of data collection and processing modules, a machine learning forecasting module, an anomaly alert module, and a web interface for viewing data in a convenient format. In the course of the work, the subject area and modern solutions in the field of environmental monitoring were analyzed, the logic of the modules was described, and the user interface was tested.

The results of this project can be used to create environmental monitoring systems in urban areas, in the agricultural sector, or for individual use in everyday life.

line number	Format.	Designation	Name	Number of sheets	№ ex.	Note
1			General documentation			
2						
3			Newly developed			
4						
5	A4	IK11.020BAK.006 PZ	Explanatory note	74		
6	A3	IK11.020BAK.006 D1	Robotic system	1		
7			monitoring of parameters			
8			air environment			
9			with analysis and alerting functions.			
10			Functional diagram.			
11	A3	IK11.020BAK.006 D2	Robotic system	1		
12			monitoring of parameters			
13			air environment			
14			with analysis and alerting functions.			
15			Data flow diagram.			
16	A3	IK11.020BAK.006 D3	Robotic system	1		
17			monitoring of parameters			
18			air environment			
19			with analysis and alerting functions.			
20			Diagram of business processes.			
21	A3	IK11.020BAK.006 D4	Robotic system	1		
22			monitoring of parameters			
23			air environment			
24			with analysis and alerting functions.			
25			Diagram of precedents.			
26						
27						
28						

					IK11.020BAK.006 TP		
See.	Letter	No. doc.	Signature				
Developed		Babukh R.Y.			A robotic system for monitoring air environment parameters with analysis and warning functions. Statement of the diploma project	Lit. T	Ark. 1 / Sheets 1
Checked.		Dragan M.S.					
						Igor Sikorsky Kyiv Polytechnic Institute	
Approved							

Explanatory note
to the diploma project
On the topic: "Robotic system for monitoring air environment parameters with analysis and warning function"

Kyiv - 2025

TABLE OF CONTENTS

LIST OF SYMBOLS AND NOTATIONS

AQI - Air Quality Index

BPMN - Business Process Model and Notation

HTTP - HyperText Transfer Protocol

IoT - Internet of Things

LAQN - London Air Quality Network

MQTT - Message Queuing Telemetry Transport

PM - Particulate Matter

SMS - Short Message Service.

INTRODUCTION

The current environmental situation requires new technical approaches to environmental monitoring. Air pollution remains one of the key challenges in many regions of the world, particularly in the context of urbanization, industrialization and global climate change. Lack of access to up-to-date and localized environmental data significantly complicates effective management decisions at both the state and individual levels. That is why the development of automated systems for collecting, processing and visualizing environmental information is becoming increasingly important.

Today, there are many commercial solutions for air quality monitoring. However, most of them have a closed infrastructure, do not allow for flexible connection of their own sensors, limit the user's choice of data sources, and do not provide the ability to modify or expand analytical modules to meet the needs of a particular application. This situation creates the need to develop open, scalable and modular systems that do not reflect the current situation.

The aim of this work is to develop a software-based air quality monitoring system capable of real-time processing of telemetry data from sensors, analyzing and detecting anomalies, predicting trends based on a trained model, and providing a user-friendly visual interface for interacting with data. The main emphasis is on simplicity, adaptability of the architecture, and the possibility of further expansion of the system for new tasks.

To achieve this goal, the following tasks need to be solved:

a) collect and pre-process telemetry data from real sensors;

b) implement the architecture of message exchange between system modules based on a message broker;

d) implement an alert system in case of detection of abnormal values;

e) create a web interface to display data and forecasts in a user-friendly way.

Key design decisions are based on the principles of distributed systems, including the use of message brokers to ensure scalability and asynchronous interaction between modules. For forecasting, a machine learning model trained on real historical data is used, which allows the system to be adapted to a specific area.

The results of this work can be applied in the field of smart cities, local environmental monitoring, agricultural technologies, as well as for individual use by residents of polluted regions. In addition, the system can be used in the educational process to demonstrate the operation of modern information technologies.

1 DESCRIPTION OF THE SUBJECT AREA

1.1 Analysis of the problem of environmental monitoring

Environmental monitoring is becoming increasingly important in the context of urbanization, industrial development, and climate change. Air quality, in particular, has a direct impact on human health, environmental sustainability, and quality of life. Exposure to air pollutants causes millions of premature deaths each year, making air monitoring an important component of public health policy.

Air pollution has become one of the most pressing environmental issues of our time. Historically, the beginning of the Industrial Revolution was marked by a significant increase in pollutant emissions as a result of the widespread use of fossil fuels. This period saw the proliferation of factories and urban centers, which led to an increase in the concentration of pollutants such as sulfur dioxide, nitrogen oxides, and particulate matter. The famous "Great Smog of London" in 1952, which resulted in thousands of deaths, emphasized the deadly danger of uncontrolled air pollution.

1.2 World situation

For decades, despite technological advances and regulatory measures in some regions, air pollution has remained a global problem. According to the World Health Organization (WHO), in 2019, exposure to atmospheric (outdoor) and household air pollution caused approximately 6.7 million premature deaths worldwide. At the same time, 99% of the world's population lives in areas where air quality exceeds WHO standards, which increases the risk of developing heart disease, stroke, chronic obstructive pulmonary disease, cancer, and pneumonia.

The burden of air pollution falls disproportionately on low- and middle-income countries. For example, in 2023, Bangladesh and Pakistan recorded average PM2.5 concentrations of 79.9 and 73.7 micrograms per cubic meter, respectively, which is about 15 times higher than the WHO recommended level of 5 micrograms. This high level of pollution is attributed to factors such as industrial emissions, vehicle exhaust, and the

burning of biomass for cooking and heating. Figure 1.1 shows a map of global pollution from fossil fuels.

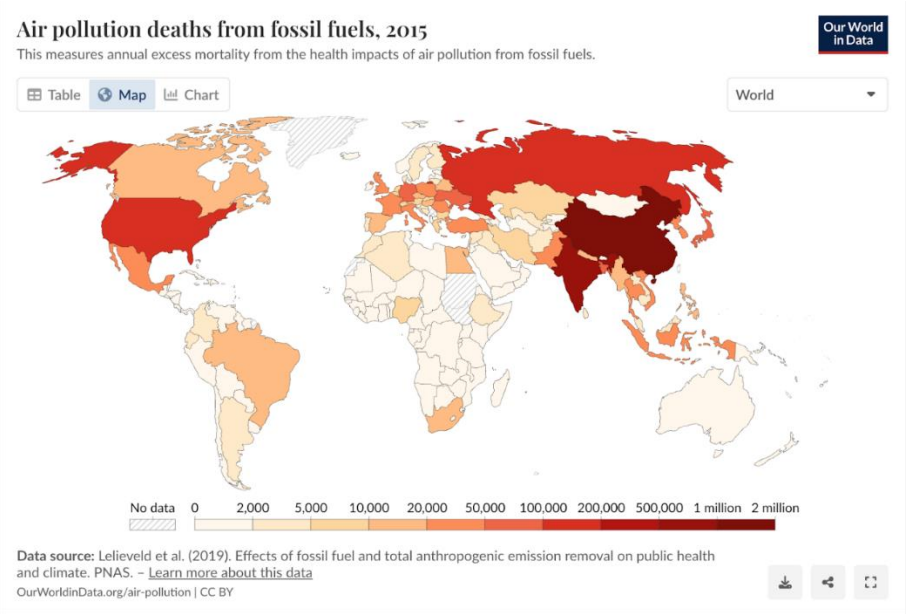

Figure 1.1 - Map of deaths from air pollution caused by the use of fossil fuels, 2015 [1].

In Europe, despite improvements in air quality in many regions due to strict regulations, problems remain. Urban areas continue to struggle with pollution from transportation, industrial activity, and residential heating. The European Environment Agency reports that in 2021, exposure to fine particulate matter led to an estimated 307,000 premature deaths in the European Union.

The health impacts of air pollution are profound. Long-term exposure is associated with respiratory and cardiovascular diseases, adverse pregnancy outcomes, and cognitive impairment. The economic costs are enormous and include health care costs, loss of productivity, and reduced quality of life. Given the scale and severity of the problem, there is an urgent need for reliable air quality monitoring systems. The number of air quality monitoring systems remains unsatisfactory; Figure 1.2 shows a map of settlements with such systems.

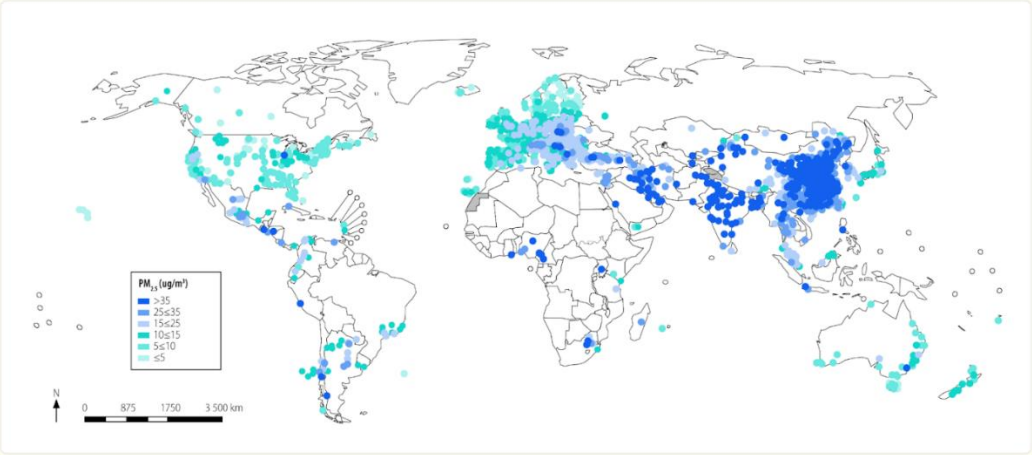

Figure 1.2 - Location of settlements with PM2.5 data, by number of ground-based measurements, 2010-2019 [2].

1.3 The situation in Ukraine

Air pollution is a serious problem for the environment and public health in Ukraine. The country faces challenges related to industrial emissions, outdated infrastructure and insufficient environmental regulations. In urban centers, in particular, there is an increased concentration of harmful pollutants due to heavy traffic, coal-fired power generation and industrial activities.

According to the World Health Organization, in 2019, Ukraine recorded PM2.5 concentrations that exceeded the recommended standards, which negatively affected public health. Cities such as Kyiv, Dnipro, Lviv, and Mariupol have consistently experienced elevated levels of air pollution. In particular, Dnipro and Kryvyi Rih, centers of heavy industry, often suffer from smog and days of poor air quality.

The ongoing war has further complicated the situation, causing infrastructure destruction, forest fires, and industrial accidents that release additional pollutants into the air. Satellite monitoring and ground-based measurements have shown an increase in emissions associated with explosions, fires, and disruption of industrial production in the conflict zones.

Despite these challenges, Ukraine has taken steps to improve environmental monitoring. The country is working to expand its air quality monitoring network, and several government and civic initiatives have emerged to address gaps in government

data. For example, the SaveEcoBot platform, shown in Figure 1.3, aggregates sensor data from across the country, making air quality data available to the public in real time.

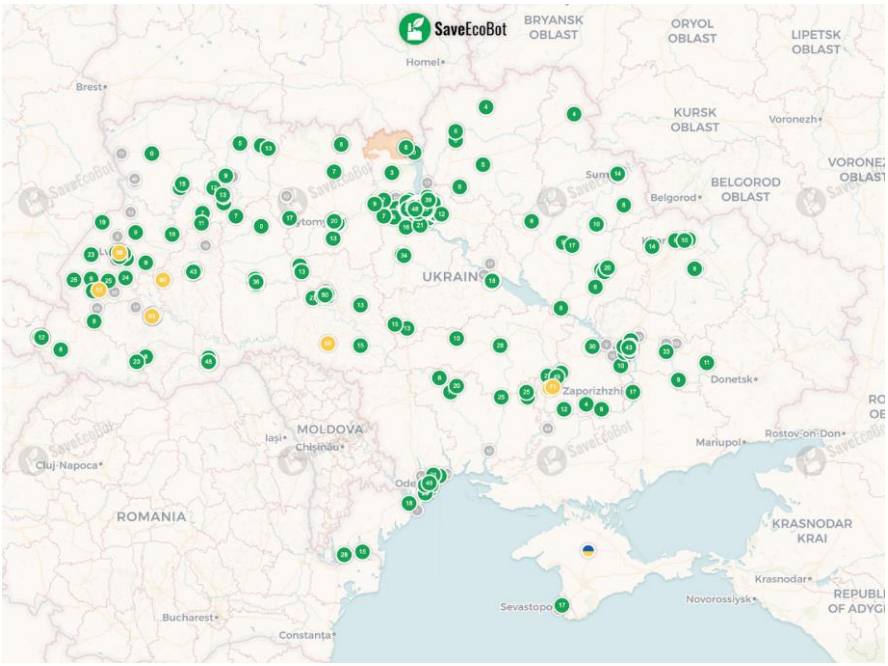

Figure 1.3 - Interactive map of SaveEcoBot [3].

However, systematic policy implementation and modernization of industrial infrastructure are still needed to bring pollution levels within safe limits. The development of scalable, real-time air monitoring systems, especially based on IoT and artificial intelligence technologies, can play an important role in Ukraine's environmental strategy and help raise awareness and action at the local and national levels.

1.4 Current approaches to air quality monitoring

Modern air quality monitoring systems use a combination of hardware and software solutions to collect, transmit and analyze environmental data. These systems range in complexity from high-precision stations operated by the government to compact, low-cost sensor nodes deployed in densely populated urban areas, industrial zones, or even residential neighborhoods. The proliferation of the Internet of Things has enabled a new

generation of air quality monitors that are affordable, scalable, and capable of transmitting and analyzing data in near real time.

1.4.1 Air environment parameters

Air pollution monitoring usually focuses on a set of key environmental parameters that allow to assess the level of pollution, its dynamics and potential impact on public health. One of the main indicators is the concentration of PM2.5 particulate matter, which is a microscopic particle up to 2.5 micrometers in diameter that can penetrate deep into the lungs and cause serious respiratory diseases. Temperature and humidity also play an important role, as they are not direct pollutants, but they have a significant impact on the spread, transformation, and deposition of harmful substances in the air. High temperature, for example, can promote chemical reactions that form secondary pollutants, while humidity can affect the aggregation of particles and change their aerodynamic properties. A comprehensive monitoring approach includes both direct indicators of pollution and the contextual conditions that determine the behavior of these substances in the environment.

