

POSEIDON

Protection and control of Secured Information by
means of a
privacy enhanced Dashboard

GRANT AGREEMENT NUMBER: 786713
H2020-DS-2016-2017/DS-08-2017

Deliverable

Web-based Dashboard
Interim Implementation

Disclaimer

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 786713

This document has been prepared for the European Commission however it reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Requirement Definition and Threat Model

Project Title:	POSEIDON
-----------------------	-----------------

Deliverable Number:	D4.1
Grant Agreement number:	786713
Funding Scheme:	HORIZON 2020
Project co-ordinator name:	Francesco Paolo Schiavo – MEF (Italian Ministry of Economy and Finance)
Title of Deliverable:	Web-based Dashboard – Interim Implementation
WP contributing to the Deliverable:	WP4
Deliverable type	Document
Dissemination level	Public
Partner(s)/Author(s):	Document: PoSeID-on WP4 Members Annex I: E-Lex

History:			
Ver.	Comments	Date	Author
0.1	Initial layout and content	2019-07-01	Joris van Rooij - Jibe
0.2	More content	2019-07-03	Joris van Rooij - Jibe
0.3	Even more content	2019-07-04	Joris van Rooij - Jibe
0.4	Added correlation information	2019-07-11	Joris van Rooij - Jibe
0.5	More content	2019-07-18	Joris van Rooij - Jibe
0.6	Expanded API spec	2019-07-24	Joris van Rooij - Jibe
0.7	Added section about frontend	2019-07-25	Joris van Rooij - Jibe
1.0	Formatting, Accenture & UC review, added T&C Annex	2019-07-30	Joris van Rooij - Jibe

Project funded by the European Commission Horizon 2020 - The EU Framework Programme for Research and Innovation.

The POSEIDON Consortium consists of:

Table 1 - Consortium Partners











Logo	Name	Country
	MEF – Ministero dell'Economia e delle Finanze	Italy
	ACN - Accenture S.p.A.	Italy
	PNO – PNO Innovation	Spain
	ELEX – e-Lex Studio Legale	Italy
	TECN – Fundacion Tecnalia Research & Innovation	Spain
	SAN – Ayuntamiento de Santander	Spain
	SOFT - Softeam	France
	UC – Universidade de Coimbra	Portugal
	MITA - Malta Information Technology Agency	Malta
	JIBE – SMARTFEEDZ B.V.	The Netherlands

TABLE OF CONTENTS

1	Introduction	7
1.1	Document Structure	7
1.2	Requirements	7
1.2.1	Web-based	7
1.2.2	Accessible and Responsive	7
1.2.3	Performant and Scalable	8
1.2.4	Secure	8
1.2.5	Maintainable	8
1.3	Approach	9
2	Overall Architecture	10
2.1	Technology Stack	11
3	Authentication Provider	13
3.1	Authentication Flow	13
3.2	Cookie	14
3.3	Pluggable Authentication Providers	15
4	Web-based Dashboard Back-end	16
4.1	Architecture	16
4.2	Authentication	17
4.3	Blockchain API	17
4.4	Messages for Data Subjects	19
4.5	Data Processor API	19
4.5.1	Data Subject Correlation	20
4.6	Personal Data Analyser and Risk Management Module	21
4.7	Technology	21
5	Web-based Dashboard Front-end	22
5.1	Persona	22
5.2	User Stories	23
5.2.1	Authentication	23
5.2.2	Data Processor Listing	23
5.2.3	Permission Listing	24
5.2.4	Permission Request Listing	24
5.2.5	Permission Granting	24
5.2.6	Permission Revocation	24
5.2.7	PII Listing	24
5.2.8	PII Supplying	24
5.2.9	PII Updating	24

5.2.10	Right to Be Forgotten	24
5.2.11	PDA/RMM Permissions	25
5.3	Screens.....	25
5.3.1	Common	25
5.3.2	Dashboard.....	25
5.3.3	Messages	26
5.3.4	Data Processors.....	26
5.3.5	Help	26
5.4	User Experience Design.....	26
5.5	Technology	30
6	Integration Approach	31
6.1	Packaging.....	31
6.2	Configuration	32
6.3	Communication.....	32
6.4	Data Storage.....	32
7	Table of Figures	34
	Annex I – Terms and Conditions	35

1 Introduction

This document describes the interim implementation of the web-based dashboard component of the PoSeID-on project. How the project came to be, the choices that have been made, and how it has been constructed. The web-based dashboard is the component responsible for all interaction between data subjects and the platform. It does this, as the name suggests, using web technologies.

The interim implementation does not, however, cover all functionality needed, nor does it meet the quality standards required for mission-critical software. It is merely a snapshot of the progress towards the final product.

1.1 Document Structure

This is a mostly technical document, intended as a guideline for implementation.

Firstly, the introduction describes the general approach and the requirements. After that the architecture is described using a top-down methodology. The individual components each get their own chapter. Finally, the approach taken to integrate the web-based dashboard into the whole PoSeID-on system is described.

1.2 Requirements

Before any implementation can be done, a set of requirements needs to be defined for the implementation to follow. Instead of detailing every aspect of every component individually, an overarching framework has been chosen that sets guidelines and boundaries for the development process.

1.2.1 Web-based

First and foremost, the web-based dashboard is, as the name clearly suggests, web-based. This means that the only expectation the application can have of the user is that the user is accessing the application using a web-browser.

What web-browser this is should, in theory, not matter. However, in practice, this does make a very big difference in both development time and user experience. Since this is a preliminary implementation of the web-based dashboard, the following list of the latest versions of browsers is targeted:

- Google Chrome on Desktop
- Google Chrome on Android
- Mozilla Firefox on Desktop
- Mozilla Firefox on Android
- Apple Safari on Desktop
- Apple Safari on iOS
- Microsoft Edge on Desktop

1.2.2 Accessible and Responsive

The PoSeID-on project aims to create a platform for *every European citizen* to use. That means that the dashboard should be usable for every citizen as well. Europe has a wide variety of different people, each with their own languages, level of (computer) literacy, and possible disabilities. From the get-go, the dashboard needs to be designed to accommodate for as many different people as possible.

