Integrated NBS-based Urban Planning Methodology for Enhancing the Health and Well-being of Citizens

D5.7

Improved Visualization Module

WP5 – Technologies to support the development of NBS in the cities

This project has received funding from the European Union's Horizon 2020 Research and Innovation Program under grant agreement No 869448. The contents of this publication are the sole responsibility of the authors and can in no way be taken to reflect the views of the European Commission.

Task Leaders	Geosystems Hellas - Eirini Marinou, George Voskopoulos, Katerina Fotiou, Betty Charalampopoulou
Contributors	

Due Date	31st December 2022
Delivery Date	12th December 2022
Type	Other
Dissemination Level	PU = Public
Keywords	Technical requirements; Visualization Module; Platform design; EuPOLIS Platform

Document History

Version	**Date**	**Description**	**Reason for Change**	**By**
0.1	9th Nov. 2022	Initial Version	Document Structure, Chapter 1	Eirini Marinou [GSH], Katerina Fotiou [GSH]
0.2	21st Nov 2022	Intermediate Version	Addition of content in Chapter 2	Eirini Marinou [GSH]
0.3	28th Nov. 2022	Intermediate Version	Addition of content in Chapter 3	Katerina Fotiou [GSH]
0.4	2nd Dec. 2022	Intermediate Version	Addition of content in Chapter 4	Eirini Marinou [GSH]
0.5	7th Dec. 2022	Intermediate Version	Update of the Verification of the Requirements, Update of the Schematic Verification	Eirini Marinou [GSH], George Voskopoulos [GSH], Katerina Fotiou [GSH]
0.6	9th Dec. 2022	Intermediate Version	Include workflows, finalize User Manual	Eirini Marinou [GSH], George Voskopoulos [GSH]
0.7	16th Dec. 2022	Corrections from the Reviewer (Biopolus)	Corrections, format suggestions	Erzsébet Poór-Pócsi [Biopolus]
0.8	17th Dec. 2022	Corrections from the Reviewer (NTUA)	Enlarge pictures, Insert deployment manuals of software	Manolis Sardis [NTUA]

Table of Contents

Executive Summary

Deliverable D5.7 with the title “Improved Visualisation Module” aims to create a WebGIS platform to visualize the environmental monitoring of the properties under Natural-Based-Solutions (NBS) interventions. euPOLIS Visualisation Module (eVM) is a web-based platform that enables users to explore, understand and analyze the optimized euPOLIS solutions spatiotemporally. This document starts by describing all the requirements that this platform needs to fulfill, such as user requirements, functional and non-functional requirements, as well as the technical specifications underlying this application. Moreover, information regarding the overall architecture, design, and technologies of the euPOLIS’ Improved Visualization Module is included, describing the data flow, the backend, and the frontend processing chain. Following the architecture, detailed screenshots are included, showing step-by-step the functionalities of the platform and the navigation possibilities via the menu. In the last section, the verification of the requirements is described, and justified by given examples.

Lastly within the euPOLIS Visualization platform in the “Information tab”, a User Manual is provided, to facilitate the user’s experience while navigating through the platform, and it was considered essential to be first introduced in this report (**Annex 2**).

List of Figures

List of Tables

List of Acronyms / Abbreviations

***Table 1**. Abbreviations*

Acronym	Explanation
BGS	Blue Green Solutions
BIM	Building Information Modelling
CRS	Coordinate Reference System
D	Deliverable
DEM	Digital Elevation Model
DMS	Data Management System
DS	Demo Site
EC	European Commission
eVM	euPOLIS Visualization Platform
FL	Follower
FR	Front-Runner
FR	Functional Requirements
GIS	Geographic Information System
ICT	Information and Communications Technology
NBS	Nature Based Solutions
OGC	Open Geospatial Consortium
PH	Public Health
UI	User Interface
UR	User Requirements
WB	Well-Being
WP	Work Package

Glossary of Terms

***Table 2**. Glossary of terms*

Term	Explanation
Back end	The back end refers to parts of a computer application or a program's code that allow it to operate and that cannot be accessed by a user. Most data and operating syntax are stored and accessed in the back end of a computer system
Blue Green Solutions (BGS)	In the euPOLIS project, we apply a methodology for innovative urban planning, called Blue Green Solutions (BGS). The methodology originates from the EU (EIT-Climate_KIC program) project *Blue Green Dream* (BGD - www.bdg.org). The planning guide: Blue Green Solutions, A Systems Approach to Sustainable, Resilient and Cost-Effective Urban Development (available from www.bgd.org.uk). The BGS methodology is considered an innovative paradigm for the planning, designing, operating, and maintaining of urban water systems (blue assets) and urban vegetated areas (green infrastructure) not as separate systems, as is the case today, but as a synergistic network interlinked with urban ecosystem services (ESS). By its virtue, BGS are intrinsic components of NBS
Client	'Client side' refers to everything in a web application that is displayed or takes place on the client (end-user device). This includes what the user sees, such as text, images, and the rest of the UI, along with any actions that an application performs within the user's browser
End-user	A person or group in a position to apply the information or tools produced, evaluated or transferred through a project
Front End	Front-end development focuses on the visual aspects of a website – the part that users see and interact with
Location/Demo site	The place where NBS will be implemented in 4 pilot cities
Nature-Based Solutions (NBS)	In the broader frameworks, NBS can be described as Implementation of the solutions based on nature and ecological functions, which address both environmental and societal challenges such as adaptation to and mitigation of climate change, natural disaster protection public health and well-being, and food security by delivering multiple ecosystem services
Web Server	On the hardware side, a web server is a computer that stores web server software and a website's component files (for example, HTML documents, images, CSS stylesheets, and JavaScript files). A web server connects to the Internet and supports physical data interchange with other devices connected to the web
User Interface	A User Interface allows the user to interact with content or software running on a remote server through a Web browser

1. Introduction

Global geopolitical, economic, climatic, as well as other developments, are posing significant social issues for European cities, placing great pressure on urban areas to create conditions that support Public Health (PH) and well-being (WB). The traditional approach to urban and revitalization planning is mostly driven by financial criteria and regular processes, frequently lacking cutting-edge integrated approaches and concepts that place a strong emphasis on societal, cultural, economic, and environmental factors. As a result, the demands of local communities are either ignored, or underappreciated, and as a result, cities make expensive investments that local populations do not support and are consequently not sustainable. To address these challenges, urban planning approaches built on the euPOLIS NBS services and enhanced with cultural and societal considerations provide the combination of a people-centered approach with the major environmental and economic benefits of Blue Green Solutions.

The main goal of the euPOLIS project is the regeneration and rehabilitation of urban ecosystems, by creating proper urban planning matrices, and inclusive and accessible urban spaces. The addressed key challenges include low environmental quality and low biodiversity in public spaces, water-stressed resources, and undervalued use of space. The project's methodologies will be tested in four front cities: Belgrade (Serbia), Lodz (Poland), Piraeus (Greece), and Gladsaxe (Denmark). The euPOLIS team will outline these critical challenges that the demonstration sites face, provide integrated solutions, and measure their positive impact on the quality of the citizens' lives, like their overall well-being (WB), physical health (PH,) and mental, as well as emotional health.

The role of **WP5** "Technologies to support the development of NBS in the cities" is to coordinate customization and integration of existing modeling and planning technologies for NBS-based implementation in the demo sites. Its basic objective is to adapt and integrate existing technologies to support the co-design, implementation, and monitoring of NBS and related technical and social aspects in the cities. WP5 consists of the following tasks:

- Task 5.1: "Technical Specifications drove from requirements" [M10-M14]
- Task 5.2: "Extending HeartAround monitoring platform" [M12-M24]
- Task 5.3: "Extending MyFeel monitoring platform" [M12-M24]
- Task 5.4: "Implementation of the Data Management System" [M12-M24]
- Task 5.5: "Networking solutions for data exchange" [M12-M24]
- Task 5.6: "Metabolism-based NBS Planning and Simulation Toolkit" [M12-M24]
- **Task 5.7: "Improved Visualisation" [M15-M28]**

This document is the report presenting the methodology followed for the development of the euPOLIS WebGIS platform and the functionalities of the platform itself. It is the result of **Task 5.7 "Improved Visualisation"** and that officially concludes Work Package 5. The following sub-sections present the scope and objectives, the structure of the document, and the relation to other WPs and Tasks.

1.1 Scope and objectives of the deliverable

The main objective of deliverable **D5.7 "Improved Visualisation"**, is to implement a 4D visualization module to enable users to explore, understand and analyze spatiotemporally the optimized euPOLIS solutions. The module will provide comprehensive 2D and 3D views of the city environment enriched with temporal data provided by the system (modeling and sensor information).

Finding the right visual metaphor to maximize the display's intelligibility and engagement is a crucial problem in this situation. Additionally, the module will be responsible for managing the geospatial data required to depict the digital mock-up of the environment by combining existing datasets such as 2D SIG, 3D textured models, BIM, ortho-images, DEM, and lateral imageries.

To create a highly detailed environment for visualization, photo-textured 3D models can be created from input photos using the cutting-edge structure of motion algorithms. OCG standards will be used to convey geospatial data, making it possible for any tool to use them for simulation. Additionally, the functionality of the base platform will also be provided by this deliverable, allowing for the addition and integration of modules and components, the containment of information, and the interaction of euPOLIS services. To depict the results of all tools embedded into the system in a single visualization application, RISA will ensure that the visualization component can communicate with them all.

1.2 Structure of the deliverable

The present document, namely **"D5.7 – Improved Visualisation Module"**, is organized into six (6) Chapters to facilitate search, reference, and further analysis as needed.

- **Chapter 2** presents the Technical Requirements as defined in D5.1 regarding the Visualization Module and is described how these requirements are met
- **Chapter 3** includes information regarding the overall architecture, design, and technologies of the euPOLIS' Improved Visualization Module
- **Chapter 4** presents the euPOLIS Platform Visualisation Module functionalities & user manual
- **Chapter 5** summarises the conclusions of this deliverable
- **Chapter 6** is the reference section
- **Annex 1** describes the verification of the requirements, justified by the given examples
- **Annex 2** *User Manual for the euPOLIS Platform*

1.3 Relation to other Work Packages (WPs)

In **Figure 1,** the demonstration of the Work Plan Structure is provided, indicating the position of the WP5 in the project, related to all the other Work Packages.

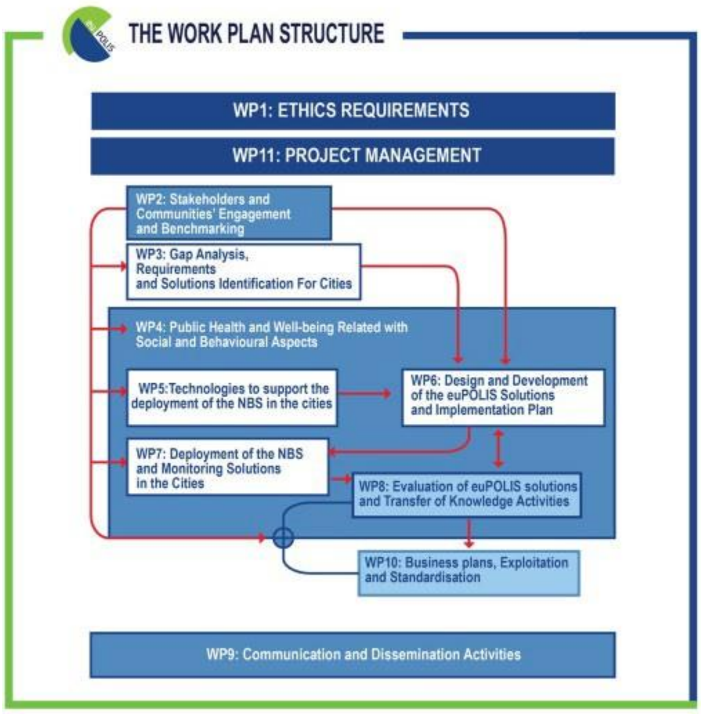

THE WORK PLAN STRUCTURE
WP1: ETHICS REQUIREMENTS
WP11: PROJECT MANAGEMENT
WP2: Stakeholders and Communities' Engagement and Benchmarking
WP3: Gap Analysis, Requirements and Solutions Identification For Cities
WP4: Public Health and Well-being Related with Social and Behavioural Aspects
WP5:Technologies to support the deployment of the NBS in the cities
WP6: Design and Development of the euPOLIS Solutions and Implementation Plan
WP7: Deployment of the NBS and Monitoring Solutions in the Cities
WP8: Evaluation of euPOLIS solutions and Transfer of Knowledge Activities
WP10: Business plans, Exploitation and Standardisation
WP9: Communication and Dissemination Activities

Figure 1. *Demonstration of the Work Plan Structure, the position of WP5, and correlation with the other Work Packages*

As it can be seen, WP5 is based on the outcomes of all previous work packages. The baseline analysis, problems and needs assessment through intense stakeholder involvement (WP2, WP3) are fed to WP4, which will provide a set of Indicators appropriate for assessment. WP3 focus on project requirements and potential solutions according to needs, concerns, and available resources. The outcomes of T3.3 (June 2021) established a set of points that need to be addressed. WP5 builds upon both WP3 and WP4, furthering the development of methods and tools to measure, and assess the impacts of NBS.

1.4 Data flow and position of the 4D Visualization

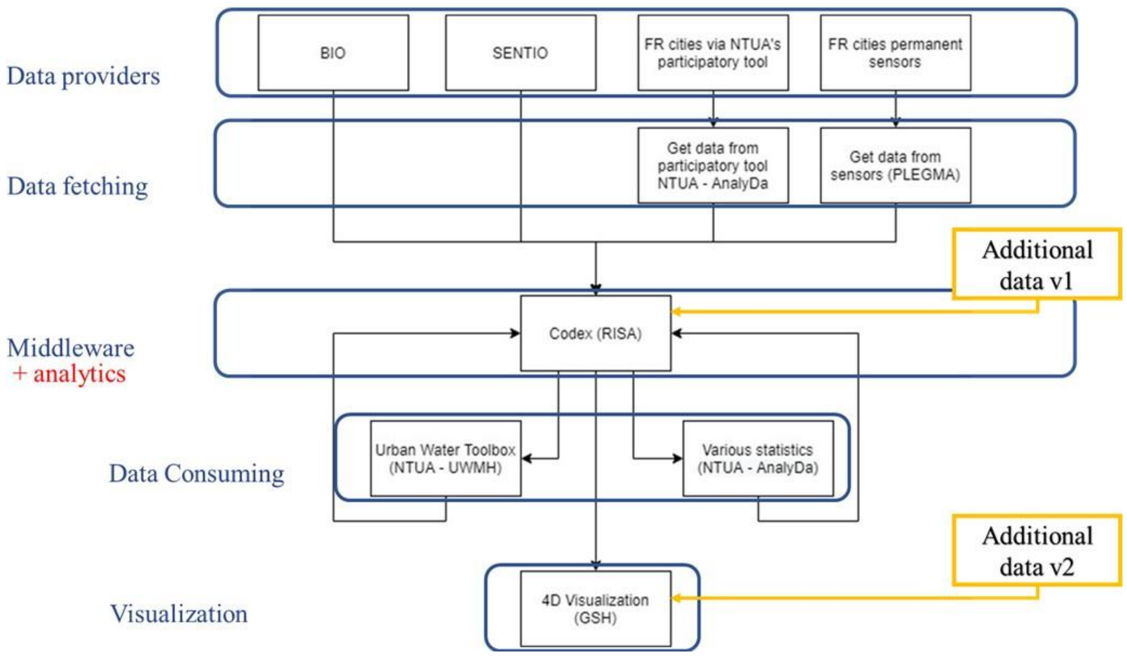

Data providers
BIO
SENTIO
FR cities via NTUA's participatory tool
FR cities permanent sensors
Data fetching
Get data from participatory tool NTUA - AnalyDa
Get data from sensors (PLEGMA)
Additional data v1
Middleware + analytics
Codex (RISA)
Data Consuming
Urban Water Toolbox (NTUA - UWMH)
Various statistics (NTUA - AnalyDa)
Additional data v2
Visualization
4D Visualization (GSH)

Figure 2. *Main data flow*

The flow depicted in **Figure 2,** is part of deliverable 5.1 and it describes the various data sources that are connected to the platform. The general position of the 4D Visualization, created by GSH, is found at the bottom of the flow.

Firstly, the multiple channels are either explicitly described in the GA or implicitly induced as secondary data sources.

The Primary data sources or **Data providers** are summarized as follows:

- The wearable device and/or software providers
- The installed (or are planned to be installed) cities' sensors
- Any available participatory tools

The **Middleware services** will be provided by RISA. The DMS is a tested proprietary solution that will be parameterized appropriately to host all related data flows for the project. The same system can support various analytics, accept data from additional sources, such as online repositories related to euPOLIS, and communicate with 3rd part implementations, like the NTUA's simulation toolkit.

The last step includes the **Visualization Toolbox**, provided by GSH. This tool described in this report provides a dynamic interface adjustable to the user's needs and is capable of illustrating various information, stored in the DMS, including measurements from weather/air pollution stations, advanced analytics, figures, and time series data. At this point, for the sake of completeness, it needs to be mentioned that the tool also uses some data that is not stored in the DMS but in the local server of GSH. This applies in the case where the data is not used for other analyzes of the project and is only used only for visualization, for example, the 3D models of the interventions.

In **Section 2**, all the requirements for the development of this platform will be presented, and then verified accordingly to the results (**Annex 1**).

2. euPOLIS Visualisation Module's Technical Requirements

The euPOLIS platform establishes an integrated solution, capable of determining, gathering, merging, and examining data collected from multiple sources, allowing end-users to assess the effectiveness and suitableness of the adopted NBS services over the citizen's WB and PH.

This platform was designed and unified based on specific requirements, allowing the integration of multiple tools, provided by relevant partners and supported by involved parties, such as urban planners, medical teams, citizens, stakeholders, and policymakers in numerous ways. In this section, a general overview of the requirements will be presented, focusing more on the **main technical requirements** that were used as the foundation for the euPOLIS platform.

2.1 General Structure of the Requirements

The methodology employed to collect and process all data needed to compile a comprehensive list of project requirements for demo sites in 4 FR cities and 4 FL cities, is described in deliverable D3.3. Following these requirements, a set of technical requirements that fulfill the overall project requirements was established and presented in Deliverable 5.1.

Having defined the methodology and the sources for the project requirements, at the design phase the identification of concrete sets of functions, processes, synergies, and differences will be identified, ready to be applied through the euPOLIS innovative design methodology. It is of high importance to gather all project and technical requirements to gain a detailed understanding so that the project resources are optimally used to maximize euPOLIS' output. All the available data have been divided into three categories based on the direct impact on physical health and well-being. The categories are illustrated in the following **Figure 3**, which was designed on the scope of the deliverable D3.3. This table was used as the baseline for the classification of each requirement that emerged, for each FR and FL city.

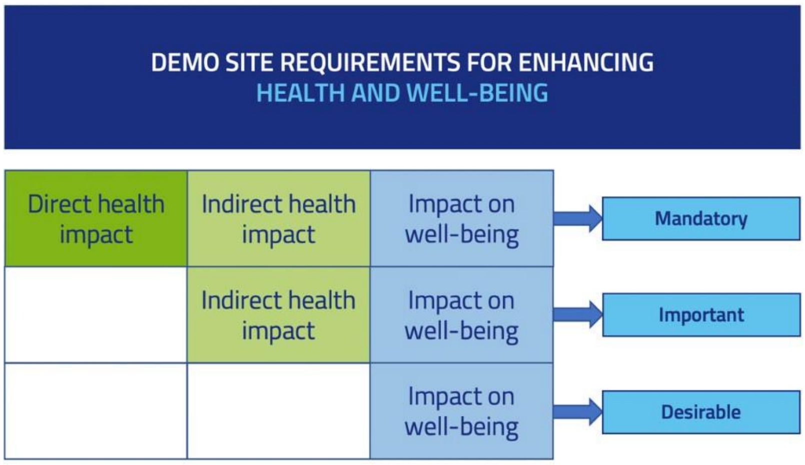


Figure 3. *Foundation for the classification of the project and technical requirements*

The technical requirements based on deliverable D5.1 were separated into three categories: **[1]** The Users' requirements related to technical perspectives, **[2]** The functional requirements (focuses on what a product must do), and **[3]** Non-functional requirements (the general properties of the system), as indicated in the following sub-sections.

2.2 Users' requirements (UR) related to technical perspectives

Table 3 describes the general end-user requirements based on the deliverable D5.1. This table is a sub-section of the **overall table 3.1 of the deliverable D5.1,** focusing only on the relevant module regarding the euPOLIS Platform, such as the "Visualization tool" and the "All systems" module. Based on **Figure 3** the notations M, I, and D stand for Mandatory, Important, and Desirable, respectively.

***Table 3**. Users' Requirements (UR)*

UR-ID / Title	Description	Priority[1]	Relevant Module
UR- 1 Intervention impact visualization	Create an environment that allows for easy display of the NBS impact on the sites	**M**	Visualization tool
UR- 2 euPOLIS outputs visualization	Create an environment for mobile monitoring of the related results to end users	D	Visualization tool
UR- 3 Raw data accessibility	Provide [user level] accessibility to the recorded data from the wearables	D	All systems
UR- 4 NBS evaluation	Provide various analytics, which can help quantify the impact of an applied NBS	**M**	All systems
UR- 5 Well-being indicators accessibility	Provide [user level] accessibility to the indicator values, related to well-being measurement	I	All systems
UR- 6 Data protection	Application of multiple procedures for anonymizing/protecting data in the euPOLIS project. The data storage shall comply with GDPR	**M**	All systems
UR- 7 Device and Data security	Create an environment that implements the security framework end-to-end for data (from source to output visualization). Including authentication of devices and confidentiality and integrity of data	**M**	All systems
UR- 8 System extensibility	The system needs to be modular. It will offer the possibility to be extended with further sensors, wearables, and larger system deployments	**M**	All systems
UR- 9 Availability to people with disabilities	Facilitate the utilization by hear/visual Impaired people	I	All systems
UR- 10 Application support	Provide technical details and a how-to-use manual, for each of the employed subsystems	I	All systems
UR- 11 Feedback provision	Allow for feedback from the end users	I	All systems
UR- 12 Local language support	Availability of the application's menu in the local language or the provision of an extensive manual in the local language	**M**	Wearables, Visualization
UR- 13 Data provision	Empowering users to perform various types of spatial and attribute quires	D	Visualization tool

[1] M – Mandatory, I – Important, D - Desirable

UR-ID / Title	Description	Priority[1]	Relevant Module
UR- 14 Data homogenization	The visualization solution should be able to deliver all available data on the unified common coordinate system	**M**	Visualization tool

2.3 Functional requirements (FR) related to technical perspectives

Table 4 describes the functional requirements based on the deliverable D5.1. This table is a sub-section of the **overall table 3.2 of the deliverable D5.1,** focusing only on the relevant module regarding the euPOLIS Platform, such as the "Visualization tool" and the "All systems" module. Based on **Figure 3** the notations M, I, and D stand for Mandatory, Important, and Desirable, respectively.

Table 4. *Functional Requirements (FR)*

FR-ID/ Description	Priority[2]	Relevant module (s)	Related URs
FR- 1 Provide a web service to visualize the interventions	**M**	Visualization, DMS	UR- 12
FR- 2 Establish a UI capable to demonstrate the impact of the NBS on a local level	I	Visualization, DMS	UR- 12
FR- 3 Provide various analytics, which can help quantify the impact of an applied NBS	**M**	All systems	UR- 15
FR- 4 Interconnection with several software and programming languages like JavaScript, PHP, Python, etc	I	DMS, Visualization, Other	UR- 12 UR- 13 UR- 28 UR- 29
FR- 5 Support various types of spatial data format including vector, raster, KMZ, Shapefiles, GeoJSON	I	DMS, Visualization, Other	UR- 12 UR- 13 UR- 28 UR- 29
FR- 6 Variability of user access levels to ensure the security of potentially sensitive data	**M**	Visualization tools	UR- 12 UR- 13
FR- 7 Ability to store different data objects, such as text files, photos, maps, etc	**M**	Visualization tools	UR- 12 UR- 13

[2] M – Mandatory, I – Important, D - Desirable

2.4 Non - Functional requirements (NFR) related to technical perspectives

Table 5 describes the non-functional requirements based on deliverable D5.1. This table is a sub-section of the **overall table 3.2 of the deliverable D5.1,** focusing only on the relevant module regarding the euPOLIS Platform, such as the "Visualization tool" and the "All systems" module. Based on **Figure 3** the notations M, I, and D stand for Mandatory, Important, and Desirable, respectively.

Table 5*. Non-functional Requirements (NFR)*

NFR-ID/ Specification category	Description	Priority[3]	Relevant module(s)	Related URs
NFR- 1 Adaptability	euPOLIS platform to be built over block components. That way any changes required can be performed fast without implications to the rest of the components	**M**	All systems	UR- 23
NFR- 2 Availability	euPOLIS platform should be able to process incoming data and check for corrupted instances or failures in transmission	**M**	Wearables, Permanent, DMS, Visualization	UR- 19 UR- 20 UR- 21 UR- 22 UR- 28 UR- 29
NFR- 3 Capacity	euPOLIS platform should support the simultaneous use of all available sensors in all cities	D	Permanent, DMS, visualization	UR- 11 UR- 13
NFR- 4 Compliance	euPOLIS services should comply with the pilot's country's legislation on data protection and privacy	**M**	All systems	UR- 21 UR- 22
NFR- 5 Configuration management	Maintain the euPOLIS platform's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life	**M**	All systems	ALL
NFR- 6 Confidentiality	Assurance that information is shared only among authorized persons or organizations	**M**	All systems	UR- 21
NFR- 7 Extensibility	euPOLIS platforms' components should support future adjustments to handle bigger network load, more types of sensors, and additional incoming data	**M**	All systems	UR- 23

[3] M – Mandatory, I – Important, D - Desirable

NFR-ID/ Specification category	Description	Priority[3]	Relevant module(s)	Related URs
NFR- 8 Fault tolerance	Handling components' failure; system distributed sources allow partial operation even if some of the components fail to operate	M	All systems	UR- 17 UR- 22
NFR- 9 Interoperability	The interfaces for each component should follow specific communication standards, fully understandable, allowing the smooth operation with other products or systems, at present or future, in either implementation or access	I	All systems	UR- 17
NFR- 10 Maintainability	euPOLIS' platform's components allow for targeted replacements/repairs/updates without compromising the other components' functionality	I	All systems	ALL
NFR- 11 Scalability	The platform must be easily scalable to handle increasing numbers of users simultaneously	I	All systems	UR- 18 UR- 19 UR- 20 UR- 23
NFR- 12 Supportability	Software should enable direct contact through mail, or communication via the application	I	wearables, visualization tools	UR- 25 UR- 26
NFR- 13 Simple security measures	Ensure regular backups are taken and that the restoration procedures work as expected; Load only required software or services; Perform regular updates	I	All systems	UR- 21
NFR- 14 Suitable Network Security Architecture	An important principle is the segregation of production environments from development or testing environments. Similar segmentation according to function can further improve security and also make security measures scalable	-	All systems	UR- 23
NFR- 15 System Documentation	Documenting system components, networks, services, and software should provide for a bird's-eye view needed to thoroughly cover and consider security concerns, attack vectors, and possible security domain bridging points	-	All systems	UR- 25 UR- 26

NFR-ID/ Specification category	Description	Priority[3]	Relevant module(s)	Related URs
NFR- 16 Continuous System Management	Apply security updates and general software updates, using configuration and patch management, and deployment tools	-	All systems	ALL
NFR- 17 Platform security disclaimer	Text explaining security (generally) for user	-	All systems	UR- 25
NFR- 18 Seamless exchange of data information among systems	Safe and secure data transmission between individual components to the centralized DMS	I	All systems	ALL
NFR- 19 equipment safety compliance	Deployed devices have passed all requested tests for safety when used by people	-	-	-

2.5 Verification of the Requirements

The development of the euPOLIS Platform was based on the specific requirements analyzed in the previous sub-sections. In the following subsections, successful coverage of the requirements is presented schematically (**Figure 4-11**), while more extensive tables are provided in **Annex 1**.

2.5.1 Mandatory Requirements (M)

User's Requirements

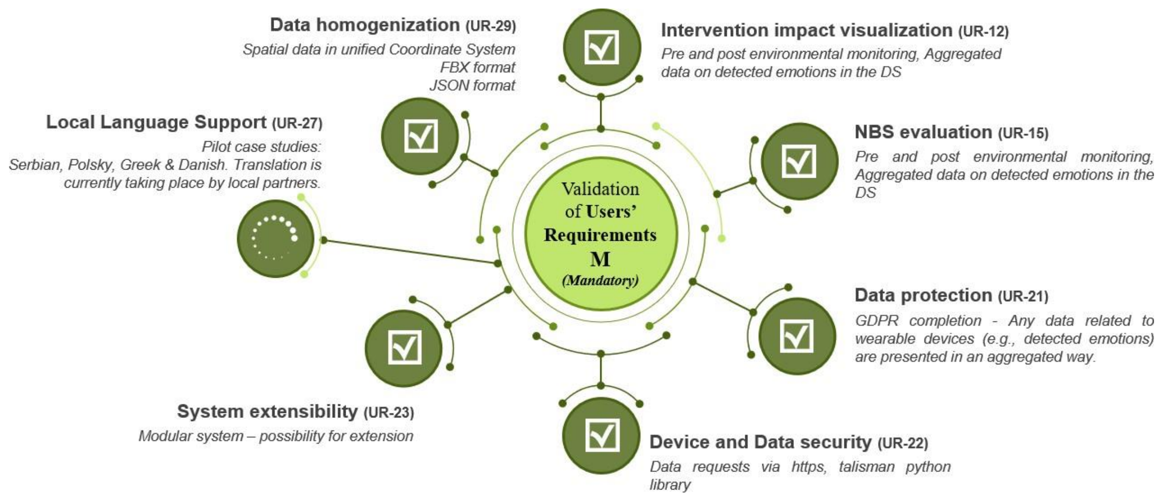

Data homogenization (UR-29)
Spatial data in unified Coordinate System
FBX format
JSON format
Intervention impact visualization (UR-12)
Pre and post environmental monitoring, Aggregated data on detected emotions in the DS
Local Language Support (UR-27)
Pilot case studies:
Serbian, Polsky, Greek & Danish. Translation is currently taking place by local partners.
NBS evaluation (UR-15)
Pre and post environmental monitoring, Aggregated data on detected emotions in the DS
Validation of Users' Requirements
M
(Mandatory)
Data protection (UR-21)
GDPR completion - Any data related to wearable devices (e.g., detected emotions) are presented in an aggregated way.
System extensibility (UR-23)
Modular system – possibility for extension
Device and Data security (UR-22)
Data requests via https, talisman python library

***Figure 4**. Schematic verification of Mandatory User's Requirements*

Functional Requirements

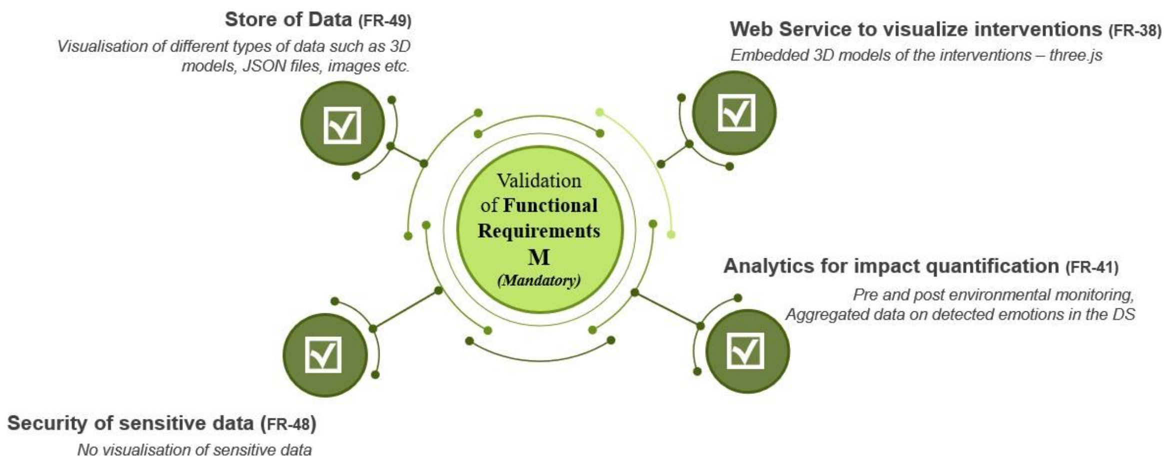

Store of Data (FR-49)
Visualisation of different types of data such as 3D models, JSON files, images etc.
Web Service to visualize interventions (FR-38)
Embedded 3D models of the interventions – three.js
Validation of Functional Requirements M (Mandatory)
Analytics for impact quantification (FR-41)
Pre and post environmental monitoring, Aggregated data on detected emotions in the DS
Security of sensitive data (FR-48)
No visualisation of sensitive data

