POLITECNICO DI TORINO

Master Degree course in Environmental and Land Engineering - Climate Change

Master Degree Thesis

Environmental Geo-Data in the Digital Era and the Importance of Open-Source Solutions for Spatial Analysis

Supervisors
Prof. Paolo Dabove
Dr. Martina Gizzi

Candidate
Fermin Maggi

Academic Year 2025-2026

Acknowledgements

Abstract

Environmental engineering increasingly depends on spatial data. Climate adaptation, water management, hazard assessment, and resource monitoring all rely on geographic information produced by satellites, ground sensors, models, and public monitoring networks. Many of these datasets are now openly available, but availability alone does not make them easy to use. In practice, data are often distributed in different formats, coordinate reference systems, metadata structures, and software-specific workflows. For research groups and small public institutions, the difficulty is therefore not only to find environmental data, but also to transform them into information that can be visualised, shared, and analysed in a reproducible way.

This thesis studies how open-source geospatial tools can help reduce this gap. It presents the design and implementation of *PoliTo EnviroHub*, a research-oriented WebGIS platform for visualising, managing, and analysing environmental monitoring data. The platform is built entirely with open-source components, including PostgreSQL/PostGIS, Python, GeoServer, FastAPI, Leaflet, Chart.js, Nginx, and Docker Compose.

PoliTo EnviroHub supports monitoring workflows where time-series sensor data need to be georeferenced, visualised, and analysed together with external spatial layers. The platform handles common CSV monitoring exports, supports raster and vector layers, and provides tools for resampling, smoothing, gap filling, trend analysis, and spatial interpolation. At the same time, the work recognises a practical limitation of this type of system as environmental data providers rarely use a single input structure. For this reason, the platform does not try to import every possible file automatically. Instead, the ingestion process is kept modular, so that known formats can be handled directly and new sources can be added through templates or specific import modules.

The platform is validated through a case study using real data from an alpine spring monitoring network in the Aosta Valley, supplied by the Applied Geology group at DIATI, Politecnico di Torino. The case study shows how long datalogger time series can be imported, stored, visualised on a map, explored through charts, and analysed through the browser. The results indicate that, for research-oriented environmental monitoring applications, an open-source stack can provide a practical and reproducible alternative to desktop-based or proprietary workflows. This thesis contributes in technical and methodological ways. It presents a working prototype and discusses the design choices, limitations, and possible extensions needed to transform environmental datasets into usable spatial information.

Contents

Chapter 1

Introduction

1.1 Context and motivation

Environmental work has become inseparable from spatial data. Mapping flood risk, monitoring water quality, analysing air pollution, evaluating land-cover change, or studying the response of alpine springs to rainfall and snowmelt all require information that is connected to a location. In the last two decades, the quantity of environmental geo-data has increased substantially. Public satellite missions such as Copernicus and Landsat, regional monitoring networks, open data portals, and global reanalysis products now provide access to datasets that were previously unavailable or difficult to obtain.

However, having available data does not mean that it will automatically become usable information. A satellite product or a datalogger export becomes useful only once someone has cleaned it, placed it in its geographic context, and compared it with other layers. This intermediate step is often the weakest part of the workflow. Data may be open but distributed in unfamiliar formats; it may have incomplete metadata, different coordinate reference systems, or file structures that depend on the provider. In practice, many research groups, small public bodies, NGOs, and local communities have access to environmental data but lack the integrated tools required to take full advantage of it.

This problem is particularly evident in environmental monitoring. Field sensors produce long time series of rainfall, temperature, water level, discharge, conductivity, or other variables. These observations are usually precise and valuable, but they are often managed through desktop software or file-based workflows that are difficult to share. A researcher may be able to visualise a single station locally, but comparing multiple monitoring points, adding contextual layers, or publishing the results to colleagues through a browser requires additional work. The gap is therefore not only technological, but also organisational, as data must be transformed from isolated files into an accessible spatial information system.

Geographic Information Systems (GIS) are the traditional tools used to manage and analyse spatial data. Desktop GIS remains essential for advanced data preparation, cartography, and specialist analysis. Nevertheless, when the main goal is to share information, centralise datasets, and provide access to users who should not need to install dedicated software, WebGIS becomes more appropriate. A WebGIS moves the data and

much of the processing logic to a server and gives users access through a standard web browser. This makes it especially useful for environmental research groups that need a common view of the same data, for public administrations that publish monitoring information, and for projects where reproducibility and accessibility are part of the scientific objective.

Figure 1.1 summarises the conceptual problem addressed by the thesis. Environmental data accessibility depends on more than publication. A dataset must pass through several intermediate steps before it becomes useful, including discovery, interpretation, format conversion, georeferencing, storage, visualisation, and analysis. Weakness at any of these steps can prevent open data from becoming usable information.

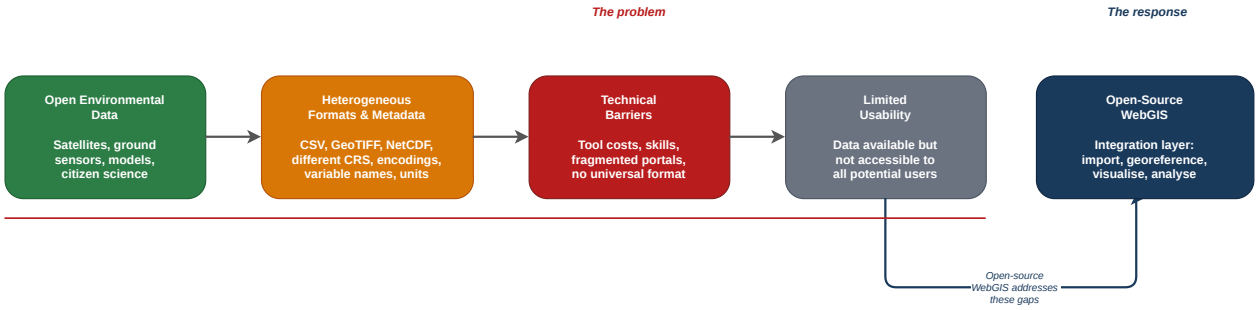

Figure 1.1. Conceptual framework of the thesis. The gap between environmental data availability and practical usability.

Open-source software is central to this discussion. Proprietary GIS solutions are powerful and widely used, but they can introduce licence costs, vendor lock-in, and reproducibility barriers. Open-source geospatial tools, by contrast, can be inspected, modified, redistributed, and deployed without licence fees. They also support many of the open standards promoted by the Open Geospatial Consortium (OGC), which makes it possible to build modular systems rather than depend on a single vendor. The practical challenge is that an open-source WebGIS is not a single product. It must be assembled from several components, including a spatial database, a map server, a backend, a frontend, and a deployment strategy. The quality of the architecture therefore determines whether the final system is maintainable or unnecessarily complex.

1.2 Aim and objectives

The central hypothesis of this thesis is that a carefully designed open-source WebGIS architecture can reduce the gap between environmental data availability and practical data usability, especially for research groups managing field monitoring networks. This

hypothesis is tested through the design and implementation of a working platform, PoliTo EnviroHub.

The aim is to develop a fully open-source WebGIS platform for the publication, visualisation, management, and basic analysis of environmental monitoring data. The platform is not intended to replace specialised desktop GIS software or advanced modelling tools. Instead, it aims to provide a simple, reproducible, and extensible environment where monitoring data can be placed on a map, explored through time-series charts, and analysed together with contextual spatial layers.

The specific objectives are:

- to review the role of environmental geo-data in spatial analysis and monitoring;
- to describe the differences between desktop GIS and WebGIS, identifying when a web-based approach is appropriate;
- to examine the open-source geospatial ecosystem and justify the selection of the tools used in the platform;
- to design a modular architecture based on open-source components and open geospatial standards;
- to implement a WebGIS prototype capable of ingesting common monitoring files, storing observations, displaying spatial layers, and running basic analysis tools;
- to validate the prototype using a real environmental monitoring dataset from an alpine spring network in the Aosta Valley;
- to discuss the limitations of the implemented system and identify future extensions, including support for additional data formats such as NetCDF.

1.3 Methodological approach

The work combines a literature and technology review with a practical software implementation. The review phase analyses environmental data sources, the function of GIS and WebGIS in environmental applications, and the current open-source geospatial stack. Particular attention is given to the distinction between open data and usable data. A dataset may be freely accessible while still requiring significant processing before it can support analysis.

The implementation phase follows an engineering approach. First, the requirements are defined for a typical monitoring scenario. Second, the software architecture is designed around separate components which are also interoperable, these are PostgreSQL/-PostGIS, FastAPI, GeoServer, Leaflet, Chart.js, Nginx, and Docker Compose. Third, the platform is implemented and validated with real data from the Aosta Valley alpine spring monitoring network. The validation focuses on practical usability, whether monitoring points can be imported, visualised, queried, and analysed through a browser, and whether the platform can replace part of a previous desktop-based workflow.

The thesis therefore has both a technical and a methodological contribution. The technical contribution is the implemented WebGIS prototype. The methodological contribution is the discussion of how open-source tools can be combined into a realistic architecture for environmental monitoring, including the unavoidable limitations that arise from heterogeneous input data.

Chapter 2

Environmental Geo-Data and WebGIS

Environmental engineering often deals with processes that vary in space and time. Floods, landslides, water resources, air quality, land-cover change, and spring discharge cannot be understood only as isolated measurements. They need to be located, compared with surrounding conditions, and followed through time. For this reason, environmental data are frequently also geographic data.

This chapter introduces the main sources and formats of environmental geo-data. It then discusses a practical problem that is central to the thesis. Many datasets are available, but they are not always ready to be used in a research workflow. The final part introduces GIS and WebGIS as the tools used to organise and share this information.

2.1 Sources and formats of environmental geo-data

Environmental geo-data come from several sources. Satellite missions, such as Copernicus Sentinel and Landsat, provide repeated observations over large areas. They are useful for applications such as land-cover mapping, vegetation monitoring, snow-cover analysis, surface-temperature estimation, and the detection of spatial patterns that would be difficult to observe from the ground.

Ground sensors provide a different type of information. Weather stations, rain gauges, river gauges, piezometers, soil-moisture probes, and water-quality sensors measure local conditions directly and often with high temporal resolution. Their limitation is spatial coverage, since each instrument describes one point, so the observations usually need to be interpreted together with other stations or contextual spatial layers.

Models and reanalyses fill part of the gap between direct observations and spatial continuity. Atmospheric reanalyses such as ERA5, climate models, and hydrological models provide gridded estimates that are widely used in environmental studies [18]. These products are useful, but they are often distributed in scientific formats such as NetCDF or GRIB, which require specific processing before they can be included in a WebGIS workflow.

Collaborative datasets can also provide useful geographic context. The most common

example is OpenStreetMap, which contains roads, buildings, land-use features, and other spatial information contributed by users. These data can be valuable in environmental applications, but their completeness and accuracy must be checked before use.

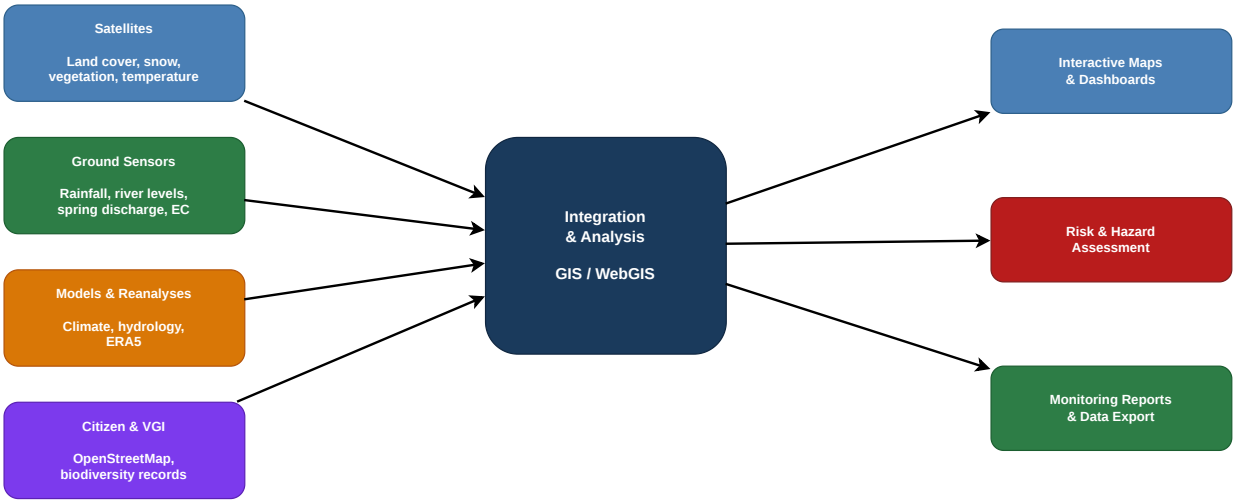

Figure 2.1. Main sources of environmental geo-data and their role in analysis. Satellites provide spatial coverage, ground sensors provide direct local measurements, models and reanalyses provide estimates where observations are missing, and collaborative data can add contextual information.

The variety of sources is reflected in the variety of formats. A monitoring station usually produces tabular time series, often as CSV files. Satellite and derived products are commonly distributed as raster grids. Boundaries, river networks, monitoring points, and infrastructure are usually vector layers. Climate and meteorological products may contain several variables and time steps in the same multidimensional file.

Table 2.1 summarises common formats used in environmental workflows. The purpose of the table is not to provide a complete catalogue, but to show why a WebGIS platform needs clear import procedures and cannot assume that every dataset will arrive in the same structure.

Table 2.1. Common environmental data formats and their implications for WebGIS integration.

Format	**Typical use**	**Main advantage**	**Main challenge**
CSV	Sensor time series, station data, regional downloads	Simple, readable, easy to export	Weakly standardised; separators, decimals, encodings, and timestamps vary
GeoTIFF	Raster layers, satellite products, derived maps	Spatial reference can be embedded; widely supported	Large files; styling and tiling may be needed for web display
Shapefile	Vector layers from legacy GIS workflows	Very common and widely supported	Multiple files; field-name limits; ageing format
GeoPackage	Modern vector/raster container	Single file; supports multiple layers	Less familiar to some non-specialist users
GeoJSON	Web vector data	Easy to use in web clients	Inefficient for very large datasets
KML	Simple geographic overlays	Easy exchange and visualisation	Limited for analytical workflows
NetCDF	Climate, oceanographic, atmospheric, and model data	Handles multidimensional scientific datasets	Requires variable/dimension selection and specialised parsing
GRIB	Meteorological forecasts and model outputs	Efficient for operational meteorology	Complex structure and metadata conventions

2.2 From available data to usable data

Open data policies have made many environmental datasets easier to find. European satellite data are distributed through the Copernicus programme [13]; national and regional agencies publish monitoring data; and research projects increasingly share datasets through public repositories. However, finding a dataset is only the first step.