Not only the people are different. The devices they choose to access the dashboard with can differ. The previous point has already addressed the different web-browsers. Different screen sizes should also be supported; from mobile phones to large desktop screens, by the same application.

1.2.3 Performant and Scalable

Performance is a very important part of user experience; people have a tendency to dislike slow applications. The web-based dashboard is no exception. It should strive to be as responsive as possible. Users should not be required to wait unnecessarily. And if they do have to wait, the dashboard should give sufficient feedback and continue to be usable.

Performance is often, if not always, tightly coupled with scale. The more users the system has to service, the higher the load, the slower the response time, which in the end causes a degraded user experience.

The coupling between scale and performance should be avoided as much as possible by making architectural choices that are (relatively) easy to scale towards millions of users. As described before, PoSeID-on aims to be a service for all European citizens.

1.2.4 Secure

PoSeID-on is a platform on which users (data subjects) manage their personal information and access hereto. Personal information is, by definition, of a sensitive nature. It goes without saying that security is one of the cornerstones of the PoSeID-on project. It even has its own letter in the project's acronym.

A security-in-depth approach is the only viable option to properly safeguard the personal information of data subjects. Every component of the web-based dashboard should be designed with security in mind and nothing should be blindly trusted. To execute security-in-depth effectively, separation-of-concerns is needed. Individual components of the web-based dashboard should only do one thing and do that securely.

All (obvious) security practices of web-based applications should be followed. This means that modern, strict TLS between the user agent and the application is mandatory. HTTP version 2. No third-party requests are allowed: no CDN, no trackers, no ads and no in-line JavaScript. These rules also allow for strict use of HTTP strict transport security, HTTP content security policy, HTTP frame options and HTTP XSS protection.

In short, all the information handled by the web-based dashboard should stay confidential no matter what.

1.2.5 Maintainable

As PoSeID-on as a consortium project, (application) code delivered for the project should be of a sufficient quality level in order to spark collaboration.

This means that application code should be written cleanly with a well-defined architecture. Documentation is a mandatory part of the code. Not only code commentary and API documentation, but also accompanying documentation describing design decisions, development practices and deployment guidelines.

Finally, tests are just as important for a maintainable software project. Not only for stability, but also as a form of documentation. Tests should cover and describe actual business logic. Test coverage metrics are mostly meaningless, as long as business logic is fully expressed in

test scenarios. Taking a look at the tests should give developers an idea of what the software is trying to achieve.

1.3 Approach

How would one go about designing and implementing a system like the web-based dashboard? In this case, the project chose to approach the problem the reverse way around; starting at the user interface design, based on known requirements, and working its way down to the implementation itself.

This has a number of obvious and subtle advantages.

Firstly, the approach invites early and detailed user feedback by presenting a (semi-polished) user interface design first. Any feedback can be quickly re-incorporated into the design without affecting any technical decisions or forcing painful changes in the underlying code; simply because this code does not exist yet. The resulting feedback loop is fast, reactive and interactive, inviting the end-users to collaborate closely with the definition of the final design.

Secondly, the process uncovers previously unknown requirements by forcing users to think about the actual usage of the application.

Lastly, it shapes the expectations of the end-users early on and therefore limits the amount of (unwanted) surprises later in the process.

Once the design is finalised, the use cases and technical requirements can be distilled out of the design. This lays the groundwork for the actual, technical implementation of the software.

During the implementation of the software, the same opposite direction can be followed. The front-end can be developed with the knowledge of all use cases and a finalised design, while the back-end plays catch-up to support the front-end's features. This, again, allows for rapid prototyping and early feedback. In the long run this approach will save time.

The reverse approach to development (front-to-back) does require a certain level of abstraction to be defined beforehand. Especially the API contract between the front-end and the back-end needs to be defined at an earlier stage than usual. Whereas in a traditional approach the back-end dictates the API contract, in the reversed approach this is more of a collaborative effort. This, too, is designed to limit surprises and to improve early collaboration and feedback.

2 Overall Architecture

With all the requirements in mind, an overall architecture can be devised. The architecture needs to be as simple as possible while still meeting all the security and performance needs.

To achieve that, the web-based dashboard has been cut up into a number of smaller services that are either written in-house or supplied by pre-existing open-source software. Each service does one thing and does it well; a microservice architecture if you will.

All communication to the “outside world”, over the internet, happens through a single entry point; the reverse proxy. Both the data subjects and administrators interact with this component using HTTPS (HTTP plus TLS). The reverse proxy’s role is threefold. Firstly, it makes HTTPS mandatory. No unencrypted traffic, except for the redirect to HTTPS, is allowed between the end user and the reverse proxy. The reverse proxy handles all HTTPS functionality, so the individual services are not burdened by this task. Secondly, the reverse proxy forwards traffic to the right service (either the authentication provider, the front-end, or the back-end) based on the incoming URI. Lastly, the reverse proxy will load-balance traffic between multiple instances of the services, if need be. Having a single entry-point like this has a number of advantages. The exposed part of the platform to the internet is much smaller; not a single core component of the platform is directly exposed. Monitoring for security or performance issues becomes much simpler through a single entry-point.