1.4.2 Types of sensors and their placement

Air quality monitoring systems typically use two categories of sensors: reference and low-cost. Reference sensors are high-precision instruments used by environmental agencies for regulatory monitoring. These devices, such as beta attenuation monitors for particulate matter, gas chromatographs for volatile compounds, and chemiluminescence analyzers for nitrogen dioxide, provide high data accuracy but are expensive, require regular calibration, and are usually installed at stationary stations.

In contrast, low-cost sensors use metal-oxide semiconductors, electrochemical or optical technologies to detect gases and environmental parameters. Although less accurate and more sensitive to ambient noise, they are affordable and suitable for distributed real-time monitoring. These sensors are usually integrated with

microcontrollers such as Arduino or Raspberry Pi. This makes them ideal for DIY systems, research projects, and citizen science initiatives. Due to their compact size and low power consumption, they can be installed anywhere, whether indoors or outdoors, allowing for scalable air quality monitoring.

1.4.3 Communication and data transfer

Modern environmental monitors based on IoT technologies use different communication means to transmit the collected data, depending on the conditions of the system deployment and the requirements for communication speed and reliability. In indoor environments or in environments with access to infrastructure, Wi-Fi or Ethernet connections are most often used to ensure stable transmission of large amounts of data. In rural areas or in large open spaces where there is no possibility of connecting to fixed networks, energy-efficient LoRaWAN technology is preferred, which allows data to be transmitted over long distances with low power consumption. More sophisticated and mobile systems, especially those requiring wide coverage or autonomy, use cellular communications, in particular 3G, 4G or 5G networks. For real-time data transmission, MQTT, HTTP, or Kafka network protocols are widely used to ensure efficient routing of messages between sensors, analytical platforms, and endpoint notification or storage systems. A flexible combination of transmission channels and protocols allows IoT monitoring systems to be adapted to a wide variety of operating conditions.

1.4.4 Data processing and analysis

Once collected, air quality data typically undergoes a series of processing steps to ensure that it is suitable for further analysis and decision-making. The first step is pre-processing, which includes reducing noise in the data, removing anomalous or damaged values, and normalizing the data to a common scale. If necessary, correction factors are applied to compensate for systematic sensor errors or variable environmental conditions. The next step is data storage and aggregation: time series databases are often used for this

purpose, which are optimized for recording, storing, and quickly accessing large amounts of telemetry information.

The next step is analysis. One of the simplest methods is based on rules: fixed thresholds of parameters trigger automatic alerts. However, machine learning methods are increasingly being used for a more flexible and accurate understanding of the situation. They allow you to detect complex anomalies, predict changes in parameters in the future, and classify types of pollution. Popular algorithms include Random Forest, ARIMA autoregressive models, and recurrent neural networks such as LSTM, which work well with data sequences and allow making predictions based on historical measurements. This approach allows not only to respond promptly to exceeding permissible levels, but also to predict potentially dangerous situations before they actually occur.

1.4.5 Alerts and visualization

Modern environmental monitoring systems include real-time visualization and alerting tools that make sensor data accessible and usable. Visualization is typically done through web dashboards or mobile apps with interactive maps, real-time graphs, and historical data charts.

Alert systems are triggered when values exceed safe thresholds or anomalies are detected. Alerts are sent via push notifications, SMS, or email. This ensures a quick response to hazardous conditions and allows for prompt action to prevent or minimize negative impacts on public health and the environment.

Conclusions to Section 1

Environmental monitoring, including air quality, is a key tool in the fight for a healthy future for society. The problem of air pollution has become a global issue and poses serious risks to human health, the economy and environmental sustainability.

Despite some progress in regulation in some regions, the scale and impact of pollution remains significant, especially in low- and middle-income countries.

In the context of global challenges, Ukraine faces its own set of problems caused by its industrial heritage, military operations, and insufficient development of environmental infrastructure. However, recent years have shown positive developments: public initiatives, the integration of satellite data, and the development of digital platforms demonstrate the readiness of society for change and the high demand for transparency of environmental information.

Technical progress in monitoring deserves special attention. Modern systems use both high-precision reference sensors and affordable, compact sensors, which opens up opportunities for scalable, decentralized data collection. The integration of the Internet of Things, flexible data transmission options, the use of cloud services, analytics, and machine learning allow not only to monitor the current state but also to make forecasts, detect anomalies, and respond quickly to threats.

The subject area demonstrates both the deep complexity of the problem and the availability of innovative approaches to solving it. This creates a solid foundation for the development of new technological solutions aimed at improving the quality of monitoring, raising public awareness, and supporting environmental policy at all levels.

In this context, the development of open and accessible monitoring platforms that can be used by both authorities and individuals is of particular importance. This approach not only helps to increase trust in environmental data, but also encourages public participation in environmental protection. The spread of open-source solutions, mobile applications, and online services makes it possible to engage a wide audience in the collection, analysis, and interpretation of environmental information, which in turn increases environmental awareness and promotes a culture of sustainable development.

2 ANALYSIS OF EXISTING SOLUTIONS

2.1 Overview of existing solutions

The purpose of the existing solutions review is to analyze the air monitoring systems available today. It is necessary to analyze existing air monitoring solutions to understand the current state of technology and practices in this area. The review includes both commercial systems, which are often designed for professional or industrial use, and open source projects, which are typically community-driven or intended for educational and research purposes. By studying these systems, it is necessary to identify best practices, common limitations, and technical gaps that could be addressed in the proposed solution. This analysis allows not only to assess the level of development of the industry, but also to formulate reasonable requirements for your own system.

An important part of the review is devoted to the methods used for data analysis. This includes methods for preprocessing raw data, noise reduction, anomaly detection, trend analysis and forecasting of air quality changes, and other advanced methods. The study shows how existing systems transform sensor readings into meaningful information for technically experienced and non-technical users.

To cover the overall picture of the modern industry, we will divide solutions into categories:

a) city and state;

1) London Air Quality Network;

2) AirNow;

b) commercial;

1) PurpleAir;

2) Kaiterra Data Platform;

c) public and open source;

1) Sensor.Community.

2.2 City and state solutions

2.2.1 London Air Quality Network

The London Air Quality Monitoring Network is one of the largest and most well-known urban air monitoring networks in the world [4]. It was founded in the early 1990s and is managed by a leading research organization in the field of ecology and the environment. The main goal of LAQN is to monitor air pollution in Greater London and surrounding areas, providing high-resolution, real-time data to support public health, environmental research, and policy development initiatives.

LAQN measures a wide range of air quality parameters. These include concentrations of nitrogen dioxide, particulate matter, ozone, carbon monoxide, and sulfur dioxide. Some monitoring stations also measure additional parameters such as volatile organic compounds and meteorological data such as wind speed and temperature. The network consists of more than one hundred monitoring stations strategically placed to collect data from a variety of environments, including roadsides, urban backgrounds, suburbs, and rural areas. This comprehensive deployment allows LAQN to assess both the direct impact of transport emissions and the background air quality in a metropolitan area.

The London Air Quality Network offers a publicly accessible web platform, shown in Figure 2.1, where real-time and historical air quality data can be explored. The website provides interactive maps, graphs, reports, and health tips based on pollution levels. The data is updated frequently, allowing residents, researchers, and policymakers to track air quality trends in near real time.

In addition to providing open access to data, LAQN actively cooperates with academic institutions, environmental organizations, and local authorities to analyze long-term pollution trends and develop effective strategies to reduce pollution. The network is also used to inform emergency services and policy makers during episodes of severe air pollution . Thanks to its reliability, high data resolution and wide coverage, LAQN has become one of the most authoritative sources of air quality information in the UK.

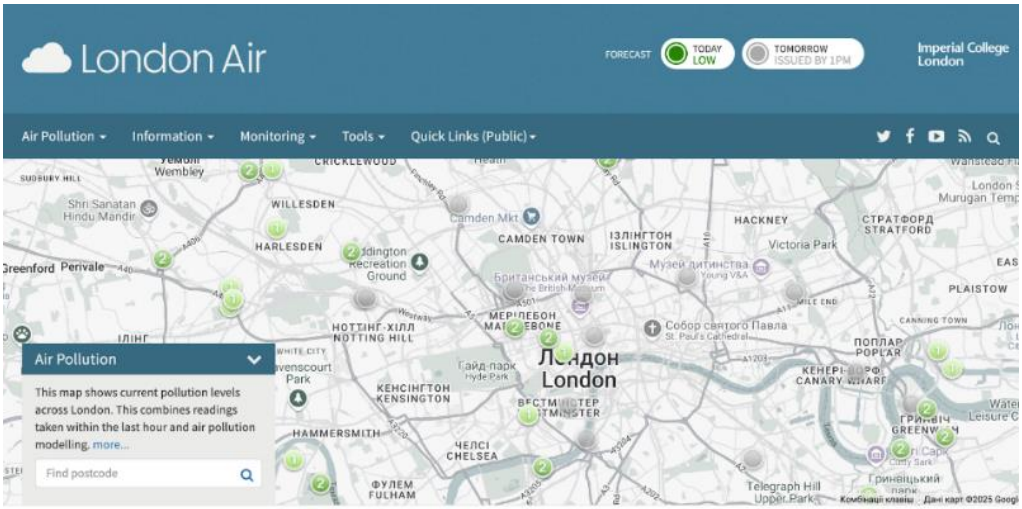

Figure 2.1 - LAQN home page with an interactive map [4].

In addition, the network supports external access to its datasets. Researchers and institutions can request raw or processed data for research or policy analysis. In some cases, standardized data feeds and APIs are available to support integration with external applications and public information services. Such openness promotes transparency, encourages data reuse, and enhances the impact of monitoring on public awareness and environmental management.

The London Air Quality Network also provides air quality forecasts that predict pollution levels for the day ahead. The forecasts are presented in a text format on the website, offering information for different pollutants such as nitrogen dioxide, ozone and particulate matter. The forecasting system is designed to help residents, vulnerable groups and city authorities prepare for periods of poor air quality. Warnings and health advisories are issued when episodes of pollution are expected, allowing proactive measures to be taken, such as adjusting transportation activities or informing vulnerable groups to limit their exposure to the outdoors.

2.2.2 AirNow

AirNow is a national air quality monitoring and information system operated by the U.S. Environmental Protection Agency in partnership with state, local, and tribal air

quality agencies [5]. Launched in 1998, AirNow was designed to provide the public with easy access to real-time air quality information, promote environmental awareness, and support decision-making by both individuals and agencies concerned with air pollution and public health.

AirNow collects, processes, and disseminates air quality data from thousands of monitoring stations located throughout the United States, as well as parts of Canada and Mexico as part of an international collaboration. The system primarily measures the concentration of ground-level ozone, particulate matter, carbon monoxide, sulfur dioxide, and nitrogen dioxide. These pollutants were chosen because they are known to cause significant health impacts and are regulated by the National Ambient Air Quality Standards.

One of the key features of AirNow is the generation and distribution of the Air Quality Index, a standardized indicator that translates the original concentrations of pollutants into easy-to-understand categories ranging from "good" to "dangerous." The AQI helps people quickly assess air pollution risks and make informed decisions about outdoor activities, especially for vulnerable groups such as children, the elderly, and people with respiratory diseases.

AirNow offers users several ways to access air quality data. The main access point is the website, which includes interactive maps, as shown in Figure 2.2, current and predicted AQI values, health tips, and educational resources. In addition, AirNow provides mobile applications that allow users to check local air quality conditions and receive alerts on their smartphones.

The EPA's AirNow mobile app provides a simple interface to quickly check current and forecast air quality information to plan your daily activities and protect your health. The app automatically displays the current air quality index for your area or any area you want to check and allows you to save multiple areas for quick access. The app is available exclusively in the United States.

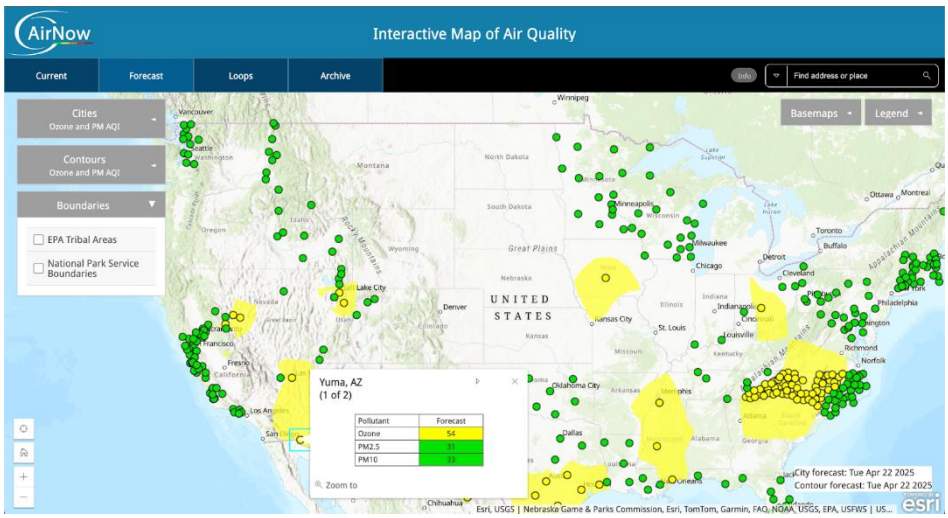

Figure 2.2 - AirNow interactive map [5].

The system supports data exchange with researchers, developers, and partner organizations. Through the AirNow application programming interface and downloadable datasets, external users can access real-time and historical air quality data for further analysis or integration into other applications. This openness promotes research in the fields of ecology, public health, and urban planning and supports the development of innovative solutions to air pollution problems.

AirNow also has a comprehensive air quality forecasting system that provides forecasts of ozone and particulate matter concentrations across the United States. These forecasts typically cover a 24-hour to 48-hour period and are published daily on the AirNow website. The forecast information is integrated into the Air Quality Index system, which helps the public understand expected conditions and plan their activities accordingly. During forest fires or other episodes of extreme pollution, AirNow provides specialized smoke forecasts and health advisories, supporting emergency response efforts and minimizing health risks to the affected population.

