COMPAIR

DELIVERABLE

D4.1 Solution Architecture Report

Project Acronym:	**COMPAIR**
Project title:	Community Observation Measurement & Participation in AIR Science
Grant Agreement No.	101036563
Website:	www.wecompair.eu
Version:	1.0
Date:	31.10.2022
Responsible Partner:	ATC
Contributing Partners:	DV, HHI, IMEC, SODAQ, TELR, UAEG
Reviewers:	All technical partners Andrew Stott, Karel Jedlicka

Dissemination Level:	Public	
	Confidential, only for members of the consortium (including the Commission Services)	X

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	28.06.2022	Marina Klitsi, Athanasios Dalianis	ATC	Draft ToC
0.2	07.10.2022	Athanasios Dalianis	ATC	Initial version
0.3	12.10.2022	Sylvain Renault, Oliver Schreer, Burcu Celikkol, Kris Vanherle, Christos Karelis	HHI, IMEC, TELRAAM, UAEG	Input on Section 3
0.4	14.10.2022	Athanasios Dalianis	ATC	Updates in Sections 2,3,4
0.5	19.10.2022	Gert Vervaet, Karel Jedlicka	DV, P4A	First internal review
0.6	20.10.2022	Athanasios Dalianis, Marina Klitsi	ATC	Version ready for internal review
0.7	20.10.2022 - 26.10.2022	Andrew Stott, Karel Jedlicka, All technical partners Gert Vervaet	External experts IMEC, TELR, SODA DV	Internal review
0.8	27.10.2022	Athanasios Dalianis	ATC	Final edits based on reviewers comments and suggestions
1.0	31.10.2022	Athanasios Dalianis	ATC	Final version

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Acronym	Description
API	Application Programming Interface
AQ	Air Quality
AQS	Air Quality Sensor
CaaS	Calibration as a Service
DEVA	Dynamic Exposure Visualisation Application
HTTP(S)	Hypertext Transfer Protocol (Secure)
NO2	Nitrogen Dioxide
PM	Particulate Matter
PMD	Policy Monitoring Dashboard
REST	Representational State Transfer
SSO	Single Sign On
UI	User Interface

Executive Summary

COMPAIR is a diverse project with several work packages working in parallel to achieve common objectives. To start this work with a mutual sense of direction, we needed a high-level blueprint of the overall technology development within the project. The role of this document is to provide a first version of the system architecture and act as a common point of reference throughout the project.

This deliverable is the outcome of the first task in WP4 and it presents the overall architecture of the **COMPAIR** platform in terms of the supported functionalities, the respective processes and the components that realise them. The document presents a coarse-grained collection of technical components that provide the necessary functionality and defines the principles for their integration in a manner that ensures completeness, safety, and efficiency. To this end, we propose a Service-oriented approach, where each functional unit is implemented as a stand-alone service, communicating in a standardised manner with other components.

The document sets the boundaries for the integration of the **COMPAIR** modules by presenting the context and use for the **COMPAIR** platform. As such, it builds on top of the technical project requirements and presents the way that the **COMPAIR** capabilities will be able to integrate into the operational environments of the pilots which are the early adopters of the proposed solution.

We consider the architecture to be a live document that will evolve and be aligned with the ongoing work in the project.

1. Introduction

Purpose and Scope

The objective of this document is to present the overall architecture of the COMPAIR platform in terms of the supported functionalities, the respective processes and the components that realise them. This document will serve as a reference point for the development work that will take place in WP3 and WP4.

The decisions presented in this deliverable are subject to refinements and modifications, based on the progress of the technical work packages, as well as the validation and evaluation phases of the project.

Structure of the document

This deliverable is organised as follows:

- Section 2 presents an overview of the technical specifications in terms of functional and non-functional requirements of the system.
- Section 3 provides the description of the high-level architecture of the COMPAIR platform as well as a description of the components comprising it.
- Section 4 presents the main workflows and component interactions.
- Section 5 describes the integration process, including information about the deployment of the components, the methodology, as well as the integration tools that will be used in the context of the project.
- Section 6 provides an overview of the methodology that will be used for the system validation of the platform.
- Section 7 presents the implementation plan of the system components.
- Section 8 concludes this deliverable

2. COMPAIR Technical Specifications

This section presents the initial set of system requirements, based on the user requirements identified in D2.4 Pilot Operations Plan in the form of agile epics. As the **COMPAIR** system design and implementation goes on, all these requirements will be further refined and specified in more detail.

2.1 Agile Epics

The epics refer to different personas that are the target for the user stories. They are depicted in the table. The personas are roles that can be taken up by people depending on their intent and context.

Table 1: COMPAIR personas

Persona	Description
Citizen	A member of the general public that uses the tools
Admin	Technical person that can assign people roles, do some maintenance activities around keeping systems running, checking log files, etcetc
Researcher	Expert of air quality and/or traffic. Will analyse data coming from the project, design and follow up experiments.

The next table presents in summary the list of agile epics defined in D2.4 as a reminder to the reader.

Table 2: List of epics

ID	EPICS
AllNf01	As a citizen, I want to use fast and efficient dashboards, so I can analyse situations well
AllExp	As a citizen, I want to be able to export the data from the dashboards in a number of formats, so I can share and work on the data outside the **COMPAIR** tools
AllL&f	As a citizen, I want to use pleasing, clear, consistent dashboards, so I can analyse situations well

ID	EPICS
Co2Cal	As a citizen, I want to know the current and historic contribution of my different activities to my Carbon Footprint, so I can maximise the impact of changes to my behaviour
Co2RRe	As a citizen, I want to get a list of recommendations on how to reduce my contribution to CO2 creation
Co2Man	As an admin, I want to manage the dashboards I am responsible for, so I can help my users be efficient
Co2Sce	As a citizen, I want to be able to create scenarios of citizen and government actions that show me how emissions can be reduced to a certain target.
DyDAoR	As a citizen, I want to see the output of air quality sensors that were worn on trips, so I can analyse the exposure of people to air pollution
AllDis	As a citizen, I can access information about air quality, best practices, ... from the **COMPAIR** tools and dashboards
DyDMan	As a researcher, I want to be able to manage experiments done
DisSha	As a researcher, I want to be able to share information, so my users know how to use **COMPAIR** tools efficiently
DEVAAnn	As a citizen, I can annotate and share information about exposure on my trips
DEVAHis	As a citizen, I want to get historical information about trips so I can assess the exposure
DEVARea	As a citizen, I want to get realtime information about trips so I can assess the exposure
DEVAUI	As a citizen, I want to use pleasing, clear, consistent dynamic exposure visualisation app, so I can analyse situations well
DEVAGam	As a citizen, I want to interact with the app and simulate how my actions would lead to reduced/increased pollution

ID	EPICS
DEVAMan	As a researcher, I can monitor how the app is being used so I can assess if actions need to be taken
DEVAUsI	As a citizen, I can update my settings in the app, so my characteristics, my sensor,.. is taken into account
DEVAViz	As a citizen, I want an intuitive and clear visualisation of the data
AllUMa	As a citizen, I can login to the tools, so my settings and personal info is used
PMDCom	As a citizen, I want to compare the output from different projects using the policy monitoring dashboard against each other
PMDAir	As a citizen, I want to see realtime and historical information about air pollution, so I can assess the impact of policy decisions
PMDCon	As a citizen, I want to see context data like weather, roadworks, so I can take this context into account when assessing the impact of policy decisions
PMDMap	As a citizen, I can use a map interface to see the location of sensors so I have an understanding where measurements are done
PMDTra	As a citizen, I want to see realtime and historical information about traffic, So I can assess the impact of policy decisions
PMDGam	As an admin, I can trigger behaviour using the dashboard by using gamification techniques, so I can increase take up of the dashboard
PMDMan	As an admin, I can manage dashboards during the lifecycle of projects so people can use the dashboards to assess impact of policy decisions
PMDUI	As a citizen, I get a user friendly, pleasing, intuitive UI, so I know how to use the dashboard and I'm motivated to use it

2.2 Non-functional requirements

This section defines the non-functional requirements that should apply for the **COMPAIR** platform components. The **COMPAIR** platform should satisfy a set of non-functional requirements, which will ensure the normal operation of the system and the provision of a proper environment for the desired system functionalities. The non-functional requirements are depicted in the following table. In this table, we provide a unique code ID for every requirement, a name for the requirement and its description.

Table 3: List of non-functional Requirements

ID	Requirement	Description
NF1	Performance	The **COMPAIR** system should respond in a timely manner using the predefined resources when an increase in users/datasets occurs. The exact response time depends on the action and more specifically on its criticality, e.g. a user interface should display an alert in 0.5 seconds, while a backup process can be done in two hours without issues.
NF2	Security	The **COMPAIR** system should be secured against sabotages arising from all types of attacks. Security techniques that discourage data loss and misuse of data for fraudulent acts should be utilised.
NF3	Privacy	The **COMPAIR** system should be GDPR compliant and employ the necessary mechanisms to ensure this e.g. sensitive personal data and user created content will never be published without user consent.
NF4	Scalability/Expandability	The system should be able to handle the increasing size of datasets, as well as a potentially increased number of users.
NF5	Usability	The **COMPAIR** system should have an attractive and intuitive User Interface. The interface needs to address various user groups and therefore should be easy to use and give access to all system functionalities providing easy navigation through all features.
NF6	Interoperability	The **COMPAIR** system should use communication protocols that allow its use by different systems and devices.
NF7	Multilinguality	The **COMPAIR** system should support multiple languages, at least English and the pilot languages for the relevant components.

3. COMPAIR Architecture

This section presents the initial blueprint of the COMPAIR platform architecture. A high-level architecture of the platform is described to set the stage for the development of the COMPAIR prototypes. It must be noted that the decisions presented in this section are subject to refinements and modifications, based on the progress of the technical work packages, as well as the validation and evaluation phases of the project.

