"Building the Next Generation Personal Data Platforms"

G.A. n. 871370

Deliverable 2.2

Final design and preliminary version of the tools to improve user's privacy

H2020-EU-2.1.1: **PIMCity**

Project No. 871370

Start date of project: 01-12-2019

Duration: 33 months

Revision: 01

Deliverable delivery: 15/06/2021

Deliverable due date: 31/05/2021

Document Information

Document Name: Final design and preliminary version of the tools to improve user's privacy

WP2 Title: Tools to improve data subjects' privacy

Task 2.1, 2.2, 2.3, 2.4

Revision: 01

Revision Date: 15/05/2021

Authors: POLITO and all WP2 partners

Dissemination Level

Project co-funded by the EC within the H2020 Programme		
PU	Public	**X**
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)	

Approvals

	Name	Entity	Date
WP Leader	Martino Trevisan	POLITO	15/5/2021
Author	Luca Vassio	POLITO	15/5/2021
Author	Nikhil Jha	POLITO	15/5/2021
Author	Stefano Traverso	ERMES	15/5/2021
Author	Davide Pozza	ERMES	15/5/2021
Author	Rodrigo Irarrazaval	WIBSON	15/5/2021
Author	Daniel Fernandez	WIBSON	15/5/2021
Author	Javier Calvo	WIBSON	15/5/2021
Author	Roberto Gonzalez	NEC	15/5/2021

Author	Daniel Oñoro	NEC	15/5/2021
Author	Bhushan Kotnis	NEC	15/5/2021
Reviewer	Roberto Gonzalez	NEC	15/5/2021
Coordinator	Marco Mellia	POLITO	15/5/2021

Document history

Revision	Date	Modification
Version 1	15/05/2021	V1

List of abbreviations and acronyms

Abbreviation	Meaning
PIMS	Personal Information Management System
PDK	PIMS Development Kit
TT	Transparency Tags
P-DS	Personal-Data Safe
P-PM	Personal-Privacy Metrics
P-CM	Personal-Consent Manager
P-PPA	Personal-Privacy Preserving Analytics

Executive Summary

This deliverable describes the final design of the main components devoted to improving the user's privacy. Moreover, a preliminary version of the tools is delivered along with this deliverable. The source code of the PDK modules is available on GitLab, under the PIMCity/WP2 project[1], and we next provide the URLs for each repository.

This document describes the final design and preliminary implementation of the following PDK modules:

1. Personal Consent Manager (P-CM)
2. Personal Privacy Metrics (P-PM):
3. Personal Privacy Preserving Analytics (P-PPA):
4. Personal Data Safe (P-DS):

An overview on the PIMCity PDK can be found in deliverable D1.1. The initial design of the modules here described is available in Deliverable 2.1, that this document updates. More details on the design of other modules are available in deliverables D3.2 and 4.1. The final design and initial implementation of the other PDK modules are described in deliverables D3.3 and D4.2. The final implementation will be released in deliverable D2.3.

[1] https://gitlab.com/pimcity/

Index

1.- Introduction

Thanks to the introduction of regulatory frameworks focused on user's privacy such as EU's General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA), we are testifying the diffusion of new systems whose purpose is to help users in storing, understanding, and, possibly, monetizing their personal data in a transparent and easy way.

In this context, WP2 aims at building a set of software modules with the goal of enhancing the users' privacy. To this end, we design components to allow users store their data in a secure way. Second, users must be provided with tools to let them control and manage consent in a transparent way, i.e., decide who, how and when can access data. Third, we aim at creating tools for allowing queries on data that respect the users' consent choices and allow privacy-preserving data analyses. Finally, it is fundamental to present detailed information about the services willing to access data (i.e., data buyers) and the purposes of their business.

In the Work Package 2 of the PIMCity project, we target the goals described above and aim at designing and implementing modules that accomplish them. In this deliverable, we describe the final design, discuss how the modules satisfy the requirements defined in Deliverable 1.1 and motivate the technical choices behind four independent modules, that, together, build PIMCity's sub-system for enhancing data transparency, query anonymization and users' awareness and control. During this time, partners who contributed to this document have dedicated their efforts to collect feedback from all involved stakeholders, run experiments, simulations and tests which ultimately led to the final design of components presented in this document. Furthermore, new requirements have emerged during after the release of D1.1, and these have been accurately addressed in this revision. Along with this deliverable, we release the initial implementations of the Work Package 2 PDK modules. We defer the reader to Deliverable 1.1 for the full requirements for all modules. This deliverable *does not* cover module integration and the overall system design, that is covered in WP1 and WP5 deliverables. The aim of the WP is to investigate tools to improve end users' privacy, focusing on the following aspects:

- Design and develop a system able to empower the users to control their consent settings in multiple account and services. It should have an easy-to-use interface and provide auto configuration options to make it easier for the users to configure complex scenarios by using aggregated/crowdsourced data of the different users to build a set of common profiles.
- Design privacy metrics to unveil and communicate end users the data collected by online services, automatically identify and pinpoint possible privacy violations in data collection, communicate these findings to the end users in an easy and intuitive user interface.
- Develop a set of general-purpose building blocks to analyze users' data without affecting their privacy. It will offer some algorithms and methodologies able to provide a certain level of anonymity using concepts as or k-anonymity and differential privacy.

2.- Deliverable Objective

The objective of **Deliverable 2.2** is to provide the final design of the main components devoted to improving the users' privacy, as well as presenting the initial implementation of these modules. These tools are part of the **PIMCity PIMS Development Kit (PDK),** basic and generic components that offer fundamental functionalities for PIMS. These modules aim at empowering the users to control how their data is stored, processed, shared, and for which purposes. In this deliverable, we describe the overall and final design of these modules, providing the overall functioning architecture, motivating the technical choices and describing how they are accessed by users or other system modules (i.e., the interfaces). Moreover, we describe in details the module APIs and provide instructions for configuring and using the PDK modules. Here, we do not describe in detail how these modules integrate together or the overall PIMCity ecosystem, being these topics out of the scope of the deliverable.

The Work Package 2 designs four PDK modules, which take care of implementing those functionalities centered on users in terms of data storage and sharing, consent handling and privacy risks presentation. The modules described in this deliverable are:

1. **Personal Consent Manager (P-CM):** The consent manager is the means to define all the user's privacy preferences. It defines which data a service is allowed to collect, process, or which can be shared with third parties by managing explicit consent. Users' settings are imposed on all participating systems. The P-CM is described in Section 3 and is available online as an open-source project at: https://gitlab.com/pimcity/wp2/personal-consent-manager
2. **Personal Privacy Metrics (P-PM):** They have the goal of increasing the user's awareness. They collect, compute and share easy to understand novel privacy metrics, indicating e.g., which information the system is collecting, how it stores and manages the data, if it shares it with third parties. This module is described in Section 4 and is available online as an open-source project at: https://gitlab.com/pimcity/wp2/privacy-metrics
3. **Personal Privacy Preserving Analytics (P-PPA):** This module has the goal of allowing data analysts and stakeholders to retrieve useful information from the data, while preserving the privacy of the users whose data are in the studied datasets. It leverages concepts like Differential Privacy and K-Anonymity so that data can be exchanged among different systems while preserving the actual information as private. It is described in Section 5 and is available online as an open-source project at: https://gitlab.com/pimcity/wp2/personal-privacy-preserving-analytics
4. **Personal Data Safe (P-DS):** It is the means to store personal data in a controlled form. It implements a secure repository for the user's personal information like navigation history, contacts, preferences, personal information, etc. It is described in Section 6 and is available online as an open-source project at: https://gitlab.com/pimcity/wp2/personal-data-safe

The objectives of this deliverable partially address the following objectives of WP2 described in the Grant Agreement:

- *Design and develop a system able to empower the users to control their consent settings in multiple account and services. It should have an easy-to-use interface and provide auto configuration options to make it easier for the users to configure*

complex scenarios by using aggregated/crowdsourced data of the different users to build a set of common profiles.

- *Design the Privacy Metrics to i) unveil and communicate end users the data collected by online services, ii) automatically identify and pinpoint possible privacy violations in data collection, iii) communicate these findings to the end users in an easy and intuitive user interface.*
- *Develop a set of general-purpose building blocks to analyze users' data without affecting their privacy. It will offer some algorithms and methodologies able to provide a certain level of anonymity using concepts as zero-knowledge proof or k-anonymity.*

We use as reference the Deliverable D1.1, in which we precisely defined the requirements of the modules included in the Work Package 2 whose design is here described. As such, we defer the reader to the Deliverable 1.1 for a broader description of the PIMCity project. This deliverable updates Deliverable 2.1, integrating it with the final design of the modules and providing the initial implementations of them. Moreover, details on the other PDK modules can be found in Deliverables 3.3 and 4.2.

Finally, we remark that further adjustment to the designs presented in this document might occur during the remaining execution of the project. However, we will avoid to revolutionize the solutions presented in this document, and possible changes will be described in future deliverables, or directly in components' code repositories.

3.- Personal Consent Manager (P-CM)

3.1.- Objective

In this section we introduce PIMCity's Personal Consent Manager, also known as P-CM.

Its primary objective is to give the users the transparency and control over their data in a GDPR compliant way. That is, give them the possibility to decide which data can be uploaded and stored in the platform, as well as how (raw, extracted or aggregated) data can be shared with Data Buyers in exchange for value when the opportunity arises.

The P-CM is presented as a web application and a REST API, not only providing users the possibility to use the component in a user-friendly way, but also enabling developers to integrate PIMCity Consent Management capabilities in their products.

Consent Management over the Internet is still brand new as regulations define new rights and responsibilities. Therefore, there is no standard way of gathering and providing consents to store and use data. With PIMCity's P-CM, we aim at:

- Defining a standard interface to store, update, delete, audit and activate consents received for multiple purposes, such as storing data and sharing it with third parties.
- Providing a web application and a REST API as an implementation of such standard interface.
- Implementing state-of-the-art techniques, using standard community-approved software libraries.
- Putting in place an integration flow that eases the deploy and allows for quick upgrades to the tool in the production environment.
- Making the component efficient, fast and scalable to accommodate the consents of a very large set of users.

3.2.- Background and state of the art

The General Data Protection Regulation (GDPR) [1] was made compulsory in May 2018. At that time, and still now, a huge number of companies are not fully prepared to comply with it. Even, some of them are widely apart from being compliant in the short term[2]. The regulation challenges all organizations on different areas such as the administrative, legal and technical ones. More specifically, they are challenged in the ability to keep data secure and updated at high scales, or even automate data protection processes. Not complying to the rules is not an option, since fines can reach 20 million euros or 4% of the company's total annual global revenues.

Moreover, the value of the data market has grown significantly in the last seven years, e.g., it grew by 9% just in 2017 and it is expected to get over 60 billion Euros after 2020. Nevertheless, the difficulties faced by the companies to comply with GDPR makes it utterly hard to keep monetizing personal data, or even data in general, since there is no clear way for other businesses to verify the conditions under which the data was collected in the first place, exposing themselves to huge fines for using or acquiring data without its proper

[2] https://www.technologymagazine.com/data-and-data-analytics/capgemini-85-companies-are-not-ready-gdpr

consent. This makes businesses increase the lack of trust for organizations that sell (or re-sell) data, preventing the data market from growing.

To tackle these problems and seize the opportunity of establishing a standard way of collecting consents along with data, several state-of-the-art solutions came to existence.

Consent Manager by *PrivacyCloud*

The PrivacyCloud Consent Manager Chrome extension[3] removes most cookie banners and declines consent by omission. It will decline all cookies on your behalf by automatically removing most consent-gathering notifications on the fly. Since no positive action will have taken place (unless you "pause" Consent Manager), you will have said "no to all" while enjoying a smooth browsing experience.

They also notify the user by flagging any company or website that is not complying with the user's revoked consent or even the ones serving cookies by default as the user is browsing. The extension overlays the number of cookies being served illegally on top of the Consent Manager icon.

The product has more than three thousand active users.

Matomo Analytics

Matomo Analytics provides an analytics platform[4] as an alternative to Google Analytics, but focused on respecting users' privacy. That is, they enable consent gathering and maintenance out-of-the-box and they keep all the data management in compliance with GDPR (or even stricter cases like the French CNIL regulation). They also claim that they don't use the data their customers collect, as opposed to Google that does it to build profiles of internet users for their own re-marketing purposes.

Matomo Analytics are open source and can be used for free on the customer's premises and under its own maintenance. However, they also offer a paid version that can be run on their infrastructure.

GDPR Transparency and Consent Framework by *IAB Tech Lab*

IAB Europe established the Transparency and Consent Framework (TCF)[5] standard to support compliance with the GDPR in the context of digital advertising. This framework is built on four components: a Global Vendor List (GVL), a Transparency and Consent String (TC String), an API for Consent Management Providers (CMPs) to create and process the TC String, and the Policies that govern how the TCF is used.

Prescribed use of the TCF may support compliance with the GDPR, but the real benefit to the digital advertising ecosystem is a safer Internet for consumers, and more reliable data for brands and publishers. As adoption of the TCF increases, the ecosystem becomes more standard, leaving room for automation and therefore compliance becomes more scalable and data becomes more meaningful.

To participate in the use of the TCF, vendors must make a public attestation of compliance with the Policies for using it. To have transparency and consent established and signalled status for their online services stored in a global database, they apply to be added to the GVL. To play a role in creating a TC String for signalling status on transparency and user

[3] https://chrome.google.com/webstore/detail/consent-manager/gpkoajillfmlpnglbagpplnphadbfalh
[4] https://matomo.org/feature-overview/
[5] https://iabeurope.eu/press-releases/iab-europe-iab-tech-lab-release-updated-transparency-consent-framework/

consent, they sign up with IAB Europe to become a CMP. CMPs must follow technical standards provided in this document for creating TC Strings in compliance with TCF Policies. They must also follow technical standards guidance for using the CMP API specified in this document to receive and process information provided in a TC String.

Quantcast Choice

Quantcast Choice is a consent management provider (CMP)[6] solution built on and registered with the IAB Europe Transparency and Consent framework. The product has been tested by international publishers and advertisers on both sides of the Atlantic.

Consumers visiting sites with Quantcast Choice will see one among a number of available user-interface options (all of which can be customized by the website operator) asking the consumer to grant consent for their data to be used in line with GDPR.

Quantcast Choice offers multiple design templates to enable website operators to customize the look of the consent interface. Regardless of the template chosen, consumers will be able to access granular options, such as vendor or purpose-level consent preferences. The consent options the consumer selects will then be applied when the consumer engages with the rest of that site, and potentially other sites and vendors using the IAB Europe framework, depending on selected consent preferences.

Wibson - The antivirus for your privacy

Wibson was launched initially as a blockchain based decentralized data marketplace where users could share their personal data to companies and obtain value for it [2].

Nowadays, Wibson also included new features to their mobile app, now users also can access, manage and control their personal data.

Users install the app and get a complete list of companies that are holding their data, which types of data these companies have about them and have the option to request their data deletion and exercise their data rights[7].

Wibson data management app works as an easy PIMS and its features are listed below:

- Discover:
  - Show a full list of companies that are holding user's data and which data these companies have from them;
  - Map of the world showing where user's data is being used;
  - If a company leaked user's data.
- Control:
  - Possibility to request to delete account and data from a specific company;
  - Stop spams from a company.

