DELIVERABLE D4.3
Release of tools for improved data management

H2020-EU: **PIMCity**
Project No. 871370
Start date of project: 01/12/2019
Duration: 33 months

Revision:
Deliverable delivery: 28/02/2022
Deliverable due date: 28/02/2022

Document Information

Document Name: Deliverable 4.3 – Release of tools for improved data management

WP4 - Tools for improved data management
Author: TID and all WP4 Partners

Dissemination Level

Project co-funded by the EC within the H2020 Programme		
PU	Public	☑
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Contributions

	Name	Entity	Date
Author	Kleomenis Katevas	TID	28/02/2022
Author	Nicolas Kourtellis	TID	28/02/2022
Author	Panagiotis Papadopoulos	TID	28/02/2022
Author	Ioannis Arapakis	TID	28/02/2022
Author	Diego Perino	TID	28/02/2022
Author	Alvaro Garcia-Recuero	IMDEA	28/02/2022
Author	Roberto Gonzalez	NEC	28/02/2022
Author	Daniel Oñoro	NEC	28/02/2022
Author	Mathias Niepert	NEC	28/02/2022
Author	Bhushan Kotnis	NEC	28/02/2022
Author	Roberto Bifulco	NEC	28/02/2022
Author	David Friede	NEC	28/02/2022
Author	Vittorio Prodomo	NEC	28/02/2022
Author	Xavi Olivares	LSTECH	28/02/2022
Author	Dimitrios Delopoulos	LSTECH	28/02/2022
Author	Evangelos Kotsifakos	LSTECH	28/02/2022
Author	Daniel Fernandez	Wibson	28/02/2022
Author	Rodrigo Irarrazaval	Wibson	28/02/2022
WP Leader	Kleomenis Katevas	TID	28/02/2022
Coordinator	Marco Mellia	POLITO	28/02/2022

Document history

Revision	Date	Modification
Version 1	16/02/2022	V1

List of abbreviations and acronyms

Abbreviation	Meaning
G.A.	Grant Agreement
CA	Consortium Agreement
GA	General Assembly
PB	Project Board
PC	Project Coordinator
PrO	Project Office
IR	Interim Reports
PIMS	Personal Information Management Systems
DA	Data Aggregation
DPC	Data Portability Control
DP	Data Provenance
UPS	User Profiling System
QS	Quantified Self
P-DS	Personal Data Safe
D-TE	Data Trading Engine
DP	Data Provenance
DKE	Data Knowledge Extraction
PDK	PIMS Development Kit

Table of Contents

Executive Summary

Deliverable 4.3 consists of the final release of the tools devoted to improving data management. It delivers the final version of the software implementation of the relevant tools, such as the Data Aggregation (DA), the Data Portability Control (DPC), the Data Provenance (DP), the User Profiling System (UPS) and the Quantified Self (QS) dashboard. The source code is available on Gitlab, under the "PIMCity/Tools for improved data management (WP4)" path.

This document is part of the PIMCity Project, funded from the Horizon 2020 Program (ICT-13-2018-2019) under Grant Agreement number 871370. The technical decisions and design choices have been carefully discussed in plenary meetings as well as specific meetings of WP4 members. Based on these decisions, the partner responsible for the development of each tool has taken care of completing the design and initial implementation description which has been subject to review by other members of the WP4. The result of this process is presented in this document.

1. Data Aggregation

1.1. Introduction

The Data Aggregation (DA) tool enables data owners (for example an Internet Service Provider -ISP that hold a bulk of their users' data) to perform two important processes on their data: aggregation and anonymization. Such processes enable data owners to share these data in a privacy-preserving way. The DA tool resides on the data owner's side and its input is the raw data that is available through the initial sources (telco data, sensor data, etc.) and it is transformed in a predefined schema / metadata model. The user (data owner) is responsible for preparing the data for processing (i.e., export from their initial source (internal database), clean them if needed, etc.). Afterwards, through the module, the user is able to choose the subset of the data to be aggregated / anonymized and set the related algorithmic parameters (for aggregation and anonymization). The output is the processed (aggregated / anonymized) data that can be exported to the PIMCity marketplace through an API that the module provides. The data resides on the data owner side and the interested party is able to retrieve them through this API.

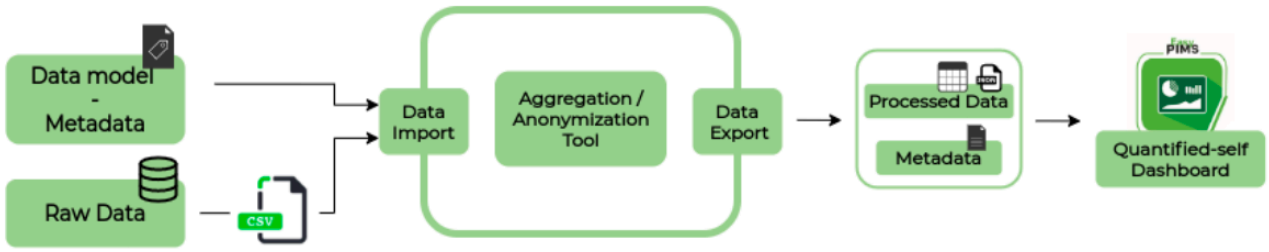


Figure 1. Data Aggregation (DA) architecture

1.2. Installation

Pre-requisites

The machine running Data aggregation API must have installed:

- Python 3.9.2 or superior
- Poetry 1.1.4 or superior
- Uvicorn 0.14.0 or superior

Deployment

First of all, copy the file or rename it from `app/app/.env-example` into `app/app/.env`.

