D2.2

Consolidated Geo-Database

CHIST-ERA-19-CES-005

SUSTAINABLE WATERSHED MANAGEMENT THROUGH INTERNET OF THINGS DRIVEN ARTIFICIAL INTELLIGENCE

chist-era

This work was supported by the CHIST-ERA grant.

- DOCUMENT RECORD

Deliverable Title	Consolidated Geo-Database
Corresponding Work Package	2.2
Related Tasks	2.3, 5.1, and 5.2
Dissemination Level	
Due Submission Date	15/01/2021
Actual Submission Date	18/02/2022
Responsible Partner	USI

Version	Date	Author	Comments
0.1	14/02/2022	Imoscopi, Riva, Lukovic	First draft
1.0	21/02/2022	Imoscopi. Riva, Lukovic	First submission

-

TABLE OF CONTENTS

1. Executive Summary

The goal of WP 2.2 is to design a unified data storage and exchange format that can work for both SWAIN use cases: river Ergene and river Kokemäenjoki.

River and weather data are usually available in heterogeneous formats, and it is difficult to build data analysis pipelines that work everywhere.
We aim at creating a modular, scalable and secure platform enabling data collection, storage and **easy access, in a unified and standardized manner**.

2. Data types

To store the data we use MongoDB. It was chosen for its popularity, flexibility and performance.
It is currently the most popular non-relational DB in the world, where everything is stored as simple and human-readable JSON documents. BSON is used to store the JSON data efficiently.

The data for our use case can be divided in two parts:

1) **Static GIS data:** mostly composed of geographical features, such as river network, basins, land use, geographical landmarks, weather stations.
   This data is composed of geo-referenced points, lines and polygons.
   Common file formats for **GIS vector data** are:
   - Shapefile, for legacy reason.
   - GeoPackage, more modern and efficient in terms of size and performance.
   - GeoJSON, the simplest one, widely used on the web and in many open-source GIS packages.

   We used GeoJSON for its simplicity, popularity and similarity with MongoDB documents, but it is straightforward to write functions to import/export also from/to other formats.

   In addition, we might need to handle also **GIS raster data**, like the ones used to store digital elevation models (DEM). As an example, we used EU-DEM v1.1 from Copernicus website, to get elevations for our regions of interest.
   The most common format for raster is GeoTIFF (geo-referenced TIFF image).
   Raster data can be very heavy in terms of memory consumption.

   We refer to GIS file formats for a more in depth discussion.

2) **Dynamic time series data:** weather data, river measurements, physical and conventional parameters, chemicals, micropollutants and other water quality indicators. All this are functions of time and space.
   Time-series data is usually stored and shared in tabular form, for example as CSV.
   In MongoDB, we can store it as documents in a TimeSeries collection. There are many ways to model the data, the simplest one is having one document per unique measurement, but we could also have one document per unique

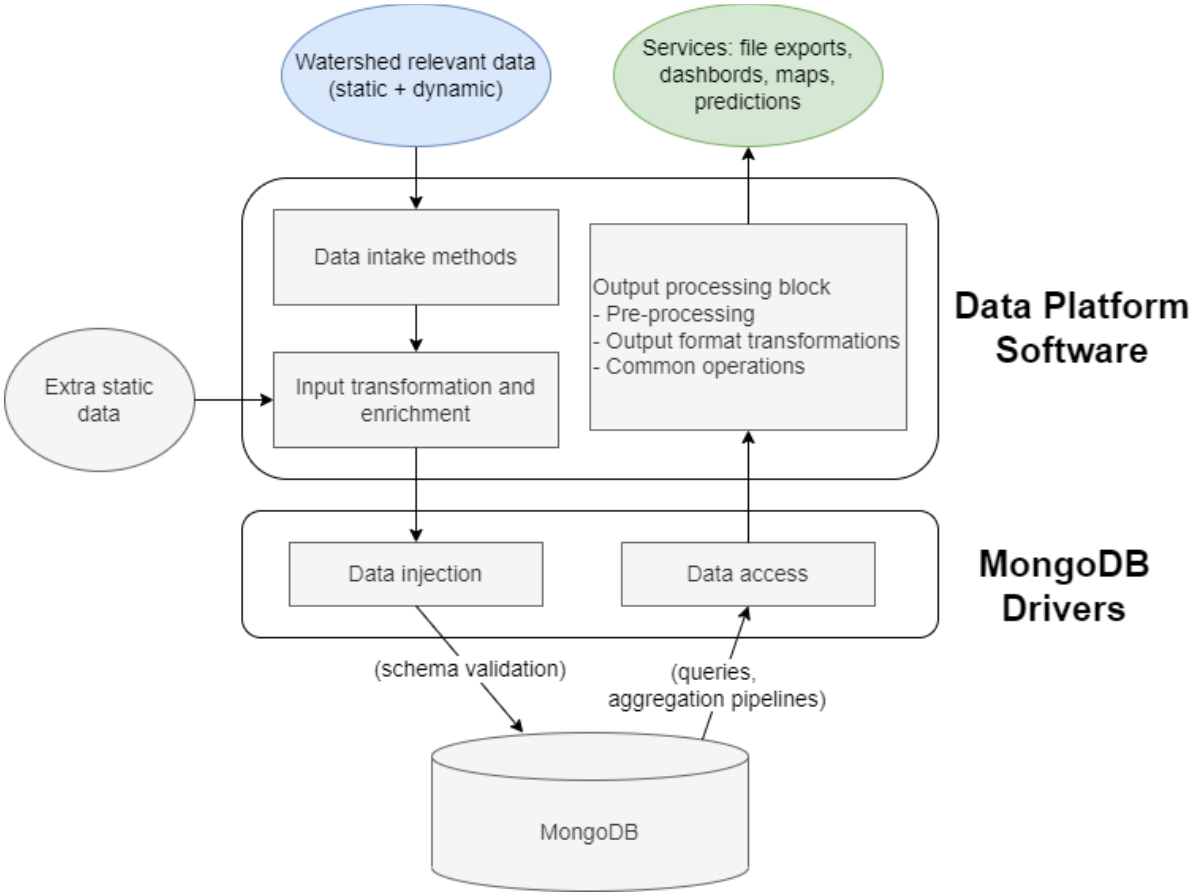
timestamp, containing multiple measurements of different type or from multiple locations.

