DELIVERABLE

D4.4 Verification, Validation and Evaluation report 1

Project Acronym:	COMPAIR	
Project title:	Community Observation Measurement & Participation in AIR Science	
Grant Agreement No.	101036563	
Website:	www.wecompair.eu	
Version:	1.0	
Date:	30.06.2023	
Responsible Partner:	ATC	
Contributing Partners:	IMEC, UAEG, SODAQ, HHI	
Reviewers:	All technical partners Andrew Stott, Karel Jedlicka	
Dissemination Level:	Public	X
	Confidential, only for members of the consortium (including the Commission Services)	

This project has received financial support from the European Union's Horizon 2020 Programme under grant agreement no. 101036563

Revision History

Version	Date	Author	Organisation	Description
0.1	17.05.2023	Athanasios Dalianis, Marina Klitsi	ATC	Draft ToC
0.2	05.06.2023	Athanasios Dalianis	ATC	Input in Section 2.1
0.3	07.06.2023	Athanasios Dalianis	ATC	Input in Section 2.2
0.4	09.06.2023	Oliver Schreer, Sylvain Renault	HHI	Input for DEVA
0.5	13.06.2023	Christos Karelis	UAEG	Input in Section 2.3
0.6	13.06.2023	Athanasios Dalianis, Richard Brown, Jaap De Winter	ATC, SODAQ	Input in Section 2.2
0.7	15.06.2023	Athanasios Dalianis, Daniel Bertocci	ATC, IMEC	Input in Sections 2.2, 2.3
0.8	19.06.2023	Athanasios Dalianis	ATC	Version ready for internal review
0.9	22.06.2023	Andrew Stott, Burcu Celikkol, Gert Vervaet,	External, IMEC, DV	Version reviewed
1.0	28.06.2023	Athanasios Dalianis	ATC	Final version

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Acronym	Description
API	Application Programming Interface
AQ CaaS	Air Quality Calibration as a Service
AR	Augmented Reality
AWS	Amazon Web Services
CI/CD	Continuous Integration / Continuous Deployment
CSP	Content Security Policy
DEVA	Dynamic Exposure Visualisation App
FTP	File transfer Protocol
GUI	Graphical User Interface
GDPR	General Data Protection Regulation
HTTP(S)	Hypertext Transfer Protocol (Secure)
HSTS	Strict-Transport-Security
JDBC	Java Database Connectivity
JMS	Java Message Service
KPI	Key Performance Indicators
OWASP	Open Web Application Security Project
REST	Representational State Transfer
SOAP	Simple Objects Access Protocol
SQL	Structured Query Language
UI	User Interface
UX	User Experience
W3C	World Wide Web Consortium
WCAG	Web Content Accessibility Guidelines
XSS	Cross-Site Scripting

Executive Summary

This document presents a comprehensive analysis of the testing of the current **COMPAIR** prototype. The report focuses on applying the ISO/IEC 25010:2011 standard methodology to assess the quality characteristics of the software system. This methodology defines a set of quality characteristics and sub-characteristics that serve as a framework for evaluating the overall performance, reliability, security, usability, and other aspects of a software system.

Each quality characteristic is further broken down into sub-characteristics, which serve as specific attributes to assess. For instance, functionality encompasses aspects such as correctness, completeness, and interoperability, while usability focuses on factors like learnability, operability, and user error protection. By addressing each sub-characteristic, the deliverable provides a comprehensive evaluation of the software system's overall quality.

The report presents detailed findings from the testing activities, providing information about the tools and methodologies used and includes the outcomes of functional testing, performance testing, security testing, and usability testing. It highlights the strengths and weaknesses of the software system, identifies areas of concern, and provides insights for improving the system's performance and adherence to the ISO/IEC 25010:2011 quality characteristics.

1. Introduction

Purpose and Scope

The objective of this report is to provide an overview of the verification, validation, and evaluation activities performed for the second major release of the **COMPAIR** prototype, highlighting the tools used, and the key findings and observations discovered. By employing the ISO/IEC 25010:2011 standard methodology, the report aims to assess the software system's compliance with the defined quality characteristics and identify any areas that require improvement.

The testing process follows the guidelines set forth by ISO/IEC 25010:2011, which involves a systematic approach to evaluate the software system's functional and non-functional aspects. The standard encompasses several quality characteristics, including functionality, reliability, performance efficiency, security, usability, compatibility, maintainability, and portability.

Structure of the document

This deliverable is organised as follows:

- Section 2 presents the validation methodology, the evaluation tools used and testing results of the current prototype
- Section 3 concludes this deliverable

2. Assessing the COMPAIR solution

In this section, we provide an overview of the evaluation methodology and tools used for the monitoring and testing of the **COMPAIR** prototype, as well as the relevant results of these tests in various quality aspects like performance, security etc. These results depict the quality of the current platform prototype and provide a good first indication on the sections that the platform as a whole performs well or more effort is needed to reach the desired levels of quality.

2.1 Validation methodology

In this section, we provide an overview of the methodology used for the system validation of the platform. More specifically, we present the ISO/IEC 25010:2011[1] and explain its quality characteristics. Out of these characteristics, we will select the most appropriate ones, in order to form the most suitable quality model for the **COMPAIR** project and perform our validation tests to the final version of the **COMPAIR** platform.

Software validation is the "confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled". Since software is usually part of a larger system, the validation of software typically includes evidence that all software requirements have been implemented correctly and completely.

In general, software validation is the process of developing a "level of confidence" that the system meets all requirements, functionalities, and user expectations as set out during the design process. It is a critical tool used to assure the quality of its components and the overall system. It allows for improving/refining the final product.

2.1.1 ISO/IEC 25010:2011

Recently, the BS ISO/IEC 25010:2011 standard about system and software quality models has replaced ISO 9126-1. Applying any of these models is not a straightforward process. There are no automated means for testing software against each of the characteristics defined by each model. For each model, the final attributes must be matched against measurable metrics and thresholds for evaluating the results must be set. It is then possible to measure the results of the tests performed (either quantitative or qualitative/observed).

The ISO/IEC 25010:2011 standard is the most widespread reference model and includes the common software quality characteristics that are supported by the other models. This standard defines two quality models providing a consistent terminology for specifying, measuring and evaluating system and software product quality, as described below.

[1] https://www.iso.org/standard/35733.html

2.1.2 Quality in use model

The Quality in use model is composed of five characteristics that relate to the outcome of interaction with the system and characterises the impact that the product can have on the stakeholders. It addresses external quality, i.e. the quality of a (software) product as perceived by its users. External quality assesses the characteristics of the product quality model by black-box measurement.

2.1.3 Product quality model

The Product quality model is composed of eight characteristics that relate to static properties of software and dynamic properties of the computer system. It is intended to measure the internal quality, i.e., the quality of the software (and, particularly, its internal components) that eventually delivers external quality. Internal quality assesses the characteristics of the product quality model by glass-box measurement, i.e. measuring system properties based on knowledge about the internal structure of the software. For our case, the product quality model is adopted. The eight quality characteristics are further divided into sub-characteristics, as shown in the following figure:

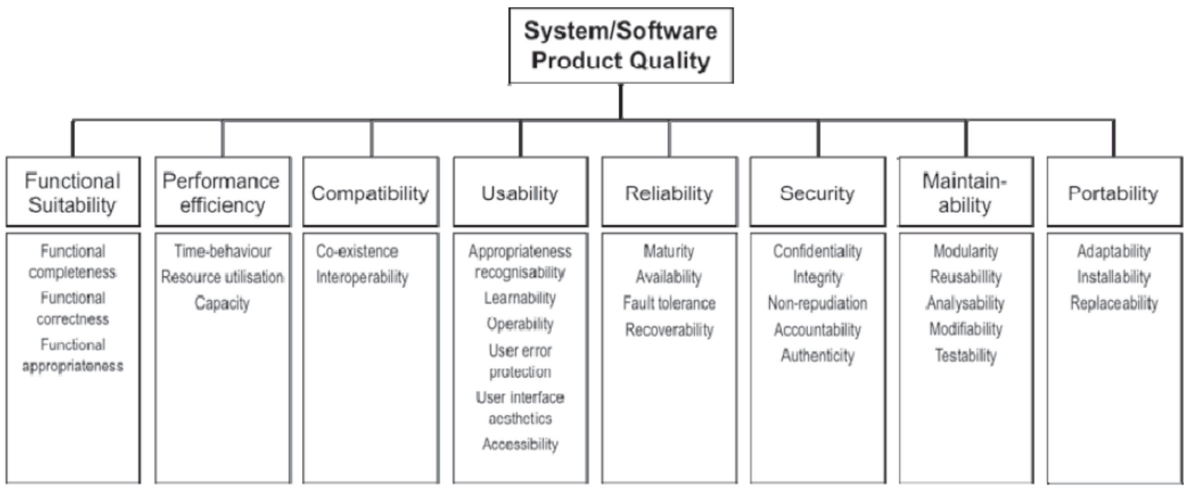


Figure 1: The ISO/IEC 25010:2011 system/software quality model characteristics

Although rather generic, not all the listed quality characteristics might be applicable for our purpose, so a tailor-made subset could be better suited. For each of the sub-characteristics, a metric/measurable attribute will be defined, along with thresholds. These metrics and thresholds are customised for each software product, which in our case is the **COMPAIR** platform (consisting of individual components). By evaluating these metrics, we will be able to assess the overall quality of our platform and the percent to which we were able to meet the user and technical requirements (reflected to system specifications and functionalities), defined during the design phase of the project.

A quality model is the cornerstone of a product quality system. It determines which quality characteristics will be considered when evaluating the properties of a software product.

The quality of a system is the degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value. Those stakeholders' needs are precisely what is represented in the quality model, which categorises the product quality into characteristics and sub-characteristics, as defined below.

2.1.4 Functional suitability

This characteristic represents the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions. This characteristic is composed of the following sub characteristics:

- Functional completeness - Degree to which the set of functions covers all the specified tasks and user objectives.

- Functional correctness - Degree to which a product or system provides the correct results with the needed degree of precision.

- Functional appropriateness - Degree to which the functions facilitate the accomplishment of specified tasks and objectives.

2.1.5 Performance efficiency

This characteristic represents the performance relative to the amount of resources used under stated conditions. This characteristic is composed of the following sub characteristics:

- Time behaviour - Degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements.
- Resource utilisation - Degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements.
- Capacity - Degree to which the maximum limits of a product or system parameter meet requirements.

In the context of the **DEVA** which is an interactive AR application developed in Unity3D, these performance characteristics are further refined:

- Frame rate is the number of frames per second that the AR application renders all the graphical elements on the device (3D rendering). A high frame rate is required to create a smooth and responsive AR experience. Here usually 15-20 frames per second are the lower limits.
- Latency is the time delay between user input and output. A low latency is essential for creating a realistic AR experience.
- Another important aspect is data consumption, if the app requires continuous access to a server and network connectivity. Frequent changes of AR parameters in **DEVA** and changes of the user's location will produce new queries to the Data Manager.

2.1.6 Compatibility

This is the degree to which a product, system or component can exchange information with other products, systems, or components, and/or perform its required functions, while sharing the same hardware or software environment. This characteristic is composed of the following sub characteristics:

- Co-existence - Degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product.

- Interoperability - Degree to which two or more systems, products or components can exchange information and use the information that has been exchanged.

2.1.7 Usability

This characteristic represents the degree to which a product or system can be used by specified users to achieve specific goals with effectiveness, efficiency, and satisfaction in a specified context of use. This characteristic is composed of the following sub characteristics:

- Appropriateness recognisability - Degree to which users can recognize whether a product or system is appropriate for their needs.

- Learnability - Degree to which a product or system can be used by specified users to achieve specific goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use.

- Operability - Degree to which a product or system has attributes that make it easy to operate and control.

- User error protection - Degree to which a system protects users against making errors.

- User interface aesthetics - Degree to which a user interface enables pleasing and satisfying interaction for the user.

- Accessibility - Degree to which a product or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use.

2.1.8 Security

This is the degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization. This characteristic is composed of the following sub characteristics:

- Confidentiality - Degree to which a product or system ensures that data are accessible only to those authorised to have access.

- Integrity - Degree to which a system, product or component prevents unauthorised access to, or modification of, computer programs or data.

- Non-repudiation - Degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later.

- Accountability - Degree to which the actions of an entity can be traced uniquely to the entity.

- Authenticity - Degree to which the identity of a subject or resource can be proved to be the one claimed.

2.1.9 Maintainability

This characteristic represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements. This characteristic is composed of the following sub characteristics:

- Modularity -. Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

- Reusability - Degree to which an asset can be used in more than one system, or in building other assets.

- Analysability - Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified.

- Modifiability - Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.

- Testability - Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met.

2.1.10 Reliability

This is the degree to which a system, product or component performs specific functions under specified conditions for a certain period. This characteristic is composed of the following sub characteristics:

- Maturity - Degree to which a system, product or component meets needs for reliability under normal operation.

- Availability - Degree to which a system, product or component is operational and accessible when required for use.

- Fault tolerance - Degree to which a system, product or component operates as intended despite the presence of hardware or software faults.
- Recoverability - Degree to which in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system.

2.1.11 Portability

Portability is the degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another. This characteristic is composed of the following sub characteristics:

- Adaptability - Degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments.
- Installability - Degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment.
- Replaceability - Degree to which a product can replace another specified software product for the same purpose in the same environment.

2.2 Overview of evaluation tools

This section presents an overview of the tools that are used for monitoring, testing and validating the **COMPAIR** solution. These tools are summarised in the following table and are described in more detail below.

Table 1:Overview of evaluation tools

Category	Tool
Performance monitoring	Prometheus, Databricks, AWS
Performance testing	Page Speed Insights, Unity Performance testing tools, JMeter
Functional suitability	SonarQube, JUnit, Mockito
Usability	Matomo, Plausible, Page Speed Insights
Security	SonarQube, SecurityHeaders, OWASP Zap
Maintainability	SonarQube
Reliability	SonarQube

As presented in the table above, in several of the quality characteristics, in order to have a more reliable and globally accepted measure of code quality, the popular SonarQube [1] quality gateway was used.

SonarQube is an open-source platform developed for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities on more than 20 programming languages. SonarQube offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security vulnerabilities.

Key features of SonarQube include:

- Static Code Analysis: SonarQube analyses source code to identify security vulnerabilities, coding best practices, and maintainability issues. It provides actionable insights and detailed reports to help developers address identified issues.
- Extensive Language Support: SonarQube supports a wide range of programming languages, including Java, C#, JavaScript, Python, PHP, and many more. This makes it applicable to a variety of software projects.
- Integration with Development Process: SonarQube integrates with popular development tools and workflows, such as IDEs, build systems, and CI/CD pipelines. It provides detailed reports and metrics that can be incorporated into the software development process to ensure code quality and security throughout.
- Custom Rules and Quality Gates: SonarQube allows users to define custom rules and quality gates to enforce specific coding standards and security practices within their projects.

SonarQube in general helps organisations to maintain code quality, identify security vulnerabilities, and track code improvements over time. It is widely used by development teams to ensure the long-term maintainability and security of their codebases.

For the purposes of the project, a SonarQube server was deployed in the Data platform of the project and can be found at https://monitor.wecompair.eu/sonarqube/. The code deployed in SonarQube, involves software components that were developed for the project.