Docker

A Dockerfile and a docker-compose file have been developed for an easier deployment.

In order to run it, execute:

```
docker-compose --env-file .\app\app\.env build
docker-compose --env-file .\app\app\.env up -d
```

As with the non-docker deployment, to access the application navigate to http://localhost:5000/.

Production deployment

When deploying in the production environment (https://easypims.pimcity-h2020.eu/dashboard/) simply pull from the branch "production" and follow the Docker deployment steps.

1.3. Usage Examples

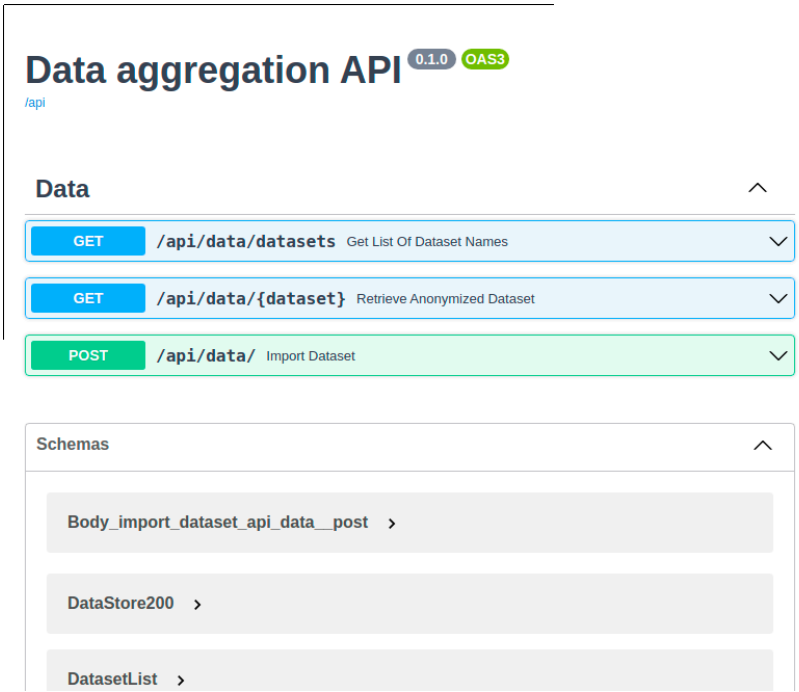
Get the list of anonymized datasets

In order to get a name list of anonymized datasets, execute this Http request:

```
GET http://localhost:5000/api/data/datasets
```

which will return an output similar to this:

```
{ "datasets": [ "Dataset A", "Dataset B", "Dataset C" ] }
```



[Figure X. DP architecture from previous deliverable]

Upload and anonymize dataset

In order to upload and anonymize a dataset, execute this Http request:

```
POST http://localhost:5000/api/data/add
```

The body request structure is formed of a Metadata Object, which has some additional information regarding the anonymized dataset and the CSV file. For more detailed information check the API documentation[1]. The request might delay a bit since it doesn't send a response until the dataset if totally anonymized.

Get the dataset anonymized

To retrieve the anonymized dataset, just execute this request:

```
GET http://localhost:5000/api/data/{dataset_name}
```

If the dataset exists, the response should be something like:

```
[ { "field_1": "test1", "sensitive_field": "****" ... }, ... ]
```

Documentation

In order to access the API docs navigate to: http://localhost:5000/docs

1.4. Changes

- Commit 851948a9: Merged master to production, added landing page.
- Commit df7f05: If the database is not healthy or has had any error during the container deployment, API won't start.
- Commit c8c56f: Added Hierarchy models and K-anonymity.

[1] https://gitlab.com/pimcity/wp4/data-aggregation-api#documentation

2. Data Portability Control

2.1. Introduction

The Data Portability Control (DPC) allows users to migrate their data to new platforms, in a privacy-preserving fashion. More specifically, it incorporates the necessary tools to import data from multiple platforms (through the available Data Sources), process the data to remove sensitive information (through the Data Transformation Engine), and outport into other platforms (through the Data Export module). The tool does not provide a dedicated UI to the users. Instead, it provides an interface in a form of a generic Control API for controlling all operations from other modules of the PIM system (e.g., the User Dashboard). The figure below depicts the DPC architecture.