PoSeID-on requires the data subjects to authenticate using eIDAS (electronic Identification, Authentication and trust Services) before allowing them to use the web-based dashboard. In theory, this is an ideal solution. Every European citizen already has a national login provider. The eIDAS project simply connects all of the providers together to allow cross-border authentication with national identification throughout the EU. A set of protocols has been devised to connect all these unique providers together into one, EU-wide, network. At the moment of writing there is, however, no standard that describes how end-users like PoSeID-on should interact with eIDAS. That is the role of the individual authentication providers. Every authentication provider needs to implement support for every other authentication provider through the eIDAS integration protocols. So, in a way, PoSeID-on will not be interacting with eIDAS directly. PoSeID-on will be interacting with authentication providers that, in turn, interact with eIDAS. In the current ecosystem there seems to be no single authentication provider that supports each and every member state’s national identity. The integration is very much still a work in progress. Furthermore, not every deployment of PoSeID-on can be forced to use the same authentication provider. What authentication provider can be used depends on both what national identities are supported by that provider and with which provider the owner of that PoSeID-on deployment has a contract. With all these technicalities in mind, the web-based dashboard needs to support pluggable authentication providers. Whether these providers adhere to eIDAS becomes, surprisingly, irrelevant. To further secure the authentication process and to ease the integration with new authentication providers in the future, the PoSeID-on authentication provider support is split off into a separate service. Since authentication providers can be any provider and no longer need to be eIDAS-connected, the first test provider implemented will be Google. This will under no circumstances be used in production, however. Google is simply an easy, well-documented, external authentication provider to get started with.

The web-based dashboard front-end is everything that will be sent to the browser to display the user interface and interact with the back-end’s API. This includes HTML, CSS, JavaScript, fonts, images and other assets. In modern front-end development, these assets need to be

assembled into a set of static files by an application. This application is run once and the assets are then served, statically from the filesystem, by a webserver.

Most of the business logic of the web-based dashboard resides in the back-end. It defines an API for the front-end to consume. The back-end has several tasks. First and foremost, it abstracts the functionality provided by the other components through the Blockchain API, Data Processor API, Personal Data Analyser and Risk Management Module into one coherent API. Second, it guards access to the APIs by constantly validating and authenticating every request made by end-users before sending it on to those services. The back-end employs Redis to temporarily cache data coming off of the message bus destined for data subjects who are currently not logged in yet, since the message bus to the dashboard can not be allowed to fill up. In the future Redis can also be used to share state between the back-end instances, if necessary.

Every service behind the reverse proxy can be scaled horizontally by adding more instances of the service. The reverse proxy will take care of the load-balancing between the instances.

2.1 Technology Stack

The authentication provider service and back-end service will be written in Python using asynchronous IO primitives through use of the "aio" libraries. Both services are expected to be IO-bound instead of CPU-bound, since both are essentially just message passing services. This also avoids an entire class of bugs introduced with multithreading solutions. Scaling will be achieved by a process model, where every process can handle a large number of asynchronous requests. No threads will be used. All requests need to be considered stateless for this model to work.

The front-end uses the modern JavaScript framework React, HTML5 and modern CSS to create a responsive, accessible web interface. The assets will be served using Node.js in development. Node.js is also used to produce production assets. In production, these can be served by any web server since all assets are static.