Figure 5. *Schematic verification of Mandatory Functional Requirements*

Non-Functional Requirements

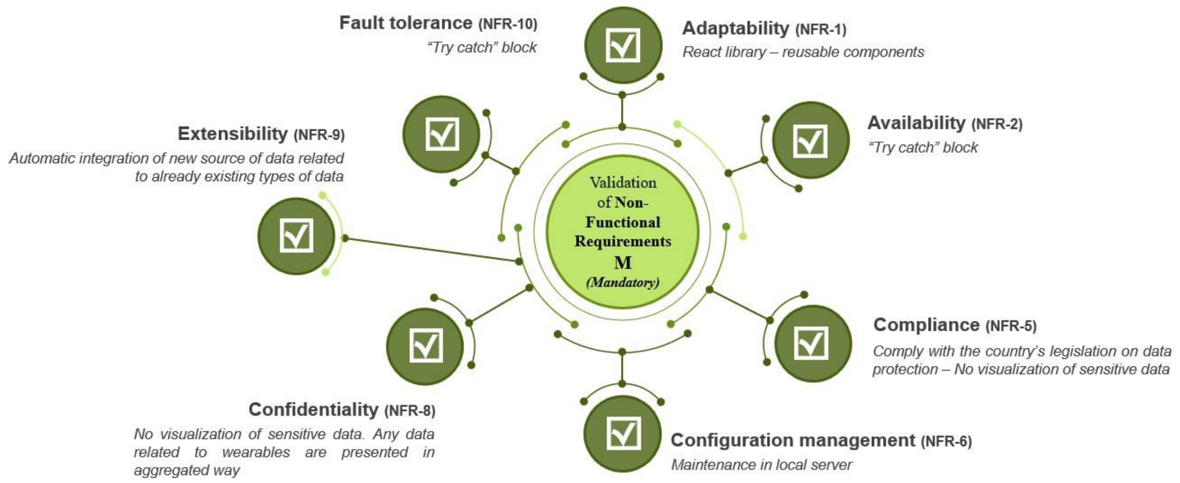

Fault tolerance (NFR-10)
"Try catch" block
Adaptability (NFR-1)
React library – reusable components
Extensibility (NFR-9)
Automatic integration of new source of data related to already existing types of data
Availability (NFR-2)
"Try catch" block
Validation of Non-Functional Requirements M (Mandatory)
Compliance (NFR-5)
Comply with the country's legislation on data protection – No visualization of sensitive data
Confidentiality (NFR-8)
No visualization of sensitive data. Any data related to wearables are presented in aggregated way
Configuration management (NFR-6)
Maintenance in local server

Figure 6. *Schematic verification of Mandatory Non-Functional Requirements*

2.5.1 Important Requirements (I)

User's Requirements

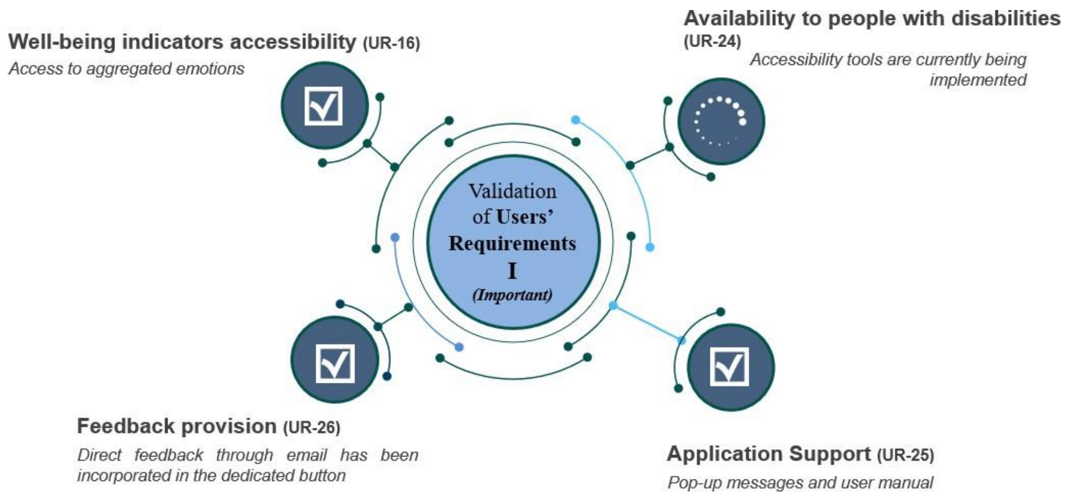

Well-being indicators accessibility (UR-16)
Access to aggregated emotions
Availability to people with disabilities (UR-24)
Accessibility tools are currently being implemented
Validation of Users' Requirements I (Important)
Feedback provision (UR-26)
Direct feedback through email has been incorporated in the dedicated button
Application Support (UR-25)
Pop-up messages and user manual

Figure 7. *Schematic verification of Important User Requirements*

Functional Requirements

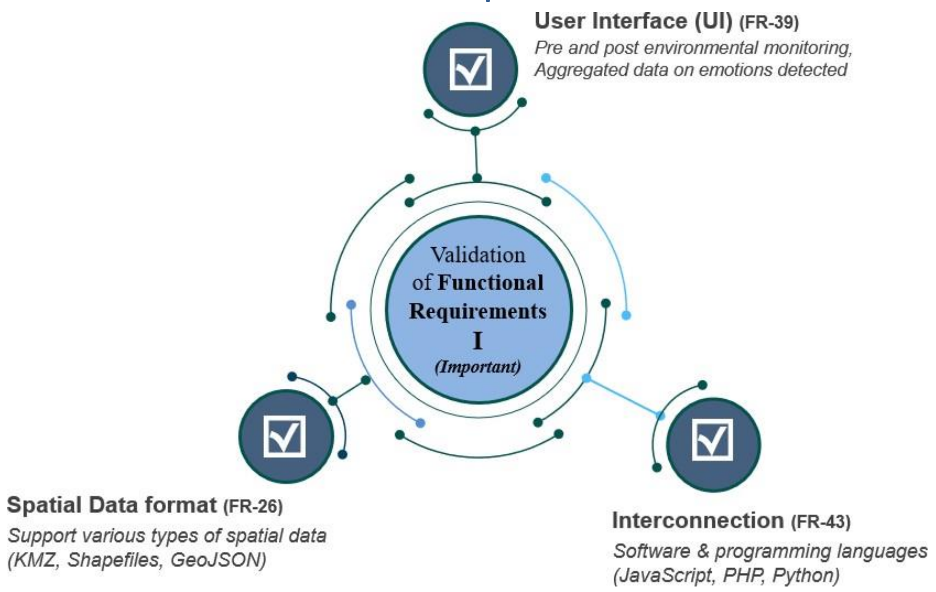

User Interface (UI) (FR-39)
Pre and post environmental monitoring, Aggregated data on emotions detected
Validation of Functional Requirements I (Important)
Spatial Data format (FR-26)
Support various types of spatial data (KMZ, Shapefiles, GeoJSON)
Interconnection (FR-43)
Software & programming languages (JavaScript, PHP, Python)

Figure 8. *Schematic verification of Important Functional Requirements*

Non-Functional Requirements

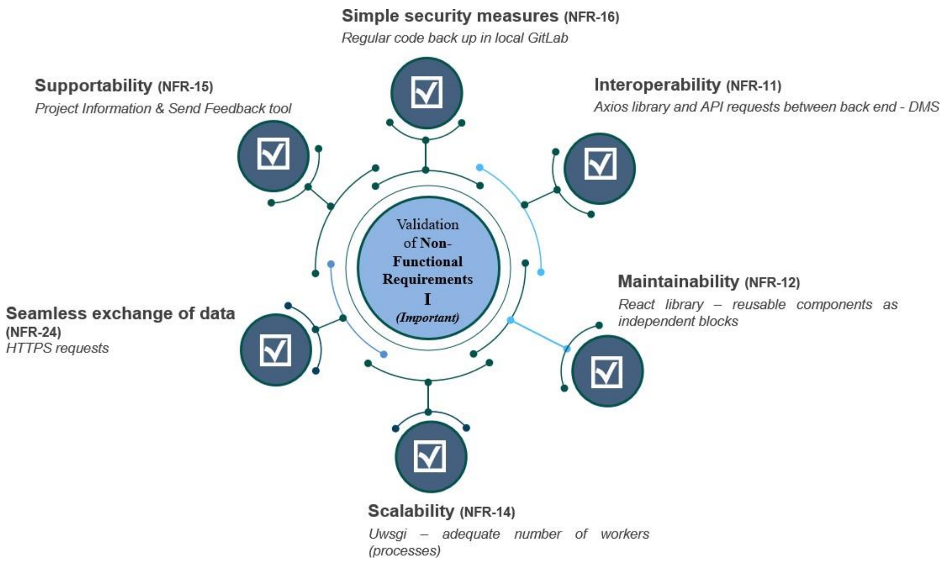

Simple security measures (NFR-16)
Regular code back up in local GitLab
Supportability (NFR-15)
Project Information & Send Feedback tool
Interoperability (NFR-11)
Axios library and API requests between back end - DMS
Validation of Non-Functional Requirements I (Important)
Maintainability (NFR-12)
React library – reusable components as independent blocks
Seamless exchange of data (NFR-24)
HTTPS requests
Scalability (NFR-14)
Uwsgi – adequate number of workers (processes)

Figure 9. *Schematic verification of Important Non-Functional Requirements*

2.5.2 Desirable Requirements (D)

User's Requirements

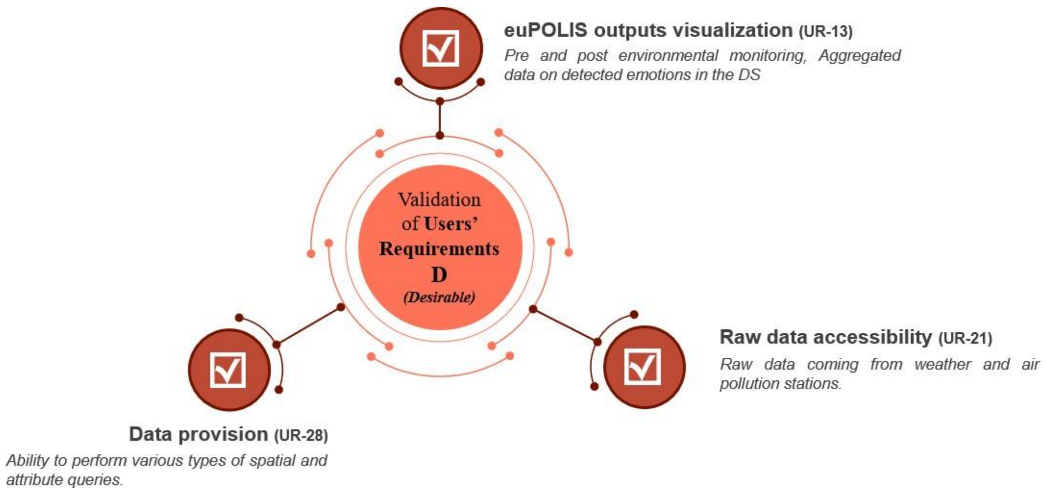

euPOLIS outputs visualization (UR-13)
Pre and post environmental monitoring, Aggregated data on detected emotions in the DS
Validation of Users' Requirements D (Desirable)
Raw data accessibility (UR-21)
Raw data coming from weather and air pollution stations.
Data provision (UR-28)
Ability to perform various types of spatial and attribute queries.

Figure 10. *Schematic verification of Desirable User's Requirements*

Non-Functional Requirements

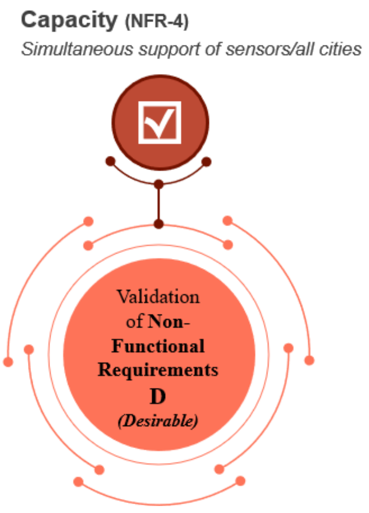

Capacity (NFR-4)
Simultaneous support of sensors/all cities
Validation of Non-Functional Requirements
D
(Desirable)

Figure 11. *Schematic verification of Desirable Non-Functional Requirements*

3 euPOLIS Visualization Module design and implementation

The euPOLIS Visualization Module was designed in such a way as to meet the requirements and specifications defined under D5.1 which were also presented in detail in the previous chapter of this document. Priority was given to the Mandatory (M) requirements but also the other categories Important (I) and Desirable (D) were addressed. More details regarding the verification of the requirements are included in **Annex 1**.

3.1 Architectural design

The eVM is a WebGIS platform that aims to provide a tool to visualize the interventions and establishes a User Interface (UI) capable to demonstrate the impact of the NBS on a local level. In this context, the architecture was designed in such a way as to respond to the project's requirements defined under D5.1 "and presented also in the previous chapter. The visualization tool offers an enhanced 3D GIS environment with more dimensions including time (4D) -as described in detail in chapter 4, the tool gives the possibility to the end-user to define the time parameter in her/his queries.

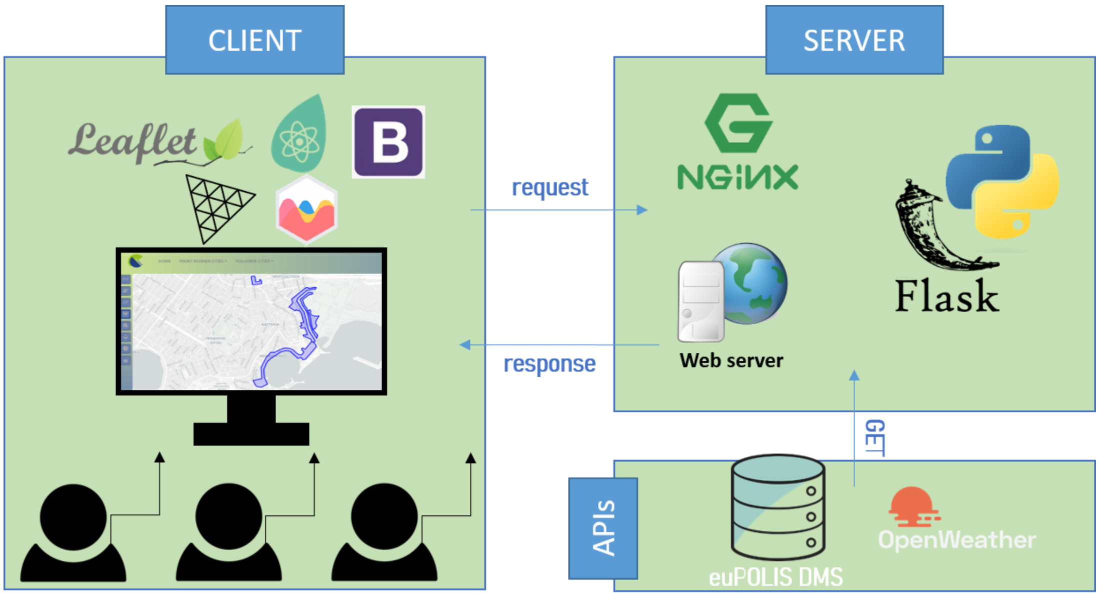


Figure 12. *euPOLIS Visualization Module architecture*

Figure 13 displays the euPOLIS architecture (as defined in D5.1– “*Technical Specifications of euPOLIS modules”*).

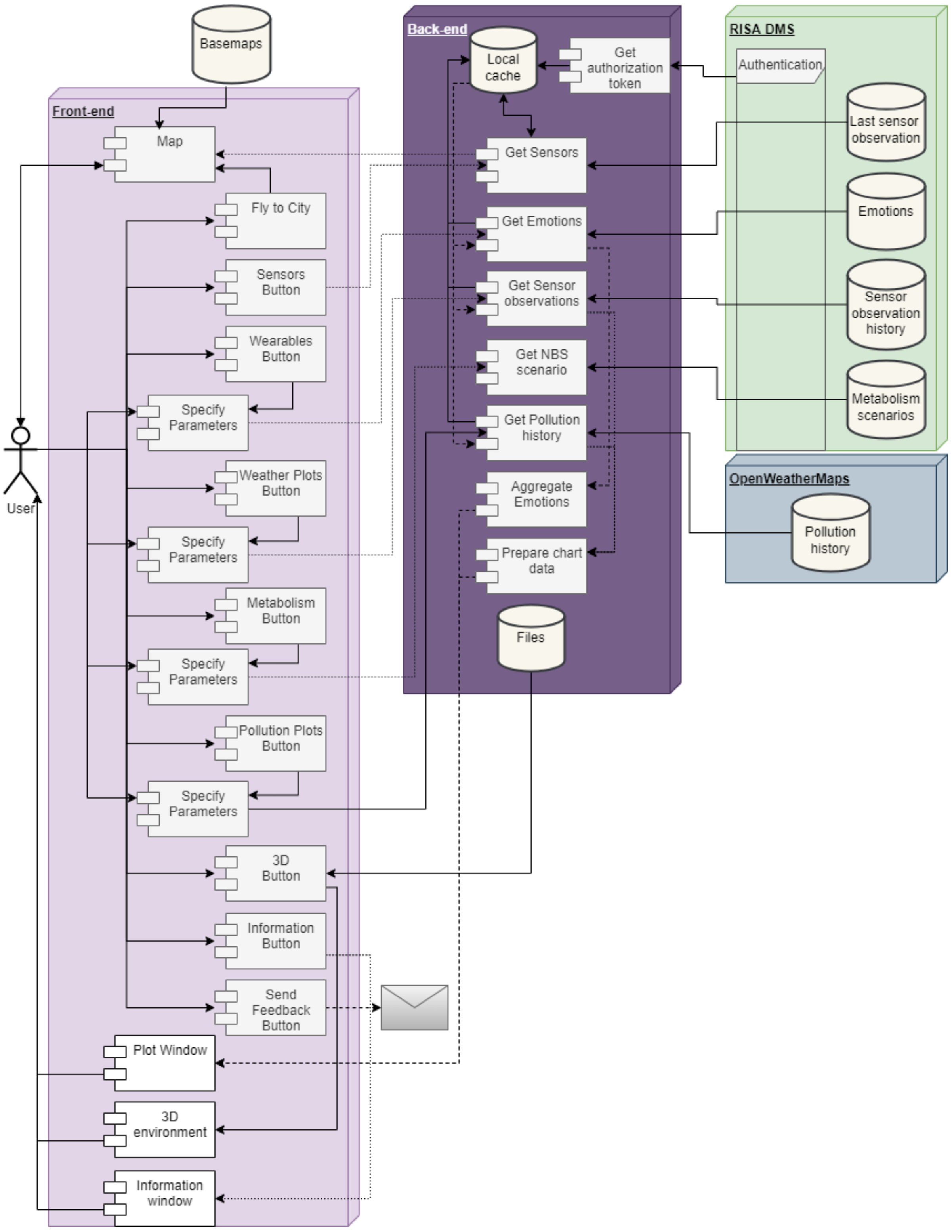


Figure 13. The architecture of the euPOLIS Platform

3.2 Front end

As it can be seen in **Figure 12**, the eVM makes use of the core languages of the Web and on top of these utilizes additional libraries/technologies to provide the user with enhanced web page navigation. In more detail, the front end is written with a combination of the high-level programming language **JavaScript (JS)**, the Hypertext Markup Language (**HTML**), and the Cascading Style Sheets (**CSS**). Complementary to these, **Leaflet** was utilized to embed the map together with interactive functionalities, which will be further presented later in this deliverable. **React** was also incorporated into the visualization module for creating interactive UI. **Three.js** was used for the sake of displaying 3D data of the interventions. For the implementation of the WebGIS application, only free and open-source libraries/tools have been utilized. The following paragraphs narrate the reasons that led to choosing them and briefly discuss how they function.
The complete list of programming -JavaScript- libraries used for the frontend development is shown in **Table 6** below:

***Table 6**. List of programming JavaScript libraries for Front end*

Library Name	Description
Leaflet	Open-source library for mobile-friendly interactive maps
React	Used for building interactive UIs
Chart.js	A JavaScript library that enables the visualization of data through creating flexible charts online
Three.js	Is a cross-browser JavaScript library and application programming interface used to create and display animated 3D computer graphics in a web browser using WebGL

3.2.1 Leaflet

Leaflet is one of the leading open-source libraries for mobile-friendly interactive maps. This library is written in JavaScript and supports various functionalities related to the map data. Indicatively, it supports clicking on objects on the map and getting additional information through pop-up windows. Another major functionality provided by Leaflet is the layer switcher (**Figure 14**) through which end-users can choose the cartographic background of the map among a predefined list of backgrounds.

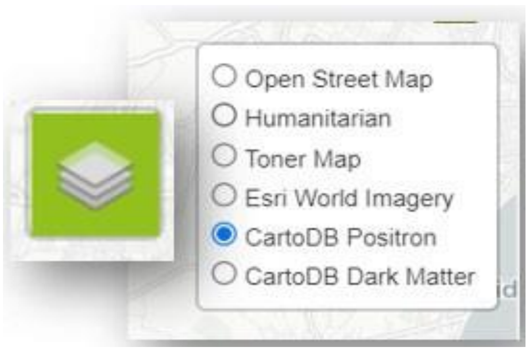


***Figure 14**. Layer switcher provided by Leaflet*

3.2.2 React

React[4] is also a JavaScript library and it is used for building interactive UIs. This library enables the development of applications by creating reusable components that can be considered independent

[4] https://reactjs.org/

blocks. These components are individual pieces of a final interface, which, when assembled, form the application's entire UI. By separating the complex UIs into reusable components, React framework combines the speed and efficiency of JavaScript with a more efficient method of manipulating the DOM to render web pages faster and create highly dynamic and responsive web applications.

In more detail, when data changes in a traditional JavaScript application, manual DOM manipulation is required for these changes to be incorporated. On the other hand, React utilizes a virtual DOM, which is a copy of the actual DOM. This virtual DOM is immediately reloaded to reflect any change in the data state and then is compared to the actual DOM to detect what exactly has changed.

An example of the utilization of React in the eVM is the following: Clicking the wearables button changes the state of the app, which triggers a DOM reload and visualizes a new component that includes a specific form about the selection of a location and a date.

Additionally, React offers components for **Chart.js**, another JavaScript library that enables the visualization of data through creating flexible charts online. In the context of eVM, chart.js is being used to provide weather and air pollution plots. The user can then select for which time period and for which weather or air pollution parameters, the chart will report.

3.2.3 Three.js

Three.js is a cross-browser JavaScript library and application programming interface used to create and display animated 3D computer graphics in a web browser using WebGL. The source code is hosted in a repository on GitHub. In the context of the eVM, three.js is used to load 3D models of the interventions in the DS. This library provides many loaders supporting various file formats. EuPOLIS Visualisation Module currently supports 3D data in FBX format.

3.3 Back End

The core of the euPOLIS WebGIS application is deployed using the **Nginx[5] Web Server**. Nginx is one of the most widely used Web Servers which can also be used as a reverse proxy and load balancer and it's distributed as open source under the terms of the 2-clause BSD license.

The WebGIS tool's backend is written in the high-level general-purpose programming language **Python**, in combination with the micro web framework **Flask**. Python is one of the most popular backend programming languages offering high scalability, an extensive standard library, and a wide variety of third-party libraries and frameworks. Flask is a lightweight WSGI web application framework with little to no dependencies on external libraries (micro-framework) and can support the creation of both small and bigger commercial websites. The backend undertakes the following tasks, also illustrated in **Figure 13**:

- Serving the front end to the client's browser
- Communicating with RISA API to receive data about:
  - Sensor observations
  - Scenario information
  - Aggregated measurements from wearables concerning GDPR and compliant with UR-21 Data Protection and UR-22 Device and Data Security defined under D5.1 and presented also in this document in **Table 3**

[5] https://www.nginx.com/

  - Aggregated emotions from wearables concerning GDPR and compliant with UR-21 Data Protection and UR-22 Device and Data Security defined under D5.1 and presented also in this document in **Table 3**
- Communicating with OpenWeatherMap API to receive data about pollution in a specific area
- Keeping some of the above information in the local memory, to limit the number of requests to the APIs
- Conveniently preparing the above data for the front end to visualize

The complete list of programming -python- libraries used for the backend development is shown in **Table 7**. In **Figure 15** and **Figure 16**, examples of the code used in the backend procedure are demonstrated mainly focusing on the import of the libraries:

```
from flask import (
Flask,
send_from_directory,
request,
jsonify,
)
import logging
import requests
import json
import os
import datetime
from talisman import Talisman
from dotenv import load_dotenv
```

Figure 15. *Example of code of imported python libraries*

```
[uwsgi]
module = wsgi:app

master = true
processes = (adequate number, e.g.,5)
socket = myproject.sock
chmod-socket = 446
vacuum = true
```

Figure 16. *Example of "uwsgi" library in the terminal*

***Table 7**. List of programming python libraries for Back end*

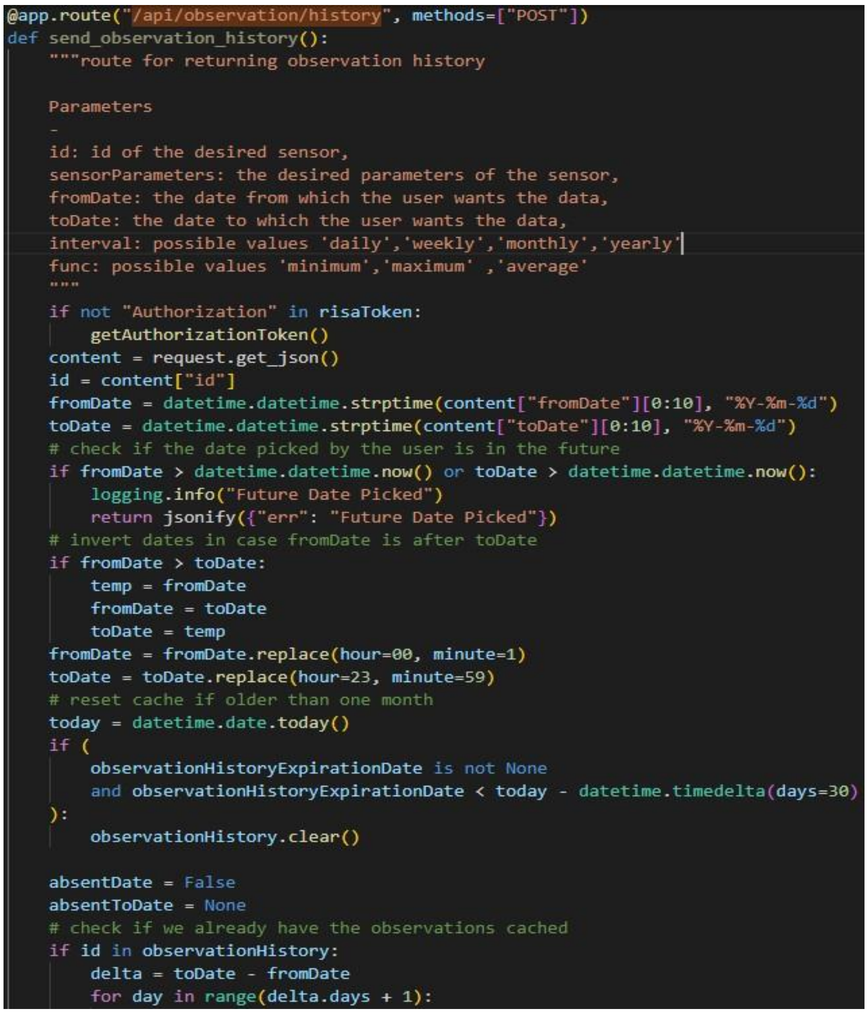
Library Name	Description
DateTime	Allows parsing and management of dates
Dotenv	Allows loading of environment variables from a file. That is used to keep API keys, passwords, etc. away from the code allowing the code to be shared separately
JSON	Allows the conversion of JSON objects to strings and the inverse procedure
Flask ***(Web framework)***	Allows: 1. Creation of endpoints in the backend allowing the frontend to communicate with it 2. Reading of data provided by the frontend when an endpoint is called upon 3. Sending of data to the frontend in response to an endpoint call
Logging	Allows the creation and management of log events

Os	Allows access to the file system and the environment variables
Requests	Allows HTTP requests from the backend
Talisman	Adds some security measures, among them is the content-security-policy
Uwsgi	The backend is making use of the "Uwsgi" library which provides an adequate number of workers (processes) to support concurrent users

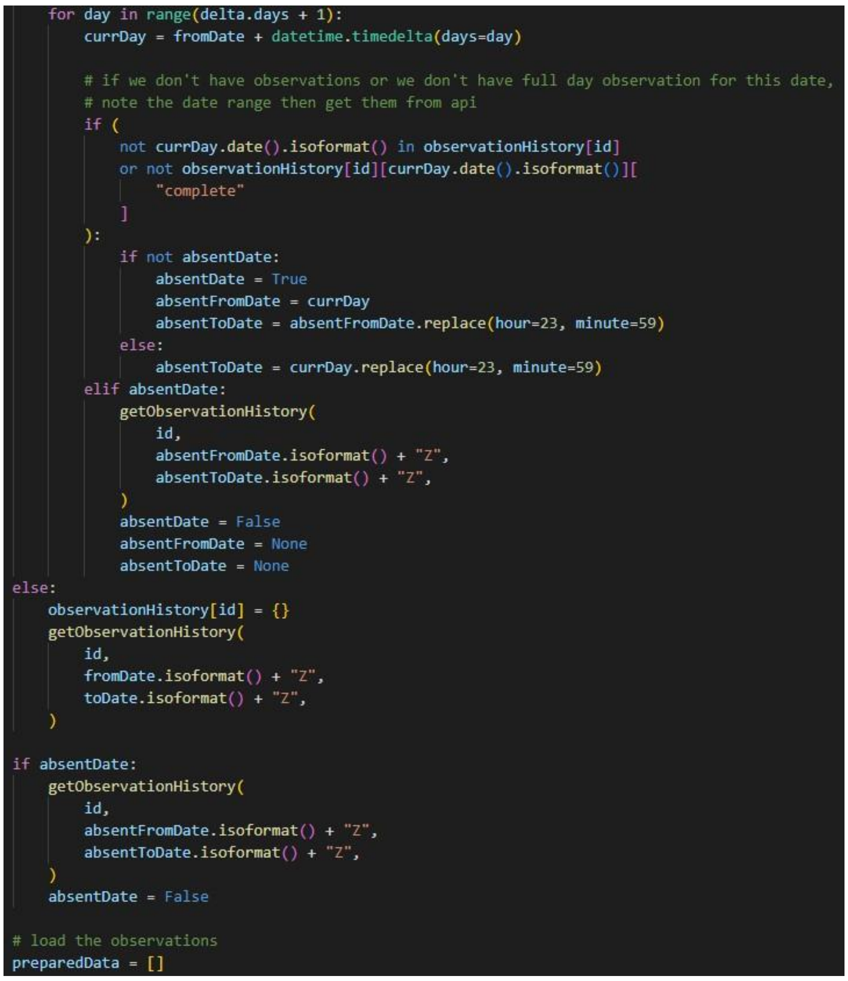
3.4 Example of Code

3.4.2 Weather Plots processing chain: Backend

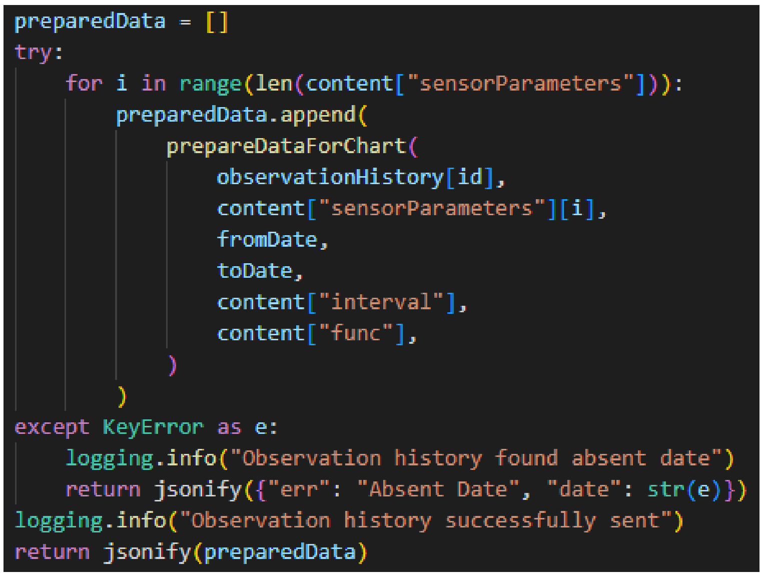
- Error checking (**Figure 17**)
- Get data from RISA's DMS (**Figure 18**)
- Process the data into intervals (**Figure 19**)
- Aggregate data with the desired function (**Figure 20**)



```
@app.route("/api/observation/history", methods=["POST"])
def send_observation_history():
    """route for returning observation history

    Parameters
    -
    id: id of the desired sensor,
    sensorParameters: the desired parameters of the sensor,
    fromDate: the date from which the user wants the data,
    toDate: the date to which the user wants the data,
    interval: possible values 'daily','weekly','monthly','yearly'
    func: possible values 'minimum','maximum' ,'average'
    """
    if not "Authorization" in risaToken:
        getAuthorizationToken()
    content = request.get_json()
    id = content["id"]
    fromDate = datetime.datetime.strptime(content["fromDate"][0:10], "%Y-%m-%d")
    toDate = datetime.datetime.strptime(content["toDate"][0:10], "%Y-%m-%d")
    # check if the date picked by the user is in the future
    if fromDate > datetime.datetime.now() or toDate > datetime.datetime.now():
        logging.info("Future Date Picked")
        return jsonify({"err": "Future Date Picked"})
    # invert dates in case fromDate is after toDate
    if fromDate > toDate:
        temp = fromDate
        fromDate = toDate
        toDate = temp
    fromDate = fromDate.replace(hour=00, minute=1)
    toDate = toDate.replace(hour=23, minute=59)
    # reset cache if older than one month
    today = datetime.date.today()
    if (
        observationHistoryExpirationDate is not None
        and observationHistoryExpirationDate < today - datetime.timedelta(days=30)
    ):
        observationHistory.clear()

    absentDate = False
    absentToDate = None
    # check if we already have the observations cached
    if id in observationHistory:
        delta = toDate - fromDate
        for day in range(delta.days + 1):
```