[6] https://www.quantcast.com/blog/quantcast-choice-your-solution-for-gdpr-consent/#:~:text=What%20is%20Quantcast%20Choice%3F,both%20sides%20of%20the%20Atlantic.
[7] https://wibson.io/

3.3.- Technical Design

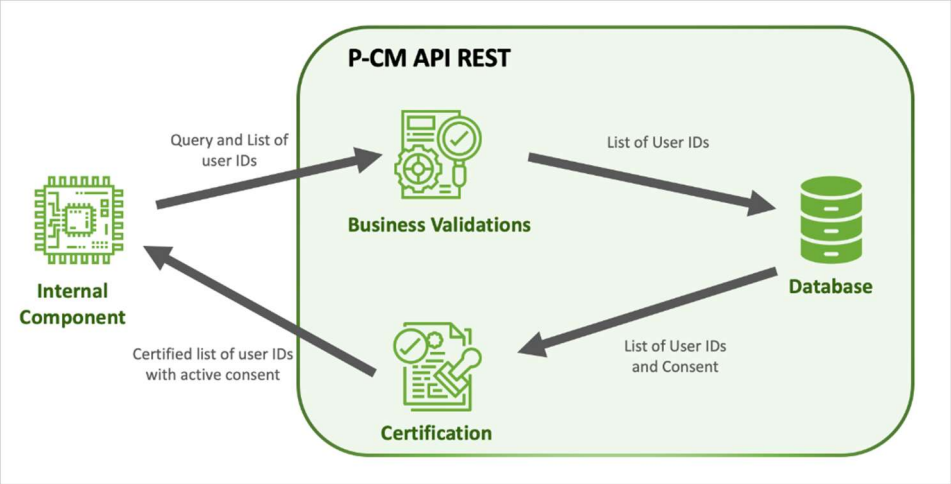
3.3.1.-Internal Operation

Basic Functionality

The P-CM maintains its own database to store all the up-to-date users' consents. That is, it receives all requests through an API REST, applies all the validations and modifies the state in the database if needed.

Afterwards, whenever an internal component from the PIMCity platform wants to query for end users with active consents, a list of user IDs is received in the API and the Personal Consent Manager looks into its database to filter out the ones that revoked such access to the data. The resulting list is signed with the P-CM private key, responding the request with the certificate.

In the following figure, we show how an internal component from the PIMCity platform can get a certified list of users with active consents for a specific query:



Roles and Access

The API is open with three different access levels:

- To the public
- To End Users
- To Internal Components

Therefore, the API defines two different roles:

- End User
- Internal Component

All endpoints related to reading, writing and removing any consent to upload data to the platform or the ability to share it with third parties are restricted to the End User role. Apart from that, an Authorization header is required to allow only the owner of those consents to perform such operations.

On the other hand, all endpoints related to querying for consents and signed certificates are restricted to the Internal Component role, i.e., only an internal component from the PIMCity platform can obtain a certified list of users with active consents.

Moreover, endpoints related to the sign in and out of users are restricted to End User and Internal Component roles for proper authentication.

Last but not least, endpoints related to the general status of the service, like exposing whether it is online or which version is deployed are open to the public.

Architecture

The P-CM is divided in four main sub-components:

- Web Interface: Front-end application used by End Users. This app is created using Typescript as the programming language and React.js as the framework.
- API: REST back-end service that uses JSON documents for payloads. Used by End Users, Internal Components and the public. This API is created using Typescript as the programming language and Node.js with Express.js as the environment and framework, respectively.
- Database: Key-Value store to persist the P-CM state. Accessed only by the P-CM API. The actual DB used is DynamoDB, created by Amazon.
- Documentation Website: Interactive online website showing the documentation for the API REST. Swagger is used to create this website.

Data Model

The P-CM uses three different Key-Value stores to persist the business state:

- Users Store: It stores the name, email and account creation date for every user.
- Data Consents Store: It stores each consent given by the user to store a piece of data in the PIMCity platform.
- Data Sharing Consents Store: It stores each consent given by the user to share a piece of data with a specific company for a specific purpose.

3.3.2.-Interfaces

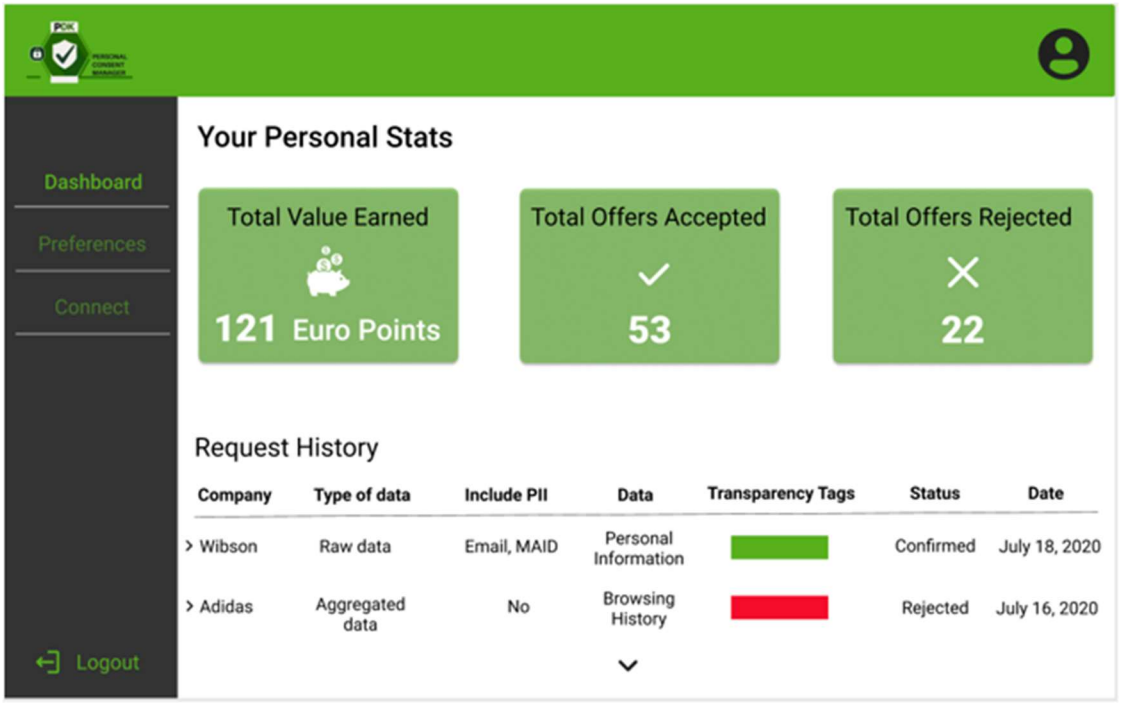
Web Application

For the end user we propose a friendly web interface for the end user. This platform must have a simple but practical user interface. Users must gain the transparency and control they deserve without being a data or technical expert. This platform aims to be used by any "data owner" (user with personal data).

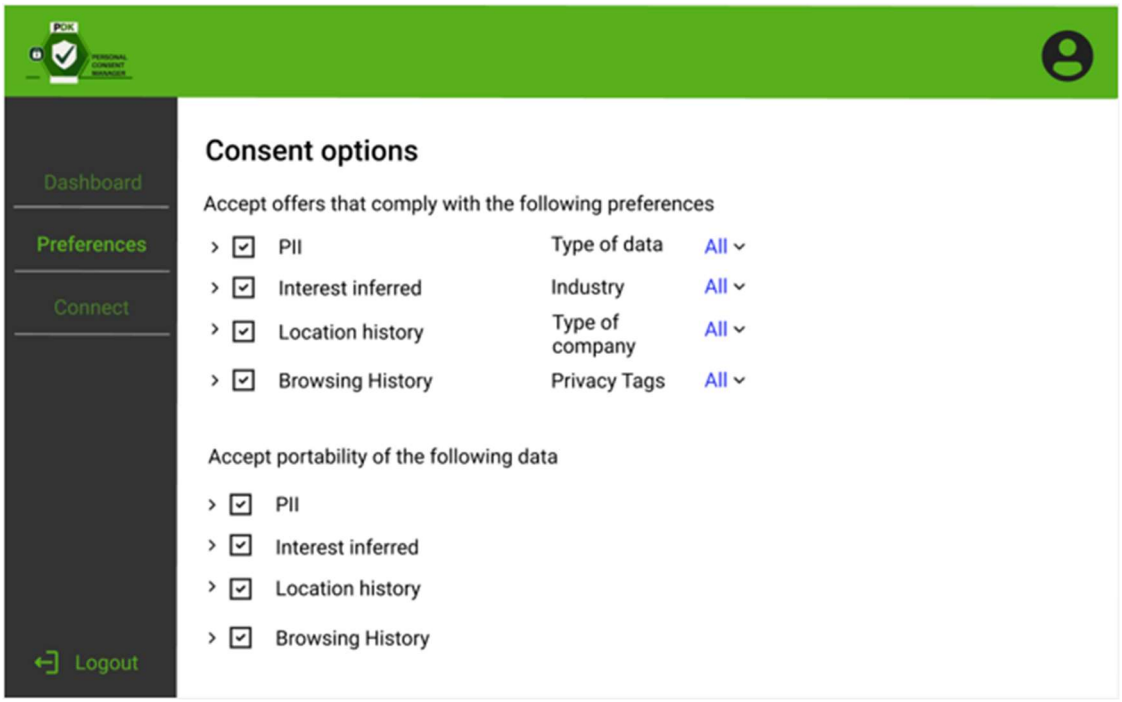
The web application will have different sections to administrate and select which data to share, to accept data offers, to change personal data preferences, to see a balance of the value generated by sharing their personal information and a transaction history to have always a register of who has their data.

Below there are two mock-ups of how this web application could be with their main features.

1. Dashboard is the main section of the web application with an overview of the main metrics and a transaction history:
   - friendly dashboard with the actual status of
     - how they have been sharing their personal data
     - what value have they obtained until that moment
   - List of history transaction to validate and have a copy of
     - who has their personal data
     - which type of data
     - since when a third party is using their data
     - other details regard the data transactions and data itself.



2. Preferences tab is the section where users can change and edit their data sharing preferences:
   - Edit/ change consent automated preferences to potentially
     - Accept data offers in an automated way
   - Select which type of data they are willing to share
     - Whitelist companies or industries



API REST

The API REST is open for the following roles:

- End User: An individual giving and revoking consents to upload data to the platform and share it with Data Buyers;
- Internal Component: a PIMCity service or component using the P-CM API, such as the Trading Engine;
- Public: No restriction applied.

The API REST service then offers the following functionalities:

- Account Management: Sign in, sign out, sign up and profile details.
- Consent Management for storing data with the platform: List, add and revoke consents.
- Consent Management for sharing data with third-parties: List, add and revoke consents.
- Consents Certificates: Provide a signed certificate with the active consents for data storage and data sharing.
- Service Status: Anyone can check if the service is up or not.

API REST Online Documentation Website

Since the API can be used directly by other Internal Components or End Users, it ends up being really powerful and handy to have an online interactive documentation website for the service. By visiting it, the reader can quickly understand and try out the API functionalities. On such site, the endpoints are divided into four different groups:

- Account details such as name, email or profile creation date.
- Consents to store data in the platform.

- Consents to share data with third-party data buyers.
- Certificates for other Internal Components to show a proof of consents to other components.

3.4.- Configuration and Use

Dependencies

- Docker Engine: v20.10.x
- Docker Compose: v1.27.x

Docker is a tool that makes development efficient and predictable.

Docker takes away repetitive, mundane configuration tasks and is used throughout the development lifecycle for fast, easy and portable application development - desktop and cloud. Docker's comprehensive end to end platform includes UIs, CLIs, APIs and security that are engineered to work together across the entire application delivery lifecycle.

Docker Engine is an opensource containerization technology for building and containerizing applications. Docker Engine acts as a client-server application with:

- A server with a long-running daemon process called *dockerd*.
- APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.
- A command line interface (CLI) client docker.

The CLI uses Docker APIs to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI. The daemon creates and manage Docker objects, such as images, containers, networks, and volumes.

Docker Compose is a tool for defining and running multi-container Docker applications. With Docker Compose, the user uses a YAML file to configure the application's services. Then, with a single command, the user creates and starts all the services from the configuration.

Docker Compose has commands for managing the whole lifecycle of the application:

- Start, stop, and rebuild services.
- View the status of running services.
- Stream the log output of running services.
- Run a one-off command on a service.

Configuration

The Personal Consent Manager can be configured in two similar ways:

1. Using environment variables.
2. Defining those environment variables in a file called ".env" at the root directory of the folder with the PDK deployed, declaring them in the same way a variable is defined in the UNIX shell. For example:

```
NODE_ENV=production
PORT=9000
HOST=localhost
CACHE=enabled
ERROR_LOG=/path/to/error.log
COMBINED_LOG=/path/to/combined.log

CONSENT_MANAGER_PRIVATE_KEY=-----BEGIN RSA PRIVATE KEY-----MIICXAIBQCqq(...)
```

The variables (in no particular order) are the following:

- NODE_ENV: development | staging | production
- PORT: the port to be used to run the service
- HOST: the host to be used to run the service (for example: "localhost")
- ERROR_LOG: the path to the file for logging errors.
- COMBINED_LOG: the path to the file for logging all messages.
- CONSENT_MANAGER_PRIVATE_KEY: The RSA private key that the P-CM uses to sign consent certificates.

Execution

In order to run the Personal Consent Manager service, the following command should be executed at the root of the repository:

```
$ docker-compose up
```

This command builds, sets up and runs all the services defined as docker containers in the configuration YAML file, along with the environment variables. Logs can be seen live on the terminal as they are generated.

4.- Personal Privacy Metrics (P-PM)

4.1.- Objective

In this section we introduce PIMCity P-PMs, the means to increase users' awareness about data they share with services. For each service, this module collects and generates privacy and transparency metrics, based on the data that the module is collecting, and the information provided by the service itself. For instance, P-PMs report information about which personal data is collected and shared by data buyers. All these pieces of information are key for a user to take informed decisions and authorize interactions with services. The P-PM presents this information using a standard interface, thus offering an open knowledge information system which can be queried using standard interfaces.

With the P-PM module, we aim to satisfy the following needs:

- The P-PM should generate and provide easy-to-understand, complete, and useful information to allow users understanding the nature of a service (e.g., a website or a data buyer): reputation, trustworthiness, aim and business.
- The P-PM should contain high-level information easy enough to be understood by inexpert users, as well as more detailed, technical information to satisfy the tech-savvy ones.
- The P-PM should summarize and integrate both information actively provided by data buyers and information collected using automatic methodologies, possibly highlighting contradictions.
- The P-PM should be made available in the format of Transparency Tags and presented to the user using the Dashboard.

4.2.- Background and state of the art

In the last decade many initiatives have focused on the design of tools to increase the user's awareness regarding privacy. In the following, we resume them and briefly discuss their design and goals.

Privacy Notices

To the best of our knowledge, the first idea of building pieces of information to inform users about privacy practices has been developed in 2009 within a project held by Carniege Mellon's CUPS (Cylab Usable Privacy and Security Laboratory) team[8]. The project aimed to develop Privacy Nutrition Labels to allow users understand and compare privacy policies made available by web services. Despite the US' Federal Trade Commission officially supported the project and recommended the usage of such approach the project did not take hold in the industry, thus vanishing its efforts.

In 2015 Metwalley *et al.* proposed in [3] the design of a distributed system called CrowdSurf, to empower the means for users to get information about web services, supervise the data shared with them, and ultimately take informed decisions regarding their privacy. The paper presents the overall architecture of the system, describes how its elements should interact

[8] https://cups.cs.cmu.edu/privacyLabel/

with each other, but does not describe which information should be provided to users in details and how.

In 2018, Karaj *et al.* proposed in [4] the design and results of Whotracks.me, a system to collect and present information about web tracking ecosystem to integrate in Cliqz browser. The system collects and presents a wide amount of information about websites, tracking and advertisement services, with many detailed statistics and scores to make the user aware of which personal data such services collect and share. Whotracks.me represents a significant source of inspiration for the design of P-PM.