3.1 High level architecture

In this section we provide a high-level overview of the platform architecture.

To tackle the non-functional requirements and technical specifications described in the previous sections, a distributed microservices based approach is proposed, where the COMPAIR components communicate with the help of a set of APIs.

The following diagram depicts the main components of the system and how these are distributed between the different servers of the system.

The COMPAIR solution architecture (Figure 1) relies on a distributed pattern, where the different components of the system are deployed to individual server nodes, each dedicated to specific tasks. In the COMPAIR project we have four server nodes, as presented in the figure below:

1) the **Traffic and Air Quality Sensor platform nodes** which provide traffic and air quality data based on their sensor readings,depicted with the dark yellow and light orange boxes,
2) the **Air Quality Calibration as a Service platform node (AQ CaaS)**, that integrates and calibrates air quality data from SODAQ, external sensors and data from reference stations, depicted with the dark orange box,
3) and the **Data platform node**, depicted with the big green box, which hosts the necessary mechanisms for data collection and transformation, user management and visualisation (light green boxes).

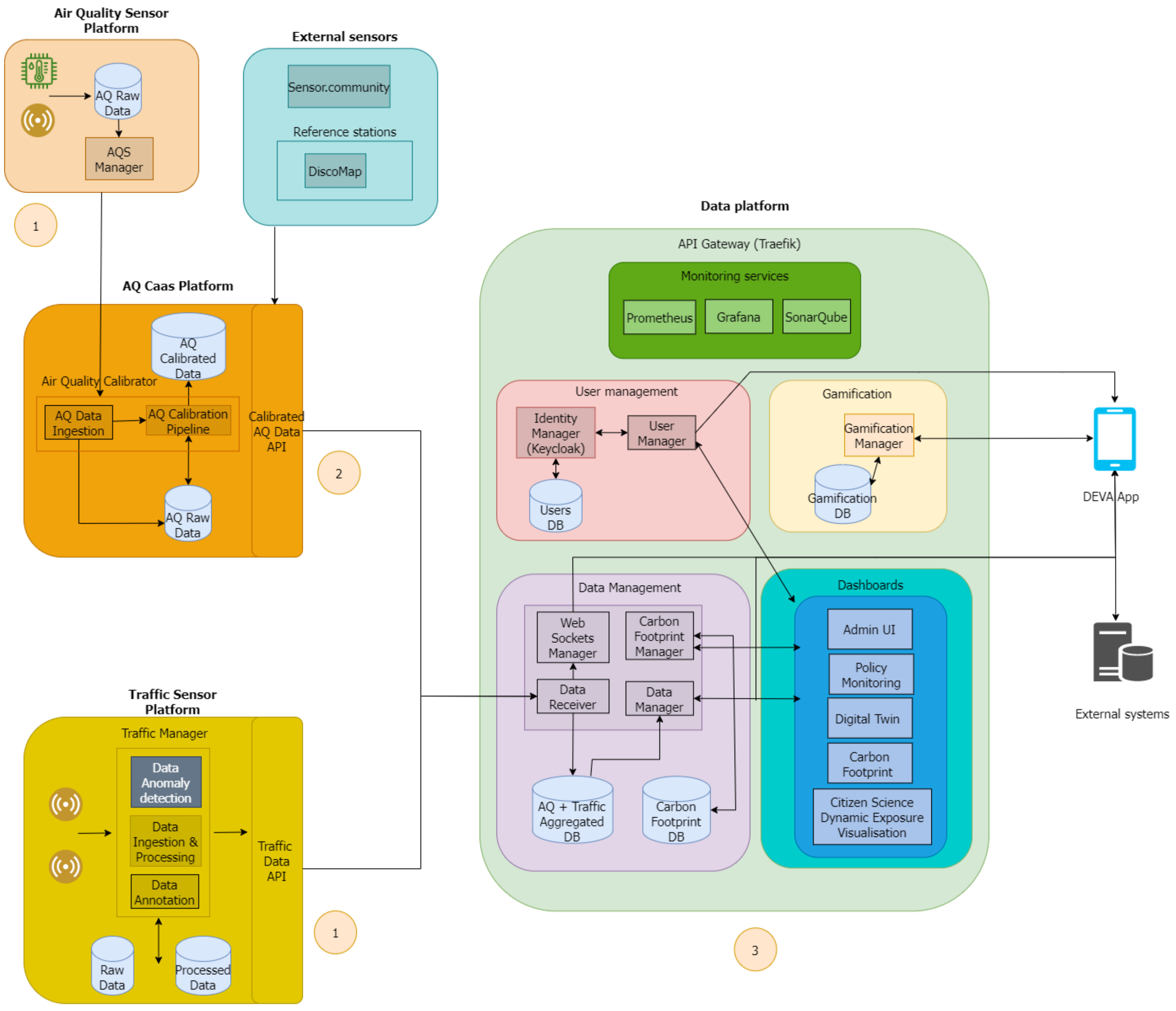


Figure 1: The COMPAIR architecture

3.2 Components Description

This section analyses the components comprising the **COMPAIR** platform. These components are presented below grouped, based on the node they will be installed.

3.2.1 AQ Sensor Platform

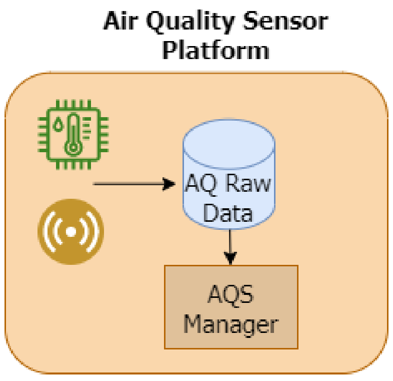


Figure 2: Air Quality Sensor Platform

The AQ Sensor Platform is responsible for gathering the air quality data provided by the static and mobile SODAQ sensors. Its main component is the AQS Manager which collects the data and stores them in a database, exposes an API providing air quality data and sensor metadata and also pushes the data to the Air Quality CaaS Platform through a REST API in real time. A copy of the data is kept and anonymised, with the sensors data shown on the knowyourair.net map.

3.2.2 Traffic Sensor Platform

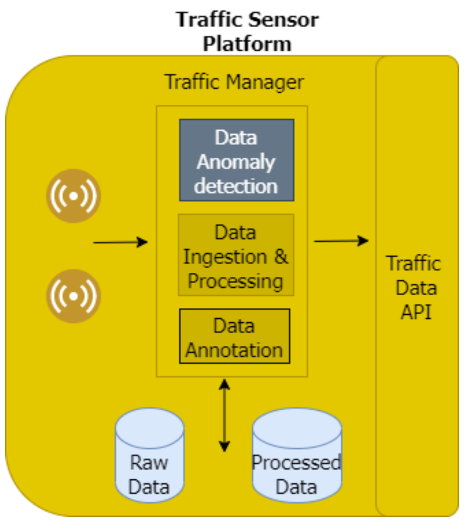


Figure 3: Traffic Sensor Platform

The Traffic Sensor Platform is responsible for gathering the traffic data from the static sensors provided by Telraam. Its main component is the Traffic Manager that is responsible for the data ingestion and processing, the data annotation and the detection of potential data anomalies. The data is stored in a database and is sent to the Data Receiver at the main node.

The Traffic data API is available as a stand-alone application as well, accessible via www.telraam-api.net. Any Telraam data used in the **COMPAIR** project is ingested by the **COMPAIR** data management components and used in the **COMPAIR** dashboards.

3.2.3 AQ CaaS Platform

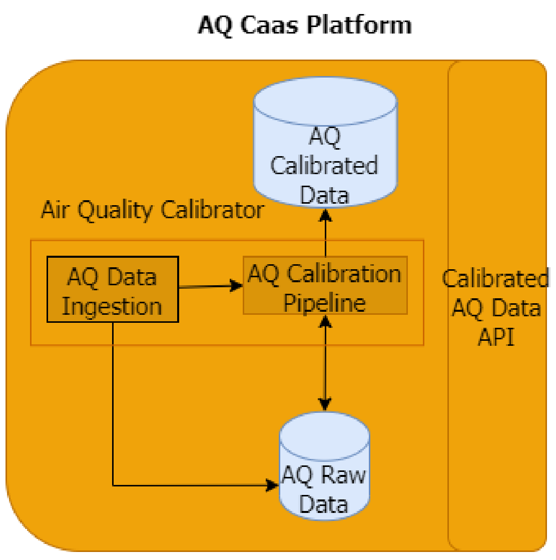


Figure 4: Air Quality Calibration as a Service Platform

The AQ CaaS Platform is responsible for ingesting and processing air quality data provided by the SODAQ sensors and other sources like reference stations. The data from SODAQ is sent to the AQ CaaS platform through a REST API provided by the Air Quality Calibrator which maps the data into a standardised format that complies with the Open Geospatial Consortium SensorThings API format adds it to the ingestion pipeline, processes, cleans and calibrates and finally stores it in a time series database. Then at frequent times it sends the data to the Data Receiver at the main COMPAIR node.

3.2.4 Data Platform

The Data platform is the main node of the **COMPAIR** system and hosts the necessary components for data integration and harmonisation, user management, gamification and monitoring services, as well as data visualisation. It gathers the data from the previously described platforms, harmonises the data to a common format and feeds it to the mobile application and the web dashboards of the system.