We refer to the specifications in the code repository for the implementation details and the data modeling choices we make.

Static and dynamic data are stored in distinct collections, where the measurements link to the geospatial part (and to other metadata) via unique IDs.

3. ARCHITECTURE

The structure of the data platform is summarized graphically:



On the left, we follow the input data processing and injection in the DB.

- **Data intake methods:** to parse the static and dynamic data of the watersheds, in various formats.
- **Input transformation and enrichment:** to convert the data to MongoDB documents for storage.
  Data enrichment can follow, optionally using *extra static data*.

For example, assigning elevation in meters to the various geographical points, by using interpolation over the EU DEM dataset.

- **Data injection:** using a MongoDB driver to push the data to the DB. We choose to use PyMongo for Python.
  Optionally, schema validation can be enforced using JSON schema, to avoid storing data in the wrong JSON format.

On the right, we follow the data output through querying and retrieval.

- **Data access**: using MQL (Mongo Query Language) and aggregation pipelines to extract what we need from the MongoDB Collections. As a driver, we can still use the language we prefer, like python, but also MongoDB Compass, a GUI that allows quick graphical inspection and export of the dataset, without having to write code.
- **Output processing block**: the next output block can provide the most common operations needed to build applications on top of the DB, like data transformation, pre-processing for data analysis and output in various formats.

The green circle, at the top right, represents the users (e.g. hydrology and machine learning researchers), who will be able to build services on top and benefit from the consolidated geo-database.

In the scope of SWAIN, the code for this work is being developed at USI and the MongoDB will be hosted on a USI virtual machine. Proper access rights will be granted to all relevant SWAIN partners, making sure to keep the access secure, with authentication. In the future, the DB could be open sourced to a larger audience.

4. Conclusions

The open-source data platform under construction will help build analysis and tools that are applicable to different watersheds in a plug and play manner.

It might also be extended to other types of data that present one static (GIS) and one dynamic (time series) component.