Collection and analysis of information about web services

There exist a wide number of works, projects and products which design and implement tools to automatically infer information about web services. These systems usually build on active measurements (web scrapers) to automatically visit websites, download data and later process it to perform classification for privacy analysis purposes. In this family it is worth citing Englehardt and Narayanan [5], Rizzo *et al.* [6], Matic *et al.* [7]. Usually, these systems focus on the complex problem of automatically collecting web information. They provide the tools rather than synthetizing the results and conceiving conclusions in an easy-to-understand format. Nevertheless, the web data collection is a fundamental part for the generation of P-PM.

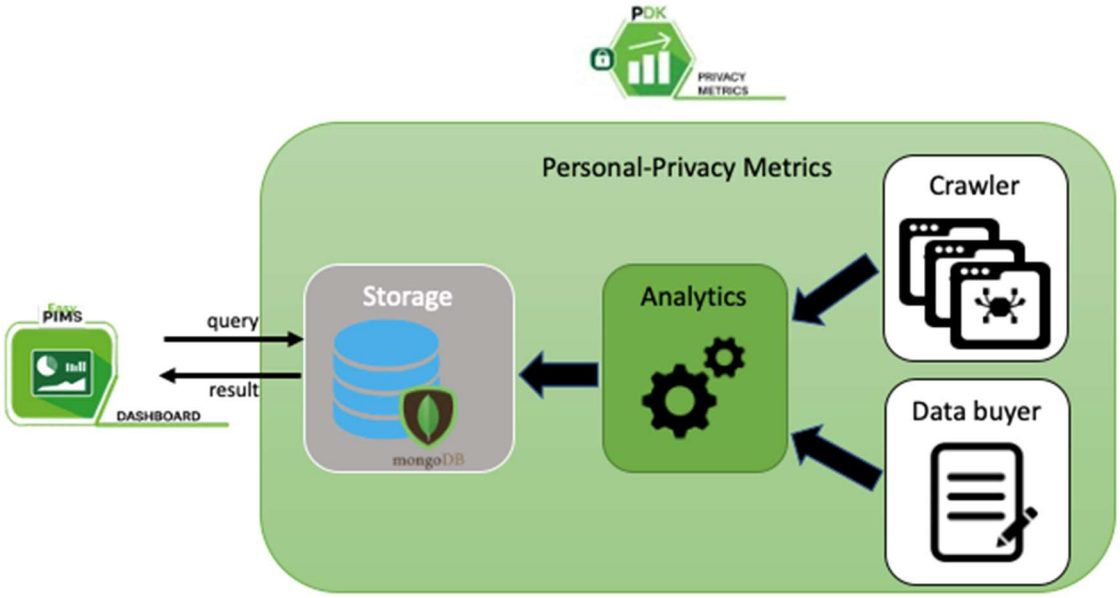
Considering the industry, there are several commercial products which provide web service classification functionalities. Among them, we mention zVelo[9], Similarweb[10], and Symantec[11]. However, little is known on which data is used for the classification and how this is actually performed[12].

4.3.- Technical Design

4.3.1.-Internal Operation

P-PM will be generated based on information actively provided by services participating PIMCity (i.e., data buyers) integrated with data gathered from measurement campaigns run using automatic web crawlers. Hence, the module is designed as a database populated with information from data buyers and automatic crawlers. Such information will be made available in read-only mode to the Dashboard which will present P-PMs as Transparency Tags. The picture below summarizes the generic design of P-PM components. Depending on P-PM provider's requirements, the P-PM may contain the database element only, or all the elements needed to build the whole data analysis, collection and storage workflow.

[9] zVelo https://www.zvelo.com/
[10] Similarweb https://www.similarweb.com/
[11] Symantec BlueCoat https://sitereview.bluecoat.com/#/
[12] https://eprints.networks.imdea.org/2183/1/paper.pdf



The fundamental block of the above architecture is the database whose content is firstly defined by schemas it will support. In D2.1 we envisioned a generic schema to use and adapt depending on the use case scenario. We carefully revised it based on the feedback we collected from different stakeholders (mainly users, data buyers and developers).

In the following we resume the main categories composing the P-PM:

- **Information from data buyers**: this is information actively provided by data buyers participating a PIMCity deployment. Data buyers will be required to provide this information themselves for transparency purposes and demonstrate to be fully compliant with GDPR requirements.
- **Information from web data:** This will present information obtained by processing web data gathered from web crawlers. In particular, crawlers will automatically visit the website (or the list of websites) provided by the data buyer and collect information useful to perform classification.
- **Scores:** these will be computed based on both information provided by data buyers and data collected from web crawlers. To enhance competitiveness among different P-PM providers, we decide not to provide specific guidelines for the computation of scores. Hence, P-PM providers are free to choose data, methods and algorithms for computing the scores, and possibly describe the generation method behind. For instance, the P-PM provider may decide to modulate the Transparency Score based on a comparison among information provided by the data buyer and public web data. We provide details on how ERMES has implemented scores for the context of EasyPIMS.

In the following we report the revised version of the P-PM template. In the subsequent text we discuss the differences with the template presented in D2.1.

Main Class	*Secondary Class*	*Field name*	*Field description*

Privacy Metric	-	ID	*Id assigned by the system at creation time*
		Name	*Domain name for the privacy metric*
	Provided Information	Declared company name	*Company name as declared by the company itself*
		Website	*Website provided by the company*
		Country	*Establishment country*
		Category	*Self-declared categorisation information based on kind of business and mission*
		Processed data	*Describes which kind of data is collected and processed*
		Data processing purpose	*Describes how collected data will be used by the company*
		Data processing legal basis	*Describes whether the processing is based on Consent (default) or Legitimate interest*
		Data for Automated decision-making	*Describes whether such data will be used for automatic decision-making, including profiling and possible consequences for the user*
		Data Controller	*Describes who owns the data subject to processing*
		Data Processor	*Describes who is going to process data*
		Data Subprocessors	*Describes possible sub-processors involved in the data processing*
		Data recipients	*Describes who will receive the personal data (if any)*
		Data retention period	*Describes how long data is going to be stored by the data owner, or the criteria used to determine the period*

		Data location	*Describes where data is going to be physically stored (in or outside EU)*
		Data transfer	*Describes whether the controller plans to transfer personal data to a third country or international organization*
		Rights	*Describes the resources where the user can get information to apply rights on her data.*
		Consent withdraw	*Describes how the user can withdraw consent to data processing.*
		Controller identity	*Identity of the controller or, controller's representative*
		Controller contact details	*Mail address, phone and physical address to contact the controller*
		DPO contact details	*Mail address, phone and physical address to contact the controller's DPO*
	Web data	Company name	*Company name gathered by crawler*
		Operates under	*Website(s) gathered by crawler*
		Category	*Categorisation information obtained by crawler*
		Rank in category	*Popularity based on users' traffic*
		Connected third-party services	*List of third-party services connected to main website*
		Connected websites	*List of websites where the company appears as a third party*
		Connected categories	*List of website categories associated to this website*
		Tracking devices	*List of user tracking technologies*
		Privacy policy	*URL to privacy policy*

		Presence in security lists	*Website/company has been reported as risky*
		Profiling purposes	*Describes which information is extracted from user's data*
	Scores	Privacy score	*[0 to 5] based on privacy-related information*
		Transparency score	*[0 to 5] based on matching between provided and gathered data*
		Security score	*[0 to 5] based on security-related information*

The schema presented above represents a design proposal based on that proposed in D2.1. This has been augmented with information and details suggested during a round of feedback collection. In particular, ERMES, with the help of KUL, Wibson, and IAB empowered the parts of P-PM describing how the data buyer will collect, process and exploit collected data. In details, the new P-PM aims at addressing the following points and questions:

- Provide identity and the contact details of the controller and, where applicable, of the controller's representative;
- Provide the contact details of the data protection officer, where applicable;
- Describe the purposes of the processing for which the personal data are intended as well as the legal basis for the processing;
- Where the processing is based on legitimate interest, is the legitimate interest pursued by the controller or by a third party?
- Describe the recipients or categories of recipients of the personal data, if any;
- Provide information about whether the controller intends to transfer personal data to a third country or international organisation - where applicable.
- Describe the period for which the personal data will be stored, or if that is not possible, the criteria used to determine that period;
- Provide information about the existence of the right to request from the controller access to and rectification or erasure of personal data or restriction of processing concerning the data subject (i.e., the user) or to object to processing as well as the right to data portability;
- Describe how the possibility to withdraw consent at any time is provided;
- The right to lodge a complaint with a supervisory authority;
- Describe whether the provision of personal data is a statutory or contractual requirement, or a requirement necessary to enter into a contract, as well as whether the data subject is obliged to provide the personal data and of the possible consequences of failure to provide such data;
- Describe whether any automated decision-making exists, including profiling and, at least in those cases, meaningful information about the logic involved, as well as the significance and the envisaged consequences of such processing for the data subject.

For what concerns the scores, ERMES has implemented a simple, yet effective, penalty-based algorithm. It builds on the information provided in Provided Information and Web data to generate the scores the user can exploit as a simple tool to understand the privacy, transparency and security posture of the data buyer. More in detail, penalizations will be applied on the three axes (privacy, transparency and security) based on the information in the P-PM. Penalizations have associated a weight and a short description that will assist the user or the data buyer itself at understanding the motivations that drove the algorithm to generate the score. Starting from the maximum value allowed (5), the algorithm will identify the penalizations (e.g., information missing, or information incoherent, etc.) and subtract the corresponding weight. The subtraction will stop when the minimum value (0) will be reached, or no more penalization should be applied. In the following we report the penalization we have envisioned so far. Penalizations and corresponding weights have been formulated by ERMES based on its expertise in the field and on feedback collected during the design process. The list will definitely be increased with the progress of the project. We can resume them as follows:

Axes	Penalization Weight	Description
Privacy	-0.5 for every GDPR-related field left empty	Any of the GDPR-related information is missing
	-0.05 for every tracking service	The website is connected to a tracking service
	-1	The website is connected to session replay scripts/web keyloggers
	-1	The website is connected to fingerprinting services
	-2	The privacy policy is missing
	-2	Data location or data transfer destination is outside EU
	-3	Information about data access rights is missing
	-2	Identity or contact details are missing
	-2	If fingerprinting is used by the website
	-1	If local storage or cookies are used to store PII
Security	-0.5 for every missing security header	Important security header is missing
	-2	The website is connected to session replay scripts/web keyloggers
	-1	The website is connected to fingerprinting services

	-4	The website is present in some security list
	-4	Any of connected third parties is present in some security list
	-0.05 for every tracking service	The website is connected to a tracking service which can increase the surface attack
Transparency	-3	The purpose of data processing is not provided
	-2.5	The privacy policy is not provided
	-1.5	Identity or contact details are missing
	-0.5 for every GDPR-related data missing	Any of the GDPR-related information is missing
	-4	The website is present in some security list

1. **P-PM for data buyer participating PIMCity**

In the following we report the example of P-PM for data buyer ACME Ltd, updated accordingly to the new format.

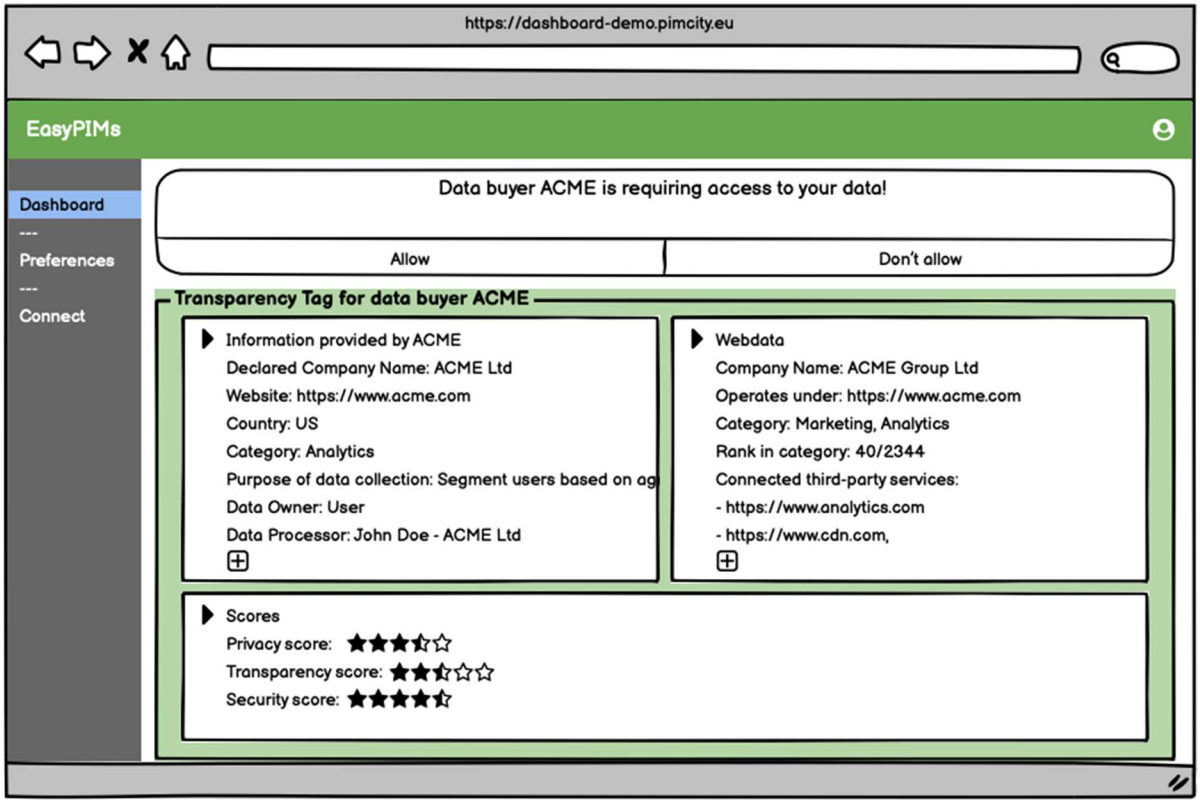
Main Class	***Secondary Class***	***Field name***	***Field description***
Privacy Metric	-	ID	602f863578d123cffc788ccf
		Name	*www.acme.com*
	Provided Information	Declared company name	*ACME Ltd*
		Website	*https://www.acme.com*
		Country	*US*
		Category	*Analytics*
		Processed data	Age, job position and net income
		Data processing purpose	Profile and segment users based on age and job position
		Data processing legal basis	*Consent*
		Data for Automated decision-making	*Data will be used for profiling. As a consequence, user will be provided with personalised*

			marketing campaigns and product offerings.
		Data Controller	*ACME Ltd*
		Data Processor	*ACME Ltd*
		Data Subprocessors	*NA*
		Data recipients	*NA*
		Data retention period	*5 years*
		Data location	*Ireland*
		Data transfer	*Personal data will not be transferred to countries outside EU or to international organizations*
		Rights	*ACME Ltd details how users can apply their access rights in the privacy policy:* Access – www.acme.com/privacy#access Rectification – www.acme.com/privacy#rectification Erasure - www.acme.com/privacy#erasure Restriction – www.acme.com/privacy#restriction Portability - www.acme.com/privacy#portability
		Consent withdraw	*ACME Ltd details how users can withdraw consent to data processing at* www.acme.com/privacy#withdraw
		Controller identity	*Jane Doe, DPO* *John Doe, CISO*
		Controller contact details	*EMail: ciso@acme.com* Phone: +3412345566 Address: ACME Ltd, Sunset Boulevard, Dublin, IRL
		DPO contact details	*EMail: dpo@acme.com*

			Phone: +3412345566 Address: ACME Ltd, Sunset Boulevard, Dublin, IRL
	Web data	Company name	*ACME Group Ltd*
		Operates under	*https://www.acme.com*
		Category	*Marketing, analytics*
		Rank in category	*40/2344*
		Connected third-party services	*https://www.analytics.com* *https://www.cdn.com* *https://www.social.com*
		Connected websites	*https://www.acme-retail.com*
		Connected categories	*Shopping, social network*
		Tracking devices	*Cookies*
		Privacy policy	*https://www.acme.com/privacy*
		Presence in security lists	*No*
		Profiling purposes	*Age, job position, net income, family role*
	Scores	Privacy score	***3.90**: The data buyer uses cookies to collect PIIs (-1). The data buyer is connected to tracker activity (-0.1).*
		Transparency score	***5:** all data provided*
		Security score	***4:** Important security header is missing on the website of the data buyer (-1)*

4.3.2.-Interfaces

The P-PM module interacts with the User Dashboard. The User Dashboard web app queries the DB contained in P-PM in read-only mode to fetch privacy metrics in JSON-like format and present them as per-data-buyer/service Transparency Tags. This provides users information to take decisions during the configuration of the Consent Manager, or simply to provide a browsable collection of P-PMs using their interface, the Transparency Tags. In the following we present a simple mockup draft for consultation of P-PMs using the interface provided by Transparency Tags:



We emphasize that the development of the interface (i.e., a web page or an app) to present the content of P-PMs will be discussed when we will design the Transparency Tags in WP5. Hence, for the sake of flexibility, choices strictly regarding the presentation of P-PMs will be discussed in the design phase of the Transparency Tags. For instance, we will let the designer decide whether to develop proper UIs to allow the user to compare information provided by data buyers and public web data or aggregate information depending on user's technical skills.