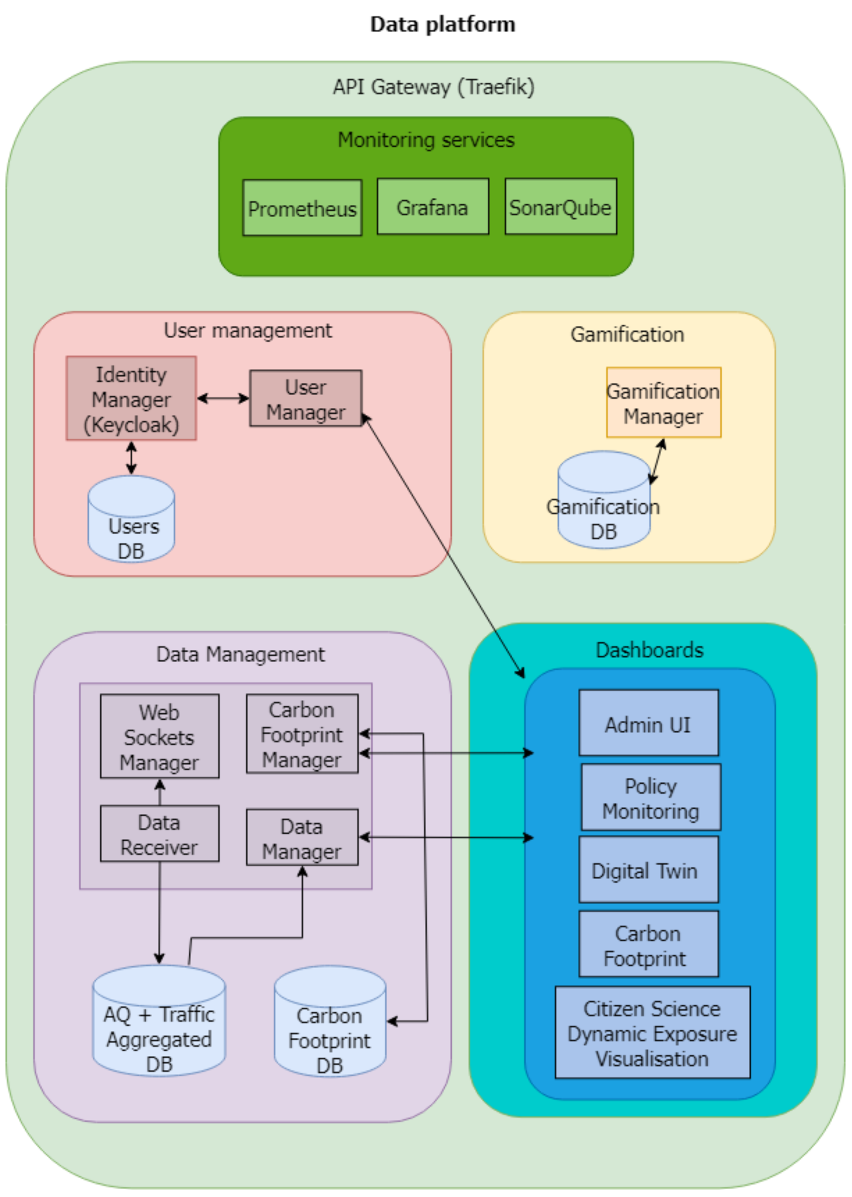


Figure 5: Data platform

It's main components are in summary:

API Gateway

In microservices architectures, where many services are deployed in several virtual or physical nodes and multiple instances of the same service can exist, an API Gateway is a necessity.

An API Gateway is a component that intervenes between a web client and the backend APIs, acting as a reverse proxy that forwards the request to the appropriate microservice, usually after proper authorization.

The API Gateway thus provides a single point of access to UIs and in general external applications, decreases the complexity of implementation and allows security measures, and functionalities such as load balancing and service discovery to be applied more easily.

For the purposes of the project we are going to use Traefik [1].

Identity Manager

The Identity Manager is responsible to store the user account data and to provide authentication and authorization services to the platform. The realisation of the Identity Manager will be based on Keycloak.

Keycloak [2] is an open-source identity and access manager for application and services, that provides several features for the project like centralised management, standard protocols like OAuth 2.0, SAML 2.0 etc, social login, single sign on (SSO) etc, giving us a variety of options to tackle the project needs in this area.

User Manager

The role of the User Manager can be summarized as follows:

- It provides authentication and authorization mechanisms based on Json Web Tokens
- It allows for user registration and user profile edit
- It allows for role definition and user assignment
- It allows for permissions definition and user groups / users assignment

The User Manager provides both APIs and a UI for managing the related entities and utilises the Identity Manager providing this way an abstraction layer in front of it.

Data Management

The Data Management components is of a set of components responsible of the following:

- The Data Receiver exposes an API in order for the other platforms to push their data to. The data is transformed to the Sensorthings OGC standard [3] when needed, e.g. the air quality data already arrives in this structure while the Telraam data is converted into it.
- Transfer real time information to the Augmented Reality app DEVA and the dashboards through web sockets (Web Sockets Manager component).
- Collect carbon consumption related data from the users, evaluate their footprint and suggest ways to reduce it, as well as collect user generated scenarios for participation in policy making (Carbon Footprint Manager component)
- Tackle the needs of the users in terms of the aggregated data (Data Manager component). The Data Manager exposes a set of APIs that is consumed by the dashboards and the DEVA app.

Gamification Manager

Gamification is the application of techniques found in games to a non game context. The gamification techniques take advantage of the intrinsic and extrinsic human motivations in order to increase the user's participation in certain aspects of a software application.

The **COMPAIR** Gamification Manager will monitor the user actions in the **COMPAIR** mobile application and the events produced in the system, and implement the necessary game mechanics that will increase the user's satisfaction with the system and engage him/her more on its functionalities.

The exact usage of this component will be further investigated during the course of the project and the evolving epics and user stories.

Dashboards

The **COMPAIR** system will offer a set of dashboards in order to tackle the user and technical requirements of the project.

These in summary include:

- The **Policy Monitoring Dashboard (PMD)** which helps users to understand and compare how environmental situations change under different actions.
- The **Carbon Footprint Simulation Dashboard (CO2)** which is designed to support specific experiments around carbon footprints or indoor footprint for any chosen air molecule.
- The **Citizen Science Dynamic Exposure Visualisation Dashboard** which will be used to show both city and CS data (with a GIS identifier) on a map and in various charts.
- The **Digital Twin Dashboard** where generated ideas for new policies will be able to be simulated and reviewed in a systematic manner against other policies.
- The **Admin UI** through which a system administrator can handle and monitor resources of the system e.g. users

More information about the **COMPAIR** dashboards can be found at **D3.4 Dashboards Design.**

3.2.5 Dynamic Exposure Visualisation application (DEVA)

The **COMPAIR** system will also offer an Augmented Reality app, the so-called Dynamic Exposure Visualisation App (DEVA). The aim is to enable people to explore their surroundings via their smartphone or tablet camera, so they see a visual overlay of environmental information such as air quality or traffic information. Hence, the DEVA app will be the link between citizen science environmental sensors provided by the project, public environmental data, CS experiments and the users.

More information about the **COMPAIR** Augmented Reality app can be found at **D3.3 AR design.**

4. Components Interaction

In this section, the architecture is described through block/sequence diagrams stressing the interactions between the different components that compose the **COMPAIR** solution in the main workflows of the system. The workflows below are indicative and may change during the course of the project as the implementation and integration of the components progresses.

Air Quality sensor data collection and processing

This workflow describes the process of gathering the air quality sensor data, its processing and transfer to the components of the main platform and the AR application.

- Initially the AQS Manager receives and stores the sensor data locally in real time.
- At frequent times, it sends the data to the Air Quality Calibrator through a REST API provided by the AQ Manager.
- The AQ Calibrator processes and calibrates the data, stores them in the AQ database and sends them to the Data Receiver in the main node by making a POST request to the Receiver's API.
- The Data Receiver saves the new data, notifies the Web Sockets Manager and updates the statistics about the sensor data used for historical information, for optimisation purposes.
- The Web Sockets Manager forwards the new sensor data to the Dashboards and DEVA through dedicated web sockets topics.