2.3 Commercial systems

2.3.1 PurpleAir

PurpleAir is a widely used commercial air quality monitoring solution that has gained significant popularity among individuals, community groups, researchers, and

local governments. Founded in 2015, PurpleAir offers affordable and easy-to-install air quality sensors that provide real-time monitoring of particulate matter concentrations. One of PurpleAir's distinguishing characteristics is its focus on building a dense public air quality data network that allows users to install sensors themselves and add their readings to a global online map.

PurpleAir's approach to data collection and processing emphasizes immediacy and openness. Once installed, the sensors continuously transmit measurements to PurpleAir's cloud servers via Wi-Fi. The data is updated on average every 80 seconds, providing users with near real-time information. Unlike traditional regulatory-grade monitors, PurpleAir sensors are inexpensive and designed for ease of use, which naturally leads to some compromises in terms of absolute accuracy. However, PurpleAir, in collaboration with academic researchers and government agencies, has developed calibration correction factors and algorithms to improve the quality and comparability of its data to official monitoring standards.

PurpleAir offers several methods for accessing and visualizing data. The main platform is the PurpleAir map, an interactive online map shown in Figure 2.3 that displays current data from thousands of sensors around the world. In addition, PurpleAir provides an API that allows developers, researchers, and government agencies to retrieve sensor data programmatically. This feature enables integration with external dashboards, smart city platforms, and custom analytical tools.

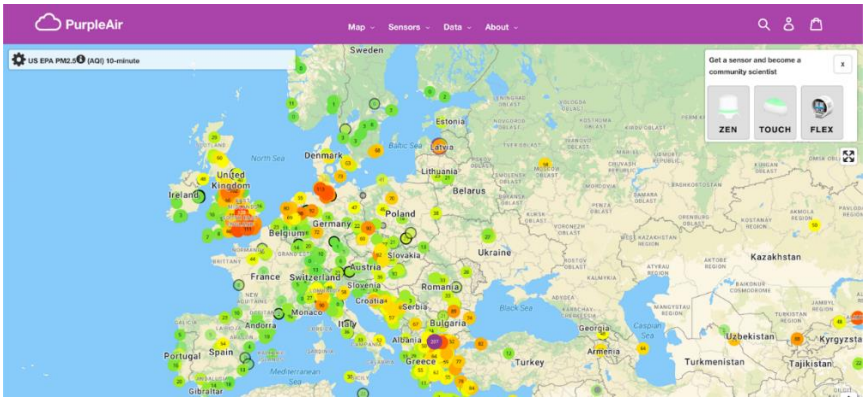

Figure 2.3 - PurpleAir interactive map [6].

In addition to the public visualization platform, PurpleAir offers access to its data through a robust API. Full access to the PurpleAir API is offered through a paid subscription model. This commercial API is designed to meet the needs of researchers, developers, companies, and institutions that require systematic access to large amounts of real-time and historical air quality data.

By offering this service, PurpleAir monetizes data access and also promotes the wider use of its sensor network for innovative applications. Access to the API is provided through the developer panel shown in Figure 2.4.

The API provides important metadata about each sensor, such as:

a) geographical coordinates;

b) information on the installation of the detector;

c) type of detector and firmware version;

d) signal strength and Wi-Fi status;

e) operating status of the sensor and quality indicators.

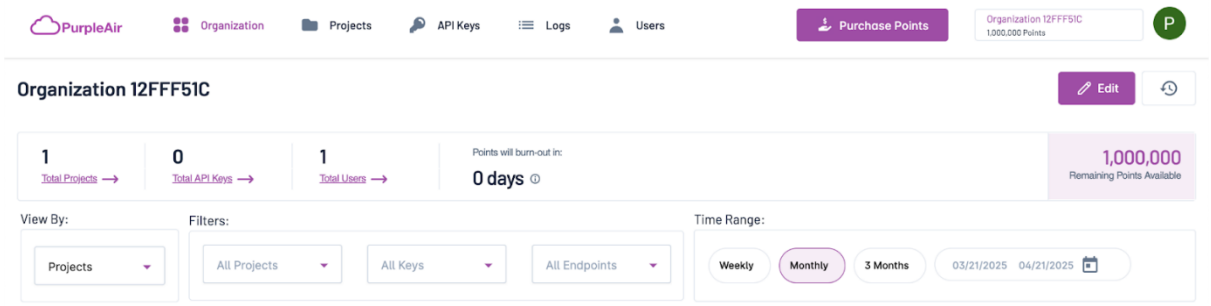

Figure 2.4 - PurpleAir developer panel [7].

Overall, PurpleAir's approach of combining affordable hardware, a participatory global network, and a professional-grade API service makes it unique in the air quality monitoring space. While PurpleAir is not intended to replace regulatory monitoring networks, it is an invaluable additional source of environmental information that increases situational awareness.

2.3.2 Kaiterra Data Platform

Kaiterra is a commercial company specializing in environmental monitoring technologies, with a particular focus on indoor and outdoor air quality [8]. Founded in 2014, Kaiterra has established itself as a provider of high-quality, enterprise-grade air monitoring solutions for businesses, building managers, government agencies, and organizations. While the company sells a variety of hardware devices, such as the Kaiterra Laser Egg and the Kaiterra Sensedge series, the key element that unites their ecosystem is the Kaiterra Data Platform, a centralized platform for data visualization, device management, analytics, and alerts.

Kaiterra Data Platform serves as the primary interface through which users interact with their deployed sensors. It is a cloud-based solution designed to aggregate, display, and analyze real-time and historical data about the environment from multiple devices simultaneously. An example of how it works is shown in Figure 2.4. This platform allows facility managers, researchers, and environmental professionals to monitor air quality in different locations, on different floors, or even in entire building complexes in a centralized, scalable way.

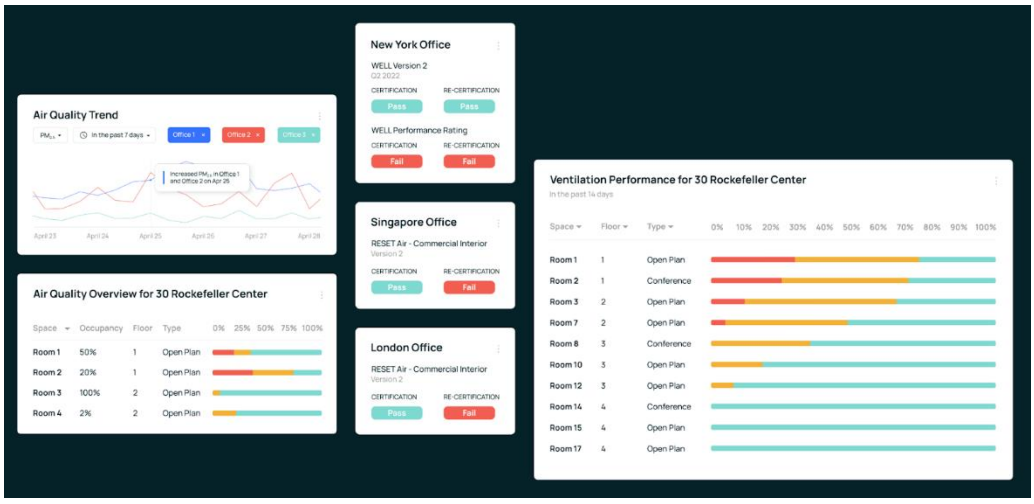

Figure 2.4 - Examples of Kaiterra Dashboard operation [8].

Kaiterra monitoring devices connected to Dashboard typically measure a wide range of environmental parameters. These include particulate matter concentration, carbon dioxide levels, temperature, humidity, and atmospheric pressure. The data is

collected in real time and securely transmitted to Kaiterra's cloud servers via Wi-Fi or Ethernet connection, depending on the device configuration.

The dashboard provides highly customizable data visualization features. Users can view real-time data streams, historical trends for different time periods (hourly, daily, weekly, monthly), and statistical summaries. The data can be visualized in the form of time series graphs, multi-sensor comparison charts, and color-coded air quality indices. The platform allows users to set individual thresholds for various parameters, triggering alerts when certain conditions are reached. Alerts can be sent via email, SMS, or integrated into building management systems via API.

Overall, Kaiterra represents a professional, scalable and flexible approach to air quality monitoring. Its emphasis on high-quality data collection, advanced visualization, forecasting capabilities, system integration and user-friendly interfaces make it an ideal solution for large-scale deployments where data-driven environmental management is critical. As the central hub of the system, the Kaiterra Dashboard exemplifies current best practices in environmental data management and user experience design.

2.4 Public open source systems

Sensor.Community, formerly known as Luftdaten, is a global open source, community-driven initiative for air quality monitoring. air quality [9]. Founded in 2015 in Stuttgart, Germany, the project was initially developed to address the lack of detailed real-time air quality data available to the public. Over time, Sensor.Community has grown to become one of the largest public air monitoring networks in the world, with thousands of sensors deployed across Europe, Asia, America, and other regions.

The fundamental principle of Sensor.Community is to democratize environmental data collection. Instead of relying solely on expensive government monitoring stations, the project empowers people to build their own low-cost air quality sensors and contribute data to a global, open-access platform. The initiative provides detailed, freely available

instructions for assembling monitoring devices using widely available hardware components.

The Sensor.Community focuses primarily on measuring particulate matter, which is a critical indicator of air pollution and is associated with significant health risks. Over time, the project has expanded to include additional measurements such as temperature, humidity, and atmospheric pressure, depending on the sensors that users integrate into their devices. While the main focus of Sensor.Community remains on particulate pollution, its modular design allows participants to experiment with different sensor configurations to monitor a wider range of environmental parameters.

Sensor.Community provides several tools for accessing and visualizing the collected data. The main visualization interface is the interactive global map shown in Figure 2.5, which displays all active sensors and their real-time measurements. Users can view particulate matter, temperature, and humidity readings, depending on the sensor configuration on each device.

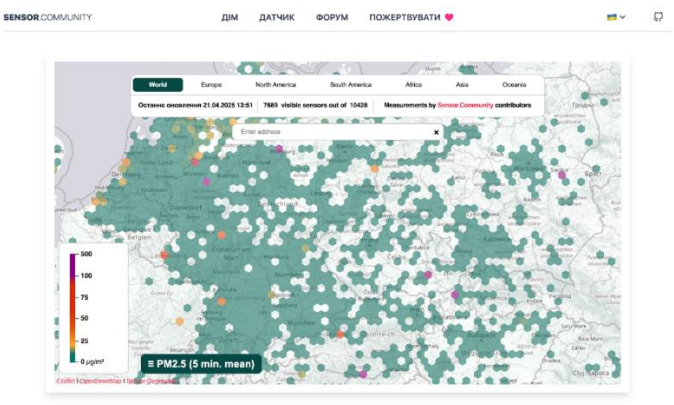

Figure 2.5 - Sensor.Commutiny platform [9].

Since Sensor.Community devices are created by citizens and use low-cost components, the accuracy and calibration of individual sensors may vary. As a result, the network data is considered indicative and not normative. Nevertheless, Sensor.Community data provides valuable information on air quality, pollution hotspots, and time trends, especially in areas that are underserved by official monitoring stations.

One of the key advantages of Sensor.Community is its open source ecosystem. The software used for data collection, device firmware, and data visualization is completely open source and available in public repositories. This openness encourages continuous improvement by a global community of developers, researchers, and environmental activists. It also allows motivated users to customize and extend the system to meet specific local requirements, such as adding support for additional sensor types or adapting the firmware for different hardware platforms.

In addition, Sensor.Community promotes cooperation with scientific institutions, city authorities, non-profit organizations, and environmental initiatives. Several cities, such as Rennes in France, have integrated Sensor.Community data into their environmental monitoring strategies to supplement official data. Educational institutions use the project as a tool for teaching ecology, developing the Internet of Things, and engaging citizens.

Unfortunately, Sensor.Community does not provide built-in data analysis or air quality forecasting functions on its platform. The network focuses primarily on collecting and disseminating open data, leaving analysis and interpretation to external users, researchers, and third-party applications. Unlike professional or government monitoring systems that often generate predictive models or medical recommendations, the Sensor.Community infrastructure is intentionally kept lightweight and decentralized

Conclusions to Section 2

A review of existing air quality monitoring systems reveals a wide variety of approaches, each with its own strengths and weaknesses. Government networks, such as the London Air Quality Network and AirNow, provide highly accurate, verified data and are reliable sources of public health information. However, these systems are closed by nature: they rely only on officially certified monitoring stations, and individuals cannot connect their own sensors. While LAQN and AirNow offer good data validation and reliable forecasting functions, their availability for independent use is limited, and the

data is typically integrated into fixed government platforms without much flexibility for third-party applications.

Commercial solutions, such as PurpleAir and Kaiterra, provide more flexibility for users, but pose other challenges. PurpleAir allows individuals to buy sensors and join a decentralized network, offering an open map and paid access to the API. However, the system lacks built-in forecasting and data analysis beyond basic corrections, and real-time alerting features are limited without external integrations.

Kaiterra is more focused on enterprise customers, offering professional dashboards, predictive analytics, and customizable alerts. However, participation is limited to customers using Kaiterra's own devices, and the data environment is closed, making it a priority to have organizational control over the sharing of open data.

Open source initiatives such as Sensor.Community represent a different philosophy, promoting citizen science and complete openness. People can create their own low-cost sensors and freely submit data to a global platform. The project supports open access to real-time and historical data and encourages public involvement. However, Sensor.Community does not have built-in forecasting, advanced data analysis, or native alerts. The data is provided in raw form, requiring users to do their own processing or create external systems for visualization and alerts.

3 FORMULATION OF SYSTEM REQUIREMENTS

3.1 Setting requirements for the system

After a thorough analysis of the solutions available on the market, it became clear that many of them have significant limitations in terms of integration with external equipment. In particular, most commercial monitoring systems do not allow users to connect their own sensors, which significantly narrows their functionality and makes it difficult to adapt to specific needs. In this regard, there was a clear request for a system that provides an open architecture and flexibility in connecting any sensors.

The functionality of real-time data analysis requires special attention. At the same time, it is important to preserve historical data: on the one hand, it serves as a basis for training machine learning models, and on the other hand, it is a source for post-analysis of situations that have already occurred.