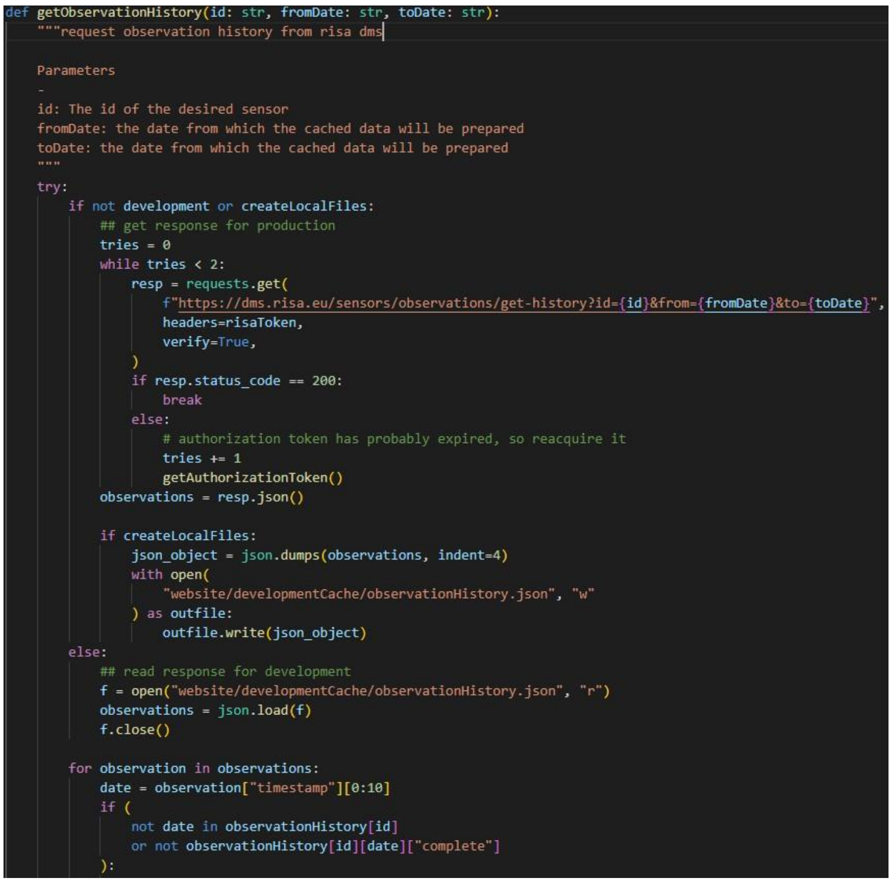


```
    for day in range(delta.days + 1):
        currDay = fromDate + datetime.timedelta(days=day)

        # if we don't have observations or we don't have full day observation for this date,
        # note the date range then get them from api
        if (
            not currDay.date().isoformat() in observationHistory[id]
            or not observationHistory[id][currDay.date().isoformat()][
                "complete"
            ]
        ):
            if not absentDate:
                absentDate = True
                absentFromDate = currDay
                absentToDate = absentFromDate.replace(hour=23, minute=59)
            else:
                absentToDate = currDay.replace(hour=23, minute=59)
        elif absentDate:
            getObservationHistory(
                id,
                absentFromDate.isoformat() + "Z",
                absentToDate.isoformat() + "Z",
            )
            absentDate = False
            absentFromDate = None
            absentToDate = None
else:
    observationHistory[id] = {}
    getObservationHistory(
        id,
        fromDate.isoformat() + "Z",
        toDate.isoformat() + "Z",
    )

if absentDate:
    getObservationHistory(
        id,
        absentFromDate.isoformat() + "Z",
        absentToDate.isoformat() + "Z",
    )
    absentDate = False

# load the observations
preparedData = []
```

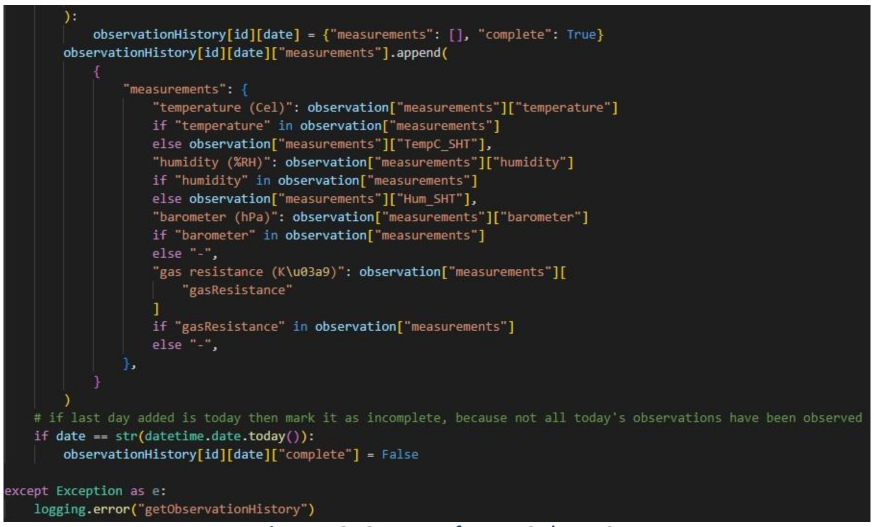


```
preparedData = []
try:
    for i in range(len(content["sensorParameters"])):
        preparedData.append(
            prepareDataForChart(
                observationHistory[id],
                content["sensorParameters"][i],
                fromDate,
                toDate,
                content["interval"],
                content["func"],
            )
        )
except KeyError as e:
    logging.info("Observation history found absent date")
    return jsonify({"err": "Absent Date", "date": str(e)})
logging.info("Observation history successfully sent")
return jsonify(preparedData)
```

Figure 17. *The code checks for errors in the date provided and checks if some or all data requested are in the local cache*

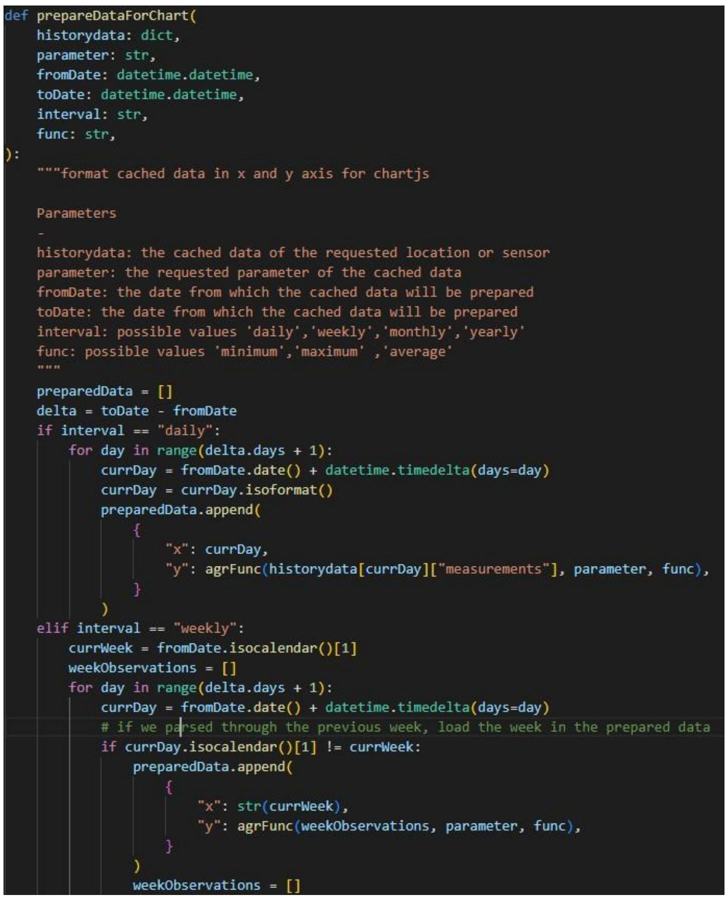


```
def getObservationHistory(id: str, fromDate: str, toDate: str):
    """request observation history from risa dms

    Parameters
    -
    id: The id of the desired sensor
    fromDate: the date from which the cached data will be prepared
    toDate: the date from which the cached data will be prepared
    """
    try:
        if not development or createLocalFiles:
            ## get response for production
            tries = 0
            while tries < 2:
                resp = requests.get(
                    f"https://dms.risa.eu/sensors/observations/get-history?id={id}&from={fromDate}&to={toDate}",
                    headers=risaToken,
                    verify=True,
                )
                if resp.status_code == 200:
                    break
                else:
                    # authorization token has probably expired, so reacquire it
                    tries += 1
                    getAuthorizationToken()
            observations = resp.json()

            if createLocalFiles:
                json_object = json.dumps(observations, indent=4)
                with open(
                    "website/developmentCache/observationHistory.json", "w"
                ) as outfile:
                    outfile.write(json_object)
        else:
            ## read response for development
            f = open("website/developmentCache/observationHistory.json", "r")
            observations = json.load(f)
            f.close()

        for observation in observations:
            date = observation["timestamp"][0:10]
            if (
                not date in observationHistory[id]
                or not observationHistory[id][date]["complete"]
            ):
```

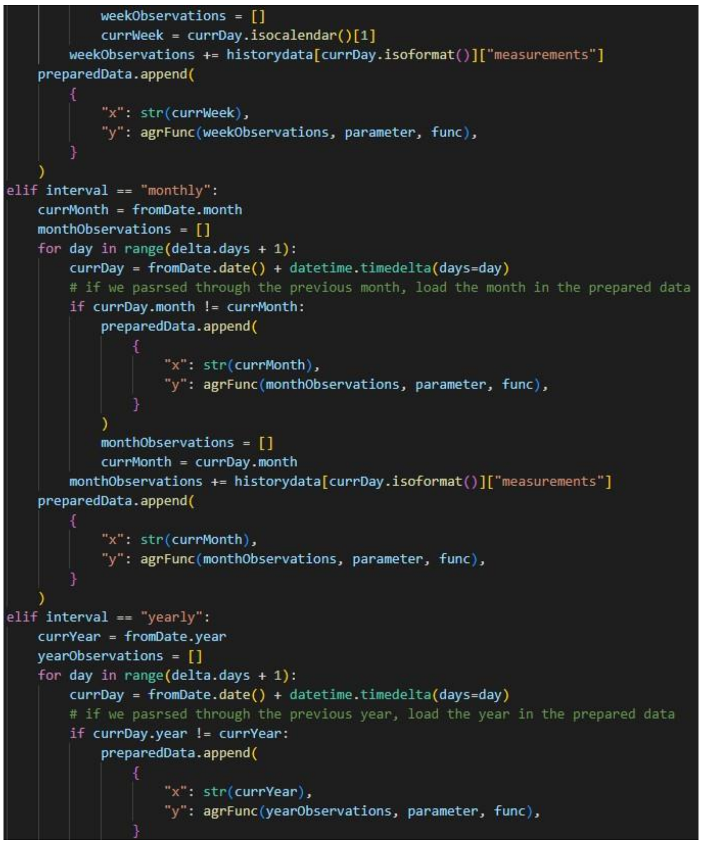


```
        ):
            observationHistory[id][date] = {"measurements": [], "complete": True}
        observationHistory[id][date]["measurements"].append(
            {
                "measurements": {
                    "temperature (Cel)": observation["measurements"]["temperature"]
                    if "temperature" in observation["measurements"]
                    else observation["measurements"]["TempC_SHT"],
                    "humidity (%RH)": observation["measurements"]["humidity"]
                    if "humidity" in observation["measurements"]
                    else observation["measurements"]["Hum_SHT"],
                    "barometer (hPa)": observation["measurements"]["barometer"]
                    if "barometer" in observation["measurements"]
                    else "-",
                    "gas resistance (K\u03a9)": observation["measurements"][
                        "gasResistance"
                    ]
                    if "gasResistance" in observation["measurements"]
                    else "-",
                },
            }
        )
    # if last day added is today then mark it as incomplete, because not all today's observations have been observed
    if date == str(datetime.date.today()):
        observationHistory[id][date]["complete"] = False

except Exception as e:
    logging.error("getObservationHistory")
```

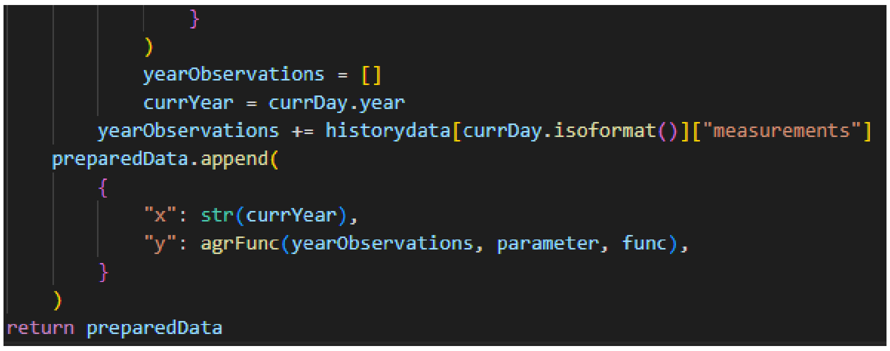
Figure 18. *Get Data from RISA's DMS*



```python
def prepareDataForChart(
    historydata: dict,
    parameter: str,
    fromDate: datetime.datetime,
    toDate: datetime.datetime,
    interval: str,
    func: str,
):
    """format cached data in x and y axis for chartjs

    Parameters
    -
    historydata: the cached data of the requested location or sensor
    parameter: the requested parameter of the cached data
    fromDate: the date from which the cached data will be prepared
    toDate: the date from which the cached data will be prepared
    interval: possible values 'daily','weekly','monthly','yearly'
    func: possible values 'minimum','maximum' ,'average'
    """
    preparedData = []
    delta = toDate - fromDate
    if interval == "daily":
        for day in range(delta.days + 1):
            currDay = fromDate.date() + datetime.timedelta(days=day)
            currDay = currDay.isoformat()
            preparedData.append(
                {
                    "x": currDay,
                    "y": agrFunc(historydata[currDay]["measurements"], parameter, func),
                }
            )
    elif interval == "weekly":
        currWeek = fromDate.isocalendar()[1]
        weekObservations = []
        for day in range(delta.days + 1):
            currDay = fromDate.date() + datetime.timedelta(days=day)
            # if we parsed through the previous week, load the week in the prepared data
            if currDay.isocalendar()[1] != currWeek:
                preparedData.append(
                    {
                        "x": str(currWeek),
                        "y": agrFunc(weekObservations, parameter, func),
                    }
                )
                weekObservations = []
```

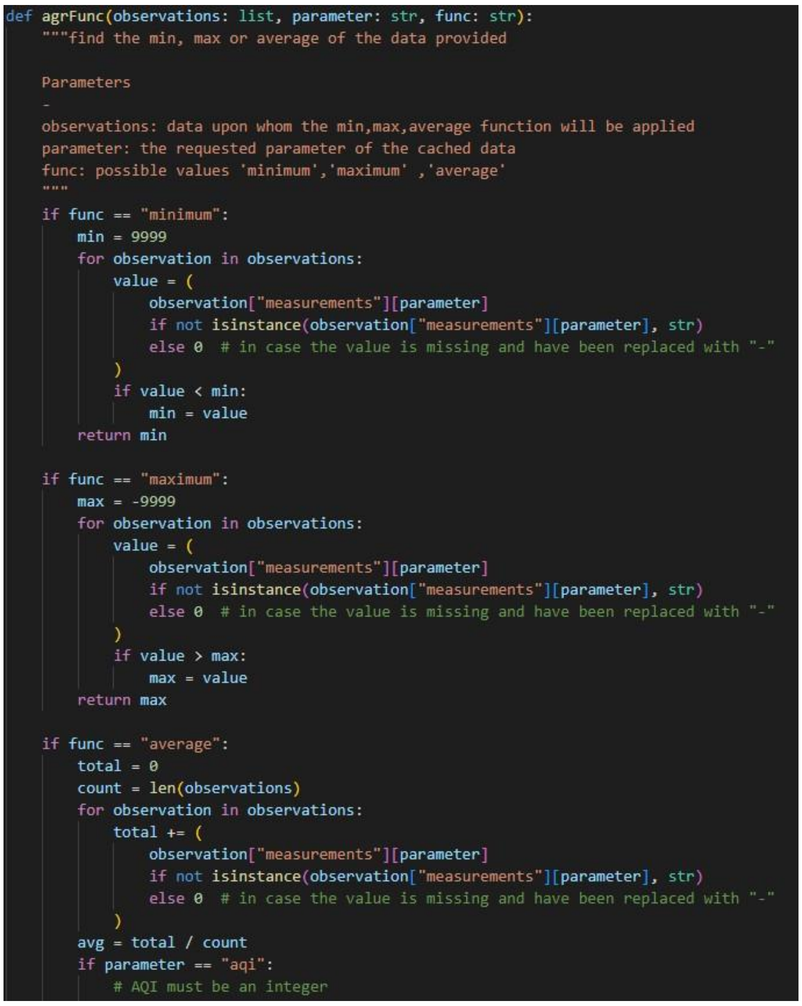


```python
                weekObservations = []
                currWeek = currDay.isocalendar()[1]
            weekObservations += historydata[currDay.isoformat()]["measurements"]
        preparedData.append(
            {
                "x": str(currWeek),
                "y": agrFunc(weekObservations, parameter, func),
            }
        )
    elif interval == "monthly":
        currMonth = fromDate.month
        monthObservations = []
        for day in range(delta.days + 1):
            currDay = fromDate.date() + datetime.timedelta(days=day)
            # if we pasrsed through the previous month, load the month in the prepared data
            if currDay.month != currMonth:
                preparedData.append(
                    {
                        "x": str(currMonth),
                        "y": agrFunc(monthObservations, parameter, func),
                    }
                )
                monthObservations = []
                currMonth = currDay.month
            monthObservations += historydata[currDay.isoformat()]["measurements"]
        preparedData.append(
            {
                "x": str(currMonth),
                "y": agrFunc(monthObservations, parameter, func),
            }
        )
    elif interval == "yearly":
        currYear = fromDate.year
        yearObservations = []
        for day in range(delta.days + 1):
            currDay = fromDate.date() + datetime.timedelta(days=day)
            # if we pasrsed through the previous year, load the year in the prepared data
            if currDay.year != currYear:
                preparedData.append(
                    {
                        "x": str(currYear),
                        "y": agrFunc(yearObservations, parameter, func),
                    }
```



```
                }
            )
            yearObservations = []
            currYear = currDay.year
        yearObservations += historydata[currDay.isoformat()]["measurements"]
    preparedData.append(
        {
            "x": str(currYear),
            "y": agrFunc(yearObservations, parameter, func),
        }
    )
return preparedData
```

Figure 19. *Process the data into intervals*

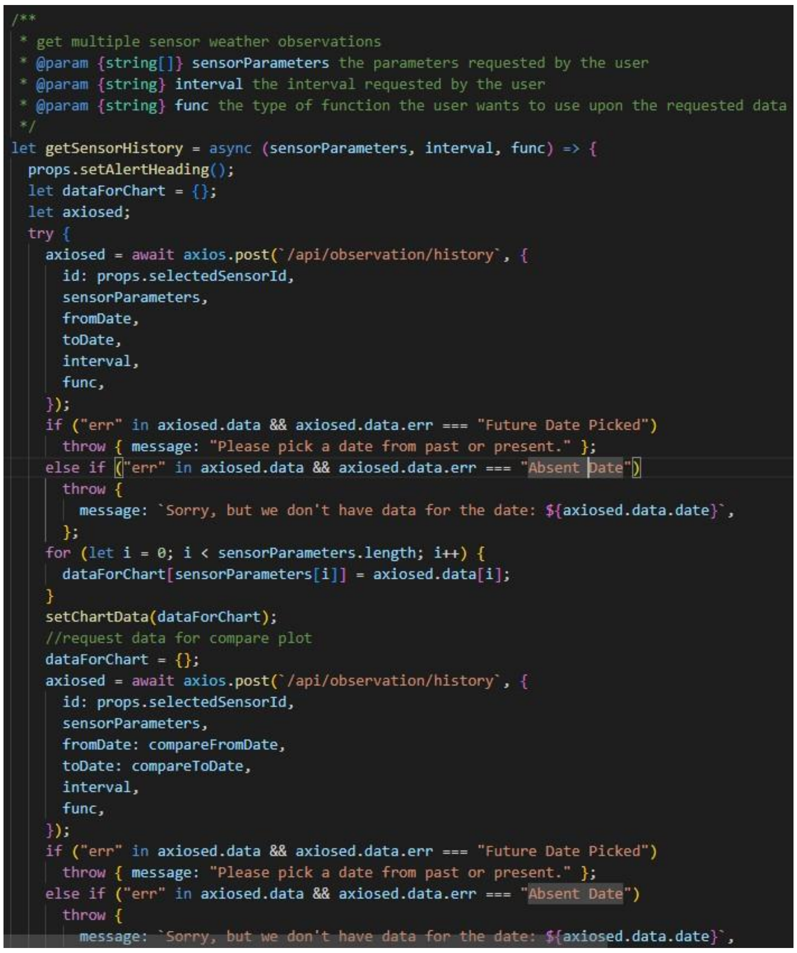


```
def agrFunc(observations: list, parameter: str, func: str):
    """find the min, max or average of the data provided

    Parameters
    -
    observations: data upon whom the min,max,average function will be applied
    parameter: the requested parameter of the cached data
    func: possible values 'minimum','maximum' ,'average'
    """
    if func == "minimum":
        min = 9999
        for observation in observations:
            value = (
                observation["measurements"][parameter]
                if not isinstance(observation["measurements"][parameter], str)
                else 0  # in case the value is missing and have been replaced with "-"
            )
            if value < min:
                min = value
        return min

    if func == "maximum":
        max = -9999
        for observation in observations:
            value = (
                observation["measurements"][parameter]
                if not isinstance(observation["measurements"][parameter], str)
                else 0  # in case the value is missing and have been replaced with "-"
            )
            if value > max:
                max = value
        return max

    if func == "average":
        total = 0
        count = len(observations)
        for observation in observations:
            total += (
                observation["measurements"][parameter]
                if not isinstance(observation["measurements"][parameter], str)
                else 0  # in case the value is missing and have been replaced with "-"
            )
        avg = total / count
        if parameter == "aqi":
            # AQI must be an integer
```

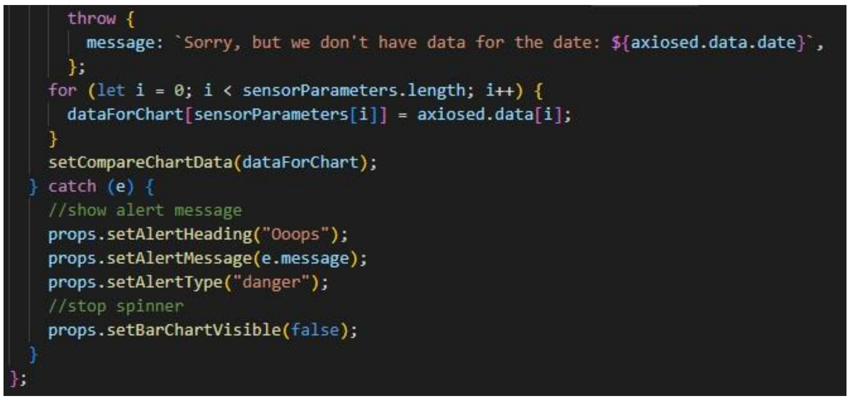
Figure 20*. Aggregate the data with the desired function*

3.4.2 Weather Plots processing chain: Frontend

- Call and retrieve data from the backend (**Figure 21**)
- Visualize the data with a plot (**Figure 22**)

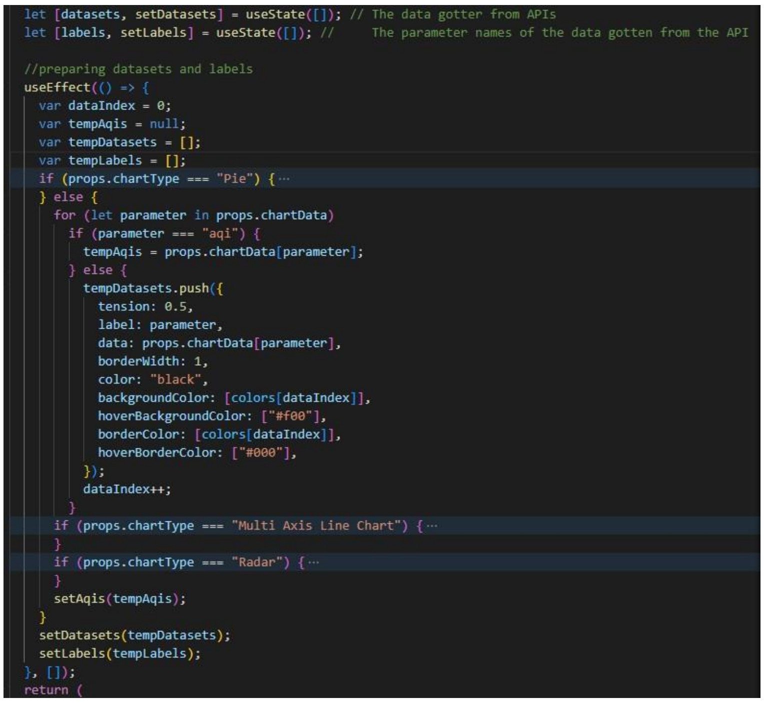


```
/**
 * get multiple sensor weather observations
 * @param {string[]} sensorParameters the parameters requested by the user
 * @param {string} interval the interval requested by the user
 * @param {string} func the type of function the user wants to use upon the requested data
 */
let getSensorHistory = async (sensorParameters, interval, func) => {
  props.setAlertHeading();
  let dataForChart = {};
  let axiosed;
  try {
    axiosed = await axios.post(`/api/observation/history`, {
      id: props.selectedSensorId,
      sensorParameters,
      fromDate,
      toDate,
      interval,
      func,
    });
    if ("err" in axiosed.data && axiosed.data.err === "Future Date Picked")
      throw { message: "Please pick a date from past or present." };
    else if ("err" in axiosed.data && axiosed.data.err === "Absent Date")
      throw {
        message: `Sorry, but we don't have data for the date: ${axiosed.data.date}`,
      };
    for (let i = 0; i < sensorParameters.length; i++) {
      dataForChart[sensorParameters[i]] = axiosed.data[i];
    }
    setChartData(dataForChart);
    //request data for compare plot
    dataForChart = {};
    axiosed = await axios.post(`/api/observation/history`, {
      id: props.selectedSensorId,
      sensorParameters,
      fromDate: compareFromDate,
      toDate: compareToDate,
      interval,
      func,
    });
    if ("err" in axiosed.data && axiosed.data.err === "Future Date Picked")
      throw { message: "Please pick a date from past or present." };
    else if ("err" in axiosed.data && axiosed.data.err === "Absent Date")
      throw {
        message: `Sorry, but we don't have data for the date: ${axiosed.data.date}`,
```

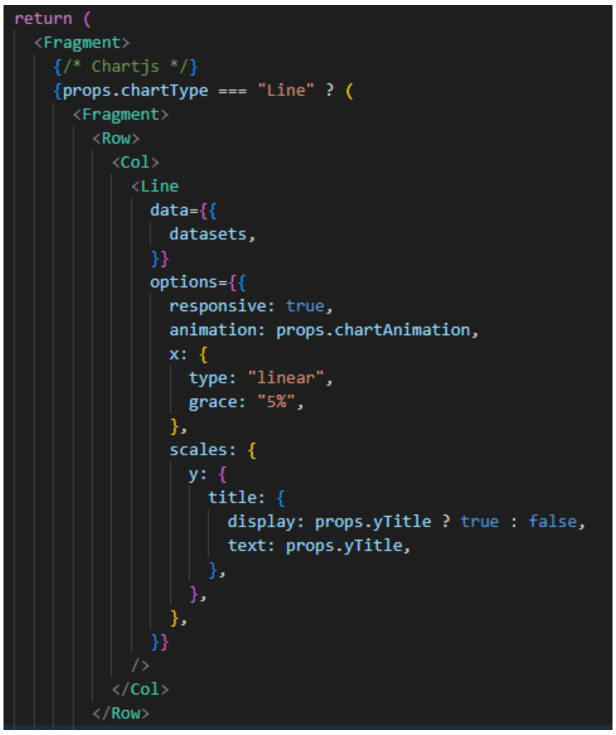


```
      throw {
        message: `Sorry, but we don't have data for the date: ${axiosed.data.date}`,
      };
    for (let i = 0; i < sensorParameters.length; i++) {
      dataForChart[sensorParameters[i]] = axiosed.data[i];
    }
    setCompareChartData(dataForChart);
  } catch (e) {
    //show alert message
    props.setAlertHeading("Ooops");
    props.setAlertMessage(e.message);
    props.setAlertType("danger");
    //stop spinner
    props.setBarChartVisible(false);
  }
};
```

***Figure 21**. Call and retrieve data from the backend*



```
let [datasets, setDatasets] = useState([]); // The data gotter from APIs
let [labels, setLabels] = useState([]); //     The parameter names of the data gotten from the API

//preparing datasets and labels
useEffect(() => {
  var dataIndex = 0;
  var tempAqis = null;
  var tempDatasets = [];
  var tempLabels = [];
  if (props.chartType === "Pie") {⋯
  } else {
    for (let parameter in props.chartData)
      if (parameter === "aqi") {
        tempAqis = props.chartData[parameter];
      } else {
        tempDatasets.push({
          tension: 0.5,
          label: parameter,
          data: props.chartData[parameter],
          borderWidth: 1,
          color: "black",
          backgroundColor: [colors[dataIndex]],
          hoverBackgroundColor: ["#f00"],
          borderColor: [colors[dataIndex]],
          hoverBorderColor: ["#000"],
        });
        dataIndex++;
      }
    if (props.chartType === "Multi Axis Line Chart") {⋯
    }
    if (props.chartType === "Radar") {⋯
    }
    setAqis(tempAqis);
  }
  setDatasets(tempDatasets);
  setLabels(tempLabels);
}, []);
return (
```



```
return (
  <Fragment>
    {/* Chartjs */}
    {props.chartType === "Line" ? (
      <Fragment>
        <Row>
          <Col>
            <Line
              data={{
                datasets,
              }}
              options={{
                responsive: true,
                animation: props.chartAnimation,
                x: {
                  type: "linear",
                  grace: "5%",
                },
                scales: {
                  y: {
                    title: {
                      display: props.yTitle ? true : false,
                      text: props.yTitle,
                    },
                  },
                },
              }}
            />
          </Col>
        </Row>
```

Figure 22. *Visualize the data with a plot*

3.4.2 Pollution Plots processing chain: Backend

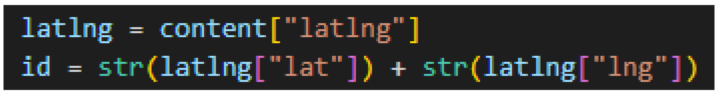
- Check for errors in the data provided and check if some or all data requested are in the local cache

  The only difference is that it has coordinates and the id is made from them (Figure 23**)*
- Get data from OpenWeatherMap (**Figure 24**)
- Process the data into intervals