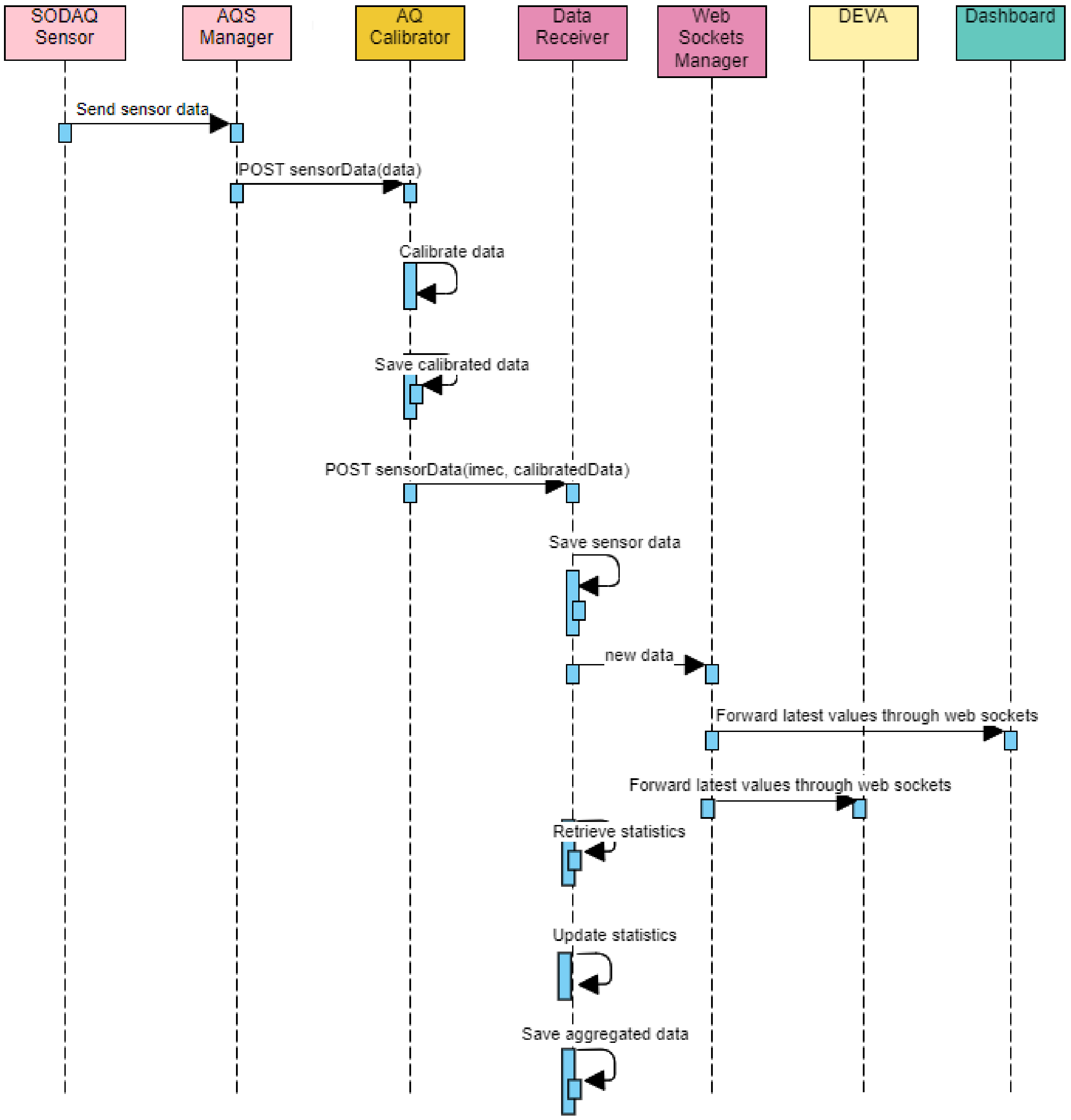


Figure 6: Air Quality sensor data collection and processing

Traffic sensor data collection and processing

This workflow describes the process of gathering the traffic sensor data, its processing and transfer to the components of the main platform and the AR application. The process is similar to the one described for the air quality data, indicating that there is a common approach for both types of sensor data. The same approach will need to be followed from future data providers in order to connect to the main platform.

- Initially the Telraam Manager receives, processes and stores the sensor data locally in real time.
- At frequent times, it sends the data to the Data Receiver through a REST API provided by the Receiver.

- The Data Receiver, transforms the new data based on the OGC standard, saves it to the database, notifies the Web Sockets Manager and updates the statistics about the sensor data used for historical information, for optimisation purposes.
- The Web Sockets Manager forwards the new sensor data to the Dashboards and AR application through dedicated web sockets topics.

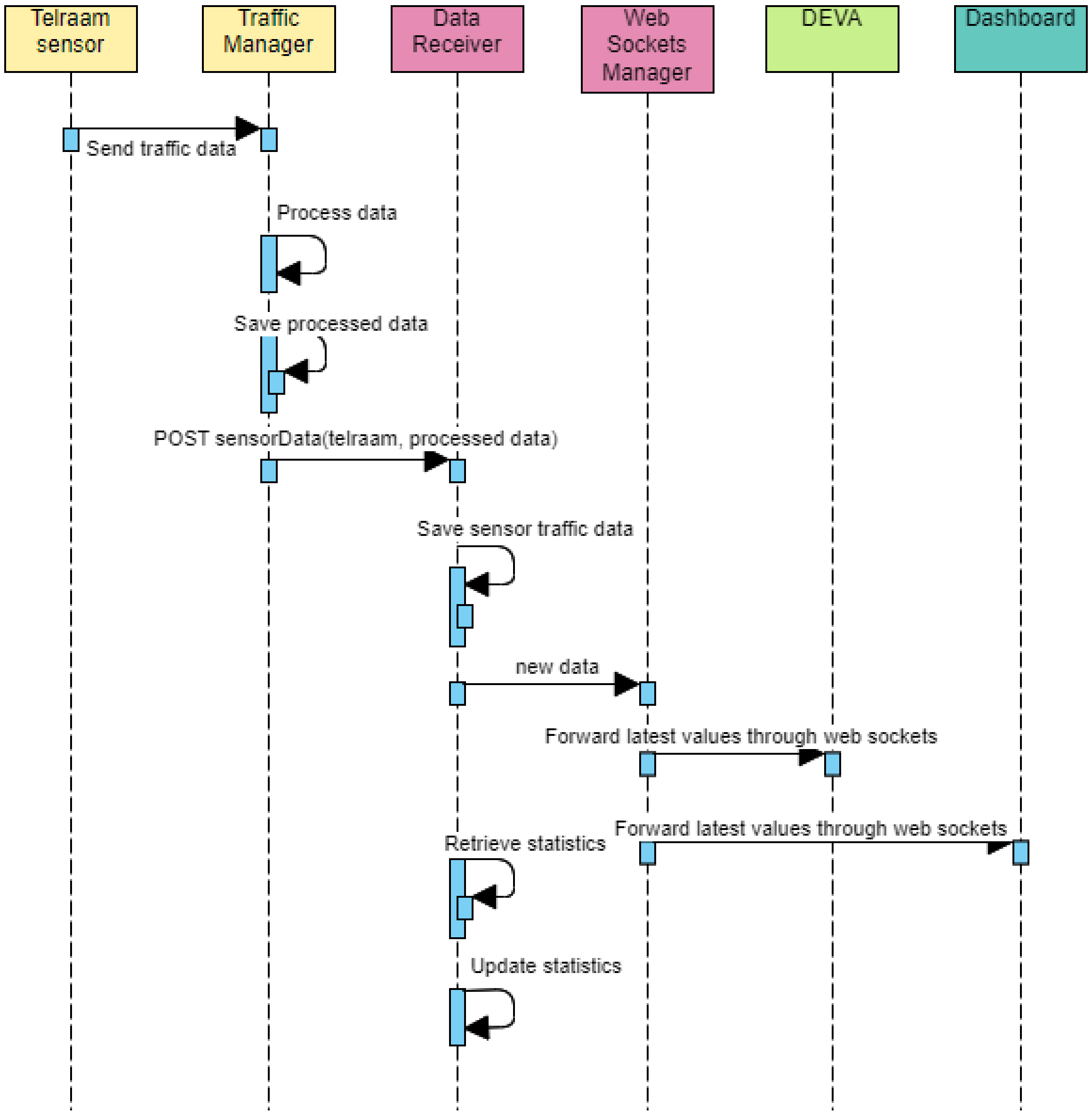


Figure 7: Traffic data collection and processing

Historical data in the Policy Monitoring Dashboard

The PMD will provide to the user the option to search for historical data for a specific sensor or group of sensors. In the following sequence diagram we present the communication between the relevant components in order for this functionality to be realised.

- Initially the user select the type of sensor e.g. PM2.5, PM10, NO2 etc. the time span, e.g. the last hour, day, week etc, the cell representing a group of sensors based on location and finally a sensor from this group

- The PMD asks the Data Manager for the statistics through a REST API call.
- The Data Manager retrieves this information from the database and returns it to the PMD
- The PMD presents to the user a chart depicting the evolution of the sensor values for the selected time period.

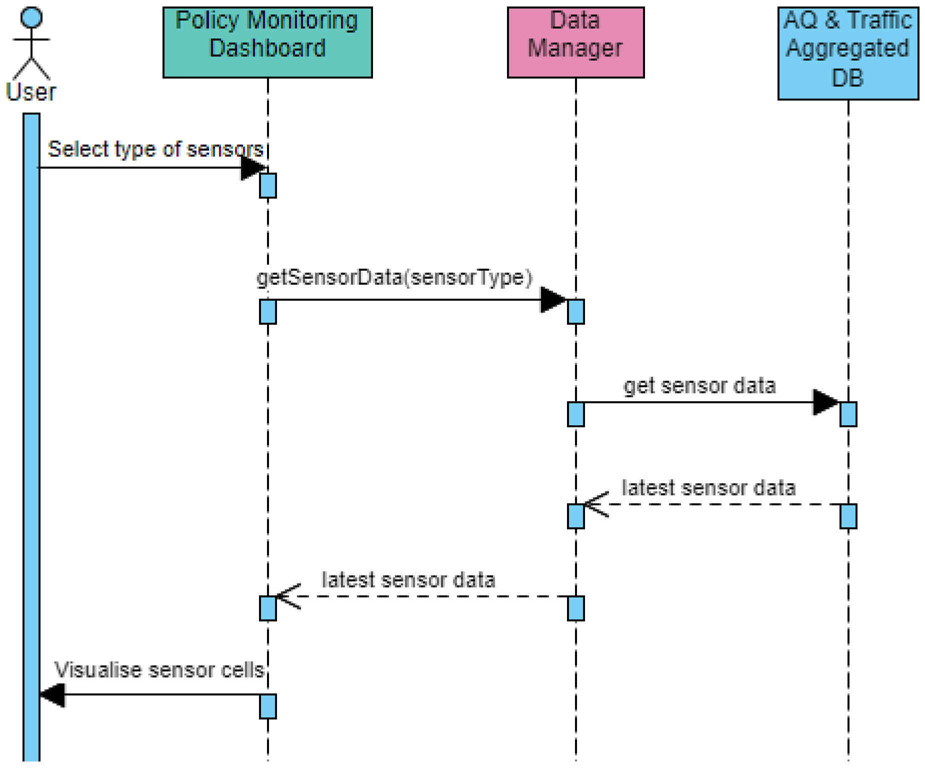


Figure 8: Historical data in PMD

Sensor data in the DEVA

The Data Manager will provide sensor related data to the DEV application in a similar way as the PMD. The following diagram depicts the communication between the relevant components in order for this functionality to be realised.

- As the user moves in the city, views information related to the pollutants. At any point, he /she can request for more information about sensors around him / her by clicking on the relevant icon.
- The DEVA requests the sensor data from the Data Manager through an API call.
- The Data Manager gets the latest values and metadata from the database and sends them back to DEVA.
- DEVA visualises the data and presents it to the user.