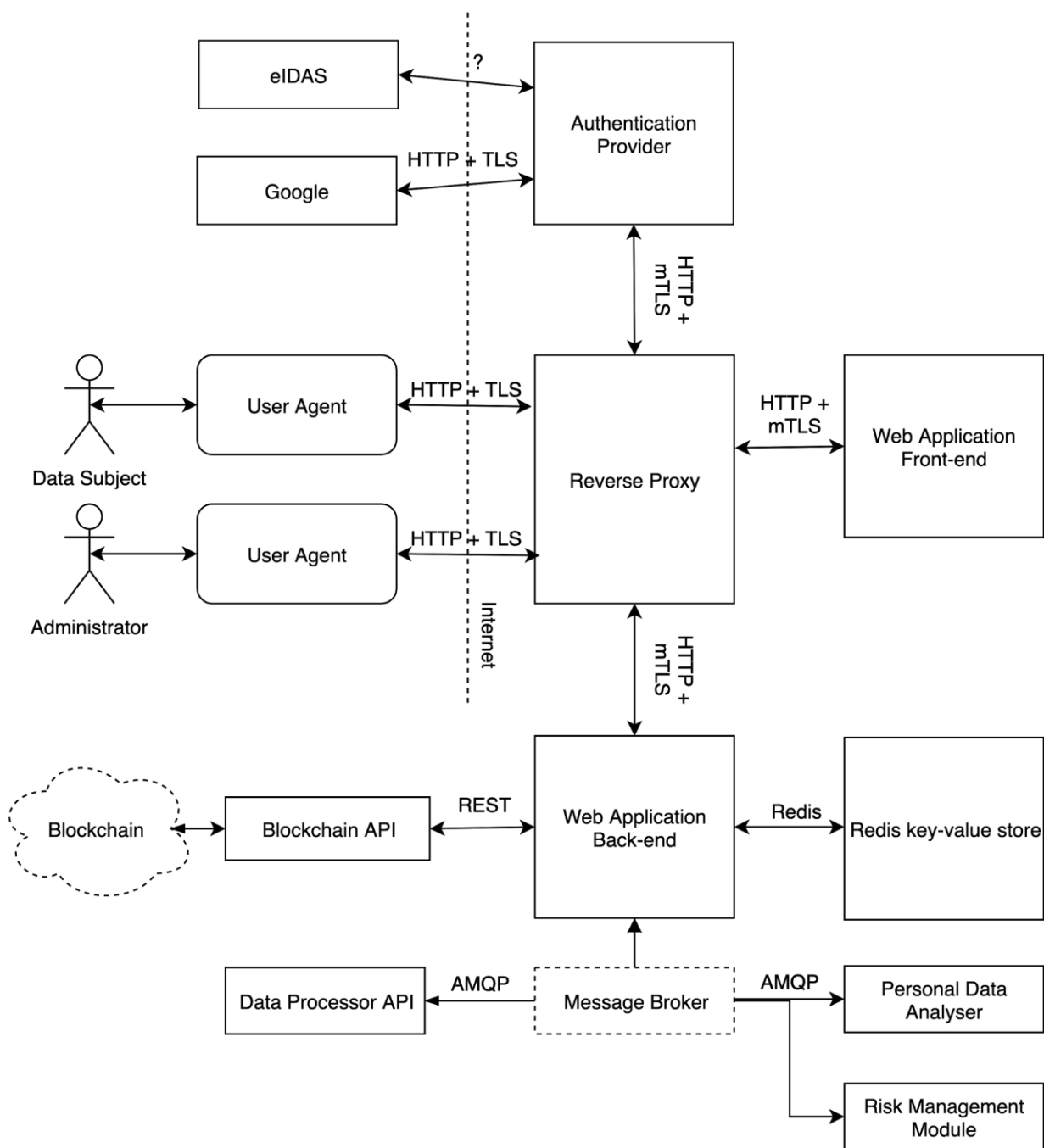


Figure 1 - Dashboard Architecture

3 Authentication Provider

The role of the authentication provider is well-defined: to accept unauthenticated requests, authenticate the user and redirect using an authenticated request. The difference between an authenticated and an unauthenticated request is simply the value of a cookie. This session cookie is stored on the user's browser and is sent with every request to the web-based dashboard.

3.1 Authentication Flow

To benefit security, the design of the authentication flow has been kept as simple as possible. It allows for multiple authentication providers, while keeping the authentication abstracted from view for the rest of the application.

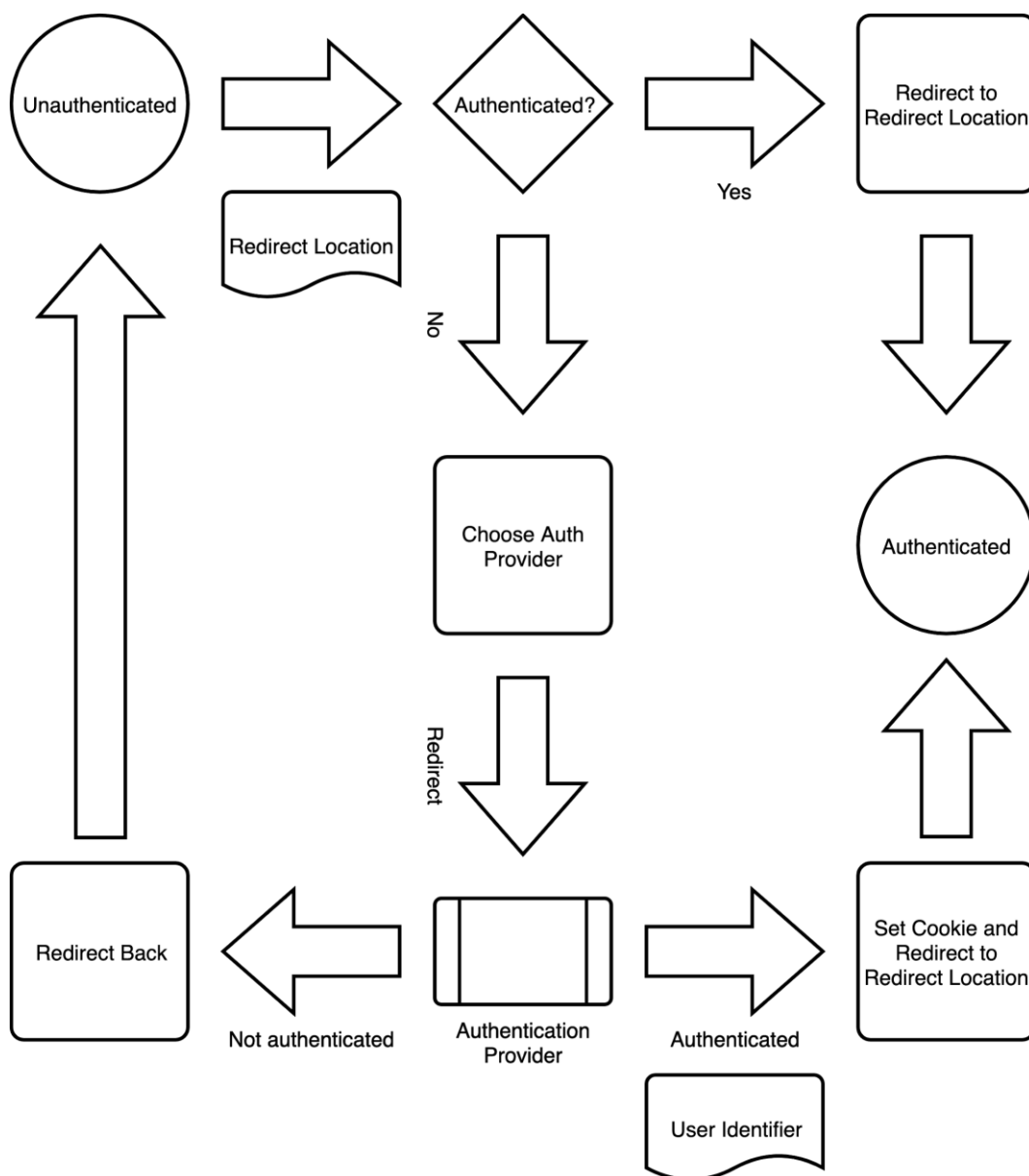


Figure 2 - Authentication Flow Chart

First, an (unauthenticated) user lands on the URI that is served by the PoSeID-on authentication provider service. With this request, the user also supplies the URI to where he/she wants to be redirected to after a successful authentication. The redirect parameter is supplied automatically by the dashboard's front-end logic. The service only accepts relative paths and no full URL, to prevent an open redirect attack. It first checks whether the request is already authenticated. It does that by checking the authentication cookie presence and validity. If it is, it simply does not have to do anything and redirects the user back to where the user requested to be redirected to. If the request is not authenticated, the user is presented with a choice (if more than one auth provider is enabled) to choose their preferred authentication provider.

The design assumes two things from an authentication provider. Firstly, it can be approached by simply clicking a button on a webpage. No server-side communication is necessary. Secondly, it supplies the caller with a unique identifier of the authenticated user upon a successful authentication.