Another critical aspect is visualization. For this type of system, a visual representation of information does not just improve perception - it is a prerequisite for quick decision-making by the user. Therefore, the system should contain an interface that allows for easy interpretation of both current indicators and forecast results.

Taking into account the specifics of the project and the goals of the demonstration, the implementation is focused on autonomous operation within a single user without the need for scaling or deployment in a distributed environment. This allows us to focus on the quality of the core functionality without complicating the architecture with unnecessary external dependencies.

3.2 Basic requirements

When shaping the architecture and functionality of the system, it is first necessary to define a set of key requirements, without which its operation would be incomplete or inefficient. These requirements reflect the basic functionality that is needed to support the main use cases and meet modern expectations for air quality monitoring systems. They

also cover both technical and analytical aspects that ensure the holistic operation of the project.

The basic requirements include:

a) the ability to process and analyze telemetry data in real time;

b) the function of forecasting air quality using machine learning for the next hour;

c) a module for detecting abnormal values, which allows timely response to potentially dangerous situations;

d) a system for notifying the user in case of exceeding the permissible thresholds;

e) support for connecting arbitrary sensors without reference to a specific manufacturer or device type;

3.3 Additional requirements

In addition to the basic functionality that ensures the system's basic performance, it is important to provide features that significantly improve the user experience and increase the practical value of the solution. These additional features are not critical to the system's launch, but their presence allows the product to become a full-fledged environmental monitoring and analysis tool.

Particular attention is paid to the visual presentation of data, which performs not only informative but also analytical functions. Thanks to the user-friendly graphical interface, the user is able to instantly assess the situation, identify patterns, and observe changes in dynamics, which significantly increases the efficiency of using the system in an everyday context and will generally improve the effectiveness of air pollution control.

Additional requirements include:

a) interactive visualization of air quality forecasting results in a web application in real time;

b) displaying historical data with the ability to navigate, scale and analyze the dynamics of changes in a convenient graphical form.

3.4 Limitations of the system

When designing and implementing the system, a number of limitations were deliberately chosen to focus on the quality of the core functionality while maintaining the clarity of the scope. These limitations define the framework within which the system operates and explain some of the architectural and technical decisions.

Since the goal of the development is to create a prototype that illustrates the full cycle of work, it was decided to work with a limited but well-prepared data set. Large amounts of real data were used for modeling, pre-processed to achieve representativeness: supplemented with artificially simulated anomalies, and brought to a single format. The source was open platforms such as Kaggle, which allowed us to obtain a diverse sample without the need to deploy a full-fledged sensor network [10].

The system also does not integrate with external notification delivery services, such as mobile applications, email, or SMS. Notifications about anomalies and forecasts are provided in the form of a local display, which simplifies the architecture and avoids dependence on third-party platforms. This solution was chosen due to the demonstration nature of the development and the focus on autonomous operation within a single user.

3.5 System architecture

The drawing ІК11.020БАК.006 Д1, which depicts the functional diagram of the system, presents the logical architecture of interaction between individual modules operating in real time. The system is based on a modular approach, which allows each component to perform a highly specialized function, while ensuring a holistic and coordinated operation through messages transmitted via a broker. This scheme allows for high flexibility, scalability, and fault tolerance.

3.5.1 Telemetry data sending module

This component acts as an entry point for sensor information, such as PM2.5, temperature, and humidity values. It generates and transmits messages with the prepared

data to the message broker at a specified frequency. This ensures a constant flow of up-to-date information, which is the basis for further processing and analysis.

As part of the development, data will not come from real sensors, but from a pre-prepared CSV file containing a synchronized time series with all the necessary parameters. This allows you to create the behavior of a real environment and test other system components without the need for hardware.

3.5.2 Analysis module

The analysis module is responsible for pre-processing the input data and generating aggregated hourly metrics. Its task is to structure the data, identify potential deviations, and generate statistical indicators that allow for an assessment of the overall air quality. In particular, the module calculates the average values of temperature, humidity, and PM2.5 concentration for each hour, which allows you to smooth out short-term fluctuations and focus on the overall dynamics of changes.

A key function of the module is also the calculation of the AQI - the Air Quality Index, which is a standardized scale for interpreting the level of pollution. The AQI value is calculated based on the average concentration of PM2.5 and reflects how safe the air is for health. Within the project, AQI is used as a universal indicator that allows for a quick assessment of air quality.

The results of the analysis, including aggregated indicators, are sent to the appropriate notification queue via the broker and stored in the database for further visualization, forecasting, and notification generation.

3.5.3 Forecasting module

This module predicts parameter values for the next hour. To do this, it uses previously obtained values and machine learning models trained on historical data. The forecast is sent through the broker to the appropriate queue and saved for further visualization.

3.5.4 Alert module

After the data is analyzed or forecasted, it is checked for compliance with critical values. If anomalies are detected, the module generates an alert that is sent to the user. Thanks to this, the user promptly receives information about dangerous levels of pollution or other emergency situations.

3.5.5 User interface

This component provides access to information for the end user. Through a web interface, they can view both current real-time forecasts and historical data, including graphs and metrics. The interface can also contain elements for navigating and zooming in on the data.

3.5.6 Message broker

The core of the system's communication infrastructure. The message broker ensures the transfer of information between modules, guarantees the reliability of delivery, and allows each type of module to work asynchronously. The queue environment - incoming data, forecasts, and analytical metrics - allows processing to be organized as independent threads, minimizing delays and increasing the scalability of the architecture.

3.6 Data flow diagram

The drawing ІК11.020БАК.006 D2 shows the general data flow within the system. The system starts with the transmission of raw data from the sensors, which is fed into the input data queue. This data is then processed in parallel by the analysis and forecasting modules. The analysis generates summary metrics that are stored in the database and transmitted to the queue for generating notifications. Forecasting, in turn, creates a

forecast based on the same input data and also writes the results to the corresponding queue and database.

The database acts as a central element for storing information: both historical metrics and forecasts. Based on this data, a user interface is formed that displays graphs and pages for the end user.

Conclusions to Section 3

The analysis of user needs and existing solutions in the field of air monitoring allowed us to formulate clear requirements for the functionality, architecture, and limitations of the system under development. It was found that the market is dominated by closed or low-flexibility systems that do not allow connecting arbitrary sensors, which significantly limits the ability to adapt to specific conditions of use. Therefore, openness and modularity became one of the key principles of the system design.

In the process of formulating the requirements, the emphasis was placed on real-time data processing and analysis, support for anomaly detection, and the ability to make short-term forecasts. Such capabilities are in line with current trends in environmental analytics and are necessary for effective response to changes in air quality. Data visualization in a clear graphical form is another important component that greatly facilitates user interaction with the system, allowing them to quickly assess the current state or view historical changes.

The proposed modular architecture using a message broker provides flexible and reliable communication between components. The system covers all stages of data processing - from sending telemetry to generating forecasts and visualization. This allows not only to monitor the current state of the air, but also to analyze long-term trends and potential risks.

The requirements and architecture chosen reflect both the needs of users and the technical realities of building modern monitoring systems. They provide a solid foundation for the implementation of an effective, adaptive, and practically meaningful solution.

4 CHOICE OF DEVELOPMENT TECHNOLOGIES

4.1 Choice of technologies

Taking into account the functional and non-functional requirements for the system, the key stage is the selection of technologies that will form the basis of the project architecture. First of all, it is necessary to define the core of the messaging - a message broker that will provide asynchronous, reliable, and scalable interaction between the system modules. It must support high throughput, low latency, and horizontal scalability, as the system operates in real time with the potential to support a large number of data sources.

The next step is to choose a programming language that can provide effective integration with the chosen message broker and have a mature ecosystem for working with telemetry data, including building machine learning models, training them, and applying them in real time.

For the user interface, the system requires a modern web framework with advanced support for interactive data visualization. The interface should be intuitive and provide access to historical data, analysis results, and real-time forecasts. Therefore, special attention is paid to the availability of libraries for graphing and intuitive user interaction.

4.2 Message broker

Since the system is based on the principles of asynchronous communication between modules, a key component of its architecture is a message broker. It acts as a transport layer for the transmission of telemetry data, forecasts, analytics, and system notifications. Among the numerous available solutions for implementing such interaction, we analyzed two of the most common options - Apache Kafka and RabbitMQ.

Apache Kafka is distinguished by its focus on processing data streams in real time [11]. It is a platform that was originally developed for high-load systems where not only data transmission but also data storage is important. Its architecture allows you to store messages in the form of sequential logs, which opens up the possibility of re-processing

or reverse analysis. Kafka is easily scalable, both in terms of the number of sources and consumers and the amount of data. This solution is often used in large analytical systems that require stability, speed, and reliability. At the same time, it is worth noting that implementing Kafka requires a deeper understanding of its operating model, proper configuration, and monitoring, which may be overwhelming for small-scale projects.

RabbitMQ, unlike Kafka, is a classic message broker with a long history of use in a wide variety of areas [12]. Its strengths are ease of integration, clear queuing and routing logic, and flexibility in working with messages. RabbitMQ supports several types of exchange that allow you to build both simple and complex delivery scenarios. Thanks to its user-friendly admin interface, extensive customer support, and active community, it can be quickly deployed and customized for the needs of an average system. However, RabbitMQ may be less efficient in the context of streaming large amounts of data. Its model is based on storing messages in queues rather than on their logical historical flow, so it is less suitable for tasks that focus on data analysis over time.

The choice between Kafka and RabbitMQ comes down to a balance between scale, the need for historical data, and the complexity of implementation. In systems focused on intensive processing of streaming data, Kafka looks like a more durable and promising solution, while RabbitMQ remains an effective option for simpler message transfer scenarios. In our case, Kafka is more suitable.

4.3 Machine learning technologies

Since one of the key modules of the system is the forecasting module, the choice of technology for implementing machine learning is of strategic importance. The chosen programming language should ensure not only the simplicity of building and training models, but also the convenient integration of the results into the overall system infrastructure.

The following programming languages were chosen for comparison:

а) Python;

б) R.

Python is the most common choice in machine learning. Its syntax is convenient for both rapid prototyping and building complex systems [13]. This language has an extremely rich ecosystem of solutions that covers all aspects of working with models: from data processing to analyzing the results and implementing them in a productive environment. Most modern research and practical cases in the field of artificial intelligence are implemented in Python, which contributes to the emergence of new tools and the rapid resolution of possible problems due to an active community.

R, for its part, is a powerful language for statistical analysis and academic research [14]. It demonstrates high efficiency in working with tabular data and is convenient for visualizing analysis results. However, in the context of building scalable systems with real-time components, as well as in terms of integration with other modules, R is inferior to Python. In addition, the choice of ready-made solutions in R is much more limited, and machine learning tools are less unified.

As part of the forecasting module, two fundamentally different approaches to time series processing were considered: ARIMA, as a classical statistical method, and LSTM, as a representative of modern neural network architectures. Both methods are well known and widely used in forecasting tasks, but their nature, flexibility, and applicability in complex systems differ significantly.

ARIMA (Autoregressive Integrated Moving Average) is an approach based on the idea that future values of a series can be modeled as a linear combination of its previous values and errors. Its main advantage is simplicity, analytical transparency, and fairly accurate forecasting in the context of well-structured data. This method is relatively easy to interpret, and its parameters have a clear economic or physical interpretation. However, ARIMA requires a stationary series, i.e., no trends and no seasonality without prior transformation. In addition, it is poorly scalable: when new variables, complex interactions, or patterns change over time, the model quickly loses relevance.

LSTM (Long Short-Term Memory), in turn, belongs to recurrent neural networks and is able to "remember" long sequences of changes without losing stability. This makes it particularly effective in situations where the dependencies between values extend far into the past, or when the dynamics of the system change over time. LSTM does not

require an explicitly stationary series, and it copes well with noise, instabilities, nonlinear relationships, and scale changes. It is free from strict mathematical assumptions and learns directly from the data, automatically optimizing internal parameters according to observed patterns. However, this flexibility comes at a price - LSTM models are more difficult to set up, require more computing resources, and do not provide interpretable coefficients, which can be a disadvantage in tasks where it is important to understand the logic behind the forecast.

In general, ARIMA can be useful for quickly creating basic models or when transparent, explainable analytics are required. But for more complex scenarios, when the model needs to adapt to changes, detect hidden trends, and learn over time, a neural network approach using LSTM is much more efficient and promising.

Taking into account the tasks set and the desired flexibility of the system, it was decided to use LSTM as the main architecture for the forecasting module. TensorFlow was chosen as the framework, which provides a stable platform for building and training models, and supports serialization of models for further use in a productive environment [15]. This solution allows further scaling of the model, improving it, or combining it with other approaches, while maintaining the integrity of the architecture.

4.4 Main programming language

The system consists of several functional modules that perform various tasks, from telemetry collection to forecasting and analytics. The choice of the main programming language should ensure not only technical compatibility between the modules, but also ease of support, extensibility, and speed of development.

In theory, the following languages could be used to implement the logic of each module:

а) C#;

б) JavaScript;

в) Python.

All of them are mature, have broad community support, and are capable of implementing the required functionality. C# is well suited for building structured server applications with high performance and has powerful development tools in the .NET environment [16]. JavaScript, in combination with Node.js, allows you to quickly build lightweight server services, especially when it comes to interacting with a web interface or simple real-time event processing [17].

However, given that the machine learning module will be implemented in Python, it is advisable to choose this language as the main language for the entire system. This approach will ensure the unity of the development environment, simplify integration between modules, and allow reuse of data processing utilities in different parts of the system. Python has a rich set of tools for working with data, which makes it especially convenient in the context of systems where analytics, processing, and data transmission occur in real time.

A metrics analysis module that aggregates statistics for a certain period, an alert module that responds to abnormal values, and a telemetry data sending module - all of these components can be effectively implemented in Python without losing performance or increasing complexity. The choice of Python as the main programming language for all internal modules is logical, reasonable, and strategically sound.

4.5 User interface

The user interface is an important component of the system, as it is the interface that provides access to analysis results, forecasts, and historical data. The interface should be not only visually clear but also technically reliable, capable of working with data in real time and providing interactive interaction with the user.