Same as weather plots (Figure 19**)*

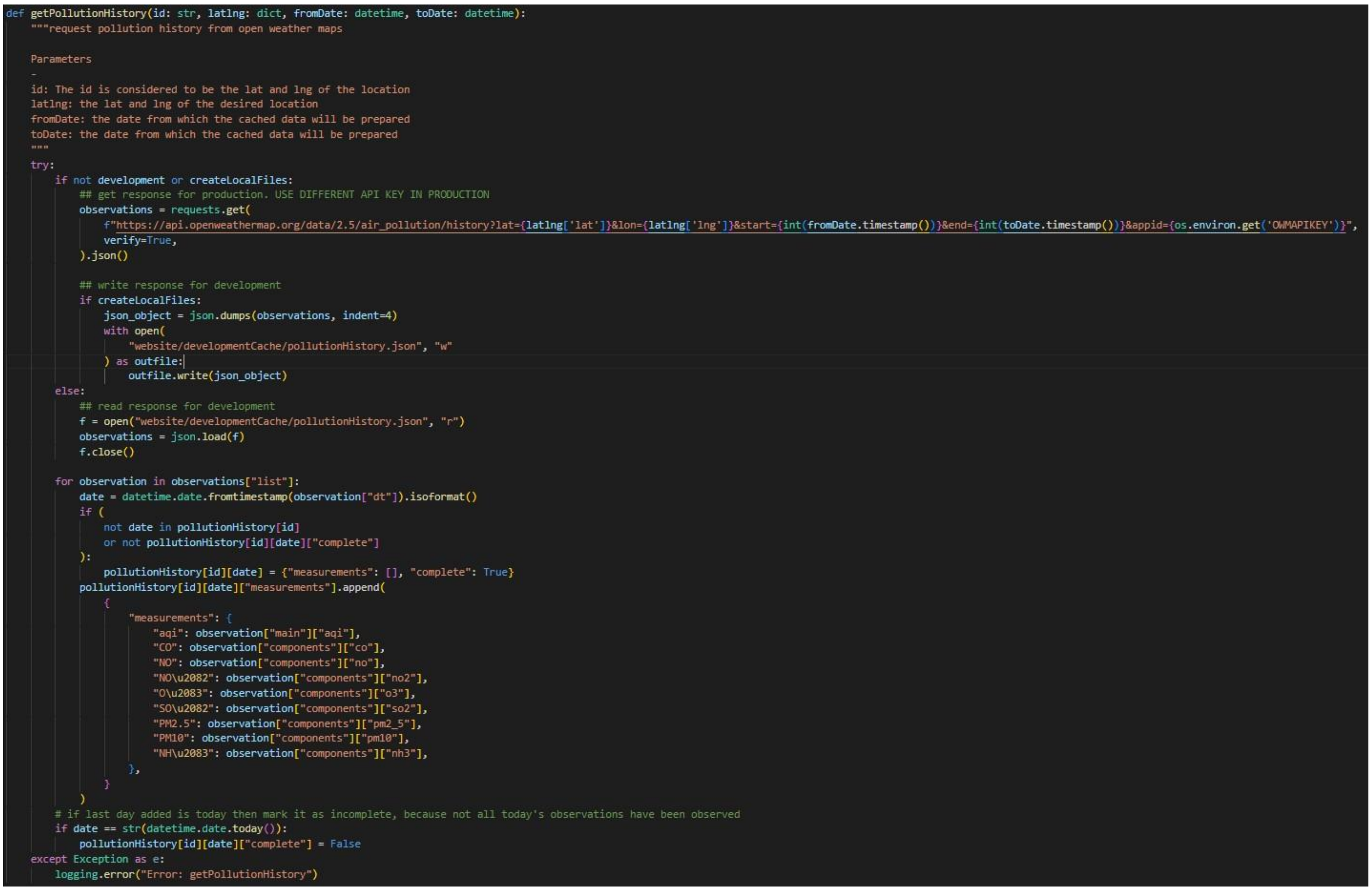
- Aggregate the data with the desired function

Same as weather plots (Figure 20**)*



```
latlng = content["latlng"]
id = str(latlng["lat"]) + str(latlng["lng"])
```

***Figure 23.** Coordinates*



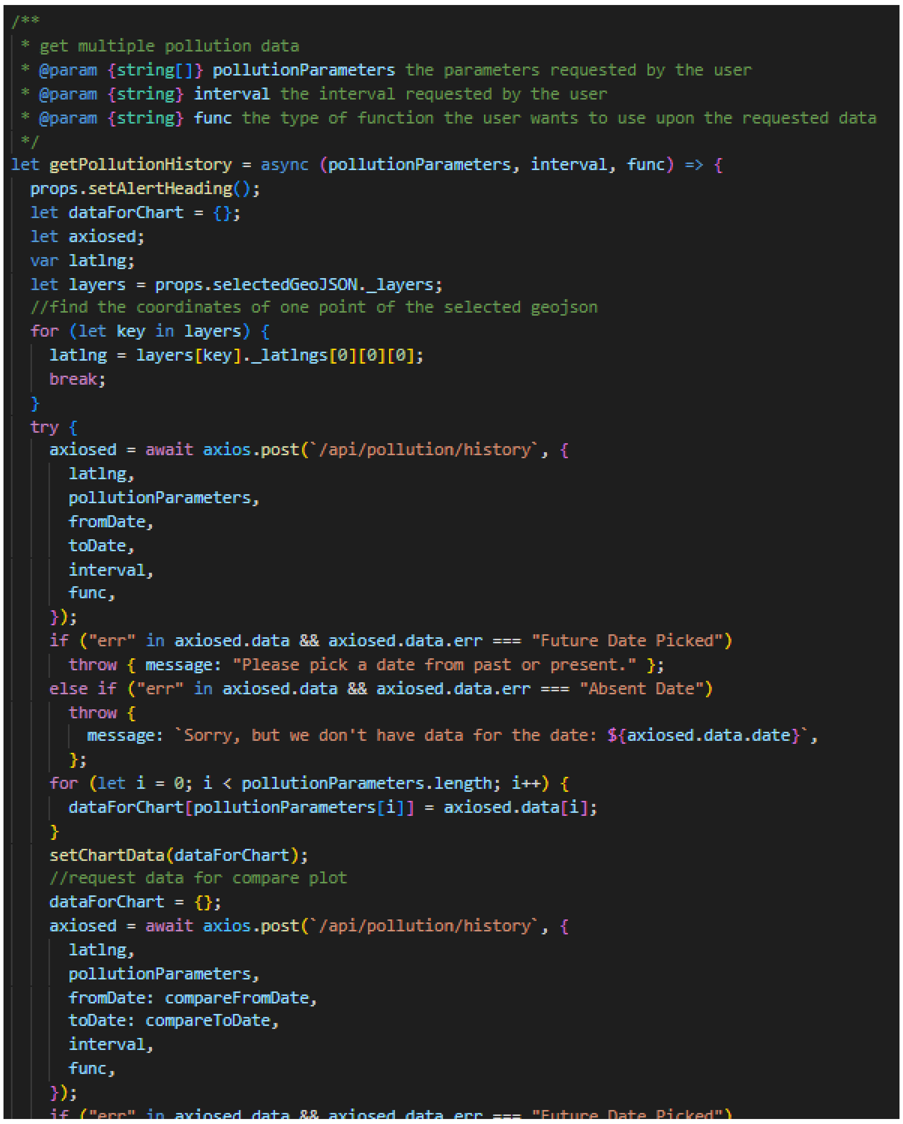
```
def getPollutionHistory(id: str, latlng: dict, fromDate: datetime, toDate: datetime):
    """request pollution history from open weather maps

    Parameters
    -
    id: The id is considered to be the lat and lng of the location
    latlng: the lat and lng of the desired location
    fromDate: the date from which the cached data will be prepared
    toDate: the date from which the cached data will be prepared
    """
    try:
        if not development or createLocalFiles:
            ## get response for production. USE DIFFERENT API KEY IN PRODUCTION
            observations = requests.get(
                f"https://api.openweathermap.org/data/2.5/air_pollution/history?lat={latlng['lat']}&lon={latlng['lng']}&start={int(fromDate.timestamp())}&end={int(toDate.timestamp())}&appid={os.environ.get('OWMAPIKEY')}",
                verify=True,
            ).json()

            ## write response for development
            if createLocalFiles:
                json_object = json.dumps(observations, indent=4)
                with open(
                    "website/developmentCache/pollutionHistory.json", "w"
                ) as outfile:
                    outfile.write(json_object)
        else:
            ## read response for development
            f = open("website/developmentCache/pollutionHistory.json", "r")
            observations = json.load(f)
            f.close()

        for observation in observations["list"]:
            date = datetime.date.fromtimestamp(observation["dt"]).isoformat()
            if (
                not date in pollutionHistory[id]
                or not pollutionHistory[id][date]["complete"]
            ):
                pollutionHistory[id][date] = {"measurements": [], "complete": True}
            pollutionHistory[id][date]["measurements"].append(
                {
                    "measurements": {
                        "aqi": observation["main"]["aqi"],
                        "CO": observation["components"]["co"],
                        "NO": observation["components"]["no"],
                        "NO\u2082": observation["components"]["no2"],
                        "O\u2083": observation["components"]["o3"],
                        "SO\u2082": observation["components"]["so2"],
                        "PM2.5": observation["components"]["pm2_5"],
                        "PM10": observation["components"]["pm10"],
                        "NH\u2083": observation["components"]["nh3"],
                    },
                }
            )
        # if last day added is today then mark it as incomplete, because not all today's observations have been observed
        if date == str(datetime.date.today()):
            pollutionHistory[id][date]["complete"] = False
    except Exception as e:
        logging.error("Error: getPollutionHistory")
```

***Figure 24.** Get data from OpenWeatherMap*

3.4.3 Pollution Plots processing chain: Frontend

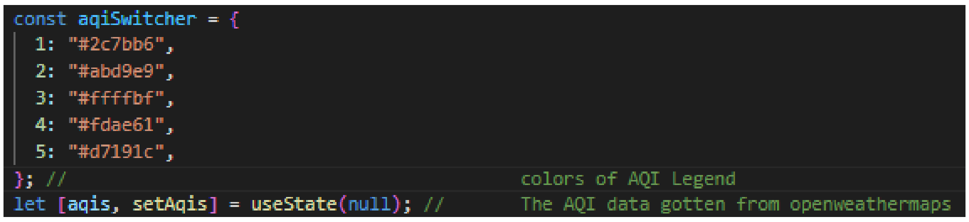
- Call and retrieve data from the backend (**Figure 25**)
- Visualize the data with a plot (**Figure 26**)

*Same as the weather plots with some additions

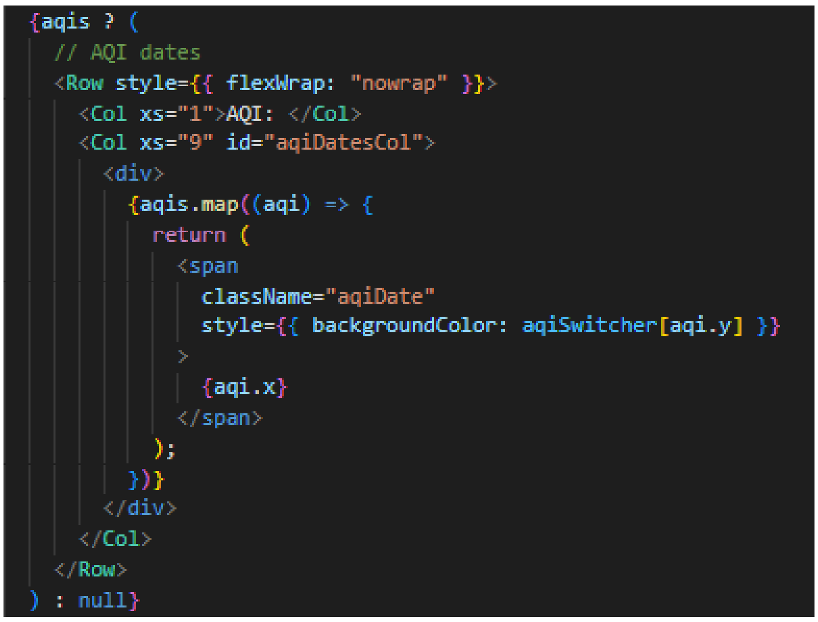


```
/**
 * get multiple pollution data
 * @param {string[]} pollutionParameters the parameters requested by the user
 * @param {string} interval the interval requested by the user
 * @param {string} func the type of function the user wants to use upon the requested data
 */
let getPollutionHistory = async (pollutionParameters, interval, func) => {
  props.setAlertHeading();
  let dataForChart = {};
  let axiosed;
  var latlng;
  let layers = props.selectedGeoJSON._layers;
  //find the coordinates of one point of the selected geojson
  for (let key in layers) {
    latlng = layers[key]._latlngs[0][0][0];
    break;
  }
  try {
    axiosed = await axios.post(`/api/pollution/history`, {
      latlng,
      pollutionParameters,
      fromDate,
      toDate,
      interval,
      func,
    });
    if ("err" in axiosed.data && axiosed.data.err === "Future Date Picked")
      throw { message: "Please pick a date from past or present." };
    else if ("err" in axiosed.data && axiosed.data.err === "Absent Date")
      throw {
        message: `Sorry, but we don't have data for the date: ${axiosed.data.date}`,
      };
    for (let i = 0; i < pollutionParameters.length; i++) {
      dataForChart[pollutionParameters[i]] = axiosed.data[i];
    }
    setChartData(dataForChart);
    //request data for compare plot
    dataForChart = {};
    axiosed = await axios.post(`/api/pollution/history`, {
      latlng,
      pollutionParameters,
      fromDate: compareFromDate,
      toDate: compareToDate,
      interval,
      func,
    });
    if ("err" in axiosed.data && axiosed.data.err === "Future Date Picked")
```

Figure 25. *Call and retrieve data from the backend*



```
const aqiSwitcher = {
  1: "#2c7bb6",
  2: "#abd9e9",
  3: "#ffffbf",
  4: "#fdae61",
  5: "#d7191c",
}; //                                     colors of AQI Legend
let [aqis, setAqis] = useState(null); //   The AQI data gotten from openweathermaps
```

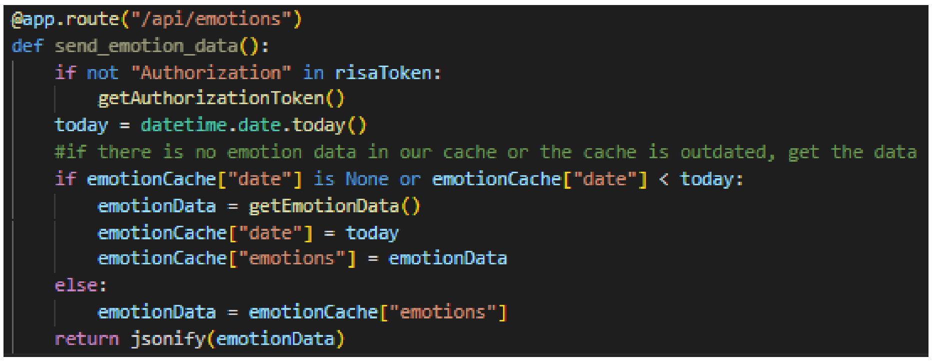


```
{aqis ? (
  // AQI dates
  <Row style={{ flexWrap: "nowrap" }}>
    <Col xs="1">AQI: </Col>
    <Col xs="9" id="aqiDatesCol">
      <div>
        {aqis.map((aqi) => {
          return (
            <span
              className="aqiDate"
              style={{ backgroundColor: aqiSwitcher[aqi.y] }}
            >
              {aqi.x}
            </span>
          );
        })}
      </div>
    </Col>
  </Row>
) : null}
```

Figure 26. *Visualize the data with a plot and additions compared to weather plots*

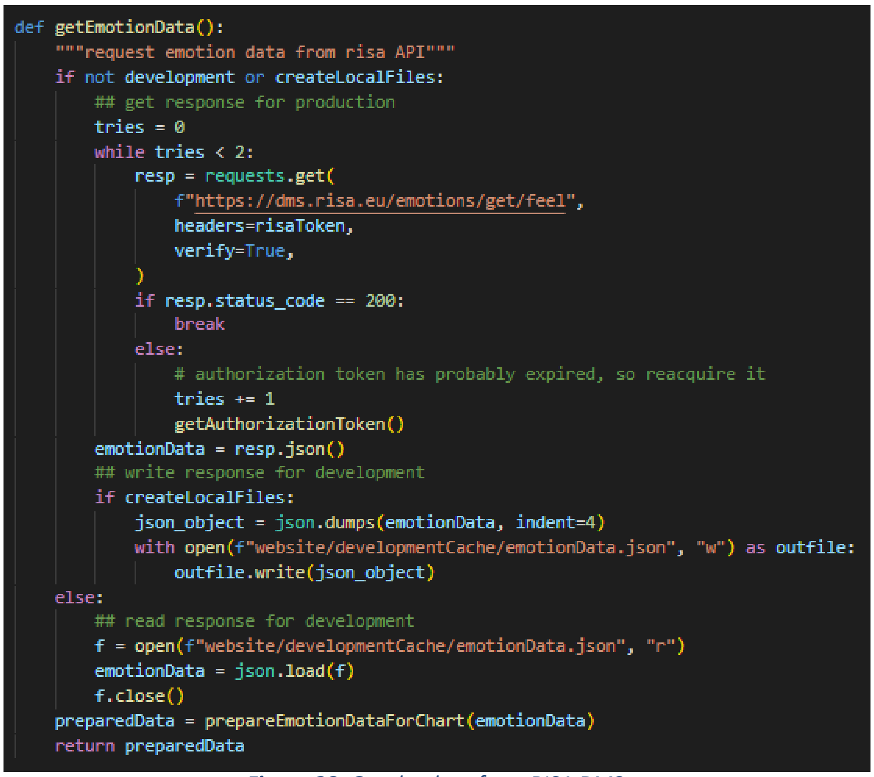
3.4.3 Emotion Pie Chart: Backend

- Check if there is up-to-date emotion data in the n local cache (**Figure 27**)
- Get the data from RISA DMS (**Figure 28**)
- Aggregate emotion values into emotion categories, then into dates, then into cities (**Figure 29**)



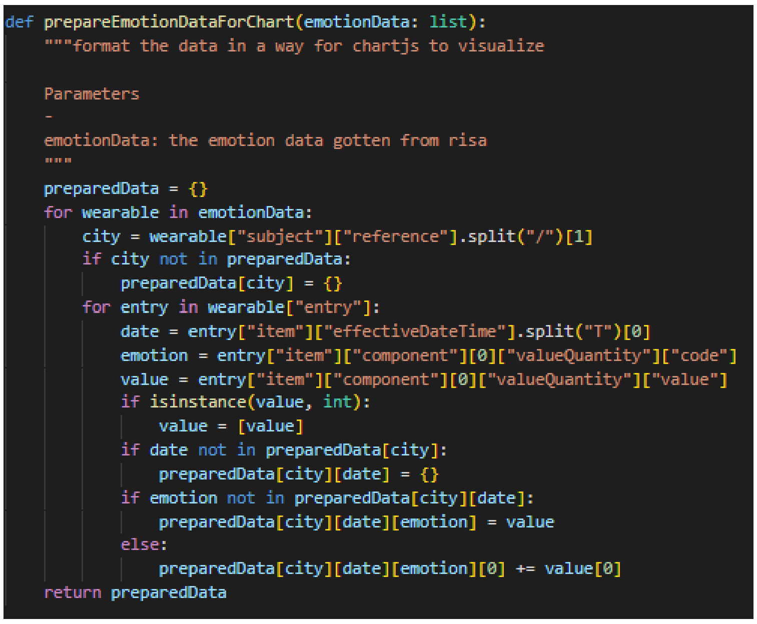
```
@app.route("/api/emotions")
def send_emotion_data():
    if not "Authorization" in risaToken:
        getAuthorizationToken()
    today = datetime.date.today()
    #if there is no emotion data in our cache or the cache is outdated, get the data
    if emotionCache["date"] is None or emotionCache["date"] < today:
        emotionData = getEmotionData()
        emotionCache["date"] = today
        emotionCache["emotions"] = emotionData
    else:
        emotionData = emotionCache["emotions"]
    return jsonify(emotionData)
```

Figure 27. *Check if there is up-to-date emotion data in the local cache*



```python
def getEmotionData():
    """request emotion data from risa API"""
    if not development or createLocalFiles:
        ## get response for production
        tries = 0
        while tries < 2:
            resp = requests.get(
                f"https://dms.risa.eu/emotions/get/feel",
                headers=risaToken,
                verify=True,
            )
            if resp.status_code == 200:
                break
            else:
                # authorization token has probably expired, so reacquire it
                tries += 1
                getAuthorizationToken()
        emotionData = resp.json()
        ## write response for development
        if createLocalFiles:
            json_object = json.dumps(emotionData, indent=4)
            with open(f"website/developmentCache/emotionData.json", "w") as outfile:
                outfile.write(json_object)
    else:
        ## read response for development
        f = open(f"website/developmentCache/emotionData.json", "r")
        emotionData = json.load(f)
        f.close()
    preparedData = prepareEmotionDataForChart(emotionData)
    return preparedData
```

Figure 28. *Get the data from RISA DMS*

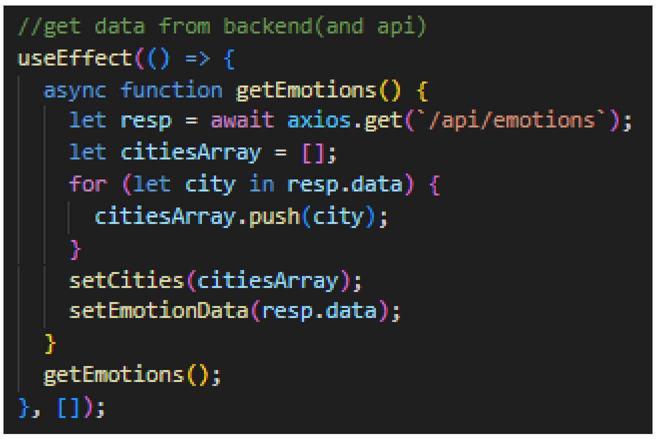


```python
def prepareEmotionDataForChart(emotionData: list):
    """format the data in a way for chartjs to visualize

    Parameters
    -
    emotionData: the emotion data gotten from risa
    """
    preparedData = {}
    for wearable in emotionData:
        city = wearable["subject"]["reference"].split("/")[1]
        if city not in preparedData:
            preparedData[city] = {}
        for entry in wearable["entry"]:
            date = entry["item"]["effectiveDateTime"].split("T")[0]
            emotion = entry["item"]["component"][0]["valueQuantity"]["code"]
            value = entry["item"]["component"][0]["valueQuantity"]["value"]
            if isinstance(value, int):
                value = [value]
            if date not in preparedData[city]:
                preparedData[city][date] = {}
            if emotion not in preparedData[city][date]:
                preparedData[city][date][emotion] = value
            else:
                preparedData[city][date][emotion][0] += value[0]
    return preparedData
```

Figure 29*. Aggregate emotion values into emotion categories*

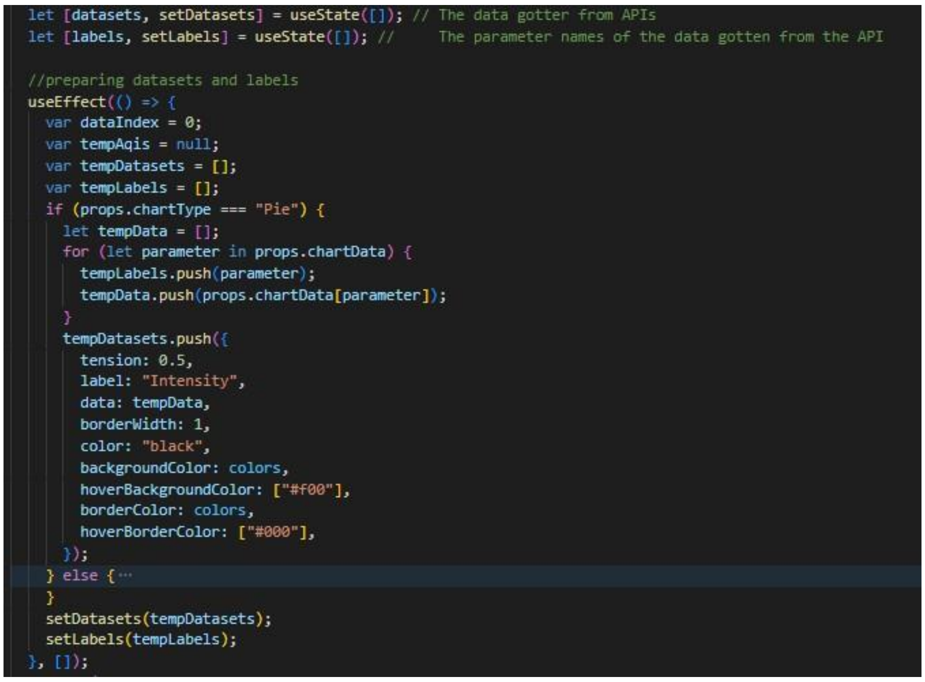
3.4.4 Emotion Pie Chart: Frontend

- Get emotion data from the backend (**Figure 30**)
- Visualize data into Pie Chart (**Figure 31**)

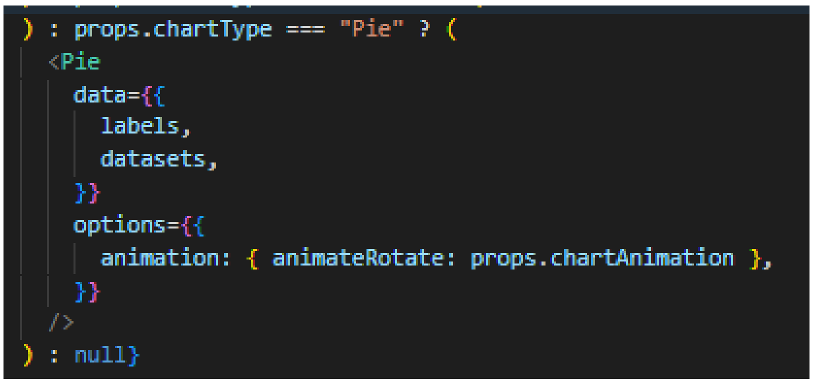


```
//get data from backend(and api)
useEffect(() => {
  async function getEmotions() {
    let resp = await axios.get(`/api/emotions`);
    let citiesArray = [];
    for (let city in resp.data) {
      citiesArray.push(city);
    }
    setCities(citiesArray);
    setEmotionData(resp.data);
  }
  getEmotions();
}, []);
```

Figure 30. *Get emotion data from the backend*



```
let [datasets, setDatasets] = useState([]); // The data gotter from APIs
let [labels, setLabels] = useState([]); //     The parameter names of the data gotten from the API

//preparing datasets and labels
useEffect(() => {
  var dataIndex = 0;
  var tempAqis = null;
  var tempDatasets = [];
  var tempLabels = [];
  if (props.chartType === "Pie") {
    let tempData = [];
    for (let parameter in props.chartData) {
      tempLabels.push(parameter);
      tempData.push(props.chartData[parameter]);
    }
    tempDatasets.push({
      tension: 0.5,
      label: "Intensity",
      data: tempData,
      borderWidth: 1,
      color: "black",
      backgroundColor: colors,
      hoverBackgroundColor: ["#f00"],
      borderColor: colors,
      hoverBorderColor: ["#000"],
    });
  } else {⋯
  }
  setDatasets(tempDatasets);
  setLabels(tempLabels);
}, []);
```



```
) : props.chartType === "Pie" ? (
  <Pie
    data={{
      labels,
      datasets,
    }}
    options={{
      animation: { animateRotate: props.chartAnimation },
    }}
  />
) : null}
```

Figure 31. *Visualize data into Pie Chart*

4 euPOLIS Visualisation Module functionalities & user manual

Within the framework of euPOLIS, a WebGIS platform was developed for the visualization of the project's outcome. In this section screenshots from the platform will be given, providing a step-by-step explanation and navigation throughout the euPOLIS Visualisation Module.

4.1 euPOLIS WebGIS Interface

Once the user enters the euPOLIS Platform, the loading screen appears (**Figure 32**) and then the main interface of the Platform is accessible, as illustrated in **Figure 33.**

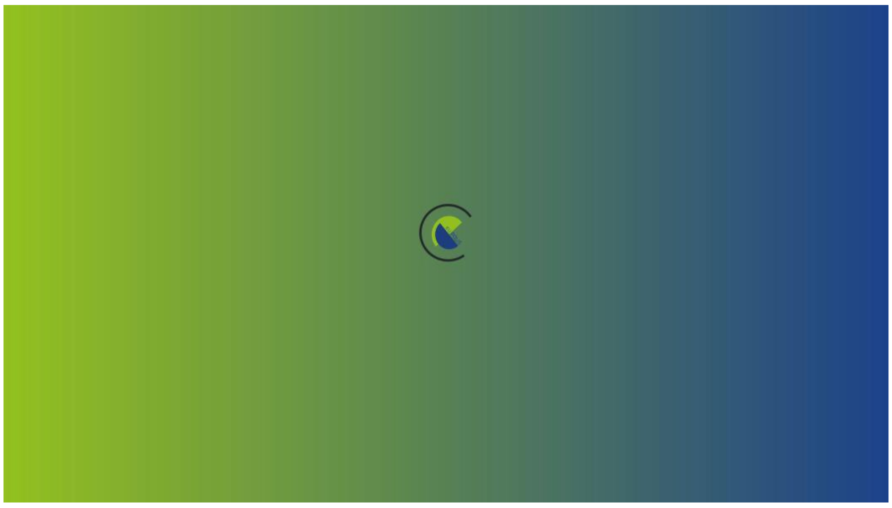


Figure 32. *Loading screen of the euPOLIS Platform.*

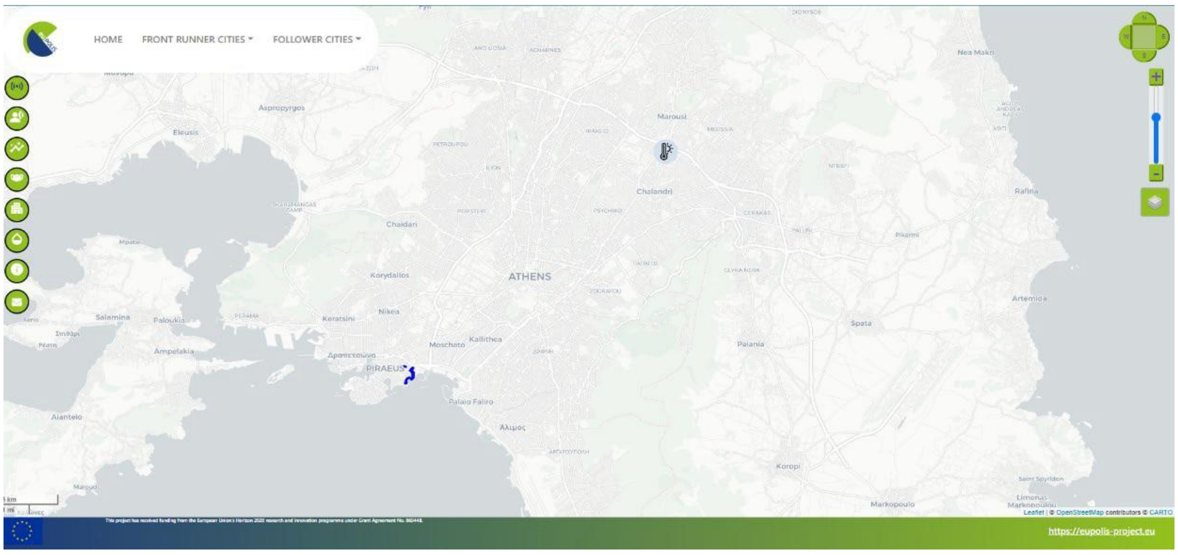


Figure 33. *The main interface of the euPOLIS Platform.*

4.2 General Features

The euPOLIS platform can be described by the following features:

1. The Main NavBar
    - Home
    - Front Runner Cities
    - Follower Cities
        - European Cities
        - International Cities
2. A Sidebar

    **Main Navigation tools described in section 4.3*
3. The Map tools
    - Pan (N, E, S, W)
    - Zoom in, Zoom out
    - Layers Selection/Background Map
4. Area of Interest/ Polygons

More specifically:

1. **The Main NavBar**

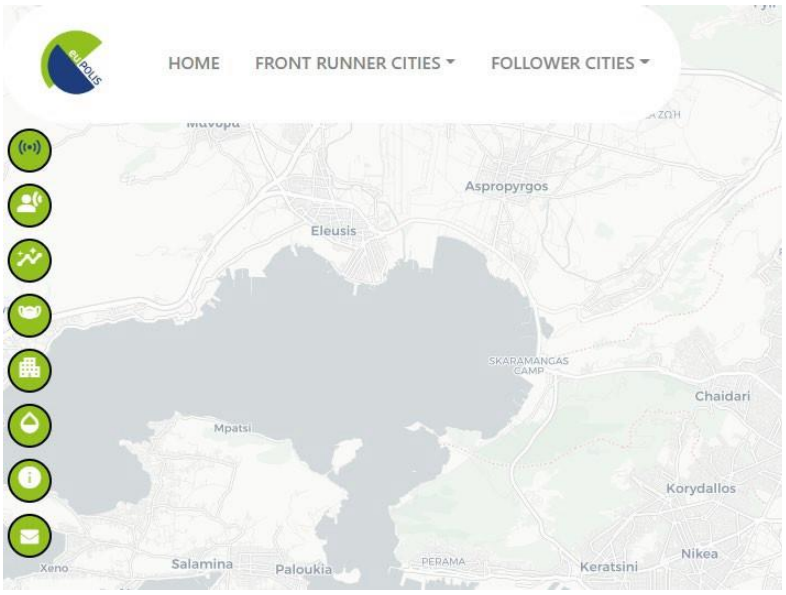


Figure 34. *The Main Toolbar is on the upper top of the Platform.*

The Main NavBar (**Figure 34**) consists of three buttons: (1) “Home”, (2) “Front Runner Cities” which are the City of Belgrade, City of Piraeus, City of Lodz, and Gladsaxe Municipality (**Figure 35**), and (3) “Follower Cities” consisting of City of Trebinje, City of Limassol, City of Palermo and City of Bogota (**Figure 36**).