After a successful authentication, the session cookie is set in an HTTP 302 redirect response back to the supplied redirect location. An unsuccessful authentication attempt redirects the user back to the beginning of the process.

3.2 Cookie

The cookie is the only piece of HTTP request data that makes the difference between an authenticated request (and thus access to all the user's private information) and an unauthenticated request. There are a few approaches to authenticating requests to an API. Cookies are just one of them. So why choose for plain old cookies over the alternatives?

The web-based dashboard is split up in two parts: the front-end and the back-end. The front-end instructs the user's browser to send requests to the back-end based on the user's interaction with the front-end. Both are hosted on the same domain behind the reverse proxy. To successfully authenticate a user, a token needs to be sent with every API request to the back-end. This can be done in one of three ways: using a field in the API, using an HTTP header like "Authorization" or using a cookie. A cookie is, by far, the oldest and most widely used method of supplying a token with every request. Cookies are a well-known and well-researched part of browsers. Local session storage or even JavaScript objects are less mature from a security perspective. There is no need for the flexibility of the other approaches because the token needs to be sent with every request and everything uses the same domain.

The cookie needs to supply the back-end with a few bits of information: the authentication provider used, when the authentication took place and the unique identifier of that user. The unique identifier should be treated as personally identifiable information and should therefore not be easily read by third parties. Even though the cookie value can be considered as secure as the web page content, security-in-depth dictates that even the cookie value should be protected. The cookie value is stored for much longer on the user's device than the web page content and is a juicy target for malware. Therefore, the cookie value should be encrypted.

The cookie can not be valid forever. Once an attacker captures a cookie, that cookie gives access to all the user's information. If the cookie is only valid for a certain amount of time, that limits the window of attack dramatically. After the validity period, the cookie should no longer be considered valid and the user is logged out. A short one-hour validity period is recommended.

To validate the cookie value, it needs to be cryptographically signed by a trusted provider. In this case, that is the PoSeID-on authentication provider. To protect the PII in the cookie value, the value needs to be encrypted as well. Finally, there has to be a way to serialise the three fields into one payload.

A popular choice for this problem domain is JWT (JSON Web Token). However, JWT has a number of properties that make the technology inappropriate for use in this context. Firstly, it does not encrypt the payload by default. There is support in the protocol for it, but it has to be enabled manually. Secondly, JWT is very complex. The specification is very broad and supports much more than needed in the web-based dashboard's case. Lastly, JWT has received a large number of critical reviews from security researchers in the field, disapproving the use of JWT altogether.

The chosen alternative is making use of the cryptography PoSeID-on is already using on the message bus; libsodium for encryption and signing, and Protobuf for serialisation. This approach is much simpler and only offers one, secure, option. The cookie is encrypted and signed using a libsodium "secret box", which provides a method of authenticated encryption using a shared secret, which uses the Poly1305 MAC and the XSalsa20 stream cipher. Both the authentication provider service and the dashboard's back-end share the same secret and can therefore validate and decrypt the cookie value. The resulting ciphertext and signature are then encoded in base64 and stored in the cookie value.

Validation of the cookie happens the other way around. First, it is base64 decoded. Then it's verified and decrypted using libsodium's secret box. Afterwards it is deserialized using Protobuf and the validity of the timestamp is checked. If everything is alright, the user is authenticated.

To help the implementation of both the back-end and the authentication provider service, the authentication token processing is abstracted into a common library.

3.3 Pluggable Authentication Providers

To support multiple authentication providers, authentication providers need to be abstracted away into a common interface.

This interface needs to supply the front-end code to supply to the user that shows an authentication button or some other code to be run in the user's browser for this provider. It also needs to uniformly return the unique identifier of the user upon a successful login or an exception upon failure.

4 Web-based Dashboard Back-end

The back-end can be seen as a translation layer between the services that comprise the PoSeID-on network and the front-end. It exposes a RESTful interface towards the front-end (and, indirectly, the user) and speaks the individual protocols towards the rest of the services to expose the needed functionality towards the user.

As with all REST APIs, the HTTP response code determines whether or not a request has been completed successfully.

The API speaks JSON in its request and response body. In the case of an error, the response body contains an object with one key, "error", with a value of an error code. This error code is not human readable and can be translated into a human-friendly message in the right language. A list of error codes will be provided in the final API specification.

4.1 Architecture

All logic is implemented in Python, in a single process. This is done to keep things simple and obvious. Scaling, as mentioned before, is done by simply spawning more of these identical processes. Every event is stateless and can be processed by any process. Communication to the outside world happens using asynchronous I/O.