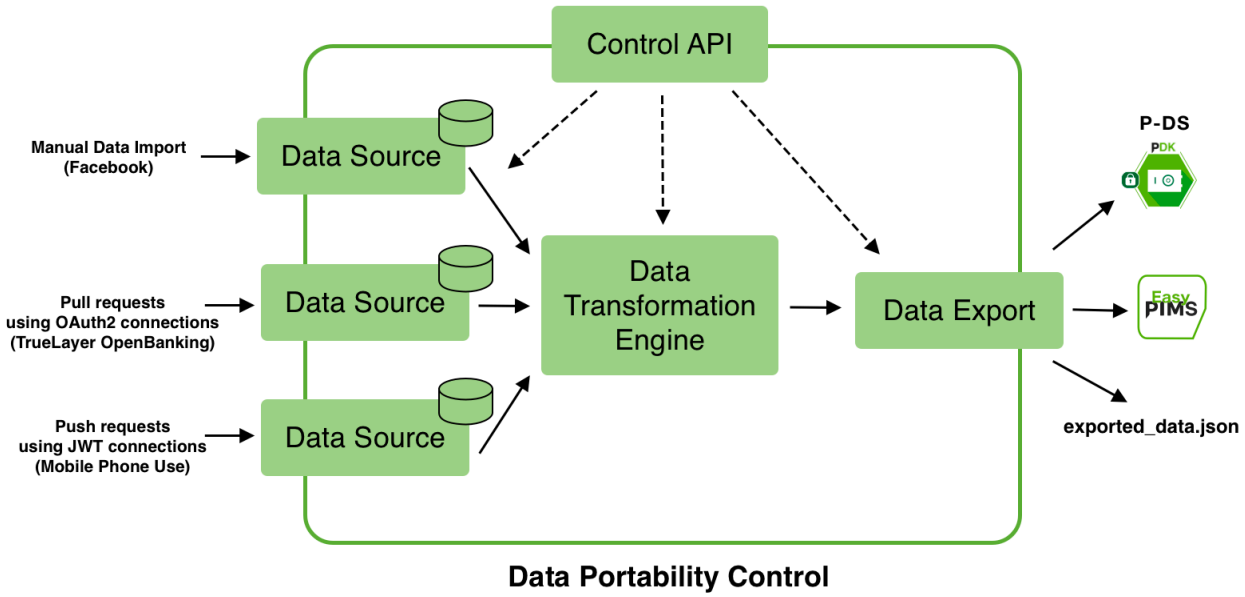


Figure 2. The Data Portability Control (DPC) architecture

2.2. Installation

The DPC tool is a Python 3 (minimum Python 3.7) system that requires the following libraries:

- flask
- connexion
- pymongo
- requests
- pyjwt
- python-keycloak

You can execute the following command to install these dependencies automatically:

```
$ pip install -r requirements.txt
```

An instance of MongoDB v4.4+ should also be installed and running.

2.3 Usage

To execute the DPC tool in development mode, please use the following command:

```
$ python app.py -dev
```

Below you can find a list of all available (optional) arguments:

```
usage: app.py [-h] [-dev] [-host HOST] [-port PORT] [-dbu DB-URL] [-cdb]

Data Portability Control (DPC) Tool operations.

optional arguments:
  -h, --help            show this help message and exit
  -dev, --development    Run DPC in development (debug) mode without KeyClock
authentication.
  -host HOST, --host HOST
                        Host that DPC will run (when in development mode).
  -port PORT, --port PORT
                        Port that DPC will run (when in development mode).
  -dbu DB-URL, --db-url DB-URL
                        MongoDB url that will be used.
  -cdb, --clean-db      Clean DB before running the tool.
```

The DPC tool does not have a dedicated UI for the users. Instead, it can be controlled using its Control API (a typical REST API) shown below:

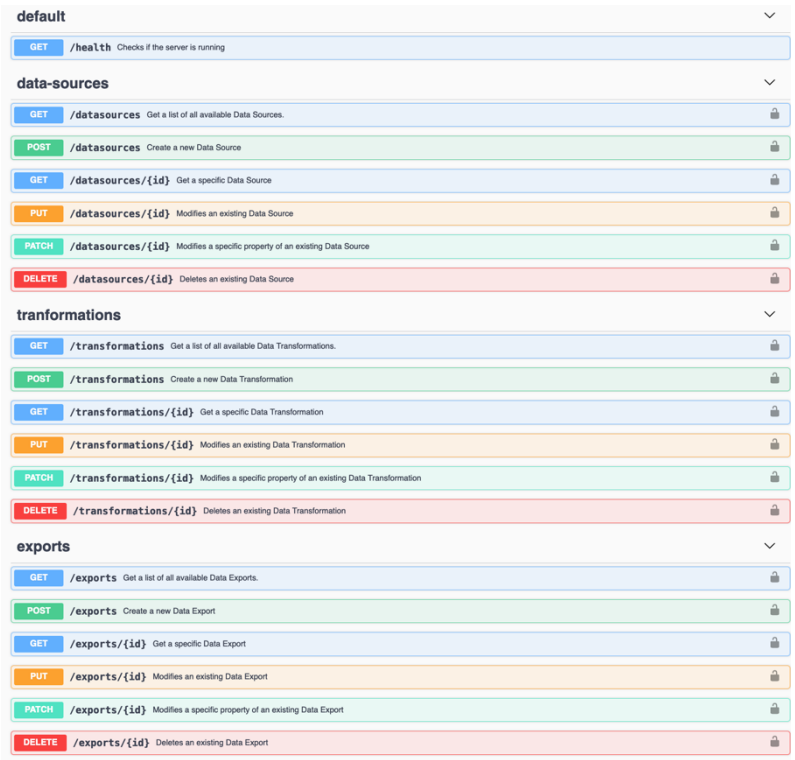


Figure 3. The DPC user interface

Assuming the DPC tool is installed and running, you can interact with its Control API for creating, configuring, and operating modules (i.e., Data Sources, Data Transformations and Data Exports). Swagger provides an interactive tool to experiment with the tool's endpoints and can be accessed from http://127.0.0.1:81/ui.

Install a Data Source

First read the list of all installed data sources, using the relevant GET request (should be empty initially):

```
$ curl "http://127.0.0.1:81/datasources/"
```

Make sure that a data source class is available under the datasources folder (e.g., datasource_truelayer).