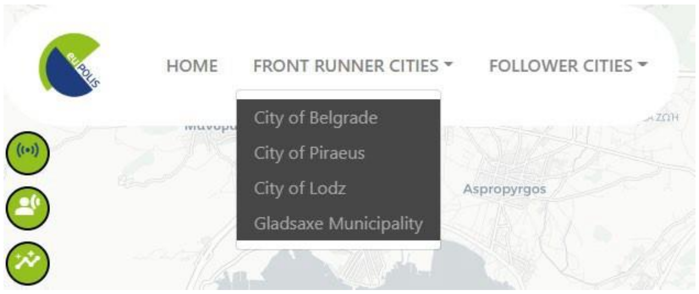


***Figure 35**. Main NavBar- Front Runner Cities (Demonstration Cities): the City of Belgrade, City of Piraeus, City of Lodz, and Gladsaxe Municipality*

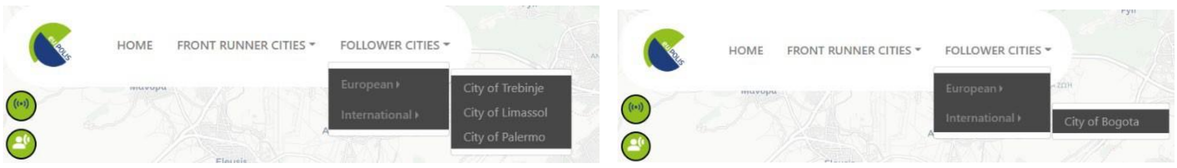


***Figure 36.** Main NavBar- Follower Cities (Demonstration Cities): (a)On the left image are the European cities: the City of Trebinje, the City of Limassol, and the City of Palermo, and (b) On the right image are the international cities: City of Bogota*

2. The Map tools

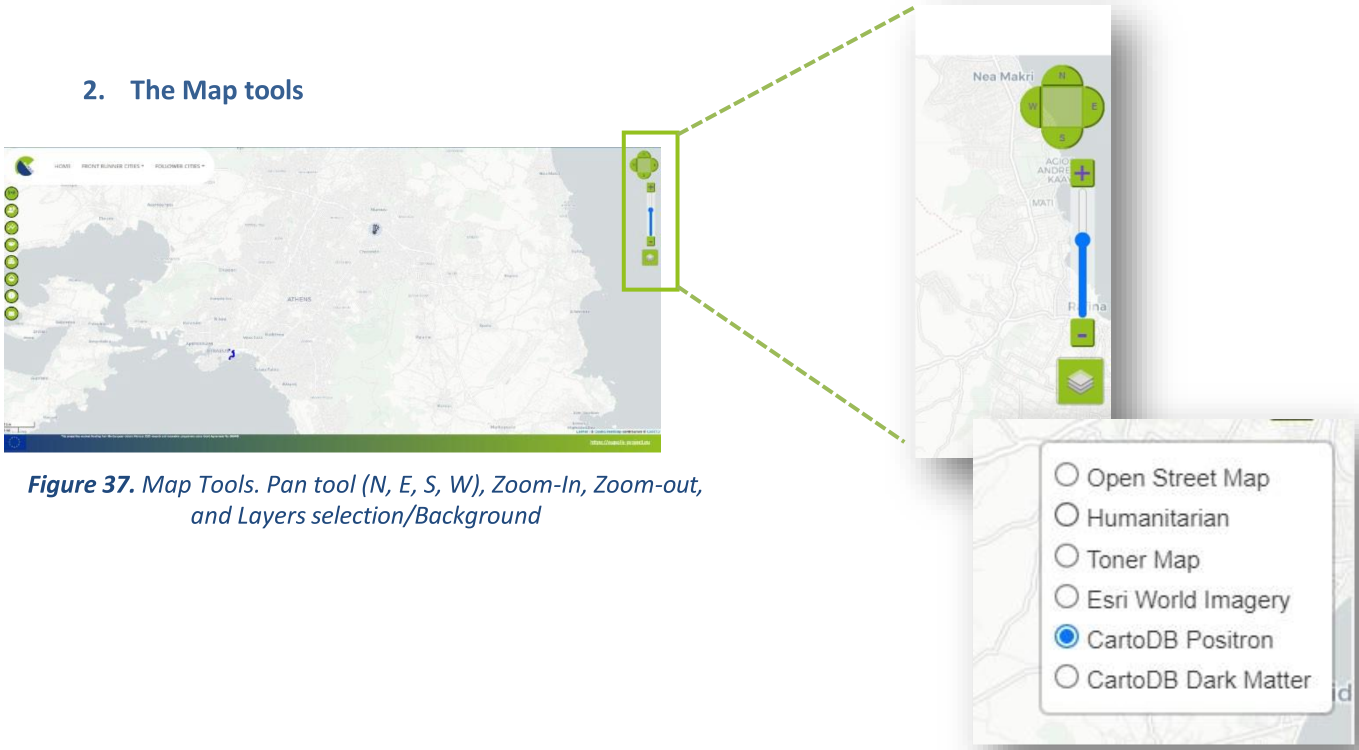


***Figure 37.** Map Tools. Pan tool (N, E, S, W), Zoom-In, Zoom-out, and Layers selection/Background*

3. A Sidebar

Main Navigation tools described in **section 4.3*

4. Area of Interest/Polygons

The demonstration sites can be found on the map, and are indicated by their polygons, as can be seen in **Figures 38, 40, 42,** and **44.** It should be mentioned that when the end user selects one of the cities either Front Runner or Follower the map zooms in on the corresponding area of interest.

City of Belgrade

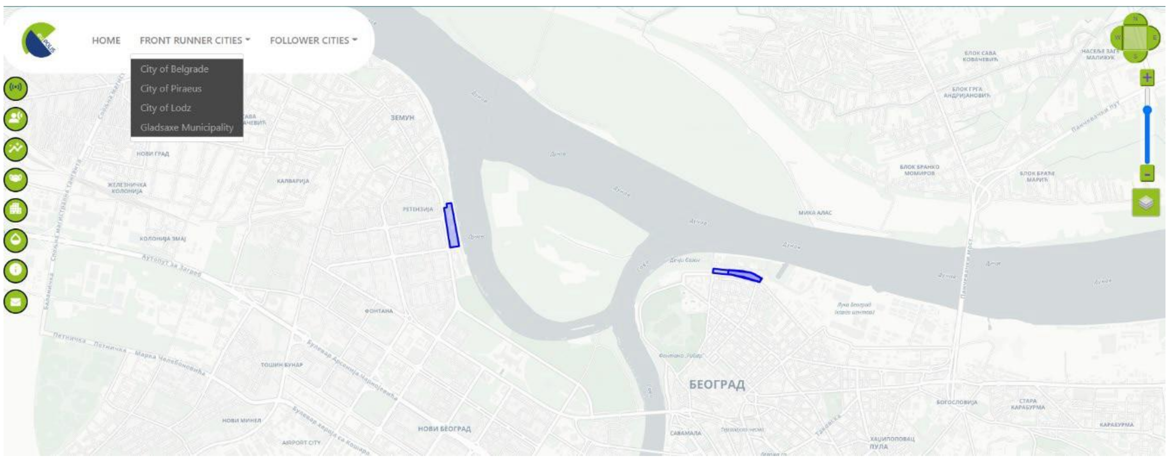
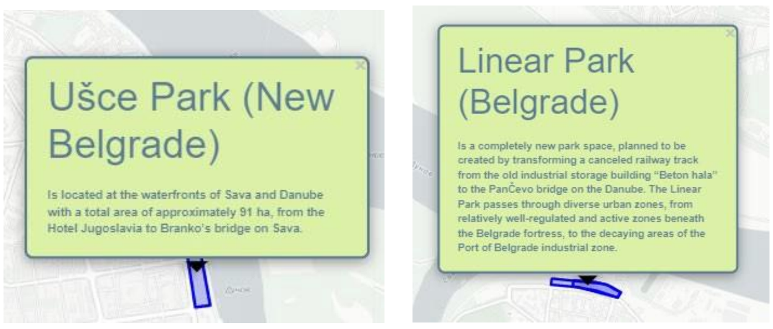


***Figure 38**. Polygon of the City of Belgrade. As indicated by the blue lines the two intervention sites are available, called "Usce Park" (on the left) and "Linear Park" (on the right)*

The City of Belgrade is comprised of two intervention sites, one called "Usce Park" which can be found in the new Belgrade, and the other one called "Linear Park" on the old side of Belgrade. The polygons indicating the intervention areas are illustrated in **Figure 38**. Once the user selects the intervention site, a brief description of this area can be found (**Figure 39**).



Ušce Park (New Belgrade)

Is located at the waterfronts of Sava and Danube with a total area of approximately 91 ha, from the Hotel Jugoslavia to Branko's bridge on Sava.

Linear Park (Belgrade)

Is a completely new park space, planned to be created by transforming a canceled railway track from the old industrial storage building "Beton hala" to the PanČevo bridge on the Danube. The Linear Park passes through diverse urban zones, from relatively well-regulated and active zones beneath the Belgrade fortress, to the decaying areas of the Port of Belgrade industrial zone.

***Figure 39**. Brief description of the intervention sites in the City of Belgrade*

City of Piraeus

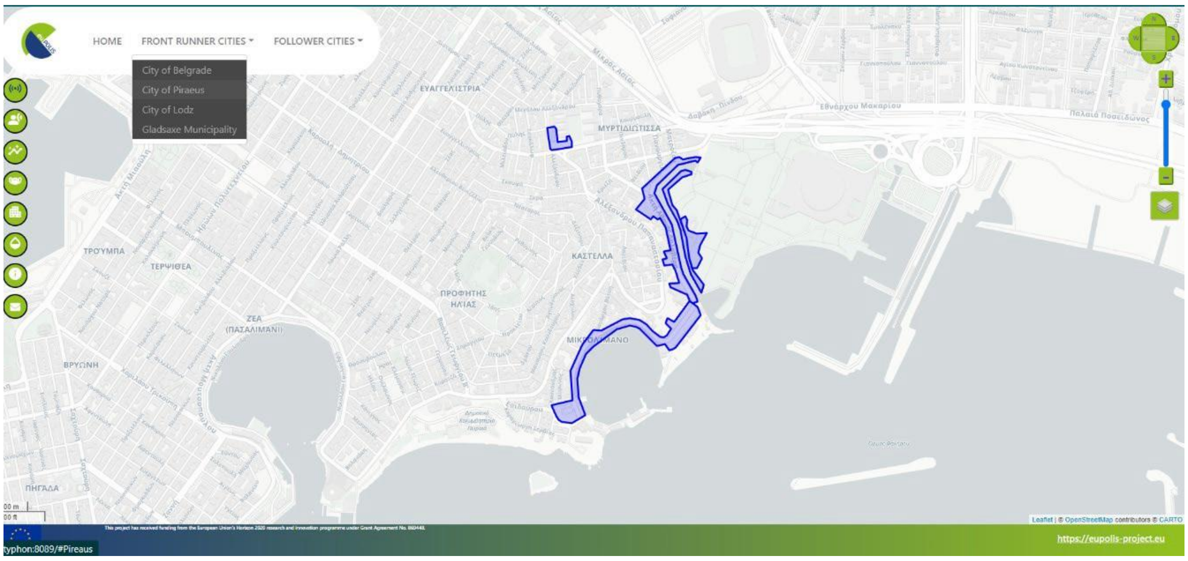
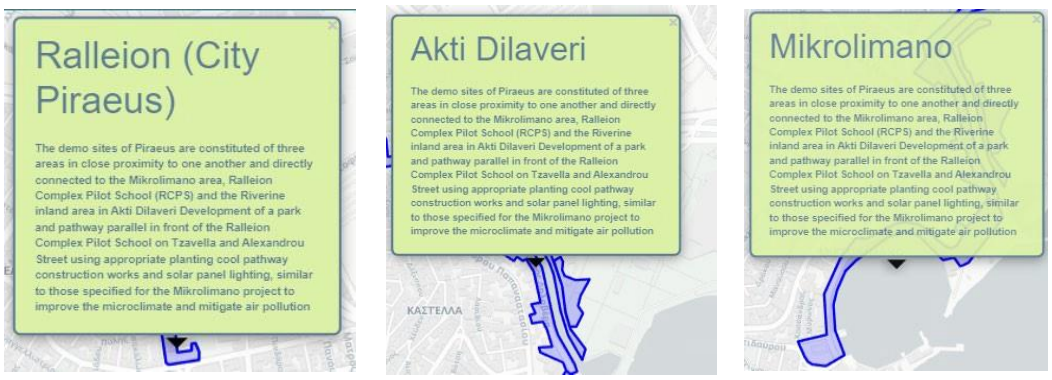


Figure 40. *Polygon of the City of Piraeus. As indicated by the blue lines the three intervention sites are available, called "Ralleion" (on the upper-left), "Microlimano" (on the bottom), and "Akti Dilaveri"*

The City of Piraeus is comprised of three intervention sites, called "Ralleion", "Microlimano" and "Akti Dilaveri". The polygons indicating the intervention areas are illustrated in **Figure 40**. Once the user selects the intervention site, a brief description of this area can be found (**Figure 41**).



Ralleion (City Piraeus)

The demo sites of Piraeus are constituted of three areas in close proximity to one another and directly connected to the Mikrolimano area, Ralleion Complex Pilot School (RCPS) and the Riverine inland area in Akti Dilaveri Development of a park and pathway parallel in front of the Ralleion Complex Pilot School on Tzavella and Alexandrou Street using appropriate planting cool pathway construction works and solar panel lighting, similar to those specified for the Mikrolimano project to improve the microclimate and mitigate air pollution

Akti Dilaveri

The demo sites of Piraeus are constituted of three areas in close proximity to one another and directly connected to the Mikrolimano area, Ralleion Complex Pilot School (RCPS) and the Riverine inland area in Akti Dilaveri Development of a park and pathway parallel in front of the Ralleion Complex Pilot School on Tzavella and Alexandrou Street using appropriate planting cool pathway construction works and solar panel lighting, similar to those specified for the Mikrolimano project to improve the microclimate and mitigate air pollution

Mikrolimano

The demo sites of Piraeus are constituted of three areas in close proximity to one another and directly connected to the Mikrolimano area, Ralleion Complex Pilot School (RCPS) and the Riverine inland area in Akti Dilaveri Development of a park and pathway parallel in front of the Ralleion Complex Pilot School on Tzavella and Alexandrou Street using appropriate planting cool pathway construction works and solar panel lighting, similar to those specified for the Mikrolimano project to improve the microclimate and mitigate air pollution

Figure 41. *Brief description of the intervention sites in the City of Piraeus*

City of Lodz

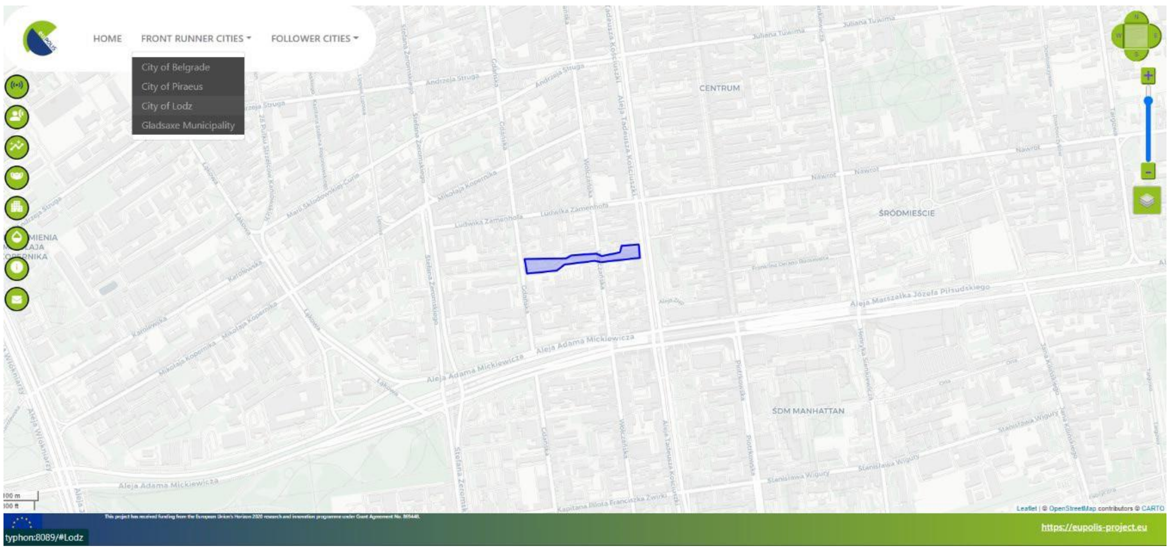
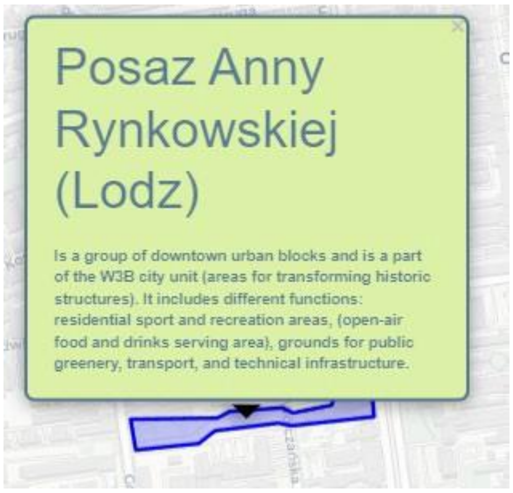


***Figure 42**. Polygon of the City of Lodz. As indicated by the blue lines the one intervention site is available, called "Posaz Anny Rynkowskiej"*

The City of Lodz is comprised of one intervention site, called "Posaz Anny Rynkowskiej". The polygon indicating the intervention area is illustrated in **Figure 42**. Once the user selects the intervention site, a brief description of this area can be found (**Figure 43**).



Posaz Anny Rynkowskiej (Lodz)

Is a group of downtown urban blocks and is a part of the W3B city unit (areas for transforming historic structures). It includes different functions: residential sport and recreation areas, (open-air food and drinks serving area), grounds for public greenery, transport, and technical infrastructure.

***Figure 43**. Brief description of the intervention site in the City of Lodz*

Gladsaxe Municipality

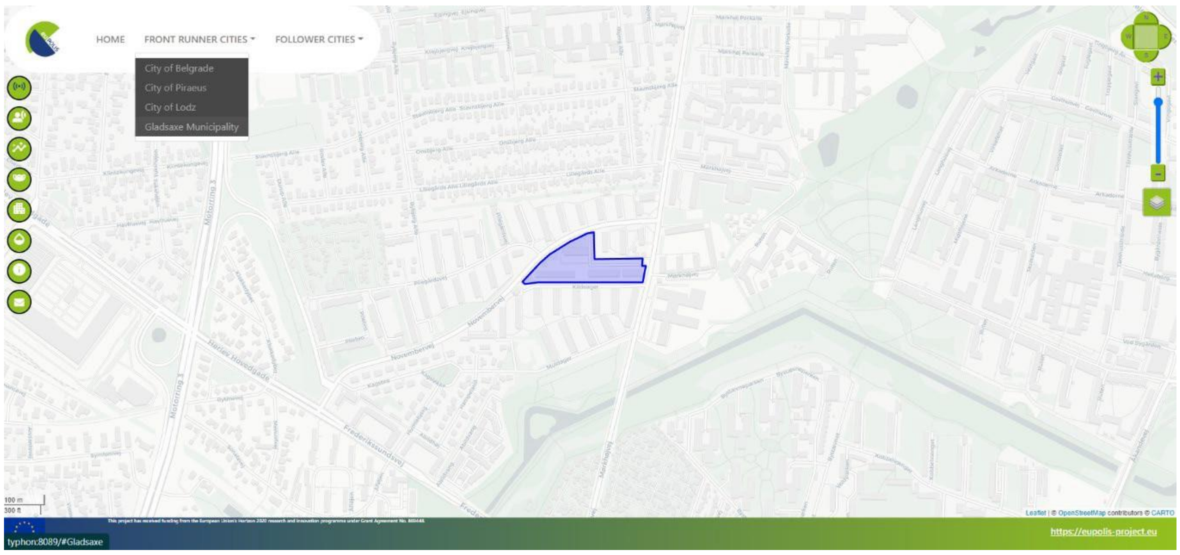
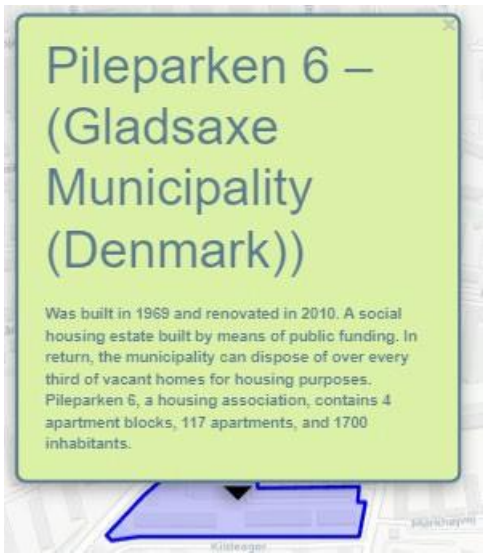


***Figure 44**. Polygon of the Gladsaxe Municipality. As indicated by the blue lines the one intervention site is available, called "Pileparken 6"*

Gladsaxe Municipality is comprised of one intervention site, called "Pileparken 6". The polygon indicating the intervention area is illustrated in **Figure 44**. Once the user selects the intervention site, a brief description of this area can be found (**Figure 45**).



Pileparken 6 – (Gladsaxe Municipality (Denmark))

Was built in 1969 and renovated in 2010. A social housing estate built by means of public funding. In return, the municipality can dispose of over every third of vacant homes for housing purposes. Pileparken 6, a housing association, contains 4 apartment blocks, 117 apartments, and 1700 inhabitants.

***Figure 45**. Brief description of the intervention site in the Gladsaxe Municipality*

4.3 Main Navigation Tools

A user can easily navigate within the euPOLIS Visualisation Module. To achieve easier and quicker navigation within the eVM, a variety of tools/sub-menus have been created. The main tools which are available in the interface of the platform are shown in **Figure 46** and can be found on the left part of the platform.

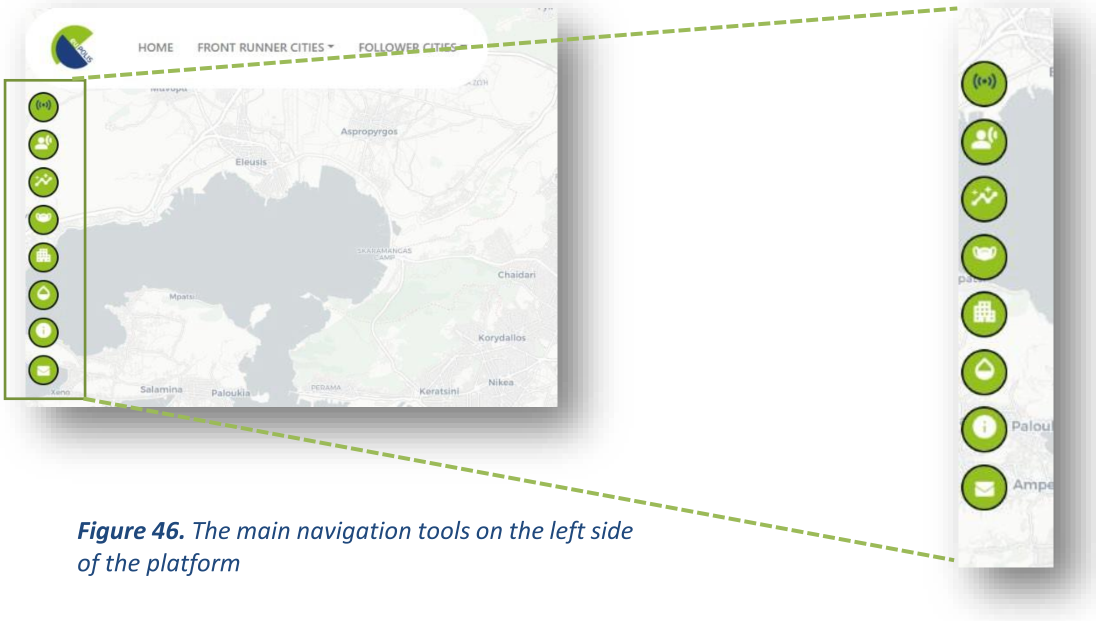
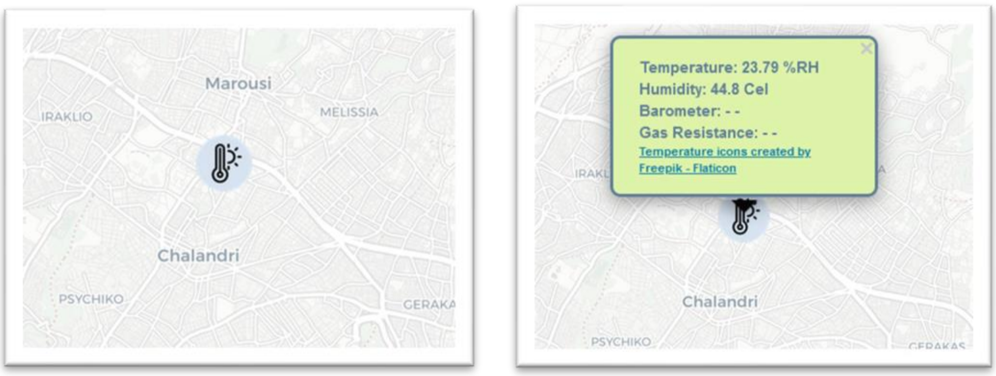


Figure 46. *The main navigation tools on the left side of the platform*

More specifically, the tools shown in **Figure 46** are described as follows (from up to bottom):

- **1st Navigation tool: Sensors**



Temperature: 23.79 %RH
Humidity: 44.8 Cel
Barometer: - -
Gas Resistance: - -
Temperature icons created by Freepik - Flaticon

Figure 47. *1st Navigation tool:* ***(a)*** *Left image- Symbol of sensors on the map, and* ***(b)*** *Right image – Pop-up window with details about the selected sensor*

The 1st Navigation Tool is the sensors (**Figure 47**). By selecting this button all available sensors appear on the map. The user to check the details of the sensor has to first select the symbol illustrated in **Figure 47 (a)** and read in the pop-up window (**Figure 47 (b)**) certain attributes regarding the selected sensor, such as temperature, humidity, barometer, and gas resistance for example. These data are obtained from RISA's API and are refreshed daily. It is noted that at the time of writing this report there are not enough sensors available as they have not been installed in the Demonstration Sites. The available ones are already existing stations in a very wide area of the Demo Sites. Once the weather and air pollution sensors are installed, corresponding updates will be done in the eVM.

- **2nd Navigation Tool: Wearables**

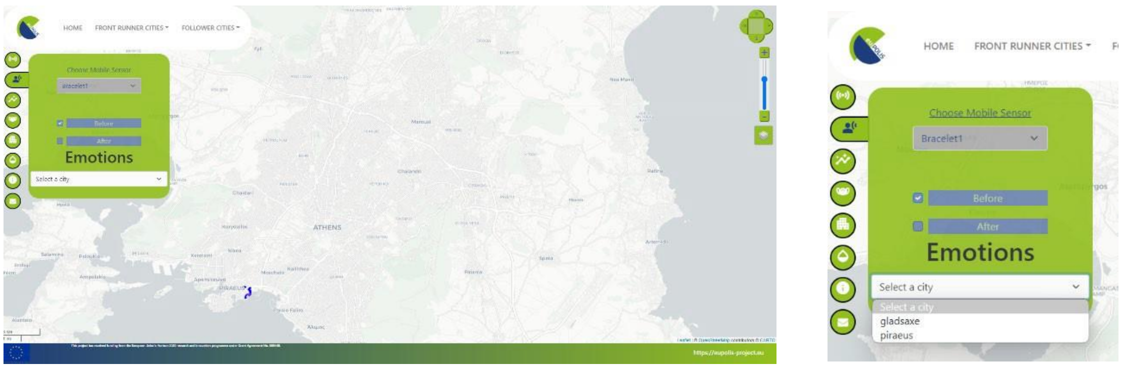


***Figure 48.** 2nd Navigation tool: **(a)** Left image- Wearables Information, and **(b)** Right image – Selection of the desired city*

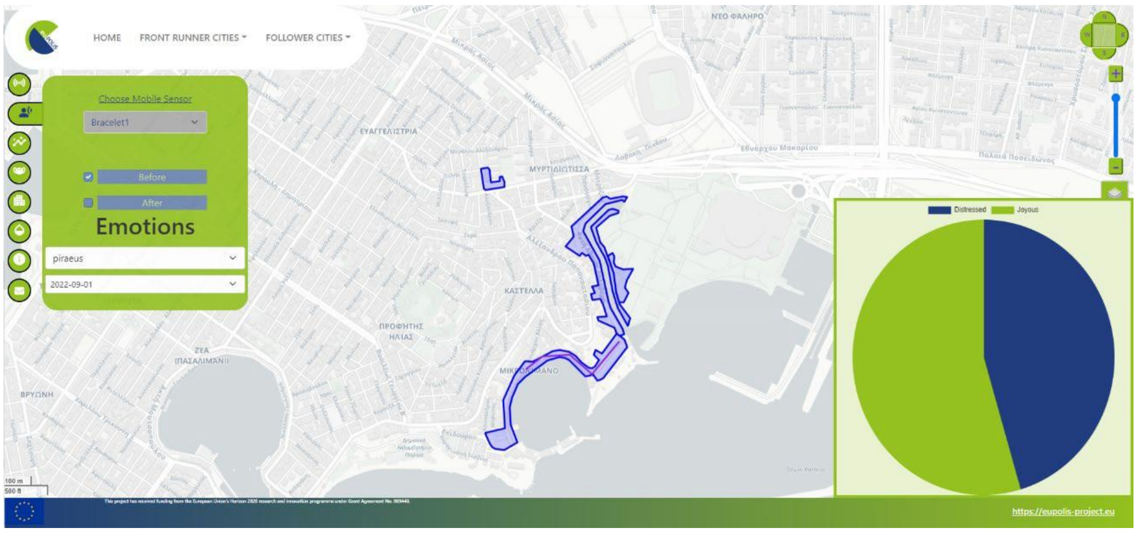


***Figure 49.** 2nd Navigation tool: Visualization of the emotions acquired from the wearables in pie charts **1st step**: Selection of City, **2nd step**: Selection of the dates*

The 2nd Navigation Tool is the wearables (**Figure 48**), in which the end user can receive information on emotions detected in the Demonstration Sites. It is highlighted that personal data are fully protected through this tool as no information is published regarding the user of the wearable and it is also aggregated data. Once the user of the eVM selects the desired city and a specific date from the lists, a pie chart appears on the right side of the platform. This tool aims to give a sense of the change that each intervention has brought to the place. In **Figure 49**, the pie chart includes information regarding some preliminary emotions only for the demonstration site of Piraeus. These data are obtained from RISA's API.

- **3rd Navigation Tool: Weather Plots**

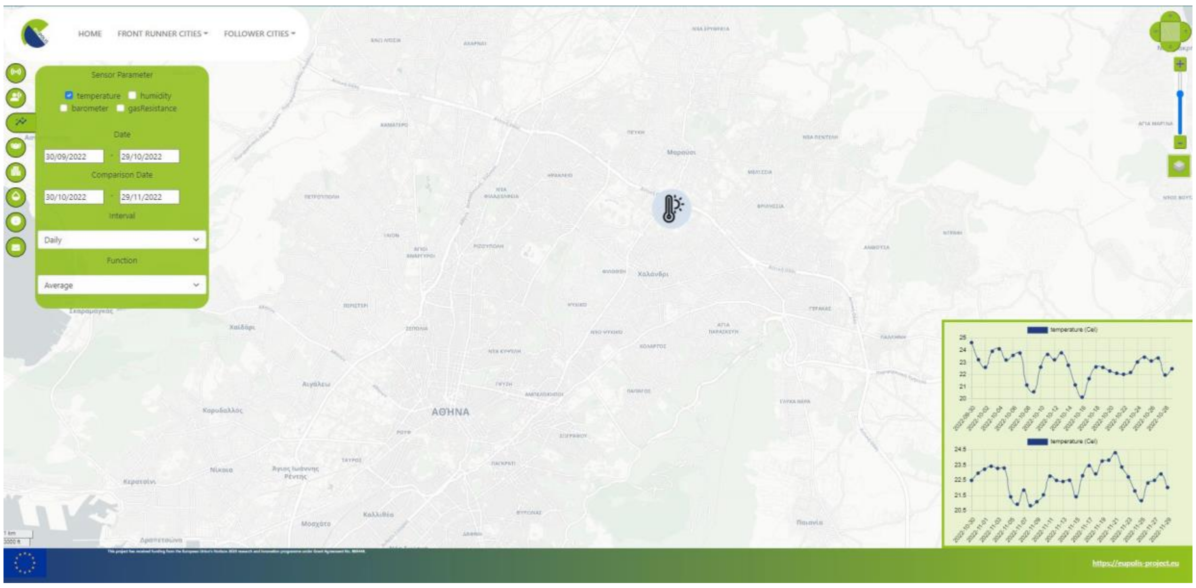


Figure 50. *3rd Navigation tool: Weather Plots. On the left side, the table of attributes of the sensor parameters is visible, on the right side the diagrams illustrating these parameters are demonstrated*

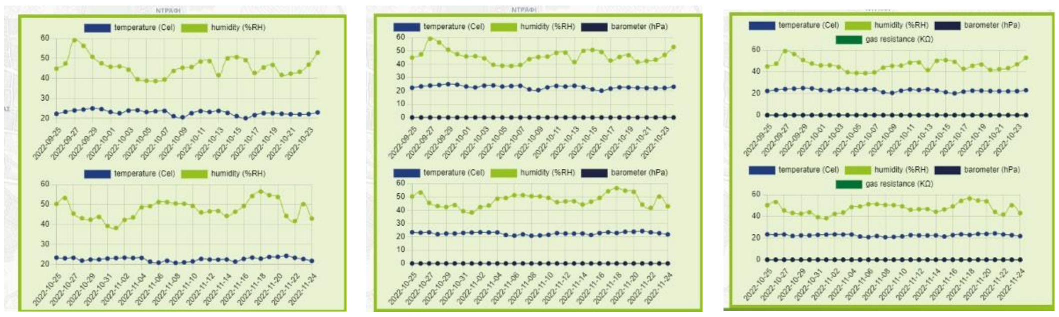


Figure 51. *Weather Plots Parameters. From left to right:* ***(a)*** *Temperature & Humidity,* ***(b)*** *Temperature, Humidity, and Barometer, and* ***(c)*** *Temperature, Humidity, Barometer & Gas Resistance*

The 3rd Navigation Tool is the weather plots (**Figure 50**). This tool offers the end user the ability to monitor weather parameters in the Demonstration Sites. First, the user needs to select a sensor on the map and once the sensor parameters are added, which are comprised of a specific date and a comparison date, an interval (e.g., daily, weekly, monthly, yearly), and a function (e.g., minimum, maximum, average) a diagram appears in the right side of the platform illustrating the result (**Figure 50 & 51**). The parameters of the sensors are Temperature, Humidity, Barometer, and Gas resistance, and can all be projected either separately or in combinations, based on the desired outcome.