A file may be public and still require work before it can be used. Coordinate reference systems may differ, metadata may be incomplete, timestamps may follow local conventions, and variable names may depend on the data provider. CSV files are a common example, as separators, decimal symbols, encodings, missing value codes, and date formats can all change from one source to another. These details are small individually, but together they determine whether a dataset can be imported directly or whether it needs preparation.

This issue is especially important for monitoring data. A spring, river gauge, weather station, or groundwater well is not only a point on a map. It is also a time series. To interpret it correctly, the user needs to know where the station is, what it measures, when the observations were collected, how many values are missing, and how the signal changes through time.

This is one of the reasons why monitoring data are a suitable case for a WebGIS. The map gives the geographic context, while charts and analysis tools help explore the observations. A platform such as PoliTo EnviroHub is therefore not only a way to display layers, but a way to keep monitoring locations, time-series data, and contextual information connected in the same interface.

2.3 European portals and monitoring-data heterogeneity

At European level, the INSPIRE Directive (2007/2/EC) has played an important role in improving the discovery and sharing of spatial datasets produced by public authorities. It covers several themes relevant to environmental engineering, including hydrography, geology, land cover, soil, atmospheric conditions, environmental monitoring facilities, and natural risk zones [12].

INSPIRE and related platforms have made official environmental datasets easier to find. The INSPIRE Geoportal, **data.europa.eu**, the **Copernicus Climate Data Store**, the **Copernicus Emergency Management Service**, and **WISE Freshwater** all contribute to this ecosystem [6,7,11,25]. For datasets such as land cover, protected areas, elevation models, hydrographic networks, or environmental indicators, these portals are valuable access points.

The situation is less uniform for observations collected by monitoring networks. Weather stations, river gauges, groundwater wells, and water-quality stations are often managed by different national, regional, or local agencies. The station locations may be discoverable through standard catalogues, but the measurements themselves may still be published through different portals, formats, variable names, temporal resolutions, and quality-control rules.

This becomes visible in border regions. In the Aosta Valley, useful meteorological or

hydrological observations may exist only a few kilometres away in France or Switzerland. Physically, these data may describe the same alpine system. From a data-management point of view, they usually belong to different monitoring networks and must be downloaded and prepared through separate workflows. A dataset from the Valle d'Aosta regional system, one from the French side of the Alps, and one from the Swiss Valais area may therefore be comparable in meaning, but not immediately compatible in structure.

For PoliTo EnviroHub, the consequence is practical. The platform cannot assume that every environmental dataset will arrive in a single standard format. It must support common inputs, but it must also leave room for templates, pre-processing, and additional import modules when new data sources are added.

2.4 GIS, WebGIS, and standards

A Geographic Information System stores, analyses, and displays data linked to a location. Its basic logic is the organisation of information in layers. Rivers, elevation, monitoring stations, land cover, buildings, or geological units can be managed separately and then analysed together. GIS operations such as overlays, buffers, coordinate transformations, spatial joins, interpolation, and raster analysis make this combination possible [29].

Desktop GIS and WebGIS are used for different parts of this workflow. Desktop GIS, such as QGIS or ArcGIS Pro, is still the most suitable environment for detailed data preparation, editing, cartography, and specialist analysis. WebGIS is more appropriate when the objective is to centralise data, publish updated layers, and allow users to access information through a browser without installing GIS software.

Table 2.2 summarises this distinction.

Table 2.2. Desktop GIS and WebGIS compared.

Aspect	**Desktop GIS**	**WebGIS**
Main purpose	Specialist analysis, editing, cartography, data preparation	Publication, sharing, browser-based access, collaborative use
Typical users	GIS specialists, analysts, researchers	Researchers, technicians, decision makers, non-specialist users
Installation	Required on each workstation	Not required by end users
Data location	Often local files or local database connections	Centralised database and server services
Collaboration	File sharing or shared databases	Multiple users access the same hosted data
Strengths	Advanced tools, flexibility, cartographic control	Accessibility, consistency, ease of dissemination
Limitations	Harder to share with non-specialists; software required	Less suitable for complex editing and advanced analysis
Best use case	Preparing and analysing data	Publishing and exploring data

For this reason, the platform developed in this thesis is not intended to replace desktop GIS. Desktop tools remain useful for preparing and checking data. PoliTo EnviroHub addresses another need, it provides a shared interface where monitoring points, time-series observations, and contextual spatial layers can be consulted and analysed through the browser.

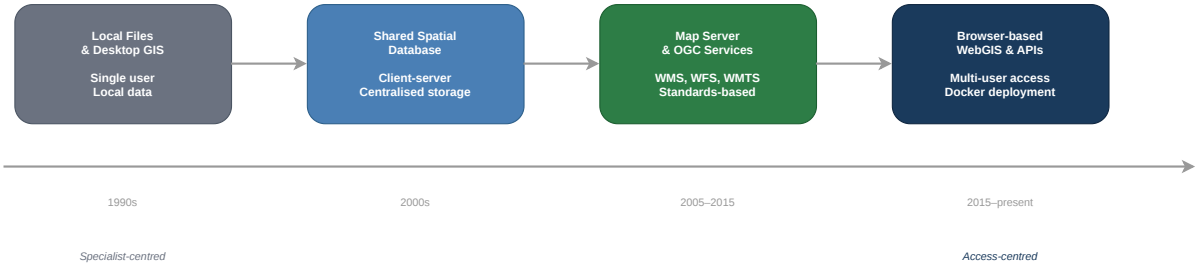

Figure 2.2. Evolution from desktop/file-based GIS workflows to web-based geospatial platforms.

A typical WebGIS includes a spatial database, a map server, and a web client (Figure 2.3). The database stores geometries, attributes, observations, and metadata. The map server publishes spatial layers through web services. The client displays the map and allows user interaction. Many systems also include a custom backend API for operations that are specific to the application, such as uploads, time-series queries, statistics, or authentication.

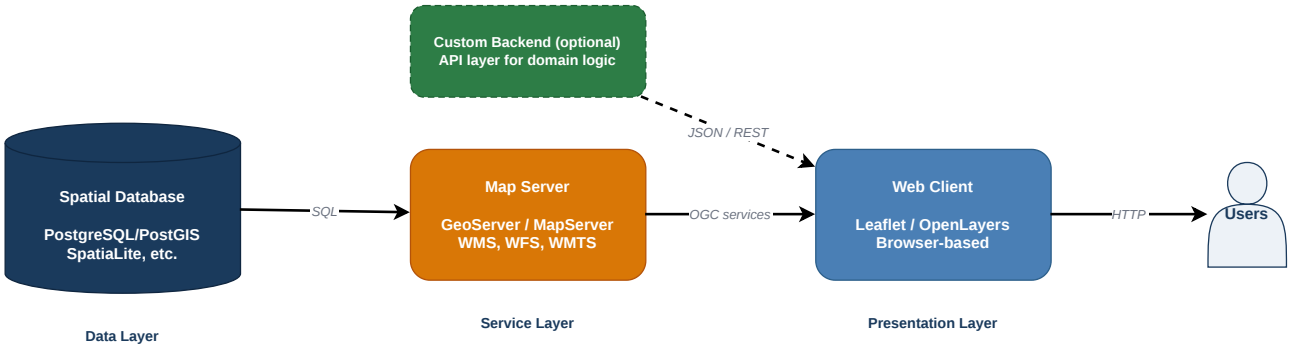

Figure 2.3. The layers of a typical WebGIS: spatial database, map server, and web client. Modern platforms often add a custom backend API to handle domain-specific logic that does not fit naturally into standard map services.

Interoperability is supported by standards defined by the Open Geospatial Consortium. WMS is commonly used to serve map images, WFS to access vector features, and WMTS to publish tiled map layers [21–23]. These standards are important because they allow different tools to communicate, but they do not cover every need of an environmental monitoring platform. Time-series queries, ingestion workflows, cleaning tools, and statistical analyses require dedicated endpoints.

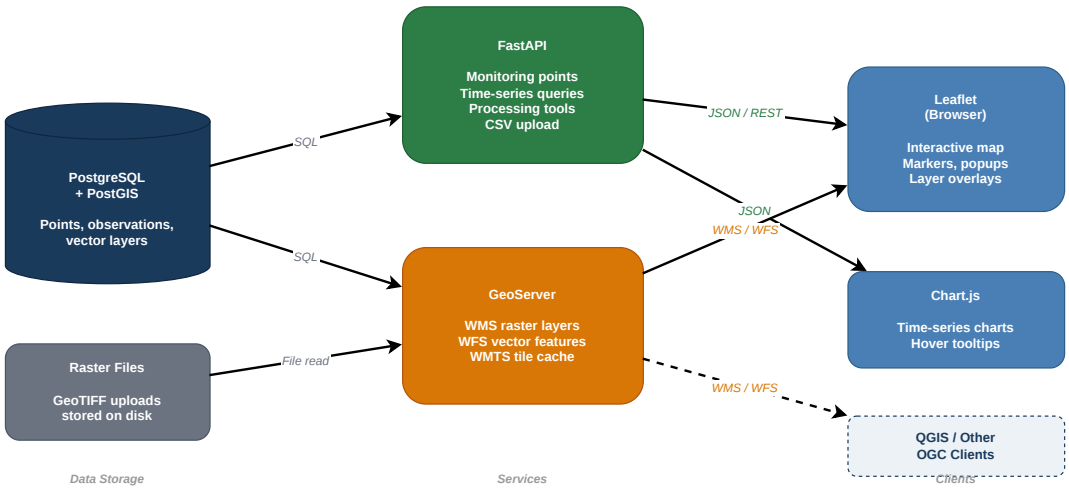

Figure 2.4. Role of open standards and custom APIs in the proposed WebGIS architecture.

This is why PoliTo EnviroHub combines GeoServer with a FastAPI backend. GeoServer is used for raster and vector layer publication through standard geospatial services. FastAPI handles the monitoring-specific operations that do not fit naturally into standard map services.

Chapter 3

Open-Source Geospatial Technologies

The previous chapter highlighted a central issue, environmental data may be open, but the tools and workflows needed to use it are not always accessible. This chapter presents the state of the art in GIS software, compares proprietary and open-source approaches, and explains why specific open-source tools were selected for PoliTo EnviroHub.

3.1 The GIS software state of the art

GIS software has evolved from specialised desktop applications to complete platforms that include desktop analysis, spatial databases, cloud services, server components, mobile applications, and web mapping tools. Proprietary platforms remain widely used in public administrations, utilities, private companies, and professional GIS environments. ESRI's ArcGIS product family is the most visible example, offering desktop, server, and cloud components in a closely integrated ecosystem.

At the same time, open-source geospatial software has matured substantially. The Open Source Geospatial Foundation (OSGeo) hosts and supports several major projects, including QGIS, GeoServer, MapServer, GDAL, and PostGIS [24]. Reviews of the open geospatial ecosystem show that open-source GIS is no longer a marginal alternative, but an established part of geospatial practice, especially in academia, research, NGOs, and public-sector organisations with strong reproducibility or budget requirements [5, 29].

The choice between proprietary and open-source GIS should not be framed only as a question of cost. Proprietary systems can reduce integration effort because they are designed as coherent product families. Open-source systems offer transparency, flexibility, and freedom from licence restrictions, but require more architectural responsibility. For a thesis project, this second aspect is valuable, designing the stack makes the technical choices explicit and reproducible.

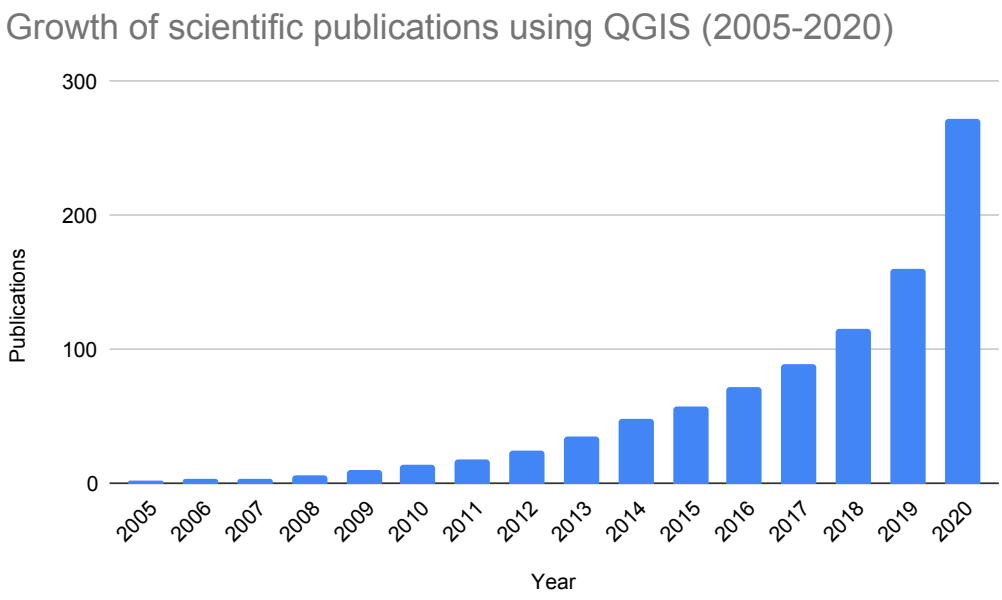

Figure 3.1. Growth of QGIS use in scientific publications (2005–2020). Data based on Duarte et al. [9].

3.2 Proprietary and open-source GIS approaches

The choice between proprietary and open-source GIS software is not only a matter of licence cost. Proprietary platforms, such as ArcGIS, provide an integrated ecosystem with desktop, server, cloud, and web components. This can be useful for organisations that need commercial support, official documentation, and a clear maintenance path.

In a research context, however, these advantages can also become limitations. Workflows based on proprietary tools may be difficult to reproduce if other users do not have access to the same licences or server components. The closed nature of the software also limits the possibility of inspecting or adapting the internal implementation.

Open-source GIS tools follow a different approach. Components such as QGIS, PostGIS, GeoServer, GDAL, and Leaflet require more integration work, but they make the workflow more transparent, adaptable, and reproducible. For this thesis, this is particularly important, as the objective is not only to use a WebGIS platform, but also to design and document an architecture that can be understood, deployed, and modified by a research group.

Table 3.1 summarises the main differences between the two approaches.

Table 3.1. Proprietary and open-source GIS stacks compared.