Performance efficiency

Software component efficiency refers to the measure of how effectively and optimally individual software components perform their intended functions within a larger system or application. It encompasses various aspects, including resource utilisation, computational speed, memory usage, and overall performance. Maximising component efficiency is crucial for ensuring optimal system performance, minimising resource wastage, and delivering a seamless user experience.

Monitoring and optimising software component efficiency are essential for maintaining system performance and scalability. It involves analysing performance metrics and profiling techniques to identify bottlenecks, optimise algorithms, and fine-tune configuration settings. By continuously monitoring and improving component efficiency, development teams can ensure that resources are utilised efficiently, minimising operational costs and maximising the system's capacity to handle increased workloads.

In this section, we consider tools both for monitoring the performance of the system in real time and for testing the performance off line.

Monitoring

Software component monitoring refers to the practice of closely observing and tracking the performance, health, and behaviour of individual software components within a larger system or application. In modern software architectures, applications are often composed of numerous interconnected components, such as microservices, libraries, databases, and third-party integrations. Monitoring these components is essential for ensuring optimal system performance, identifying issues or bottlenecks, and maintaining a high level of reliability.

The primary goal of software component monitoring is to gain real-time insights into the behaviour and performance of individual software components, as well as their interactions with other components. This involves capturing and analysing relevant data, such as resource utilisation, response times, error rates, and other key performance indicators (KPIs). By monitoring these metrics, development and operations teams can proactively detect and address potential issues, optimise system performance, and ensure a seamless user experience.

Monitoring software components often involves the use of specialised tools and technologies that collect and analyse data from various sources, including logs, metrics, traces, and events. These tools provide visualisations, alerts, and dashboards that allow teams to monitor the health of individual components, identify patterns and anomalies, and diagnose the root causes of performance problems or failures. With comprehensive component monitoring in place, organisations can improve the overall stability, scalability, and resilience of their software systems, leading to enhanced user satisfaction and better business outcomes.

Several monitoring tools have been deployed in the physical nodes that comprise the **COMPAIR** solution, as described in D4.1 Solution architecture report, in order to keep track of the user activity and the good operation of the **COMPAIR** components and sensors in real time.

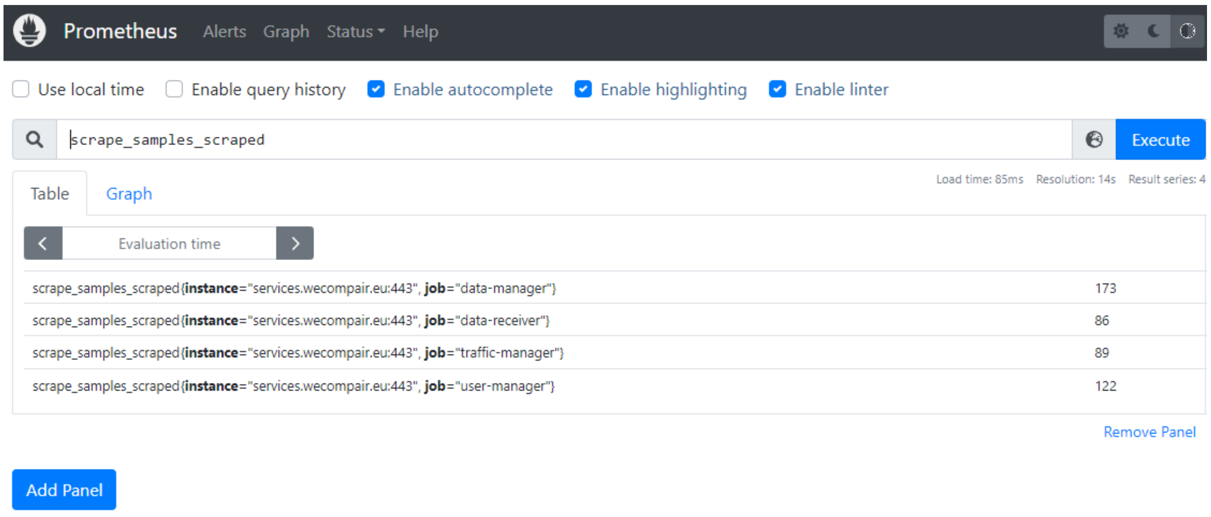
These monitoring tools are complemented by custom components that monitor the communication between data providers and data receivers and react upon the lack of data. For example, if the Data Manager (deployed in the Data Platform) does not receive traffic data from the Traffic Sensor Platform in a specific time interval, the incident is recorded and the relevant technical partner is notified for further investigation.
The monitoring tools used in every **COMPAIR** node for tracking performance are described below.

Prometheus and Grafana

Prometheus [2] is an open-source tool under Apache Licence, used for event monitoring and alerting. It records real time metrics and stores them in a time series database. It features functionalities like distributed storage, multiple nodes of graphing and dashboarding support and can collaborate with a wide range of tools like Docker, Kubernetes and Grafana. Although Prometheus has a user interface of its own (Figure 2), this cannot be used easily by novice users, so a tool like Grafana is usually suggested in combination.

Grafana [3] is an open source and extendable analytics and interactive visualisation web application that allows a user to query and visualise data, through a set of charts, graphs and alerts, no matter where this data is stored, e.g. in a database, Prometheus etc.

For the purposes of the project, Prometheus and Grafana are installed in the Data Platform and monitor the components and APIs deployed there, under https://monitor.wecompair.eu/prometheus/ and https://monitor.wecompair.eu/grafana/ respectively. Currently the health and performance of the APIs are monitored in real time, that is, whether the APIs are up and running, if the consumption of memory and CPU is high, the API's response times, etc. (figure 3).

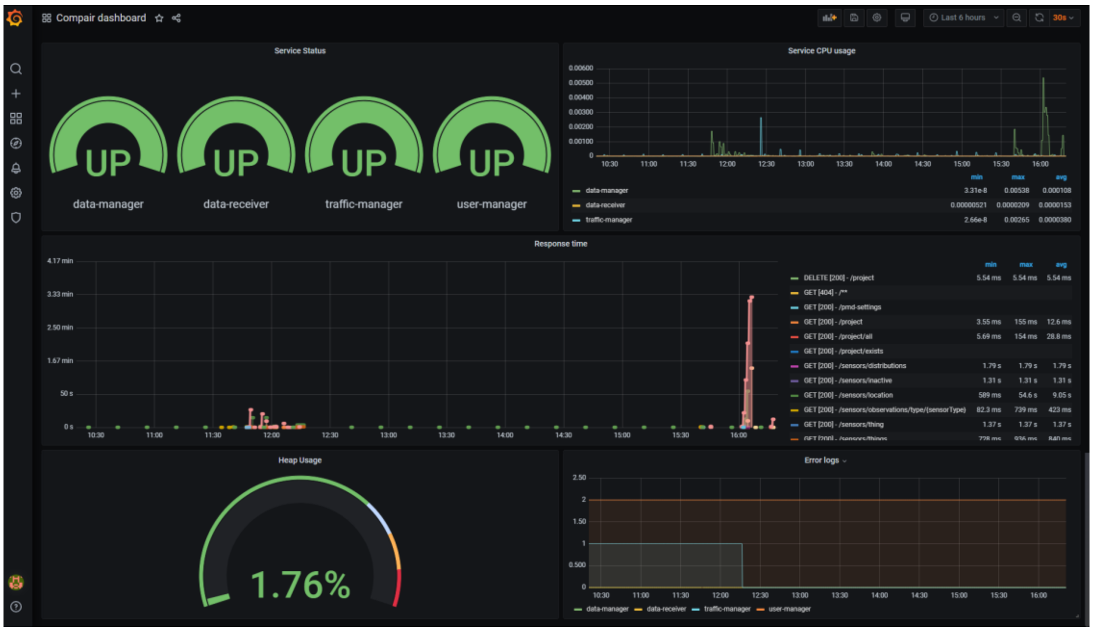


```
scrape_samples_scraped
```

Load time: 85ms   Resolution: 14s   Result series: 4

scrape_samples_scraped{**instance**="services.wecompair.eu:443", **job**="data-manager"}	173
scrape_samples_scraped{**instance**="services.wecompair.eu:443", **job**="data-receiver"}	86
scrape_samples_scraped{**instance**="services.wecompair.eu:443", **job**="traffic-manager"}	89
scrape_samples_scraped{**instance**="services.wecompair.eu:443", **job**="user-manager"}	122

Figure 2:Prometheus dashboard



Service CPU usage

	min	max	avg
data-manager	3.31e-8	0.00538	0.000108
data-receiver	0.00000521	0.0000209	0.0000153
traffic-manager	2.66e-8	0.00265	0.0000380

Response time

	min	max	avg
DELETE [200] - /project	5.54 ms	5.54 ms	5.54 ms
GET [404] - /**			
GET [200] - /pmd-settings			
GET [200] - /project	3.55 ms	155 ms	12.6 ms
GET [200] - /project/all	5.69 ms	154 ms	28.8 ms
GET [200] - /project/exists			
GET [200] - /sensors/distributions	1.79 s	1.79 s	1.79 s
GET [200] - /sensors/inactive	1.31 s	1.31 s	1.31 s
GET [200] - /sensors/location	589 ms	54.6 s	9.05 s
GET [200] - /sensors/observations/type/{sensorType}	82.3 ms	739 ms	423 ms
GET [200] - /sensors/thing	1.37 s	1.37 s	1.37 s
GET [200] - /sensors/things	728 ms	936 ms	840 ms

Figure 3: Grafana dashboard for CompAir

The AQ CaaS Platform is based on Spark jobs that perform the data collection and processing. These jobs are configured in the Databricks [4] tool, which is a powerful cloud-based data engineering and data science platform that simplifies the process of building, managing, and scaling big data and artificial intelligence (AI) applications. It is designed to provide an integrated environment for data processing, analytics, and machine learning, however it also provides monitoring capabilities to help track the performance, health, and resource utilisation of the data processing and analytics workloads. These monitoring features allow an administrator to gain insights into the behaviour and efficiency of the Databricks clusters and jobs.

Some key monitoring operations that Databricks supports include:

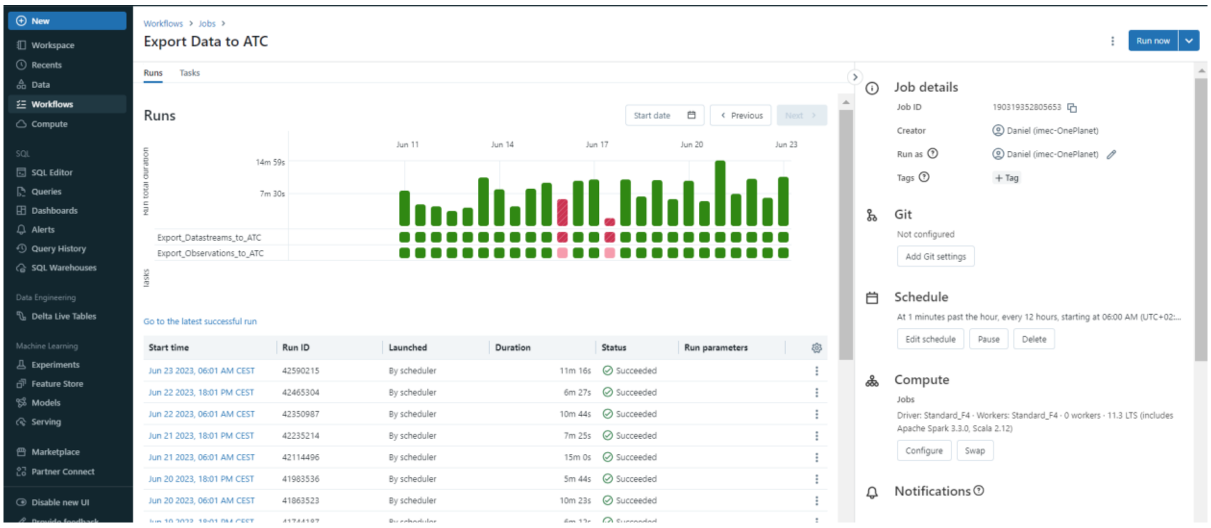
- Cluster Monitoring: Databricks provides a Cluster UI that allows the monitoring of the performance and resource utilisation of the Spark clusters, providing information for metrics such as CPU and memory usage, disk I/O, network activity, and job execution statistics. This information helps to understand the workload patterns and identify potential bottlenecks or performance issues.
- Job Monitoring: Databricks allows the monitor of the execution of Spark jobs running on the clusters. Someone can track job progress, view detailed logs, and monitor job-specific metrics such as task duration, data shuffle operations, and resource consumption. Monitoring job performance helps in optimising job execution and identifying any errors or issues during the process.
- Alerting and Notifications: Databricks supports alerting and notification mechanisms to keep informed about important events and conditions. An admin can configure alerts based on specific metrics thresholds or job status changes. When an alert condition is triggered, the assigned person receives notifications via email or other channels, enabling proactive monitoring and timely response to critical situations.
- Integration with External Monitoring Tools: Databricks can integrate with external monitoring and logging systems such as Azure Monitor, Amazon CloudWatch, and Datadog. This integration allows an admin to centralise the monitoring efforts and leverage existing tools and workflows for data platform monitoring.

By monitoring the Databricks jobs, an admin can gain valuable insights into their performance, identify optimization opportunities, and troubleshoot any issues that arise. Effective monitoring helps ensure the smooth functioning of the data processing workloads and helps in making informed decisions regarding resource allocation and workload management.

In the context of **COMPAIR**, the jobs are used mainly as pipeline elements to:

- Ingest data from different sources (e.g. Discomap and Sensor.Community)
- Transform the custom format they use into OGC SensorThings API standard
- Forward data to Data Manager
- Store in a separate storage for calibration purposes

An example of these jobs is presented in Figure 4.



Export Data to ATC

Runs

Start time	Run ID	Launched	Duration	Status	Run parameters
Jun 23 2023, 06:01 AM CEST	42590215	By scheduler	11m 16s	Succeeded	
Jun 22 2023, 18:01 PM CEST	42465304	By scheduler	6m 27s	Succeeded	
Jun 22 2023, 06:01 AM CEST	42350987	By scheduler	10m 44s	Succeeded	
Jun 21 2023, 18:01 PM CEST	42235214	By scheduler	7m 25s	Succeeded	
Jun 21 2023, 06:01 AM CEST	42114496	By scheduler	15m 0s	Succeeded	
Jun 20 2023, 18:01 PM CEST	41983536	By scheduler	5m 44s	Succeeded	
Jun 20 2023, 06:01 AM CEST	41863523	By scheduler	10m 23s	Succeeded	

Job details

Job ID	190319352805653
Creator	Daniel (imec-OnePlanet)
Run as	Daniel (imec-OnePlanet)
Tags	+ Tag

Git

Not configured

Add Git settings

Schedule

At 1 minutes past the hour, every 12 hours, starting at 06:00 AM (UTC+02:...