4.5.1 Choosing a web framework

In modern web interface development, using only HTML, CSS, and JavaScript without frameworks is considered an outdated approach. This approach does not provide

enough flexibility, makes it difficult to scale, makes it impossible to reuse components, and does not guarantee security when working with dynamic data. To implement complex interfaces that have to communicate with the database, render graphs, and update in real time, you need to use a modern framework. React and Angular dominate among the most popular options today.

React is a library for building interfaces that is distinguished by its flexibility and component-based architecture [18]. It allows you to split an interface into independent parts, each of which manages its own state and logic. React has a low entry threshold and is actively supported by the community, which guarantees access to a large number of supporting libraries, examples, and templates. One of the important advantages is deep integration with TypeScript, which allows you to use static typing and improves control over data in your application. React does not impose a rigid structure, so it is suitable for both simple and complex applications.

Angular is a full-fledged framework with built-in tools for routing, forms, validation, state management, and much more [19]. Its main strength lies in the rigor of its structure, which is well suited for large teams and corporate projects. Angular was created with TypeScript in mind, so the entire framework is built with typing in mind from the very beginning. At the same time, Angular is more complex than React, requires more time to set up and learn, and requires more code to achieve the same goals.

As part of this project, it was decided to avoid creating a separate API server between the frontend and the database. This approach simplifies the architecture, reduces the number of components, and speeds up development. Instead of the classic client-server separation, an architectural model with built-in server logic is used, which is implemented by the Next.js framework, an add-on to React [20].

In this approach, server-side logic is implemented as part of a web application. It allows you to directly access the database from a React application through server-side functions that run on the server side, not in the browser. This provides secure access to data, control over its processing, and flexibility in implementing business logic. There is no need to create a separate API, which greatly simplifies the system structure.

4.5.2 Selecting additional libraries

In addition to basic technologies, to create a full-fledged user interface, there was a need to use additional libraries that simplify the implementation of complex interface elements, including graphical visualization and styling. The right choice of such tools significantly affects the speed of development, the quality of user interaction, and the future support of the project.

4.5.2.1 Library for graphs

Visualization of data in the form of graphs is one of the central functions of the interface, as the user should be able to quickly assess changes in indicators over time. Two options were considered to implement this function:

a) plotly;

b) chart.js;

Plotly offers high quality visualization out of the box, has many pre-made graph types, and is well suited for scientific or presentation purposes [21]. However, in the context of a project that requires adaptive integration of graphs into the overall design, flexible customization of the visual style, and close interaction with interface events, Plotly is less convenient. Its structure is less suitable for the component-based approach used in React.

Chart.js, on the other hand, allows you to customize the appearance, behavior, and integration of charts with more control [22]. It works better with component logic and allows you to dynamically update data and chart views in response to changes in the application state. This provides not only better integration into the structure of a React application, but also more control in real-time.

4.5.2.2 Library for styling the interface

Another important architectural decision was to use a utilitarian styling library instead of traditional CSS. It was decided to use Tailwind CSS, a tool that allows you to create responsive, modern interfaces without the need to write separate CSS files [23]. Tailwind is based on the atomic class approach, where styles are set directly in the HTML structure of components.

This approach ensures greater style consistency, speeds up development, facilitates refactoring, and eliminates the need to maintain separate global style sheets. In addition, it allows you to see the final result immediately without unnecessary switching between HTML and CSS, which is especially useful when prototyping and working on responsiveness.

4.6 Database

The database is the central element of storing all the information of the system - from telemetry data received every few minutes to generated forecasts and analytical metrics. The peculiarity of this system is active work with time series and storage of predicted values, which can be in the form of arrays. Accordingly, when choosing a database, it is important to consider not only performance but also flexibility in presenting the data structure, the ability to effectively aggregate, filter, and view historical data.

It was considered:

а) Microsoft SQL Server;

б) PostgreSQL;

в) MongoDB.

Microsoft SQL Server is a commercial platform with a wide range of features focused on large corporate solutions [24]. It provides stable performance, has convenient administration tools, and is well suited for classic relational structures. However, the use of SQL Server involves license costs, as well as tying to the Microsoft ecosystem. In the context of an open, experimental, or academic project, this creates unnecessary restrictions. In addition, working with time series requires additional manual optimization, and support for storing structured arrays in a composite form is limited.

PostgreSQL is a completely free, open-source database that is as good as its commercial counterparts in terms of functionality [25]. It supports advanced data types, such as arrays, which is especially useful in the context of storing forecasts consisting of several values for the future period. PostgreSQL is also good at storing time series - it allows you to conveniently filter data by time intervals, perform aggregations, and index by time stamps. If necessary, it can be extended with specialized extensions for time series analytics. In the context of the project, PostgreSQL seems to be the most optimal solution - free, scalable, easy to use, and with a sufficient level of flexibility.

MongoDB is a document-oriented database that stores data in the form of JSON-like objects [26]. It is free, although it is developed and maintained by a commercial company. MongoDB is often used for rapid development, especially when the data structure changes dynamically. MongoDB can store arrays. Although it has mechanisms for storing time series, they are less flexible and productive than solutions based on classical relational models. For tasks where it is important to process sequences of values over time and perform complex aggregations, MongoDB may be less convenient.

Given the specifics of the data, the need to store both time series and arrays of forecasts, and the desire to avoid license dependency, PostgreSQL seems to be the most balanced solution - reliable, flexible, and technologically appropriate for the tasks of this project.

4.7 Additional tools

To facilitate the development, testing, and monitoring of the system, a number of auxiliary tools were selected that are not part of the main logic but significantly affect the efficiency of the project.

The key tool for local deployment is Docker [27]. Its use allows you to run individual system components - database, message broker, web application - in separate containers with a clearly defined environment. This guarantees the stability and repeatability of the configuration at any stage of development and allows you to run the entire system as a whole with one set of commands.

To work with the database, we will use pgAdmin, which provides a graphical interface for viewing tables, writing SQL queries, analyzing the data structure, and monitoring the current state of connections [28]. This is especially useful during development when you need to quickly check what data has been saved and how it is being processed.

To monitor the message broker, we will use Kafka UI, a tool that provides access to tops, shows the latest messages, allows us to analyze consumers, and check the logic of communication between modules [29]. This will allow us to detect errors in routing or lagging modules when processing data in real time.

Change management and version control will be carried out using Git [30]. This will allow you to store the history of changes, work in isolated branches, synchronize updates between system components, and ensure code stability at each stage.

Visual Studio Code [31] was chosen for development. This is a flexible environment that allows you to work simultaneously with multiple terminals, run the server and client side in parallel, edit TypeScript and Python code, and integrate with Docker and Git. This approach provides full control over the system during its development.

Conclusions to Section 4

As a result of the analysis of the project requirements and technical features, a coordinated technology stack was formed, which provides an efficient, scalable, and easy-to-develop system architecture.

To organize interaction between the modules, we chose the Apache Kafka message broker, which allows you to transfer data in real time, store messages in the form of streams, and scale the system without losing performance. An alternative option, RabbitMQ, was rejected due to its lesser suitability for the streaming nature of the task.

Python was chosen as the main programming language for the system, as it has the widest ecosystem of tools for data processing and implementing machine learning

models. Python will provide the implementation of all the main modules: sending telemetry data, analyzing metrics, forecasting, and alerts.

To implement the forecasting module, we also used Python as the most flexible language with the largest number of solutions for machine learning tasks. This choice was made not only because of the broad support from the community, but also because of the availability of powerful frameworks, such as TensorFlow, that allow implementing modern neural network architectures. Within the system, it was decided to focus on using an LSTM model, which is the most effective in predicting time series in complex dynamic conditions.

PostgreSQL was chosen as the database, which is a free, open, and flexible platform. It supports array storage, which is necessary for storing forecasting results, and also works well with time series, which is the key data type in the system.

The user interface will be implemented using React in combination with Next.js, which allows combining client and server logic within a single project. The main interface language is TypeScript, which ensures typing, reliability, and ease of development. It was decided not to create a separate API for interacting with the database - instead, the server-side functions of the Next.js framework are used, which allows you to directly access the database from the interface.

Chart.js was chosen for data visualization in the web application because this library provides more flexibility in customization and integrates better with the React component architecture than the alternatives. Tailwind CSS is used for styling the interface, which allows you to quickly create modern, responsive interfaces without cumbersome CSS files.

5 SYSTEM DEVELOPMENT

5.1 Data preparation

The first stage of the system development was the formation of a high-quality dataset that would allow us to build a reliable forecasting model and adequately reproduce the system's behavior in the real environment. We analyzed a large array of real environmental data covering the key parameters of the air environment - temperature, humidity, and concentration of fine particles PM2.5.

To create a universal and comprehensive data set for model training, the original values were not only cleaned but also purposefully enriched with synthetic anomalies. This approach made it possible to model non-standard situations that the system may encounter in real use: sharp jumps in indicators, gradual trends, noise bursts, etc. This is especially important for anomaly detection and correct forecasting beyond typical scenarios.

To build the dataset, we used the Kaggle platform, which provides access to open datasets with historical air quality data for urban areas [10]. This data covers a wide range of parameters, including concentrations of suspended particles, levels of carbon monoxide, nitrogen dioxide, ozone, temperature, humidity, and accompanying meteorological information. After filtering, normalizing, processing missing values, and merging data from multiple sources, a single structured dataset of about 100,000 labels was formed, each corresponding to a 5 minute time interval.

Such a high sampling rate ensures a high density of records in the time dimension, which allows us to accurately model the dynamics of changes in the state of the environment not only within one day, but also on interweekly horizons. Thanks to this approach, the generated dataset not only adequately reflects the behavior of the system in real conditions, but also lays the foundation for building reliable models for predicting environmental parameters. Figure 5.1 shows the generated dataset with the values of temperature, humidity and fine particles of PM2.5 for the entire time.

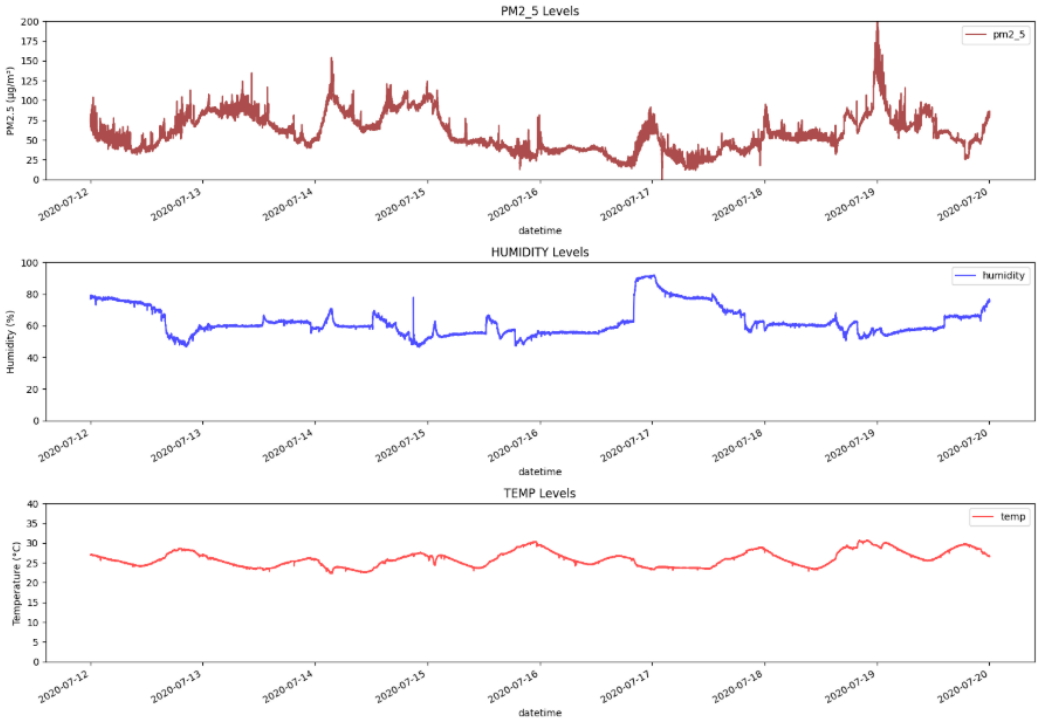

Figure 5.1 - Generated dataset with PM2.5, humidity, temperature values for the whole time

5.2 Machine learning model

To implement the predictive module, we chose an LSTM neural network architecture implemented using the TensorFlow library. The task of the model is to predict the PM2.5 concentration for the next hour based on the data for the previous 24 hours. This approach allows to take into account the dynamics of changes in a long-term time window. The use of LSTM provides flexible training on time series without the need to bring them to a stationary form, allowing the model to detect hidden patterns and complex nonlinear dependencies in a dynamic environment.

5.2.1 Training the machine learning model

A clear structure of input and output data was defined. The model receives 288 previous marks as input - equivalent to 24 hours of data, given a five-minute interval between records - and generates a forecast for the next 12 marks, i.e. one hour ahead.

Before training, all input data was normalized to the range from 0 to 1 using the Min-Max Scaling method. The data was divided into three parts: 70% for training, 15% for validation, and 15% for testing.

The model architecture is built sequentially. The input layer is a 128-neuron LSTM layer, followed by a fully connected layer that converts the last LSTM output into a 12-value forecast for the next hour.

The model was trained using the Adam optimizer with a learning rate of 0.001 and a Mean Squared Error (MSE) loss function.