Aspect	**Proprietary GIS stack**	**Open-source GIS stack**
Licence cost	Licence-based; potentially expensive for server use	No licence fees; maintenance still requires expertise
Source code	Closed	Open and inspectable
Integration	Strong product ecosystem	Modular; integration must be designed
Customisation	Limited by vendor tools and APIs	High; code and configuration can be modified
Vendor lock-in	Potentially high	Lower, although custom architectures can still create dependencies
Support	Commercial support available	Community support and specialised companies
Reproducibility	Dependent on licence availability	Easier with documented and containerised environments

3.3 Related WebGIS platforms for environmental monitoring

PoliTo EnviroHub is part of a broader group of WebGIS platforms and environmental data portals that use interactive maps to make spatial information easier to access. Public

environmental agencies commonly use these systems to publish monitoring data, thematic layers, indicators, and geospatial services. A short comparison with existing platforms is useful to clarify the position of the present work and to highlight the specific need addressed by PoliTo EnviroHub.

3.3.1 Examples from Italian environmental agencies

In Italy, regional environmental agencies and functional centres use WebGIS platforms to publish environmental and territorial information. **ARPA Piemonte** operates a geoportal where users can consult, download, and access environmental geodata through interactive maps and web services [4]. This type of platform shows how regional environmental information can be organised around map-based access, supporting both technical users and the general public.

A second relevant example is the **Centro Funzionale** of the Regione Autonoma Valle d'Aosta, which operates the *DataView* portal for consulting meteorological, hydrological, snow, and environmental observations from the regional monitoring network [28]. This case is particularly relevant for the present thesis because the case study uses data from the Aosta Valley monitoring context. It also illustrates how point-based environmental observations can be made accessible through a web interface, together with spatial and temporal information.

These platforms are valuable references because they show that WebGIS has become a standard approach for regional environmental communication and data dissemination. However, they are institutional systems maintained by public bodies and connected to official monitoring networks. Their main objective is to publish validated environmental information, not to provide a flexible system that a small research group can install, adapt, and extend for its own datasets.

3.3.2 European environmental portals

Comparable examples can also be found at the European and national scale. The **European Environment Agency** provides several interactive maps and dashboards for environmental information. One relevant example is the European Air Quality Index, which presents air-quality measurements from monitoring stations across Europe through an interactive map interface [10]. This platform illustrates the importance of combining monitoring points, measured indicators, and simple visual communication for non-specialist users.

In Switzerland, the **Federal Office for the Environment** (BAFU/FOEN) provides access to current hydrological monitoring data, including information on rivers, lakes, groundwater, meteorological variables, and snow conditions [14]. The Swiss case is relevant because it shows how hydrological monitoring networks can be connected to maps, data tables, and time-dependent observations.

Figure 3.2 shows two examples of this type of platform. The first is closely connected to the case-study area, while the second is useful for comparison with another Alpine monitoring context.

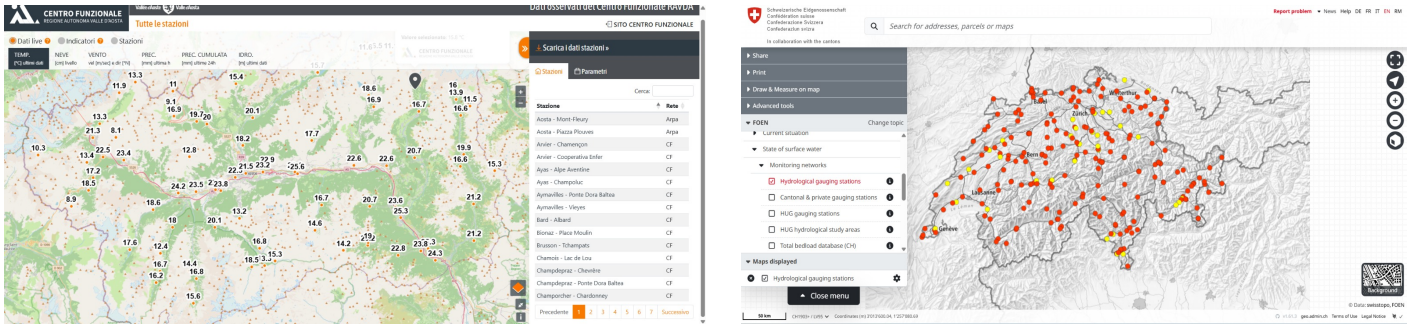

Figure 3.2. Examples of institutional web platforms used to consult environmental monitoring data. Centro Funzionale Valle d'Aosta DataView on the left and Swiss FOEN hydrological monitoring on the right. Both examples show how monitoring observations can be connected to map-based interfaces and time-dependent information.

These examples confirm that WebGIS platforms are widely used to communicate environmental information and monitoring data. At the same time, they also show that most public platforms are designed for institutional publication and long-term operational use. They usually depend on stable data infrastructures, dedicated technical teams, and official data flows. A university research group may instead need a smaller and more adaptable system, able to import project-specific datasets, test analysis tools, and support exploratory work.

3.3.3 Positioning of PoliTo EnviroHub

The comparison with existing platforms clarifies the role of PoliTo EnviroHub. Agency portals such as those of ARPA Piemonte, the Centro Funzionale Valle d'Aosta, the European Environment Agency, and FOEN demonstrate the value of map-based access for environmental monitoring data. They centralise information, make spatial data easier to consult, and reduce the need for users to work directly with raw GIS files.

PoliTo EnviroHub addresses a different scale and purpose. It is not intended to replace official environmental portals or to reproduce their operational coverage. Instead, it is a research-oriented WebGIS prototype designed to be lightweight, open-source, and adaptable. Its objective is to support common research-group needs, importing monitoring files, storing observations, displaying points and spatial layers on a map, exploring time series, and applying basic analysis tools through a browser.

Table 3.2 summarises this positioning.

Table 3.2. Comparison of selected WebGIS platforms for environmental monitoring.

Platform type	**Examples**	**Main purpose**	**Relevance for PoliTo EnviroHub**
Italian regional environmental portals	ARPA Piemonte, Centro Funzionale Valle d'Aosta	Publish environmental maps, monitoring information, and regional datasets	Show how regional environmental information can be organised through map-based access
European and national environmental portals	EEA Air Quality Index, FOEN hydrological data	Provide public access to official monitoring data and environmental indicators	Relevant examples of combining monitoring points, maps, and time-dependent data
Research-oriented WebGIS	PoliTo EnviroHub	Manage, visualise, and analyse project-specific environmental monitoring data	Smaller, open-source, and adaptable to university research workflows

3.4 Selection criteria for the platform stack

The open-source tools used in PoliTo EnviroHub were selected according to a set of practical criteria. These were maturity, documentation, community adoption, integration with other components, support for open standards, ease of deployment, and maintainability by a small research group. The objective was not to select the most complex or most recent technology, but the most appropriate combination for a master's thesis prototype that could realistically be understood, deployed, and extended.

Table 3.3. Technology alternatives considered during the design of PoliTo EnviroHub.

Role	**Chosen tool**	**Alternatives considered**	**Main reason for selection**
Desktop preparation	QGIS	ArcGIS Pro, GRASS GIS	Open-source, mature, cross-platform, widely used for inspection and preprocessing
Spatial database	PostgreSQL/PostGIS	SpatiaLite, MySQL Spatial, file-only storage	Robust spatial database, strong GeoServer and QGIS integration
Map server	GeoServer	MapServer, GeoNode, pygeoapi	OGC services, web administration, REST API, PostGIS integration
Backend API	FastAPI	Flask, Django	Clear API structure, automatic documentation, Python scientific ecosystem
Web map client	Leaflet	OpenLayers, MapLibre GL JS	Simplicity, lightweight client, easy maintenance
Charts	Chart.js	Plotly, D3.js	Simple interactive time-series visualisation
Deployment	Docker Compose	Manual installation, Kubernetes	Reproducible local/server deployment without excessive complexity

3.5 Leading open-source tools

The tools selected for PoliTo EnviroHub were chosen because they are mature, well documented, widely adopted, and interoperable. The objective was not to use the newest possible technology, but to build a stable system that a research group could understand and maintain.

3.5.1 Desktop GIS: QGIS

QGIS is the most widely adopted open-source desktop GIS. It supports major vector and raster formats, integrates with PostGIS, GDAL, GRASS GIS, and SAGA, and can be extended through a large plugin ecosystem [26]. A recent study of the QGIS project describes its development, governance, and adoption as a mature open-source geospatial platform [17]. In this thesis, QGIS is not part of the deployed WebGIS platform, but it remains an important supporting tool for data inspection, preparation, and validation.

3.5.2 Spatial database: PostgreSQL/PostGIS

PostGIS extends PostgreSQL with spatial data types, spatial indexes, and a large set of spatial functions accessible through SQL [2]. It is a common foundation for WebGIS applications because it can store geometries, attributes, metadata, and time-series relationships in a single relational model. Its integration with QGIS and GeoServer also makes it a natural choice for an open-source geospatial stack. In PoliTo EnviroHub, PostGIS stores monitoring points, variables, observations, uploaded layer metadata, and vector geometries.

3.5.3 Map server: GeoServer

GeoServer is an open-source server for sharing geospatial data through open standards [1]. It can publish data from PostGIS and file-based sources using services such as WMS, WFS, WCS, and WMTS. It also provides a browser-based administration interface and a REST API, both of which are useful in a research prototype. GeoServer was selected for this thesis because it is mature, widely used, compatible with PostGIS, and convenient to deploy in a containerised architecture.

GeoServer is not the only possible option. MapServer is powerful and efficient, but its configuration is less convenient for a project where ease of management is important. GeoNode provides a complete geospatial content-management platform, but it would be too heavy for a thesis focused on designing a custom monitoring workflow. pygeoapi is promising for modern OGC APIs, but the traditional WMS/WFS publication model remains simpler for the raster and vector overlays required here. Tile servers are useful for high-performance basemaps or vector tiles, but they do not provide the broader geospatial service layer needed by this project. Table 3.4 summarises the choice.

3.5.4 Web mapping: Leaflet

Leaflet is a lightweight JavaScript library for interactive maps in the browser [3]. Its main strength is simplicity. It provides the core features needed for map navigation, markers, popups, layer control, and raster overlays without requiring a complex frontend architecture. OpenLayers offers more built-in GIS functionality and may be preferable for advanced web mapping applications, but its API is more complex. Leaflet was selected because the platform is intended to remain understandable to researchers with limited frontend experience.

Table 3.4. Map server alternatives considered for the platform.

Tool	**Strengths**	**Reason for not selecting it as main option**
GeoServer	Mature, PostGIS integration, web interface, OGC services, REST API	Selected as the most suitable balance between capability and usability
MapServer	Very efficient and mature	Configuration less accessible for this project
GeoNode	Complete geospatial portal with user management and cataloguing	Too heavy; would hide much of the thesis contribution
pygeoapi	Modern OGC API implementation	Less aligned with the simple WMS/WFS layer-publication workflow required here
TileServer GL	Good for vector tiles and basemaps	Too specialised; not sufficient for monitoring and analysis needs

3.5.5 Backend: Python and FastAPI

FastAPI is a Python framework for building web APIs [27]. It was chosen because it provides automatic API documentation, type-based validation, and clean integration with Python's scientific libraries. Python is also a practical choice for environmental researchers, who frequently use NumPy, Pandas, SciPy, Rasterio, and GeoPandas for analysis. In PoliTo EnviroHub, FastAPI handles monitoring-specific logic that does not fit naturally into standard map services, including CSV ingestion, time-series queries, resampling, statistical tools, and interpolation.

3.5.6 Supporting tools

GDAL/OGR is the fundamental library for reading, writing, and converting geospatial formats [15]. Many other geospatial tools depend on it, including QGIS, Rasterio, and GeoPandas. Docker and Docker Compose package the platform into reproducible containers [8]. This is especially useful for open-source systems because it reduces the installation burden created by multiple independent components.

3.6 Open-source, reproducibility, and FAIR principles

Open-source software is closely related to reproducibility. A workflow is more reproducible when its code, configuration, dependencies, and data assumptions can be inspected and repeated by another user. This aligns with the FAIR principles for scientific data management, which emphasise that digital objects should be Findable, Accessible, Interoperable, and Reusable [30]. Although FAIR principles are often discussed in relation to datasets, they also apply to the tools and workflows used to process those datasets.

In this thesis, reproducibility is addressed in three ways. First, the software stack uses open-source components. Second, the deployment is containerised, so the same configuration can be started on different machines. Third, the data model explicitly stores information about monitoring points, variables, units, and observations, instead of leaving these details implicit in external files. The platform does not implement a complete metadata catalogue, but it adopts a minimum metadata structure that supports interpretation and future extension.

Chapter 4

PoliTo EnviroHub: Design and Implementation

This chapter describes the design and implementation of PoliTo EnviroHub, a general-purpose WebGIS platform for environmental monitoring built entirely with open-source tools. The goal is to show the choices made at each level of the system so that the methodology can be understood, reproduced, and adapted to similar projects.

4.1 Requirements

The platform was designed around a common scenario in environmental research, where a group operates a network of field sensors and needs to store, visualise, and analyse the data they produce. Typically, the raw files are kept in proprietary desktop software or spreadsheets, disconnected from geographic context and difficult to share with other users. The requirements were therefore defined to address both the data-management problem and the accessibility problem at the same time.

Table 4.1 separates functional and non-functional requirements. The functional requirements describe what the system must do. The non-functional requirements describe how the system should behave and what constraints it must satisfy.

These requirements led to a few design principles that guided the rest of the work. The platform favours a small set of reliable functions over a large set of fragile ones, since a research prototype is more useful when its core features can be trusted. Its parts are kept modular, so that data storage, map publication, analysis logic, and the user interface can evolve independently. The design is extensible, allowing new monitoring types and parsers to be added without rewriting the application. Finally, the whole system is reproducible, with installation and configuration documented and containerised.

The platform supports five monitoring point types out of the box. These are alpine springs, weather stations, river level gauges, water reservoirs, and water quality stations. Each type defines its own set of variables, and new types can be added by extending the database seed file and the variable definitions. This choice keeps the core code stable while allowing researchers to adapt the platform to new monitoring contexts.

Table 4.1. Functional and non-functional requirements of PoliTo EnviroHub.