Edit schedule | Pause | Delete

Compute

Jobs

Driver: Standard_F4 · Workers: Standard_F4 · 0 workers · 11.3 LTS (includes Apache Spark 3.3.0, Scala 2.12)

Configure | Swap

Notifications

Figure 4: CaaS Databricks

The Traffic Sensor Platform (run by Telraam) utilises a custom web application [5] to provide sensor information to visitors and to allow the administrators to monitor the health of the sensors. This “Telraam network management dashboard” clusters devices & users at a (sub-project level). It is governed by a local project lead. Functionalities include:

- Status of user in the installation process: selected/device received/device installed/device inactive/etc)
- Telemetry data of the sensor: timestamp of sensor events such as reboots, display on/off input, data connection signal strength, firmware version, user action log, etc.
- Thumbnail snapshot: low resolution picture of the view on the street, to verify correct installation of the sensor

Below we present three screenshots of the dashboard:

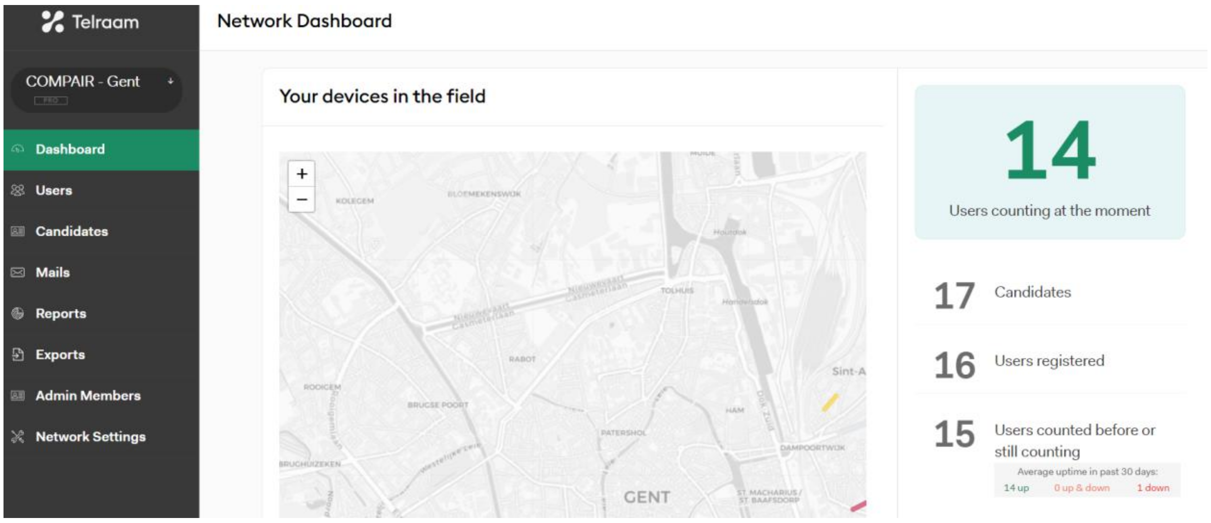
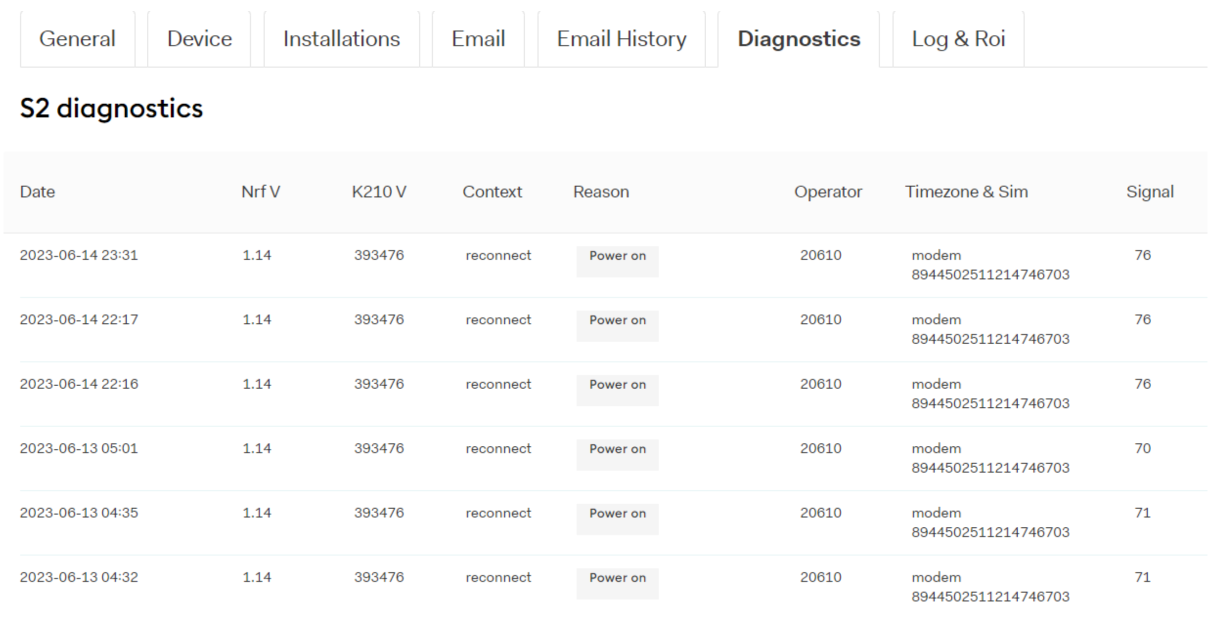


Figure 5:Telraam Network dashboard

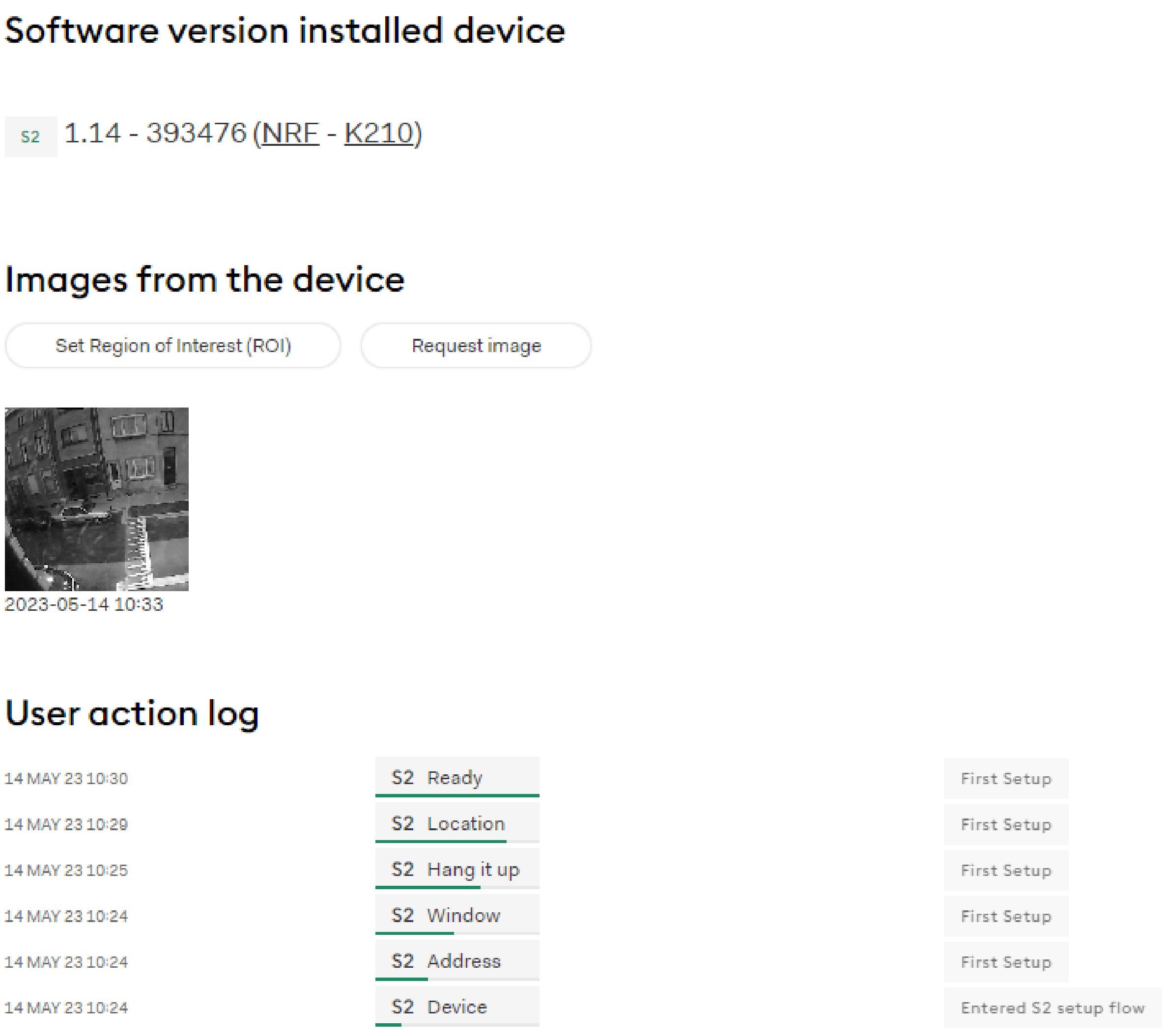


General | Device | Installations | Email | Email History | **Diagnostics** | Log & Roi

S2 diagnostics

Date	Nrf V	K210 V	Context	Reason	Operator	Timezone & Sim	Signal
2023-06-14 23:31	1.14	393476	reconnect	Power on	20610	modem 8944502511214746703	76
2023-06-14 22:17	1.14	393476	reconnect	Power on	20610	modem 8944502511214746703	76
2023-06-14 22:16	1.14	393476	reconnect	Power on	20610	modem 8944502511214746703	76
2023-06-13 05:01	1.14	393476	reconnect	Power on	20610	modem 8944502511214746703	70
2023-06-13 04:35	1.14	393476	reconnect	Power on	20610	modem 8944502511214746703	71
2023-06-13 04:32	1.14	393476	reconnect	Power on	20610	modem 8944502511214746703	71

Figure 6:Telraam Diagnostics



Software version installed device

S2 1.14 - 393476 (NRF - K210)

Images from the device

Set Region of Interest (ROI) | Request image

2023-05-14 10:33

User action log

14 MAY 23 10:30	S2 Ready	First Setup
14 MAY 23 10:29	S2 Location	First Setup
14 MAY 23 10:25	S2 Hang it up	First Setup
14 MAY 23 10:24	S2 Window	First Setup
14 MAY 23 10:24	S2 Address	First Setup
14 MAY 23 10:24	S2 Device	Entered S2 setup flow

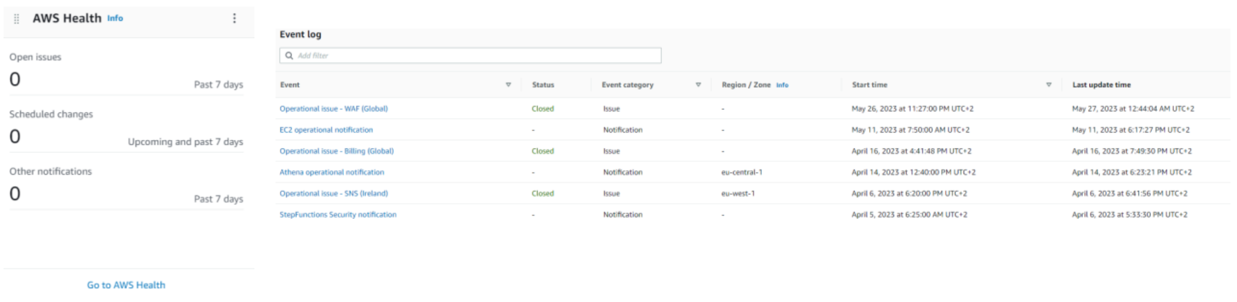
Figure 7:Sensor Data

For health monitoring of the internal components and databases of the Traffic Sensor Platform, the AWS Health tool [6] is used. AWS Health is a service provided by Amazon Web Services (AWS) that offers insights and notifications regarding the operational health and performance of AWS resources and services. It provides real-time information, proactive notifications, and remediation guidance to help users maintain the availability, security, and performance of their AWS infrastructure.

AWS Health monitors the status of various AWS services and regions and provides a centralised dashboard where users can view the overall health status of their resources. It offers several key features:

1. Personalised Dashboards: AWS Health provides personalised views of the health status and events specific to a user's AWS account. It highlights ongoing and historical events, proactive recommendations, and maintenance notifications.
2. Event Notifications: Users can configure AWS Health to send proactive notifications via email, SMS, or other communication channels when there are service disruptions or other events that may impact their resources. In the case of Telraam, we monitor load on Lambda functions from data-ingestion (API-gateway) to database.
3. Event History and Analysis: The service maintains an event history that users can reference to understand past incidents, their duration, and the impact on their resources. This allows for analysis and post-incident evaluations.
4. Trusted Advisor Integration: AWS Health is integrated with AWS Trusted Advisor, which provides automated recommendations for optimising AWS resources and costs. Users can access Trusted Advisor recommendations directly from the AWS Health console.
5. API Integration: AWS Health provides an API that allows users to programmatically retrieve information about the health status of their resources. This enables integration with custom monitoring and notification systems.

AWS Health is particularly useful for organisations that rely on AWS services for their infrastructure. It helps users stay informed about the status of their resources, receive timely notifications about service disruptions or maintenance events, and take proactive measures to mitigate any impact on their applications or business operations. By leveraging AWS Health, users can enhance their operational awareness, reduce downtime, and improve the overall resilience of their AWS deployments.



Open issues	0	Past 7 days
Scheduled changes	0	Upcoming and past 7 days
Other notifications	0	Past 7 days

Event log

Event	Status	Event category	Region / Zone	Start time	Last update time
Operational issue - WAF (Global)	Closed	Issue	-	May 26, 2023 at 11:27:00 PM UTC+2	May 27, 2023 at 12:44:04 AM UTC+2
EC2 operational notification	-	Notification	-	May 11, 2023 at 7:50:00 AM UTC+2	May 11, 2023 at 6:17:27 PM UTC+2
Operational issue - Billing (Global)	Closed	Issue	-	April 16, 2023 at 4:41:48 PM UTC+2	April 16, 2023 at 7:49:30 PM UTC+2
Athena operational notification	-	Notification	eu-central-1	April 14, 2023 at 12:40:00 PM UTC+2	April 14, 2023 at 6:23:21 PM UTC+2
Operational issue - SNS (Ireland)	Closed	Issue	eu-west-1	April 6, 2023 at 6:20:00 PM UTC+2	April 6, 2023 at 6:41:56 PM UTC+2
StepFunctions Security notification	-	Notification	-	April 5, 2023 at 6:25:00 AM UTC+2	April 6, 2023 at 5:33:30 PM UTC+2

Figure 8: Telraam AWS Health

The SODAQ AIR is a portable air quality monitor designed for bicycle users that provides real-time, location-specific air quality insights. Equipped with an array of sensors, the AIR provides in-depth readings on particulate matter (PM) concentrations, temperature, and humidity creating a holistic representation of the air quality.

To anyone interested in air quality, the Know Your AIR Map provides a global perspective on air quality. The aggregation of all the data in a 100×100 metre area, is averaged on an hourly basis and uploaded to the global map as open data. This creates a network of detailed air measurements across the globe while protecting user anonymity. All of the collective air quality measurements are shared as open-source data giving everyone, everywhere access.