- **4th Navigation Tool: Pollution Plots**

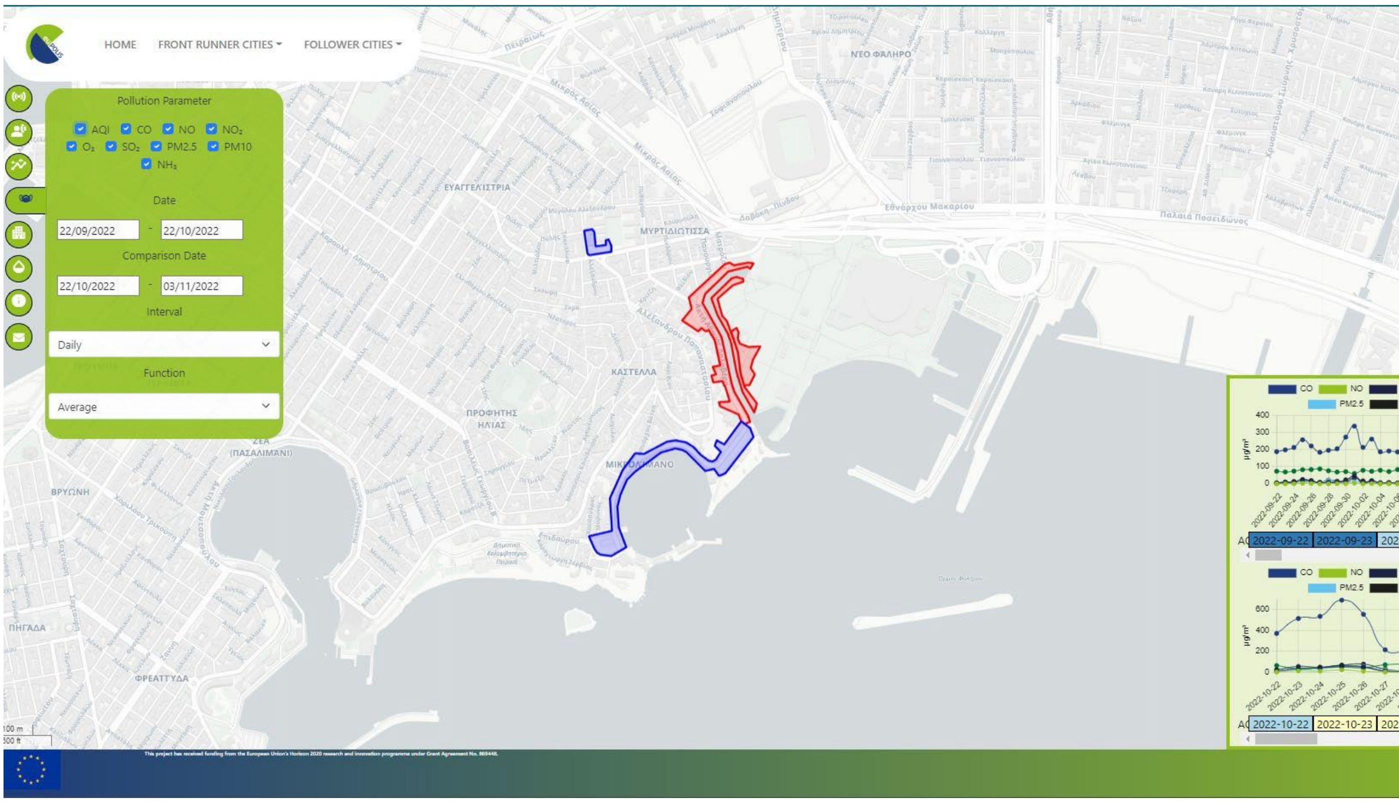


Figure 52. 4th Navigation tool: Pollution Plots. On the left side, the table of attributes of the Pollution parameters is visible, on the right side the diagrams illustrating these parameters are demonstrated

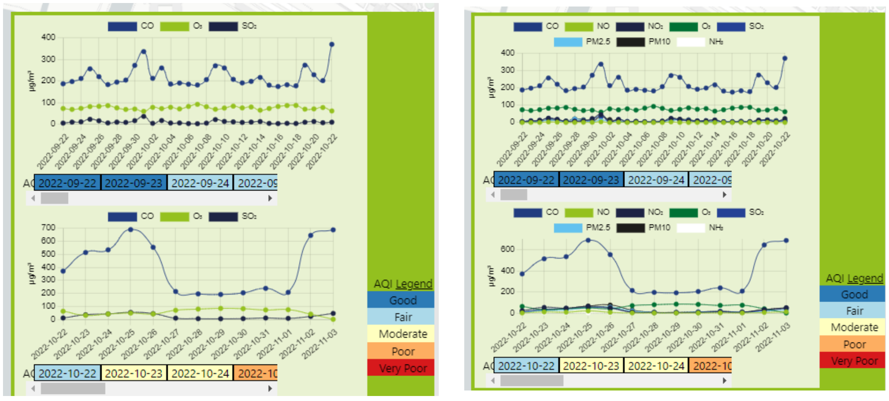


Figure 53. *Pollution Plots Parameters. From left to right some indicative plots based on the selected attributes:* ***(a)*** *CO, O_2 & SO_2* ***(b)*** *CO, NO, NO2, O_2, SO_2, $PM_{2.5}$, PM_{10} & NH_3*

The 4th Navigation Tool is the pollution plots (**Figure 52**). This tool is similar to the previous, weather plots, but the difference is that air pollution parameters are being visualized in the plot.

First, the user needs to select a marked area on the map, and then, once pollution parameters are applied, which are comprised of a specific date and a comparison date, an interval (e.g., daily, weekly, monthly, yearly) and a function (e.g., minimum, maximum, average) the corresponding plot appears in the right side of the platform illustrating the result (**Figure 52 & 53**).

The current pollution parameters are AQI (Air Quality Index), CO (Carbon Monoxide), NO (Nitric Oxide), NO_2 (Nitrogen Dioxide), O_2 (Oxygen), SO_2 (Sulfur Dioxide), $PM_{2.5}$ (a complex mixture of solids and aerosols), PM_{10} (Particulate matter suspended in air or water), and NH_3 (Ammonia). These data are obtained from the API of the OpenWeatherMap, which gives access to current weather data for any location, processed and collected from different sources such as global and local weather models, satellites, radars, and a vast network of weather stations. It should be noted that once the new stations will be installed in each Demo Site, eVM will be updated to utilize the data coming from these stations.

- **5th Navigation Tool: 3D Model**

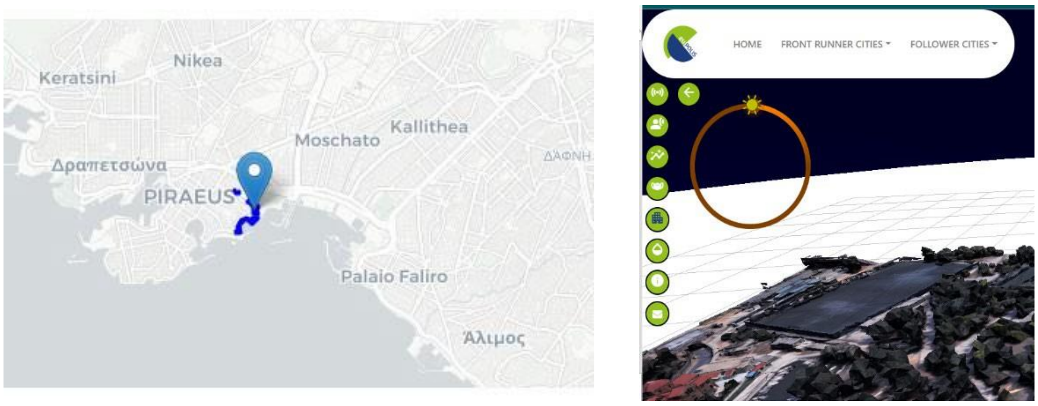


Figure 54. *5th Navigation tool: 3D Model.* ***(a)*** *On the left image is the market,* ***(b)*** *On the right image the sun simulation ring*

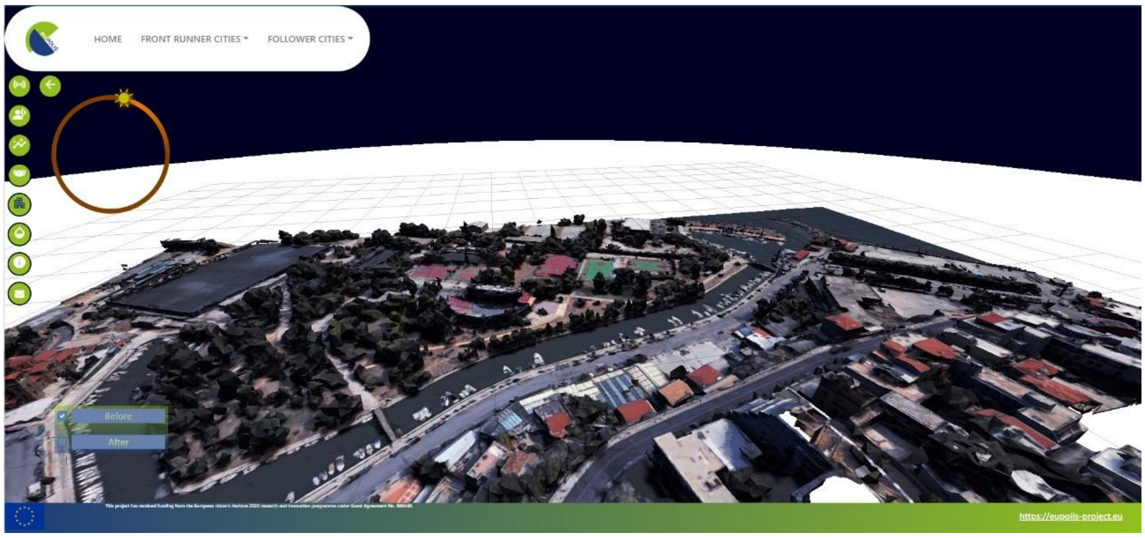


Figure 55. *5th Navigation tool: 3D Model in an FBX format*

The 5th Navigation Tool is the 3D Model. Through this, the end user can visualize in 3D the designed interventions for the Demonstration Sites. First, the user needs to select a marker on the map as indicated in **Figure 54 (a)**, and then the 3D Models appear illustrating the representation of the area, as demonstrated in **Figure 55. In Figure 54 (b),** there is a sun simulation ring, enabling it to change from night to day, and receive an illustration throughout the day.

- **6th Navigation Tool: Metabolism-based NBS Planning & Simulation**

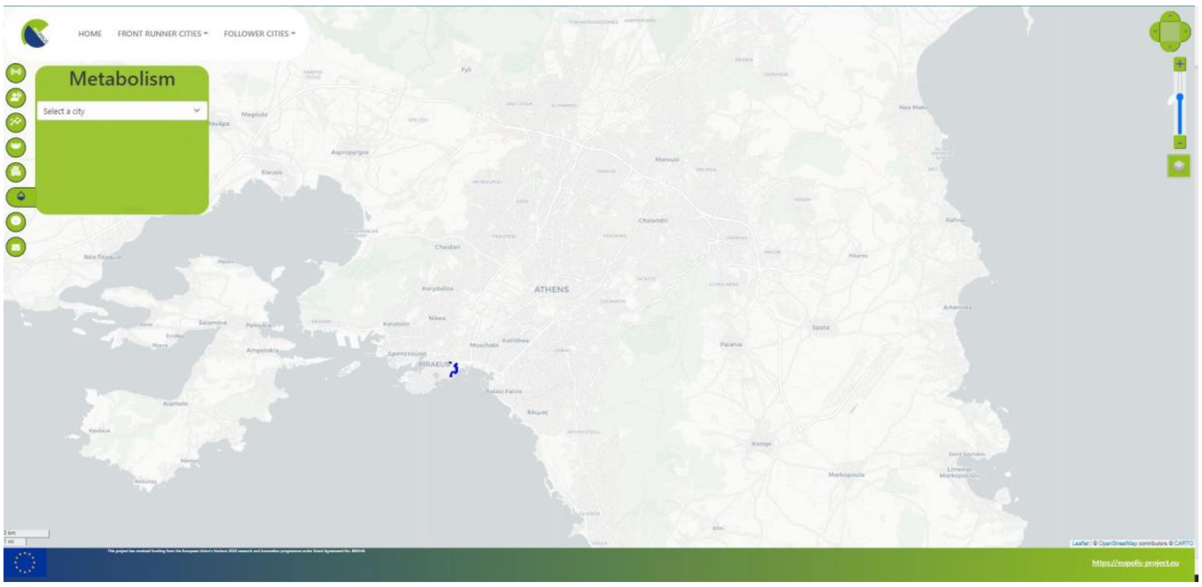
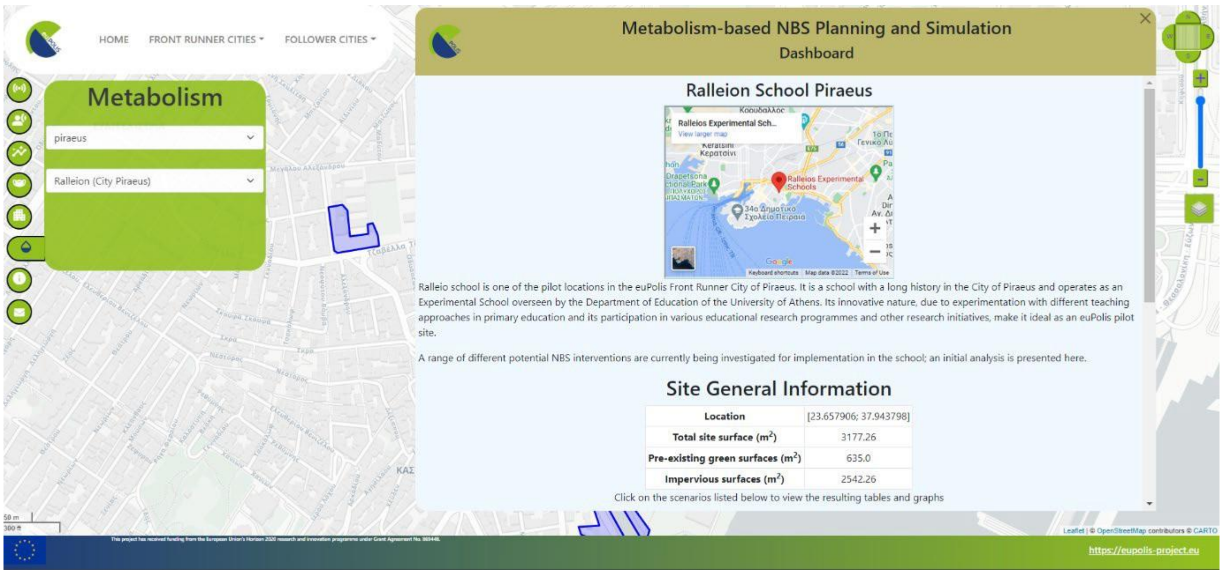


***Figure 56**. 6th Navigation tool: Metabolism-based NBS Planning & Simulation*



Metabolism-based NBS Planning and Simulation
Dashboard

Ralleion School Piraeus

Ralleio school is one of the pilot locations in the euPolis Front Runner City of Piraeus. It is a school with a long history in the City of Piraeus and operates as an Experimental School overseen by the Department of Education of the University of Athens. Its innovative nature, due to experimentation with different teaching approaches in primary education and its participation in various educational research programmes and other research initiatives, make it ideal as an euPolis pilot site.

A range of different potential NBS interventions are currently being investigated for implementation in the school; an initial analysis is presented here.

Site General Information

Location	[23.657906; 37.943798]
Total site surface (m^2)	3177.26
Pre-existing green surfaces (m^2)	635.0
Impervious surfaces (m^2)	2542.26

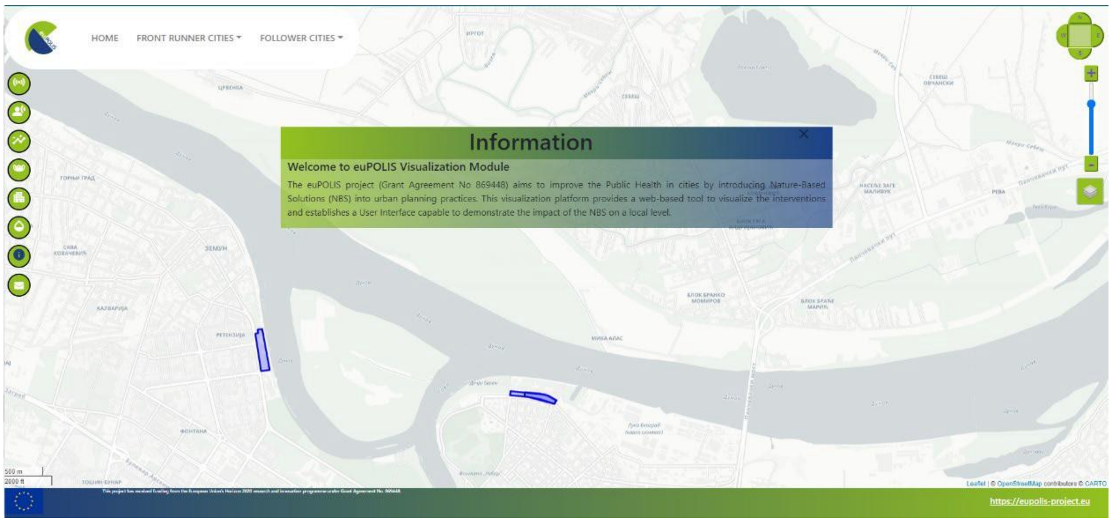
Click on the scenarios listed below to view the resulting tables and graphs

***Figure 57.** Information about the different scenarios in the selected demonstration site*

The 6th Navigation Tool is the Metabolism-based NBA Planning and Simulation. This tool provides the results of the modeling that took place under the Task The user needs to select the desired city and the intervention area (e.g., Ralleion School of Piraeus), as indicated in **Figure 56**, and then the dashboard will appear, as demonstrated in **Figure 57.** More information regarding the "*Metabolism-based NBA Planning and Simulation"*, are explained in deliverable 5.6.

- **7th Navigation Tool: Project Information & Send Feedback**

The last Navigation tool is the Project Information button explaining the euPOLIS project, followed by the send feedback option, by which the user can provide valuable information regarding the platform and help with its improvement (**Figure 58**).



Information

Welcome to euPOLIS Visualization Module

The euPOLIS project (Grant Agreement No 869448) aims to improve the Public Health in cities by introducing Nature-Based Solutions (NBS) into urban planning practices. This visualization platform provides a web-based tool to visualize the interventions and establishes a User Interface capable to demonstrate the impact of the NBS on a local level.

***Figure 58**. 7th Navigation tool: Project Information & Send feedback option*

4.4 euPOLIS platform Development Cycle

To sum up, in performing the analysis tasks, eVM is similar to the client/server typical three-tier architecture. The geo processing is breaking down into server-side and client-side tasks. A client typically is a Web browser. The server-side consists of a Web Server and an eVM software (**Figure 12, 13**).

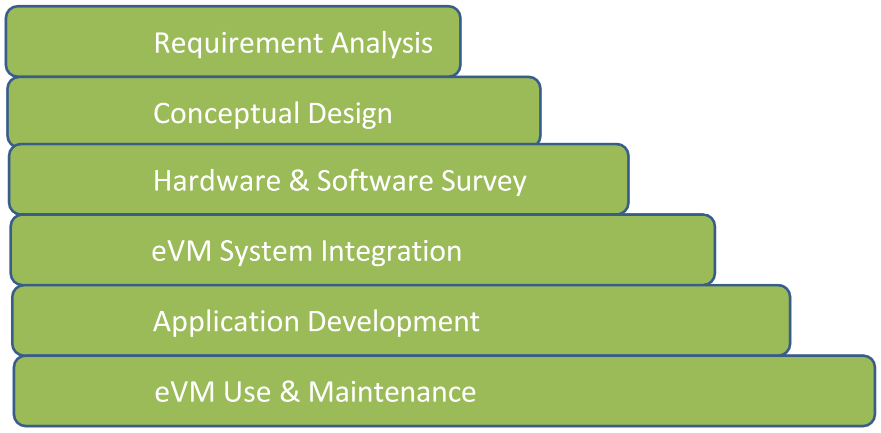


Figure 59. *Development Cycle*

- **Requirement Analysis**
  **see section 2 and Annex 1 for their detailed verification*
- **Hardware & Software Survey**
  Here are some of the hardware and software components that may be required for developing a web-based visualization module:
  Prior to the local hosting of the platform, the PC used for the development has the following characteristics (**Table 8**):

Table 8. *Device Specifications*

Category	Device Specifications
Processor	Intel®Core™i7-4720HQ CPU@2.60GHz
Installed RAM	16.0 GB
System Type	64-bit operating system, x64-based processor

Hardware

- ➢ Server: A server is typically required to host the platform and handle requests from users. This can be a physical server or a virtual server hosted in the cloud. In euPOLIS case, the server used is, for now, in a **local** one
- ➢ Storage: The platform will likely require storage to store data, such as maps, spatial data and metadata. This can be a **local storage system** or a cloud-based storage solution. In euPOLIS case, for example the 3D models are saved locally, data like the ones from DMS or RISA are kept in the local Cass. After one month those data are being deleted automatically

Software

- Operating System: The server will need an operating system to run the WebGIS platform. In this case **Ubuntu 22.04 LTS OS** was used
- Web Server Software: A web server, in this case **NGINX** (**see section 3.3**), was used to host the platform and handle requests from users. Inside this nginx docker container gets the source code Copy the source code into the nginx docker container and then run the following commands in order to deploy the eVM:

```
cd eupolis

#frontend installation
npm install -legacy-peer-deps
npm run build

#backend installation
python3 -m venv env
source env/bin/activate
pip install -r requirements.txt

#for production
uwsgi -ini proj.ini

#for development
flask -debug run
```

- Database Management System: A database management system (DMS) is typically needed to store and manage the data for the WebGIS platform. In euPOLIS case there is **not a local database**. The required data are requested with the use of an API through RISA's DMS
- Programming Language: A programming language, such as Python, Java, or PHP, is typically used to develop the platform and its associated features and functionality. In eVM **Python was used for the Backend**, and **JavaScript, CSS** and **HTML for the Frontend**

Selecting suitable software is an important step in a successful implementation. Software was evaluated on functionality and performance, and independent of the hardware and operating system. With respect to the required functions and cost the software that was selected is **Visual Studio Code** in order to build the backend. Furthermore, it requires specific hardware configuration. Since the volume of transferred data is big, the speed of Internet connection is vitally important. For example, when a request of monthly data is made on the backend while clicking on a sensor to produce the data plots speed has been identified to be approximately 1ms. Based on the experience gained, a **i7 4720 HQ processor** computer with **16 GB RAM** has been selected for this project.

- **eVM Use & Maintenance**

The final step in the implementation of this module is to put the system to use. The eVM presented in this report is a demo of the platform, and after some discussions and with system integration and testing being finalized all applications could be available for end-users. Furthermore, the system maintenance (database, software, hardware) is an essential part, to be considered during and after the project’s lifetime.

5 Conclusion & future work

In conclusion, the aim of the euPOLIS Visualisation Module can be summarized in two main elements; the first is providing the end users with a powerful ICT tool for visualizing the designed and implemented urban planning interventions and the second is monitoring the environmental and societal impact. The first objective is fulfilled through the support of visualizing 3D models. The other objective is achieved through the following features: environmental monitoring by creating weather and air pollution plots and societal monitoring by visualizing aggregated emotions. A significant element in the above is the fact that the dimension of time is taken into consideration. In more detail, the user can define the time period for which he/she wishes to receive this information (e.g., weather plots for a specific Demonstration Site) enabling this way a comparison of pre- and post-intervention situations.

Taking into consideration that the monitoring phase of the project as well as the implementation of the designed interventions have not yet been completed in all study areas, it is expected that after the final submission of this report, additions will need to be implemented to the euPOLIS visualization module. These additions include indicatively establishing new connections through APIs with the euPOLIS DMS to visualize data coming from the newly installed in-situ sensors (weather and air pollution stations) or incorporating some new 3D models of the designed interventions. In this context, it is implied the probability of claiming effort in WP5 despite it will have typically completed. It should be highlighted that at its current stage, all appropriate tools and technologies for any further development needed are already incorporated. The design, implementation, and functionalities of the eVM have been thoroughly presented in this deliverable and future additions will not alter the core of its implementation.

6 References

https://blog.hubspot.com/website/react-js

Annex 1: Verification of Requirements

***Table 9**: Verification of Mandatory (M) Requirements.*

Title	Description of Verification	Priority[6]	Status
UR- 15 Intervention impact visualization	eVM offers pre- and post-environmental monitoring (weather and air pollution parameters) by enabling the user to select the specific timeframe for receiving these kinds of data. Additionally, enables users to receive information on the detected emotions in the DS.	**M**	✓
UR- 16 NBS evaluation	eVM offers pre- and post-environmental monitoring (weather and air pollution parameters) by enabling the user to select the specific timeframe for receiving these kinds of data. Additionally, enables users to receive information on the detected emotions in the DS.	**M**	✓
UR- 17 Data protection	Anonymization/protection of the recorded data in the euPOLIS project. The data storage complies with GDPR. No cookies are stored. The eVM does not store any data – it visualizes data from the DMS and from the local server (3D models which are not personal data). Any data related to wearable devices (e.g., detected emotions) are presented in an aggregated way.	**M**	✓
UR- 18 Device and Data security	The security framework is end-to-end for data (from source to output visualization). Data requests between the backend and the DMS are done securely via HTTPS. Also, the talisman python library was used to specify the content security policy.	**M**	✓
UR- 19 System extensibility	eVM's design allows further extensions such as connection with new sensors, wearables, and larger system deployments. Any source of related data added to the DMS is designed to be automatically integrated into the eVM.	**M**	✓
UR- 20 Local language support	The GSH team has created a document with all eVM fields and texts in English. It is distributed to local partners to be translated. Once completed, GSH will integrate the other languages into the eVM.	**M**	**In progress**
UR- 21 Data homogenization	All spatial data are projected in the same CRS (Coordinate Reference System) which is the WGS 84 / Pseudo-Mercator (EPSG 3857). This is the default CRS to be used in Leaflet. All 3D data are in FBX format. Environmental data are in JSON format.	**M**	✓

[6] M – Mandatory, I – Important, D - Desirable

Title	Description of Verification	Priority[6]	Status
FR- 8 Provide a web service to visualize the interventions	eVM provides visualization of the interventions through embedding the 3D models of the designed and implemented interventions in the DS. This functionality is supported by three.js.		✓
FR- 9 Provide various analytics, which can help quantify the impact of an applied NBS	eVM offers pre- and post-environmental monitoring (weather and air pollution parameters) by enabling the user to select the specific timeframe for receiving these kinds of data. Additionally, enables users to receive information on the detected emotions in the DS.		✓
FR- 10 Variability of user access levels to ensure the security of potentially sensitive data	eVM doesn't include the visualization of sensitive data. Any data related to wearable devices (e.g., detected emotions) are presented in an aggregated way.	**M**	✓
FR- 11 Ability to store different data objects, such as text files, photos, maps, etc	It visualizes different data either from the euPOLIS DMS or from the local server (3D models). The different data objects include weather and air pollution parameters (json), and 3D models (FBX). Additionally, pdf files and images (.png, .jpeg) can be visualized.	**M**	✓
NFR- 20 Adaptability	GSH used the "React" library. This library enables the development of applications by creating reusable components that can be considered independent blocks. These components are individual pieces of a final interface, which, when assembled, form the application's entire UI.	**M**	✓
NFR- 21 Availability	In case of a failed or bad request to the backend, an alert appears and the rest of the application continues to operate normally. This was achieved by making use of the "try catch" block, which in case of an exception an alert window appears.	**M**	✓
NFR- 22 Compliance	eVM doesn't include the visualization of sensitive data. Any data related to wearable devices (e.g., detected emotions) are presented in an aggregated way.	**M**	✓
NFR- 23 Configuration management	eVM will remain in a local server throughout the project's lifetime and also during the exploitation procedure. After the end of the project, further discussion among partners will take place, and its maintenance/host will be decided.	**M**	✓
NFR- 24 Confidentiality	eVM doesn't include the visualization of sensitive data. Any data related to wearable devices (e.g., detected emotions) are presented in an aggregated way. In this context, eVM does not have to address confidentiality issues.	**M**	✓
NFR- 25 Extensibility	eVM's design allows further extensions such as connection with new sensors, wearables, and larger	**M**	✓

Title	Description of Verification	Priority[6]	Status
	system deployments. Any source of related data added to the DMS is designed to be automatically integrated into the eVM.		
NFR- 26 Fault tolerance	In case of a failed or bad request to the backend, an alert appears and the rest of the application continues to operate normally. This was achieved by making use of the "try catch" block, which in case of an exception an alert window appears.	**M**	✓

***Table 10**: Verification of Important (I) Requirements.*

Title	Description of Verification	Priority[7]	Status
UR- 22 Well-being indicators accessibility	The eVM provides access to aggregated emotional data (**see Figure XX**) which can be interpreted as well-being indicator measurements.	I	✓
UR- 23 Availability to people with disabilities	According to the euPOLIS site, similar accessibility tools will be implemented.	I	In progress
UR- 24 Application support	Technical details for each step are shown in a pop-up window when the user selects one of the tools. In addition to this, a user manual is available through the eVM in the information tab.	I	✓
UR- 25 Feedback provision	Direct feedback through email (**see Figure 32**) has been incorporated into the eVM.	I	✓
FR- 12 Establish a UI capable to demonstrate the impact of the NBS on a local level	eVM offers pre- and post-environmental monitoring (weather and air pollution parameters) by enabling the user to select the specific timeframe for receiving these kinds of data. Additionally, enables users to receive information on the detected emotions in the DS.	I	✓
FR- 13 Interconnection with several software and programming languages	The development of the eVM relies on several software and programming languages to fulfill all project requirements. • Python (back end) • JavaScript (front end)	I	✓
FR- 14 Support various types of spatial data format including vector, raster, KMZ, Shapefiles, GeoJSON	eVM supports various types of spatial data both raster and vector through OGC services (WMS, WFS) and other data formats such as GeoJSON, shapefiles, and KMZ.	I	✓
NFR- 27 Interoperability	eVM establishes a connection between frontend - backend via the "axios" library and between backend-DMS via API requests.	I	✓
NFR- 28 Maintainability	GSH used the "React" library. This library enables the development of applications by creating	I	✓

[7] M – Mandatory, I – Important, D - Desirable

Title	Description of Verification	Priority[7]	Status
NFR- 29 Scalability	The backend is making use of the "Uwsgi" library which provides an adequate number of workers (processes) to support concurrent users	I	✓
NFR- 30 Supportability	eVM supports contact through email. In more detail, eVM provides a corresponding tool, the 7th Navigation Tool: Project Information & Send Feedback (**see Figure 32**).	I	✓
NFR- 31 Simple security measures	eVM's code is backed up in the local GitLab server regularly. In case of an update, corresponding actions are taken to ensure the best possible maintenance of the eVM.	I	✓
NFR- 32 Seamless exchange of data information among systems	eVM supports the seamless exchange of data information among systems through HTTPS requests.	I	✓

***Table 11**: Verification of Desirable (D) Requirements.*

Title	Description of Verification	Priority[8]	Status
UR- 26 euPOLIS outputs visualization	eVM offers pre- and post-environmental monitoring (weather and air pollution parameters) by enabling the user to select the specific timeframe for receiving these kinds of data. Additionally, enables users to receive information on the detected emotions in the DS.	D	✓
UR- 27 Raw data accessibility	eVM provides access to raw data coming from weather and air pollution stations using API requests (see Figure XX)	D	✓
UR- 28 Data provision	eVM gives the end users the ability to perform various types of spatial and attribute queries. In more detail, end users can visualize specific parameters (e.g., PM2.5, temperature) for specific DS and specific dates.	D	✓
NFR- 33 Capacity	The user can display on the map all the available sensors simultaneously.	D	✓

[8] M – Mandatory, I – Important, D - Desirable

Annex 2: User Manual

EuPOLIS Visualization Module MANUAL

The euPOLIS project has received funding from the European Union's Horizon 2020 Research & Innovation programme H2020-EU.3.5.2., under **grant agreement No 869448**

https://www.facebook.com/eupolis2020

https://twitter.com/eu_polis

https://www.linkedin.com/company/eupolis

https://eupolis-project.eu

EuPOLIS' approach connects Nature-Based Solution interventions for open public spaces with citizens' needs for improved public health and Well-Being