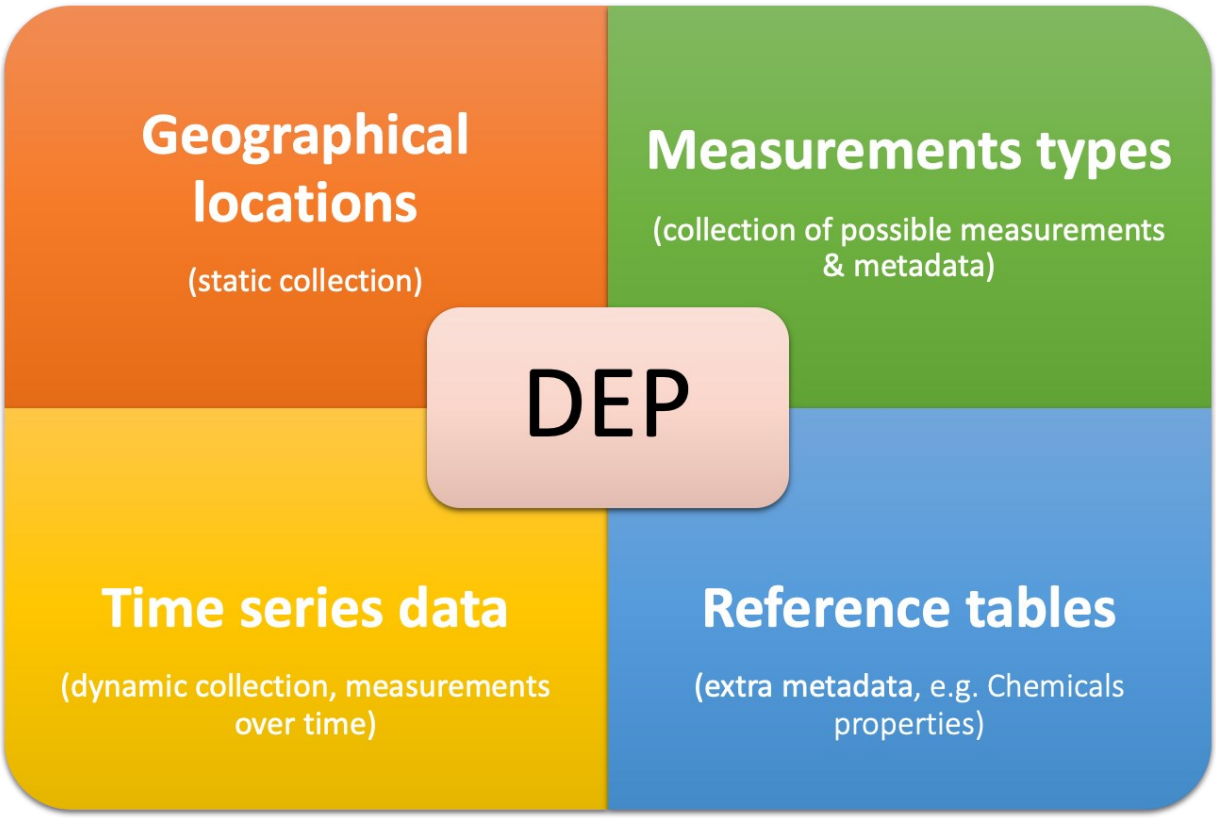
The only requirement would be to convert the data in the standardized way specified on the code repository, and add it to the DB.

Other hydrology datasets might be converted and added in the future, allowing to re-use the same tools. For example, the LamaH or CAMELS datasets, widely used in computational hydrology research.

In the future, other tools could be written to export the data from MongoDB in various standardized formats. This might be useful for legacy compatibility, file exchange and to adapt to other data analysis pipelines.

- ANNEX A
    - DATA EXCHANGE PROTOCOL

The exchange of data between SWAIN partners has been formalized with a Data Exchange Protocol (DEP), which consolidates a common and unique format. The DEP has been designed with the SWAIN data in mind, and is flexible enough to support the exchange of all types of data related to SWAIN. The origin of the data can be one of three sources: either directly measured at the gauges, catchments, and meteorological stations of the Kokemajoki or the Ergene river. The last source of data can be instead physical simulations.



The files formatted according to the DEP can be directly parsed into the unified MongoDB storage solution. The DEP is composed of 4 parts, one specifically devoted to one kind of data or other pieces of information. Geographical information and time series data make the core of the DEP, and are represented with two CSV files: *LOCATIONS_META* and *TS_DATA.*

The first of these files contains all the relevant geographical locations where some time-series has been recorded, e.g. meteorological station, catchment.

-
-
- ***A.1 LOCATIONS_META.csv***

Location_ID *(unique, int)*	**Loc_Name** *(string)*	**Loc_Type** *(string)*	**River_Name** (string)	**Longitude** *(decimal DEG)*	**Latitude** *(decimal DEG)*	**Altitude** *(m)*	**Info** (string)
1		WEA	Ergene	8.951052	46.003677	38.00	

a. Location Types

WEA	Weather station
FLW	Flow Rate station
WAL	Water levels location
CSD	Cross section depth location
CHE	Chemicals sampling location
MSC	Misc location, multivariate time series
SIM	Simulated location, multivariate time series

The TS_DATA is structured to contain all datapoints of each time-series. Each timestamp is also associated not only with a location, but also with a measurement type, which records what type of measurement has been taken.

Measurement types are unique, and are collected under a different CSV file called MEASUREMENT_TYPES_META.

- ***A.2 TS_DATA.csv***

MeasurementT_ID *(int)*	**Location_ID** *(unique, int)*	**Timestamp** *(timestamp)*	**Measurement** *(float)*
7	1	2020-06-01-12-30-35	3.0

- ***A.3 MEASUREMENT_TYPES_META.csv***

MeasurementT_ID *(unique, int)*	**Class** *(string)*	**Measurement_Type** *(string)*	**Unit** *(string)*	**Info** *(string)*
1	Weather	Rain accumulation	mm	