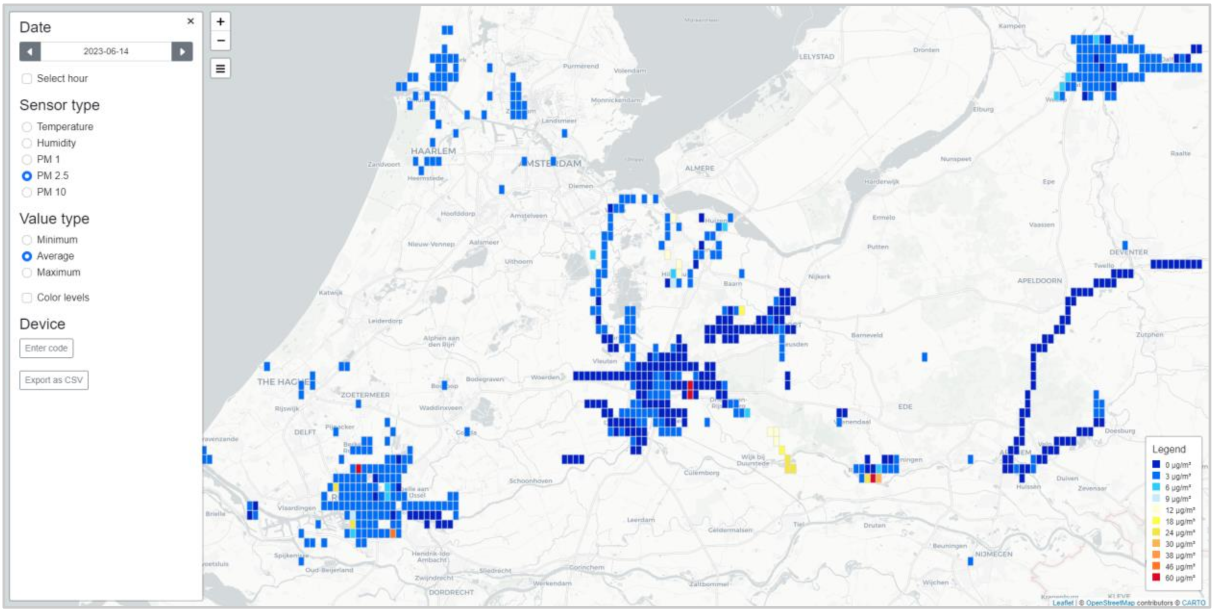


Figure 9: Know Your Air Map

In addition to that, users of the SODAQ Air can see the air quality of their own routes by inputting with their device's unique identifier code. This enables the users to track the air quality during their bicycle trips and to consider taking alternative routes to reduce their exposure to polluted air.

The third option, which is provided, is to forward all the data generated by the AIR devices to (generally larger) customers. Within COMPAIR SODAQ is forwarding the SODAQ AIR data from each of the pilots to the COMPAIR platform for data interpretation and representation.

Just like Telraam, SODAQ runs its cloud services in an AWS environment. For more information on cloud services in AWS, reference is made to the description (above) about Telraam. The data from the individual Air sensors is received on a Node Red server in the SODAQ AWS environment. From the Node Red server the data is forwarded both to the knowyourair platform and to the CompAir server, as represented below.

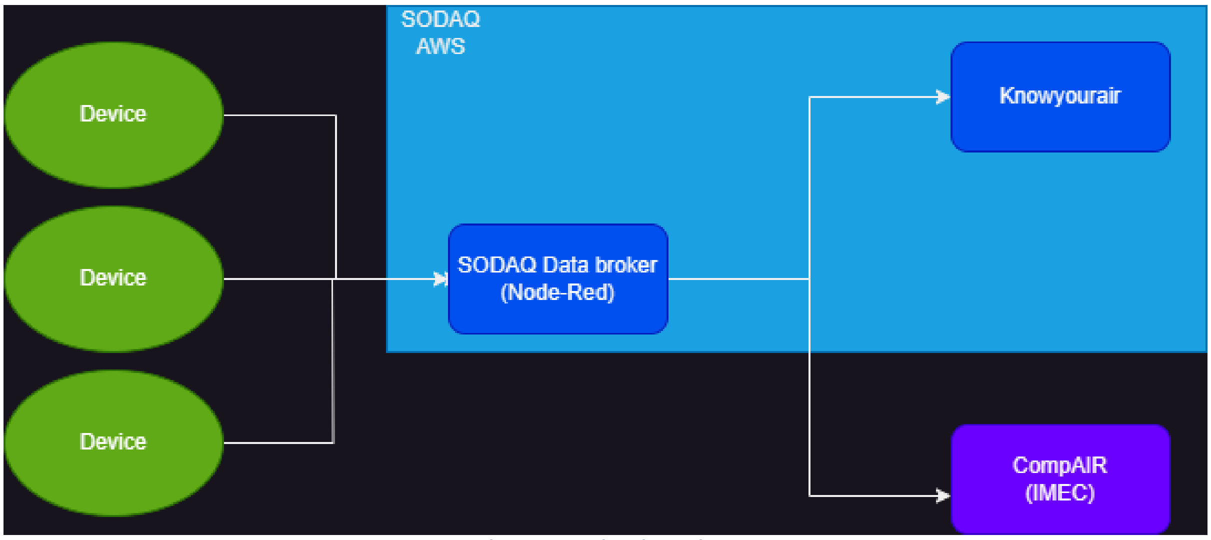


Figure 10: cloud services

As an additional service, the knowyourair platform allows users and user groups to check the status of their devices. All CompAir devices are part of a subgroup and can thus be selected and monitored: https://knowyourair.net/reports/status/ Based on the device IMEI, a quick individual health check can be done: https://knowyourair.net/device/status/

Performance testing tools

Page Speed Insights

Page Speed Insights [7] is a tool developed by Google to assess and provide insights into the performance and speed of web pages. It analyses web pages and generates reports with suggestions and recommendations to improve their speed and overall user experience.

Page Speed Insights evaluates the performance of a web page on both mobile and desktop devices. It measures various aspects that contribute to page load times, including server response times, render-blocking resources, image optimization, CSS and JavaScript efficiency, and caching practices.

The tool assigns a score to each page, ranging from zero to 100, indicating its performance level. A higher score signifies better performance and faster page load times. Additionally, Page Speed Insights provides specific recommendations on how to optimise the page, highlighting areas that need improvement.

The insights provided by Page Speed Insights can help website owners and developers identify performance bottlenecks and implement optimizations to enhance the speed and responsiveness of their web pages. By following the recommendations, such as minimising file sizes, leveraging browser caching, or eliminating render-blocking resources, websites can provide a smoother user experience, reduce bounce rates, and potentially improve search engine rankings.

Page Speed Insights is a widely used tool by web developers, SEO professionals, and website administrators to measure and enhance the performance of their web pages. Its goal is to encourage the development of faster, more efficient websites that provide a better user experience across different devices.

In the context of the project, Page Speed Insights is used to test the performance of the Dashboards developed so far and their accessibility degree.

JMeter

JMeter [8] is an open-source performance-testing tool developed by the Apache Software Foundation. It is designed to load test functional behaviour and measure the performance of web applications, APIs, and other types of software systems.

Some key features and aspects of JMeter are:

1. Load Testing: JMeter allows testers to simulate high user loads by creating virtual users (threads) that send requests to the target system. It can generate large amounts of concurrent requests, helping assess how the system performs under different levels of load.
2. Protocol Support: JMeter supports a wide range of protocols, including HTTP, HTTPS, FTP, SOAP, REST, JDBC, JMS, and more. It can be used to test various types of systems, including web applications, databases, message queues, and web services.
3. Test Plan Creation: JMeter uses a graphical user interface (GUI) to create and configure test plans. Testers can define scenarios, set up thread groups, specify requests and parameters, and add assertions to validate responses. The tool also provides scripting capabilities for complex scenarios.
4. Test Execution and Monitoring: Once the test plan is set up, JMeter can execute the load test and monitor various performance metrics in real-time. It captures response times, throughput, error rates, and other key performance indicators. Testers can view the results in various formats, including graphs, tables, and reports.
5. Distributed Testing: JMeter supports distributed testing, allowing multiple JMeter instances to work together and generate high loads from different machines. This enables testers to simulate realistic scenarios with a distributed user base.
6. Extensibility: JMeter offers extensibility through its plugin architecture. Users can extend its functionality by installing additional plugins that provide features such as additional listeners, samplers, and scripting capabilities.
7. Integration with CI/CD: JMeter integrates well with continuous integration and delivery (CI/CD) pipelines. It can be automated and incorporated into the testing phase of the software development lifecycle, enabling performance testing as part of the deployment process.

Overall, JMeter is a powerful tool for load testing and measuring the performance of software systems. Its flexibility, protocol support, and extensibility make it suitable for a wide range of testing scenarios. Whether you need to assess the performance of a web application, API, or other types of systems, JMeter provides a robust solution for load testing and performance analysis.

In the context of the project, JMeter is used to measure the performance of the APIs under certain conditions, e.g. certain number of simultaneous users.

Functional suitability

To ensure functional suitability, organisations employ various testing techniques and tools. In the context of the project, we utilise unit and integration tests.

Unit tests and integration tests are two distinct types of tests used in software development to validate the functionality and behaviour of different components or units of code.

Unit tests focus on testing individual units or components of code in isolation. These units are typically small and represent a specific functionality or behaviour within the software. Unit tests help ensure that each unit of code behaves as expected and meets its functional requirements. By testing units in isolation, developers can easily identify and fix bugs or issues at an early stage.

Integration tests, on the other hand, aim to test the interaction and integration between multiple units or components of code. They validate that these components work correctly together, exchanging data and communicating as intended. Integration tests help uncover issues that may arise when different parts of the system are integrated and can help identify problems related to data flow, API interactions, or dependencies between components.

Several frameworks exist that realise the unit and integration tests, depending on the programming language used for the developed component.

JUnit

JUnit [9] is a popular testing framework for Java that is widely used for writing and executing unit tests. It provides a set of annotations, assertions, and APIs that make it easy to define and run test cases. JUnit allows developers to create test methods that verify the behaviour and output of individual units of code. It provides features for test setup and teardown, test fixture management, and reporting of test results.

Mockito

Mockito [10], on the other hand, is a mocking framework for Java that is often used in conjunction with JUnit. Mockito helps create mock objects that simulate the behaviour of dependencies or collaborators in unit tests. By using Mockito, developers can isolate the unit under test and control the behaviour of external dependencies, making it easier to write focused and deterministic unit tests. Mockito provides features for defining mock objects, setting expectations, and verifying interactions with these objects during testing.

Both JUnit and Mockito are powerful tools in the Java ecosystem for writing effective unit tests. They contribute to the overall quality and reliability of the codebase by allowing developers to validate individual units and their interactions, catch defects early, and provide a safety net for refactoring of code changes.

Security

Software security refers to the practice of protecting software systems and applications from unauthorised access, data breaches, vulnerabilities, and other malicious activities. It involves implementing measures to prevent, detect, and mitigate security risks throughout the software development lifecycle. By prioritising security, organisations can safeguard sensitive data, maintain user trust, and reduce the likelihood of security incidents that can result in financial losses or reputational damage.

So far, three main tools have been used to evaluate the security of the system, SonarQube to evaluate the security of the code, OWASP Zap [11] to evaluate the security of the dashboards and APIs and security headers to have a quick indication on the security of the web applications against specific attacks.

OWASP Zap

OWASP Zap is a free and open-source web application security scanner and penetration testing tool, provided by the Open Web Application Security Project (OWASP) which is a nonprofit foundation dedicated to improving software security. It helps identify vulnerabilities and security issues in web applications. Zap can be used by both developers and security professionals to perform various security testing activities, including:

- Automated Scanning: Zap can scan web applications to detect common vulnerabilities like cross-site scripting (XSS), SQL injection, broken authentication, insecure direct object references, and more.
- Manual Testing: It provides a user-friendly interface for manually testing and exploring web applications, enabling users to intercept and modify requests, analyse responses, and manipulate application parameters.
- Active and Passive Scanning: Zap can actively scan web applications by automatically sending requests and analysing responses. It can also passively monitor traffic to identify potential security issues.
- Scripting and Automation: Zap supports scripting and automation through its powerful API, allowing users to create custom tests and integrate Zap into their continuous integration (CI) pipelines.

OWASP Zap is well-documented, actively maintained by the OWASP community, and regularly updated with new security checks and features. It is a versatile tool that helps identify and address security vulnerabilities in web applications.

Security Headers

The Security Headers tool [12] is an online service that allows you to analyse the security headers implemented on a website. Security headers are HTTP response headers that provide additional security measures to protect web applications against various types of attacks. This tool helps in assessing the security posture of a website by examining the presence and configuration of these security headers.

Some key features of the tool include:

- Analysis of Security Headers: The tool performs a scan of the website's HTTP response headers and identifies the presence and configuration of important security headers such as Content Security Policy (CSP), X-XSS-Protection, X-Frame-Options, Strict-Transport-Security (HSTS), and more. It checks if the headers are properly set up, correctly configured, and follow security best practices.
- Security Recommendations: The tool provides recommendations and guidance on improving the security of the website based on the analysis of the security headers. It highlights any missing or misconfigured headers and suggests appropriate configurations to enhance the security posture. This helps website owners and developers take proactive measures to strengthen their website's security.
- User-Friendly Reporting: The Security Headers tool generates a detailed report that summarises the findings of the analysis. The report includes information on each security header, its configuration status, and any recommendations for improvement. This makes it easy to understand and communicate the security status of the website to stakeholders, developers, or security teams.

Using the Security Headers tool can help website owners and developers identify and address potential security vulnerabilities in their web applications. By ensuring proper configuration and implementation of security headers, they can mitigate risks associated with attacks such as cross-site scripting (XSS), clickjacking, content injection, and session hijacking.

Usability

An important aspect of software usability is the user interface (UI) design. A well-designed UI considers user behaviour, cognitive abilities, and task requirements to create a visually appealing and intuitive interface. It involves elements such as layout, navigation, controls, feedback mechanisms, and information organisation to enable users to interact with the software seamlessly. To this end, UX heuristics and accessibility rules are widely used.

UX heuristics, also known as usability heuristics or Nielsen's heuristics, are a set of broad guidelines or principles that are used to evaluate and assess the usability of user interfaces. They were first introduced by renowned usability expert Jakob Nielsen and have become widely adopted in the field of user experience design.

Heuristics serve as a framework for identifying potential usability issues and areas for improvement within a user interface. They provide a set of criteria that designers and evaluators can apply to assess the overall usability and user-friendliness of a product or system.

To ensure that web content is accessible to a wide range of people, including those with disabilities, the World Wide Web Consortium (W3C) has issued a series of internationally recognised guidelines with the name Web Content Accessibility Guidelines 2.1 (WCAG 2.1).

The primary goal of WCAG 2.1 [13] is to make web content perceivable, operable, understandable, and robust for all users, including those with visual, auditory, physical, cognitive, and neurological disabilities. It provides technical standards and guidance for web developers, designers, content authors, and others involved in creating and maintaining web content.

Usability testing is a common practice in evaluating and improving software usability. It involves observing real users as they perform tasks with the software and collecting feedback to identify areas of improvement. Usability testing helps uncover usability issues, such as confusing workflows, unclear instructions, or inefficient processes, which can then be addressed to enhance the overall usability of the software.

It is worth noting that automated tools for evaluating the overall usability of a web application based on UX heuristics alone can be challenging to find. Usability evaluation often requires a combination of techniques, including expert reviews, usability testing, user feedback and user monitoring, to gain a comprehensive understanding of the usability of a web application.

In the context of the project, we are following the UX heuristics and accessibility guidelines during the design and implementation of the Dashboards and we are using tools for testing the accessibility compliance and also for user behaviour monitoring. More precisely, we are using ColorBlindly [14] for making sure that the UIs are accessible to people with some form of colorblindness, PageSpeed Insights to measure the degree of accessibility compliance and Matomo [15] for monitoring the user behaviour in the project's Dashboards and DEVA.