The model training process was based on the proven practices of modern machine learning for time series and resulted in a stable, sensitive, and dynamically adaptive forecasting system. Figure 5.2 shows the model training process.

```
Epoch 1/8
578/578 - 112s - 194ms/step - loss: 0.0010 - rmse: 0.0322 - val_loss: 2.5623e-04 - val_rmse: 0.0160
Epoch 2/8
578/578 - 111s - 192ms/step - loss: 2.3568e-04 - rmse: 0.0154 - val_loss: 2.0241e-04 - val_rmse: 0.0142
Epoch 3/8
578/578 - 191s - 330ms/step - loss: 2.2127e-04 - rmse: 0.0149 - val_loss: 3.3350e-04 - val_rmse: 0.0183
Epoch 4/8
578/578 - 109s - 189ms/step - loss: 2.2257e-04 - rmse: 0.0149 - val_loss: 3.5671e-04 - val_rmse: 0.0189
Epoch 5/8
578/578 - 116s - 201ms/step - loss: 2.1582e-04 - rmse: 0.0147 - val_loss: 3.7316e-04 - val_rmse: 0.0193
Epoch 6/8
578/578 - 117s - 203ms/step - loss: 2.1243e-04 - rmse: 0.0146 - val_loss: 4.3831e-04 - val_rmse: 0.0209
Epoch 7/8
578/578 - 115s - 199ms/step - loss: 2.1058e-04 - rmse: 0.0145 - val_loss: 4.1163e-04 - val_rmse: 0.0203
Epoch 8/8
578/578 - 112s - 194ms/step - loss: 2.0654e-04 - rmse: 0.0144 - val_loss: 6.0672e-04 - val_rmse: 0.0246
```

Figure 5.2 - the process of training the LSTM model over eight epochs

Figure 5.3 demonstrates the change in the loss function on the training set obtained by plotting the graph during the model training. This graph displays the dynamics of error reduction with each epoch, which allows you to visually evaluate the efficiency of optimization, the stability of the training process, and identify possible signs of overtraining or stalled gradients.

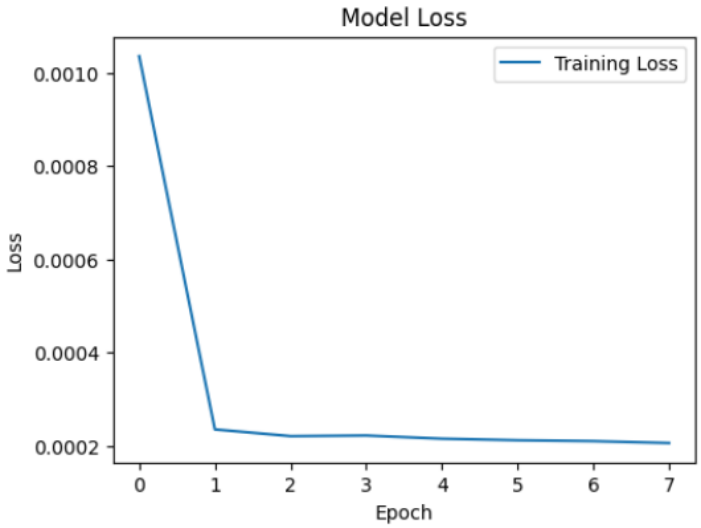

Figure 5.3 - Graph of the loss function on the training sample

5.2.2 Evaluation of the trained machine learning model

After completing the training phase, the model was tested on an independent test dataset that was not used during training or validation. Two metrics were chosen to assess the accuracy of the predictions: MAE (Mean Absolute Error) and RMSE (Root Mean Squared Error). The first metric - MAE - shows the average absolute error between the actual and predicted values, regardless of the direction of deviation. The second one, RMSE, is more sensitive to large errors and provides information on the standard deviation of forecasts from reality.

On the test set, the model achieved MAE = 1.850 and RMSE = 3.379, which indicates high accuracy. Given that the input values of PM2.5 in the dataset range from 0 to 200 µg/m³, these metric values can be considered relatively low, and thus the quality of the forecasts is satisfactory.

Figure 5.4 shows a scatter plot of model errors where the X-axis is the actual values and the Y-axis is the predicted values. An ideal forecast would have formed a dense cluster along the diagonal. As you can see from the graph, most of the points are indeed concentrated along this line, which confirms the quality of the model. There are minor deviations at high PM2.5 concentrations, but even in these cases, the predictions remain close to the actual values.

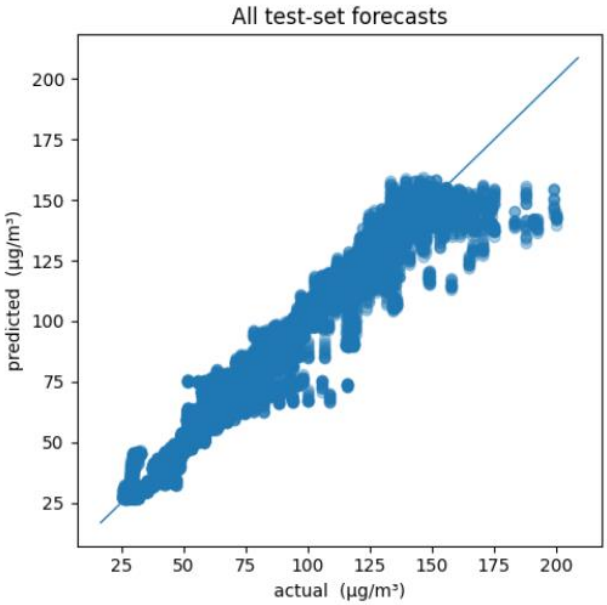

Figure 5.4 - Scatter plot of model errors on test data

In general, the model demonstrates stable behavior, high accuracy, and adaptability to complex dynamics of changes, which makes it suitable for use in real-world air quality monitoring and forecasting.

5.3 Creating a database

To store the results of forecasting, statistical analysis, and subsequent visualization, the system created two main tables in the PostgreSQL database. They form the basis for long-term storage of aggregated and model data, which is used by both internal modules and the user interface.

The first table - forecast - is intended to store the results of the forecasting module. It contains the following fields:

and) id - a unique record identifier;

b) date_forecasted - the date and time when the forecast was made;

c) date_target - the date and time for which the forecast was made;

d) input_sequence - an array of values that were fed to the model input (for example, the last 288 points);

e) predicted_sequence - an array of predicted values for the next hour;

f) pm2_5 - the average value of PM2.5 for the forecast period.

This table is used to analyze the accuracy of the model, build graphs, and further calculations that require historical forecasts.

The second table - hourly_sensor_metrics - stores aggregated hourly statistics generated by the analysis module. It includes:

a) id - a unique identifier;

b) window_end_time - the end of each hourly window;

c) avg_temp - average temperature per hour;

d) avg_humidity - average humidity;

e) avg_pm2_5 - the average value of PM2.5;

f) aqi - the calculated Air Quality Index.

These metrics allow the system to record the general state of the environment at any given time, as well as to respond to deteriorating air quality.

In the future, these tables will be actively used by the analysis and forecasting modules to read, update, and save data, and will become the main source of information for the user interface, which will visualize the dynamics of changes in the air environment, historical trends, and predicted values.

5.4 Data sending module

This module serves as a source of telemetry data in the system. It is implemented as a simple Python script that reads values from a pre-prepared CSV file and sends them to Kafka on the corresponding top named "in". Each record in the file contains temperature, humidity, and PM2.5 values along with a time stamp, and is submitted as a message to the broker.

The module allows you to configure basic parameters, such as specifying the path to the data file and setting the interval between sending messages. This makes it possible to emulate real-time data flow or to run the system at an accelerated pace for testing and demonstration. The data sending module generates the initial flow of information for the

entire system, simulating the behavior of real sensors without the need for physical hardware.

5.5 Data forecasting module

The prediction module is implemented in Python, and its main task is to listen to the "in" top in Kafka, where new telemetry messages from the sending module or real sensors appear, accumulate them in the form of sequences, and prepare them for prediction. After accumulating a sufficient number of values (288 labels, i.e. 24 hours of data), the module formats this data in a form compatible with the LSTM model. Based on this sequence, the model makes a forecast for the next hour (12 labels).

The forecast results are transmitted in two directions. First, they are sent to the Kafka top "forecast", which allows other modules to receive up-to-date forecasts in real time. Secondly, the forecasts are saved to the database, in particular to the forecast table, which allows for historical analysis or visualization.

5.6 Data analysis module

The analysis module is responsible for processing the incoming telemetry data stream from the "in" Kafka top. It aggregates this data into hourly windows and calculates key statistical metrics: average temperature, humidity, and PM2.5 concentration. Based on the average PM2.5 value, it additionally calculates the AQI - - a standardized indicator that allows you to classify the degree of air pollution according to a health risk scale.

AQI is calculated using linear interpolation according to national environmental standards. Table 5.1 shows the scale used by the module:

Table 5.1 - Scale of conversion of PM2.5 to AQI

PM2.5 (μg/m³)	Relevant AQI	Air quality category
0.0 - 12.0	0 - 50	Okay.
12.1 - 35.4	51 - 100	Moderate
35.5 - 55.4	101 - 150	Unhealthy for sensitive groups

PM2.5 (µg/m³)	Relevant AQI	Air quality category
55.5 - 150.4	151 - 200	Unhealthy
150.5 - 250.4	201 - 300	Very unhealthy
250.5 - 350.4	301 - 400	Dangerous pollution
350.5 - 500.4	401 - 500	Extremely dangerous pollution

After calculation, all metrics - average values of temperature, humidity, PM2.5, and AQI - are stored in the database in the hourly_sensor_metrics table. In addition, the result is sent to the Kafka top "metrics" so that other components of the system, including the alert module and the user interface, can quickly respond to changes in the air condition.

5.7 Alert module

The alerting module is responsible for responding to critical values of the air quality index and informing the user about potential health hazards. It constantly listens to the "metrics" Kafka top, which contains hourly aggregated metrics from the analysis module.

As soon as the incoming message detects that the AQI value exceeds 150, the module automatically generates a text warning that is displayed in the console. Such a threshold corresponds to the "Unhealthy" category, when air pollution begins to pose a serious threat not only to sensitive groups but also to the general population.

The message contains a concise and informative interpretation of all key parameters, including recommendations for the user. Figure 5.5 shows an example of such a notification.

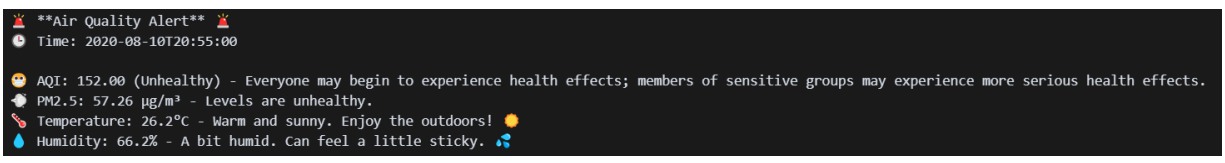

Figure 5.5 - An example of an alert message generated by the module

The message contains the date and time, an explanation of the AQI value, and brief comments on PM2.5, temperature, and humidity. This form of information presentation allows the user to quickly assess the level of danger and decide on their further actions.

This module can be expanded in the future, for example, by sending push notifications, emails, or integrating with other notification systems.

5.8 User interface

The system's graphical interface is implemented as a web application called SensorData, built using the modern Next.js framework. Its architecture combines client and server logic, and thanks to server actions, the application can securely connect directly to the database without the need for an intermediate API. This provides quick access to up-to-date information and minimizes delays between data acquisition and visualization.

The interface contains three functional pages:

а) Live;

б) History;

в) About.

"Live" is the main monitoring page that displays the current forecast for the next hour and current aggregated metrics - average temperature, humidity, PM2.5, and the Air Quality Index (AQI).

The forecast is implemented in the form of an interactive graph, where the last two hours of real data are displayed in blue, and the forecasted values for the next hour are displayed in yellow. This visual approach allows you to intuitively see the continuation of the dynamics and assess how well the forecast matches previous trends. The page is shown in Figure 5.6.

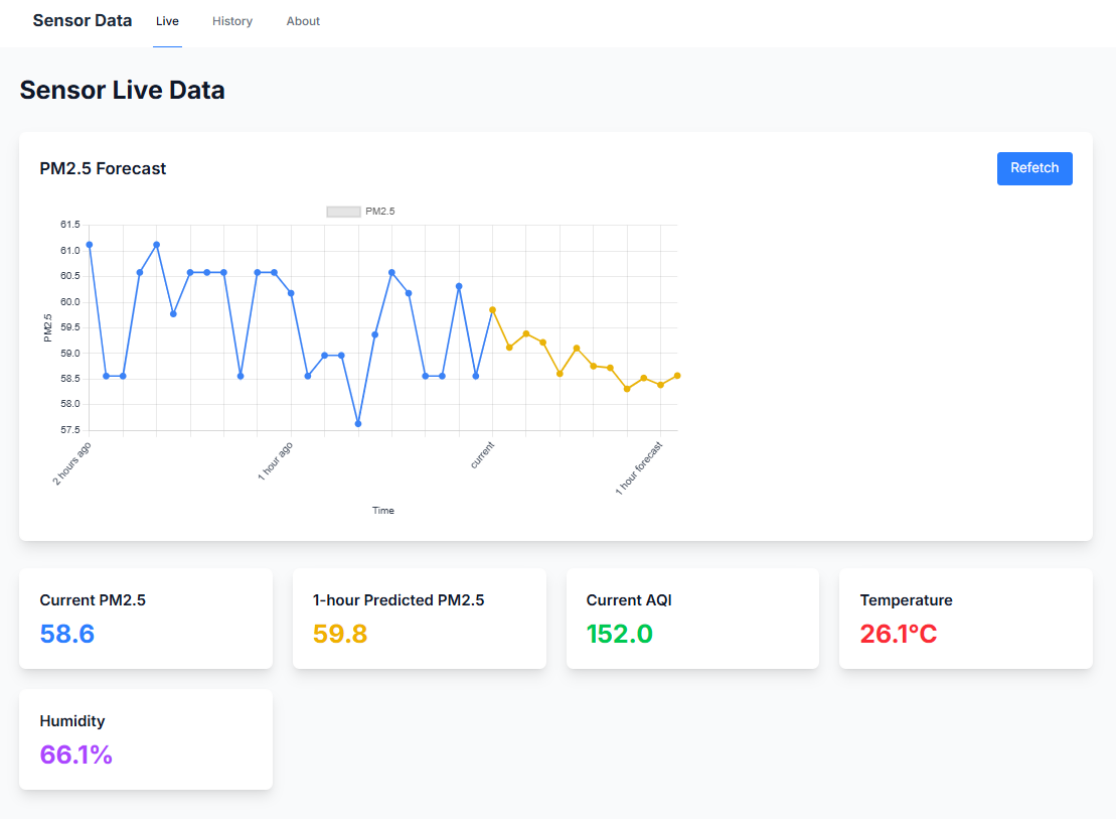

Figure 5.6 - Live main page with current forecast and metrics

The History page displays graphs of four key parameters - AQI, PM2.5, temperature and humidity - with an hourly resolution for the entire period of system operation. All data is interactively aggregated and displayed in a user-friendly way, allowing you to track both short-term changes and long-term trends in the air environment in real time.