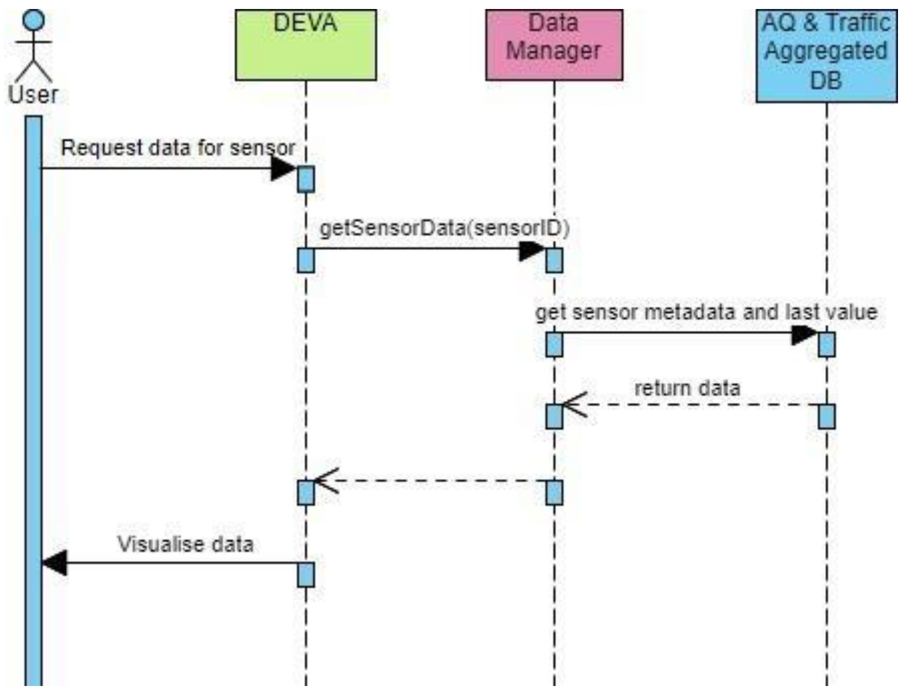


Figure 9: Sensor data in the DEVA

CO2 Dashboard - Calculation of CO2 Footprint

In the following diagram we present the communication between the components of the system in order to allow the user to calculate his / her CO2 footprint.

- Initially the user logs in to the platform through the Container UI, the UI that includes the CO2 Dashboard among other UIs. The user provides his / her credentials and the Container UI sends the credentials to the User Manager.
- The User Manager checks the credentials with Identity Manager (Keycloak) and after successful confirmation it sends the user token to the UI.
- The user navigates to the CO2 Dashboard main page and selects to calculate the CO2 footprint.
- The user fills in information such as demographic data, information about his car, building and travels etc.
- The data is sent to the Carbon Footprint Manager, which calculates the user’s footprint and generates recommendations for improving it.
- The calculation and recommendations are sent to the user.

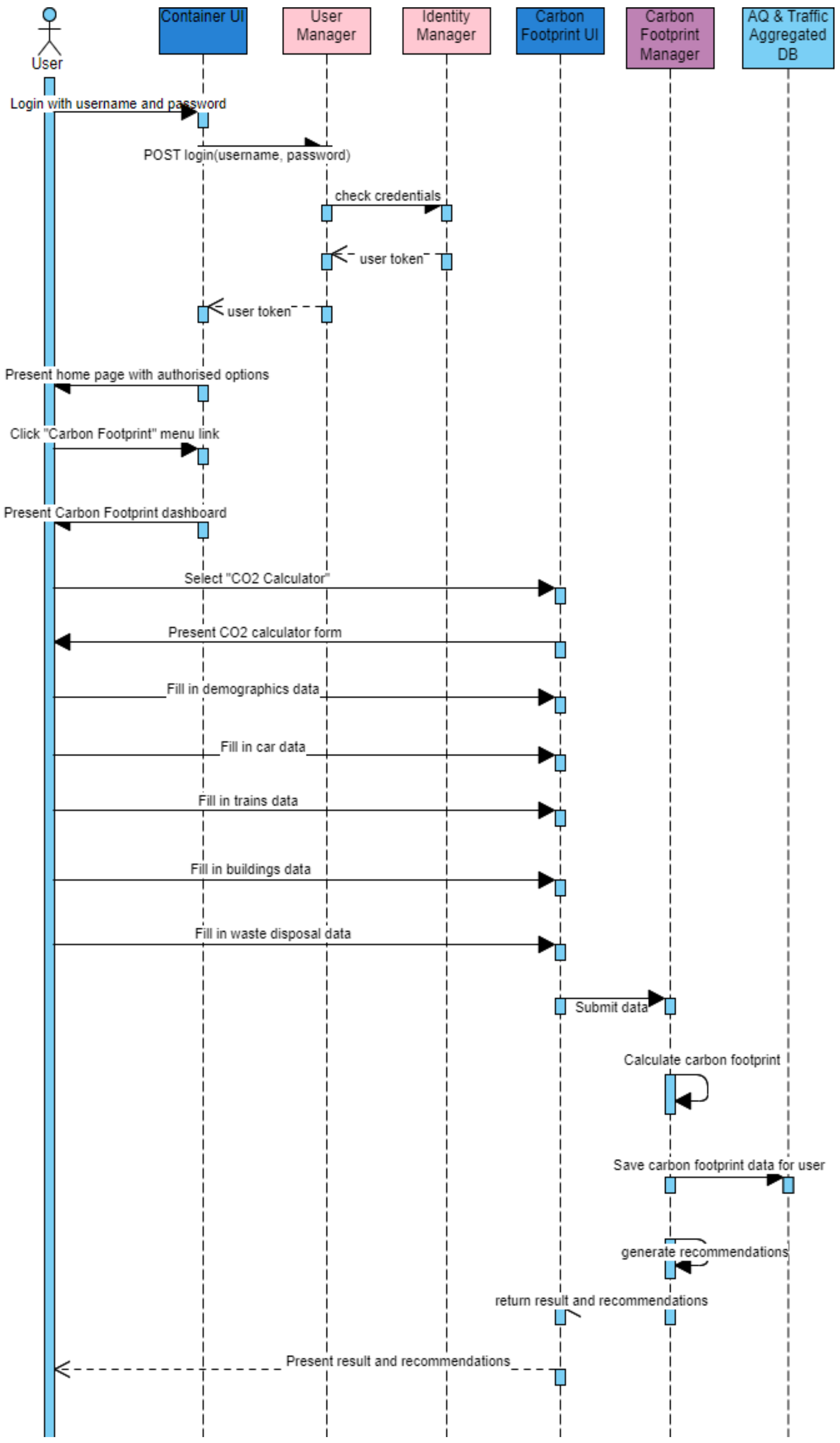

User
Container UI
User Manager
Identity Manager
Carbon Footprint UI
Carbon Footprint Manager
AQ & Traffic Aggregated DB
Login with username and password
POST login(username, password)
check credentials
user token
user token
Present home page with authorised options
Click "Carbon Footprint" menu link
Present Carbon Footprint dashboard
Select "CO2 Calculator"
Present CO2 calculator form
Fill in demographics data
Fill in car data
Fill in trains data
Fill in buildings data
Fill in waste disposal data
Submit data
Calculate carbon footprint
Save carbon footprint data for user
generate recommendations
return result and recommendations
Present result and recommendations

Figure 10: CO2 Dashboard - Calculation of CO2 Footprint

5. Integration process

For the integration purposes of the **COMPAIR** project we are following the Agile Software Development Practices with frequent integration cycles, rapid prototyping, and close collaboration between self-organising, cross-functional teams. Based on agile principles, we intend to apply Continuous Integration techniques for performing automated building, testing and deployment of the provided components. For these purposes we have set up a development environment containing a set of continuous integration and deployment tools for the components of the main platform.

5.1 Deployment

As mentioned in Section 3, in the context of the project, several servers will be employed, each with a specific role in the whole system. Essentially these servers constitute a fog architecture, since some nodes act as “fog” nodes where the collection of the raw data and a first level of its processing takes place, while the main node (data platform) acts as the “cloud” one where the aggregated data are collected, stored and presented to the user in a common integrated environment.

Since the architecture is service oriented, the communication of the fog nodes with the cloud one will be done through the component APIs, using the HTTPS protocol. In every communication an authentication token will be used in order to increase further the level of security

In the following subsections we provide an initial list of hardware and software requirements of the components, as well as the deployment tools that will be used.

5.1.1 Hardware and software Requirements

The following table summarises an initial estimation of the requirements the components have in terms of hardware, as well as their software dependencies, as these were derived from the discussions with the technical team of the project.

Table 4: List of hardware and software requirements

Component	RAM	CPU	Disk Space	Software Dependencies
Air Quality Sensor Platform				
AQS Manager	AWS Fargate, RDS (PostgreSQL)			
Traffic Sensor Platform				

Component	RAM	CPU	Disk Space	Software Dependencies
Traffic Manager	AWS Lambda functions, Python, PostgreSQL			
Air Quality CaaS Platform				
Air Quality Calibrator	16GB	8 cores	300GB	Azure, .net, TimeSeriesDB
Data Platform				
Identity Manager (KeyCloak)	2GB	2 cores	2GB	PostgreSQL, Docker
User Manager	2GB	2 cores	1GB	Spring Boot, Keycloak, Docker
Data Manager	4GB	4 cores	40 GB	Spring Boot, MongoDB, Docker
Gamification Manager	4GB	2 cores	2GB	Spring Boot, MongoDB, Docker
API Gateway (Traefik)	2GB	2 cores	200MB	Docker
Monitoring services like Prometheus, Grafana, SonarQube	8 GB	4 cores	30GB	PostgreSQL, Docker
Augmented Reality Application				
DEVA Augmented Reality app for mobile phones and tablets with **Android**	4GB	4 cores	300MB	ARCore from Android, Android OS up to version 8 ‘oreo’. Recommended Android 10 or higher.