Colorblindly

Colorblindly is a Chrome extension that helps developers create websites for the people with colorblindness by allowing them to simulate the experience those users have on websites.

There are eight different settings to experience based on the different types of colour blindness:

- Blue Cone Monochromacy / Achromatomaly
- Monochromacy / Achromatopsia
- Green-Weak / Deuteranomaly
- Green-Blind / Deuteranopia
- Red-Weak / Protanomaly
- Red-Blind / Protanopia
- Blue-Weak / Tritanomaly
- Blue-Blind / Tritanopia

PageSpeed Insights

For measuring the accessibility of a web site, PageSpeed Insights uses behind the scenes, a tool called Lighthouse [16] also developed by Google. Lighthouse is an automated testing tool built into the Chrome browser's DevTools. While it is primarily designed for performance and best practices audits, it also includes accessibility testing capabilities based on WCAG 2.1 guidelines. Lighthouse generates an accessibility report that outlines areas of improvement and provides suggestions for addressing accessibility issues.

Matomo

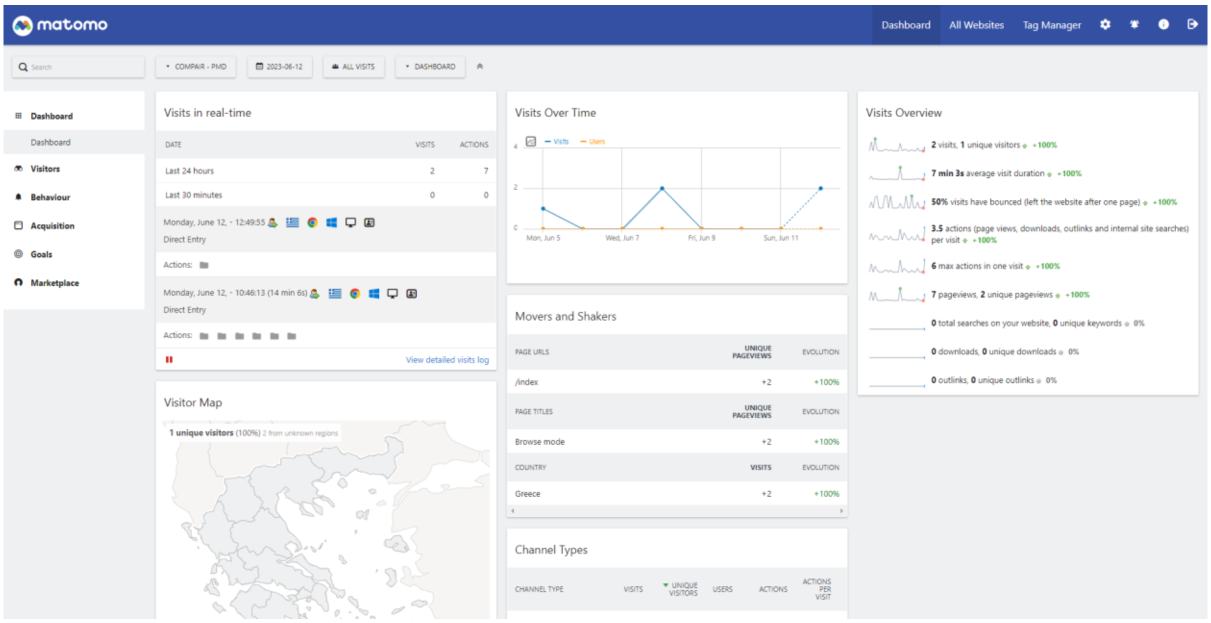
Matomo is a free open source web analytics application similar to Google Analytics. The key difference is that the application is provided as a Docker image and thus can be easily installed to a server keeping the data private within the consortium, allowing for better application of the GDPR rules.

Matomo can be used for monitoring the user actions in web sites or mobile devices providing information like page views, button clicks etc., by adding the code snippet provided by Matomo or using the appropriate library based on the programming language of the application. The Dashboards of the project are developed using the React JS framework so the Matomo Tracker [17] npm library is used while the **DEVA** application, which is implemented with Unity3D, utilises the Unity Matomo library [18].

These libraries enable more monitoring features, for example, the Unity Matomo library can track and analyse user sessions and behaviour by sending compatible metadata within the Unity application as produced from common web browsers and customising events simulating a navigation in the app like in a website. In this case, various elements of the application can be used like the Hamburger menu, the top bar, the toolbar (and its toolbox), the configuration windows, individual functions and data filters, etc.

Matomo, as a web analytics tool, provides valuable insights into user behaviour and engagement on a website or application. It helps organisations understand how users navigate through their site, which pages are visited most frequently, how long visitors stay on certain pages, and how they interact with various elements. These metrics contribute to evaluating the usability of the system by identifying areas that may require improvement, enhancing the user experience, and optimising the overall design and flow of the website or application.

For the purposes of the project, Matomo is installed in the Data Platform under https://matomo.wecompair.eu/ and currently four sites have been configured, one for every dashboard and **DEVA**. For the moment, the applications gather basic information page views, location of the visitors, etc. as depicted in Figure 11:Matomo statistics, but more types of information can be monitored in future releases upon pilot request.



Visits in real-time

DATE	VISITS	ACTIONS
Last 24 hours	2	7
Last 30 minutes	0	0

Monday, June 12, - 12:49:55
Direct Entry
Actions:

Monday, June 12, - 10:46:13 (14 min 6s)
Direct Entry
Actions:

View detailed visits log

Visitor Map

1 unique visitors (100%) 2 from unknown regions

Visits Over Time

Movers and Shakers

PAGE URLS	UNIQUE PAGEVIEWS	EVOLUTION
/index	+2	+100%
PAGE TITLES	**UNIQUE PAGEVIEWS**	**EVOLUTION**
Browse mode	+2	+100%
COUNTRY	**VISITS**	**EVOLUTION**
Greece	+2	+100%

Channel Types

CHANNEL TYPE	VISITS	UNIQUE VISITORS	USERS	ACTIONS	ACTIONS PER VISIT

Visits Overview

2 visits, **1** unique visitors +100%
7 min 3s average visit duration +100%
50% visits have bounced (left the website after one page) +100%
3.5 actions (page views, downloads, outlinks and internal site searches) per visit +100%
6 max actions in one visit +100%
7 pageviews, **2** unique pageviews +100%
0 total searches on your website, **0** unique keywords 0%
0 downloads, **0** unique downloads 0%
0 outlinks, **0** unique outlinks 0%

Figure 11:Matomo statistics

2.3 Evaluation results

This section provides an overview of the preliminary results for the current prototype in the different areas described in the validation methodology.

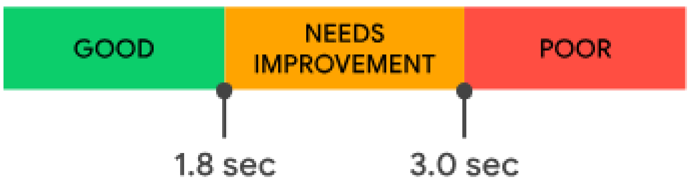
Performance

In this section, we present the results from the performance tests conducted by using the Page Speed Insights for the Dashboards, and the JMeter tool for the services of the project. The following table presents the results performed for the COMPAIR dashboards as these are derived from the Page Speed Insights tool. In this table, we present the overall score the Dashboard received from the tool and the relevant values and scores of the sub metrics used to derive this score. The thresholds are **0-49 Poor, 50-89 Moderate, 90-100 Good**

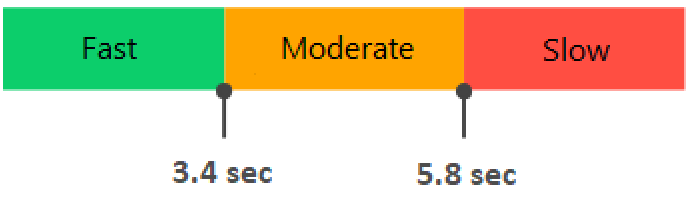
▲ 0–49 ■ 50–89 ● 90–100

More precisely, the metrics used to calculate the overall score are:

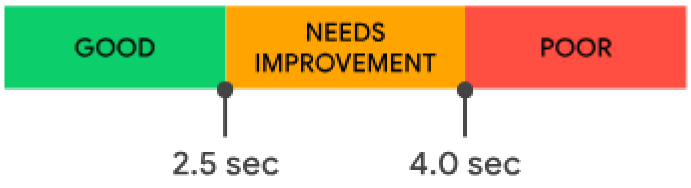
- **First Contentful Paint (FCP)**: Measures the time from when the page starts loading to when any part of the page's content is rendered on the screen. For this metric, "content" refers to text, images (including background images), <svg> elements, or non-white <canvas> elements. To provide a good user experience, sites should strive to have a First Contentful Paint of 1.8 seconds or less. It weights in the final score by default by **10%**



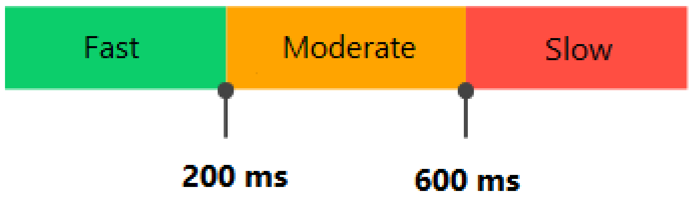
- **Speed Index (SI)**: Measures how quickly content is visually displayed during page load. The thresholds are 0-3.4 sec (Fast), 3.4-5.8 sec (Moderate), 5.8 and above (Slow). It weights in the final score by default by **10%**



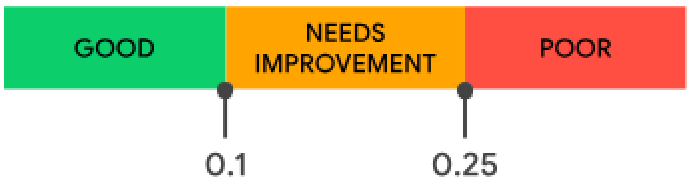
- **Largest Contentful Paint (LCP)**: Represents how quickly the main content of a web page is loaded. Specifically, LCP measures the time from when the user initiates loading the page until the largest image or text block is rendered within the viewport. To provide a good user experience, sites should strive to have an LCP of 2.5 seconds or less for at least 75% of page visits. It weights in the final score by default by **25%**



- **Total Blocking Time (TBT)**: Measures the amount of time during which Long Tasks (all tasks longer than 50ms) block the main thread and affect the usability of a page. The main thread is "blocked" because the browser cannot interrupt a task that is in progress. Therefore, in the event that a user does interact with the page in the middle of a long task, the browser must wait for the task to finish before it can respond. To provide a good user experience, sites should strive to have a Total Blocking Time of less than 200 milliseconds. It weights in the final score by default by **30%**



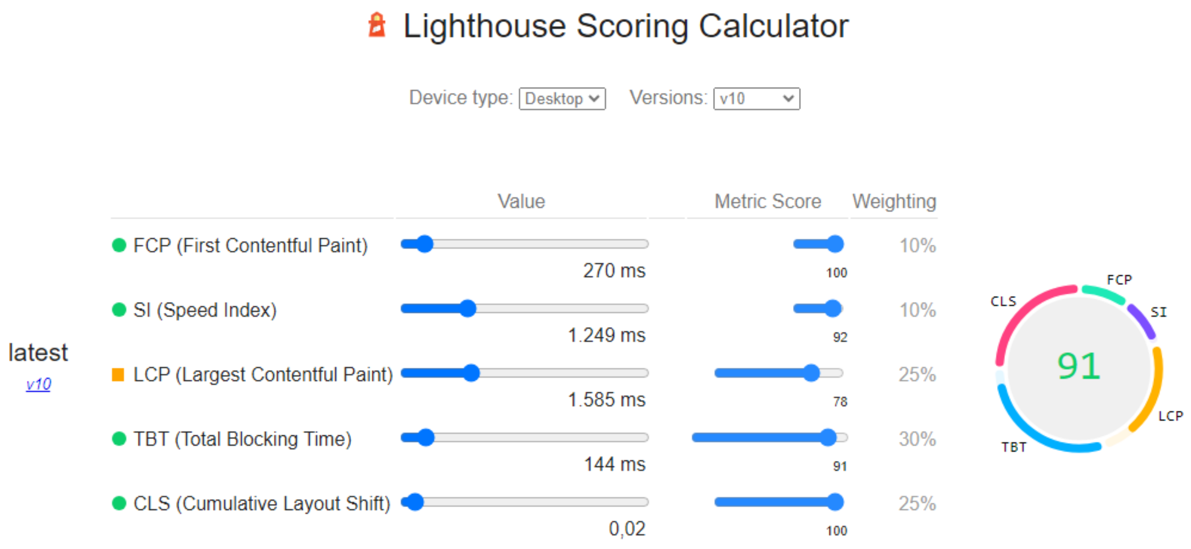
- **Cumulative Layout Shift (CLS)**: It is an important, user-centric metric for measuring visual stability because it helps quantify how often users experience unexpected layout shifts—a low CLS helps ensure that the page is delightful. To provide a good user experience, sites should strive to have a CLS score of 0.1 or less. It weights in the final score by default by **25%**



Overall, as presented in the table, all the project's Dashboards fall into the Moderate category, with the CO2 Dashboard however requiring the most effort to improve its performance based on the recommendations of the tool.