Category	**Requirement**
Functional	Import monitoring points with geographic coordinates and metadata
Functional	Upload common CSV-based monitoring files and map columns to internal variables
Functional	Store observations as time-stamped values linked to points and variables
Functional	Display monitoring points on an interactive web map
Functional	Visualise time series through interactive charts
Functional	Support raster and vector overlays for contextual spatial information
Functional	Provide basic time-series tools, including resampling, smoothing, gap filling, and trend analysis
Functional	Provide basic spatial analysis, especially interpolation between monitoring points
Non-functional	Use only open-source components and avoid proprietary licence dependencies
Non-functional	Be accessible through a browser without installing GIS software on the client machine
Non-functional	Be deployable through a documented and reproducible containerised environment
Non-functional	Keep the architecture simple enough to be maintained by a research group
Non-functional	Make data ingestion extensible, recognising that no universal parser can handle every provider format
Non-functional	Preserve minimum metadata needed to interpret points, variables, units, and sources

4.2 Architecture

The platform follows a four-tier architecture (Figure 4.1), made of a browser-based client, a reverse proxy, two backend services, and a spatial database. All components run as Docker containers on a single host and are orchestrated by Docker Compose.

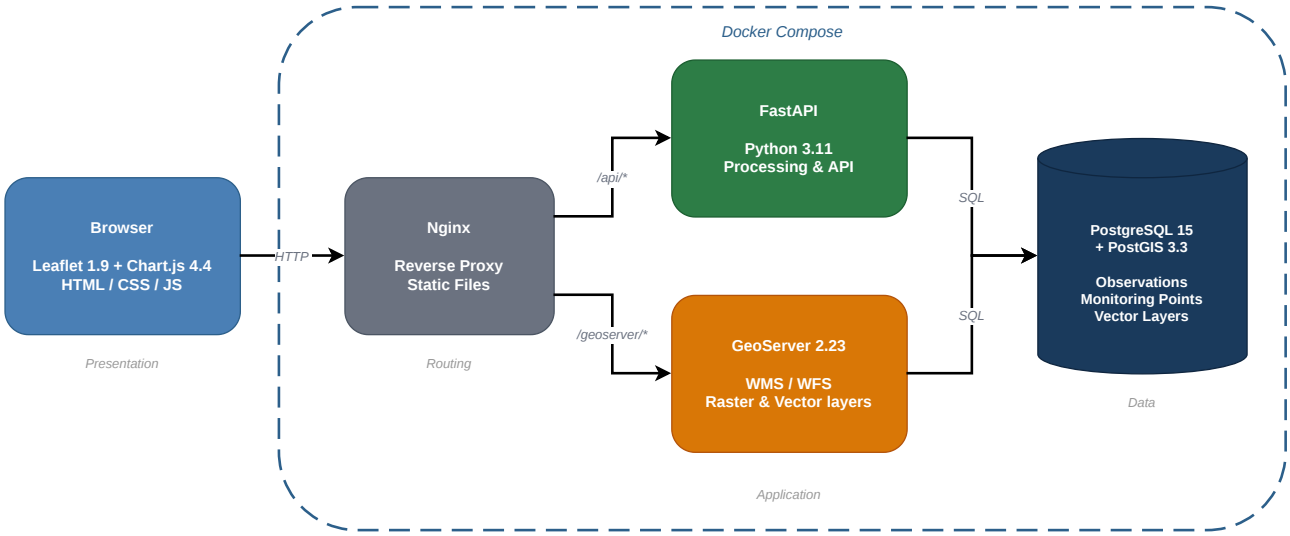

Figure 4.1. Architecture of PoliTo EnviroHub. The browser communicates with Nginx, which routes requests to the FastAPI backend for monitoring data and processing, or to GeoServer for raster and vector layers. Both services are connected to PostgreSQL/PostGIS. The dashed line indicates the Docker Compose boundary.

A key design decision is that GeoServer is not on the critical path for the everyday monitoring workflow. Monitoring points and observations are served directly by FastAPI as JSON, while GeoServer is reserved for raster overlays and larger vector layers that benefit from standard geospatial services. This separation brings two advantages. The platform can offer monitoring-specific queries and analysis tools without forcing time-series data into WMS or WFS responses, and the user interface stays usable even when GeoServer takes time to initialise after startup.

The architecture therefore gives each component a clear responsibility. PostGIS stores spatial and temporal data, FastAPI exposes domain-specific endpoints and processing tools, GeoServer publishes geospatial layers through standards, Nginx routes requests and serves static files, and Leaflet with Chart.js provides the browser interface. This modularity matters because the system can be extended one component at a time. OpenLayers could replace Leaflet without touching the database, and new processing tools could be

added to FastAPI without modifying GeoServer.

Figure 4.2 describes the logical data flow. Raw input files are not exposed directly to the user interface. They are first parsed, validated, normalised, and stored in the database, and only then become available through API endpoints, charts, maps, or GeoServer-published layers. This separation between raw input and internal representation is essential for handling heterogeneous provider formats.

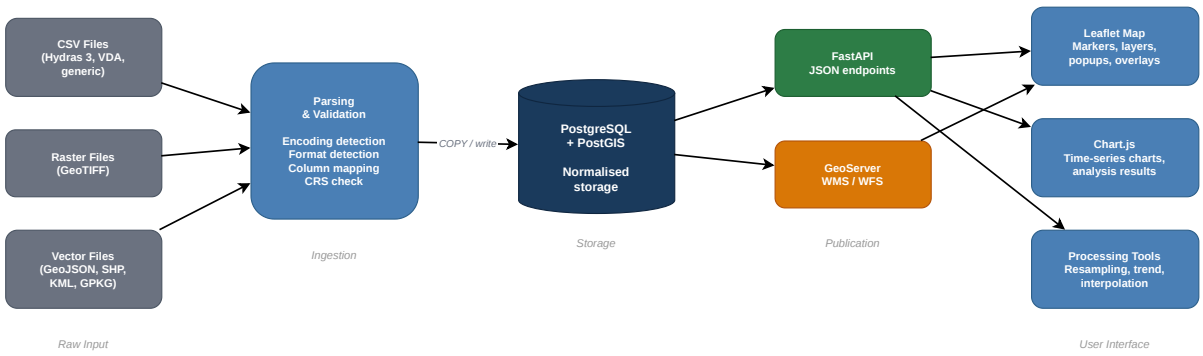

Figure 4.2. Logical data flow from raw environmental files to WebGIS visualisation and analysis.

4.3 Components and tool choices

Each component was chosen to meet a specific need, following the criteria discussed in Chapter 3. Table 4.2 lists the main tools and their role in the platform.

The choice of vanilla JavaScript instead of a framework such as React or Vue is deliberate. The platform is meant to stay understandable to researchers with limited frontend experience, and a framework would add build steps, extra dependencies, and a steeper learning curve without a clear benefit at the current scale. If the platform grows into a larger application, migration to a frontend framework would be possible, but it is not necessary for the thesis prototype.

Table 4.2. Components of PoliTo EnviroHub and their role.

Layer	**Tool**	**Role**
Database	PostgreSQL 15 + PostGIS 3.3	Spatial and time-series storage
Map server	GeoServer 2.23	Publishing raster and vector layers
Backend	Python 3.11 + FastAPI 0.111	Custom API and processing logic
Frontend	Vanilla JS + Leaflet 1.9	Interactive map and user interface
Charts	Chart.js 4.4	Time-series visualisation
Reverse proxy	Nginx (Alpine)	Routing and static file serving
Pre-processing	GDAL, Rasterio, GeoPandas	Format conversion and reprojection
Orchestration	Docker Compose	Containerisation and deployment

4.4 Representative implementation extracts

The complete source code is not reproduced here, but a few short extracts help document the main implementation choices. They show how the architecture described above turns into a reproducible deployment, API-based access to the monitoring data, and an ingestion path fast enough for large datalogger files. The listings are lightly abridged for readability, since error handling, logging, and some configuration are omitted, but they otherwise reflect the deployed code.

```
services:
  db:
    image: postgis/postgis:15-3.3
    environment:
      POSTGRES_DB: envirohub
      POSTGRES_USER: envirohub
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    volumes:
      - pg_data:/var/lib/postgresql/data
      - ./docker/init.sql:/docker-entrypoint-initdb.d/init.sql
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U envirohub"]
      interval: 5s
      retries: 20

  geoserver:
    image: kartoza/geoserver:2.23.0
    volumes:
      - geoserver_data:/opt/geoserver/data_dir
      # Raster directory shared with the backend: the backend writes
      # GeoTIFFs here and GeoServer reads them from the same path.
      - raster_shared:/opt/geoserver/data_dir/rasters
    ports:
      - "8080:8080"

  backend:
    build: ./backend
    environment:
      DATABASE_URL: postgresql://envirohub:***@db:5432/envirohub
      GEOSERVER_URL: http://geoserver:8080/geoserver
      RASTER_DIR: /geoserver_rasters
      # (admin, session and SMTP variables omitted)
    volumes:
      - ./data:/app/data
      - raster_shared:/geoserver_rasters
    depends_on:
      db:
        condition: service_healthy

  frontend:
    image: nginx:alpine
```

```
    volumes:
      - ./frontend:/usr/share/nginx/html:ro
      - ./docker/nginx.conf:/etc/nginx/conf.d/default.conf:ro
    ports:
      - "80:80"
    depends_on:
      backend:
        condition: service_healthy

volumes:
  pg_data:
  geoserver_data:
  raster_shared:
```

Listing 4.1. Extract from `docker-compose.yml` with the four core services and the raster volume shared between the backend and GeoServer.

```
@router.get("/{point_id:int}/timeseries")
async def get_timeseries(
    point_id: int,
    variable_key: str = Query(...),
    from_date: Optional[datetime] = Query(None),
    to_date: Optional[datetime] = Query(None),
    max_points: int = Query(4000, le=50000), # cap on points sent to the chart
):
    # If no window is given, return the whole series.
    fr = from_date or datetime(2000, 1, 1)
    to = to_date or datetime(2099, 12, 31)

    # Count first, then thin server-side: keep every Nth row.
    n_total = (await database.fetch_one("""
        SELECT COUNT(*) AS n FROM observations
        WHERE point_id = :pid AND variable_key = :vk
          AND timestamp BETWEEN :fr AND :to
    """, {"pid": point_id, "vk": variable_key, "fr": fr, "to": to}))["n"]

    thin = max(1, n_total // max_points)

    rows = await database.fetch_all("""
        SELECT timestamp, value, quality FROM (
            SELECT timestamp, value, quality,
                   ROW_NUMBER() OVER (ORDER BY timestamp) AS rn
            FROM observations
            WHERE point_id = :pid AND variable_key = :vk
              AND timestamp BETWEEN :fr AND :to
        ) sub
        WHERE rn % :thin = 0 OR rn = 1 OR rn = :n_total
        ORDER BY timestamp
    """, {"pid": point_id, "vk": variable_key,
          "fr": fr, "to": to, "thin": thin, "n_total": n_total})

```

```
    return [dict(r) for r in rows]
```

Listing 4.2. FastAPI endpoint for retrieving a monitoring point's time series, thinned server-side so long charts stay responsive.

```
async def _fast_bulk_insert(database, point_id, records):
    """Bulk␣load␣via␣PostgreSQL␣COPY␣into␣a␣temp␣table,␣then␣upsert."""
    tmp = f"_obs_import_{uuid.uuid4().hex[:8]}"
    tuples = [
        (point_id, r["variable_key"], r["timestamp"],
         float(r["value"]), r["quality"])
        for r in records
    ]

    async with asyncpg_pool.acquire() as conn:
        async with conn.transaction():
            await conn.execute(f"""
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣CREATE␣TEMPORARY␣TABLE␣{tmp}␣(
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣point_id␣INTEGER,␣variable_key␣TEXT,␣timestamp␣TIMESTAMPTZ,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣value␣NUMERIC,␣quality␣TEXT
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣)␣ON␣COMMIT␣DROP
␣␣␣␣␣␣␣␣␣␣␣␣""")

            # The big load: stream all rows over the COPY protocol.
            await conn.copy_records_to_table(
                tmp, records=tuples,
                columns=["point_id", "variable_key", "timestamp",
                         "value", "quality"],
            )

            # Move temp -> permanent. A new raw Hydras value may replace an
            # earlier estimated one, but an estimate never overwrites raw data.
            result = await conn.execute(f"""
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣INSERT␣INTO␣observations
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(point_id,␣variable_key,␣timestamp,␣value,␣quality)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣SELECT␣point_id,␣variable_key,␣timestamp,␣value,␣quality
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣FROM␣{tmp}
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ON␣CONFLICT␣(point_id,␣variable_key,␣timestamp)␣DO␣UPDATE
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣SET␣value␣=␣EXCLUDED.value,␣quality␣=␣EXCLUDED.quality
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣WHERE␣EXCLUDED.quality␣=␣'raw'␣AND␣observations.quality␣<>␣'raw'
␣␣␣␣␣␣␣␣␣␣␣␣""")

    inserted = int(result.split()[-1])
    return inserted, len(tuples) - inserted
```

Listing 4.3. Bulk loading with PostgreSQL `COPY` into a temporary table, followed by an upsert that preserves raw data over estimated values.

4.5 Data model and stored metadata

The database schema is built around three ideas, monitoring points, variable definitions, and observations (Figure 4.3). Each point belongs to a monitoring type, each type defines the variables it can record, and each observation links a value to a point and a variable at a given time. A separate table holds conversion curves, which derive secondary variables from raw measurements, for example turning water level into discharge through a rating curve.

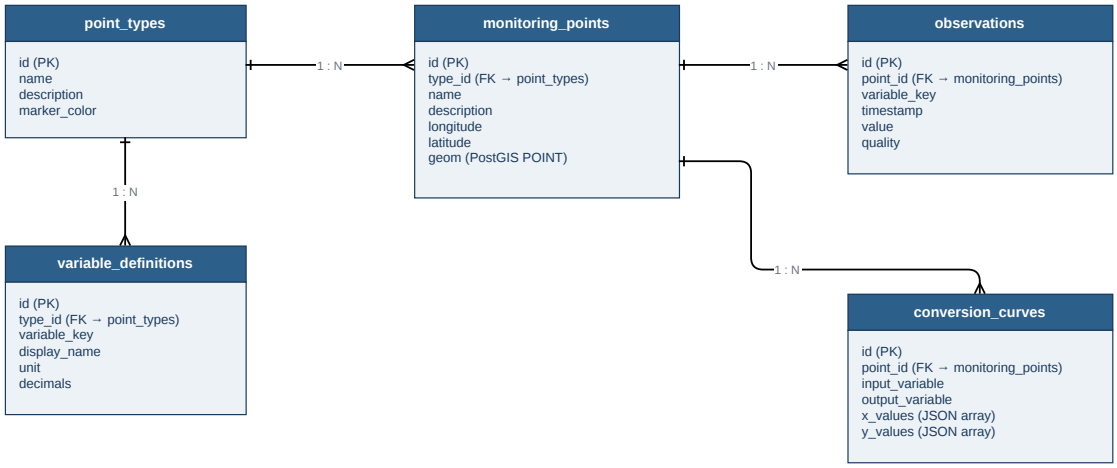

Figure 4.3. Simplified entity-relationship diagram. Monitoring points are typed, each type defines its variables, and observations link a value to a point and a variable at a given time.