Component	RAM	CPU	Disk Space	Software Dependencies
	AR compatible models only. More info at: https://developers.google.com/ar/devices			
DEVA Augmented Reality app for mobile phones and tablets with **iOS**	4GB	4 cores	300MB	ARKit from Apple, requires iOS 11.0 or later and an iOS device with an A9 or later processor.
	AR compatible models only. More info at: https://developer.apple.com/documentation/arkit/verifying_device_support_and_user_permission			

5.1.2 Deployment Tools

For component isolation and easy deployment, we are going to use Docker [4] whenever possible, that is, the components developed for the main **COMPAIR** platform will have to be dockerized mandatorily while the components deployed in the other nodes of the system optionally.

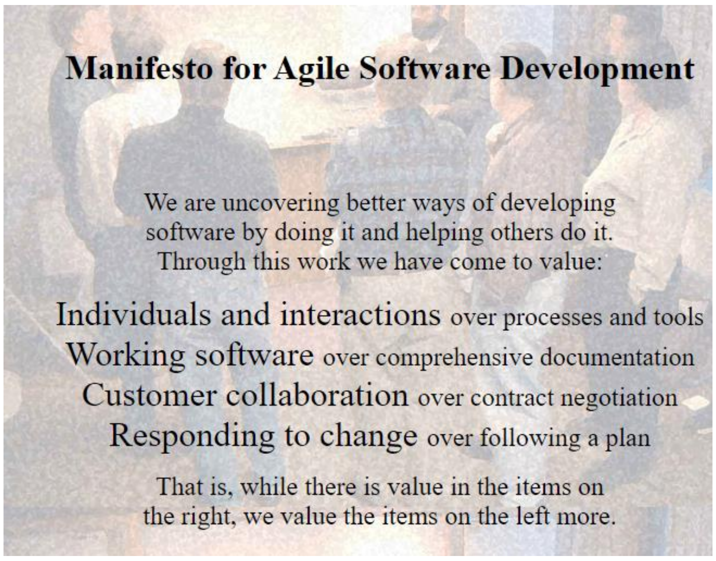
Docker packages and runs applications in Docker images. A Docker image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. Docker images are stored in a Docker registry and in order to run them in virtual or physical machines, the machines need to have installed the Docker Engine software. The running instance of a Docker image is called a Docker container. A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.

For orchestrating and monitoring the Docker containers of the projects we will use Docker compose [5]. Compose is a tool for defining and running multi-container Docker applications. With Compose, we use a YAML file to configure the application's services. Then, with a single command, we create and start all the services defined in the configuration.

5.2 Agile Methodology

A research among the most dominant development methodologies [6] indicates that the most appropriate way of implementing integration mechanisms for the **COMPAIR** platform would be 'Rapid Application Development'. This implies that a system prototype is implemented, tested, and evaluated in an iterative manner, using short cycles to add functionality to the prototype. This is more suitable for an Innovation action project aiming to deliver a system prototype, since it enables end users to continuously participate in the development of the integration mechanisms and guide the development towards their needs.

In this manner, the processes of implementation and definition of the integration mechanisms will proceed in parallel until the end of the project by means of close collaboration between all the teams. One of the most popular types of Rapid Application Development is the 'Agile Methodology', which is associated with a list of terms and rules that must be followed during development as described in the 'Agile Manifesto' [7].



Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Figure 11:Agile Manifesto

Agile methodology implies and enforces collaboration between self-organising, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change.

The methodology workflow could be reflected in the following diagram:

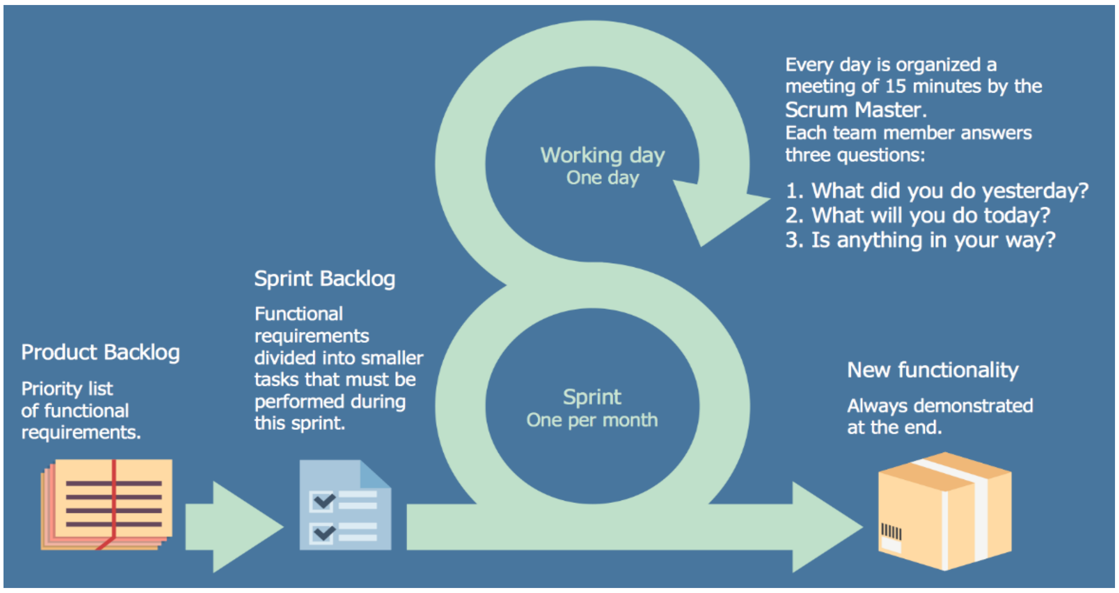

Working day
One day
Every day is organized a meeting of 15 minutes by the Scrum Master.
Each team member answers three questions:
1. What did you do yesterday?
2. What will you do today?
3. Is anything in your way?
Sprint Backlog
Functional requirements divided into smaller tasks that must be performed during this sprint.
Product Backlog
Priority list of functional requirements.
Sprint
One per month
New functionality
Always demonstrated at the end.

Figure 12: Agile methodology workflow

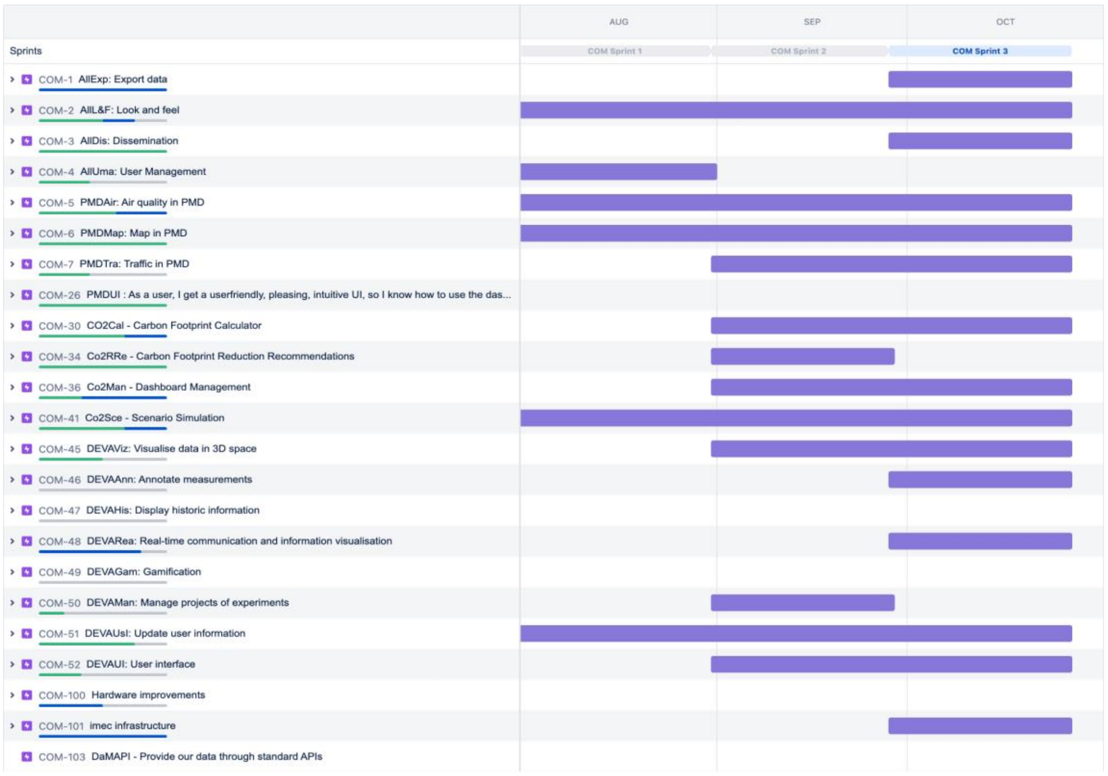
5.2.1 Task management

In order to coordinate the technical efforts between the technical partners and break down the user requirements into technical tasks, in compliance to the agile principles, we have set up a **COMPAIR** space on Jira Cloud [8].

Jira is a tool that provides an easy way to create epics, user stories and tasks, to plan agile sprints and assign work to the agile teams, and also to keep track of the progress done.