V

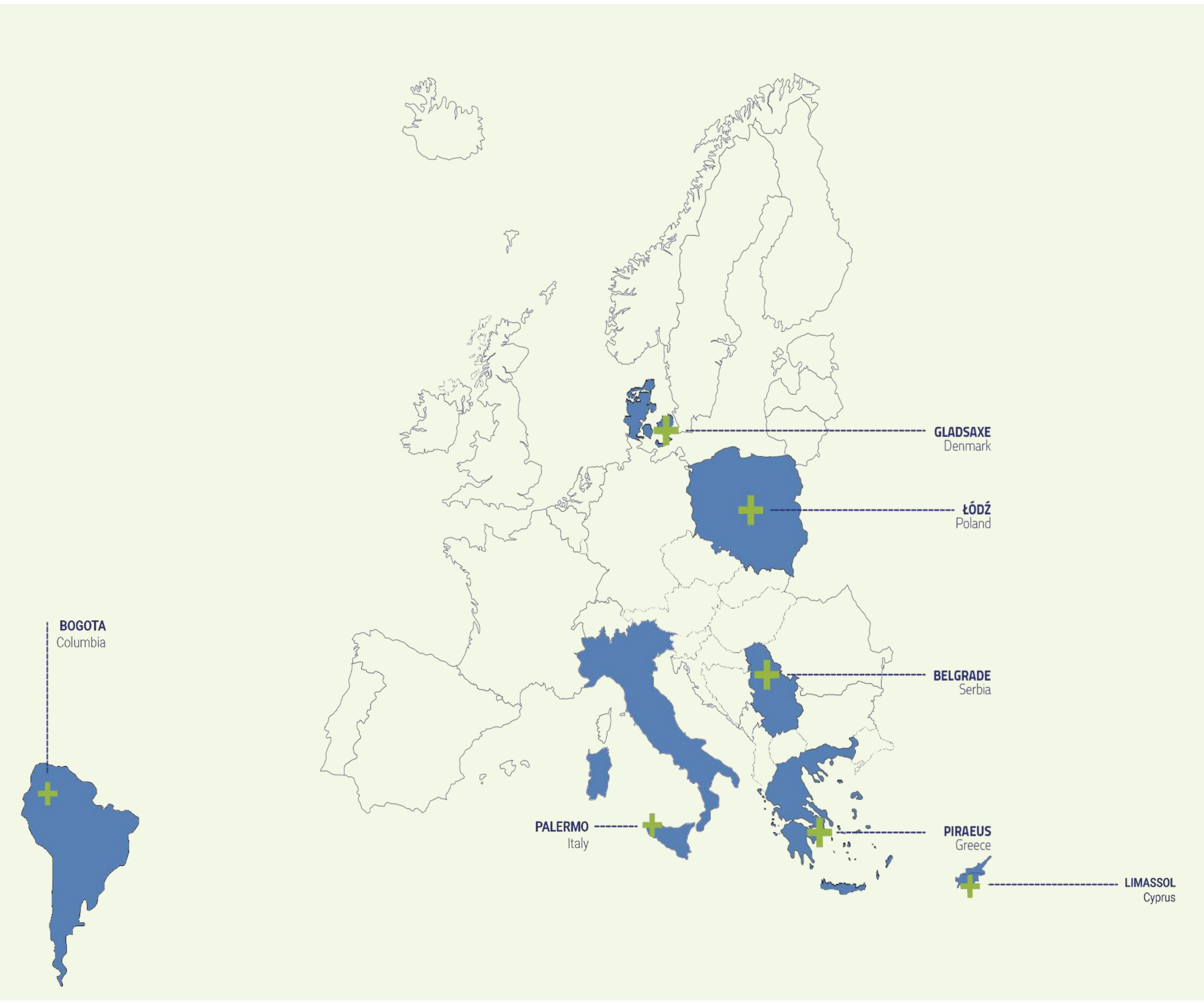

GLADSAXE
Denmark
ŁÓDŹ
Poland
BOGOTA
Columbia
BELGRADE
Serbia
PALERMO
Italy
PIRAEUS
Greece
LIMASSOL
Cyprus

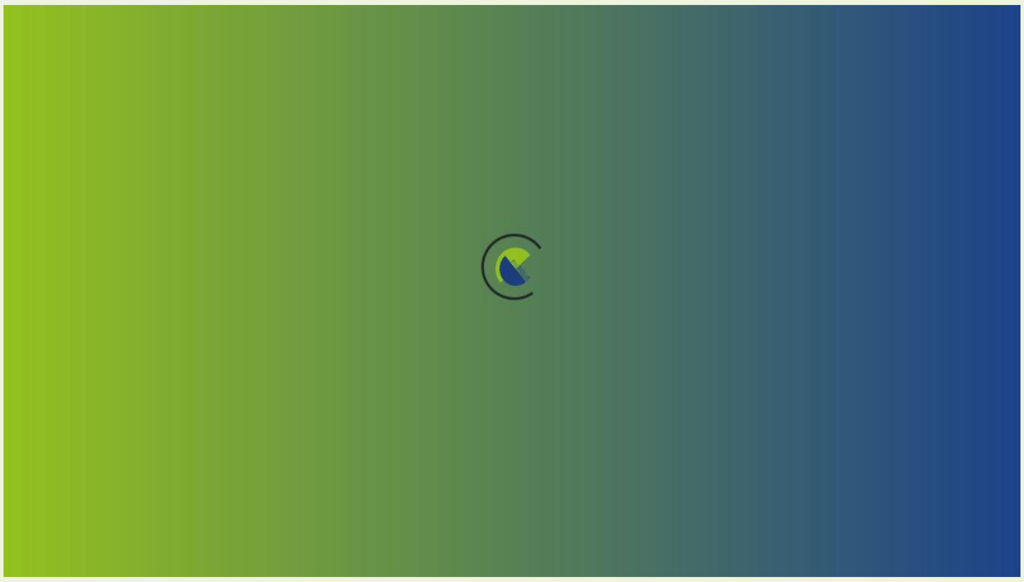
TABLE OF CONTENTS

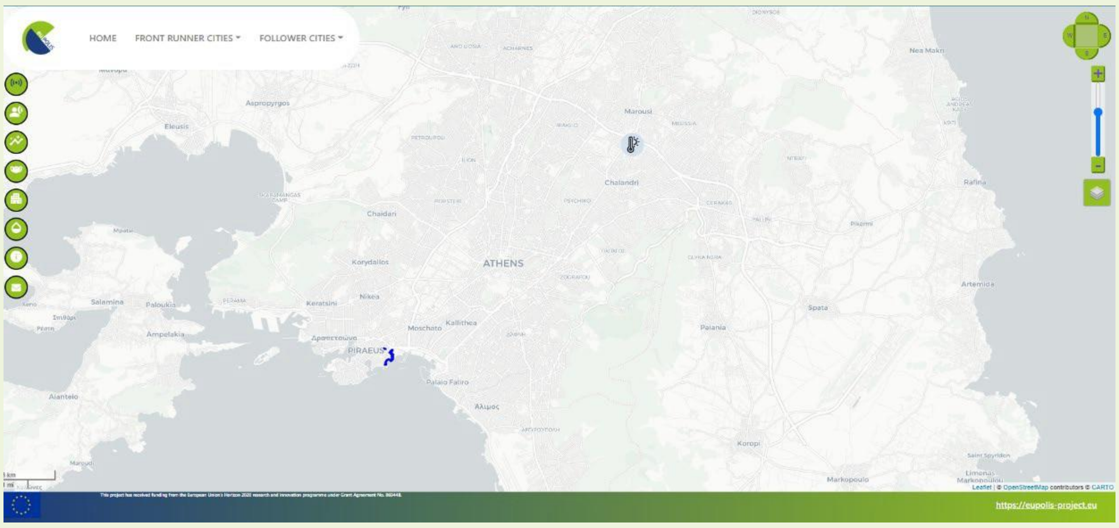
01

HOW TO USE THE PLATFORM

Once you enter the euPOLIS Platform:

- ❖ The loading screen appears
- ❖ The main interface of the Platform is accessible





02

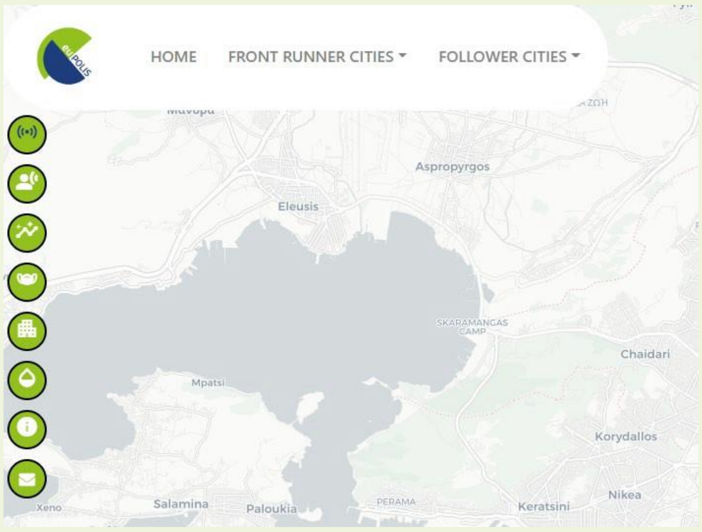
GENERAL FEATURES OF euPOLIS PLATFORM

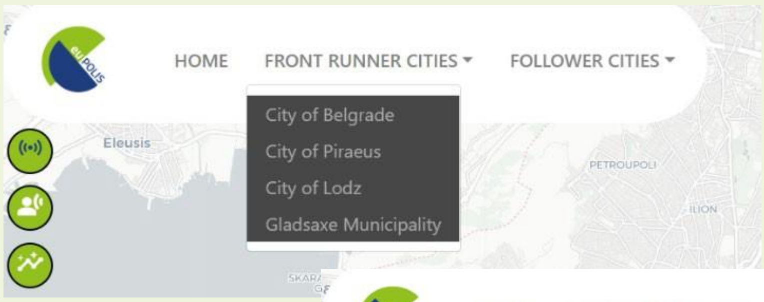
MAIN NAVBAR

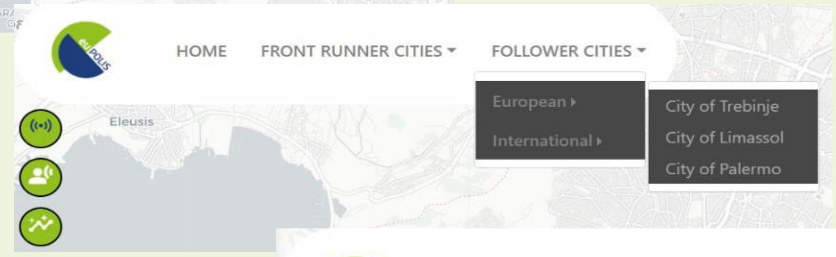
The euPOLIS platform can be described by the following features:

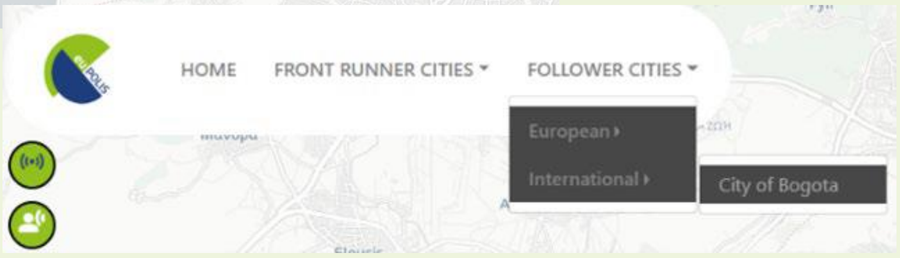
The Main NavBar

- ➢ Home
- ➢ Front Runner Cities
- ➢ Follower Cities
    - ❖ European Cities
    - ❖ International Cities





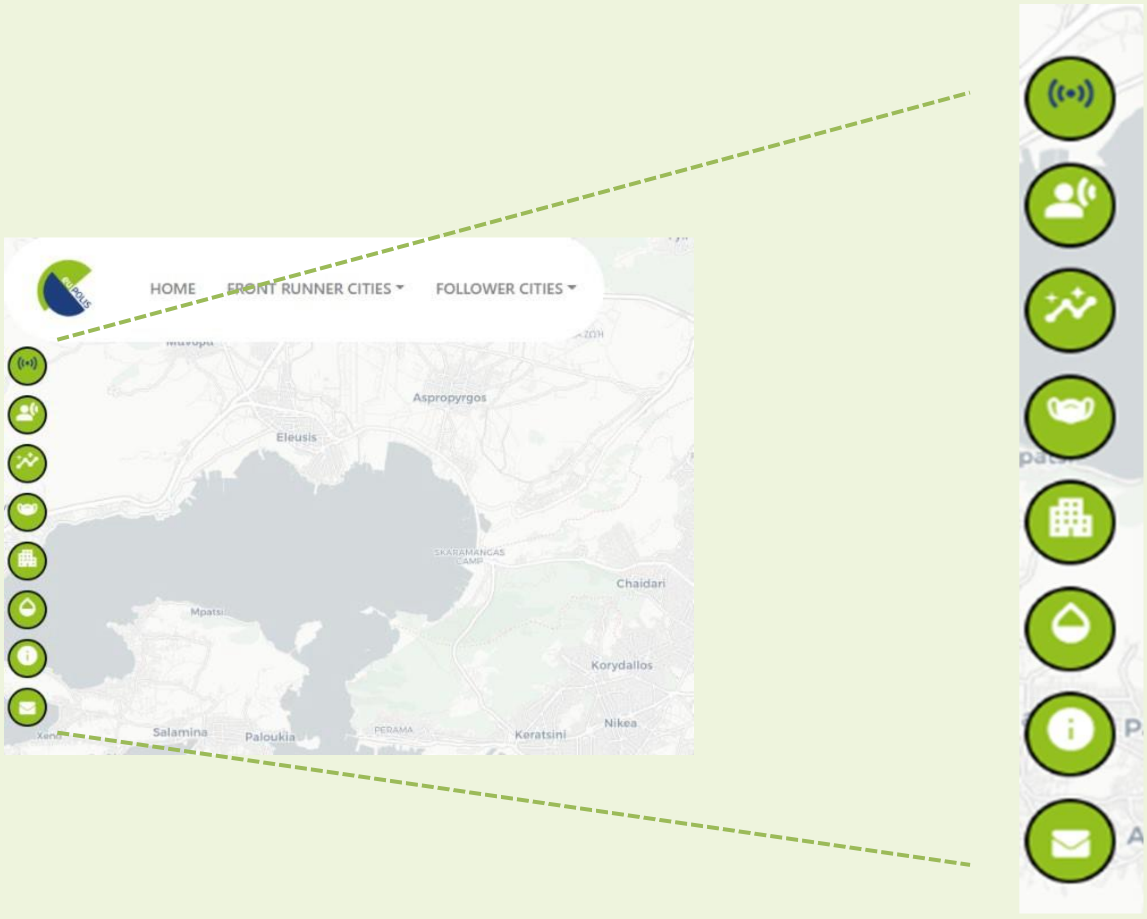




SIDEBAR

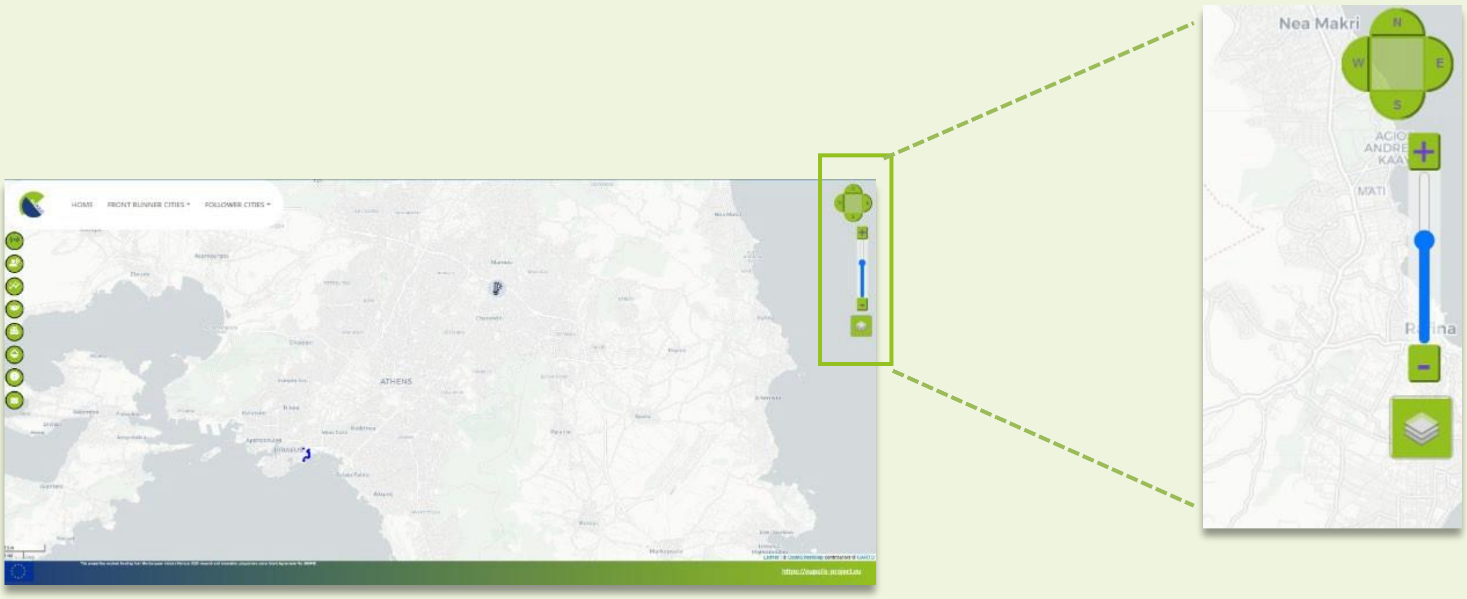
You can easily navigate within the euPOLIS Visualisation Module.

To achieve easier and quicker navigation within the eVM, a variety of tools/sub-menus have been created. The main tools are available in the platform's interface and can be found on the left part of the platform.



MAP TOOLS

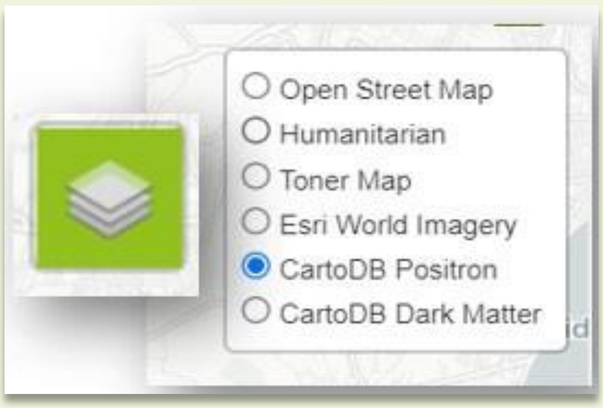
You can access to the Map Tools, via the right part of the platform:



❖ Pan tool (N, E, S, W)

❖ Zoom-In, Zoom-out

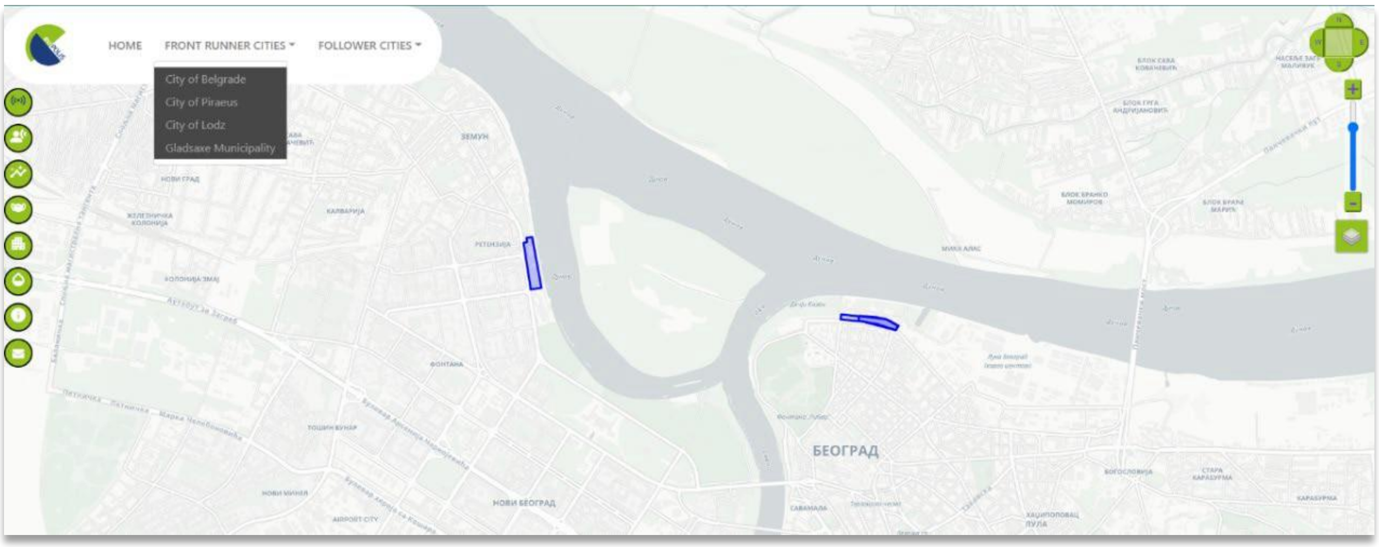
❖ Layers selection/Background (Layer switcher provided by Leaflet)



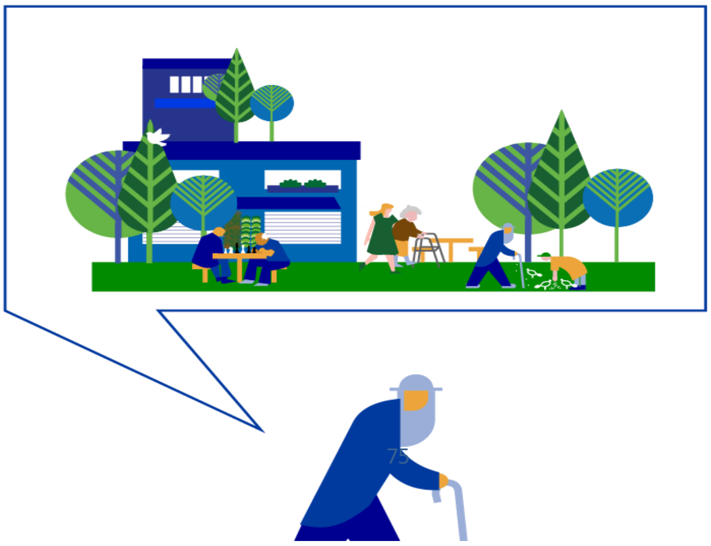
- ○ Open Street Map
- ○ Humanitarian
- ○ Toner Map
- ○ Esri World Imagery
- [✓ ● CartoDB Positron]
- ○ CartoDB Dark Matter

City of Belgrade

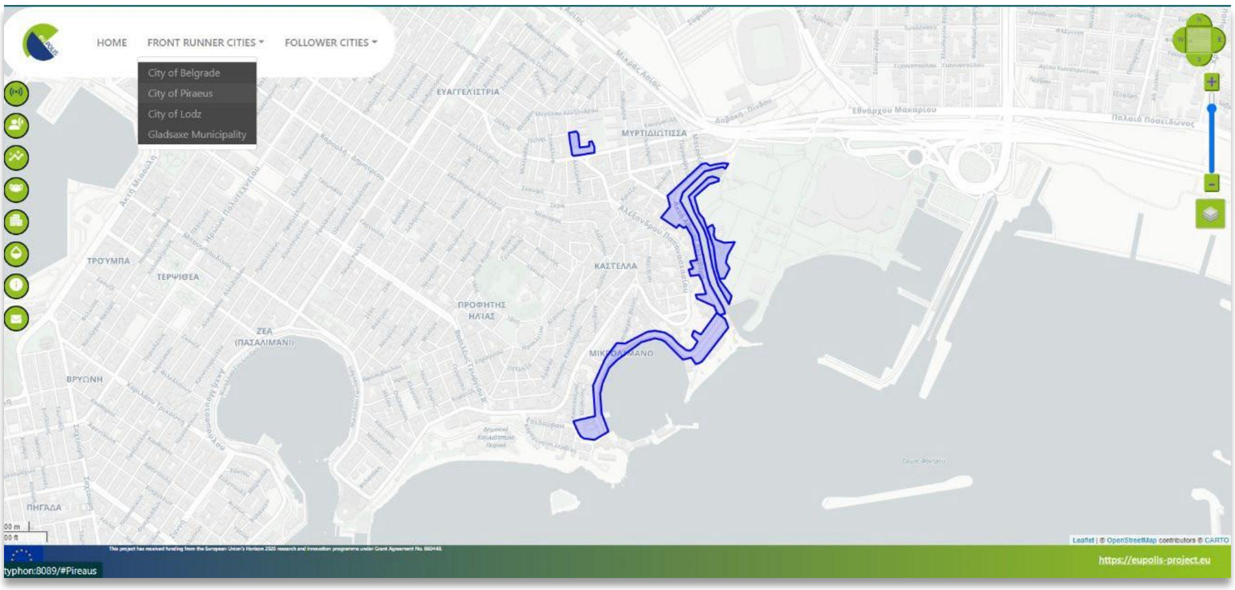
You can visualize the polygon of the City of Belgrade, as indicated by the blue lines the two intervention sites are available.



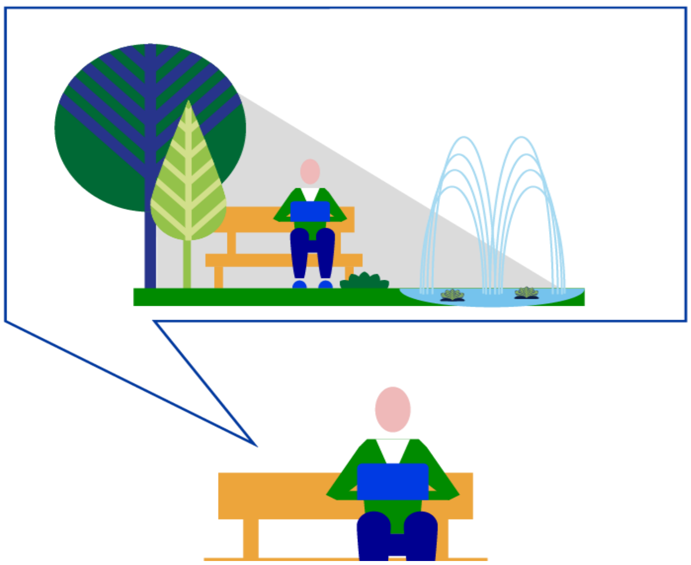
The City of Belgrade is comprised of two intervention sites, one called "Usce Park" which can be found in the new Belgrade, and the other one called "Linear Park" on the old side of Belgrade. The polygons indicating the intervention areas. Once the user selects the intervention site, a brief description of this area can be found.



You can visualize the polygon of the City of Piraeus, as indicated by the blue lines the three intervention sites are available.

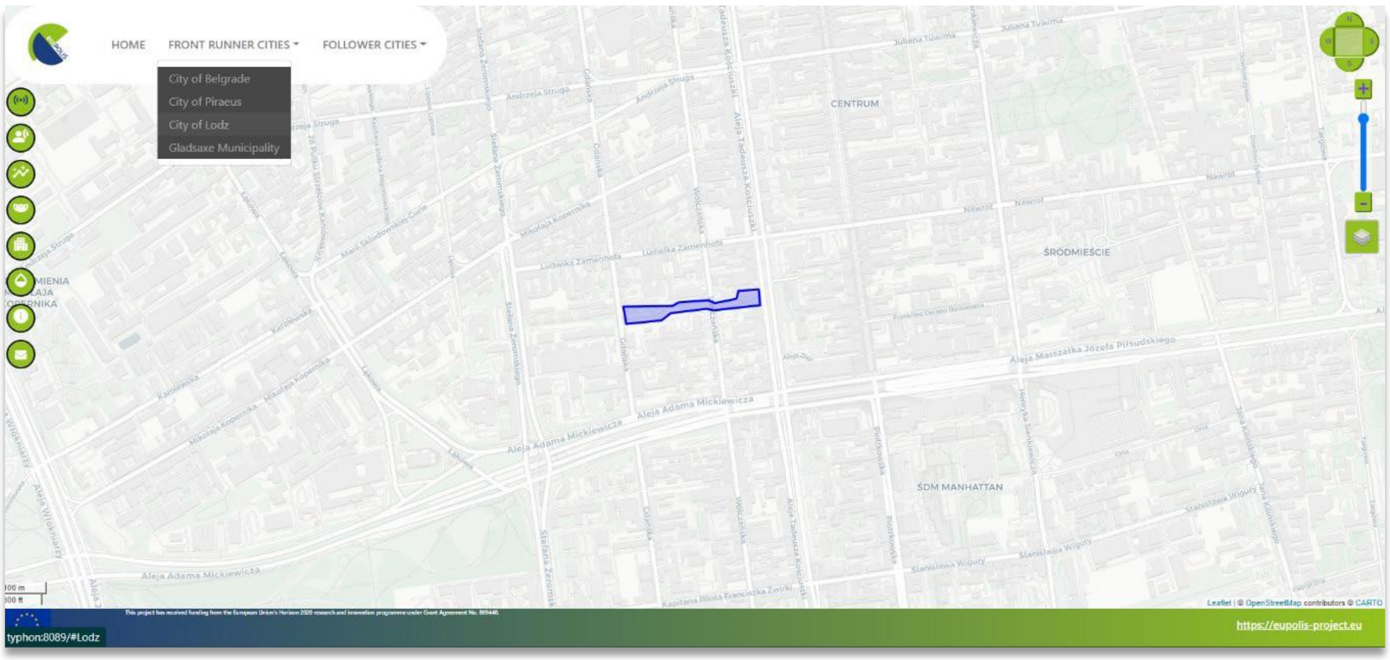


The City of Piraeus is comprised of three intervention sites, called "Ralleion", "Microlimano" and "Akti Dilaveri". The polygons indicate the intervention areas. Once the user selects the intervention site, a brief description of this area can be found.

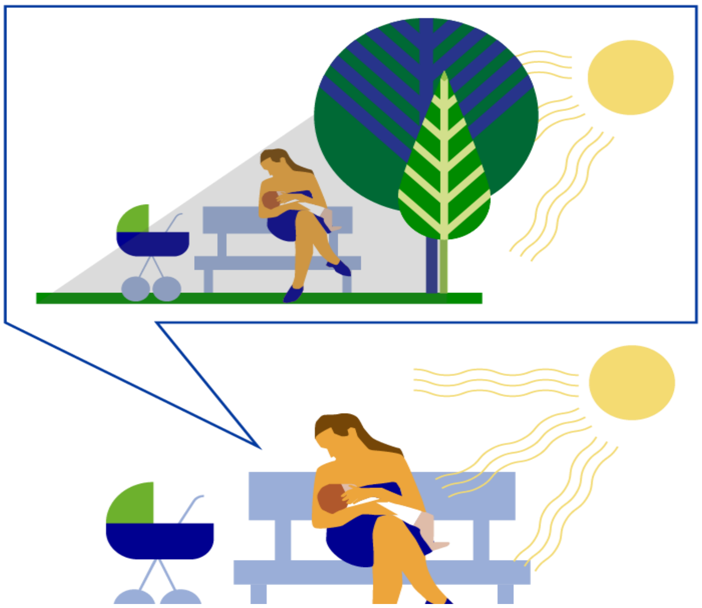


City of Lodz

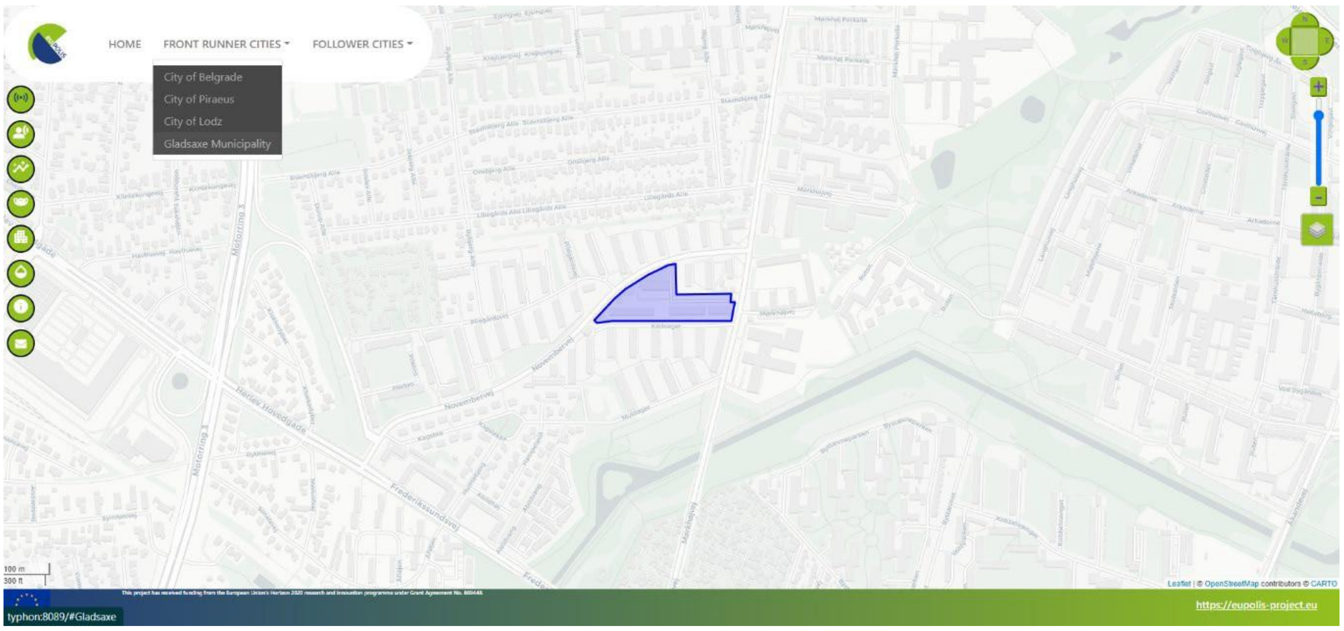
You can visualize the polygon of the City of Lodz, as indicated by the blue lines the one intervention site is available.