Table 2:Dashboard performance scores

Dashboard	Performance score	FCP	SI	LCP	TBT	CLS
PMD	91/100 Good	270 ms 100/100	1249 ms 92/100	1585 ms 78/100	144 ms 91/100	0,02 100/100
CO2	54/100 Moderate	730 ms 97/100	2806 ms 33/100	3118 ms 31/100	513 ms 28/100	0.02 100/100
Monitoring	75/100 Moderate	524 ms 100/100	1706 ms 75/100	2693 ms 42/100	193 ms 82/100	0,09 92/100



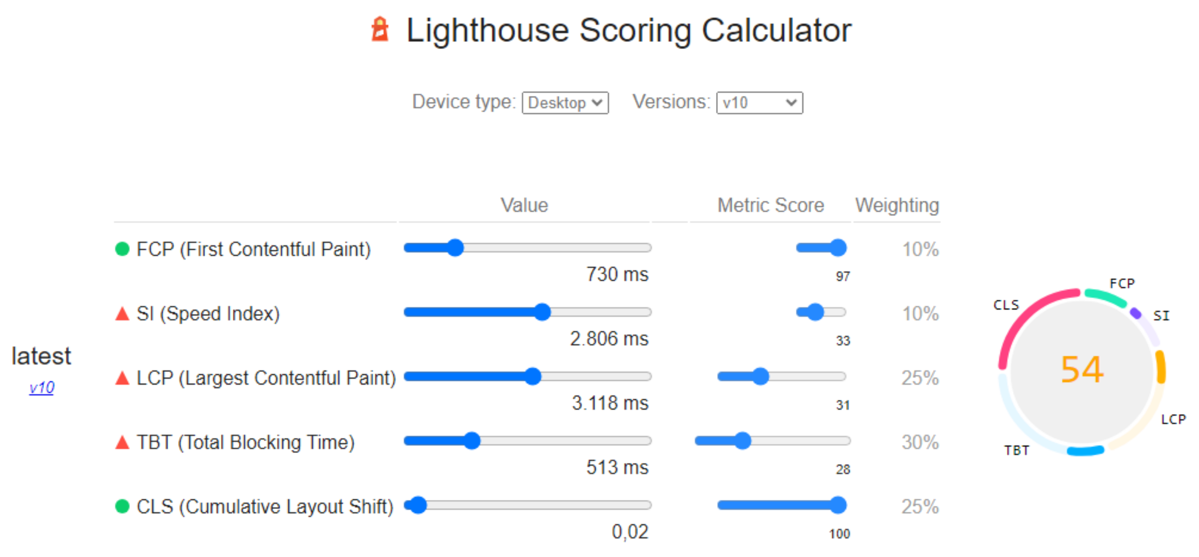
Lighthouse Scoring Calculator

Device type: Desktop Versions: v10

latest v10

	Value	Metric Score	Weighting
● FCP (First Contentful Paint)	270 ms	100	10%
● SI (Speed Index)	1.249 ms	92	10%
■ LCP (Largest Contentful Paint)	1.585 ms	78	25%
● TBT (Total Blocking Time)	144 ms	91	30%
● CLS (Cumulative Layout Shift)	0,02	100	25%

Figure 12:PMD Performance Score



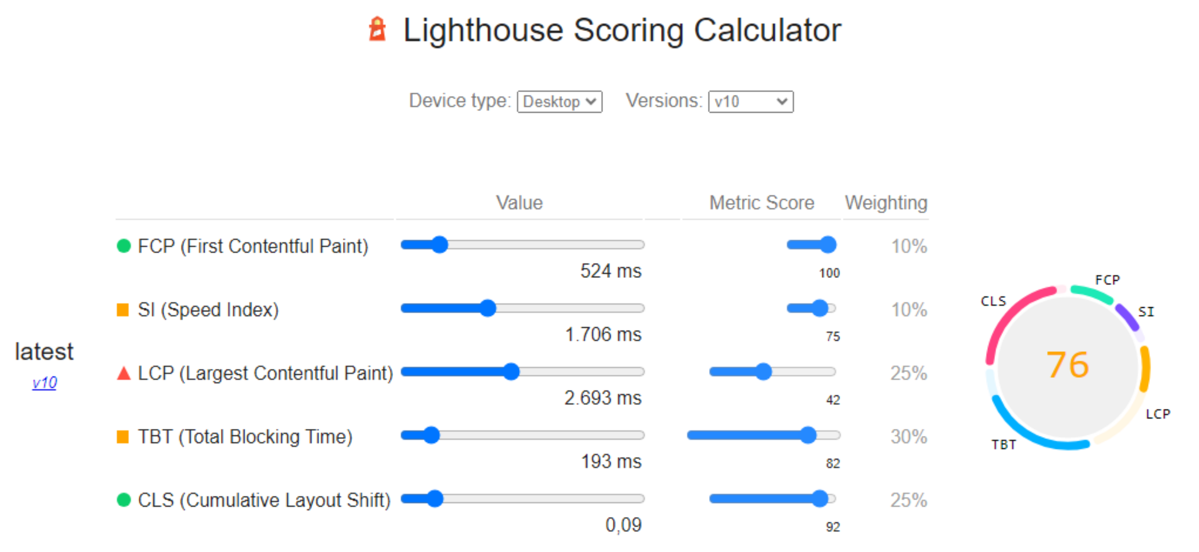
Lighthouse Scoring Calculator

Device type: Desktop Versions: v10

latest
v10

	Value	Metric Score	Weighting
● FCP (First Contentful Paint)	730 ms	97	10%
▲ SI (Speed Index)	2.806 ms	33	10%
▲ LCP (Largest Contentful Paint)	3.118 ms	31	25%
▲ TBT (Total Blocking Time)	513 ms	28	30%
● CLS (Cumulative Layout Shift)	0,02	100	25%

Figure 13:CO2 Performance Score



Lighthouse Scoring Calculator

Device type: Desktop Versions: v10

latest
v10

	Value	Metric Score	Weighting
● FCP (First Contentful Paint)	524 ms	100	10%
■ SI (Speed Index)	1.706 ms	75	10%
▲ LCP (Largest Contentful Paint)	2.693 ms	42	25%
■ TBT (Total Blocking Time)	193 ms	82	30%
● CLS (Cumulative Layout Shift)	0,09	92	25%

Figure 14: Monitoring Performance Score

In order to evaluate the overall performance and scalability of the web services of the **COMPAIR** Data Platform involved in the user interfaces, we have examined the overall response times of several of these services in various scenarios.

To implement the performance tests, we used the services running on the production server of **COMPAIR**, which has the following specifications:

- runs an Ubuntu 20.04 LTS
- 16 cores CPU AMD's Ryzen™ 9 5950X
- 128GB DDR4 ECC RAM
- 2 x 3.84 TB NVMe SSDs

We have tested the services considering a 10 seconds spike in the load, that is, all the simulated users arrive within this period of 10 seconds evenly distributed, e.g. if we consider 100 users in a period of 10 seconds, this means 10 users every second making **all the service calls** under test.

The following tables present the results of the web tests for the selected number of users. The first column shows the services that were tested, while the rest of the columns represent the various metrics measured by JMeter. More specifically, through JMeter the following results are retrieved:

- Service - The web service that has been tested;
- Samples - The number of users for the service;
- Average - The average time of response of a set of results;
- Min - The shortest time of response for the samples of the service;
- Max - The longest time or response for the samples of the service;
- Throughput – the throughput is measured in requests per time unit (second/minute/hour) and in Kbytes/sec

The scenarios under consideration are:

- Exploring the sensors in the PMD, which involves selecting a source type e.g. PM2.5, then selecting a sensor and viewing the hourly or weekly statistics.
- Exploring the projects in the Monitoring Dashboard, which involves retrieving all the available projects, selecting one and retrieving the relevant statistics.
- Exploring the DEVA, which includes retrieving the devices and sensors and requesting for observations of a certain type.

The following tables present the results of stress tests performed in various APIs of the **COMPAIR** system, following the scenarios above, using the JMeter tool. For the purposes of the test, we considered two cases for 100 and 300 simultaneous users, a high number compared to the project's needs.

Table 3: PMD exploring the sensors - stress tests

Component	**Samples**	**Min (ms)**	**Max (ms)**	**Avg (ms)**	**Throughput**
PMD - Exploring the sensors					
Data Manager (/data/sensors/observations/type/)	100	515	689	617	9.5/sec
Data Manager (/data/sensors/observations/type/)	300	802	23975	10065	10.9/sec
Data Manager (/data/sensors/thing/)	100	77	120	84	10.0/sec
Data Manager (/data/sensors/thing/)	300	77	13599	285	11.3/sec

Data Manager (/data/sensors/distributions)	100	91	153	112	10.0/sec
Data Manager (/data/sensors/distributions)	300	96	7548	534	11.3/sec
Data Manager (/data/sensors/inactive)	100	104	182	125	10.0/sec
Data Manager (/data/sensors/inactive)	300	102	10739	318	11.3/sec
Data Manager (/data/sensors/timeseries)	100	101	143	114	10.0/sec
Data Manager (/data/sensors/timeseries)	300	173	27467	1926	10.6/sec

Table 4: Monitoring Dashboard - Exploring the projects - stress tests

Component	**Samples**	**Min (ms)**	**Max (ms)**	**Avg (ms)**	**Throughput**
Monitoring Dashboard - Exploring the projects					
Data Manager (/data/project/all?status=public)	100	325	338	333	9.8/sec
Data Manager (/data/project/all?status=public)	300	445	19615	4297	20.6/sec
Data Manager (/data/project/)	100	379	402	389	9.7/sec
Data Manager (/data/project/)	300	658	12999	2036	19.4/sec
Data Manager (/data/sensors/aggregations-summary/)	100	324	443	335	9.8/sec
Data Manager (/data/sensors/aggregations-summary/)	300	435	12438	4394	16.9/sec
Data Manager (/data/sensors/aggregations-summary/)	100	269	443	335	9.9/sec
Data Manager (/data/sensors/aggregations-summary/)	300	305	11460	4862	17.0/sec

Table 5: Exploring DEVA - stress tests

Component	Samples	Min (ms)	Max (ms)	Avg (ms)	Throughput
DEVA - Exploring the DEVA					
DEVA Manager (/deva/sensors/things)	100	256	376	270	9.8/sec
DEVA Manager (/deva/sensors/things)	300	248	430	306	29.5/sec
DEVA Manager (/deva/sensors/)	100	89	198	101	10/sec
DEVA Manager (/deva/sensors/)	300	90	174	112	29.9/sec
DEVA Manager (/deva/sensors/observations/type/PM_2p5)	100	80	192	91	10/sec
DEVA Manager (/deva/sensors/observations/type/PM_2p5)	300	78	148	100	29.9/sec

Overall, the average response time of the services is within accepted ranges in most cases,, based on the requirements of UX as defined by Jakob Nielsen [19], with a few exceptions that need to be improved. According to these requirements, the limits are:

- 0.1 second is the limit that the user feels that the system is reacting instantaneously
- 1 second is the limit that user's flow of thought stays uninterrupted
- 10 seconds is the limit for keeping the user's attention

Although the limits were set for the response time of a UI, we can also use them also for the response times and APIs and thus several solutions can be investigated in the rest of the project to improve the performance of the services, e.g. by:

- The development of a caching mechanism at least for the most common API calls.
- Improving the underlying algorithms, utilising also parallel programming
- Increasing the instances of the services and load balance further
- Identifying the most frequent queries in the database and use more indexes, etc.

DEVA

As mentioned, the **DEVA** will be evaluated for this category, in terms of frame rate, latency of processing the data, data transfer and data consumption.

The results presented here differ from call to call, depending on the process running on the device (background activities, services etc.) and the current connectivity to the Data Manager (speed of connection, type of connection, and internet loads). Therefore, the results in the tables below are rounded or average values.

Frame rate

Depending on the application mode, i.e. the visualisation style or the time range of the observations (and amount of data), the render speed may differ.

The speed shall also differ for different mobile devices with different CPU and GPU (graphic processor) on it. Devices are classified in three groups: Basic range, mid-range and high range. Inside these groups, many distinctions and types exist making a comparison between all kinds of phones and tablets complicated. For the first metric, we used a mobile phone and a tablet.

Unity3D offers the possibility to measure frames per second in (fps, in Hz). For the current implementation, the following figures were retrieved from the tool for these scenarios:

- No user movement, no rotation:
  User is staying at position (+/- 1 metre), targeting a fixed direction.

- No user movement, with rotation:
  User is staying at position (+/- 1 metre); looking around.

- Slow movement, no rotation:
  User is walking straight forward, targeting a fixed direction.

- Slow movement, with rotation:
  User is walking straight forward, looking around.

- Quick movement, no rotation:
  User is running/driving a bicycle, not looking around.

The results demonstrate that under various usage conditions, the render speed is always fast enough to provide a responsive visualisation. While looking around, the speed mainly stays the same then without this pivot because all the data surrounding the user are loaded, no matter at which direction the user is oriented. By walking quickly, running or driving the system has additionally to sort data for the Near-n-Far view mode (if active).

The result was produced for two different mobile devices: a smartphone of type Huawei P30 (mid-range) and a tablet of type Samsung GALAXY S8 (high range) using an extreme scenario (visualising 1.000 values around the users). The result in fps is given for both devices: P30 / S8. To show the difference between a *Sphere* S (low complex geometry), the *Pin* visual P (middle complex geometry) and the *Display* visual D (high complex geometry) the results are mentioned in the table (S and P values, without Near-n-Far mode).

Table 6:FPS of DEVA for various scenarios

Scenarios	**FPS (looking straight ahead) 1.000 Values P30 / S8**	**FPS (looking around) 1.000 Values P30 / S8**
User doesn't move	S: 30 Hz / 30 Hz P: 26 Hz / 27 Hz D: 16 Hz/ 17 Hz	S: 29 Hz / 30 Hz P: 26 Hz / 26 Hz D: 15 Hz / 17 Hz
Slow movement (walking)	S: 30 Hz / 30 Hz P: 25 Hz / 25 Hz D: 15 Hz / 15 Hz	S: 29 Hz / 30 Hz P: 24 Hz / 24 Hz D: 15 Hz / 16 Hz
Quick movement (running, driving)	S: 30 Hz / 30 Hz P: 23 Hz / 24 Hz D: 14 Hz / 15 Hz	Not applicable

If the system slows down too much (< 10 fps), an idea should be to disable the Near-n-Far mode when the movement of the device exceeds a speed. This should be implemented in a future version.

Remark:

- Activating the *Gamification* mode (currently implemented as a simple graphical extension, currently only available for the *Display* mode) will decrease the frame rate by approx. 1 frame. By increasing the complexity, the frame rate should decrease further.
- Activating the Near-n-Far mode speeds up the rendering because the far observations are rendered as spheres.
- The results always include the rendering of the GUI. Those elements are not very complex, Unity uses mainly simple squares with static textures to render them.
- When using 100 observation values, the frame rate stays stable by 30 Hz no matter which visualisation style was selected.

Latency

Within Unity3D, there is the possibility to measure run-times for different parts of the application such as time for initialisation, loading the user profile, loading times for observations/sensor data and the processing in the DEVA pipeline. Those measurements provide an additional indication about the responsiveness of the application.

Latency depends also from the type of communication: The usage of wi-fi, 4G or 5G, and the kind of contract the user has with his provider, has a big impact over the download speed.

The CPU and the range of the device (basic, mid, high; see "Frame rate") also influence the speed of processing, especially the processing of data in the pipeline. The same devices should be used as before. To show the difference between a *Sphere* S (simple geometry) and the *Display* visual D (high complex geometry with many sub objects) both results are mentioned in the table (S and D values).
The following measurements where retrieved from the app:

Table 7:Latency of DEVA for various scenarios

Scenarios	**100 Values, in ms P30 / S8**	**1.000 Values, in ms P30 / S8**
Ping to the Data Manager (ping aver JSON)	100 ms / 100 ms	100 ms / 100 ms
Querying data from Data Manager (Observations)	225 ms / 233 ms	2250 ms / 2328 ms
Preparing the data for the pipeline (Instantiate AR geometries)	S: 50 ms / 44 ms D: 170-240 ms / 190 ms	S: 300 ms / 170 ms D: 3500 ms / 2200 ms
Updating the pipeline (Changing AR visuals…)	S: 42 ms / 4 ms D: 94 ms / 28 ms	S: 680 ms / 35 ms D: 1250 ms / 300 ms
Changing the Near-n-Far distance	S: 7 ms / 6 ms D:18-25 ms / 4-20 ms	S: 35 ms / 25-30 ms D: 120-340 ms / 40-300 ms
Deleting the data (e.g. to reset the pipeline)	S: 12 ms / 11 ms D: 42 ms / 35 ms	S: 40 ms / 36 ms D: 446 ms / 349 ms

Here, in comparison to the fps, the amount of data has a significant impact on the latency, because the data has to be read, transposed by the pipeline and the resulting graphical objects created by Unity. Depending on the visual style of an observation (simple 3D sphere to complex display with text, icons and graphic), the scene graph is less or more complex.

Remark:

- The results ignore the first calls to the Data Manager needing more times for the same request.
- Changing the value of the Near-n-Far distance using a complex visual (e.g. *Display*) will affect the latency because far data is rendered as simple spheres.