For the purposes of the **COMPAIR** project:

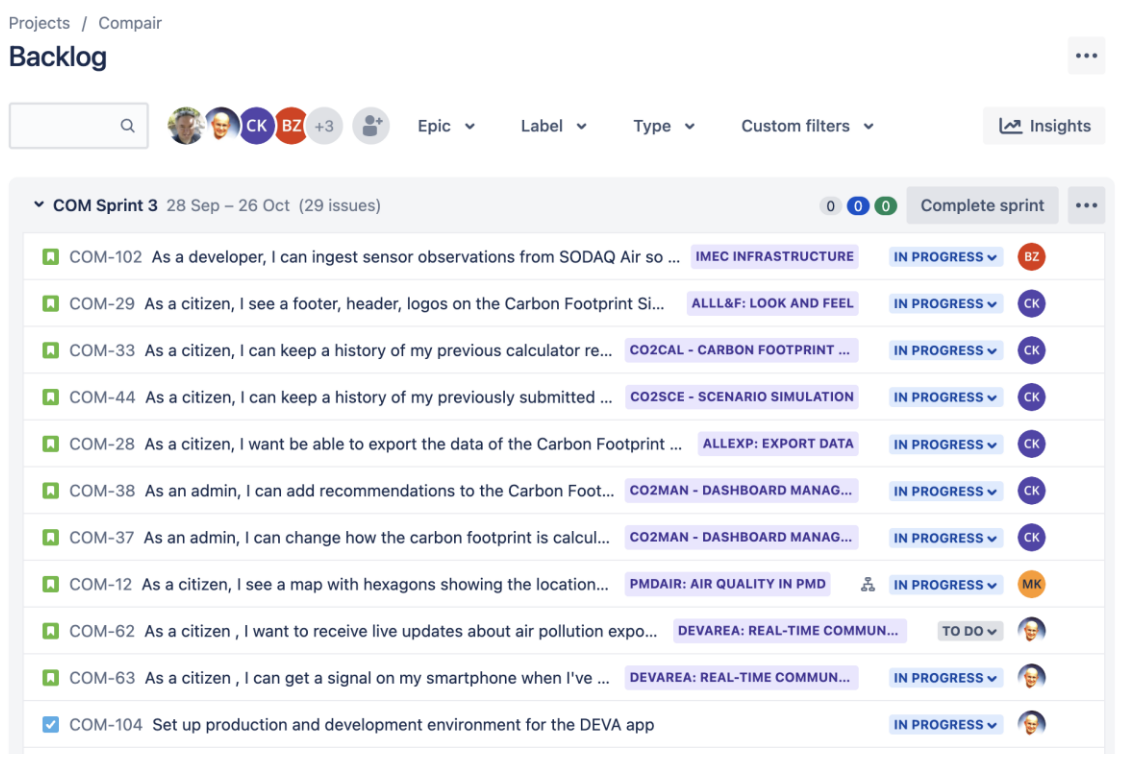
- Epics, user stories and tasks are being set (for the Alpha version this is completed already). These are being refined frequently based on the users' feedback.
- One Product Owner per component has been assigned.
- Monthly sprint planning sessions take place, where each sprint contains the prioritised tasks that can be implemented in the given time, after discussion with all the technical teams.
- Monthly sprint reviews take place where the results of each sprint are presented to the pilots
- Bi-weekly technical sessions take place, where the progress of the tasks is being reported and technical matters are discussed.



Sprints

- COM-1 AllExp: Export data
- COM-2 AllL&F: Look and feel
- COM-3 AllDis: Dissemination
- COM-4 AllUma: User Management
- COM-5 PMDAir: Air quality in PMD
- COM-6 PMDMap: Map in PMD
- COM-7 PMDTra: Traffic in PMD
- COM-26 PMDUI : As a user, I get a userfriendly, pleasing, intuitive UI, so I know how to use the das...
- COM-30 CO2Cal - Carbon Footprint Calculator
- COM-34 Co2RRe - Carbon Footprint Reduction Recommendations
- COM-36 Co2Man - Dashboard Management
- COM-41 Co2Sce - Scenario Simulation
- COM-45 DEVAViz: Visualise data in 3D space
- COM-46 DEVAAnn: Annotate measurements
- COM-47 DEVAHis: Display historic information
- COM-48 DEVARea: Real-time communication and information visualisation
- COM-49 DEVAGam: Gamification
- COM-50 DEVAMan: Manage projects of experiments
- COM-51 DEVAUsI: Update user information
- COM-52 DEVAUI: User interface
- COM-100 Hardware improvements
- COM-101 imec infrastructure
- COM-103 DaMAPI - Provide our data through standard APIs

Figure 13: Jira roadmap view



Projects / Compair

Backlog

Epic · Label · Type · Custom filters · Insights

COM Sprint 3 28 Sep – 26 Oct (29 issues) 0 0 0 Complete sprint

COM-102	As a developer, I can ingest sensor observations from SODAQ Air so ...	IMEC INFRASTRUCTURE	IN PROGRESS	BZ
COM-29	As a citizen, I see a footer, header, logos on the Carbon Footprint Si...	ALLL&F: LOOK AND FEEL	IN PROGRESS	CK
COM-33	As a citizen, I can keep a history of my previous calculator re...	CO2CAL - CARBON FOOTPRINT ...	IN PROGRESS	CK
COM-44	As a citizen, I can keep a history of my previously submitted ...	CO2SCE - SCENARIO SIMULATION	IN PROGRESS	CK
COM-28	As a citizen, I want be able to export the data of the Carbon Footprint ...	ALLEXP: EXPORT DATA	IN PROGRESS	CK
COM-38	As an admin, I can add recommendations to the Carbon Foot...	CO2MAN - DASHBOARD MANAG...	IN PROGRESS	CK
COM-37	As an admin, I can change how the carbon footprint is calcul...	CO2MAN - DASHBOARD MANAG...	IN PROGRESS	CK
COM-12	As a citizen, I see a map with hexagons showing the location...	PMDAIR: AIR QUALITY IN PMD	IN PROGRESS	MK
COM-62	As a citizen , I want to receive live updates about air pollution expo...	DEVAREA: REAL-TIME COMMUN...	TO DO	
COM-63	As a citizen , I can get a signal on my smartphone when I've ...	DEVAREA: REAL-TIME COMMUN...	IN PROGRESS	
COM-104	Set up production and development environment for the DEVA app		IN PROGRESS	

Figure 14: Jira backlog view

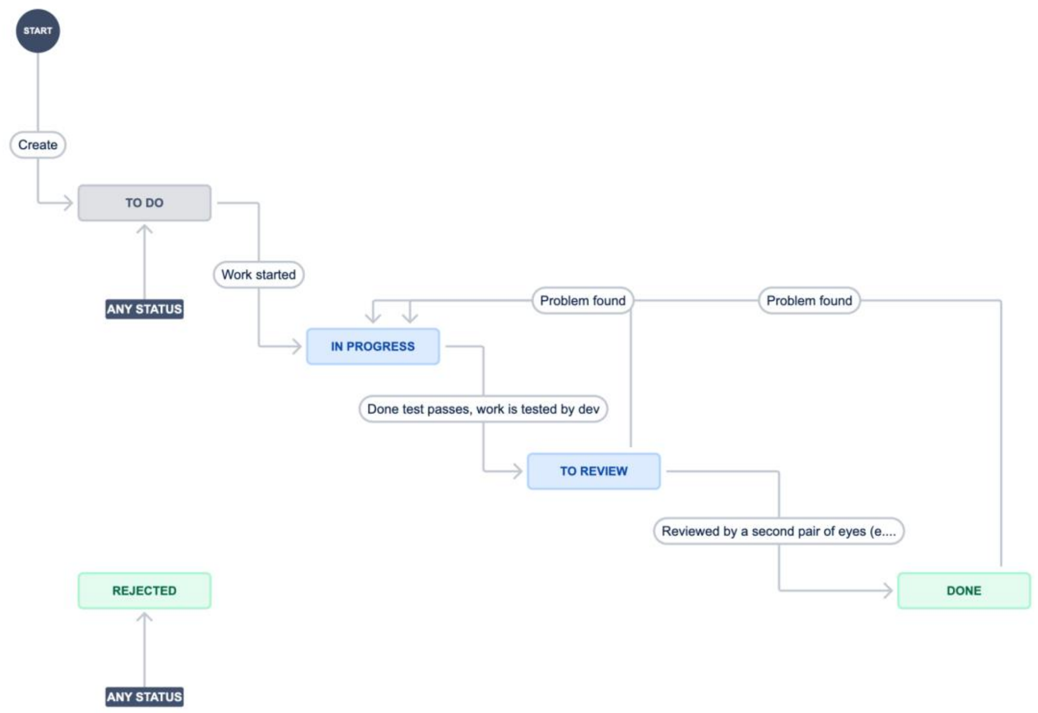


Figure 15: Development process

5.3 API Guidelines

Some of the components developed in **COMPAIR** will have APIs exposed. It is important that these APIs follow best practices for better understanding and communication. The minimum API guidelines that need to be followed are summarised in the following list:

- All APIs will use the JSON / REST protocol, over HTTP for internal APIs or HTTPS for publicly available ones
- APIs must return the appropriate HTTP status codes based on status code definitions [9]
- APIs must use the correct service methods [10] for their operations
- APIs must be cacheable when possible
- APIs must support versioning
- APIs must support security measures such as authorization headers
- APIs must support pagination
- APIs must be documented using OpenAPI

5.4 Integration Tools

In this section we present the tools that can be used for the integration of the components of the **COMPAIR** system. An overview of these tools is presented in the table below.

Table 5: List of integration tools

Category	Tool
Task management	Jira
Component packaging	Docker
Component orchestration	Docker compose
Component images repository	Canister
Code repository	GitLab
Component deployment	Jenkins pipelines
Code Testing	JUnit, Mockito, Mocka, Jest, Nose
Monitoring	Prometheus, Grafana
Code Quality	SonarQube

5.4.1 CI/CD

For the purposes of Continuous Integration and Deployment regarding the components of the main node, we are using the GitLab pipelines feature. Pipelines are the top-level component of continuous integration, delivery, and deployment and are composed of Jobs that define what needs to be done and Stages that define when the jobs must run.

The Jobs of each Stage can be executed in parallel while the Stages can only be completed sequentially. If all the Jobs of a Stage complete successfully, then the pipeline proceeds to the next Stage. If a Job fails, then the whole pipeline fails.