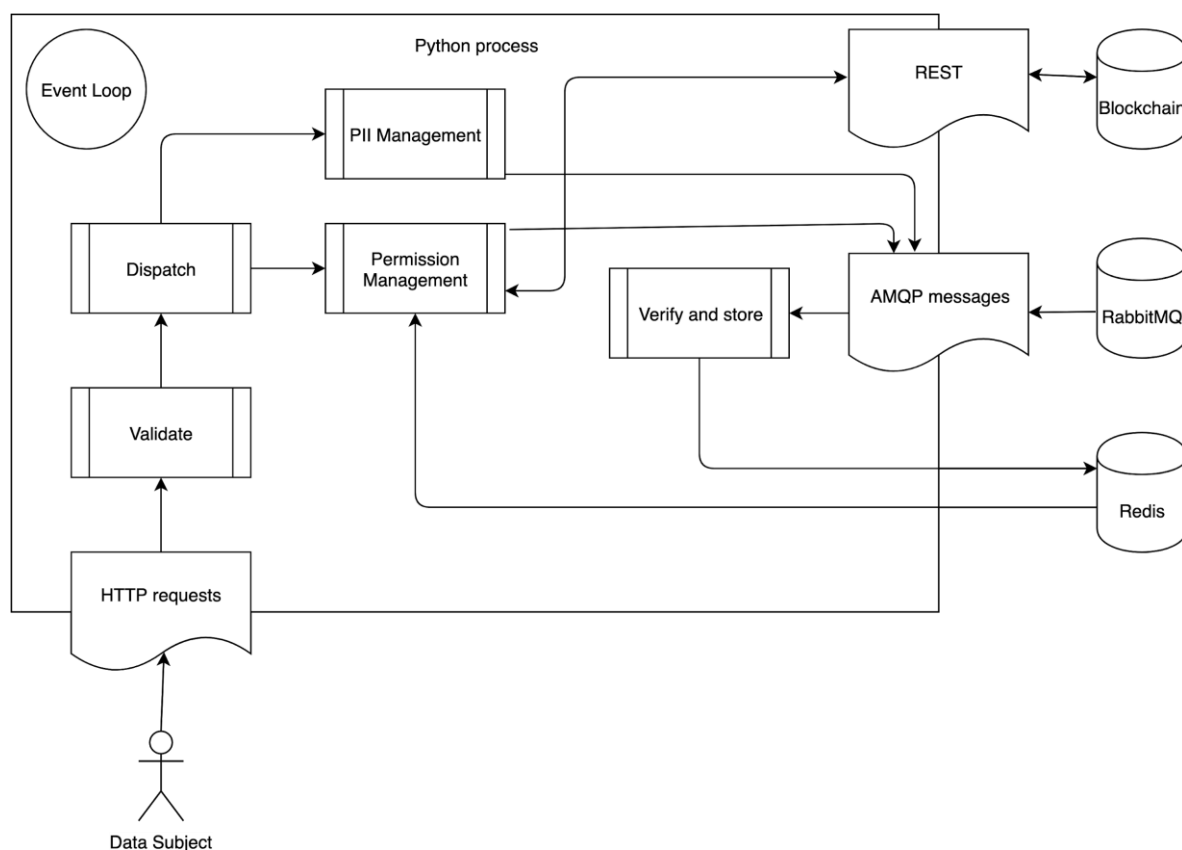


Figure 3 - Back-end Functional Schematic

The code, running in this single Python process, is made up out of several functionalities. Data subjects send HTTP requests (through the reverse proxy) to the back-end process. Here, all

requests are first validated for authentication and authorisation. The messages are then dispatched to either permission management or PII management business logic.

PII management takes care of the actual *data* of the PII. It makes use of the message bus (through AMQP) to send and receive values. Messages about permission management coming into the back-end through the message bus are verified and stored into Redis for later use. This is done for both security and performance; the messages can simply not be decrypted yet without the data subject being logged in on the dashboard yet. Once the data subject has logged in, the back-end has access to the right cryptographic material (retrieved through the Blockchain API) to decrypt the stored messages. The data subject can then change the state of permissions, which will be propagated through the Blockchain API and RabbitMQ message bus.

4.2 Authentication

The front-end must be able to check whether or not the current session is valid and authenticated. If it is not, it must redirect the user to the authentication service.

Every API call to the back-end needs to be authenticated, so any API call will be sufficient to check whether or not the current session is valid. To see whether the session is valid but the authentication to the Blockchain API (see below) has not completed yet, there is the following API call:

- GET /api/auth
 - Returns the current state of authentication.
 - Parameters
 - nothing
 - Return
 - auth_provider: boolean (true if authenticated against auth provider)
 - blockchain: boolean (true if authenticated against blockchain API)
- DELETE /api/auth
 - Logs out the user
 - Parameters
 - nothing
 - Return
 - nothing

4.3 Blockchain API

After the user has successfully authenticated using the authentication provider, the user needs to authenticate against the Blockchain API. This will unlock the access to the Blockchain and give the session access to the transport key and certificate, needed for communication over the message bus.

If no keys are available yet, the user needs to be able to generate them using the same API.

For both situations, the user provides a passphrase to unlock the access. Either the password is used for decryption of the keys, or the password is used to encrypt the freshly generated keys. All the cryptography is done by the Blockchain API and is simply abstracted by the dashboard's back-end.

Following is the preliminary API for key material:

- POST /api/keys/authenticate

- Authenticate against the blockchain API. Populates your cookie with the transport key material for use in later API calls.
- Parameters
 - passphrase: string
- Return
 - nothing
- POST /api/keys/me
 - Register a new user. Does the same as authenticate to the cookie.
 - Parameters
 - passphrase: string
 - Return
 - recovery: string
- GET /api/keys/me
 - Returns the key information for the currently logged in user. User must be authenticated.
 - Parameters
 - nothing
 - Return
 - certificate: string (base64-encoded certificate)
- PUT /api/keys/me
 - Recovers the keys using the recovery phrase or the current passphrase, sets the new passphrase. Can be used for both account recovery and passphrase rotation. Does the same as authenticate to the cookie.
 - Parameters
 - passphrase: string
 - recovery: string
 - - OR -
 - old_passphrase: string
 - Return
 - recovery: string

Following is the preliminary API for permissions:

- GET /api/permissions/\$processor
 - Gets all permissions given to a data processor
 - Parameters
 - processor: string (base64-encoded certificate)
 - Return
 - permissions: array
 - field: string
 - allowed: boolean
 - mandatory: boolean
 - motivation: string
 - from: string (optional; base64-encoded certificate of other data processor)
- PUT /api/permissions/\$processor
 - Update the permissions given to a data processor
 - Parameters
 - processor: string (base64-encoded certificate)
 - field: string
 - allowed: boolean
 - Return

- success: boolean (false if it wasn't allowed)

4.4 Messages for Data Subjects

Data processors and other services like the risk management module can send data subjects direct messages over the message bus. However, the messages can not stay on the message bus while the receiving data subject is not logged in. This would clog up the bus. Therefore the back-end needs to pick up these messages and put them into storage for later use. The storage back-end used is Redis. Since the messages are encrypted, they can be safely stored until they can be decrypted by the user.

The front-end can read the messages through the back-end's REST API. To decrypt the messages, the transport key needs to be supplied.