Data consumption

The **DEVA** visualises environmental data that are stored on a server and managed by the Data Manager. Depending on the user position and the movement of the mobile AR device, new data requests are sent from **DEVA** to the Data Manager in order to get an update of the measurements in the surrounding area.

The number of sensors and measurements in the neighbourhood is quite small and in the order of several dozen measurements. Taking the data protocol currently implemented into account just a few Kbytes of data are required to get an update. As described in the section on monitoring DEVA, the amount of data per request and the number of requests can be monitored. The resulting figures will provide details that are more accurate as soon as the app is distributed to the pilot partners and in use.

The following measurements where retrieved from the logs of the app (for the same scenarios as before):

Table 8: Data consumption for various scenarios

Scenarios	Package Size (JSON) 100 Values	Package Size (JSON) 1.000 Values	Package Size (JSON) 10.000 Values
Querying Observations	17,67 KB	176,72 KB	1,72 MB
Querying Sensors	14,35 KB	143,55 KB	1,40 MB
Querying Things	18,76 KB	187,66 KB	1,76 MB

Querying hypothetical 100 things, each with two sensors and N observation values result in a payload of approx.:

- for N=100 observations per sensors:
  18,76 (things) + 2 x 14,35 (sensors) + 200 x 17,67 (observations) = **3,49 MB**

- for N=1.000 observations per sensors:
  18,76 (things) + 2 x 14,35 (sensors) + 200 x 176,72 (observations) = **34,56 MB**

Remarks:
(1) The size of packages depends a lot on the content of some attributes like textual information: name, description and unit. Other values like integer and float numbers (id, observation values, GPS locations) as well as Boolean values (yes/no) are more of a constant size.
(2) Extending the OpenAPI protocol in future development stages (e.g. adding more attributes) shall also affect the payloads considerably.

Functional suitability

In this section, we present the results for the functional suitability of various components and dashboards based on the measurements gathered from SonarQube regarding unit / integration tests and code coverage. As a target, we have set an overall code coverage for the backend components at least **40%** and for the dashboard at least **20%**. The reason that the dashboards have a lower percentage is that they contain many files that are related to the presentation of the UI but not the interaction with the user, so there is an increase in the total number of lines that cannot be tested. In any case, as we progress with the project, more tests will need to be implemented in order to achieve the required threshold.

Table 9:Functional suitability measurements

Component	Unit tests	Overall Coverage
Data Manager	39	78,3%
User Manager	6	40,5%
Deva Manager	14	82,2%
PMD	9	21,3%
CO2 Dashboard	2	0.1%
Monitoring Dashboard	3	9,5%

Compatibility

All the components in the **COMPAIR** project use well established communication protocols to exchange information. The main mode of communication is performed through JSON REST web services, a type of web service that enables communication and data exchange between different software applications over the internet. JSON (JavaScript Object Notation) is a lightweight data interchange format that is commonly used for structuring and transmitting data in web services.

JSON Web Services are commonly used in various scenarios, such as retrieving and updating data from remote servers, integrating different applications or systems, and building web and mobile applications that consume data from external sources.

Overall, JSON Web Services provide a flexible and lightweight approach for applications to exchange data and functionality, enabling interoperability and integration between disparate systems and platforms.

Additionally, for the collection of the sensor data, we define the structure of the messages based on the OGC SensorThings standard [20]. OGC SensorThings is an open standard developed by the Open Geospatial Consortium (OGC) that focuses on enabling the interoperable discovery, querying, and integration of real-time sensor data from diverse sources. It provides a standardised framework for managing and accessing sensor data over the web, allowing developers and organisations to build applications and systems that leverage real-time sensing capabilities.

The OGC SensorThings standard aims to simplify the discovery, integration, and consumption of real-time sensor data from diverse sources. It provides a common framework that enables developers to build applications that leverage sensor data for a wide range of applications, such as environmental monitoring, smart cities, agriculture, and industrial monitoring.

By adhering to the SensorThings standard, organisations can ensure interoperability and compatibility between different sensor networks, platforms, and applications. This facilitates the integration and aggregation of sensor data from multiple sources, enabling comprehensive analysis, decision-making, and insights based on real-time information.

In addition, the fact that all the components of the Data Platform are dockerized, guarantees that two or more components can coexist in the same common environment, sharing the resources of the host server, without issues, since the docker images run in isolation and any dependency of one component, cannot affect another component.

Finally, the code base of the DEVA is implemented in a way that multiple platforms are supported. For Augmented reality applications, Android and iOS are the two operating systems and thanks to the Unity3D AR Foundation classes, a single code base can serve both operating systems. As long as this library is used, the required methods behave identical on both platforms. The only requirement regarding software from DEVA is that the device running the app has to be AR compatible.

Usability

In this section, we report the results for accessibility from the PageSpeed Insights for the dashboards of the project. As mentioned previously in the document, the accessibility rules are defined by the WCAG 2.1 guidelines. The thresholds are **0-49 Poor, 50-89 Moderate and 90-100 Good**. Overall, the Dashboards have a good degree of accessibility compliance with some room of improvement.

▲ 0–49 ■ 50–89 ● 90–100

Table 10: PageSpeed Usability measurements

Dashboard	Accessibility Score
PMD	81 / 100 Moderate
CO2	69/100 Moderate
Monitoring	77/100 Moderate

Considering that the Open Group testing planned by the pilots, where this prototype will be tested by a larger number of users, has not started at the time of writing this document, it is not possible to provide useful insights regarding the user satisfaction based on questionnaires or the user behaviour based on Matomo. This information will be provided in the second version of this document in M30.

Security

In this section, we report the results based on the security of the components and dashboards of the project as these are derived from SonarQube as part of the static analysis of the code and SecurityHeaders and OWASP ZAP for the overall security.

The following table provides the results for the components and Dashboards of the project from the analysis performed by SonarQube in regards to security. More precisely we report the number of vulnerabilities and rating at the time of writing the document. The aim is to achieve the highest rating (A) for all the components under investigation by the end of the project. Currently this goal is partially achieved.

Table 11:SonarQube security measurements

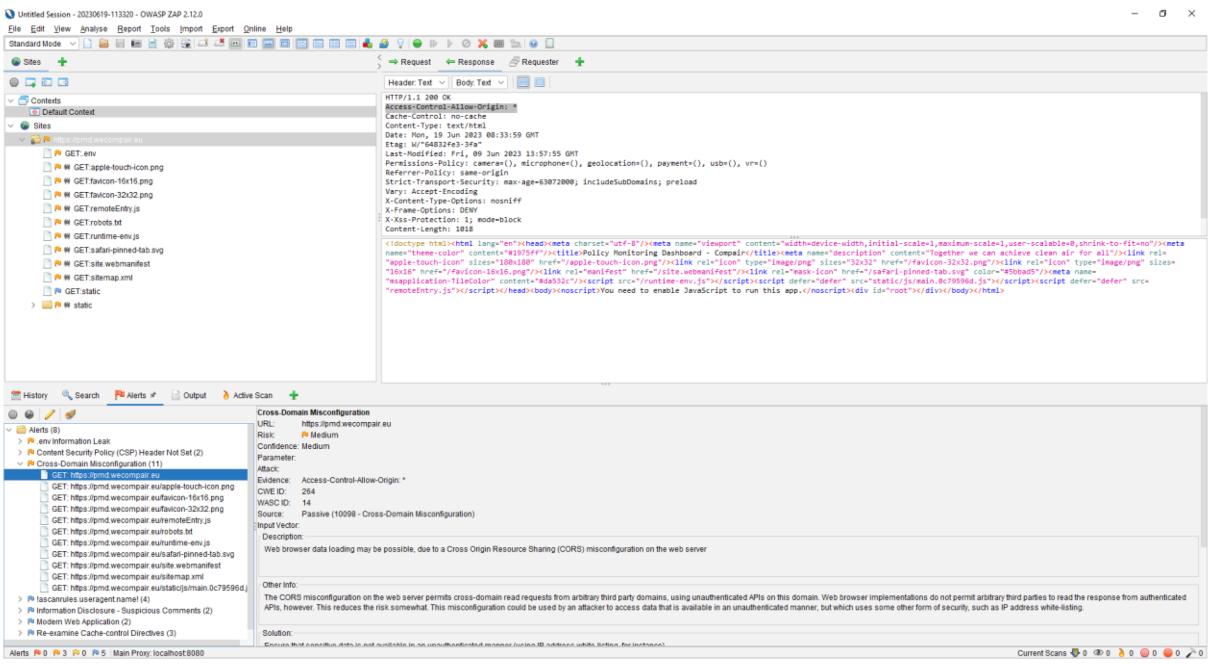
Component	Number of Vulnerabilities	Rating
Data Manager	2	D
User Manager	0	A
Deva Manager	0	A
PMD	0	A
CO2 Dashboard	1	D
Monitoring Dashboard	1	D
Air Quality Calibrator	0	A

The following table presents the results of the scan performed by the SecurityHeaders tool to the Dashboards of the project indicating a score based on the existence of specific security headers at the time of writing this document.

Table 12:Security Headers scores

Dashboard	Security Headers score
PMD	A
CO2	F
Monitoring	A

At the time of writing the document the project's Dashboards were also tested by the OWASP Zap tool. The tool uses a "spider app" in order to identify all the links of the dashboards and performs an automated attack looking for specific weaknesses.


Untitled Session - 20230619-113320 - OWASP ZAP 2.12.0

Sites
- Contexts
  - Default Context
- Sites
  - https://pmd.wecompair.eu
    - GET:.env
    - GET:apple-touch-icon.png
    - GET:favicon-16x16.png
    - GET:favicon-32x32.png
    - GET:remoteEntry.js
    - GET:robots.txt
    - GET:runtime-env.js
    - GET:safari-pinned-tab.svg
    - GET:site.webmanifest
    - GET:sitemap.xml
    - GET:static
    - static

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Cache-Control: no-cache
Content-Type: text/html
Date: Mon, 19 Jun 2023 08:33:59 GMT
Etag: W/"64832fe3-3fa"
Last-Modified: Fri, 09 Jun 2023 13:57:55 GMT
Permissions-Policy: camera=(), microphone=(), geolocation=(), payment=(), usb=(), vr=()
Referrer-Policy: same-origin
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-Xss-Protection: 1; mode=block
Content-Length: 1018
```

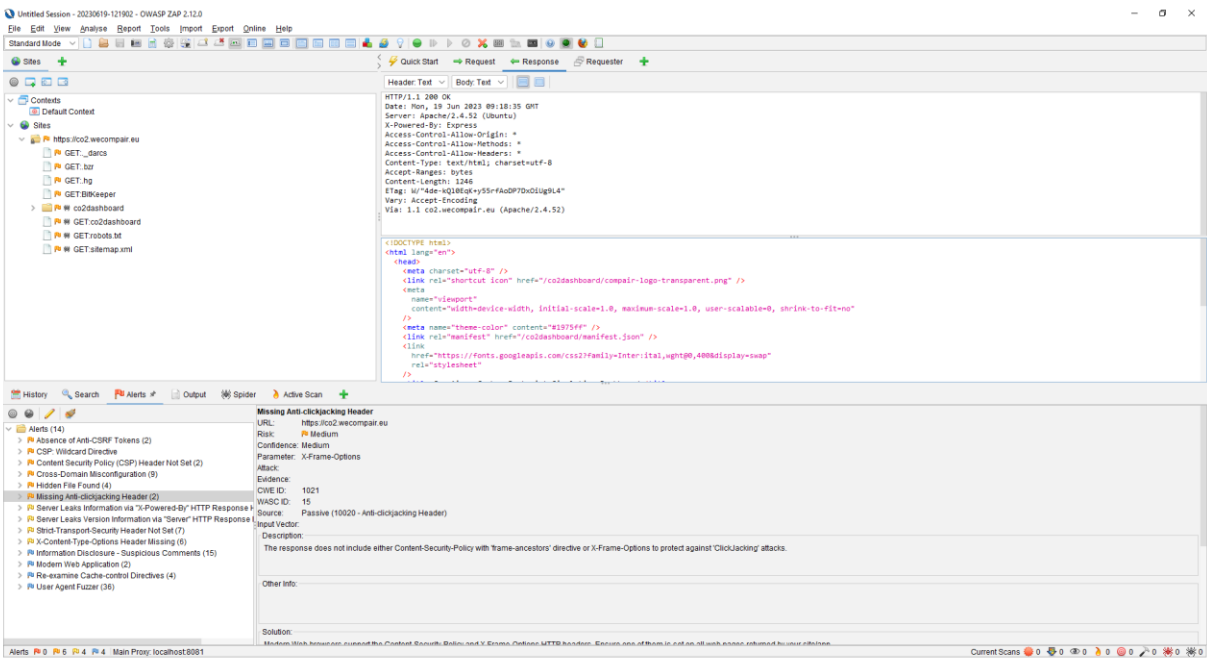
```
<!doctype html><html lang="en"><head><meta charset="utf-8"/><meta name="viewport" content="width=device-width,initial-scale=1,maximum-scale=1,user-scalable=0,shrink-to-fit=no"/><meta
name="theme-color" content="#1975ff"/><title>Policy Monitoring Dashboard - Compair</title><meta name="description" content="Together we can achieve clean air for all"/><link rel=
"apple-touch-icon" sizes="180x180" href="/apple-touch-icon.png"/><link rel="icon" type="image/png" sizes="32x32" href="/favicon-32x32.png"/><link rel="icon" type="image/png" sizes=
"16x16" href="/favicon-16x16.png"/><link rel="manifest" href="/site.webmanifest"/><link rel="mask-icon" href="/safari-pinned-tab.svg" color="#5bbad5"/><meta name=
"msapplication-TileColor" content="#da532c"/><script src="/runtime-env.js"></script><script defer="defer" src="static/js/main.0c79596d.js"></script><script defer="defer" src=
"remoteEntry.js"></script></head><body><noscript>You need to enable JavaScript to run this app.</noscript><div id="root"></div></body></html>
```

Alerts (8)
- .env Information Leak
- Content Security Policy (CSP) Header Not Set (2)
- Cross-Domain Misconfiguration (11)
  - GET: https://pmd.wecompair.eu
  - GET: https://pmd.wecompair.eu/apple-touch-icon.png
  - GET: https://pmd.wecompair.eu/favicon-16x16.png
  - GET: https://pmd.wecompair.eu/favicon-32x32.png
  - GET: https://pmd.wecompair.eu/remoteEntry.js
  - GET: https://pmd.wecompair.eu/robots.txt
  - GET: https://pmd.wecompair.eu/runtime-env.js
  - GET: https://pmd.wecompair.eu/safari-pinned-tab.svg
  - GET: https://pmd.wecompair.eu/site.webmanifest
  - GET: https://pmd.wecompair.eu/sitemap.xml
  - GET: https://pmd.wecompair.eu/static/js/main.0c79596d.j
- !ascanrules.useragent.name! (4)
- Information Disclosure - Suspicious Comments (2)
- Modern Web Application (2)
- Re-examine Cache-control Directives (3)

Cross-Domain Misconfiguration
URL: https://pmd.wecompair.eu
Risk: Medium
Confidence: Medium
Parameter:
Attack:
Evidence: Access-Control-Allow-Origin: *
CWE ID: 264
WASC ID: 14
Source: Passive (10098 - Cross-Domain Misconfiguration)
Input Vector:
Description:
Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server
Other Info:
The CORS misconfiguration on the web server permits cross-domain read requests from arbitrary third party domains, using unauthenticated APIs on this domain. Web browser implementations do not permit arbitrary third parties to read the response from authenticated APIs, however. This reduces the risk somewhat. This misconfiguration could be used by an attacker to access data that is available in an unauthenticated manner, but which uses some other form of security, such as IP address white-listing.
Solution:

Alerts 0 3 0 5 Main Proxy: localhost:8080

Figure 15:PMD Dashboard ZAP results


Untitled Session - 20230619-121902 - OWASP ZAP 2.12.0

Sites
- Contexts
  - Default Context
- Sites
  - https://co2.wecompair.eu
    - GET:_darcs
    - GET:.bzr
    - GET:.hg
    - GET:BitKeeper
    - co2dashboard
    - GET:co2dashboard
    - GET:robots.txt
    - GET:sitemap.xml