Now, install an instance of this class, using the relevant POST request:

```
$ curl -X "POST" "http://127.0.0.1:81/datasources/" \      -H 'Content-Type: application/json; charset=utf-8' \     -d $'{  "class": "datasource-truelayer",     "name":  "Test",     "manifest-version":  1,     "type": "datasources"}'
```

You Data Source is now installed and run within the DPC too. To verify this, repeat the previous GET request to see its status:

```
$ curl "http://127.0.0.1:81/datasources/"
```

When installed, a module (in this case the Data Source) starts in a deactivated state (enabled is false). To enable it, send a PATCH request on the enabled property:

```
curl -X "PATCH" "http://127.0.0.1:81/datasources/<datasource-id>/" \     -H 'Content-Type: application/json; charset=utf-8' \     -d $'{  "enabled": true}'
```

Similar approach can be followed for installing a Data Transformation or a Data Export module. For more information, please refer to the Control API mentioned above.

2.4. Changes

v0.0.2 (31/01/2022)

- Replace flask-restful with connexion framework
- Add Authentication using KeyCloak
- Add implementation for EasyPIMS demonstration

v0.0.1 (10/06/2021)

- First public release of the DPC tool

3. Data Provenance

3.1. Introduction

The Data Provenance (DP) is a data management tool to watermark sensitive data as user web browsing history while accounting for user data ownership. It implements algorithms from the database watermarking literature (e.g., VLDB), aiming to bring new research into the area in order to use it in real world data management use cases as ours. We focus in web browsing data, namely URLs, which are a valuable piece of information about user's preferences and behavior, yet not monetizable by data owners in a decentralized manner in the real world yet (only centralized companies as Comscore exist for that). Therefore, out tool opens a new possibility to users to sell watermarked data with the support of the Trading Engine component (out of scope in this demo and intro) so that users just need to rely on REST-based APIs or a web interface to control their data ownership. Thanks to the REST API endpoints, the DP tool can be accessed also by other components of the PDK (https://easypims.pimcity-h2020.eu/intro-provenance.html). Internally, it uses the SpringBoot framework and will store user data on a secure PostgreSQL database as well as decentralized storage thanks to the support of IPFS (InterPlanetary File System) as middleware. Note, in the future watermarked datasets will be encrypted with the appropriate public and or private keys, but that is out of the scope for now. The Web interface is provided by the Swagger OpenAPI tools in our deployment as a single page application.

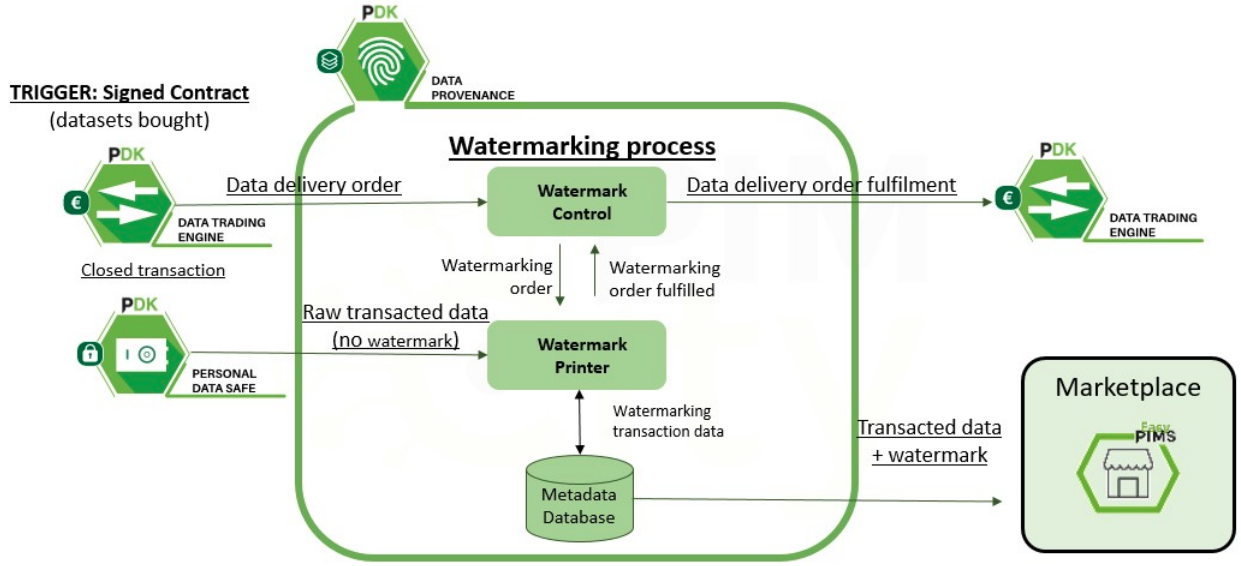


Figure 4. The Data Provenance (DP) architecture

3.2. Installation

The DP OpenAPI is written in Java and the [pom.xml] file in the project contains the list of libraries dependencies we require at the application level. For the API to run you also need the following system dependencies for some of such libraries.

Dependencies

1. You will need openjdk11 and both the JAVA_HOME AND JAVA_OPTS configured according to your system, in our case:
2. JAVA_OPTS = -DJava net.preferIPv4Stack=true -Djava.net.preferIPv6Addresses=false JAVA_HOME = /usr/lib/jvm/java-11-openjdk-amd64
3. Then, install maven, ipfsv0.4.16 and postgres at the very least.

4. Start ipfs:

```
ipfs daemon
Initializing
Daemon...
go-ipfs version: 0.4.23-
Repo version: 7
System version: amd64/darwin
Golang version: go1.13.7
```

5. The travis.yml file provices a self-contained setup of dependencies if you use CI/CD in a platform as github or gitlab.
6. An instance of Postgres12+ and a running service of IPFS and mandatory without IPFS. psql (PostgreSQL) 12.8 should also be installed and running at command line.