Following is the preliminary API for messages:

- GET /api/messages
 - Gets the messages intended for this data subject from the message store.
 - Parameters
 - nothing
 - Return
 - messages: array
 - id: string
 - read: boolean
 - timestamp: int (UNIX, UTC)
 - sender: string (base64-encoded certificate)
 - identifier: string
 - params: object (key, value)
- PUT /api/messages
 - Mark message as read.
 - Parameters
 - id: string
 - Return
 - nothing

4.5 Data Processor API

The data processor API has a single endpoint for each data processor, reachable over the message bus. It exposes functionality for the transport and management of personal data. Permissions are managed by the Blockchain API.

The back-end expands on the functionality provided to offer a full API towards the front-end to cover everything regarding data processors. This includes a list of all known data processors.

Fields are identified by standardised field names. It's up to the front-end to translate these field names to human-readable labels.

Following is the preliminary API for accessing data processors:

- GET /api/data-processors
 - Gets a list of data processors, their information and all the fields they support.
 - Parameters
 - nothing

- Return
 - processors: array
 - certificate: string (base64-encoded certificate)
 - name: string
 - description: string
 - image_url: string
 - fields: array
- POST /api/data-processors/correlate
 - Send a correlation ID back to the data processor that requested it, after login or sign-up. See Data Subject Correlation.
 - Parameters
 - processor: string (base64-encoded certificate)
 - correlation_id: string
 - Return
 - nothing
- GET /api/data-processors/\$processor
 - Retrieves all fields that are known by this data processor for this data subject.
 - Parameters
 - processor: string (base64-encoded certificate)
 - Return
 - fields: array
- GET /api/data-processors/\$processor/\$field
 - Reads data the data processor has stored for this data subject.
 - Parameters
 - processor: string (base64-encoded certificate)
 - field: string
 - Return
 - value: string
- PUT /api/data-processors/\$processor/\$field
 - Update information stored by the data processor.
 - Parameters
 - processor: string (base64-encoded certificate)
 - field: string
 - value: string
 - Return
 - nothing
- DELETE /api/data-processors/\$processor/\$field
 - Remove information stored by this data processor.
 - Parameters
 - processor: string (base64-encoded certificate)
 - field: string
 - Return
 - nothing

4.5.1 Data Subject Correlation

The data subject, coming from the website of a data processor, needs to be correlated with their internal personal identifier before any communication over the Data Processor API would work.

The data processor sends a GET request to the dashboard with two parameters in the URL:

- GET /

- Show the dashboard and, optionally, correlate a user.
- Parameters
 - processor: string (base64-encoded certificate)
 - correlation_id: string

4.6 Personal Data Analyser and Risk Management Module

Information sent to data subjects by the personal data analyser or the risk management module will be in the form of direct messages. The data subject does not communicate with the two subsystems directly, other than giving them permission to access personal data through the permissions API.

For all intents and purposes seen from the web-based dashboard, the two subsystems are just ordinary data processors.

4.7 Technology

The web-based dashboard uses, as mentioned before, Python asynchronous IO. Communication with Redis and the message bus happens completely asynchronously using their respective asynchronous client libraries; *aiomq* and *aioredis*.

Tests are written using *pytest*, for both unit- and integration tests.

5 Web-based Dashboard Front-end

The web-based dashboard front-end is all code and layout that will run in the end user's (the data subject, mostly) web browser. It represents the user interface and business logic directly running inside of the client's browser. Whereas the back-end is mostly just a message passing system, the front-end turns the available APIs into a usable application.

5.1 Persona

Before any design can take place, a good start is to have a clear picture of who the end user is going to be. Since most design and software engineering is done by technically literate - often highly educated - individuals, having a description of the average end user helps tremendously. It allows for designers and engineers to better take into account the end user's expectations and experience.

An often-used approach is to create a *persona*, a fictional person that somewhat accurately describes the average end user. A persona comes with a number of personal details, including a short backstory, to give people reading it a general idea of who this fictional person is.

For PoSeID-on, the following persona has been developed:

Name: Pierre Lacroix

Age range: 18-70. 59 years old in this scenario.

Location: France, Clichy-sous-Bois, a commune in the eastern suburbs of Paris, France.

Gender: Male

Income: 51.840 €/year

Profession: Electrician

Other: Has little knowledge about computers or using the internet.

Personality traits:

Extrovert and moderately active in his day to day life. Adaptable, focused, honest, methodical, curious, protective, cheerful, kind.

Goals and motivations:

Pierre is easy going, likes to be surrounded by family and friends and enjoys the occasional barbeque with neighbours.

Pierre is curious and interested in what happens around him. He reads the newspapers and is up to date with political and economic events.

Being concerned with his family's well-being and safety, he is always on time with the payment for his mortgage, bills, and taxes and has all the paperwork sorted out. As he is close to retirement, he has more free time and would like to learn more about the internet and security to make sure his bank account and other assets are safe, especially now that everything happens online.

Environment:

He asked one of his friend's daughters, Deborah, to teach him how to use the computer and how to surf the internet. He is at home, using the family PC. He is open-minded and eager to learn. Pierre is not bothered by the fact that he needs to ask many questions before he does anything online. He even writes down (using pen and paper) directions from Deborah so he will know what to do when she is not there.

5.2 User Stories

With the persona and requirements in hand, a set of user stories for the front end can be compiled. These user stories expand the requirements into explicit user actions.

The user stories serve as an expansion on the PoSeID-on D2.2 deliverable, mainly its chapter 7.2. The stories do not try to force a certain structure onto the user interface; they merely split up the user stories into groups based on similarity in context. The (most popular) normal form of user stories is "As a *subject* I would like to *action* so I can *result*". However, since the subject in this set of user stories is always the data subject (the end user; the common citizen), the subject has been omitted.

5.2.1 Authentication

As highlighted in chapter two, authentication and de-authentication are done by eIDAS, a system beyond our control. A user is either logged in or they are not. Once logged in, eIDAS supplies us with the name of the user and a method to log out again.