This structure was preferred over a separate table for each monitoring type because it avoids duplicating logic. A spring, a weather station, and a river gauge differ in the variables they measure, but the relationship between point, variable, timestamp, and value is the same for all of them. Storing observations in a common table makes it easier to build generic time-series tools and to add new monitoring types later.

Indexes on point identifiers, variable identifiers, and timestamps are needed because time-series queries are the most frequent operation, and spatial indexes on the geometries support map queries and future spatial filters. The model is intentionally simple, favouring clarity and maintainability over specialised time-series database features. For the

scale of the case study, PostgreSQL with PostGIS is sufficient and avoids adding another component to the stack.

The same schema also carries the metadata needed to interpret the data. Each monitoring point stores a name, a type, geographic coordinates, and optional descriptive attributes such as municipality or elevation. Each variable definition stores an internal key, a display name, a unit, and the point types it applies to. Because observations are linked to both a point and a variable, there is no ambiguity when different sensors measure similar quantities. Uploaded spatial layers keep basic information about their source, type, coordinate reference system, and publication status.

This is not a full metadata catalogue, and it does not implement standards such as ISO 19115. It does, however, record the minimum needed for a user to understand what a value represents, where it was measured, when it was collected, and which variable it belongs to. Together with the open-source stack and the containerised deployment described later in this chapter, this structure is what makes the workflow reproducible rather than dependent on details hidden in external files.

4.6 Data ingestion and input formats

Importing real sensor files is the part of the workflow where most practical issues arise, and in this platform it is often more delicate than displaying the data. Monitoring files arrive from datalogger software, manual spreadsheets, regional portals, GIS exports, and already processed raster products. Even when the file extension is the same, the internal structure can change considerably.

The platform handles CSV uploads through a pipeline (Figure 4.4) designed around three recurring problems. The first is encoding. Environmental CSVs are not always written in UTF-8, and Italian regional files or legacy datalogger exports may use Latin-1 or CP1252, so the pipeline tries several encodings in turn and keeps the first one that decodes the file correctly. The second is format. The structure of a file often reveals its origin, so the pipeline recognises known formats and applies the matching parser, while a guided mode lets the user preview an unfamiliar file and map its columns to internal variables by hand. The third is performance. Long monitoring files can contain hundreds of thousands of rows, and an early row-by-row insertion strategy caused timeouts, so the platform streams the data into a temporary table with PostgreSQL's `COPY` protocol before validation and deduplication.

Parameter names written in Italian, such as "Temperatura", "Precipitazione", and "Umidità relativa", are normalised by stripping accents and matched against a dictionary of internal variable keys. This makes ingestion more robust to the language of the source file, but it does not remove the need for manual checks in every case.

For this reason the platform does not treat ingestion as a fully automatic step. A CSV is examined for encoding, separator, decimal convention, date format, metadata rows, and variable names before anything is written to the database. If the file matches a known structure, such as the Hydras 3 exports used in the case study, the corresponding parser is applied. If it does not, the guided upload lets the user align the original columns with the internal variables of the platform. Spatial files follow the same logic. Common formats can

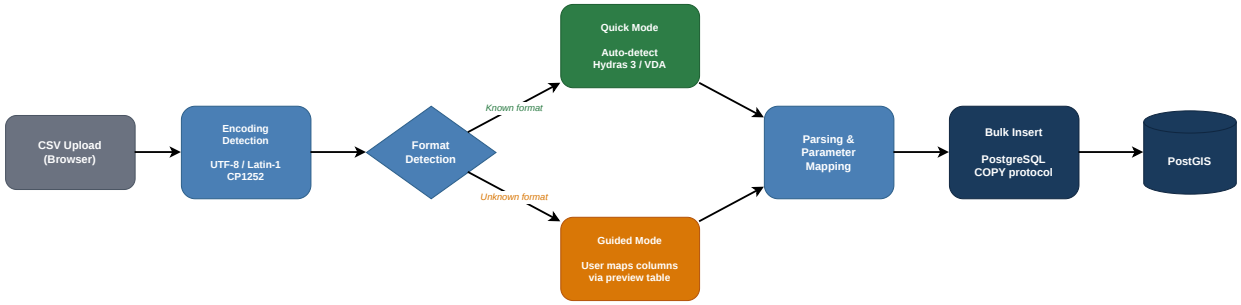

Figure 4.4. CSV ingestion pipeline. The Quick mode auto-detects known formats; the Guided mode lets the user manually map columns. Both paths converge on a bulk-insert phase using PostgreSQL's `COPY` protocol.

be imported directly, while files that need reprojection, field cleaning, or layer selection are better prepared beforehand in QGIS, GDAL, or Python. The platform therefore combines direct import with external pre-processing, using templates that describe the expected structure of the input data.

Table 4.3. Input formats considered in PoliTo EnviroHub.

Format	**Current status**	**Use in the platform**
CSV	Supported	Main format for monitoring time series; known structures are parsed automatically, while other files can be imported through guided column mapping.
GeoPackage, GeoJSON, Shapefile, KML	Supported	Used for vector layers and monitoring points, after checking geometry, attributes, and coordinate reference system.
GeoTIFF	Supported	Used for raster overlays, generally after inspection or preprocessing outside the platform.
NetCDF, GRIB, HDF	Future extension	Relevant for climate, meteorological, and remote-sensing products, but they require dedicated modules for variables, dimensions, and time steps.
Sensor APIs	Future extension	Useful for near-real-time workflows, but not implemented in the current prototype.

This solution is deliberately simple. The aim is not to build a universal converter for every environmental dataset, but to keep ingestion clear, reproducible, and open to extension when a new data source has to be added.

4.7 Spatial layer management

Beyond monitoring data, the platform supports raster and vector overlays. Raster layers such as GeoTIFF files are inspected with GDAL, reprojected to EPSG:4326 if needed, and published through GeoServer. Vector layers such as GeoJSON, Shapefile, KML, and GeoPackage files are read with GeoPandas, stored in PostGIS, and published in GeoServer. Small layers can also be loaded directly into Leaflet as interactive GeoJSON with click-to-inspect popups.

This distinction between monitoring data and contextual layers is important. Monitoring observations are dynamic, queried frequently, and connected to analysis tools, whereas contextual layers are often larger and more suitable for standard map publication. GeoServer is therefore used where it adds value, while FastAPI remains responsible for domain-specific logic.

4.8 Analysis tools

The platform provides three families of analysis tools, namely data-cleaning, time-series, and spatial tools. The objective is to support exploratory analysis directly in the browser, without forcing the user to export the data to an external environment for routine operations. More advanced or domain-specific analysis is intentionally left to specialised software.

Data-cleaning tools address the most common quality issues that appear in long monitoring records. The main one is outlier detection. Long time series almost always contain anomalous values, like sudden spikes caused by sensor malfunctions, instrumental drift after maintenance, or non-physical readings produced by power failures. Removing or flagging these values is a prerequisite for any meaningful analysis. The platform provides a dedicated outlier-detection panel where the user selects a point, a variable, and a time window, and chooses a detection method. Detected outliers are highlighted on the time-series chart so that the user can inspect them before deciding whether to flag or delete them. Two related operations support the cleaning workflow, the deletion of a selected data range, which removes a continuous block of observations identified as invalid, and the application of conversion curves, which derive secondary variables such as discharge from water level using a per-point rating curve stored in the database.

Time-series tools operate on a single variable at a single monitoring point. They include resampling at user-defined frequencies (hourly, daily, weekly, monthly) with aggregation by mean, minimum, maximum, or sum; gap filling through linear, nearest-neighbour, or forward-fill interpolation; linear trend analysis returning slope per unit time, coefficient of determination, p-value, and a significance flag at $\alpha = 0.05$; recession-curve fitting of the form

$$Q(t) = Q_0\, e^{-\alpha t} \tag{4.1}$$

on continuous decline segments above a minimum duration, returning the recession coefficient α and the corresponding half-life for each segment; and annual statistics computed on a hydrological-year basis (October to September), returning minimum, mean, maximum, and selected percentiles per year. These tools are intended as exploratory and first-level analytical support. They do not replace specialised hydrological or statistical software, but they cover the operations that monitoring researchers most frequently need.

Spatial tools operate across multiple points. The main implemented function is spatial interpolation. Given a variable and a timestamp, the platform collects the nearest observation from each relevant point, builds a regular grid covering the bounding box of the selected points, and interpolates values using SciPy's `griddata` with linear, nearest-neighbour, or cubic interpolation. The result is rendered on the map as a semi-transparent colour surface with a legend. This is useful as exploratory visualisation, but it is not equivalent to a full geostatistical analysis. Future work could add kriging, cross-validation of interpolated surfaces, and explicit uncertainty visualisation.

4.9 User interface and interaction model

The analytical capabilities described in the previous sections are exposed through a browser-based interface designed to be usable by researchers without specific GIS or programming experience. The interface follows three principles, keeping the visual structure simple, exposing the most frequent operations easily, and avoiding hiding the underlying parameters from users who want to understand them.

The main view is organised around an interactive map (see Figure 5.2 in Chapter 5). Monitoring points are displayed as markers, with a colour and shape that reflects the monitoring type (alpine spring, weather station, river gauge, water reservoir, water-quality station). A side panel lists the available monitoring types, each acting as a filter, and clicking a type hides or shows the corresponding markers. This filtering is particularly useful in mixed networks where only a subset of points is relevant to a specific analysis.

When a marker is selected, the detail panel opens with the metadata of the point, the most recent values for each variable, and a time-series chart. The chart includes a variable selector showing the available measurements for the point, and a time-window selector providing predefined ranges (last 7, 14, 30, or 90 days) as well as a custom range option. This combination lets the user switch quickly between different views without constructing complex queries.

4.9.1 Upload workflow

The data upload interface is offered in two complementary modes, Quick and Guided. The Quick mode is the standard path for files in a recognised format. The user selects the destination point, optionally specifies the variable, and uploads the file. The platform then detects the encoding, identifies the format, parses the data, and reports the result of the ingestion, including the format detected, the number of rows processed, the number of valid observations inserted, the number of duplicates skipped, and the number of missing values encountered. A simple progress indicator gives feedback during the upload itself.

The Guided mode is designed for files that do not match a recognised template. After upload, the platform shows a preview of the first lines of the file, highlights detected timestamp and numeric columns, and lets the user manually configure the parsing options, namely the number of header lines to skip, the column separator, the decimal symbol, and the column-to-variable mapping. This mode is slower for the user, but it makes it possible to import datasets that the automated parsers do not recognise, without modifying the platform code.

The two modes work together with the template-based pre-processing approach described in Section 4.6. Files that follow a template are imported through Quick mode, files prepared outside the platform but not yet fitted to a template can be imported through Guided mode, and provider-specific formats that are likely to be encountered repeatedly can eventually be wrapped into a new automatic parser.

4.9.2 Layer management

Spatial overlays are managed through a dedicated panel that lists the raster and vector layers available in the platform. Each layer can be toggled on or off, made more or less transparent, and reordered in the stacking sequence. Uploading a new layer is handled through a separate upload interface that accepts the supported file formats and triggers the appropriate publication path, with GeoServer for raster layers, PostGIS plus GeoServer for larger vector layers, and direct Leaflet GeoJSON rendering for smaller vector layers where interactive feature inspection is useful.

The layer panel also supports the connection of external WMS layers from remote servers. Given a service URL, the platform reads the WMS GetCapabilities response, lists the available layers, and lets the user add the chosen layer as an overlay. This is useful when a reliable dataset, such as a regional hydrography layer or a flood hazard map, is already published by an agency through a WMS. Instead of downloading and re-importing the data, the platform can use it directly through the service.

4.9.3 Map interactions and exploratory tools

Beyond the basic operations of zooming, panning, and switching the base map, the interface integrates the analytical tools described in Section 4.8. The user runs an analysis from the same panel that displays the data, which keeps the workflow continuous. The user selects a point, chooses a variable, defines a time window, applies a tool, and sees the result either as an updated chart or as a new overlay on the map. The spatial interpolation tool, for example, is launched from a dedicated panel where the user selects the point type, the variable, and the timestamp, and the resulting colour surface appears on the map with a legend and can be toggled like any other layer.

This integration is one of the practical reasons for choosing a custom backend rather than relying on standard map services alone. Many of the operations a monitoring researcher needs, such as running a recession analysis, filtering outliers, or applying a conversion curve, do not fit naturally into the WMS or WFS model. Exposing them through dedicated FastAPI endpoints keeps the user interface coherent, with the map as the common visual layer and the analytical tools reachable without switching to a separate application.

4.10 Containerisation and deployment

The whole platform is packaged as Docker containers coordinated by a single Docker Compose file. The same configuration can run on Windows, macOS, and Linux, which reduces the installation burden for future users. Containerisation is particularly important for open-source stacks because each component has its own dependencies and configuration, and Docker Compose makes the system easier to start, document, and reproduce.

The current deployment is local. Migration to a server at Politecnico di Torino would require additional production steps, among them HTTPS configuration, authentication, user roles, database backups, logging, and possibly a reverse proxy managed by the

institution. These aspects fall outside the current prototype, but they are important for any future operational deployment.

Chapter 5

Case Study: Alpine Springs in the Aosta Valley

To validate the platform, it was applied to a real dataset, the long-term monitoring of alpine springs in the Aosta Valley, carried out by the Applied Geology group at DIATI within the Reservaqua project [19]. The choice of this dataset is practical and methodological. It was available, it contains long time series from field sensors, and it is representative of the kind of environmental monitoring workflow that the platform is designed to support.

5.1 Study area and monitoring network

The Aosta Valley is the smallest Italian region, located in the north-western Alps, with elevations ranging from approximately 300 m to more than 4,800 m. Mountain areas are particularly relevant for climate-change studies because snowpack, precipitation, temperature, and groundwater dynamics strongly affect water-resource availability. Alpine springs are therefore important monitoring points, as they provide information on groundwater circulation, recharge processes, seasonal storage, and the response of hydrogeological systems to climatic variability.

The DIATI group has been monitoring a network of nine alpine springs since 2010–2011, recording variables such as water level, temperature, and electrical conductivity at hourly resolution [16, 20]. The springs span different geological and hydrochemical settings (Table 5.1), which makes the dataset suitable for testing a platform that needs to handle heterogeneous environmental observations.