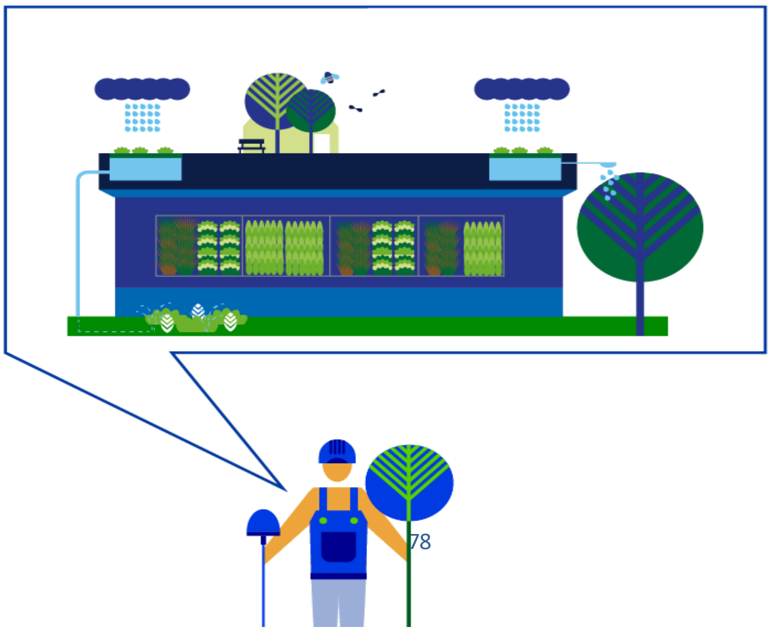
The City of Lodz comprises one intervention site, "Posaz Anny Rynkowskiej". The polygon indicates the intervention area. Once the user selects the intervention site, a brief description of this area can be found



You can visualize the polygon of the Gladsaxe Municipality. As indicated by the blue lines the one intervention site is available.



Gladsaxe Municipality is comprised of one intervention site, called “Pileparken 6”. The polygon indicating the intervention area. Once the user selects the intervention site, a brief description of this area can be found.



03

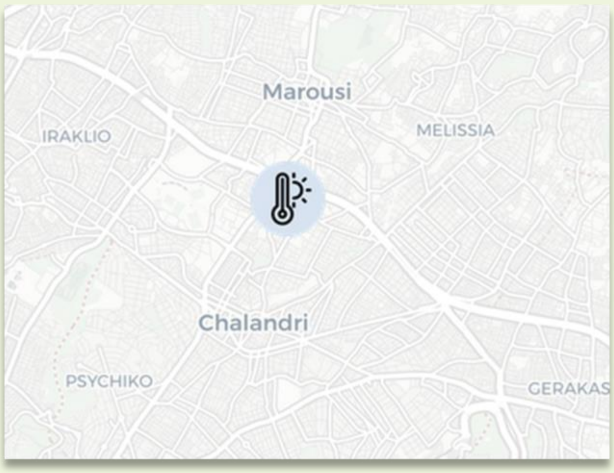
MAIN NAVIGATION TOOLS

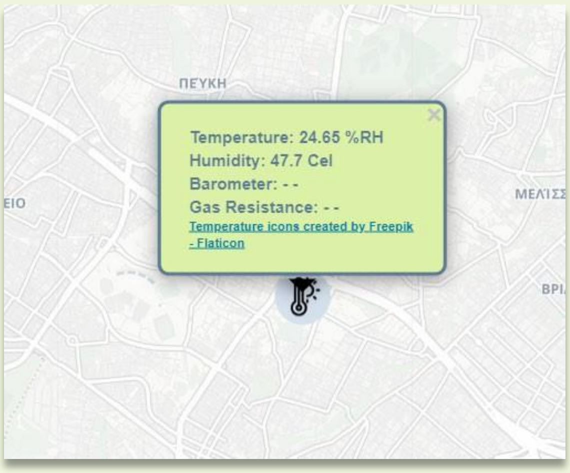
- SENSORS
- WEARABLES
- WEATHER PLOTS
- POLLUTION PLOTS
- 3D MODEL
- METABOLISM-BASED NBS PLANNING & SIMULATION
- PROJECT INFORMATION & FEEDBACK

SENSORS

You can visualize the information related to Sensors by pressing the button

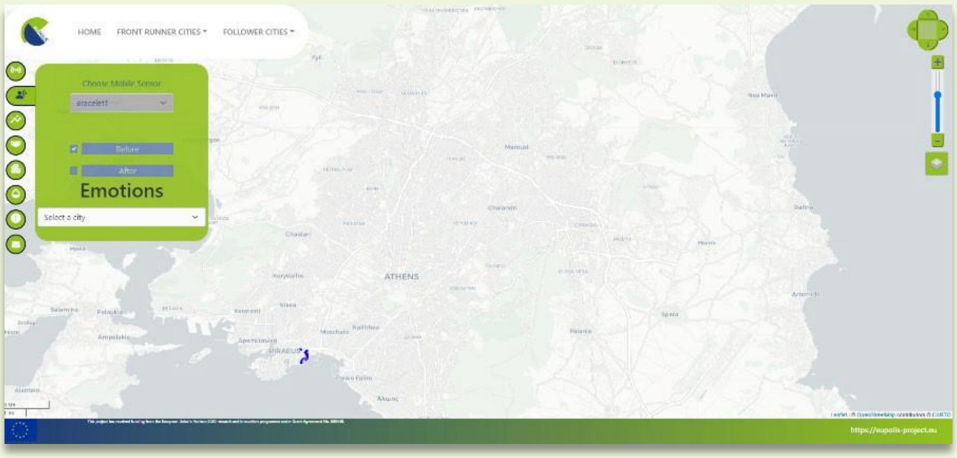
- ❖ You can select the symbol of sensors on the map
- ❖ A Pop-up window with details about the selected sensor appears

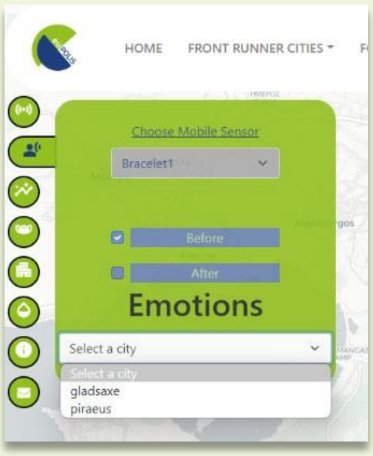



Temperature: 24.65 %RH
Humidity: 47.7 Cel
Barometer: - -
Gas Resistance: - -
Temperature icons created by Freepik - Flaticon

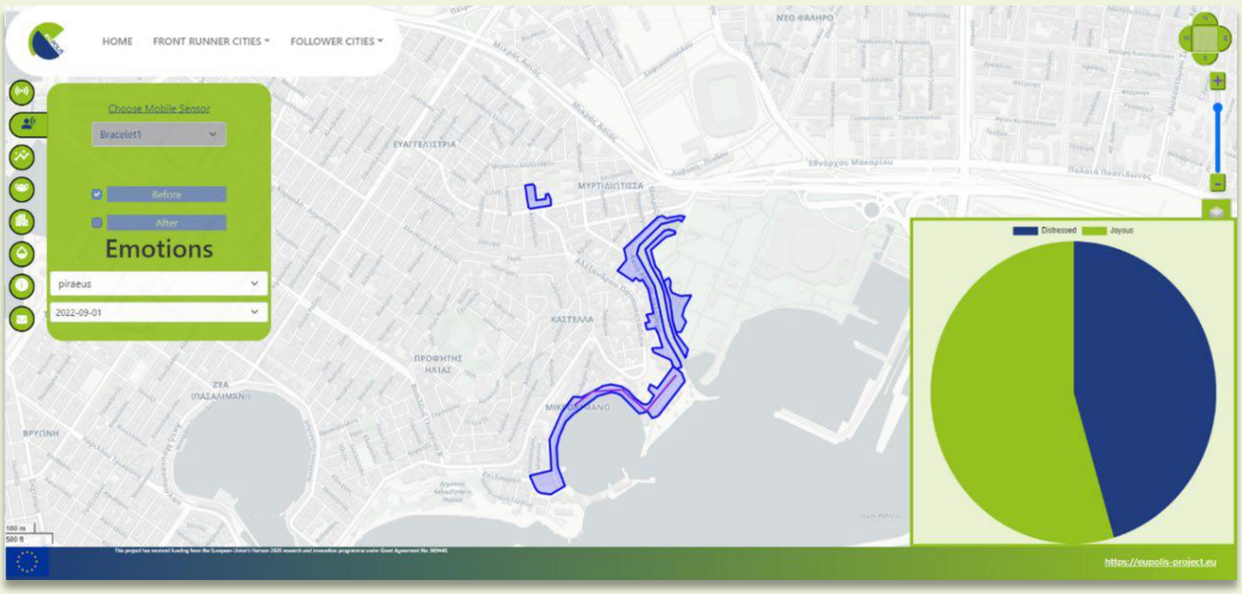
WEARABLES

You can visualize the information related to Wearables by pressing the button



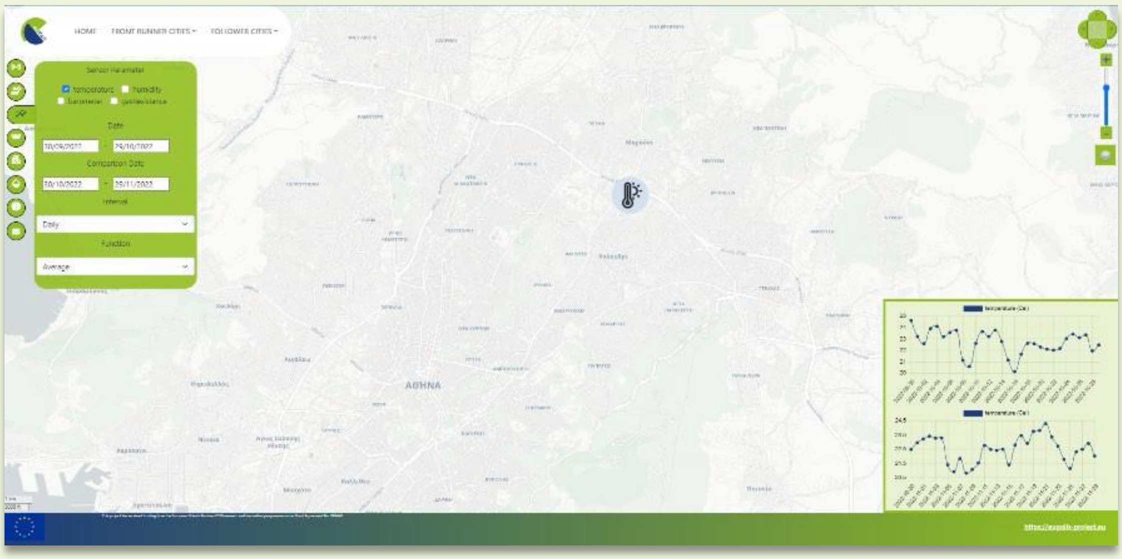


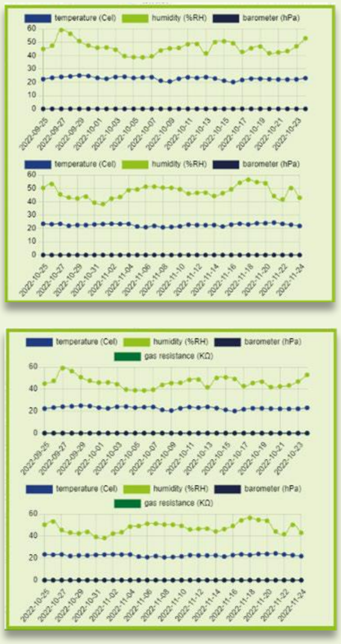
- You can receive information on emotions detected in the Demonstration Sites.
- You can select the desired city and a specific date from the lists, a pie chart appears on the right side of the platform.



WEATHER PLOTS

You can visualize the Weather Plots by pressing the button



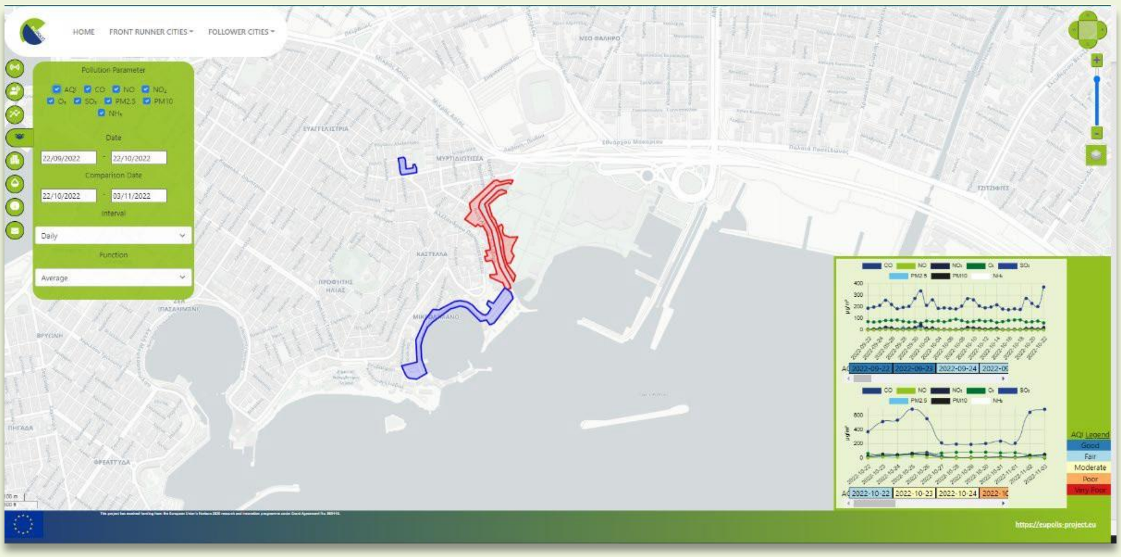


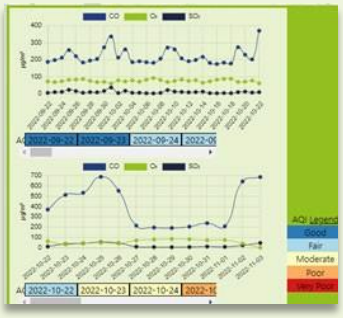
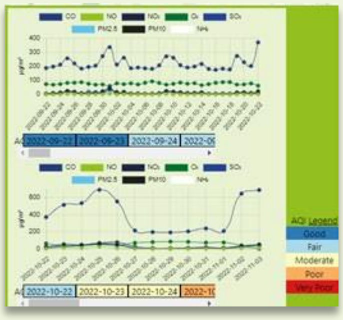
- ❖ Select a sensor on the map
- ❖ Add sensor parameters: date, comparison date, an interval and a function
- ❖ A diagram appears in the right side of the platform illustrating the result.
- ❖ Parameters of the sensors:
  - Temperature
  - Humidity
  - Barometer
  - Gas resistance

** All can be projected either separately or in combinations, based on the desired outcome*

POLLUTION PLOTS

You can visualize the Pollution Plots by pressing the button



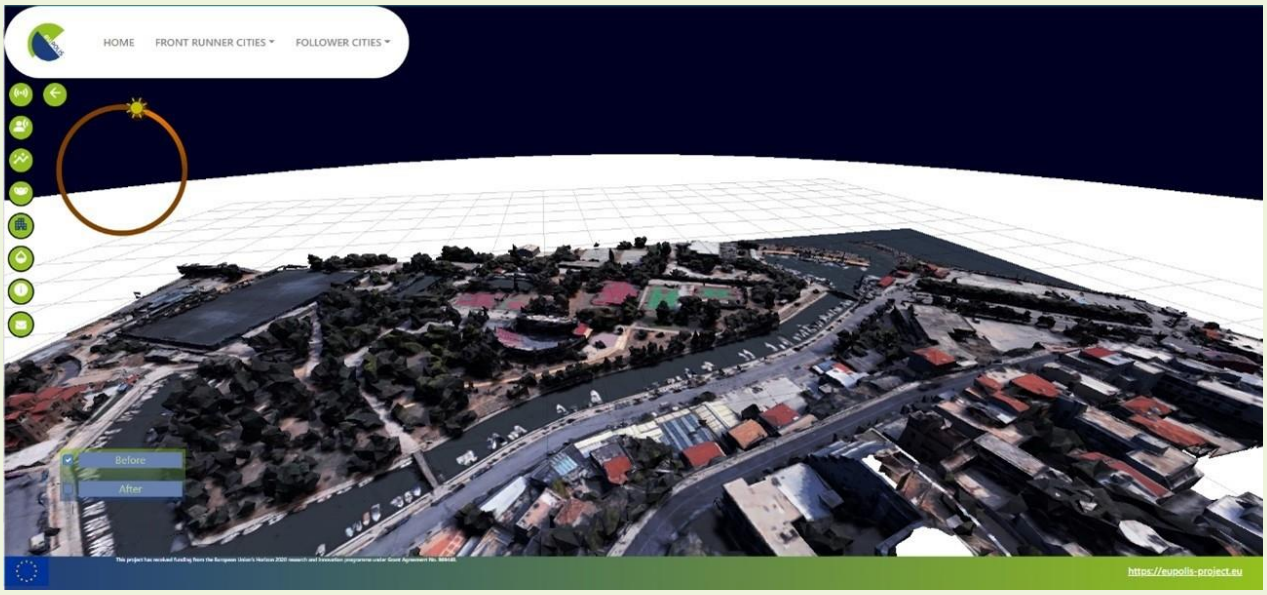



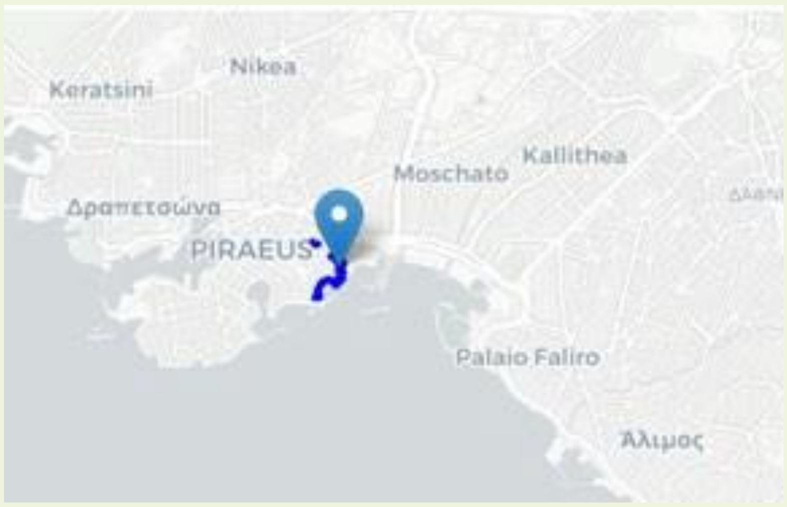
- ❖ Select a marked area on the map
- ❖ Apply pollution parameters: date, a comparison date, an interval and a function
- ❖ Plot appears in the right side of the platform illustrating the result.

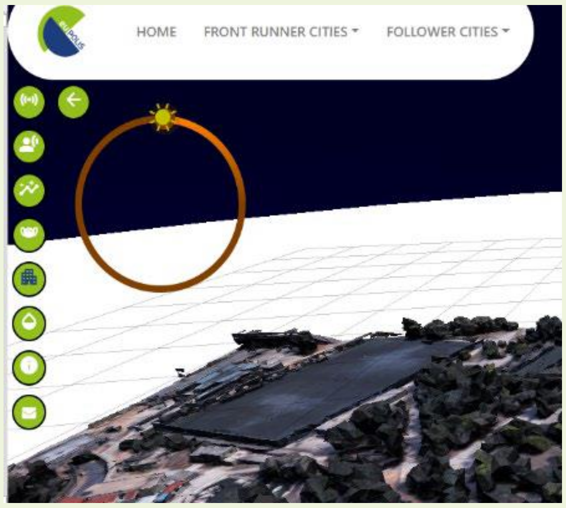
3D MODEL

You can visualize the 3D Models by pressing the button

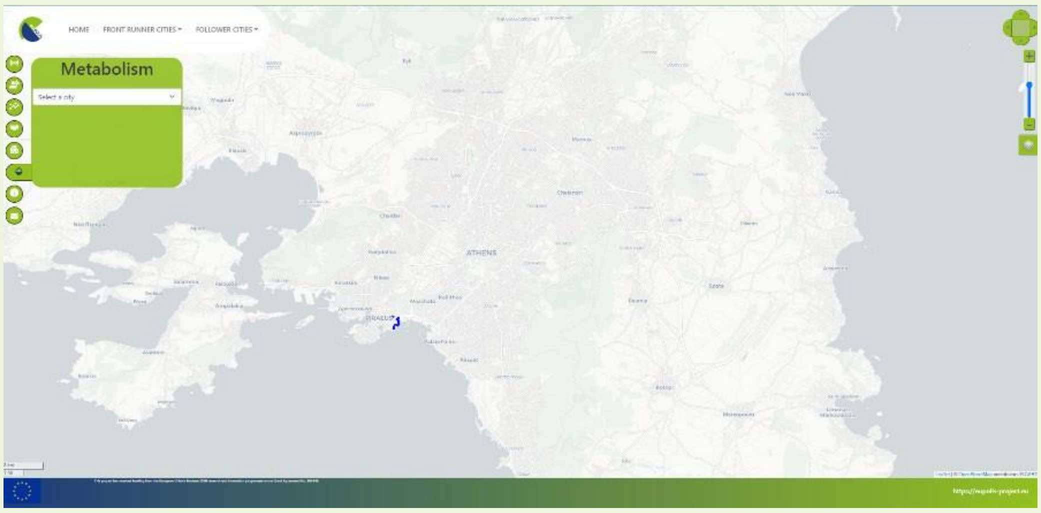
Through this, you can visualize in 3D the designed interventions for the Demonstration Sites. You need to select a marker on the map, and then the 3D Models appear illustrating the representation of the area.

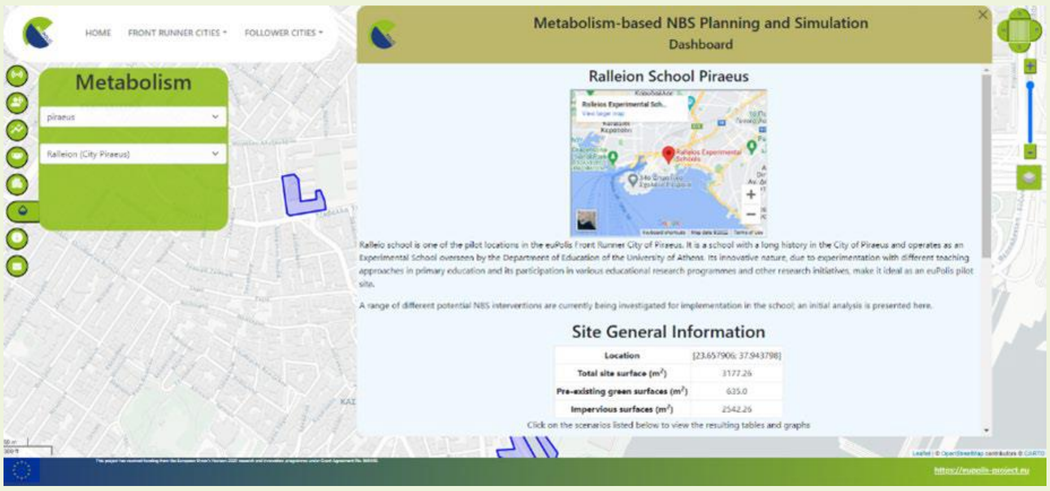






You can read the Metabolism-based NBS planning & Simulation by pressing the button





Metabolism-based NBS Planning and Simulation

Dashboard

Ralleion School Piraeus

Ralleio school is one of the pilot locations in the euPolis Front Runner City of Piraeus. It is a school with a long history in the City of Piraeus and operates as an Experimental School overseen by the Department of Education of the University of Athens. Its innovative nature, due to experimentation with different teaching approaches in primary education and its participation in various educational research programmes and other research initiatives, make it ideal as an euPolis pilot site.

A range of different potential NBS interventions are currently being investigated for implementation in the school; an initial analysis is presented here.

Site General Information

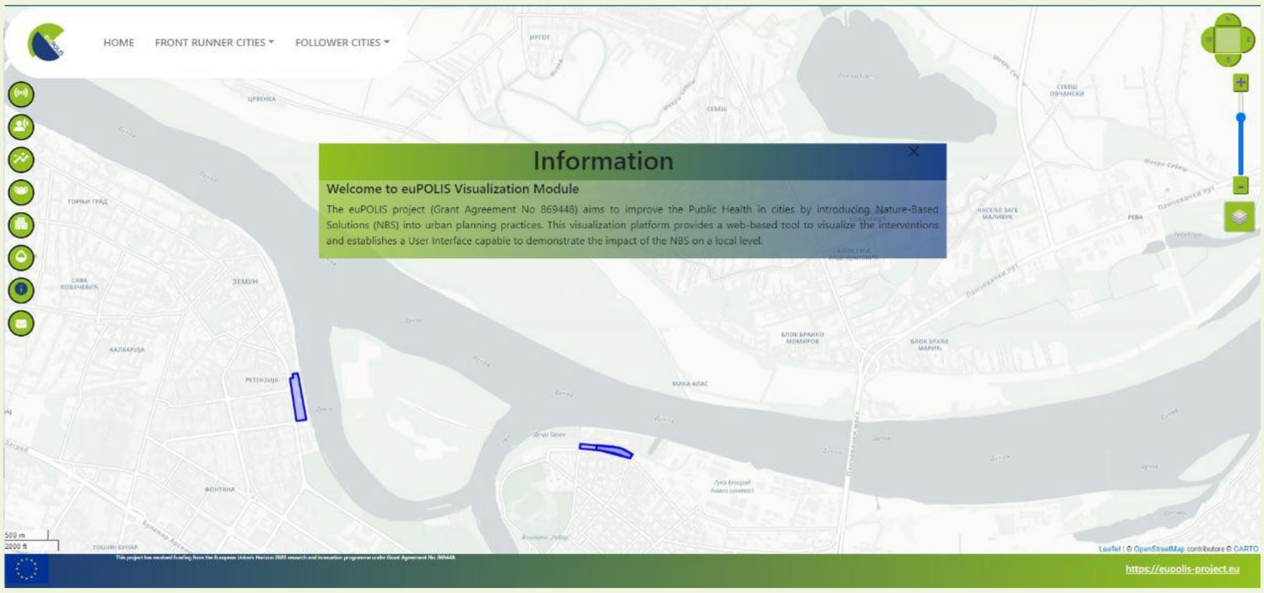
Location	[23.657906; 37.943798]
Total site surface (m^2)	3177.26
Pre-existing green surfaces (m^2)	635.0
Impervious surfaces (m^2)	2542.26

Click on the scenarios listed below to view the resulting tables and graphs

❖ This tool provides the results of the modeling that took place under the Task The user needs to select the desired city and the intervention area and then the dashboard will appear.

PROJECT INFORMATION & FEEDBACK

You can read the Project Information by pressing the button

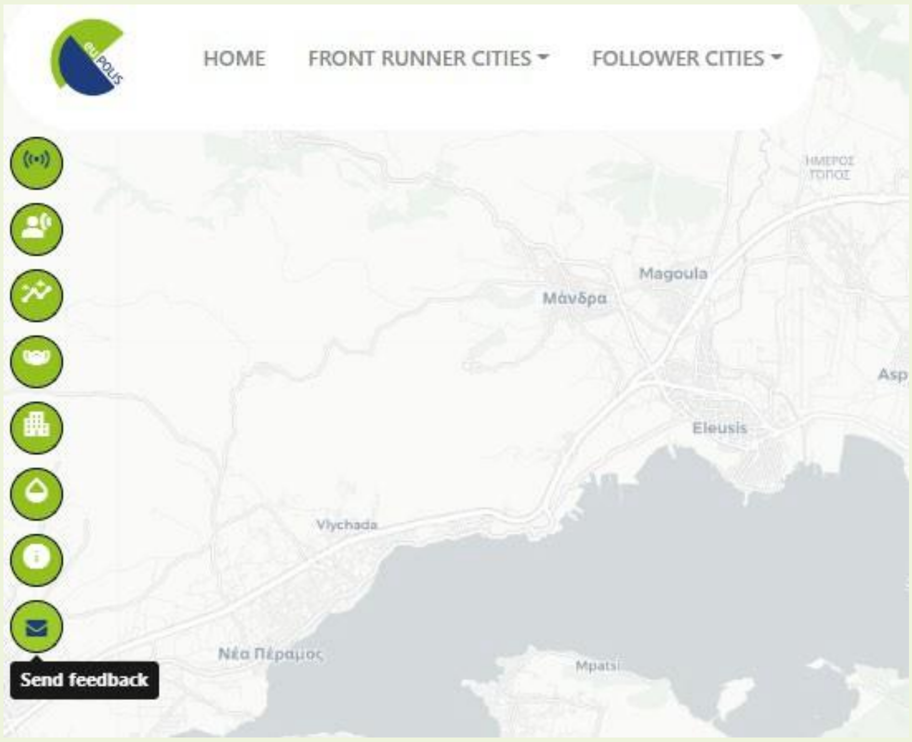


Information

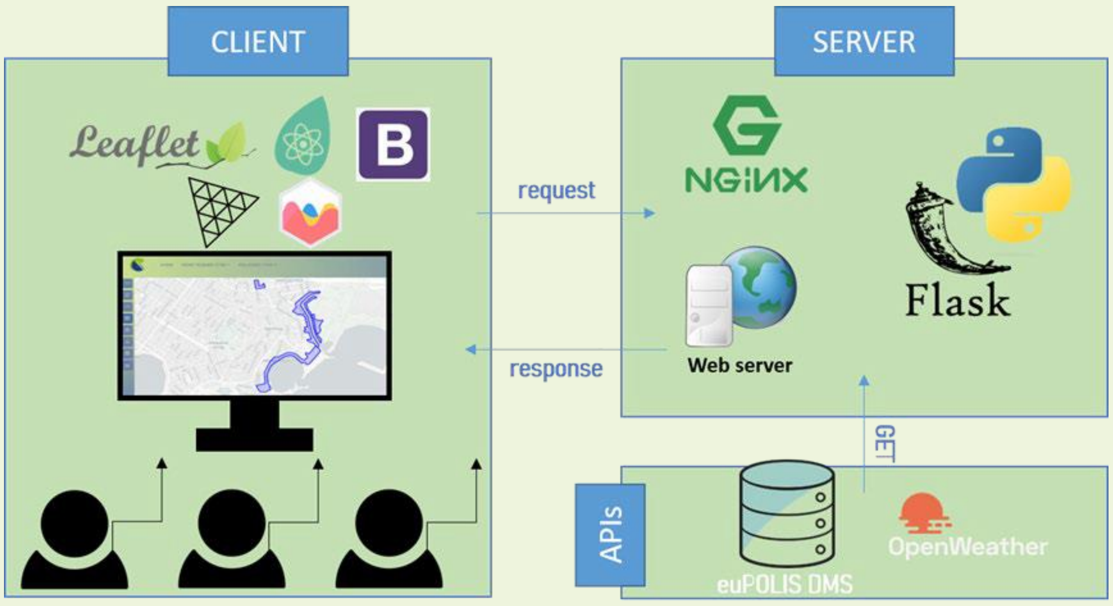
Welcome to euPOLIS Visualization Module

The euPOLIS project (Grant Agreement No 869448) aims to improve the Public Health in cities by introducing Nature-Based Solutions (NBS) into urban planning practices. This visualization platform provides a web-based tool to visualize the interventions and establishes a User Interface capable to demonstrate the impact of the NBS on a local level.

You provide feedback, via email, by pressing the button



DEVELOPMENT SECTION (Hardware-Software)



❖ Server: Local

❖ Storage: Local Storage System

❖ Operating System: Windows 10 Pro

❖ Web Server Software: NGINX, Ubuntu 22.04 LTS OS

❖ Programming Language:
- Python (Backend)
- JavaScript (Frontend)
- CSS (Frontend)
- HTML (Frontend)

❖ Visual Studio Code

❖ Device Specifications:

Intel®Core™i7-4720HQ CPU@2.60GHz

16.0 GB RAM

64-bit operating System, x64-based processor

https://www.facebook.com/eupolis2020

https://twitter.com/eu_polis

https://www.linkedin.com/company/eupolis

https://eupolis-project.eu

euPOLIS has received funding from the European Union's Horizon 2020 program