Note: If you run into garbage collection problems with IPFS, you may need:

- `ipfs pin ls --type recursive | cut -d' ' -f1 | xargs -n1 ipfs pin rm`

then optionally run storage garbage collection to actually remove things:

- `ipfs repo gc`

3.3. Usage

Starting the API on a server

- Configure [application.properties] accordingly fine and the OpenApiConfig.java file if you change server address in the config package.
- To compile the OpenAPI as a .jar run the following:
  `$ mvn spring-boot:run`
- Once it compiles correctly without errors the API is running locally as a local demonstrator, it should be also easy to test locally at: http://localhost:8090/swagger-ui.html

Using the API on the server

- Go the easypims Data Provenance site: https://easypims.pimcity-h2020.eu/intro-provenance.html
- Try out the online demonstrator: https://easypims.pimcity-h2020.eu/provenance/swagger-ui/index.html?url=/provenance/docs

Your main operating endpoints are:

- POST: /ipfs/dataset For creating new file with urls, one per file line)
- POST: /ipfs/dataset/{queryId} For sending the IPFS hash of a file already in IPFS to the watermarking algorithm.
- GET: /dp /wm /{datasetId} Gets a WM Dataset by IPFS hash (future use, instead of manual inspection using new IPFS hash).
- For a sample text file with urls:
  http://www.marca.es
  https://www.nhl.com
  https://ethsat.com

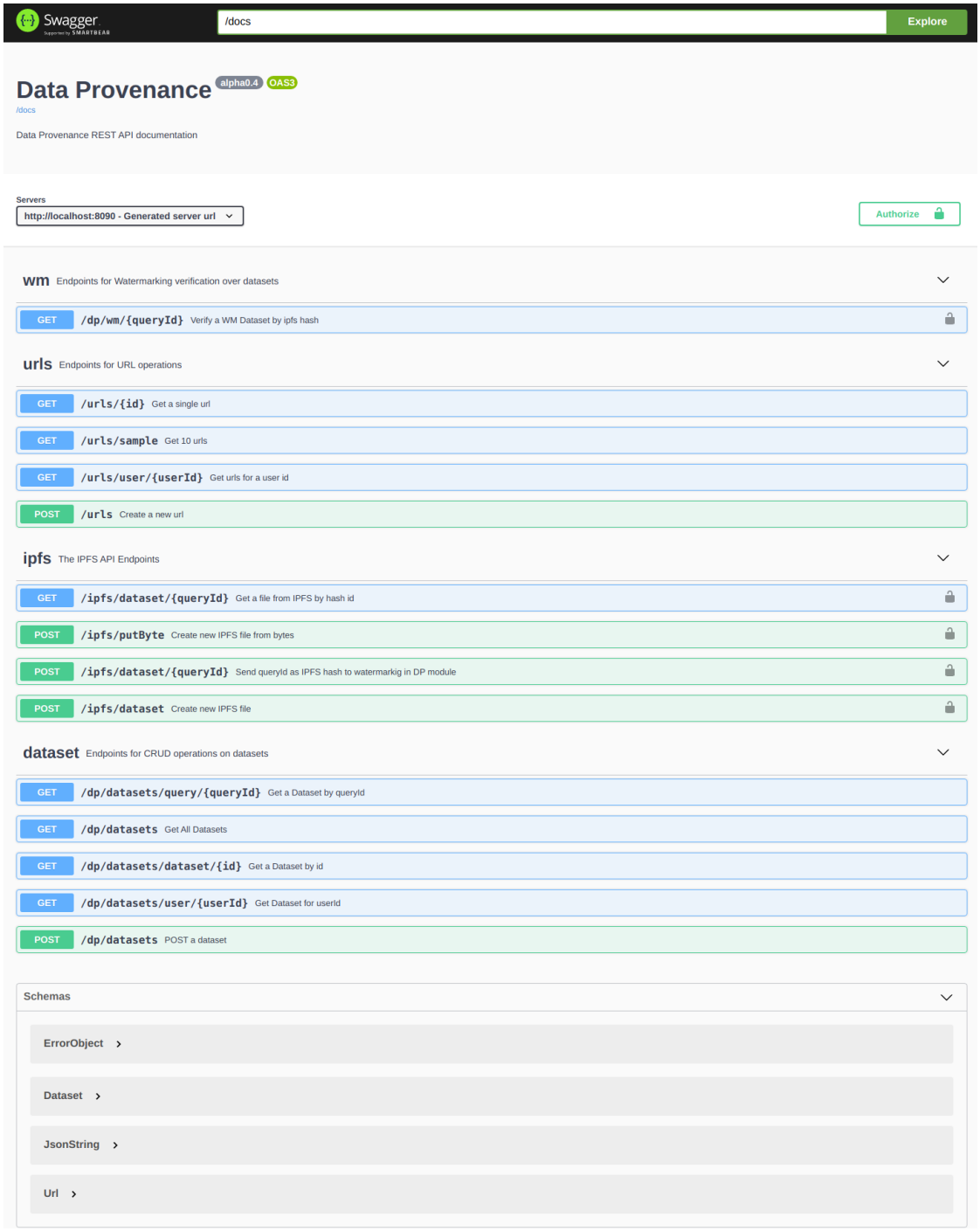


Figure 5. The DP user interface

3.4. Changes

v0.0.4-SNAPSHOT with added Authentication to involve Oauth2 in necessary endpoints of the system design above.