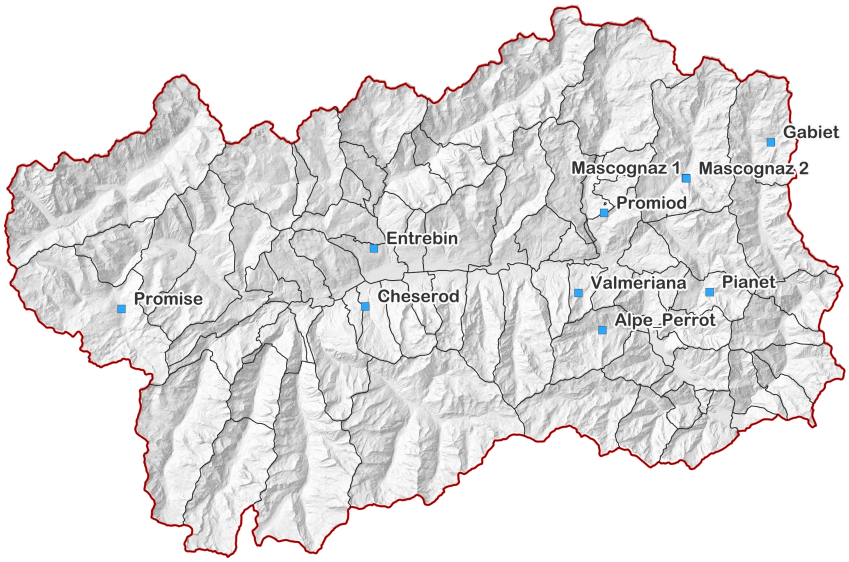

Figure 5.1. Location of the nine monitored springs in the Aosta Valley. Adapted from material supplied by the DIATI Applied Geology group.

Table 5.1. The nine monitored alpine springs used as test data.

Name	**Municipality**	**Elevation (m a.s.l.)**
Cheserod	Gressan	1,105
Entrebin	Aosta	981
Promise	La Thuile	1,580
Alpe Perrot	Champdepraz	1,280
Promiod	Châtillon	–
Mascognaz 1	Ayas	1,850
Mascognaz 2	Ayas	–
Gabiet	Gressoney-La-Trinité	2,340
Valmeriana	Pontey	1,698

5.2 Datasets

Three groups of datasets were integrated in the case study (Table 5.2), spring locations from a GeoPackage, spring time series from Hydras 3 datalogger exports, and regional meteorological data from the Centro Funzionale of Regione Valle d'Aosta [28].

Table 5.2. Datasets integrated in the case study.

Dataset	Source	Type	Coverage
Spring locations	DIATI (Reservaqua)	Vector (GeoPackage)	9 springs
Spring time series	DIATI (Hydras 3 export)	CSV	2010–present, hourly
Regional meteorology	Centro Funzionale VdA	CSV	2013–present, hourly

One issue surfaced during ingestion, the coordinate columns in the GeoPackage were labelled `X_WGS84` and `Y_WGS84`, but the numerical values corresponded to a projected coordinate system rather than geographic WGS84 coordinates. A conversion step was therefore required to produce correct longitude and latitude values for web mapping. This is a typical example of the difference between open data and ready-to-use data, even when the file is available, the structure and metadata must be verified before integration.

5.3 The previous monitoring workflow

Before turning to the validation, it is worth looking at how this monitoring data was handled before PoliTo EnviroHub, because that is the workflow the platform is meant to simplify. The process described here is the one followed by the Applied Geology group at DIATI for the Aosta Valley springs, but it is representative of how many small research groups manage field data.

The springs are not connected to the internet, so the process always starts with a field visit to download the records manually from the datalogger at each spring. Back in the office, the raw series is cleaned by hand, checking for gaps, outliers, and obvious sensor errors, and then imported into Hydras 3, the datalogger software. For every spring a rating formula has to be entered to convert the measured water level into discharge, and only then can the graphs for the different measured parameters be produced.

Weather data follows a parallel and equally manual path. The relevant series are downloaded from the Centro Funzionale portal of the Aosta Valley, which exports one CSV file per parameter per station. Three stations measuring precipitation, air temperature, humidity, and wind speed already means twelve separate files. If data from a neighbouring region such as Piedmont is also needed, the procedure changes again, since each agency publishes through its own portal and format. The weather series are then plotted in a different program from the one used for the springs.

The last step is presentation. To show results to the project stakeholders, the springs and weather stations have to be placed on a map, while the time-series graphs are prepared separately. Combining the two—showing a chart on the map, next to the station it belongs to—is difficult with the available tools, so in practice the map and the graphs stay two separate deliverables.

This workflow works, but it is fragmented. Several programs, manual file handling, repeated format-specific downloads, and a persistent gap between the spatial view and the temporal one. PoliTo EnviroHub is designed to fold these steps into a single environment. Spring and weather data are imported through the same interface; rating curves are stored once per point and applied automatically; the charts for every parameter are generated in the browser; and the stations and their time series share one map-based view, so a stakeholder can click a point and see its data immediately. Section **??** and Table 5.6 summarise this change.

5.4 Validation procedure

The case study was used to verify whether the platform satisfied the requirements defined in Chapter 4. The validation was not intended as a formal benchmark against commercial software. Instead, it focused on the practical questions that matter for a research group. Can the system be started from a clean environment; can monitoring points be loaded and displayed on a map; can long time series be imported without timeout errors; can users explore variables through charts; can contextual layers be added; and can basic processing tools be applied from the browser?

Table 5.3 summarises the validation criteria.

Table 5.3. Validation criteria used in the case study.

Criterion	**Expected result**
Deployment	Platform starts through Docker Compose without manual installation of each component
Point management	Spring locations are stored, georeferenced, and displayed as map markers
CSV ingestion	Long Hydras 3 monitoring files are parsed, validated, and bulk inserted
Time-series exploration	Imported observations are visible through interactive charts
Analysis tools	Resampling, trend analysis, and recession tools run from the interface
Spatial context	Raster and vector overlays can be added to the map
Extensibility	New monitoring types and variables can be defined without redesigning the database

5.5 Results

Setting up the platform took approximately half an hour from a clean Docker installation, including image download and container initialisation. Once the images were available, the platform started in roughly 30 seconds in the local development environment. All operations described below were performed through the browser.

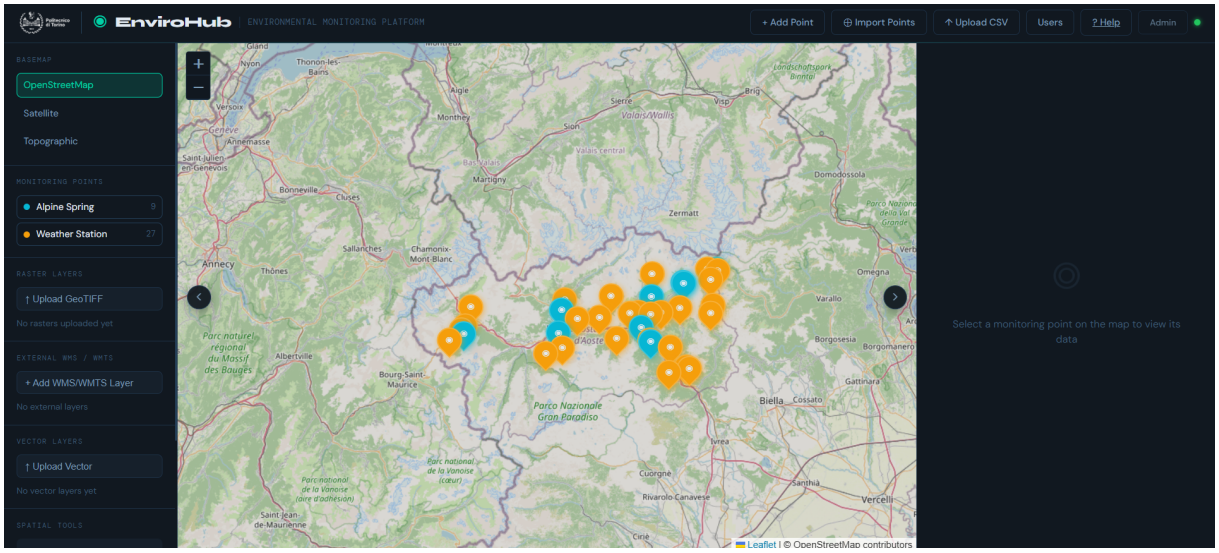

Figure 5.2. Main view of PoliTo EnviroHub with the nine springs visible as markers on an OpenStreetMap base layer.

Map and metadata. The nine springs were displayed as markers on an OpenStreetMap base layer (Figure 5.2). Clicking a marker opened a detail panel with name, location, monitoring type, and latest values. This confirmed that the platform could connect geographic location and monitoring metadata in the same interface.

Data ingestion. A Hydras 3 file from the Entrebin spring containing 105,116 rows of hourly water-level data was uploaded through the Quick mode (Figure 5.3). The platform detected the encoding, file structure, and monitored parameter, then inserted the valid observations using the bulk-loading procedure. Approximately 12% of the rows contained missing values and were skipped. This test was important because files of this size are common in long-term environmental monitoring and cannot be handled efficiently with row-by-row insertion.

Time-series exploration and analysis. The imported observations were visible as an interactive chart (Figure 5.4). The user could change the time window, select variables, and apply processing tools from the same panel. Resampling, trend analysis, and recession-curve fitting were applied successfully. The recession tool detected decline segments and fitted exponential models, producing a recession coefficient and half-life for each segment.

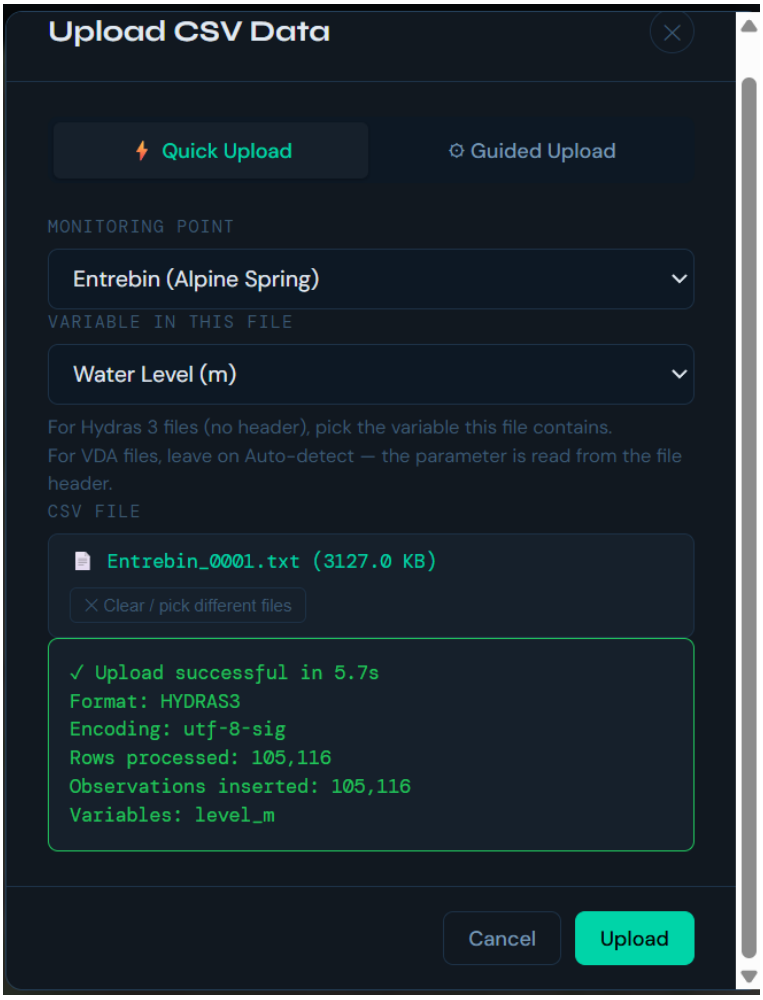

Figure 5.3. Quick upload of a datalogger file. The platform reports the encoding, format, parameter detected, rows processed, observations inserted, and missing values skipped.

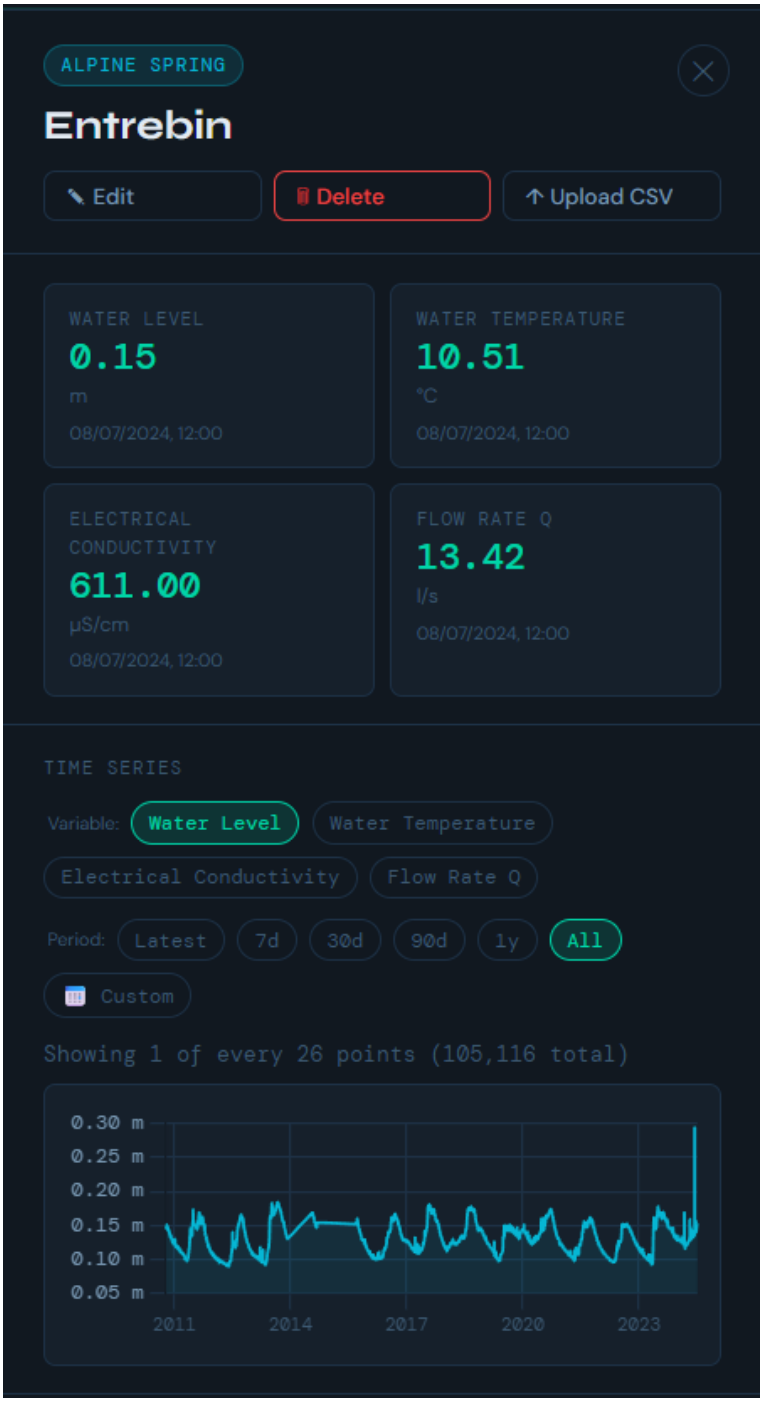

Figure 5.4. Interactive time-series chart for the Entrebin spring. The user can switch variables, change the time window, and apply processing tools from the same panel.