The P-PM module will be made interactable with other two components. These are i) the Analytics, which are in charge of creating the items in P-PM, update the Web data part and compute the Scores, and ii) the Data Buyer Dashboard, which allows the Data Buyer to populate the Provided Information part of the Privacy Metrics. The development of the Analytics is carried out by ERMES, while the Data Buyer Dashboard is part of the Trading engine component. We highlight that scope-based authorization (and possibly authentication) are mandatory for all above components: User Dashboard, Data Buyer Dashboard and Analytics.

4.3.3.-Implementation

Implementation blocks

In the context of PIMCity, ERMES is in charge of the development of the PDK component of the Privacy Metrics. ERMES has developed all the fundamental blocks needed to the constructions of P-PMs, i.e., the crawler to fetch the web data, the analytics to process and aggregate such data as well as the database block and the server interface implementing the RESTful APIs to access and modify the data. However, considering that the crawler and the analytics represent two critical pieces of ERMES' intellectual property which are exploited to build and increase its own products and protection spectrum, in agreement with

the consortium, ERMES has decided to release the code implementing the server and the database populated with data from 10,000 websites. Apart from protecting ERMES IP, this choice allows users and developers to re-use and deploy the code quite freely, contributing to increase the awareness of users and data buyers on important aspects of online privacy and existing regulations. Moreover, we believe this approach will make the tool easy to access, notably contributing to the dissemination effort of the project.

For the sake of transparency, ERMES has also released the algorithm and the corresponding code to generate the Scores. This will allow both Data Buyers and End Users to fully understand the mechanism behind the score. Data Buyers will be provided with all data to understand and improve their scores. On the other hand, the Users will be provided with a simple and transparent tool to understand what is good and what is bad in terms of online privacy, security and transparency.

Finally, we remark that the web interface needed by the Data Buyers to read and modify their own P-PMs has been developed. As such, Data Buyers will be provided the tools to provide their data and increase their transparency to the users.

In the following we describe the APIs to build and deliver the P-PM component to meet the requirements that have emerged during the first half of the project, both technical and content-wise.

Roles and Access

The P-PM RESTful API is open with three different access levels:

- To the public
- To End Users
- The Data Buyer
- The Internal component

Therefore, the API defines three different roles:

- **End User**: An individual registered to the PIMS platform and willing to access data provided by Data Buyers and other popular services;
- **Data Buyers:** a company registered to the PIMS and willing to provide its data to End Users in a transparent way.
- **Internal Component**: another component being part of the PIMS platform. In this specific case, it consists of the analytics needed to develop the P-PMs.

In order to guarantee a sufficient level of security, humans (End Users and Data Buyers) interacting with the P-PMs component will be required to use the OAuth2/AuthorizationCode workflow (see D5.1 for details). Instead, internal components will be required to use the authorization workflow for M2M interactions, OAuth2/ClientCredentials.

API REST

The API REST is open for all roles above. However, access must be authorized (authenticated, if possible) for the first three roles, while no restriction is required for the last one.

The API REST service then offers the following functionalities:

- Obtain P-PMs: All authorized roles can fetch the list of P-PMs as well as their internal content.
- Manage content of P-PMs: Analytics are authorized to modify Web data and Scores sections of P-PMs. Data Buyers are authorized to modify Provided Information only.
- Create and Delete P-PMs: Analytics are allowed to create to P-PM objects as well as delete them.
- Service Status: Anyone can check if the service is up or not.

API REST Online Documentation Website

In the following, we report the summary of API interfaces that can be used directly by Internal Components, End Users and Data Buyers.

- **GET /privacy-metrics** allows all stakeholders to obtain the list of services for which Privacy Metrics are available.
- **POST /privacy-metrics** allows the analytics to create a new Privacy Metric.
- **GET /privacy-metrics/{identifier}** allows all stakeholders to obtain the Privacy Metric corresponding to id
- **DELETE /privacy-metrics/{identifier}** allows the analytics to delete the Privacy Metric corresponding to id
- **PUT|PATCH /privacy-metrics/{identifier}/provided-information** allows the Data Buyer to substitutes|modify its Provided Information
- **PUT|PATCH /privacy-metrics/{identifier}/webdata** allows the analytics to substitutes|modify webdata corresponding to Privacy Metric id
- **PUT|PATCH /privacy-metrics/{identifier}/scores** allows the analytics to substitutes|modify scores corresponding to Privacy Metric id

We remind that all P-PMs will be created and managed in JSON-like format.

TOOLS

For implementing the P-PM component that will be made available as open source, ERMES has used MongoDB to store the P-PMs and Swagger to design the web server exposing the API interfaces, determining the authorization workflow and shape the data schema. For what concerns the implementation of the server, the choice fell on Python using module Flask and MongoDB. The code implementing the server as well as the database exposing the P-PMs of 10,000 websites are available in the WP2 code repository of PIMCity's official Gitlab.

4.4.- Configuration, Installation and Use

The implementation of Privacy Metrics module is available on PIMCity's official GitLab (https://gitlab.com/pimcity/wp2/privacy_metrics). We implemented it using Python. For the sake of simplicity and ease of use, we used Poetry and Cookiecutter to build and maintain the code. This will allow users and maintainers to neglect the typical issues related to code readability and linting. Indeed, the Cookiecutter template, when possible, automatically adjusts lines of code not respecting readability and linting rules, or notifies the developer the lines of codes which require manual fixes.

Instructions for installing and use the code are available at https://gitlab.com/pimcity/wp2/privacy_metrics/-/blob/master/README.md.

5.- Personal Privacy Preserving Analytics (P-PPA)

5.1.- Objective

In this section we introduce PIMCity's P-PPA, a tool to allow analysts and stakeholders to retrieve useful information from personal data provided by the users, while preserving their privacy. When talking about analysis of personal data, we must balance the aim of companies to draw knowledge from personal data – also by the means of Machine Learning (ML) algorithms – and users' requirement of maintaining privacy intact by not exposing their private information. However, usually the user wants to achieve this goal – preserving its own privacy – without having to give up the online service she is providing her data to.

With the P-PPA module, our goal is to achieve the following objectives:

- Personal data exporting to third parties is kept under control with the privacy-preserving goal in mind.
- The P-PPA should be able to provide different types of output, depending of the need of the analysis: this includes aggregating data, minimizing loss of information and more.
- The P-PPA should act both in a static way (for analysts who require already-cumulated data) and in a dynamic way (for those who are instead interested in analysing a stream of incoming data, with the minimum possible delay).
- We want to design P-PPA component in a modular and extensible way, in order to include possible future mechanism at a low cost

Finally, notice that the Data Aggregation PDK module provides data anonymization techniques, but covers a different use case (see Deliverable 4.1). It provides anonymization functions to allow bulk insert of data into EasyPIMS, while the P-PPA regulate how these data are exported to the stakeholders.

5.2.- Background and state of the art

In the past decades, several data anonymization techniques have been proposed to build privacy-preserving analytics. Since the analytics are strictly tailored to the needs of the inquirer, there exists no standard approach, while many alternatives aim at solving the problem in different scenarios. In the following, we present the most popular data anonymization approaches.

K-anonymity

Many researches in the past years showed that removing the identifiers from a dataset is not a sufficient way to protect privacy: in 1997, Sweeney was able to identify the Massachusetts governor in a publicly listed set of medical records, using its birth date, its zip code and its gender [8]. In 2006, Netflix published a dataset while launching a contest to improve its recommendation algorithm: the dataset consisted of rating provided by users, whose identities were not revealed. However, linking the information provided by Netflix

with the one publicly available on sites like IMDB, Narayanan and Shmatikov were able to identify some of the Netflix users [9].

K-anonymity [10] aims at reducing the probability for a user in a dataset to be identified down to 1/k. The goal of k-anonymity is to group the users according to their quasi-identifiers (those attributes who are apparently harmless, but whose combination may lead to re-identification – such as zip code, birth date and so on) and enforce that each one of these groups (called *Equivalent Classes* – ECs) contains at least k rows of the dataset – i.e., k users: it is common to assume that each user only appears in a row. This property guarantees that if a user is linked with a sensitive information (such as a medical record, the income, and so on), it can hide himself among other k-1 users. K-anonymity is typically achieved through two methods: generalization and suppression. In the latter method, some rows may be removed from the published dataset in order to form ECs with appropriate size. With generalization, instead, the single attributes may be modified in order to find k indistinguishable users. For numeric attributes, the actual number is replaced by a range that includes it. For categorical attributes, a common parent value in a hierarchy tree may replace the child ones (e.g., two values such as "French" and "Spanish" may be generalized as "European").

Being k-anonymity a data property and not an algorithm, several mechanisms to obtain it have been proposed, such as the Incognito [11] and Mondrian [12] algorithms.

K-anonymity suffers from a number of known flaws that affect its privacy preservation promises. First, it is susceptible of the *homogeneity attack*: if all the users in an equivalence class have the same sensitive attribute, there is no protection for a user in it (we do not know *which her identity*, but we know her sensitive attribute). K-anonymity also suffers *background knowledge attack*. For instance, if we are aware that Japanese people are less affected by heart attack than the average, we could use this information to weaken the k-anonymity protection in a dataset reporting heart attack records of a multi-ethnic population.

Hence, alternatives to k-anonymity are l-diversity and t-closeness. The former adds the constraint «that the values of the sensitive attributes are "well represented" in each group» [13] (and thus faces the homogeneity attack), while the latter requires «that the distribution of a sensitive attribute in any equivalence class is close to the distribution of the attribute in the overall table» [14] (tackling the background knowledge attack). Regardless on the introduction of new privacy properties, k-anonymity, for its simplicity, is still regarded as the golden standard for anonymization.

Differential privacy

Differential privacy aims at providing a formal definition of privacy. This definition claims that a user's privacy is protected if the knowledge an attacker has on a queried dataset is not too affected by whether the user is in the dataset or not. The "too affected" notion is parametrized by ε [15].

A common mechanism to obtain differential privacy is to apply the Laplace mechanism, which consists in adding a Laplace-distributed noise to the query result. The noise is extracted from a Laplace distribution which depends both on the ε value and on the *sensitivity* of the query – i.e., on how much the presence or absence of a user can affect the result. It is proved that the Laplace mechanism guarantees differential privacy.

Existing libraries

Many libraries implement algorithms for achieving k-anonymity. Some are based on the Mondrian algorithm[13]. Programs exist that apply k-anonymity principles to Microsoft Excel, CSV files or SQL databases tables, privacy-protecting them[14].

For what concerns Differential Privacy, an interesting tool has been provided by IBM[15]. The library offers the possibility to apply differential private algorithms to perform machine learning tasks such as classification and clustering, or to directly interact with the primitives to create differentially private queries to the database.

5.3.- Technical Design

5.3.1.-Internal Operation

The P-PPA module aims at offering different analytics solutions, depending on the available data and on the requirements of the querying party. Each analytics solution is implemented into a separate function. In our view, the functions should be the following, but a subset or a superset of these may be implemented by the P-PPA provider:

- k-anonymity,
- differential privacy,
- z-anonymity (specifically designed for PIMCity a described below).

The P-PPA module can work both with *historic* data – i.e., values that are single for each user and rarely or not-editable, such as date of birth, address and so on – and with *recurrent data*, which may instead be exposed by the user several times: they include browsing history (one record per site visited by a user), location history (a periodic update on user's location) and similar.

k-anonymity

The k-anonymity function aims at implementing k-anonymity on incoming datasets, especially those composed of historic, static datasets. We assume that the data will arrive in a structured form (e.g., a Python Pandas Dataframe object). The function will take the data and k-anonymize it; to perform k-anonymization, we will use existing open-source implementations. They will be tested to be ensure they work in a correct and efficient way. This holds for l-diversity and t-closeness.

At the end of the anonymization process, the data can be provided to the third party in the same structural form it has been received by the P-DS.

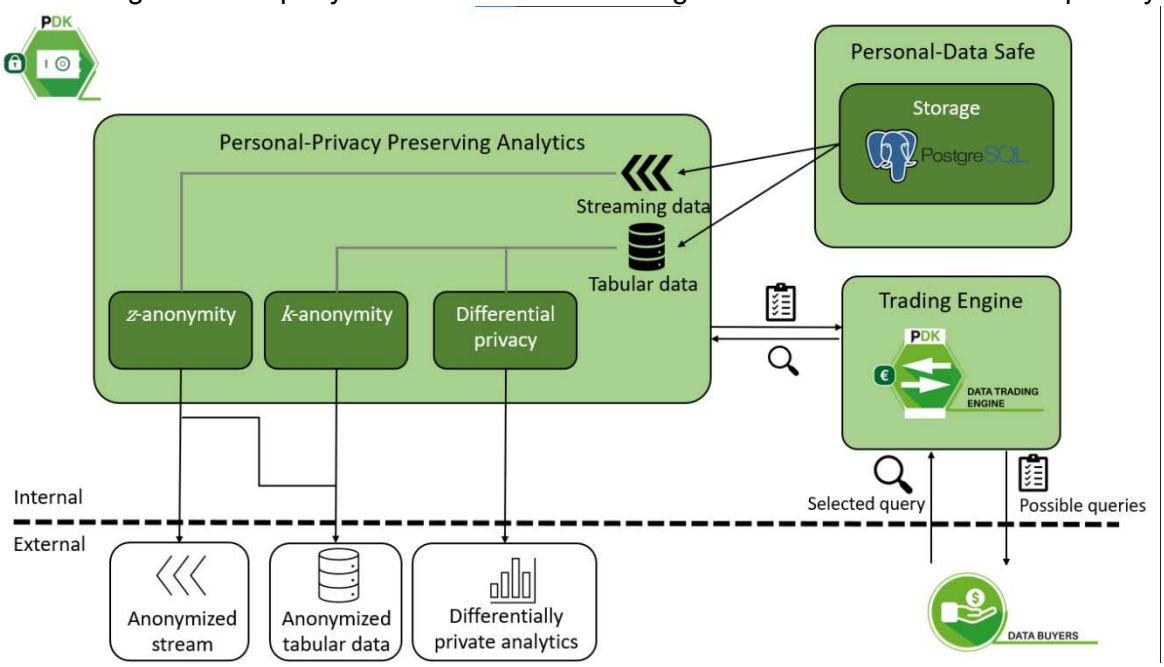
Notice that k-anonymity is adopted in the Data Aggregation (DA) PDK module as well, with the goal of achieveing anonymized bulk insert into EasyPIMs. However, it has a very

[13] https://github.com/qiyuangong/Mondrian
[14] https://github.com/arx-deidentifier/arx
[15] https://github.com/IBM/differential-privacy-library

different goal, as it seeks at anonymizing data at insertion-time, while the P-PPA aim at providing the data buyers with anonymized data. More details on the DA module can be found in Deliverable 4.1.