- Commit 7dd15d5164b09ea007f2476e0ab0b67893a25fd6: Upload json and yaml with security on. Signed-off-by: agr alvaro.garcia@imdea.org
- Commit f74d79f3e4e0325281f8196101799885ede44684: Complete security_auth with authCode. Signed-off-by: agr alvaro.garcia@imdea.org Mon Jan 10 15:22:43 2022 +0100
- Commit a8c5d4445713fac40da7265e0ecee6869b4fefee: Attempt to add SwaggerConfig for Oauth. Signed-off-by: agr alvaro.garcia@imdea.org Tue Nov 2 12:16:04 2021 +0100

- Commit 32dbec5bb75aa6865d18f56a9df3fba51a658fef: Attempt to add dependencies to pom. Signed-off-by: agr alvaro.garcia@imdea.org Mon Oct 25 10:29:26 2021 +0200

4. User Profiling System

4.1. Introduction

The User Profiling System is able to automatically generate a profile of the user, while considering their browsing patterns. The profile will indicate the interests of the user in each of the categories defined by the IAB.

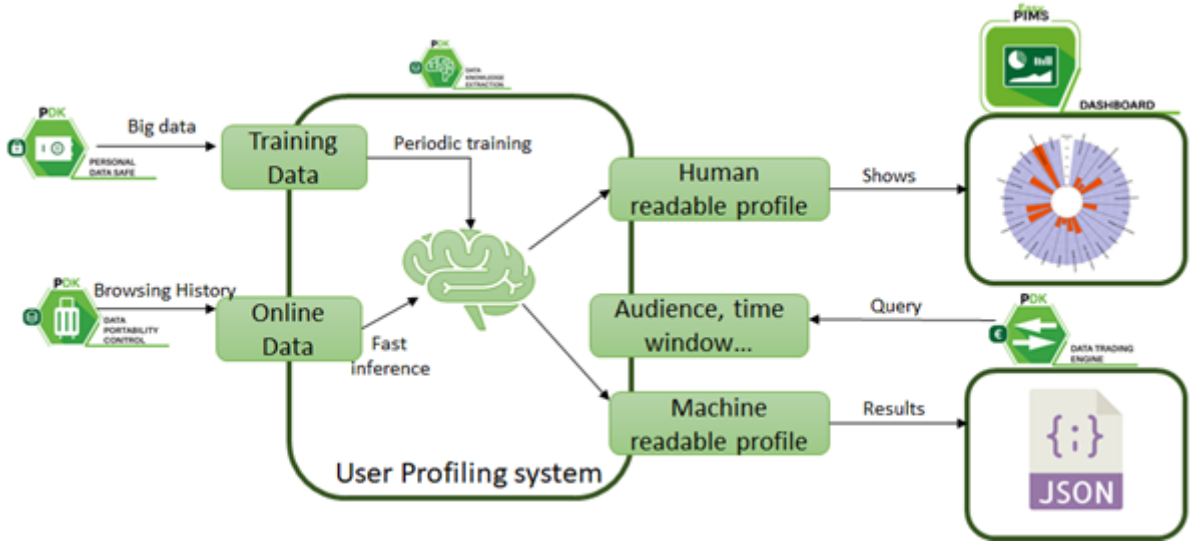


Figure 6. The User Profiling System (UPS) architecture

4.2. Installation

We recommend you use the python code under the folder "profilingAlgorithm". You can directly copy the folder and start using it. We also provide an REST server as reference on how you can integrate the User Profiling System into a platform like EasyPIMS.

Dependencies

- SQLAlchemy 1.4.1
- tornado 6.1
- tornado-sqlalchemy 0.7.0
- genism 3.8.3

4.3. Usage

The profiling algorithm uses the standard scikit-learn API.

```
def __init__(self, name, infer=True):
    """Return a SeqModel object whose name is *name* and which is either used
for training or inference."""
    self.name = name
    self.infer = infer

def load(self, path):
    """Loads a pretrained model from a file."""

def train(self, xdata, ydata):
    """Trains the model from scratch. Set params in separate method."""
```

```
def update(self, tseq, target):
    """Update the model (only possible if online SGD training possible)."""

def update_batch(self, xdata, ydata):
    """Update the model with a batch of sequences."""

def predict(self, seqData):
    """Apply the predict method (or equivalent) of the model and return
directly"""

def get_predict(self):
    """Return the value of the last predict call."""

def predict_probab(self, seqData):
    """Apply the predict_probab method (or equivalent) of the model and return
directly"""

def get_predict_probab(self):
    """Return the value of the last predict_probab call."""

def set_params(self, params):
    """Set the parameters of the internal machine learning model."""
```

Train the model

```
from profilingAlgorithm.models.SequenceModel import SeqModelGensimWord2Vec

myModel = SeqModelGensimWord2Vec("PreviousDay_model", infer=False)
#sequences should ve a list of lists. In each list we should have the list of
hosts visited by each user in order.
sequences = loadSequences()
myModel.train(sequences, None)
myModel.save("./models")
```