Spatial interpolation. With multiple meteorological stations loaded, a temperature surface was generated for a selected timestamp and rendered as a semi-transparent overlay on the map. The result demonstrated that the platform can move from point observations to a spatially continuous exploratory surface. However, the interpolation should be interpreted as visual analysis rather than as a validated geostatistical product.

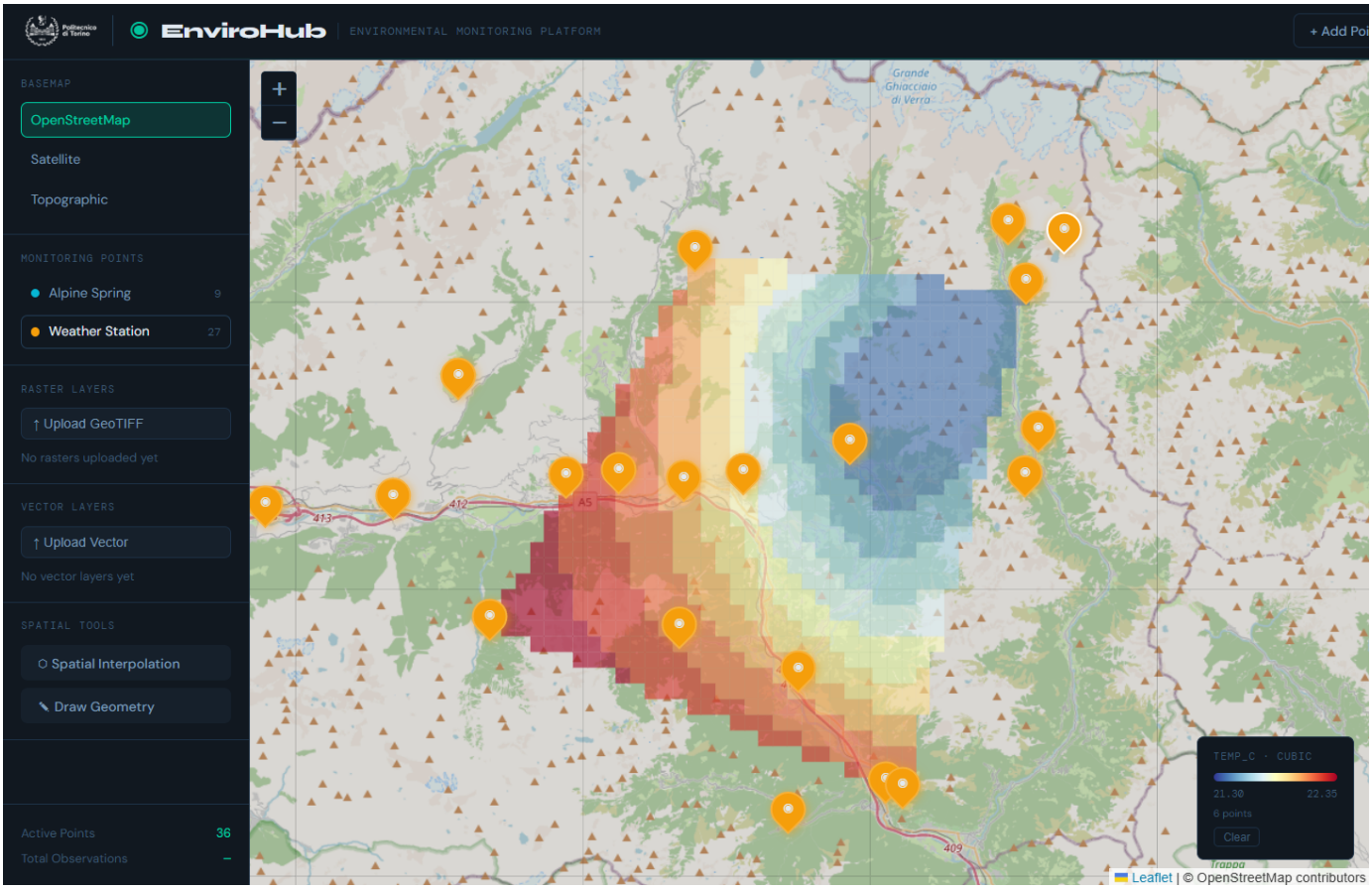

Figure 5.5. Example of spatial interpolation result displayed in the WebGIS interface.

Table 5.4 summarises the main practical results.

Table 5.4. Practical results from the case-study validation.

Aspect	Result
Deployment	Docker-based local deployment completed from a clean environment
Startup	Approximately 30 seconds after images were available
Monitoring points	9 alpine springs loaded and displayed on the map
Large CSV ingestion	105,116-row Hydras 3 file imported through bulk loading
Missing values	Approximately 12% of rows identified as missing and skipped
Temporal coverage	Multi-year hourly observations handled through the interface
Analysis tools	Resampling, trend analysis, and recession fitting executed successfully
Spatial layers	Raster and vector overlays supported through GeoServer and Leaflet

5.5.1 Detailed import statistics

Table 5.5 reports the import statistics for the Hydras files used in the validation. Each spring was associated with four datalogger files, covering more than ten years of hourly observations. The results show the scale of the dataset and the need for an ingestion procedure able to manage both large files and missing values.

Figure 5.6 shows that most springs have a completeness higher than 99.7%. Entrebin is the only clear exception, with almost 12.3% of the rows skipped during import. This difference confirms that the ingestion pipeline must be able to handle incomplete datalogger exports without requiring manual cleaning of the original files.

5.5.2 Example analytical outputs

The following examples show the type of analytical output obtained from the imported datasets. They are not intended as a complete hydrological study of the springs, but as a validation of the tools available in the platform.

Trend analysis. A linear trend was fitted to the daily averaged water-level series of Entrebin, from 18 October 2010 to 8 July 2024. The test used 4,385 daily values. The fitted slope was 1.29×10^{-6} m/day, corresponding to approximately 4.70×10^{-4} m/year. The coefficient of determination was low ($R^2 = 0.0075$), while the p-value was below 0.05 ($p \simeq 9.6 \times 10^{-9}$). The result therefore shows a statistically detectable trend, but also that the linear model explains only a very small part of the observed variability. In this

Table 5.5. Import statistics for the case-study spring Hydras files used in the validation. Each spring dataset consists of 4 files.

Spring	Total rows	Inserted	Skipped	Valid (%)	Temporal coverage
Entrebin	479,320	420,468	58,852	87.72	2010-10-18–2024-07-08, hourly
Cheserod	476,912	476,696	216	99.95	2010-12-02–2024-07-09, hourly
Promise	455,576	455,429	147	99.97	2011-07-12–2024-07-08, hourly
Promiod	481,736	481,594	142	99.97	2010-10-14–2024-07-10, hourly
Gabiet	445,568	444,447	1,121	99.75	2011-10-27–2024-07-11, hourly

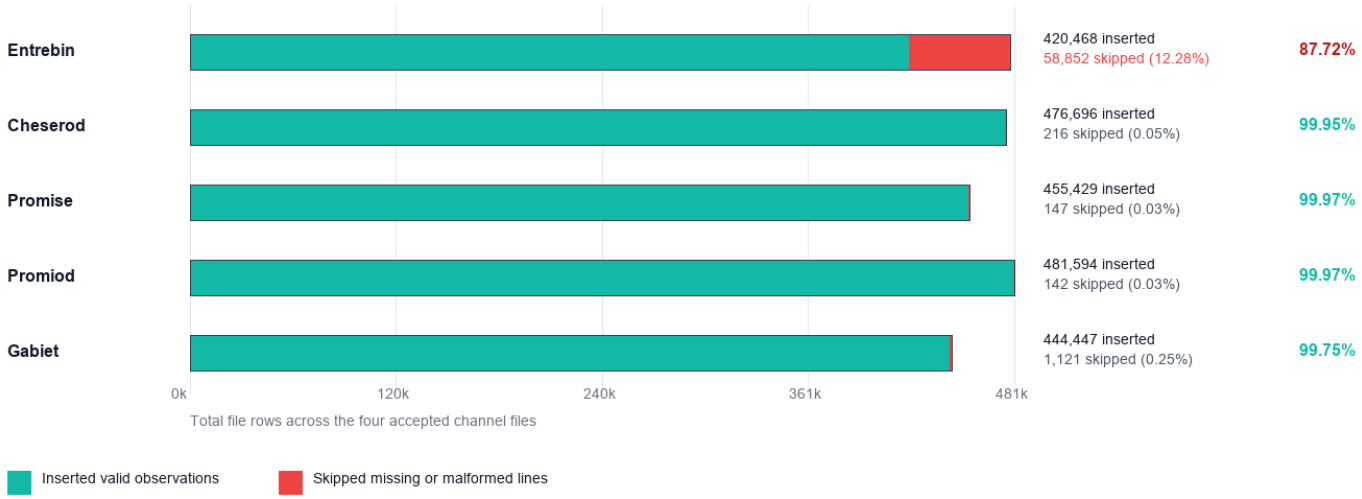

Figure 5.6. Completeness of the Hydras spring imports used for validation. Inserted observations are shown separately from missing or malformed lines skipped during ingestion.

case, the trend tool is useful as a first diagnostic rather than as a complete interpretation of the spring behaviour.

Rainfall and spring response. A rainfall–response comparison was also generated for a selected period, using precipitation data from the meteorological network and the corresponding spring time series. This type of plot is useful because it links the monitoring record to one of the main external forcing variables. It allows the user to visually inspect whether rainfall events are followed by changes in the spring response, and whether the reaction is immediate or delayed.

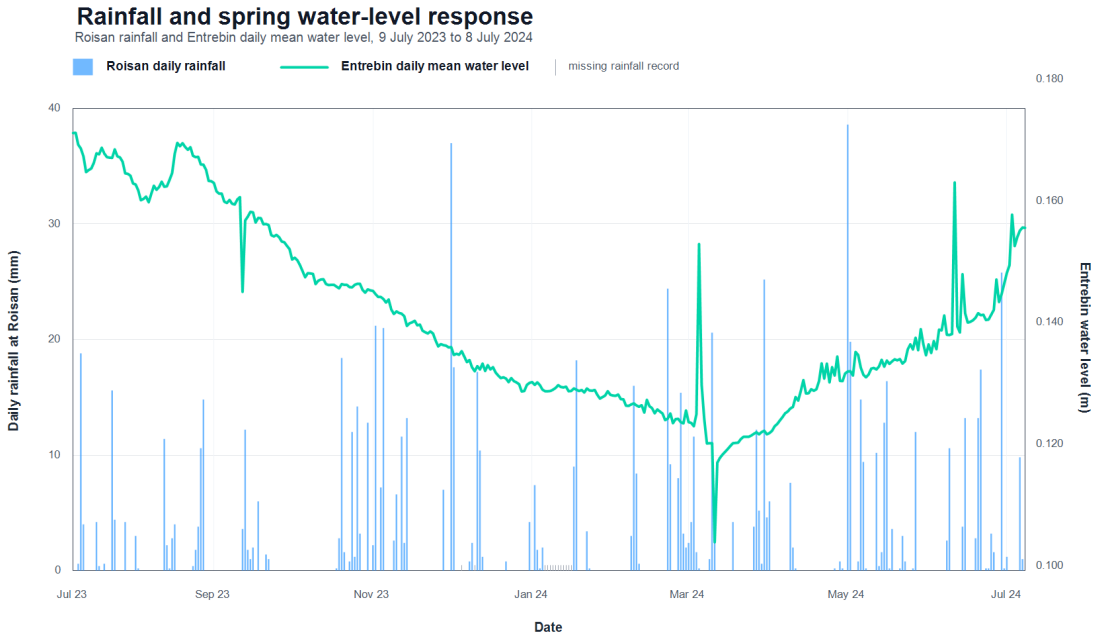

Figure 5.7. Example comparison between rainfall and spring response for a selected period in the case study. The figure shows how precipitation input can be compared with the temporal behaviour of the monitored spring.

Recession analysis. The recession-curve tool was tested on a continuous declining segment of the Entrebin discharge series between 30 October 2013 and 5 December 2013. Over this interval, the daily mean discharge decreased from approximately 13.49 L/s to 8.71 L/s. Fitting the exponential model $Q(t) = Q_0 e^{-\alpha t}$ gave a recession coefficient $\alpha = 0.01236$ d^{-1}, corresponding to a half-life of 56.1 days ($R^2 = 0.991$ for the log-linear fit). Figure 5.8 shows the selected recession segment together with the fitted curve. This example shows how the platform can extract recession indicators similar to those used in hydrogeological analyses of alpine springs [16].

Spatial interpolation. With multiple meteorological stations loaded, a temperature surface was generated for a selected timestamp (Figure 5.5). The result is rendered as a semi-transparent overlay on the map, with a colour legend connecting colours to values.