Graphs allow you to analyze seasonal fluctuations in detail, detect pollution outbreaks, and identify periods of stability or abnormal deviations from the norm. Thanks to data visualization, users can not only assess the current state of the air, but also compare different time periods, which is especially useful for identifying the impact of external factors such as weather conditions or anthropogenic load. The data collected on this page can be used for both rapid response and long-term planning of air quality improvement measures. The page itself is a key element of the system's interface, as it provides transparency, convenience and visibility of monitoring, as shown in Figure 5.7.

Figure 5.7 - History page, AQI and PM2.5 graphs

The "About" page contains a brief description of the project, its goals, implemented architecture, and main ideas. It serves as a reference section for a user or potential researcher to familiarize themselves with . The page is shown in Figure 5.8.

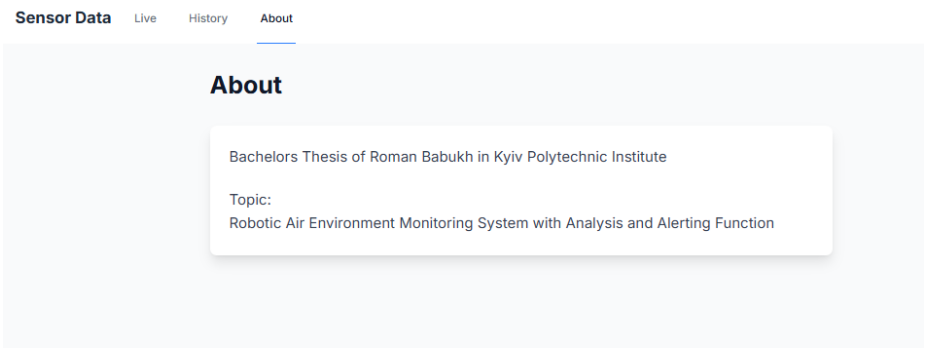

Figure 5.8 - "About" page

All charts in the interface are implemented using the Chart.js library, which provides high interactivity: charts can be scaled, moved, and the cursor can be moved to specific points to see the exact value of a parameter at a selected time. This allows you to

study the behavior of data in detail during specific periods and makes analysis easier for the user.

5.9 BPMN diagram

To illustrate the logic of the developed air environment monitoring system, a business process diagram was constructed, shown in drawing ІК11.020БАК.006 Д3, which demonstrates the sequence of key processes. The diagram is divided into logical blocks corresponding to the individual modules of the system: data transmission, processing, forecasting, alerting, and user interface.

The process starts with sending raw data from sensors to the Kafka message queue. The data is transferred to the analysis and forecasting modules in parallel. In the analytical unit, the data is aggregated for an hour, AQI is calculated, and then the metrics are saved to the database.

The forecasting module prepares the data, generates a forecast using a machine learning model, and the results are also saved to the database and published in the appropriate queue.

Next, the system checks whether the AQI value exceeds the set threshold of 150 and, if it does, generates appropriate notifications. At the same time, the user interface periodically accesses the database, renders pages and graphs, displaying all relevant information in a clear way for the user.

This diagram reflects the actual implementation, which allows you to clearly trace the data flow and the logic of interaction between system components.

Conclusions to Section 5

Within this section, we implemented a system for monitoring air quality parameters, which includes data collection, processing, forecasting, analysis, data visualization, and alerts. Each module performs a separate function, and the interaction between them is provided through the Kafka message broker and a centralized database.

The user interface is implemented as a web application called "SensorData" using Next.js. It connects directly to the database using server-side actions. The interface consists of three pages: Live - for viewing current metrics and forecast in the form of a combined graph; History - with historical graphs of AQI, PM2.5, temperature, and humidity; and About - with a description of the project. The visualization is based on the open-source Chart.js library, which allows you to scale the graphs, navigate them, and view specific point values.

A BPMN diagram was created to review the processes in the developed system.

6 SYSTEM TESTING

6.1 Overview of system testing

This section will describe the testing process of the developed air environment monitoring system. The testing will be carried out on the basis of the precedent diagram shown in the drawing ІК11.020БАК.006 Д4, taking into account the actions of all actors indicated in the diagram. The main goal will be to verify that the implemented functionality meets the requirements defined at the design stage, as well as to assess the stability of the system and its usability.

The testing will check whether the data is stored correctly, whether the analytical and predictive modules work, and whether notifications are received in case of anomalies. Particular attention will be paid to the interaction between the modules via the message broker, as well as the correct updating of information in the interface.

6.2 Starting the system

The process of launching the system will involve the step-by-step deployment of all necessary infrastructure components and program modules by the Administrator. The main platform for managing services will be Docker, which provides an isolated environment for each component.

The infrastructure will be deployed using the docker compose up command, which will bring up all services according to the configuration.

The infrastructure includes:

a) Apache Kafka;

b) Zookeeper auxiliary component [32];

в) PostgreSQL;

г) Kafka UI;

е) pgAdmin.

Figure 6.1 shows a screenshot from the Docker Desktop showing the running containers.

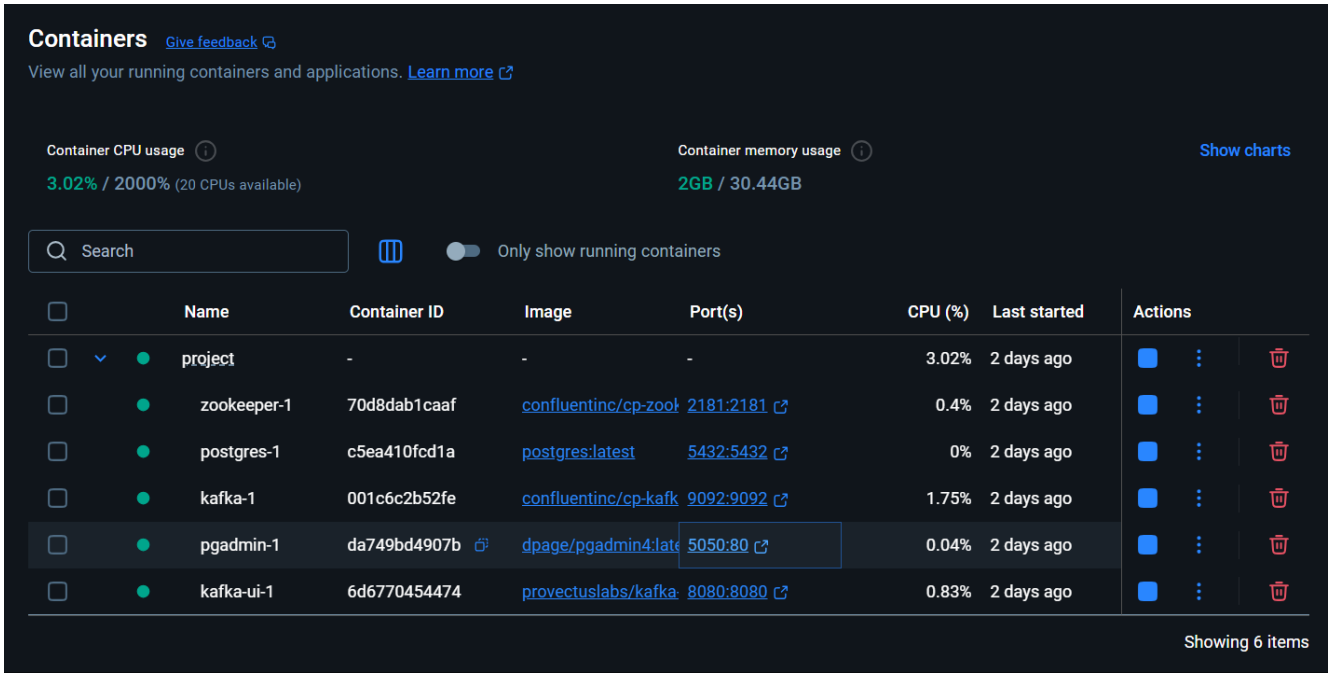

Figure 6.1 - Infrastructure components in Docker Desktop

After initializing the containers, you will need to install all the dependencies for Python scripts and the frontend. To do this, run the commands in the appropriate directories:

a) pip install -r src/requirements.txt;

b) npm install (in the src/sensor-data-web-app directory).

The next step is to create the necessary database structure. To do this, run the initialization script: python src/db_setup/init_db.py

After that, the main computing modules responsible for analysis, forecasting, and alerts will be launched. All of them can be launched in separate terminals with the following commands:

a) python src/sensor-data-analyzer/main.py;

b) python src/sensor-data-forecaster/main.py;

c) python src/sensor-data-notifier/main.py.

The final step is to launch the web application that provides the user interface. To do this, run the command in the frontend directory: npm run start (in the src/sensor-data-web-app directory).

6.3 Connecting the sensors

According to the precedent diagram, the role of connecting sensors is performed by the Administrator. In the context of testing, this function will be realized by uploading a prepared data set to the system.

For testing purposes, a set of 15 thousand telemetry records will be used that were not included in the training set of the machine learning model. This will allow not only to check the stability of data processing in the stream, but also to evaluate the accuracy of forecasting and the system's behavior in cases of anomalies.

Data will be transferred to the system by running a separate script: python src/sensor-data-sender/main.py

This process activates the sending of data to the message broker, after which the entire infrastructure can start working in real time. Figure 6.2 shows the logs of sending telemetry records.

Figure 6.2 - Logs of telemetry data sending

6.4 Testing the modules

After sending the input data, we tested the functioning of all key modules of the system: analysis, forecasting, and alerting. According to the diagram of precedents, interaction with these modules takes place on behalf of the Sensors that send input data to the system. The purpose of testing is to make sure that all components respond to

incoming information, perform the necessary calculations, and generate the expected results.

The analysis module is responsible for processing incoming telemetry values and generating hourly aggregated metrics. Its work is confirmed by the logs shown in Figure 6.3.

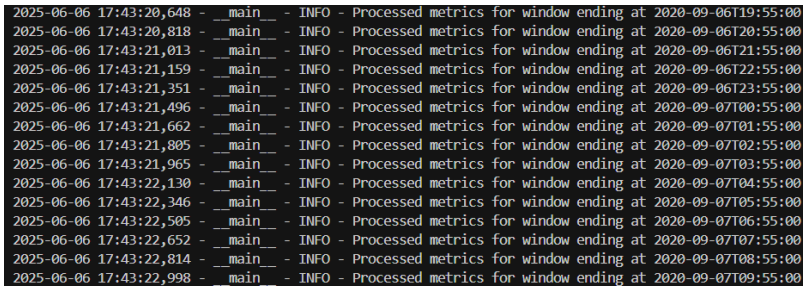

Figure 6.3 - Logs of the analysis module

In addition, the preservation of the processed metrics is confirmed by viewing the contents of the hourly_sensor_metrics table in pgAdmin - its example is shown in Figure 6.4. You can see the presence of various data in the tables that appeared as a result of stable operation of the system

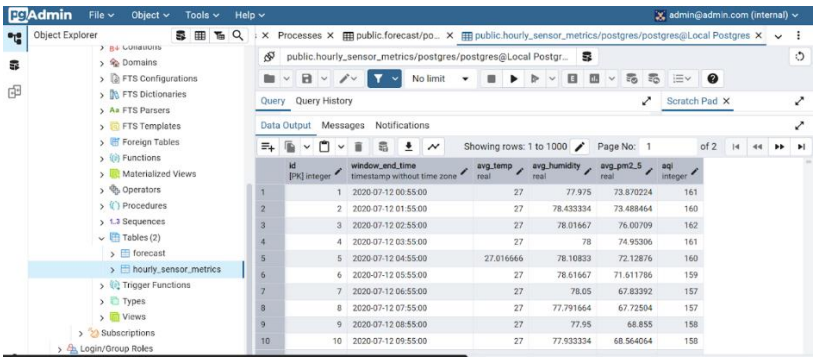

Figure 6.4 - Screenshot from pgAdmin, hourly_sensor_metrics table

The forecasting module provides air quality forecasts for the next hour. It runs a machine learning model using the latest telemetry values. The forecast results and dates are displayed in the logs, a fragment of which is shown in Figure 6.5.

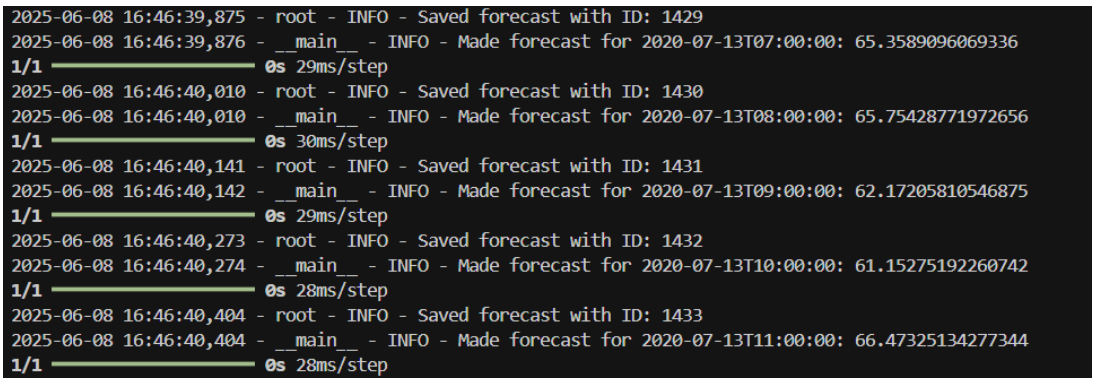

Figure 6.5 - Logs of the forecasting module

The alert module generates notifications when dangerous or critical parameters are detected. Figure 6.6 shows an example of such an alert. It indicates:

a) AQI: 168.00, which is classified as "Unhealthy", meaning that the air quality is harmful to the health of all people, especially sensitive groups;

b) pm2.5: 89.10 µg/m³, which exceeds the permissible limits and is labeled "unhealthy";

c) temperature: 24.6°C, described as warm and sunny - encouraging outdoor activities;

d) humidity: 59.8%, considered comfortable.