Differential privacy

For the differential privacy function, the P-PPA module exposes a set of queries that the querying party can use on the dataset. The queries could be stand-alone or parametrized, with the possibility for the inquirer to model the query following its own preferences, while maintaining a query structure that guarantees differential privacy.



Following the principles of differential privacy, every querying party is given a *privacy budget* to spend, meaning that it cannot perform an unlimited number of queries. Indeed, each query sent to the P-PPA has a privacy cost that decrements the budget of the querying party. To avoid behind-the-scenes data exchange between querying parties to avoid the budget constraint, a global budget could also be set. The privacy budget could be checked on a moving time window.

A differentially-private algorithm returns the query results as a JSON file, composed by key-value pairs: a single pair is needed if the output of a query is a value, several pairs are used if the output consists of a histogram of values, whose bin values are used as key of the JSON file, and whose value are the respective values.

When inquiring a static dataset, the differential privacy function may work especially well for historic data.

z-anonymity

Another function inside P-PPA implements the *z-anonymity* [16]. z-anonymity is a privacy property that aims at protecting *z-private* attributes. An attribute is said to be z-private if it has not been exposed by at least z users in the past time period Δt.

z-anonymity is suited especially for streaming datasets, that need to be analyzed while protecting users' privacy – e.g., web traffic monitoring. It guarantees a zero-delay (hence the name) evaluation of the attribute privacy and its potential output. If the attribute is not z-private, z-anonymity assumes that the information is not potentially harmful for the user and thus can be published (the user's identifying attributes, such as the name, are replaced by random, identifying codes that are assigned to each user). Otherwise, the incoming information is blurred not to risk user's private information leakage.

By design, the z-anonymity function suits the usage of recurrent data.

5.3.2.-Interfaces

The P-PPA module interacts with two separate entities: on one side, it obtains data from the P-DS module. On the other one, it interacts with the Trading Engine to take into account the queries from the Data Buyers.

Interface with Trading Engine

The P-PPA module will interact with the Trading Engine as an interface to the data buyer. The P-PPA module will communicate to the Trading Engine the functionalities it supports (at the moment k-anonymity, z-anonymity and differential privacy), combined with the available queries for the differential privacy function – e.g., in a JSON-like format; on the other side, it is expected to receive the type of the analysis the data buyer is willing to perform and the pointer to the appropriate dataset to query, along with appropriate parameters. The need of the Trading Engine as a proxy between the P-PPA module and the Data Buyers is motivated by a number of reasons, including the desire to keep the P-PPA modular, as it only processes a query over a dataset, evaluates the result and returns it, without worrying for the interaction with an external party; Moreover, the Trading Engine has a complete view over the state of the Data Buyer (e.g., the number of queries it has performed, its billing condition), thus it is more suited to manage the interaction to it. Following the choice of keeping the P-PPA module covered from the outside, even the results of the queries are first transmitted to the Trading Engine, which in turn forwards them to the Data Buyers it is in contact to.

As a web framework, we chose Flask[16] among different most famous possibilities, because of its simplicity and its expandability capacities. First released in 2010 and coded in Python, Flask is currently used by applications such as Pinterest and LinkedIn. It provides all the basic functions needed for the PPPA implementation that aims to maintain a simple and modular approach.

Flask is defined as a microframework, since it does not come with third-party, built-in libraries, but allows the user to define and implement components that best fit with the project design choices. It provides a sort of agnostic connecting layer through which it is possible to expand the project functionalities such as a database layer or an authentication

[16] https://github.com/pallets/flask

module. The Flask adoption permits to dispatch the required RESTful API set, necessary to make the PPPA module functionalities to be accessed from others PIMCity components.

Interface with P-DS

To retrieve the data to treat in order to deliver P-PPA, the module has to receive the users' data via the P-DS module (see section 6). In particular, the data will be retrieved from the database that lays in the P-DS – whose choice is detailed in Section 6.3.1.

The module receives the data in a JSON-like format. The file(s) to retrieve from the database is evaluated by parsing the queries from the data-querying parties (see the next section for details).

The P-PPA module can offer the results under several shapes, related to which of the three functions have been applied to the incoming data: k-anonymization returns another table with the same format of the input, but with generalized (and/or suppressed, depending on the used algorithm) attributes to ensure users' anonymity. For what concerns differentially private queries, the results may be broadcasted in the form of a single value or a histogram, depending on the query that have been required to the module. Finally, the output of the z-anonymity module can be both broadcasted as a table – which cumulates the real-time output of the system – or tuple-by-tuple. All the query results will be offered to the inquiring third parties as an answer to the query, through an API.

5.4.- Configuration and Use

At the moment, the P-PPA module is designed as a Python project. The project is self-contained and does not need any external components to be installed.

Installation

It is possible to access the current P-PPA module implementation at its GitLab location (https://gitlab.com/pimcity/wp2/personal-privacy-preserving-analytics). The Python requirements are listed as follows:

```
python==3.6
numpy==1.18.5
diffprivlib==0.3.0
matplotlib==3.3.3
pandas==1.1.5
psycopg2==2.8.6
pymongo==3.11.2
SQLAlchemy==1.3.20
requests==2.24.0
Flask_RESTful==0.3.8
Flask==1.1.2
PyYAML==5.4.1
```

The z-anonymity project (https://gitlab.com/pimcity/wp2/zeta-anonymity), which will be later implemented inside the P-PPA one, only requires:

```
python==3.6.
```

6.- Personal Data Safe (P-DS)

This section describes the final design and implementation of the Personal Data Safe (P-DS). We first introduce the overall goal of the module and then describe the state of the art on the existing solution for the storage of personal data. We also carefully describe the final architecture and design of the PIMCity P-DS in terms of internal functioning and interfaces for users and other PIMCity components. We motivate our design choices and briefly discuss potential alternatives.

We release the code of the Personal-Data Safe as open source, and it is available online at: https://gitlab.com/pimcity/wp2/personal-data-safe

6.1.- Objective

The objective of this section is to describe the PIMCity P-DS, the PDK component to store securely and transparently the users' data, as well as providing a set of innovative functionalities. While regulations such as (GDPR) defined guidelines and rules on personal data management, there is no precise and clear regulation for governing the storage of personal data. Nowadays, users' data are distributed over different systems and platforms: Facebook, for example, knows users' likes and interests, Google holds the list of the web search queries and Amazon the list of purchased items. The European law mandates that users have the right of retrieving their data at any time, but this fragmentation leads to a loss of control because the user does not have a centralized tool for handling these data. A possible solution to this problem is represented by particular storage systems, called Personal Information Management Systems (PIMS), which store data in dedicated Personal Data Safes (P-DS). The objective of this section is to describe the PIMCity Personal-Data Safe.

PIMSs and P-DSs are still at an early stage of development, and there is no standard definition. With the PIMCity P-DS, we aim at fulfilling the following requirements:

- The P-DS is the means to store personal data in a controlled form.
- The data structure shall be flexible in order to accommodate diverse types of user data, such as generic personal information but also, e.g., browsing or location history.
- It shall allow automatic and manual data import for users as well as interfaces for data buyers that shall retrieve data upon the users' consent.
- The P-DS gives users the possibility to store new data manually or import them from existing platforms. As such, it must offer a user interface as well as APIs for automatic data management.
- The P-DS shall also be easy to deploy and use state-of-the-art techniques and software libraries.
- The P-DS shall be efficient, fast and scalable to accommodate the data of a very large set of users.

6.2.- Background and state of the art

The advent of the GDPR fostered the creation of personal information management systems. Companies and organizations proposed their own solutions over the last (few) years, without following any standard or common guidelines. Indeed, the world of P-DS is

very heterogeneous, and each solution is peculiar, and differs from the others. Here, we aim at providing a brief description of the popular products available nowadays in the market and as open-source solutions.

OpenPDS

OpenPDS[17] is an open-source project, implemented at the MIT. It consists of a personal metadata management framework that allows individuals to collect, store, and give fine-grained access to their metadata to third parties. It also introduces the SafeAnswer (SA) algorithm, a practical way of protecting the privacy of metadata at an individual level: The SA module processes raw metadata to compute relevant metrics within the safe environment of the PDS. In this way only a processed result is returned to the third parties, that can thus access only aggregated or processed information.

OpenPDS is composed of three main blocks:

- Database: metadata are stored in a CouchDB database[18], a NoSQL store that provides built-in functionalities to reduce the dimensionality of the metadata.
- PDS front-end: composed of several SA modules. All the SA access to the database must be authorized, and each SA module executes inside a sandbox. Only processed data (the so-called safe answers) leave the SA module, so third parties get the minimal amount of information they need and not additional metadata that could harm users' privacy.
- Data requester: any application or website that want to access user personal information.

Issues: Although OpenPDS may be a potential standard solution for personal metadata management, it still faces a number of challenges. The development of SafeAnswer privacy-preserving techniques at an individual level hardly scales for high-dimensional and ever-evolving data. Moreover, the development or adaptation of privacy, preserving data-mining algorithms to an ecosystem consisting of distributed PDSs is still an open issue.

Mydex

The Mydex platform[19] is a commercial tool that provides its users with the Mydex Trusted Framework and Platform, which enables to define single Personal Data Stores. Each PDS allows the collection, the management and the distribution of personal data, that are generally stored in the cloud, where the majority of the operations are performed as well. The Personal Data Store is an independent collection of files that are encrypted using a private encryption key known only by the individual. The data can be accessed uniquely by the owner user and the third party with a permission.

Mydex offers its PDS and additional tools free of charge. The user has full control over his data and can decide what kind of information to insert in the PDS. The user can decide which people or organizations can access and define connections, that enable to send or receive data to or from others.

[17] https://openpds.media.mit.edu/
[18] https://couchdb.apache.org/
[19] https://mydex.org/

Issues: Mydex is not an open-source platform, and, as such, it is not possible to run new installations of the PDS, neither to modify nor extend it to implement customization or additional functionalities.

Hub of All Things

Promoted by HAT Community Foundation, the Hub of All Things (HAT) microserver is a scalable technology to enhance the digital rights of users through the ownership of a personal data server. The HAT microserver is hosted in the cloud and individuals can install plugins to bring their data in from the Internet, exchange data with applications and install tools in their microservers to have private analytics for insights into their data.

A Hat microserver consists of four main components:

- HAT Web Server: each HAT microserver offers web-based APIs, which are customizable by the users.
- HAT Database: users own a HAT Database and they have complete control over the contents of the database. Each HAT Database contains a data schema, allowing to store individual's data from any source without losing the structure specific to the source. The Database is characterized by namespaces, that identify, which group data just like folders in normal operating systems.
- File Storage System: files are held in the Amazon S3 storage system offered by AWS and managed by Dataswift.
- HAT Computation: this component enables the creation of private analytics. HAT Computation provides an environment where third-party code written in different languages can operate on HAT data.

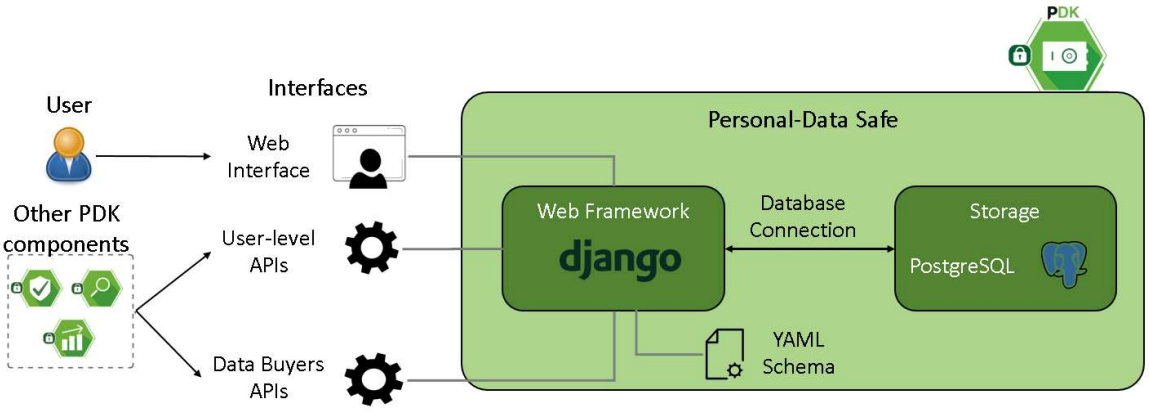
Issues: The Hub of All Things, despite being an open-source and flexible solution, also has a few drawbacks. In particular, the extreme customization offered to the users makes it difficult to achieve a minimum data standardization. This is an issue in particular to achieve a transparent data market, in which data buyers shall buy structured user data, which can be used for automatic analyses, data mining and business intelligence. Indeed, we aim at creating a solution that can help users in exchanging their data transparently in a consistent and interoperable fashion. Moreover, it lacks all the surrounding tools fundamental to build a fully-fledged PIMS, e.g., a consent manager or data market. Filling this gap is the goal PIMCity through the PDK modules.

6.3.- Technical Design

In this section, we describe the main design choices that have been evaluated for developing the PIMCity P-DS. Indeed, different technical solutions can be exploited for the various aspects of the P-DS, such as the web interface, data storage, etc. Moreover, in this section, we also describe the external interfaces of the P-DS, in terms of user interface and APIs for automatic operations. The PIMCity P-DS is a system which decouples data presentation from the operation and storage. The backend of the system implements the data model and takes care of handling the data, authenticating the users. The frontend provides a simple user interface to allow the user to perform necessary management actions on the stored personal information. Plus, the system provides APIs to allow other components to automatically access data. Indeed, we want the P-DS to be flexible and interoperable, and, like most of the modern IT systems, will be reachable through the network, with a Web interface and network APIs. To improve the modularity and flexibility

of the tool, users shall be authenticated *either* locally on the P-DS *or* by external providers using authorization token based on the standard OAuth[20] or JWT[21] mechanisms. Both ways will be considered.

The overall architecture of the P-DS is depicted in the figure below. The operation of the building blocks is explained in detail in the next sections.



6.3.1.-Internal Operation

Here, we describe the P-DS backend, while the interfaces for the users and external APIs are described in the next section. The backend is composed of a web component which handles the access to the P-DS data and a database which stores the data in a reliable and scalable way.

Web framework

As mentioned previously, we want the P-DS to be reachable through the network to allow universal access from any user, location and device. To this end, we opt for using a web framework to ease the development of all network-based P-DS components. Today, web frameworks are very common and popular, and they are also used for big and complex applications with thousands of users. They offer useful built-in functionalities (security, authentication, logging, auditing, etc.) and can speed up the implementation as they generally come as middlewares that connect the high-level language used by the developer with the underlying storage layer. This allows the developer to avoid writing boilerplate code to access data.

We have evaluated all the web frameworks available in the literature to find the most appropriate for our application. For each candidate web framework, we have evaluated its ease of use, the implemented features, its scalability and security as well as how it integrates with the other project modules. We have taken into account Django[22], Flask[23] and Spring[24].

In the PIMCity P-DS, we finally opt to use ***Django***, a high-level, open-source, full-stack web framework written in Python. It is used by many popular sites and applications such as

[20] https://oauth.net/2/
[21] https://jwt.io/
[22] https://www.djangoproject.com/
[23] https://flask.palletsprojects.com/en/1.1.x/
[24] https://spring.io/

Mozilla, Instagram, The Washington Post, and many others. Django was released publicly as a BSD license in July 2005, and follows a Model-View-Controller (MVC), offering a huge set of functionalities. The framework follows the "batteries included" philosophy and provides almost everything developers might want to do "out of the box", such as a robust Object Reference Model (ORM), an admin panel and user authentication. Django also implements state-of-the-art security features, providing a secure way to manage user accounts and passwords, avoiding common mistakes like directly storing passwords rather than a password hash. It also enables protection against many vulnerabilities by default, including SQL injection, cross-site scripting, cross-site requests.