The pipelines are defined in specific files (GitLab-ci.yml), that are stored in the root folder of the code repository and involve the creation of integration parameters on the administration pages of GitLab.

The **COMPAIR** code projects will have to run the pipeline depicted in Figure 16 and has the following Stages:

1. Build the code. This can be considered for example the equivalent of mvn build or npm build in Java and Node JS respectively
2. Run the unit and integration tests defined in the code project. If one of the tests fails, the pipeline fails
3. Produce the quality metrics and push them to the project's SonarQube for further evaluation
4. Create the Docker image of the component and push it to the relevant Docker image registry
5. Deploy the component to the project's servers and run docker-compose

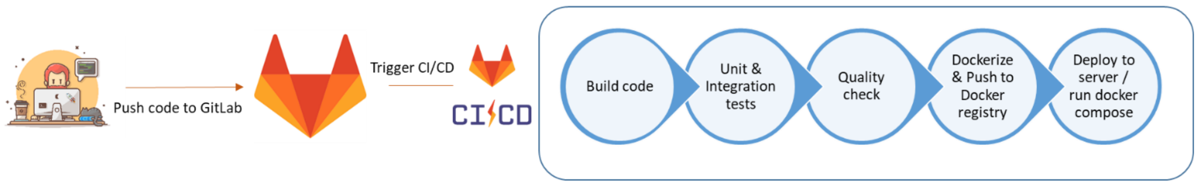


Figure 16: CI/CD Process

5.4.2 Testing and code quality

The system validation methodology is described in more detail in Section 6 System validation of the current document. In the current subsection we present some of the tools that can be used to retrieve the measurements that will be defined based on the methodology metrics.

All components developed for **COMPAIR**, will include unit and/or integration tests, to promote a higher level of code quality.

Unit tests are automated tests that check if a small part of the application, known as unit, behaves as it is intended to. In unit testing, any dependencies the unit has are replaced by "mock" units, that is, units that just return a defined response without implementing any actual functionality.

Integration tests check the behaviour of not only one unit, but a group of units that work together for the completion of a specific functionality. All the dependencies, in this case, including external ones like databases, are real and the tests are performed with test or even real data.

Various tools exist for implementing unit and integration tests, in all popular programming languages like:

- Junit [11], Mockito [12] for Java
- Karma [13], Mocha [14], Chai [15] for Node JS
- Unittest [16], nose [17] for Python
- Jest [18], Mocha for React JS etc.

Besides the unit and integration tests, the **COMPAIR** APIs will undergo stress tests to measure their performance under load. A tool that can be used for this purpose is JMeter.

JMeter [19] is designed to load test functional behaviour and measure performance of web applications and web services by defining a set of Web Services Test Plan, which include information like the parameters of the service, the number of concurrent users, the time frame etc.

In order to have a more reliable and globally accepted measure of code quality, for the various quality metrics defined in the validation methodology, the popular SonarQube a quality gateway will be used.

SonarQube [20] 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.

5.4.3 Monitoring

In systems like **COMPAIR**, it is important to ensure that the different system element services are running smoothly. To this end, the overall performance of the system needs to be constantly monitored and actions to be taken by a system administrator in case of performance degradation.

The **COMPAIR** system should be in general cloud agnostic and since during the project more performance metrics may be defined, additional monitoring tools should be deployed in the platform. Some of these tools can be Prometheus [21] and Grafana [22].

Prometheus 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 dash boarding support and can collaborate with a wide range of tools like Docker, Kubernetes and Grafana.

Grafana is 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.

6. System validation

In this section we provide an overview of the methodology that will be used for the system validation of the platform. More specifically we present the ISO/IEC 25010:2011 [23] 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 end product.

6.1 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 the above 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.

6.1.1 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 pertains to the notion of 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.

6.1.2 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:

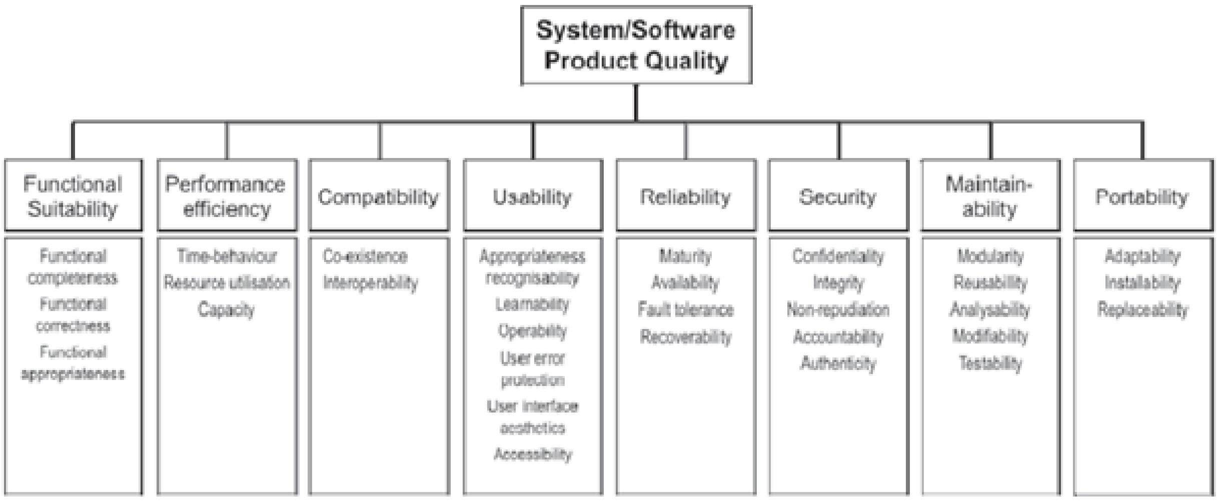


Figure 17: The ISO/IEC 25010:2011 system/software quality model characteristics

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.

6.2 Designing a quality model

As we have seen, a quality model is the cornerstone of a product quality evaluation system. It determines which quality characteristics will be considered when evaluating the properties of a software product.

6.2.1 Understanding the product quality model

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 categorizes the product quality into characteristics and sub-characteristics, as defined below.

6.2.2 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.

6.2.3 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.

6.2.4 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.

6.2.5 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.

6.2.6 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 authorized to have access.
- Integrity - Degree to which a system, product or component prevents unauthorized 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.

6.2.7 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.

6.2.8 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.

6.2.9 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.

7. Implementation Plan

Following the general plan for the execution of the COMPAIR project, that has been listed on the Description of Action of the COMPAIR Grant Agreement, in this section we present the plan for the delivery of the COMPAIR system environment.

More specifically, the integration of the COMPAIR system lasts until M30 of the project. Apart from the final release for the integrated platform, we consider at least two intermediate major releases by M12 and M20.

Thus, the COMPAIR implementation plan distinguishes between the following major milestones for the go-live scenarios:

- **Milestone 1: Closed User Group (Alpha) version of the COMPAIR platform available for internal testing (M12)**

Scope: To provide an early version of the COMPAIR platform that contains the features as detailed in the first selection of epics. It works as a proof of concept of what we want to implement within COMPAIR.

Outcome: In this milestone, we emphasise on a set of COMPAIR capabilities that enable the end users to see a 2D map of the area of interest with information related to the sensors in the area (Policy Monitoring Dashboard) as well as to calculate their CO2 footprint (CO2 Dashboard). It also allows them to use the first version of the mobile application (DEVA). The main functionalities that are addressed in this milestone are those included in Section 2.1.

- **Milestone 2: Open User Group (Beta) version available for user testing (M20)**

Scope: To provide a working prototype of the COMPAIR capabilities that will be used in the piloting phase of the project. To fine tune the COMPAIR platform, following the user validation in the pilot cases. This version will be tested by a small number of external users.

Outcome: In this milestone, we emphasise on a set of the COMPAIR capabilities that enable the end users to use the available functionalities in the COMPAIR dashboards and DEVA.

- **Milestone 3: Public Round version available for user testing (M30)**

Scope: CS data integrated in digital twins and all components in mature state ready to be tested by external users.

Outcome: In this milestone, we emphasise on the refinements of the COMPAIR functions according to the feedback resulting from the evaluation that will be made by the representatives of the pilot cities. This version will be tested by a larger number of external users.

The final version of the COMPAIR integrated prototype will be ready by M36 of the project. Technical partners will make sure that all components are functional and resolve any bugs identified during the pilots.

8. Conclusion

The current document includes the initial architecture specifications and design of the **COMPAIR** platform and serves as the basis for the development tasks of the project. Information about the functionalities from the system point of view, the characteristics of the components of the system and the interaction between them, as well as the integration activities and tools, were presented in detail.

This is the first version of the system's reference architecture as this will continue to evolve throughout the project and it is important to make sure that it is consistent and in line with the design and implementation work being described in all technical work packages, assisting the early pilot activities of the project.

This deliverable acts as the reference point for the actual development of the platform and offers a shared and common background for the Consortium participants on the envisaged technologies that are necessary to build such a platform.

9. References

1. https://traefik.io/
2. https://www.keycloak.org/
3. https://docs.ogc.org/is/18-088/18-088.html
4. https://www.docker.com/
5. https://docs.docker.com/compose/
6. http://en.wikipedia.org/wiki/Software_development_methodology
7. http://agilemanifesto.org/
8. https://www.atlassian.com/software/jira
9. https://restfulapi.net/http-status-codes/
10. https://restfulapi.net/http-methods
11. https://junit.org/junit5/
12. https://site.mockito.org/
13. https://karma-runner.github.io/latest/index.html
14. https://mochajs.org/
15. https://www.chaijs.com/
16. https://docs.python.org/3/library/unittest.html
17. https://pypi.org/project/nose/
18. https://jestjs.io/
19. https://jmeter.apache.org/
20. https://www.sonarqube.org/
21. https://prometheus.io/
22. https://grafana.com/
23. https://www.iso.org/standard/35733.html