Inference

```
from profilingAlgorithm.models.SequenceModel import SeqModelGensimWord2Vec

myModel = SeqModelGensimWord2Vec("PreviousDay_model", infer=False)
myModel.load('./models')

# host_cats_vector is a dict that includes the assigned categories for the
known hostnames in the form of a vector.
# host_cats_vectors = {"google.com": [1, 0, 0, 0, 0, 0, 0],
#                      "bbc.co.uk": [1, 0, 0, 0, 1, 0.5, 0.5],
#                      "nyt.com": [0, 0, 0, 0.3, 0, 0, 0],
#                      "espn.com": [0, 1, 0, 0, 0, 0, 0],
#                      "realmadrid.com": [1, 0, 1, 0, 0, 0, 0],
#                      }
myModelset_domainToCategories(host_cats_vectors)

# listURLs is a list with the hostnames we want to obtain a profile for (order
is not important, repetition is not important.)
# normalizePrediction="Norm" returns a vector of categories normalized (it
sums 1)
# normalizePrediction="Max1" returns a vector of categories whith values
between 0 and 1 that is not normalized.
```

```
predict_probab_array = net2vec.predict_probab(listURLs,
normalizePrediction="Norm")
```

4.4. Changes

N/A

5. Quantified-Self Dashboard

5.1. Introduction

The purpose of the Quantified-self dashboard is to present to the user an analysis of their behavior based on the data available, especially related to their location. Through comparing their data with other users' aggregated data, the user will be able to better understand not only their behavior in relation to other similar users, but also to see the benefits of sharing data in the EasyPIMS setting and the value of his/her data. A simple, yet intuitive UI will guide the user to a set of basic actions regarding their profile, the data they have uploaded and the graphical representation of their behavior.

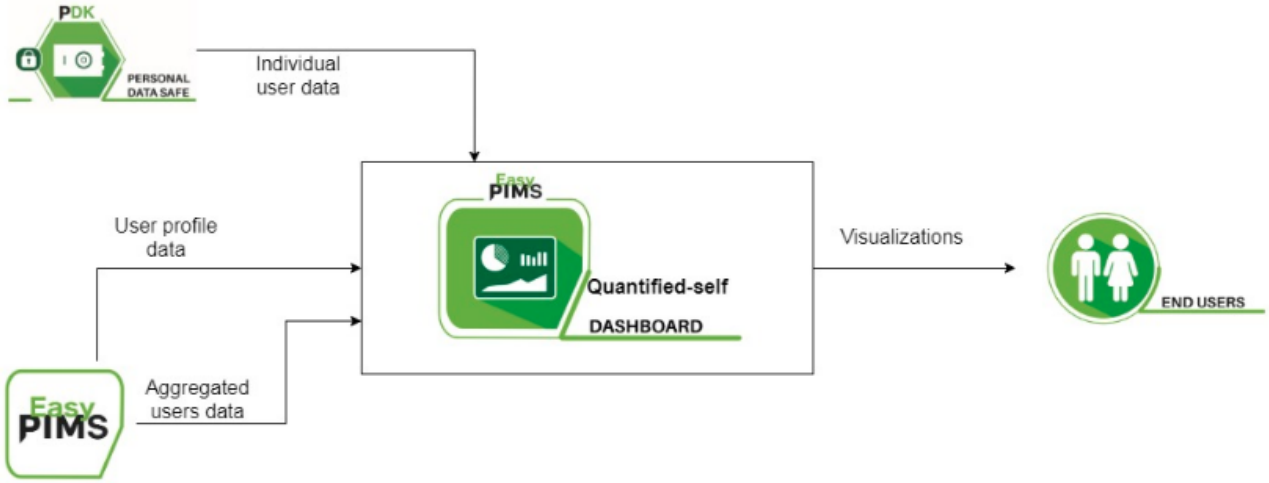


Figure 7. The Quantified-self dashboard (QSD) architecture

5.2. Installation

Pre-requisites

The machine running QSD must have installed:

- Angular version 11.0 or superior
- NodeJS version 15.4.0 or superior

Pre-requisites

Once all the prerequisites are fulfilled, run:

```
ng serve
```

Navigate to http://localhost:4200/. The app will automatically reload if you change any of the source files.

Docker deployment

A Dockerfile and a docker-compose file have been developed for an easier deployment.

In order to run it, execute:

```
docker-compose -f .\docker-compose-dev.yml build
docker-compose -f .\docker-compose-dev.yml up -d
```

As with the non-docker deployment, to access the application navigate to http://localhost:4200/.

5.3. Usage

Location patterns

In this page you can view the different location history of the user, having a more detailed view of every location.