```
HTTP/1.1 200 OK
Date: Mon, 19 Jun 2023 09:18:35 GMT
Server: Apache/2.4.52 (Ubuntu)
X-Powered-By: Express
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: *
Access-Control-Allow-Headers: *
Content-Type: text/html; charset=utf-8
Accept-Ranges: bytes
Content-Length: 1246
ETag: W/"4de-kQ10EqK+y55rfAoDP7DxOiUg9L4"
Vary: Accept-Encoding
Via: 1.1 co2.wecompair.eu (Apache/2.4.52)
```

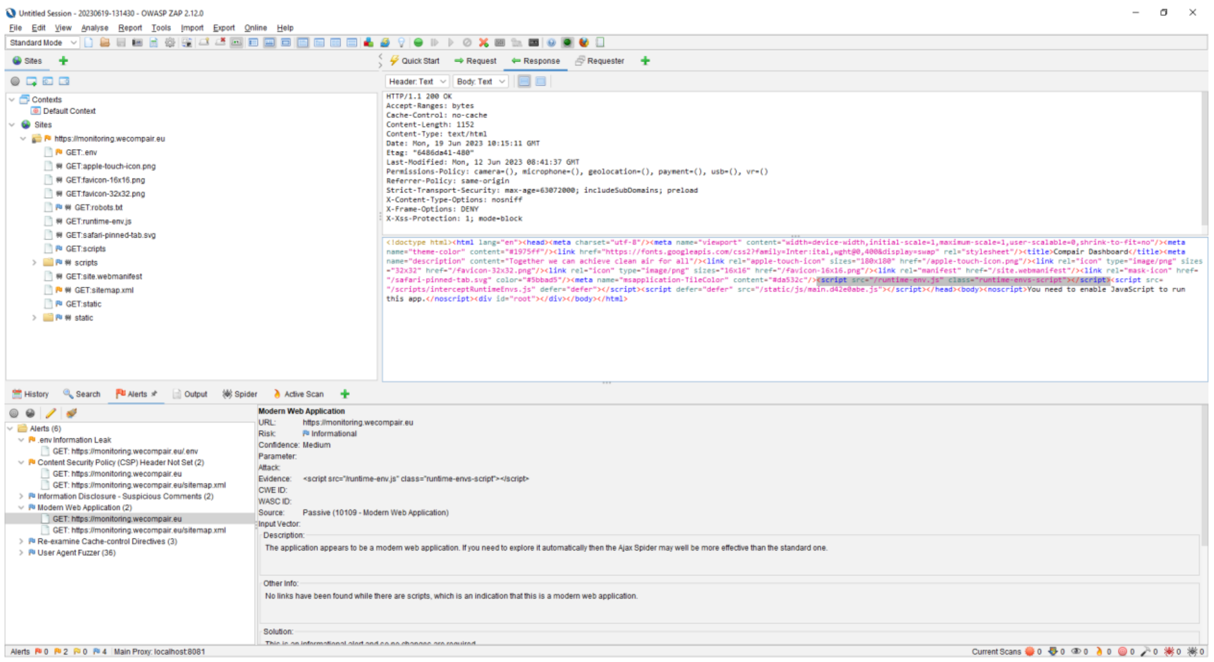
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="shortcut icon" href="/co2dashboard/compair-logo-transparent.png" />
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0, shrink-to-fit=no"
    />
    <meta name="theme-color" content="#1975ff" />
    <link rel="manifest" href="/co2dashboard/manifest.json" />
    <link
      href="https://fonts.googleapis.com/css2?family=Inter:ital,wght@0,400&display=swap"
      rel="stylesheet"
    />
```

Alerts (14)
- Absence of Anti-CSRF Tokens (2)
- CSP: Wildcard Directive
- Content Security Policy (CSP) Header Not Set (2)
- Cross-Domain Misconfiguration (9)
- Hidden File Found (4)
- Missing Anti-clickjacking Header (2)
- Server Leaks Information via "X-Powered-By" HTTP Response H
- Server Leaks Version Information via "Server" HTTP Response
- Strict-Transport-Security Header Not Set (7)
- X-Content-Type-Options Header Missing (6)
- Information Disclosure - Suspicious Comments (15)
- Modern Web Application (2)
- Re-examine Cache-control Directives (4)
- User Agent Fuzzer (36)

Missing Anti-clickjacking Header
URL: https://co2.wecompair.eu
Risk: Medium
Confidence: Medium
Parameter: X-Frame-Options
Attack:
Evidence:
CWE ID: 1021
WASC ID: 15
Source: Passive (10020 - Anti-clickjacking Header)
Input Vector:
Description:
The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks.
Other Info:
Solution:

Alerts 0 6 4 4 Main Proxy: localhost:8081

Figure 16:CO2 Dashboard ZAP results



Contexts
- Default Context

Sites
- https://monitoring.wecompair.eu
  - GET:.env
  - GET:apple-touch-icon.png
  - GET:favicon-16x16.png
  - GET:favicon-32x32.png
  - GET:robots.txt
  - GET:runtime-env.js
  - GET:safari-pinned-tab.svg
  - GET:scripts
  - scripts
  - GET:site.webmanifest
  - GET:sitemap.xml
  - GET:static
  - static

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: no-cache
Content-Length: 1152
Content-Type: text/html
Date: Mon, 19 Jun 2023 10:15:11 GMT
Etag: "6486da41-480"
Last-Modified: Mon, 12 Jun 2023 08:41:37 GMT
Permissions-Policy: camera=(), microphone=(), geolocation=(), payment=(), usb=(), vr=()
Referrer-Policy: same-origin
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-Xss-Protection: 1; mode=block
```

```
<!doctype html><html lang="en"><head><meta charset="utf-8"/><meta name="viewport" content="width=device-width,initial-scale=1,maximum-scale=1,user-scalable=0,shrink-to-fit=no"/><meta
name="theme-color" content="#1975ff"/><link href="https://fonts.googleapis.com/css2?family=Inter:ital,wght@0,400&display=swap" rel="stylesheet"/><title>Compair Dashboard</title><meta
name="description" content="Together we can achieve clean air for all"/><link rel="apple-touch-icon" sizes="180x180" href="/apple-touch-icon.png"/><link rel="icon" type="image/png" sizes
="32x32" href="/favicon-32x32.png"/><link rel="icon" type="image/png" sizes="16x16" href="/favicon-16x16.png"/><link rel="manifest" href="/site.webmanifest"/><link rel="mask-icon" href=
"/safari-pinned-tab.svg" color="#5bbad5"/><meta name="msapplication-TileColor" content="#da532c"/><script src="/runtime-env.js" class="runtime-envs-script"></script><script src=
"/scripts/interceptRuntimeEnvs.js" defer="defer"></script><script defer="defer" src="/static/js/main.d42e0abe.js"></script></head><body><noscript>You need to enable JavaScript to run
this app.</noscript><div id="root"></div></body></html>
```

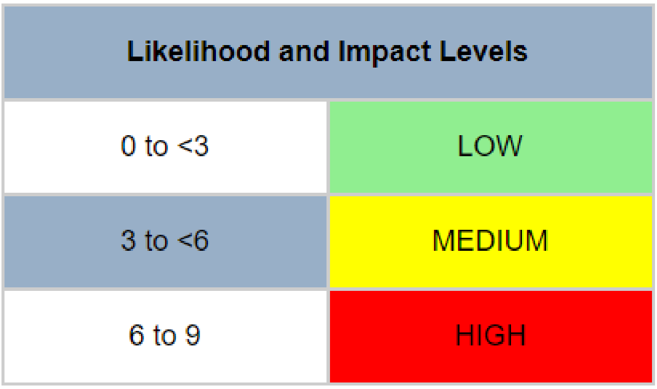
Alerts (6)
- .env Information Leak
  - GET: https://monitoring.wecompair.eu/.env
- Content Security Policy (CSP) Header Not Set (2)
  - GET: https://monitoring.wecompair.eu
  - GET: https://monitoring.wecompair.eu/sitemap.xml
- Information Disclosure - Suspicious Comments (2)
- Modern Web Application (2)
  - GET: https://monitoring.wecompair.eu
  - GET: https://monitoring.wecompair.eu/sitemap.xml
- Re-examine Cache-control Directives (3)
- User Agent Fuzzer (36)

Modern Web Application
URL: https://monitoring.wecompair.eu
Risk: Informational
Confidence: Medium
Parameter:
Attack:
Evidence: <script src="/runtime-env.js" class="runtime-envs-script"></script>
CWE ID:
WASC ID:
Source: Passive (10109 - Modern Web Application)
Input Vector:
Description:
The application appears to be a modern web application. If you need to explore it automatically then the Ajax Spider may well be more effective than the standard one.
Other Info:
No links have been found while there are scripts, which is an indication that this is a modern web application.
Solution:
This is an informational alert and so no changes are required.

Alerts 0 2 0 4 Main Proxy: localhost:8081
Current Scans 0 0 0 0 0 0 0 0

Figure 17:Monitoring Dashboard ZAP results

The following table summarises these results. The scan groups the threats into four main categories, High, Medium, Low and Informative. The overall risk of a threat falls into these categories based on a likelihood estimation, that is, how likely is that a related attack appears and the impact estimation, that is, the damage this attack may do to the system. This is done by figuring out whether the likelihood is low, medium, or high and then do the same for impact [21]. The 0 to 9 scale is split into three parts as depicted in the following image:



Likelihood and Impact Levels	
0 to <3	LOW
3 to <6	MEDIUM
6 to 9	HIGH

Figure 18:Likelihood and Impact levels

The Informative alerts essentially do not pose a threat but do provide some information about the server or the application that potentially can use.

Based on the results of the ZAP scans, there is not an immediate threat to the system but some Medium and Low-level alerts, mostly about the lack of some security headers and more restrictive rules for the CORS configurations that need to be tackled in the following releases of the Dashboards.

Table 13:Zap security results

Dashboard	Zap results
PMD	3 Medium level threats about one security header missing and CORS misconfiguration and 5 Informative warnings.
CO2	6 Medium level threats about the missing security headers and CORS misconfiguration, 4 Low level threats about leaking server information and 4 Informative warnings
Monitoring	2 Medium level threats about one security header missing and 4 Informative warnings.

Overall, based on the tests performed, the security of the tools under investigation is at a quite high level, however some effort is still required to reach the highest possible level of security. All tools used in this section, provide recommendations on how the security can be improved, so following the provided guidelines and solving the relevant issues for every component, we can achieve an overall rating of A.

Maintainability

This characteristic represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements.

The architecture of the project is modular which means that the components of the system are loosely coupled, thus internal changes in one component will not affect the operations of another. Changes however in the inputs or outputs of a component, may affect the operations of the integrated system.

In the following table we present the code analysis results in code maintainability as these were calculated by SonarQube. Code smells is code that may be confusing and difficult to maintain, while the debt is the time estimated to fix all the code smells. From the results obtained, the number of the code smells is low and thus the overall technical debt is really low, resulting in the highest rating in all projects. The goal is to reach the highest rating for all components (A) which currently is achieved.

Table 14:SonarQube maintainability measurements

Component	Code smells	Debt	Rating
Data Manager	31	7h 12 min	A
User Manager	13	3h 37 min	A
Deva Manager	19	5h 30 min	A
PMD Dashboard	124	1d 4h	A
CO2 Dashboard	312	3d 1h	A

Monitoring Dashboard	38	3h 34 min	A
Air Quality Calibrator	28	1h 50min	A

Reliability

This is the degree to which a system, product or component performs specific functions under specified conditions for a specified period. In the following table, we present the results of the code analysis in terms of code reliability as these are derived from SonarQube's static analysis. Overall, all the components investigated have a small number of bugs and thus they receive a high rating with most of them getting the highest possible (A). The goal is by the end of the project all the components to have a rating of A by resolving the underlying bugs identified.

Table 15:SonarQube reliability measurements

Component	Bugs	Rating
Data Manager	0	A
User Manager	0	A
Deva Manager	0	A
PMD Dashboard	0	A
CO2 Dashboard	5	D
Monitoring Dashboard	1	C
Ait Quality Calibrator	0	A

Portability

Portability is the degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another. All the core components deployed in the Data Platform are dockerized and pushed into the project’s private repositories. All the supporting tools of these components are also dockerized and the relevant docker compose files have been prepared in a way that can easily be configured so that it can be used in other servers.

A backup mechanism for the databases is already in place and through supporting tools can be restored in another environment easily. Any file needed can also be pulled into a new environment through the project’s private repositories.

The majority of the components of AQ CaaS Platform utilise Databricks which in turn can be used with various cloud providers (AWS, Microsoft Azure and Google Cloud Platform). By supporting multiple cloud providers, Databricks provides flexibility and choice to organisations, allowing them to leverage their preferred cloud platform while taking advantage of the unified analytics capabilities offered by Databricks. This enables seamless data integration, scalable data processing, and advanced analytics across different cloud environments.

Overall, the components developed for the project have a high degree of transferability.

3. Conclusion

In conclusion, this document provides a comprehensive assessment of the software components developed for the **COMPAIR** project, utilising the ISO/IEC 25010:2011 standard methodology. By following this methodology, the report evaluates the software system's compliance with the defined quality characteristics, identifies areas of improvement, and offers a better perspective for enhancing its performance, reliability, security, usability, and other important aspects.

The findings and recommendations presented in this report serve as valuable insights for the project stakeholders, enabling them to make informed decisions regarding the software system's deployment, further development, and overall success. The report plays a crucial role in ensuring the software system's quality, efficiency, and alignment with the project's objectives, ultimately contributing to the overall success of the project.

4. References

1. https://www.sonarsource.com/products/sonarqube/
2. https://prometheus.io/
3. https://grafana.com/
4. https://www.databricks.com/
5. https://telraam.net/
6. https://docs.aws.amazon.com/health/latest/ug/getting-started-health-dashboard.html
7. https://pagespeed.web.dev/
8. https://jmeter.apache.org/
9. https://junit.org/junit5/
10. https://site.mockito.org/
11. https://www.zaproxy.org/
12. https://securityheaders.com/
13. https://www.w3.org/TR/WCAG21/
14. https://wearecolorblind.com/resources/colorblindly-colorblindness-simulator/
15. https://matomo.org/
16. https://github.com/GoogleChrome/lighthouse
17. https://www.npmjs.com/package/@datapunt/matomo-tracker-react
18. https://github.com/lumpn/unity-matomo
19. https://uxplanet.org/how-page-speed-affects-web-user-experience-83b6d6b1d7d7
20. https://www.ogc.org/standard/sensorthings/
21. https://owasp.org/www-community/OWASP_Risk_Rating_Methodology