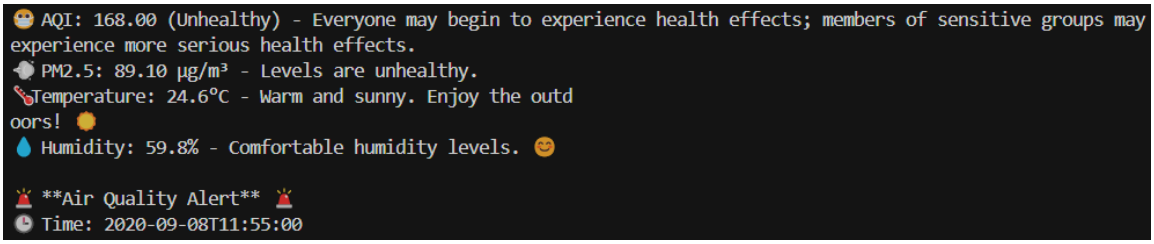

Figure 6.6 - Example of an air quality alert

Additionally, the stable operation of the message broker was confirmed through the Kafka UI. Figure 6.7 shows the availability of the corresponding data queues, as well as the number of messages in each of them. This allowed us to make sure that no data was lost and that all modules received the necessary information.

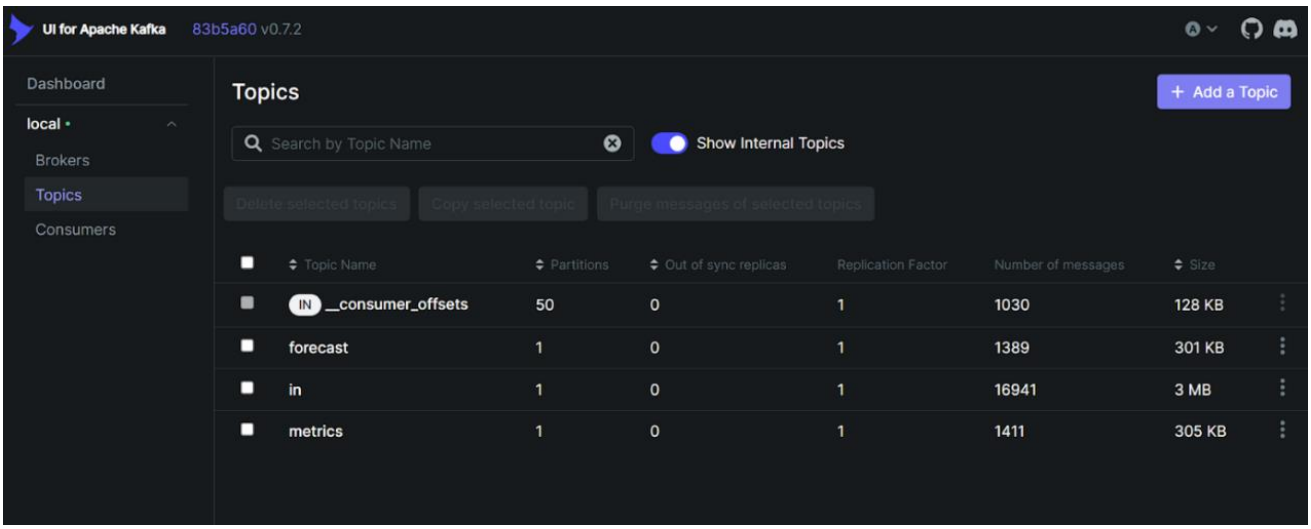

Figure 6.7 - List of data queues from the Kafka UI

6.5 Testing the user interface

In the previous section, Figures 5.6 and 5.7 demonstrated the correct operation of the web application, where the display of forecasts, metrics, and historical data was confirmed. At this stage, testing will focus on identifying possible errors in the interface, as well as checking the usability and functionality of navigation elements.

The interface successfully displays the state of missing data: all metrics are zero, graphs are empty, but the overall appearance of the page remains neat, without shifts, errors, or incorrect formatting. This will indicate that the system adequately handles cases when data has not yet been received and does not create an unnecessary burden on the user.

Figure 6.8 shows the main page of the application before the data is sent from the sensors.

Particular attention was paid to the adaptability of the interface - the design remains coherent and harmonious on different devices, regardless of screen size. The transition between pages and data display is smooth, without noticeable delays or visual artifacts, which ensures a positive user experience.

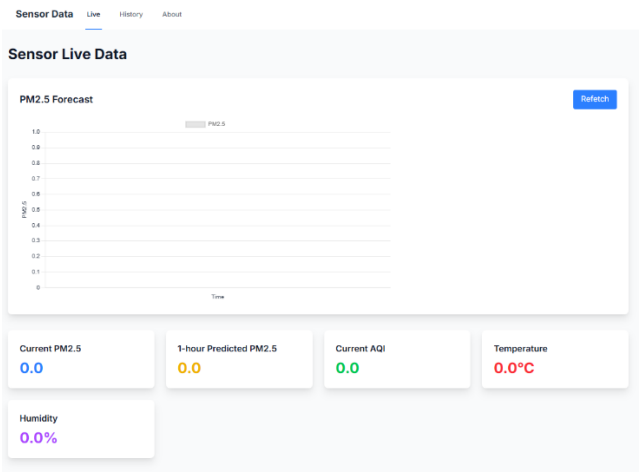

Figure 6.8 - User interface before sending sensor data

The next step is to test the interactivity of the graphical representation of the data. One of the key features is the ability to zoom in on individual graph fragments, which is important for easy analysis of trends and anomalies.

Figure 6.9 shows a graph of historical air quality index data and a red rectangle marks the area that will be approximated for testing.

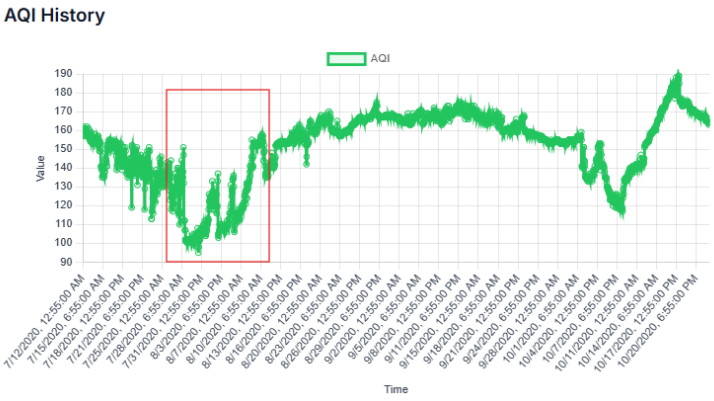

Figure 6.9 - Plot of the graph for the approximation

Figure 6.10 shows the result of scaling the selected area. As you can see, the region with the data has zoomed in, and it has become much easier to view and study data and trends on . The functionality works stably, the interface responds smoothly, providing comfortable navigation.

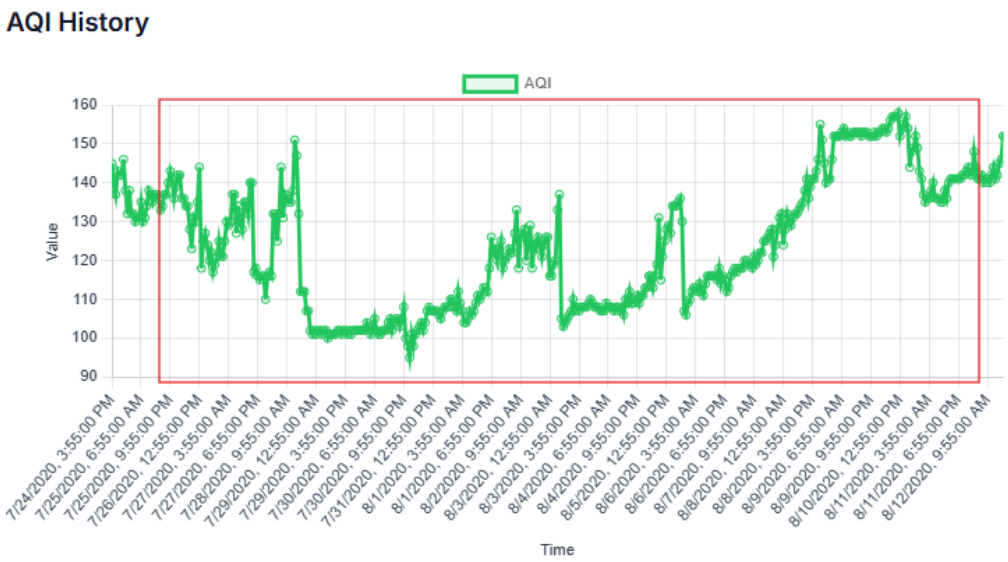

Figure 6.10 - Graph plot after approximation

The test will confirm that the user interface correctly displays data, provides intuitive user interaction, and adapts to changes in the system state.

Conclusions to Section 6

In this section, we performed comprehensive testing of the air monitoring system to verify the correct functioning of all components and the compliance of the implemented architecture with the requirements set at the design stage. The testing covered the technical aspects of launching and the user experience of interacting with the interface. The testing was carried out using a precedent diagram.

At first, the system infrastructure was deployed using Docker, including Kafka, PostgreSQL, and auxiliary tools such as Kafka UI and pgAdmin. After installing all the dependencies and creating the database structure, the system was ready for testing.

The analysis, forecasting, and alerting modules were tested. Each of them demonstrated the expected behavior: hourly metrics were generated, forecasts were made, and alerts were generated in case of detection of dangerous values. Attention was also paid to checking inter-module communication through the message broker, as well as storing information in the database. These processes were recorded in logs and further confirmed by screenshots from Kafka UI and pgAdmin.

In addition, the user interface was tested. It was checked how the system reacts to the absence of input data, whether the interface displays zero values correctly, and whether it is convenient for the user to interact with the graphs. It was confirmed that the visual components work stably, the scaling of graphs is smooth, and the overall structure of the pages remains intuitive.

The adaptability and quality of the interface design were also assessed. The system scales correctly to different screen sizes and devices, while maintaining the structure, aesthetic appearance, and readability of all elements. Transitions between pages and data updates are smooth, without visual defects or delays, which ensures a comfortable and pleasant user experience regardless of the scenario.

CONCLUSIONS

As a result of the diploma project, a full-fledged air quality monitoring system was implemented that allows analyzing sensor data in real time, making predictions using a machine learning model, generating alerts about environmental degradation, and visualizing the results for the user through a web interface. The main idea of the project was to create a universal solution that can be adapted to urban, agricultural, or individual needs.

During the development process, modern technologies were used, including the Next.js framework for building a user-friendly interface, Apache Kafka as a message broker for communication between modules, Python for implementing data processing and training the LSTM forecast model, and a database management system for storing historical values and metrics. The interface is implemented using a server-side rendering approach and supports modern functions for interacting with dynamic data.

The architecture of the system is based on the principles of asynchronous interaction between modules, where each component performs its own function and can develop independently. This makes it easy to adapt or replace any unit in the future without the need to modify the entire system. Particular attention was paid to the separation of duties between services, automation of processing processes, and reliability of message transmission.

The result is a system that meets all the requirements: it stably processes incoming environmental data, allows you to assess the state of the air, detects abnormal indicators in a timely manner, and offers the user a clear, user-friendly visualization. Such a solution can be implemented as a separate local tool for private use as part of a larger environmental or research project.

LIST OF REFERENCES

1. Map of deaths from air pollution due to the use of fossil fuels, 2015. URL: https://ourworldindata.org/data-review-air-pollution-deaths (accessed 25.05.2025).

2. Location of settlements with PM2.5 data, by number of ground-based measurements, 2010-2019. URL: https://www.epa.gov/outdoor-air-quality-data (accessed on May 25, 2025).

3. Interactive map of SaveEcoBot. URL: https://www.saveecobot.com/en (accessed on May 25, 2025).

4. LAQN home page with an interactive map. URL: https://www.londonair.org.uk/ (accessed May 25, 2025).

5. Interactive map of AirNow. URL: https://www.airnow.gov/ (accessed 25.05.2025).

6. PurpleAir interactive map URL: https://map.purpleair.com/ (accessed 25.05.2025).

7. PurpleAir developer panel URL: https://develop.purpleair.com/ (accessed May 25, 2025).

8. Kaiterra Data Platform URL: https://www.kaiterra.com/dashboard (accessed May 25, 2025).

9. Home page of Sensor.Community. URL: https://sensor.community/ (accessed May 25, 2025).

10. Kaggle is a platform for data analysis. URL: https://www.kaggle.com/.

11. Apache Kafka streaming data processing platform. URL: https://kafka.apache.org/.

12. RabbitMQ messaging system. URL: https://www.rabbitmq.com/.

13. Python programming language. URL: https://www.python.org/.

14. The R programming language. URL: https://www.r-project.org/.

15. TensorFlow machine learning library. URL: https://www.tensorflow.org/.

16. C# programming language. URL: https://learn.microsoft.com/en-us/dotnet/csharp/

17. Javascript programming language. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript

18. React frontend framework. URL: https://react.dev/.

19. Angular frontend framework. URL: https://angular.io/.

20. Next.js framework for server-side rendering. URL: https://nextjs.org/.

21. Library for data visualization Plotly. URL: https://plotly.com/.

22. Chart.js library for building graphs. URL: https://www.chartjs.org/.

23. CSS-framework Tailwind. URL: https://tailwindcss.com/.

24. Microsoft SQL Server (MSSQL) database management system. URL: https://learn.microsoft.com/en-us/sql/sql-server/.

25. PostgreSQL database management system. URL: https://www.postgresql.org/.

26. Document-oriented database MongoDB. URL: https://www.mongodb.com/.

27. Docker containerization platform. URL: https://www.docker.com/.

28. Interface for managing PostgreSQL pgAdmin. URL: https://www.pgadmin.org/

29. A graphical interface for managing Kafka Kafka UI. URL: https://provectus.io/open-source/kafka-ui/.

30. Git version control system. URL: https://git-scm.com/

31. Visual Studio Code code editor. URL: https://code.visualstudio.com/.

32. Zookeeper service coordination system. URL: https://zookeeper.apache.org/.

APPENDIX A

Program code

Link to the program code on GitHub:
https://github.com/chechmek/bachelors-thesis-air-monitoring

Figure A.1 - Link to the program code