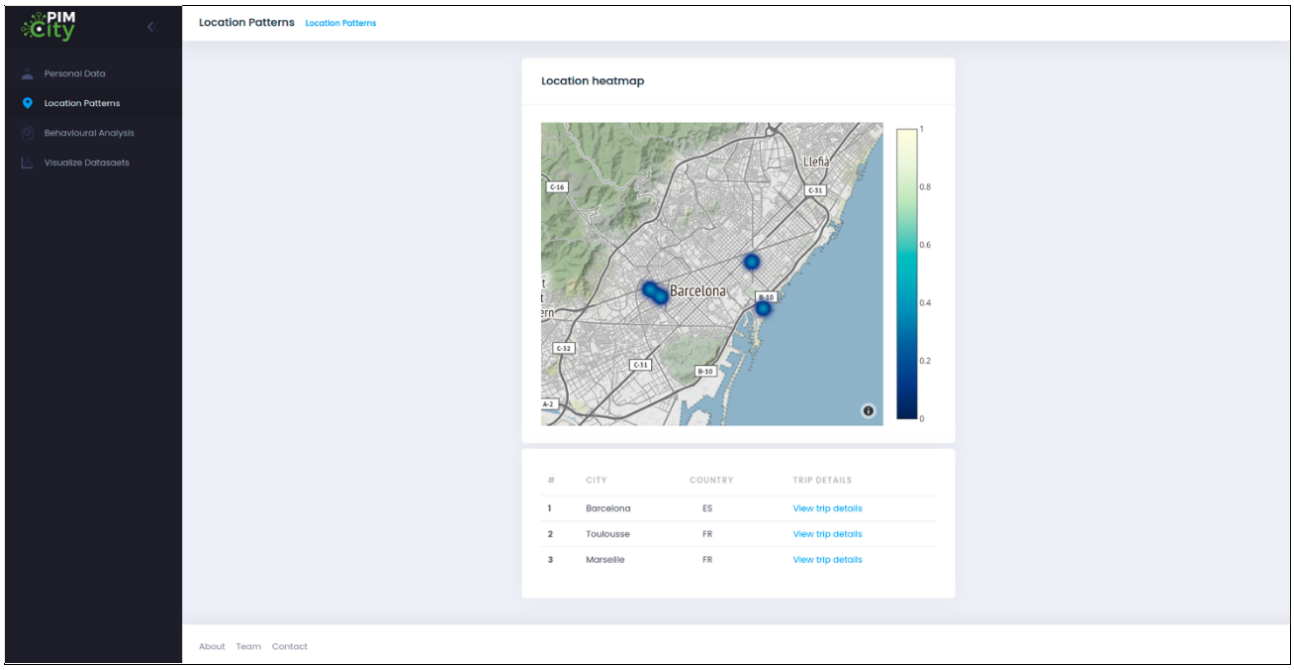


Figure 8. The Quantified-self dashboard user interface

Behavioral analysis

The view allows the user to check the daily activities and the time spent on them, it also has a calendar to filter by day.

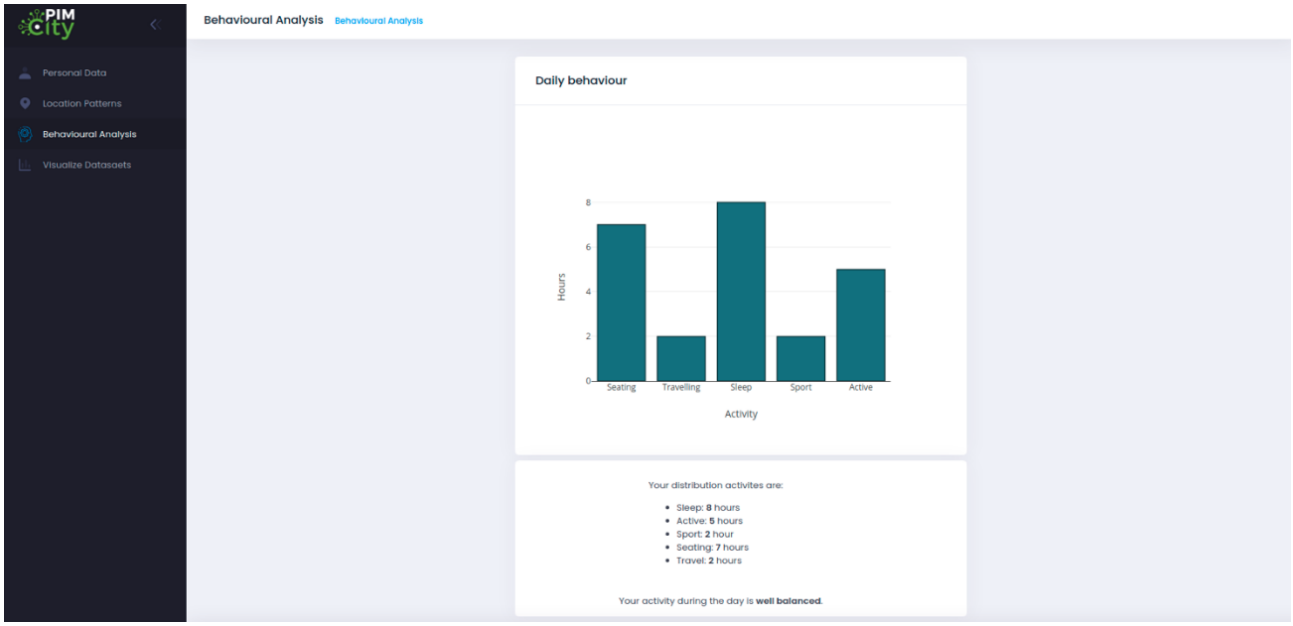


Figure 9. The Quantified-self dashboard user interface

5.4. Changes

Commit dcb4f093: Added calendar to Behavioral analysis page.

Commit 3ffbe24b: Added trip details component and updated "Location Patterns".
Commit 1c94845b: Nginx dockers: Dev and Prod.

6. Conclusions

This document discusses the final release of the tools devoted to improving data management. In particular, this deliverable describes the final version of the software implementation of the relevant tools, such as the Data Aggregation (DA), the Data Portability Control (DPC), the Data Provenance (DP), the User Profiling System (UPS) and the Quantified Self (QS) dashboard. The implication and development of these tools has been done collaboratively between the WP4 members, guaranteeing an easy integration. The different WP4 partners have carefully reviewed the document. During the testing and implementation process, some of the presented design decisions may be subject to revision and changes. The source code is available on Gitlab, under the "PIMCity/Tools for improved data management (WP4)" path.