Data storage

The main goal of the P-DS is to be a secure repository for user personal information, so it is fundamental to choose the nature of the data storage component carefully. We evaluate the usage of both standard SQL databases as well as the NoSQL approach, in which the Database Management System abandons the use of the SQL language and prefer simpler APIs. In both cases, we aim at finding solutions that enable the flexible insertion and the processing of data that do not follow a fixed and rigid schema, very useful in the case of applications that need to evolve rapidly or that deal with mutable data.

To make the final choices, we evaluated all the state-of-the art SQL and NoSQL databases, comparing them in terms of flexibility, ease of use, scalability. Moreover, we considered how they integrate with the selected Web framework, giving priority to couples of web framework and database for which has been proved a solid and reliable compatibility. We evaluated as potential candidate SQL database MySQL[25], MariaDB[26] and PostgreSQL[27]. As candidates for NoSQL databases, we considered MongoDB[28], HBase[29], Couchbase[30] and Redis[31].

We finally opt to use ***PostgreSQL***, an open-source SQL database. Born at the University of Berkeley in the 80s with the name of INGRES, it was conceived to support flexible and extensible data types. Nowadays PostgreSQL is widely used in commercial deployments and it perfectly integrates with Django, our choice for the web framework. Indeed, PostgreSQL has a number of features which are not shared by the other databases Django supports. This optional module contains model fields and form fields for a number of PostgreSQL specific data types. In particular, it supports the so called `JSONField`, a field that holds JSON encoded data which is not bound to a particular schema. This is fundamental in the PIMCity P-DS, successfully fulfilling our requirement of flexibility in the storage of Personal Information. Moreover, PostgreSQL is fully deployable in multi-node and fault-tolerant scenarios, allowing robust high-availability applications.

Data Model

The P-DS configuration is based on a schema, that is stored as a YAML file and contains the possible type of information that can be stored, listing all the possible fields for each

[25] https://www.mysql.com/
[26] https://mariadb.org/
[27] https://www.postgresql.org/
[28] https://www.mongodb.com/it
[29] https://hbase.apache.org/
[30] https://www.couchbase.com/
[31] https://redis.io/

type group. The schema has the primary goal of controlling the data that can be inserted on the Personal Data Safe, in order to prevent the user from inserting any data, that may be too complex to handle and process, or just erroneous. On the other side, the schema can be easily changed or expanded, providing the flexibility required by the module goals. Examples of fields that can be inserted in the schema are personal information such as name or phone number, as well as specific data like browsing and location history. Fields are clustered in **macro groups**, which represent semantic sets of fields organized in a hierarchical structure. We report an example schema in the figure below, in which a field group called "personal-information" defines two string fields called first-name and last-name and date field called "birth-data".

```
name: ''PIMCity default P-DS schema''
version: 0.1
author: ''John Doe''
content:
- group-name: personal-information
  types:
  - name: first-name
    type: string
  - name: last-name
    type: string
  - name: birth-data
    type: date
```

The P-DS will come with a **default schema** covering most of the PIMCity and EasyPIMs use cases. The default schema will contain three macro groups, namely **Personal Information, Location History and Browsing History**. Further customization will however be allowed.

6.3.2.-Interfaces

The PIMCity P-DS includes two types of interfaces for external access. First, a web interface allows users to interact with the P-DS directly. Second, web APIs allow automatic access by other systems, helping in obtaining a fully distributed system. The web APIs also provide functionalities to interface with the Data Portability module for portable data transfers between systems. See Deliverable 4.1 for details on the Data Portability.

Web interface

The P-DS is the means the individual can use to store her personal information and, as such, offers a web interface, that enables to view, organize and possibly update or insert data. The user interface is composed by an HTML-based website, that leverage Django templates, the Bootstrap library[32] for the graphic aspect and JavaScript code for all the client-side dynamic parts. All the elements are designed to be as dynamic as possible, in order to have a structure that can evolve based on the P-DS the schema configuration.

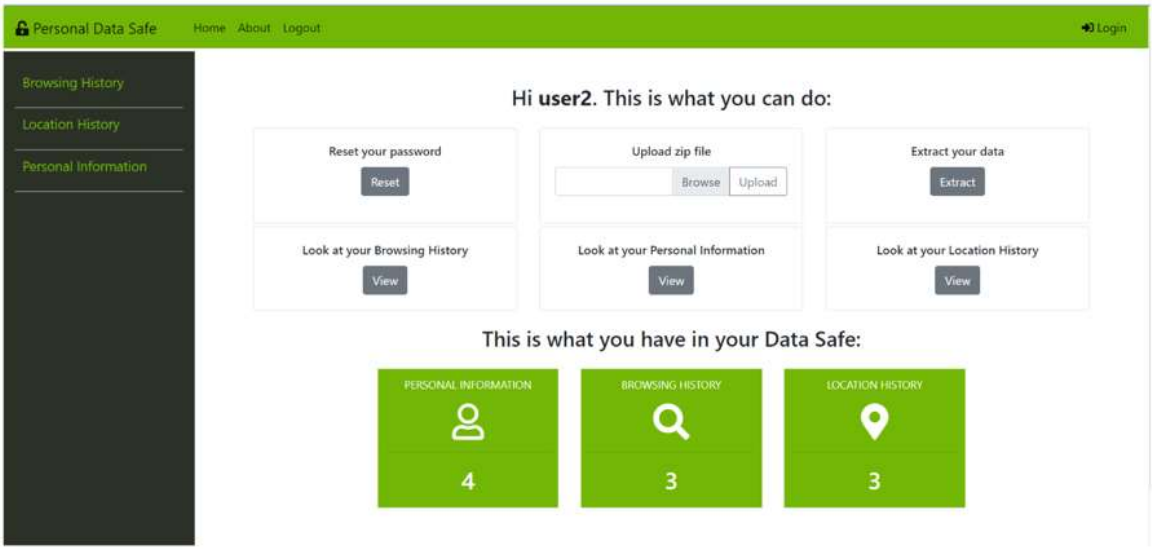
The main actions a user can perform are:

- *Add (or update) personal information:* the user can insert new data entries or modify the existing ones. In order to avoid insertions not compliant with the schema, controls on type consistency are applied on the server-side.

[32] https://getbootstrap.com/

- *Delete personal information*: the user can select a subset of information and delete them.
- *Data Extraction*: the user has the right to get a copy of her data, the user can download a zip file, which contains the stored data entries in JSON format.
- *Visualization*: the user can view the stored data entries, by applying flexible filters (for example by date in case of entries with time information)

Below, we provide a simple mock-up sketching how the user interface may look like. Once logged-in, the user may visualize, insert, modify, add or delete the personal information stored on the P-DS.



Web APIs

The P-DS functionalities are exposed to external components through REST APIs. REST is the acronym of REpresentational State Transfer and represents a programming approach that enables the building of distributed systems, achieving properties such as scalability, possibility of evolving, efficiency, and resilience. The REST paradigm is based on the client-server model. It has a stateless approach, in which any request from a client to the server must contain all of the information necessary to be understood. The server is composed by one or more services, each of which manages a set of information, that have a globally unique name (URI), can have multiple equivalent representations (generally JSON) and support CRUD operations (Create Read Update Delete). The URLs' names follow a uniform convention, and it fits well with the HTTP protocol, whose verbs support a CRUD environment natively, as in the following list:

- Create → POST.
- Read → GET.
- Update → PUT.
- Delete → DELETE.

We design **two sets of APIs**, one to be used on behalf of the users to manipulate the personal information stored in the P-DS, and the other to be used by data buyers or by any system authorized to retrieve users' data. Indeed, using the REST APIs, any PDK component or application on behalf of the **user** may store new data, which must comply

with the P-DS schema. With these APIs, it is also possible to modify or delete existing data. The second set of APIs is reserved for **data buyers**, which have the possibility of retrieving users' data upon receiving their consent from a P-CM. The P-DS will verify the consent presented at request-time by the data buyer (or any system on behalf of it) and will provide the requested data in the response. Suitable security mechanisms will be designed at the project-level to enforce the desired sharing policies. We will use certificate and signature-based security mechanisms to allow data sharing to data buyers.

6.4.- Configuration and Use Configuration and Use

Installation

The Personal-Data Safe is a Django application that can be installed by simply cloning the official GitLab repo, available at https://gitlab.com/pimcity/wp2/personal-data-safe. As such, it has a few Python dependencies listed below. For each package, we report the name and the version:

```
asgiref==3.2.7
bson==0.5.8
certifi==2020.6.20
cycler==0.10.0
dataclasses==0.6
Django==2.2.12
django-bootstrap-base-template==0.0.2
django-cors-headers==3.2.1
django-extensions==3.0.3
django-jsoneditor==0.1.6
django-leaflet==0.26.0
django-showurls==1.0.1
django-sql-explorer==1.1.3
djangorestframework==3.11.0
djongo==1.3.3
drf-extensions==0.6.0
geographiclib==1.50
geopy==2.0.0
jsonfield==3.1.0
kiwisolver==1.2.0
matplotlib==3.3.2
numpy==1.19.2
packaging==20.4
Pillow==7.2.0
psycopg2==2.8.5
pymongo==3.7.2
pyparsing==2.4.7
python-dateutil==2.8.1
pytz==2020.1
PyYAML==5.3.1
six==1.14.0
sqlparse==0.2.4
unicodecsv==0.14.1
```

To run the P-DS it is necessary use a PostgreSQL and configure the access hostname, port and credentials in `the personal_data_safe/settings.py` configuration file. To install, configure and run a PostgreSQL from scratch, please refer to the official documentation available online.[33] Before running the application, it is necessary to create the needed collections in the database with these commands.

```
python manage.py makemigrations
python manage.py migrate
python manage.py createsuperuser
```

Finally, it is possible to run the server with:

```
python manage.py runserver
```

Web Interface

The P-DS can be accessed with a web interface, composed by a set of pages that allow users to:

- See and manage data separately by macro groups. Notice that Macro Groups clusters Personal Information related to the same field. For example, Name, Surname, Gender may belong to the Macro Group "Person Details".
- Visualize in the home page statistics about the personal information stored into the system, such as the number of entries for each macro group or the last update.
- Change the password and run other utility functionalities.

Web API

The P-DS exposes JSON Web API to allow Data Subjects (users) and Data Buyers to access the data, provided with the required consent. We here report the list of Web APIs and their description in an easy-to-read format. The same documentation can be accessed in the standard OpenAPI format online at the URL: https://gitlab.com/pimcity/wp5/open-api/-/blob/master/WP2/personal-data-safe.yml

7.- Conclusions

This deliverable presented the final design of the following PIMCity PDK modules:

- **Personal Consent Manager (P-CM)**, from Task 2.1, the means to define once and for all the user's privacy preferences for consent management.
- **Personal Privacy Metrics (P-PM)**, from Task 2.2, easy to understand novel privacy metrics.
- **Personal Privacy Preserving Analytics (P-PPA),** from Task 2.3, controlling which data users are exposing
- **Personal Data Safe (P-DS),** from Task 2.4, the means to store personal data in a controlled form.

Along with the final design, we release the initial implementations of these modules, that are available online in the PIMCity GitLab repository. The current modules are currently under testing and we will release their final implementation in the upcoming Deliverable 2.3.

[33] https://www.postgresql.org/docs/9.3/index.html

8.- References

[1] *Regulation (EU) 2016/679 of the European Parlament and of the Council,* 2016.

[2] D. Fernandez, A. Futoransky, G. Ajzenman, M. Travizano and C. Sarraute, *Wibson Protocol for Secure Data Exchange and Batch Payments},,* 2020.

[3] H. Metwalley, S. Traverso, M. Mellia, S. Miskovic and M. Baldi, "Crowdsurf: Empowering transparency in the web," *ACM SIGCOMM Computer Communication Review,* vol. 45, no. 5, pp. 5--12, 2015.

[4] A. Karaj, S. Macbeth, R. Berson and J. M. Pujol, "WhoTracks. Me: Shedding light on the opaque world of online tracking," *arXiv preprint arXiv:1804.08959,* 2018.

[5] S. Englehardt and A. Narayanan, "Online Tracking: A 1-Million-Site Measurement and Analysis," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, Association for Computing Machinery, 2016, p. 1388–1401.

[6] V. Rizzo, S. Traverso and M. Mellia, "Unveiling Web Fingerprinting in the Wild Via Code Mining and Machine Learning," 2020. [Online]. Available: https://www.pimcity-h2020.eu/publication/unveiling-web-fingerprinting-in-the-wild-via-code-mining-and-machine-learning/.

[7] S. Matic, C. Iordanou, G. Smaragdakis and N. Laoutaris, "Identifying Sensitive URLs at Web-Scale," 2020. [Online]. Available: https://laoutaris.info/wp-content/uploads/2020/09/imc20.pdf.

[8] P. Ohm, "Broken promises of privacy: Responding to the surprising failure of anonymization," *UCLA Law Review,* p. 1701, 2010.

[9] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *2008 IEEE Symposium on Security and Privacy*, 2008, pp. 111-125.

[10] P. Samarati, "Protecting respondents identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering,* vol. 13, no. 6, pp. 1010-1027, 2001.

[11] K. LeFevre, D. J. DeWitt and R. Ramakrishnan, "Incognito: Efficient Full-Domain K-Anonymity," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, Baltimore, Maryland, Association for Computing Machinery, 2005, p. 49–60.

[12] K. LeFevre, D. J. DeWitt and R. Ramakrishnan, "Mondrian multidimensional k-anonymity," in *22nd International conference on data engineering (ICDE'06)*, IEEE, 2006.

[13] A. Machanavajjhala, D. Kifer, J. Gehrke and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," *ACM Transactions on Knowledge Discovery from Data,* vol. 1, no. 1, pp. 3-es, 2007.

[14] N. Li, T. Li and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *2007 IEEE 23rd International Conference on Data Engineering*, 2007, pp. 106-115.

[15] C. Dwork, F. McSherry, K. Nissim and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*, Springer, 2006, pp. 265--284.

[16] N. Jha, T. Favale, L. Vassio, M. Trevisan and M. Mellia, "z-anonymity: Zero-Delay Anonymization for Data Streams," to appear in *2020 IEEE International Conference on Big Data (Big Data)*, 2020.