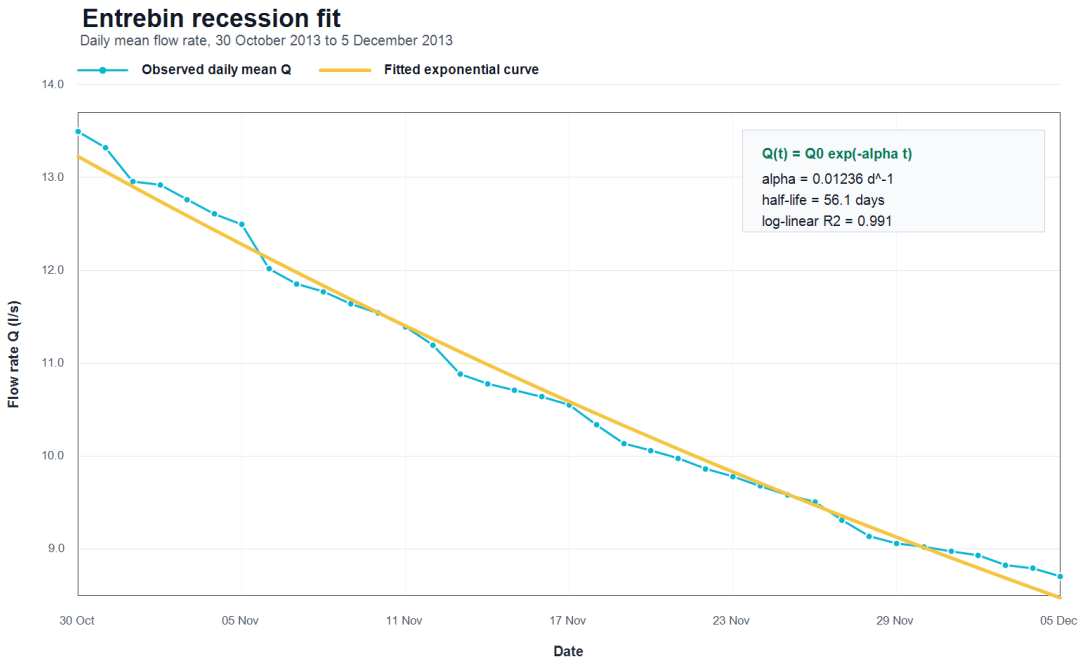

Figure 5.8. Example recession analysis for the selected Entrebin discharge segment between 30 October 2013 and 5 December 2013. The figure shows the observed recession behaviour and the fitted exponential curve used to estimate the recession coefficient and the corresponding half-life.

This output is useful for exploratory visualisation of the regional temperature field at a given instant. It should not, however, be interpreted as a full geostatistical product. The current prototype implements deterministic interpolation methods and does not estimate the uncertainty of the interpolated surface.

5.6 Comparison with the previous desktop-based workflow

The purpose of the comparison is not to claim that the WebGIS replaces all functions of specialised datalogger software. Desktop tools remain useful for instrument configuration and detailed local inspection. The comparison instead shows what changes when monitoring data are moved into a spatial, web-based, open-source environment.

Table 5.6. PoliTo EnviroHub compared with the previous desktop-based workflow.

	Previous desktop-based workflow	**PoliTo EnviroHub**
Access	Single user, local software	Multi-user potential, browser-based access
Geographic context	Limited or external to the monitoring tool	Monitoring points visible on a map
Format support	Focused on specific export formats	Multiple CSV structures plus extensible parsers
Time-series tools	Available in desktop environment	Integrated with map and web interface
Spatial layers	Managed separately in GIS software	Raster and vector overlays in the same platform
Sharing	File exchange or screenshots	Shared interface connected to central data
Licence dependency	Depends on the specific desktop software	No proprietary licence required for the implemented stack

The main advantage of the WebGIS is integration. The user does not have to move between a datalogger application, a spreadsheet, and a desktop GIS to obtain a basic spatial view of the monitoring network. The limitation is that advanced analysis and instrument-specific functions still belong in specialised tools. This confirms the complementary role of desktop GIS and WebGIS discussed in Chapter 2.

5.7 Lessons learned

The case study showed that the most difficult part of the workflow is often the import of the data, not their visualisation. Real monitoring files contain many small inconsistencies, like different encodings, unexpected column names, non-standard missing-value codes, uncertain coordinate reference systems, and formats that depend on the software used by the provider. For this reason, the ingestion pipeline had to be designed with checks and fallback options. Without a reliable import step, the rest of the platform would have limited practical use.

The tests also confirmed the importance of performance. Files with more than 100,000 rows are normal in long-term environmental monitoring. A simple row-by-row insertion strategy is easy to implement, but it quickly becomes inefficient and may cause timeouts. The use of PostgreSQL's `COPY` protocol for bulk loading was therefore one of the most important technical improvements of the implementation.

A final lesson concerns the level of automation that should be expected from a platform of this type. Environmental data are too heterogeneous to assume that every file can be imported without preparation. The objective should instead be to make the common workflow simple, while keeping the system open to new templates, parsers, variables, and analysis tools when new datasets are added.

Chapter 6

Discussion and Conclusions

6.1 Main contribution

The central hypothesis of this thesis was that a carefully designed open-source WebGIS architecture could reduce the gap between environmental data availability and practical usability, in particular for research groups running field monitoring networks. The work supports that hypothesis. PoliTo EnviroHub, the prototype built to test it, puts spatial data, monitoring time series, basic analysis tools, and web-based visualisation in a single open-source platform.

The contribution is the design approach as much as the software. The platform shows how a modular stack can be assembled from mature open-source components, each responsible for one part of the workflow. PostGIS stores spatial and temporal data; FastAPI handles monitoring-specific logic; GeoServer publishes standard geospatial layers; Leaflet provides the map interface; Chart.js displays time series; and Docker Compose makes deployment reproducible. This separation of responsibilities keeps the system understandable and extensible.

The Aosta Valley spring case study is where the hypothesis was actually tested. The platform loaded the monitoring points, imported long datalogger series, displayed the observations on a map, plotted their temporal patterns, and ran the processing tools through a browser, replacing a workflow that had relied on desktop tools, local files, and manual sharing. The evidence is a proof of concept rather than a general proof: it holds for one representative monitoring network, and confirming it for other groups and data types remains future work.

6.2 What the work shows about open-source WebGIS

Three findings stand out.

First, open-source geospatial tools are mature enough for research-oriented environmental monitoring. The components used here are widely adopted and well documented: PostGIS, GeoServer, GDAL, QGIS, Leaflet, and the Python scientific libraries. Together they offer a working alternative to proprietary stacks when cost, transparency, and reproducibility matter.

Second, the bottleneck is integration, not access. The datasets exist and the tools are capable; the hard part is turning a messy datalogger export into something that can be queried and mapped. The most time-consuming work was not the map display but encoding detection, format parsing, bulk loading, coordinate correction, and designing a data model flexible enough for different monitoring types.

Third, a WebGIS is most useful for shared access rather than specialist analysis. Desktop GIS stays essential for advanced processing, cartography, and expert workflows. A WebGIS becomes preferable when data has to be centralised, kept consistent, reached by several users, and explored by people who should not need to install or learn full GIS software. The platform complements desktop GIS rather than replacing it.

6.3 Evaluation, limitations, and future work

The case study suggests that the architecture suits a research-oriented environmental monitoring WebGIS, and it also marks the boundary between what the prototype already solves and what stays outside its current scope. Table 6.1 pairs each current result with its main limitation and the improvement that would address it, across accessibility, data ingestion, spatial context, analysis, reproducibility, and maintainability.

Table 6.1. Summary evaluation of the implemented approach.

Aspect	**Current result**	**Limitation**	**Future improvement**
Accessibility	Browser-based access to monitoring data	Tested locally, not yet as institutional service	Server deployment with HTTPS and authentication
Data ingestion	Robust CSV import for known/guided formats	No universal parser; no NetCDF support yet	Parser plugin system and xarray/NetCDF module
Spatial context	Raster and vector overlays supported	Advanced styling and cataloguing limited	Better layer manager and metadata catalogue
Analysis	Basic time-series and interpolation tools	No kriging, uncertainty, or domain-specific modelling	Advanced geostatistical and hydrological modules
Reproducibility	Docker Compose and open-source stack	No full automated test suite	Tests, CI pipeline, documented releases
Maintainability	Simple modular architecture	Long-term maintenance requires ownership	Institutional deployment and documentation

Two limitations are worth stating in their own right, because they are structural rather than a matter of adding one more feature. The first is that no platform can accept every incoming environmental file automatically: providers differ in structure, encoding, units, coordinate systems, and date conventions, so known-format parsers, guided column mapping, and extensible modules reduce the manual work but never remove it. The second is that the built-in analysis tools are meant for exploration, not for final results.

Interpolation uses standard SciPy methods without kriging or uncertainty estimation, and the recession and trend tools give useful first-level indicators that still need expert interpretation.

Beyond the improvements listed in Table 6.1, two directions would change what kind of tool the platform is. Real-time or near-real-time ingestion would turn it from a store of uploaded historical files into a live monitoring dashboard, which would need scheduled ingestion, validation rules, alert thresholds, and quality flags. Testing the same architecture in other domains, such as air-quality monitoring, river networks, urban climate adaptation, or soil moisture, would show how general it is and which parts need adapting. New analysis tools should be added carefully: piling on complex features without a clear user need would make the platform harder to maintain and work against its original goal of simplicity.

6.4 Final remarks

This thesis started from a practical problem: environmental data is more available than it used to be, but it is not always easy to use. In many monitoring projects the difficulty is not obtaining the data but organising it, linking it to its spatial context, and sharing it with others in a reproducible way.

PoliTo EnviroHub is one response to that problem. It shows how an open-source WebGIS can hold monitoring points, time-series observations, spatial layers, and basic analysis tools in a single browser-based environment. The Aosta Valley spring case study confirmed that the approach works with real datalogger files and long records, and it showed how much depends on robust ingestion and clear data structures.

The platform remains a prototype. It does not replace specialised GIS, statistical, or hydrological software, and it does not remove the need to prepare messy input files. Its value is narrower but concrete: a reproducible, extensible base that other groups can adapt to similar monitoring workflows without rebuilding it from scratch.

Bibliography

[1] GeoServer – open source server for sharing geospatial data. https://geoserver.org. Accessed: 2026.

[2] PostGIS – spatial and geographic objects for PostgreSQL. https://postgis.net. Accessed: 2026.

[3] V. Agafonkin. Leaflet – an open-source JavaScript library for interactive maps. https://leafletjs.com. Accessed: 2026.

[4] ARPA Piemonte. Geoportale ARPA Piemonte. https://geoportale.arpa.piemonte.it/. Accessed: 2026.

[5] S. Coetzee, I. Ivánová, H. Mitasova, and M.A. Brovelli. Open geospatial software and data: A review of the current state and a perspective into the future. *ISPRS International Journal of Geo-Information*, 9(2):90, 2020.

[6] Copernicus Climate Change Service. Climate data store. https://cds.climate.copernicus.eu. Accessed: 2025.

[7] Copernicus Emergency Management Service. European Flood Awareness System (EFAS). https://www.efas.eu. Accessed: 2025.

[8] Docker Inc. Docker – accelerated container application development. https://www.docker.com. Accessed: 2026.

[9] L. Duarte, A.C. Teodoro, and H. Guedes Soares. QGIS: a constantly growing free and open-source geospatial software contributing to scientific development. *Cuadernos de Investigación Geográfica*, 48(1):197–213, 2022.

[10] European Environment Agency. European air quality index. https://www.eea.europa.eu/en/analysis/maps-and-charts/index. Accessed: 2026.

[11] European Environment Agency. WISE freshwater. https://water.europa.eu/freshwater. Accessed: 2025.

[12] European Parliament and Council. Directive 2007/2/EC establishing an infrastructure for spatial information in the European Community (INSPIRE). https://inspire.ec.europa.eu, 2007. Accessed: 2025.

[13] European Union. Copernicus – Europe's eyes on Earth. https://www.copernicus.eu. Accessed: 2026.

[14] Federal Office for the Environment. Current hydrological monitoring data from the FOEN. https://www.bafu.admin.ch/en/view-current-water-monitoring-data. Accessed: 2026.

[15] GDAL/OGR contributors. GDAL/OGR geospatial data abstraction software library. https://gdal.org. Accessed: 2026.

[16] M. Gizzi, M. Mondani, G. Taddia, E. Suozzi, and S. Lo Russo. Aosta Valley mountain springs: A preliminary analysis for understanding variations in water resource availability under climate change. *Water*, 14(7):1004, 2022.

[17] Anita Graser, Tim Sutton, and Marco Bernasocchi. The QGIS project: Spatial without compromise. *Patterns*, 6(7):101265, 2025.

[18] H. Hersbach, B. Bell, P. Berrisford, S. Hirahara, A. Horányi, J. Muñoz-Sabater, J. Nicolas, C. Peubey, R. Radu, D. Schepers, A. Simmons, C. Soci, S. Abdalla, X. Abellan, G. Balsamo, P. Bechtold, G. Biavati, J. Bidlot, M. Bonavita, et al. The ERA5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020.

[19] Interreg Italia–Svizzera. Reservaqua – gestione integrata delle risorse idriche in alta quota. https://www.interreg-italiasvizzera.eu, 2019–2023. Accessed: 2026.

[20] M. Mondani, M. Gizzi, and G. Taddia. Role of snowpack-hydrometeorological sensors for hydrogeological system comprehension inside an Alpine closed-basin. *Sensors*, 22(19):7130, 2022.

[21] Open Geospatial Consortium. Web Feature Service (WFS) standard. https://www.ogc.org/standards/wfs/. Accessed: 2026.

[22] Open Geospatial Consortium. Web Map Service (WMS) standard. https://www.ogc.org/standards/wms/. Accessed: 2026.

[23] Open Geospatial Consortium. Web Map Tile Service (WMTS) standard. https://www.ogc.org/standards/wmts/. Accessed: 2026.

[24] OSGeo. Open source geospatial foundation. https://www.osgeo.org. Accessed: 2026.

[25] Publications Office of the European Union. data.europa.eu – the official portal for European data. https://data.europa.eu. Accessed: 2025.

[26] QGIS Development Team. QGIS geographic information system. https://qgis.org. Accessed: 2026.

[27] S. Ramírez. FastAPI. https://fastapi.tiangolo.com. Accessed: 2026.

[28] Regione Autonoma Valle d'Aosta. Centro funzionale – bollettini e dati meteorologici. https://cf.regione.vda.it. Accessed: 2026.

[29] S. Steiniger and A.J.S. Hunter. The 2012 free and open source GIS software map – A guide to facilitate research, development, and adoption. *Computers, Environment and Urban Systems*, 39:136–150, 2013.

[30] M.D. Wilkinson, M. Dumontier, I.J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.W. Boiten, L.B. da Silva Santos, P.E. Bourne, J. Bouwman, A.J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C.T. Evelo, R. Finkers, A. Gonzalez-Beltran, A.J.G. Gray, P. Groth, C. Goble, J.S. Grethe, J. Heringa, P.A.C. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S.J. Lusher, M.E. Martone, A. Mons, A.L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M.A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3:160018, 2016.