9.- Annex

In the appendix we put the status of the software repositories as available at the time of finalizing this document

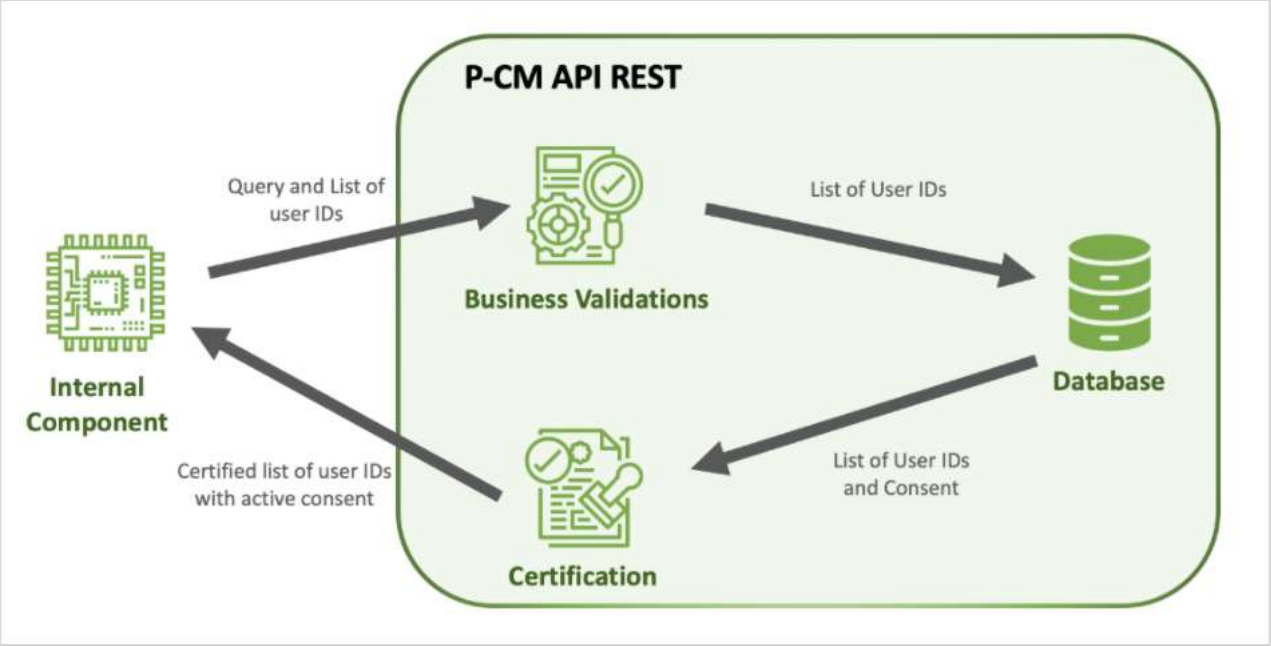
README.md 2.86 KB

Personal Consent Manager

Introduction

The primary objective of the Personal Consent Manager (P-CM) is to give the users the transparency and control over their data in a GDPR compliant way. That is, give them the possibility to decide which data can be uploaded and stored in the platform, as well as how (raw, extracted or aggregated) data can be shared with Data Buyers in exchange for value when the opportunity arises.

The P-CM is presented as a web application and a REST API, not only providing users the possibility to use the component in a user-friendly way, but also enabling developers to integrate PIMCity Consent Management capabilities in their products. The architecture of the PDK is depicted in the figure below.



Installation

The documentation and the following instructions refer to a Linux environment, running with **Docker Engine v20.10.x** and **Docker Compose: v1.27.x**. The P-CM project has been cloned from this GitLab repository.

Follow accurately the next steps to quickly set-up the P-CM backbone on your server. All relevant steps are designed for a Linux machine, perform the equivalent procedure with other environments.

1. Prepare the environment:

```
> sudo apt-get update
> sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
> curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring
> echo \
  "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
> sudo apt-get update
> sudo apt-get install docker-ce docker-ce-cli containerd.io
```

2. Import the project from the GIT repository:

```
> git clone https://gitlab.com/pimcity/wp2/personal-consent-manager.git
```

Execution

To run the P-CM service, the following command should be executed at the root of the repository:

```
> docker-compose up
```

Configuration

The P-CM can be configured in three similar ways:

1. Using environment variables.
2. Defining those environment variables in a file called ".env" at the root directory of the folder with the PDK deployed, declaring them in the same way a variable is defined in the UNIX shell. You can check this example.
3. Modifying docker-compose.yml file.

Usage Examples

You can check here the Swagger OpenAPI definition to see how the API is defined and used. Examples are provided as well.

License

Updates README.md with architecture picture.
Stefano Traverso authored 3 days ago

05d6012d

README.md 3.32 KB

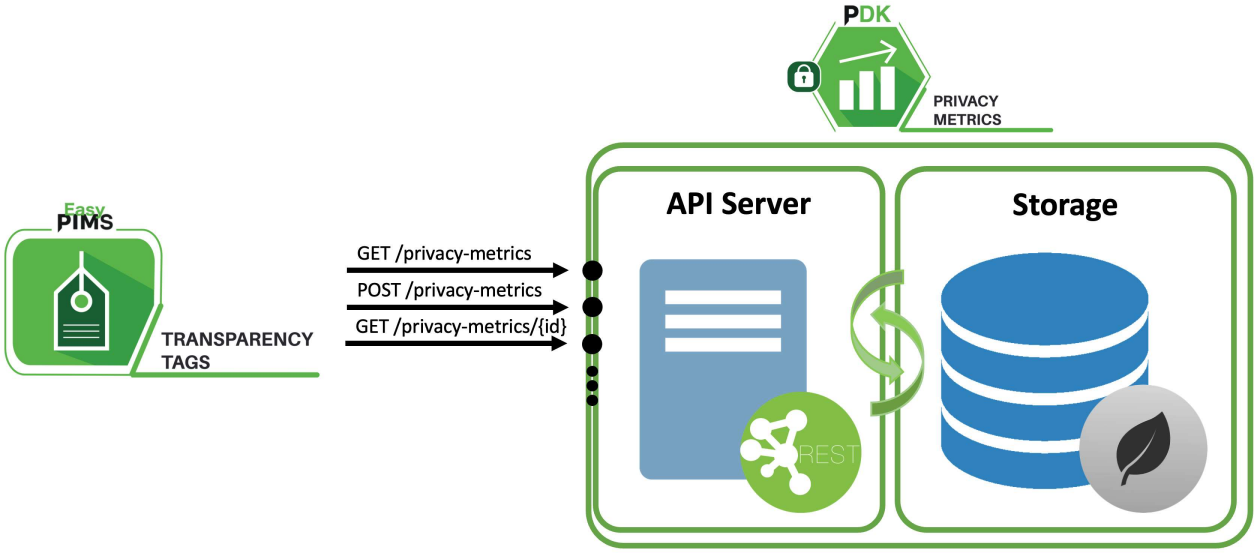
Privacy Metrics

code style black

Introduction

Privacy Metrics represent the means to increase the user's awareness. This component collects, computes and shares easy-to-understand data to allow users know how a service (e.g., a data buyer) stores and manages the data, if it shares it with third parties, how secure and transparent it looks, etc. These are all fundamental pieces of information for a user to know to take informed decisions. The PM computes this information via a standard REST interface, offering an open knowledge information system which can be queried using an open and standard platform. PMs combine information from supervised machine learning analytics, services themselves and domain experts, volunteers, and contributors. The Open API implementation of Privacy Metrics component is available at official PIMCity's Gitlab code repository, under WP5 project: https://gitlab.com/pimcity/wp5/open-api/-/blob/master/WP2/privacy-metrics.yml.

In this repo we provide the implementation of the backend offering authenticated access to PMs. It builds on MongoDB (for the database), Python/Flask and Swagger (for the server). The repo builds on Poetry for the management of Python packaging and dependencies.



Installation

Intall Python3 using this guide.

Install Poetry using this guide.

Install MongoDB using this guide.

On macOS, we recommend to install above software with Brew.

```
# Install dependencies
poetry install
```

Usage

Start MongoDB and import data

```
# Start MongoDB
brew services start mongodb/brew/mongodb-community

# Import data
mongoimport --db testdb --collection privacy_metrics --drop --file data/privacy_metrics.json --jsonArray
```

Run the API server

```
# Run the API server
poetry run python3 -m privacy_metrics.main
```

Open your browser on http://localhost:8080/privacy-metrics/ui. This page provides the Swagger UI describing Privacy Metrics APIs.

Dev Setup and Testing

Dev Setup

```
# Setup pre-commit and pre-push hooks
poetry run pre-commit install -t pre-commit
poetry run pre-commit install -t pre-push
```

Testing

To quickly run tests in the same environment you use for developing, you can directly invoke pytest:

```
# Install package in "editable" mode (plus dependencies).
# This is required because, as a python best practice,
# tests must always be executed against an installed package.
poetry install

# Run tests
poetry run pytest

# Run tests, with code coverage check
poetry run pytest --cov
```

tox

tox is configured to run both linting and tests in dedicated virtual environments. This ensures running tasks in a clean environment. Also, tox automatically creates a source distribution (`sdist`) of the python package and installs it before running tests.

```
# Run linting and tests with tox
poetry run tox

# Run linting and tests with tox, also recreating virtual environments from scratch
poetry run tox --recreate
```

Credits

This package was created with Cookiecutter.

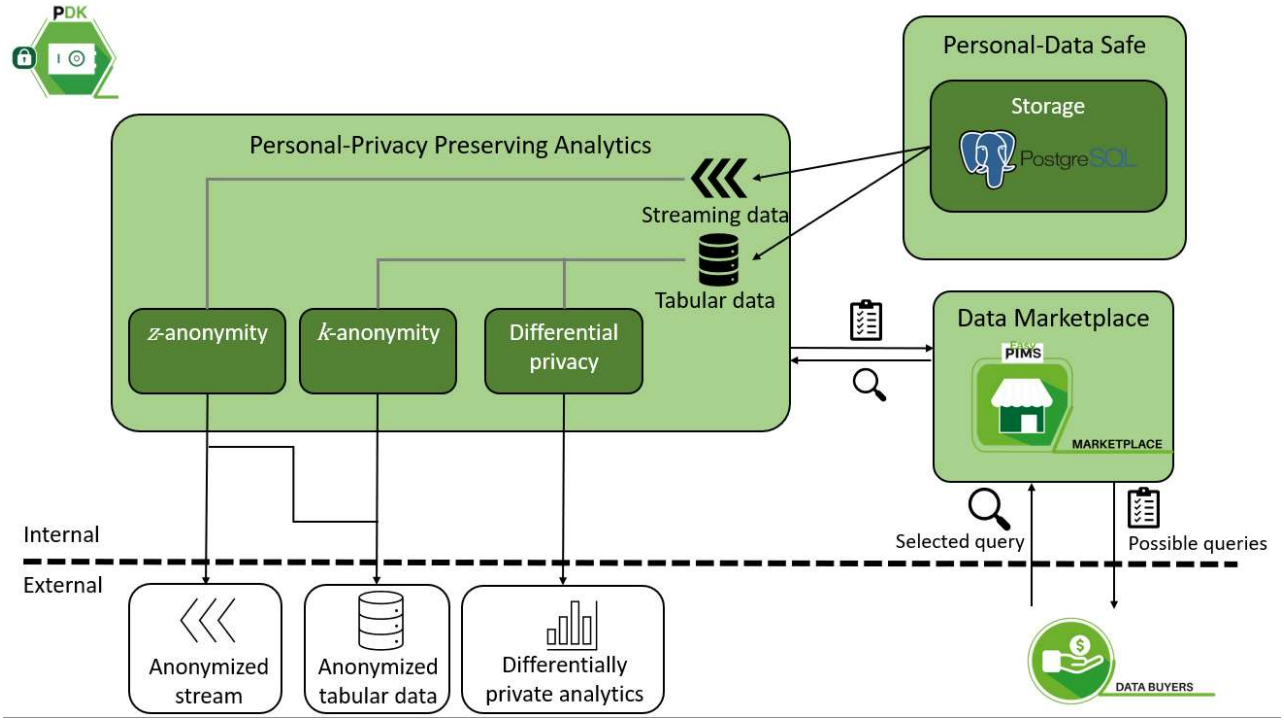
README.md 5.95 KB

Personal Privacy Preserving Analytics (P-PPA)

Intro

The **Personal Privacy Preserving Analytics (P-PPA)** module has the goal of allowing data analysts and stakeholders to extract useful information from the raw data while preserving the privacy of the users whose data is in the datasets. It leverages concepts like Differential Privacy and K-Anonymity so that data can be processed and shared while guaranteeing privacy for the users

P-PPA includes a set of functionalities that allow perform data operations preserving the major privacy properties: *k-anonymity*, *z-anonymity*, *differential privacy*. P-PPA is capable to handle different sources of data inputs, that define which kind of privacy property is called into account: we have design solutions for tabular and batch stream, handled with *PostgreSQL*, *MongoDB*, and *CSV* modules, and live stream data. The figure below depicts the P-PPA architecture.



Installation

P-PPA is a Python3 module that requires the following libraries:

```
numpy==1.18.5
diffprivlib==0.3.0
matplotlib==3.3.3
pandas==1.1.5
psycopg2==2.8.6
pymongo==3.11.2
SQLAlchemy==1.3.20
requests==2.24.0
Flask_RESTful==0.3.8
Flask==1.1.2
PyYAML==5.4.1
```

Usage Examples

Please take in mind that the input data need to be in pandas.Dataframe format; in these examples, it is supposed to be already present. The chosen datasets for these examples is "Adult", available in "data_manage/data_samples" folder. "Credit" and "Diabetes" datasets are also available in the same folder

Following there are some usage examples:

1. K-anonymity, performing Mondrian algorithm.

Creating the *Mondrian* class with just the k parameter: all column attributes will be taken into account to perform the Mondrian algorithm.

```
from algorithms.kanonymity.mondrian.mondrian import Mondrian
mondrian = Mondrian(3)
k_anonymized_dataframe = mondrian.perform(input_dataframe)
```

2. K-anonymity, without performing Mondrian algorithm.

Differently from the previous example, here are selected the column indexes 2, 3, 4 and 5, corresponding to *fnlwgt*, *education*, *education-num* and *marital-status* attributes. Having specified these columns, the Mondrian algorithm will not be performed (obviously for this particular data and columns. For further details please check *"perform"* and *"_check_kanon_from_columns"* class methods documentation in the *"mondrian.py"* module).

```
from algorithms.kanonymity.mondrian.mondrian import Mondrian
mondrian = Mondrian(3, user_choice_index=[2,3,4,5])
k_anonymized_dataframe = mondrian.perform(input_dataframe)
```

3. K-anonymity, select QIs and whitelist columns.

In this example the parameters "*qi_indexes*" and "*whitelist*" are used to select respectively:

- which columns need to be considered as quasi-indetifiers, selected to achieve k-anonymity
- white-list columns, selected to be displayed among transformed attributes but conserved untouched as they are (they don't contribute to k-anonymity conversation: their use is related to performing anonymization on some attributes and conserving one o more possible labels (the white-listed attributes) to extract statistics and be able to train machine learning models).

Some more details:

- if "*qi_indexes*" isn't specified, all columns are considered as quasi-identifiers.
- if a column index in the "*whitelist*" list is also present in "*qi_indexes*" one, it's behavior will be overwritten and treated as white-listed.

Note: this to parameters have been introduced for an administrator use.

```
from algorithms.kanonymity.mondrian.mondrian import Mondrian
mondrian = Mondrian(3, qi_indexes=[1,2,4,6], whitelist=[8,9])
k_anonymized_dataframe = mondrian.perform(input_dataframe)
```

4. Differential privacy, performing the mean of the first and third columns.

For further details please check *"Dp_IBM_wrapper"* class documentation in the *"dp_IBM_wrapper.py"* module.

```
from algorithms.differential_privacy.dp_IBM.dp_IBM_wrapper import Dp_IBM_wrapper
mean = Dp_IBM_wrapper([0,2], "mean", 0.6)
ret_mean = mean.perform(input_dataframe)
```

5. Differential privacy, performing the histogram on the first column.

For further details please check *"Dp_IBM_wrapper"* class documentation in the *"dp_IBM_wrapper.py"* module.

```
from algorithms.differential_privacy.dp_IBM.dp_IBM_wrapper import Dp_IBM_wrapper
histogram = Dp_IBM_wrapper([0], "histogram", 0.6)
hist, bins = histogram.perform(input_dataframe, bins=6)
```

This is the same output that you would obtain exploiting numpy library. The following code it's just an example of a way with which you can use this output.

```
from matplotlib import pyplot as plt
width = 0.7 * (bins[1] - bins[0])
center = (bins[:-1] + bins[1:]) / 2
plt.bar(center, hist, width=width)
plt.show()
```

6. Differential privacy, performing the 2d histogram on the first and the third columns.

For further details please check *"Dp_IBM_wrapper"* class documentation in the *"dp_IBM_wrapper.py"* module. Please take in mind that the histogram from the first and the third column of *"Adult"* dataset has no semantic meaning.

```
from algorithms.differential_privacy.dp_IBM.dp_IBM_wrapper import Dp_IBM_wrapper
histogram2d = Dp_IBM_wrapper([0,2], "histogram2d", 0.6)
matrix2d, xedge, yedge = histogram2d.perform(input_dataframe)
```

This is the same output that you would obtain exploiting numpy library.

License

The P-PPA are distributed under AGPL-3.0-only, see the `LICENSE` file.

Update README.md
Martino Trevisan authored 4 days ago

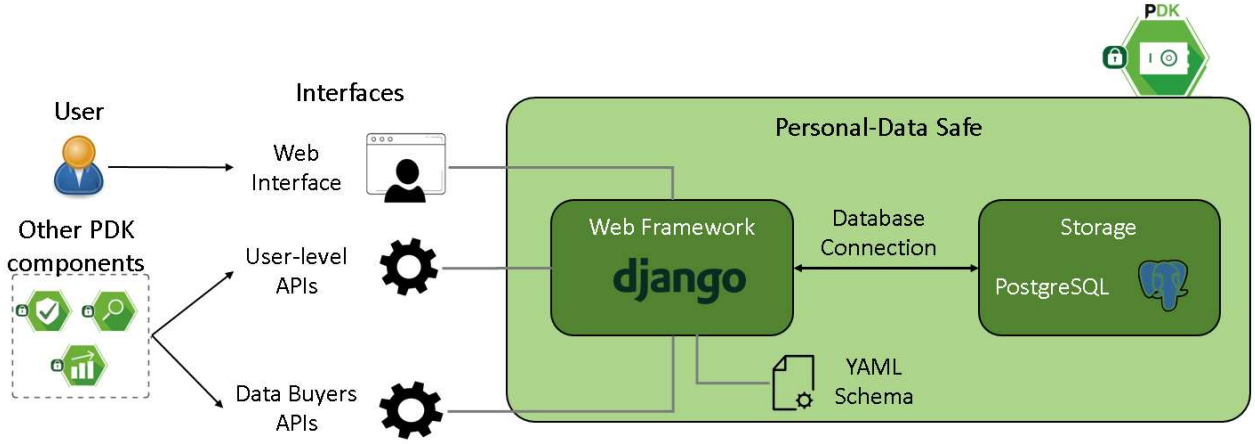
e5c59918

README.md 17 KB

Personal Data Safe

Intro

The Personal Data Safe (P-DS) is the means to store personal data in a controlled form. It implements a secure repository for the user's personal information like navigation history, contacts, preferences, personal information, etc. It gives the possibility to handle them though REST-based APIs or a web interface. Thanks to the REST APIs, the P-DS can be accessed also by other components of the PDK. The architecture of the PDK is depicted in the figure below.



Installation

The documentation and the following instructions refer to a Linux environment, with **Python 3.8.2** and **pip 20.0.2** installed. The P-DS project has been cloned from this GitLab repository.

Follow accurately the next steps to quickly set-up the PDS backbone on your server. The package comes with a frontend and a backend already implemented, but if the needs calls only for a ready API you can cancel the folder '/frontend' and skip the relative steps. All relevant steps are designed for a Linux machine, perform the equivalent procedure with other environments.

First Steps

Update all packages:

```
sudo apt-get update sudo apt-get upgrade sudo apt-get dist-upgrade
```

Install pip/pip3:

```
sudo apt-get python3-pip
```

Import the project from the GIT repository:

```
git clone https://gitlab.com/pimcity/wp2/personal-data-safe.git
```

Backend setup

Now enter the backend folder:

```
cd backend/
```

Create a python virtual environment and activate it:

```
python3 -m venv venv source venv/bin/activate
```

Install the python-dev library and also add wheel:

```
sudo apt-get install libpq-dev python-dev pip install wheel
```

Install all requirements:

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

Now we need to set-up the database, for this project the default one is PostegreSQL. If there is no need to change database, then the app will be ready to use after the next steps. Instead, if another database is needed, check the django documentation here to find out what steps to follow.

- First install PostegreSQL:

```
sudo apt-get update sudo apt-get install python-pip python-dev libpq-dev postgresql postgresql-contrib
```

- Create a database and database user. The default settings are:

```
db_name = pds_postgres db_username = admin user_password = admin_secret_password
```

- The steps to create the new database are the following:

```
sudo su - postgres psql CREATE DATABASE <db_name>; CREATE USER <db_username> WITH PASSWORD '<user_password>'; ALTER ROLE <db_username> SET client_encoding TO 'utf8'; ALTER ROLE <db_username> SET default_transaction_isolation TO 'read committed'; ALTER ROLE <db_username> SET timezone TO 'UTC'; GRANT ALL PRIVILEGES ON DATABASE <db_name> TO <db_username>; \q exit
```

- It is now possible to use the database using the user and credentials registered:

```
psql --host <localhost or ip_addr> --port <port num> --username <db_username> <db_name>
```

Connect Django to the PostgreSQL

```
pip install django psycopg2
```

If you don't want to use the default username and password, you will need to change them in the settings in the project file `backend/config/settings.py` .

```
DATABASES = {
        'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': '<_db_name_>',
        'USER': '<db_username>',
        'PASSWORD': '_<user_password>_',
        'HOST': 'localhost',
        'PORT': '',
        }
}
```

Now Migrate the database. NB that it's better and safer to perform migrations for each app:

```
python manage.py makemigrations app_users python manage.py makemigrations app_personal_data python manage.py migrate
```

Create a Django SuperUser, which will also work as admin in the admin page:

```
python manage.py createsuperuser
```

It is now possible to run server and access admin:

```
python manage.py runserver 0.0.0.0:8000 admin found @ localhost:8000/admin
```

Frontend setup

This section focuses on the frontend setup of the application. The frontend is developed in javascript using the ReactJS framework. To begin, enter the frontend folder:

```
cd frontend/
```

First install NodeJs.

- Enable the NodeSource repository by running the following curl command as a user with sudo privileges:

```
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
```

- Once the NodeSource repository is enabled, install Node.js and npm by running:

```
sudo apt install nodejs
```

- Verify that the Node.js and npm were successfully installed by printing their versions:

```
node --version npm –version
```

Install dependencies for React:

```
cd Frontend npm install
```

In case of some error like `"This version of npm is compatible with lockfileVersion@1, but package-lock.json was generated for lockfileVersion@2. I'll try to do my best with it!”` try running this command:

```
sudo rm -rf node_modules package-lock.json && npm install
```

Run React Server:

```
npm run
```

The project should now be working. Activate concurrently the Django and the ReactJS server and the platform should be fully functional.

Test Data

If needed some scripts are already present for quickly creating some test data and check that the platform is correctly working.

Random users

To create random users use the script `create_random_user.py` . This script will generate a given amount of users, with a random username and a fixed password: `test_password` . Only need to keep track of the usernames created. Users will have some personal fields already randomly generated.

```
Usage: python create_random_user.py <int:number_of_new_users>

Example of usage: python create_random_user.py 1

Example of output: {'username': 'isabella_wilson'} {'id': 43280, 'value': {'birth-data': '1999-03-16'}, 'metadata': 'metadata-birth-data', 'data_type': 'birth-data', 'group_name': 'personal-information', 'description': 'description-birth-data', 'created': '2021-05-03T14:42:27.345054Z'}
```

Random browsing history

To generate random browsing history data for a specific user, use the script `create_random_browsing_history.py` . This script will generate a given amount of browsing history data, and assign them to a given user.

```
Usage: python create_random_browsing_history.py <str:username> <int:number_of_new_data>

Example of usage: python create_random_browsing_history.py isabella_wilson 2

Example of output: {'id': 43281, 'value': {'visited-url': {'url': 'http://www.msn.com', 'title': 'Msn', 'time': '2016-03-06 04:58:20'}}, 'metadata': 'metadata-visited-url', 'data_type': 'visited-url', 'group_name': 'browsing-history', 'description': 'description-visited-url', 'created': '2021-05-03T14:48:16.934886Z'} {'id': 43282, 'value': {'visited-url': {'url': 'http://www.onet.pl', 'title': 'Onet', 'time': '2018-08-29 03:49:59'}}, 'metadata': 'metadata-visited-url', 'data_type': 'visited-url', 'group_name': 'browsing-history', 'description': 'description-visited-url', 'created': '2021-05-03T14:48:16.941765Z'}
```

Random location history

To generate random location history data for a specific user, use the script `create_random_location_history.py` . This script will generate a given amount of location history data, and assign them to a given user.

Usage: python create_random_location_history.py <str:username> <int:number_of_new_data>

Example of usage: python create_random_location_history.py isabella_wilson 2

Example of output: {'id': 43286, 'value': {'visited-location': {'latitude': 46.05635313711818, 'longitude': 12.831298674353643, 'time': '2010-03-04 04:45:20', 'description': 'example'}}, 'metadata': 'metadata-visited-location', 'data_type': 'visited-location', 'group_name': 'location-history', 'description': 'description-visited-location', 'created': '2021-05-03T14:49:59.668810Z'} {'id': 43287, 'value': {'visited-location': {'latitude': 43.880077139481415, 'longitude': 15.797250794542236, 'time': '2012-07-28 10:55:07', 'description': 'example'}}, 'metadata': 'metadata-visited-location', 'data_type': 'visited-location', 'group_name': 'location-history', 'description': 'description-visited-location', 'created': '2021-05-03T14:49:59.675079Z'}

Usage

Data Model

The data model of the application consists of 2 main classes:

- User class
- PersonalInformation class

The User class is used to store the information about the single user and it extends the *AbstractUser* class provided by Django. The class does not modify much the parent class, since it doesn't need to store additional information. Each user is linked to a set of personal information; the information about this relationship can be found in the PersonalInformation class.

On the other side, personal information are modeled by the *PersonalData* class that is used to store arbitrary types of data, e.g., user static information (name, surname, year of birth, email, etc.), browsing history, location history. The *value* field of the PersonalData class is defined as a *JSONField* object; in this way, user can store both elementary data, such as int, string, etc., and more structured data, such as dictionaries, that are represented as JSON objects. The PersonalData class has also *user* attribute that is defined as a *ForeignKey* object, so that each PersonalData instance has a reference to the user that owns the entry. This solution emulates what is done in a classic relational system and it has been chosen because it proved to be the most efficient especially with big volumes of data. Others field of the PersonalData class provides additional information related such as:

- group_name: it's the semantic group the entry belongs to.
- metadata: it's the name that identifies a personal information inside a group
- type: it's the type of the information. Possible types are int, string, date, boolean, float, dict. Since each entry is modeled as a JSON object, elementary type information are stored in the form `{'type': value}`, while dict information in the form `{'subfield1': value1, 'subfield2', value2}`.

Schema

The schema is stored in the data safe file system and loaded at application initialization, specific functions ensure that the schema defined follows some basics guidelines that will be defined later in this paragraph. The whole project logic has been developed prioritizing ease of usage: the main goal is to only change/modify the schema for the whole application (backend and frontend) to adapt to the change. The schema defines the kind of information that the data safe can accept; each information is characterized by the following basic attributes in the schema:

- **group-name**: it defines the high level group that the data belongs to, such as *personal-information*, *browsing-history*, ect.
- **name**: it's the name that identifies the information inside a group, e.g. *birth-data* in the *personal-information* group
- **type**: it defines the type associated to the information, e.g. *birth-data* must be a Date

The schema is used to validate the input provided by the user, in order to control that the inserted data complies with the information that the P-DS can store. The schema can be updated if the user wants to store additional information: new types must be declared using the same name-type syntax used for old types, so that the application can handle changes in the schema.

The schema is used also to read data from the database: data that no longer match the schema are simply ignored, so that users can still have access to them (returning to an old version of the schema for example).

```
name: "PIMCity default PDS schema"
version: 0.2
author: "Federico Torta, Annaloro Enrico, Martino Trevisan"
content:
- group-name: personal-information (1)
  user-insertion: true (2)
  user-update: true (3)
  add-zip-file: false (4)
  extract-json: true (5)
  types: (6)
    - name: first-name
      type: string
    - name: last-name
      type: string
    - name: birth-data
```

```
      type: date
    - name: age
      type: int
- group-name: browsing-history (7)
  user-insertion: false
  user-update: false
  add-zip-file: true
  extract-json: true
  types:
    - name: visited-url
      historical: true (8)
      type: dict (9)
      fields: (10)
        - url: string
        - page-title: string
        - time: date
- group-name: location-history (11)
  user-insertion: false
  user-update: false
  visualization-hint: map (12)
  add-zip-file: true
  extract-json: true
  types:
    - name: visited-location
      historical: true
      type: dict
      fields:
        - latitude: float
        - longitude: float
        - description: string
        - time: date
```

Explanation

- Each *group-name* field represents a possible semantic high-level group. In this case the first group accepted by the P-DS is **personal-information** (1). The possible data that are considered part of the personal-information group are defined in the *types* field (6) which lists all the variants of a personal-information entry. The *types* field is basically a list of *name-type* pairs where *name* is the common name of the entry, while *type* is the actual type of the information. Each data is represented with the name and the type in order to let the P-DS store any kind of information, without being tied to a particular elementary type. The user can store any data he want but the P-DS will cast the inserted data to control the type compliance. Moreover each group can presents additional settings:
- *user-insertion* (2): if this field is true, the user will be able to add the information manually. The frontend will show an **Add** button that allows the user to directly insert new data from the page. Schema and type compliance controls are executed.
- *user-update* (3): if true, the user can update a P-DS entry from the UI, using an **Edit** button that allows to change just the value of the stored data.
- *add-zip-file* (4): if true, the user can add new data from a zip file, which contains a JSON file with the information to be inserted. The zip file is uploaded using an **Upload ZIP file** button of the UI. This function is just a prototype and aims to show the potentiality of the P-DS: the data import feature will be presumably used by automatic software systems that can create the correct JSON file, giving the possibility to import user data (even in big volumes) from other data stores.
- *extract-json* (5): if true, the user can click an **Extract data** button to download the stored data in JSON format, inside a zip file. As well as the data import functionality, also the data extraction can be performed both at global level (all the groups together) or at group-level (so download just the data related to a certain group).
- (8) and (11) means that the other possible groups that the P-DS supports are *location-history* information and *browsing-history* information.
- some information types can be stored multiple times, because they are linked to a particular timestamp, such visited urls or the visited locations. In this case they are enriched with the flag *historical*, so that the user can have multiple entries for that type.
- in order to let the user store more complex and structured data, the schema supports the *dict* type (10). A dict object is basically a JSON object with a set of key-value pair. Each entry of the dict is a piece of the complete information and is represented as a name-value pair just as the elementary entry of the P-DS. In this way the same controls can be applied recursively on the elements of a dict object. The single component of a dict are displayed in the *fields* key (11).
- The visualization-hint (12) is a special property used for visualization aids in the frontend. Some keywords are mandatory and additional set-up in the fields will be mandatory to support the feature. As of today the active features are:
  - `map` : allows to plot, in a google maps canvas, one or more points given its coordinates. For this reason, when this field is set, it must also be set up two mandatory fields (10) `latitude` and `longitude`, as shown in the example above. Failing to do so will prevent the application from starting.

- On the frontend, it has also been implemented a sorting logic to order items based on a specific field. To automate this we introduced a set of fields which support this feature, this means that, by using specific names for the fields, some special features (such as ordering) will be available. The fields are the following
    - `time: date` : ascending and descending sort based on date
    - `title: string` : ascending and descending sort based on alphabetical order

```
# This example can be sorted both by time and by title
- group-name: browsing-history
  user-insertion: false
  user-update: false
  add-zip-file: true
  extract-json: true
  types:
    - name: visited-url
      historical: true
      type: dict
      fields:
        - url: string
        - title: string # <-- here
        - time: date # <-- here
```

License

The P-DS is distributed under AGPL-3.0-only, see the `LICENSE` file.