If a Data Subject logs in for the first time, a special flag will be set so the Dashboard can behave differently. Mainly, the application must ask the Data Subject for a personal password (another one from the eIDAS password) for encryption purposes. Secondly, the Data Subject will be presented with a tour of the system.

1. I would like to log in so I can make use of the system.
2. I would like to see my name at all times so I can see that I'm logged in as myself.
3. I would like to log out so I can let someone else log in again.
4. I would like to supply a personal password upon my first login so I can have the system encrypt my personal details securely.
5. I would like to be shown a recovery passphrase after entering my password so I can recover my password if I forget it.
6. I would like to be shown the Terms and Conditions (see Annex I) upon my first login so I can make an informed decision about whether to use the platform.
7. I would like to be shown the tour upon my first login so I can get to know the system and enter some PII.
8. I would like to be able to change my password by using my previous password so I can keep my security top-notch.
9. I would like to be able to change my password by using my recovery passphrase so I can change my password even if I forgot my password.

5.2.2 Data Processor Listing

1. I would like to see a list of all Data Processors I can send my PII to so I can start a procedure at that Data Processor.
2. I would like to see the Data Processor logos in the list so I can easily identify them.
3. I would like to be able to open a Data Processor from that list so I can interact with that Data Processor.
4. I would like to be able to search in the list of Data Processors if the list is very long, so I do not have to manually search a long list of Data Processors.

5.2.3 Permission Listing

1. I would like to see what permissions I have given over a certain PII so I can have an overview over my permissions.
2. I would like to see who I have given access permission to a certain PII so I can know who has access to my PII.
3. I would like to see for how long the access permission is granted so I can anticipate the future permissions.

5.2.4 Permission Request Listing

1. I would like to see all permission requests I have received from data processors to my PII so I can either grant or deny the permission request.
2. I would like to be able to filter in the permission requests so I can more efficiently interact with them.

5.2.5 Permission Granting

1. I would like to grant permission on a certain PII to a data processor so I can give that data processor access to my PII.
2. I would like to limit the amount of time the data processor has access to my PII so I can make sure my PII is not accessible forever.

5.2.6 Permission Revocation

1. I would like to be able to revoke permission to a data processor over a PII so I can make sure they do not have access to it anymore.
2. I would like to see a notification if permission revocation is not possible so I can be informed about why it is not possible.

5.2.7 PII Listing

1. I would like to list all the PII known about me by a data processor so I can be informed about the current state of my details.

5.2.8 PII Supplying

1. I would like to be able to supply PII to a data processor at their request after I have granted the permission request, so the data processor can process my PII.
2. I would like to be able to see whether the supplied PII comes from another data processor or if I have to supply the data myself so I can either pass it along or fill it out.

5.2.9 PII Updating

1. I would like to update already supplied PII so I can keep the PII up-to-date.
2. I would like to be able to see if PII is read-only so I can be kept from even trying to update the PII.

5.2.10 Right to Be Forgotten

1. I would like to exercise my right to be forgotten with a data processor so that they can destroy all information they have on me.

2. I would like to be able to see if the right to be forgotten can not be executed because of legal reasons so I can be informed of the law.

5.2.11 PDA/RMM Permissions

1. I would like to grant the PDA/RMM permission to access my transactions so I can be informed of any security issues by the PDA/RMM.
2. I would like to revoke the permission to the PDA/RMM to have access to my transactions so I have full control over who I share my PII with.
3. I would like to be informed about the trade-off between functionality and security so I can make an informed decision.

5.3 Screens

With the requirements, user stories and the persona in hand, a rough sketch of the needed screens can be made. Screens are separate “pages” in a user interface, i.e. significant changes in what information is presented. At the time of writing, this is an interim view of the screens the dashboard will contain. More functionality and detail will be added at time progresses.

As a concept that is easy to understand for (most) end users, a message-based approach has been chosen. The look and feel can therefore mimic email or any other popular messaging system. Whenever the data subject interacts with a data processor or vice versa, it is represented as a message.

5.3.1 Common

Common features of all screens will be the navigation of the main menu items, and a way to access user preferences and authentication on the top-right corner.

The menu should be short and easy to skim. The menu items are:

- Dashboard;
- Messages;
- Data Processors;
- Help.

These menu items will take the user to the respective screens, described below.

The top-right corner menu is where the user can find three elements:

- Who the user is logged in as;
- A way to log out;
- A link to personal preferences.

What these personal preferences will be is yet to be determined.

5.3.2 Dashboard

The main page of the application will be called the Dashboard. Here, the user will be able to see the latest PII exchanges involving them and what data processors were involved to give the data subject an immediate feeling of transparency. There will also be a summary, in numbers, of the amount of PII requests, grants and data processors involved with the logged in user.

5.3.3 Messages

The message screen lists the messages the data subject has received in reverse chronological order. Every message represents an event involving that data subject. If the message requires an action from the data subject, a button to do that action will be presented in the message. Every message has a read-status, so the user can keep track of what messages they have read.

5.3.4 Data Processors

The data processors screen (the name presented will probably need to change into something more user-friendly) lists all connected data processors. It allows for the user to search in the list. The list prioritises the data processors the data subject is involved with.

The list can be enriched with summary information of each data processor, depending on technical possibilities.

A single data processor can be expanded, showing the data shared and/or requested, the data exchanged with other data processors and the controls to update information and/or permissions.

5.3.5 Help

Many citizens are unaware of the rights they have under the new GDPR legislation. This help section explains both the usage of the PoSeID-on platform and the rights the user can exercise in a very approachable manner.

5.4 User Experience Design

With the aforementioned screen descriptions, requirements, user stories and persona a preliminary UX (user experience) design can be produced. This is done using a mockup tool.

The mockup tool used is Sketch with the Marketch plugin to produce HTML. The latest mockups have been added to this document to give an initial, non-binding, preview of the design direction.

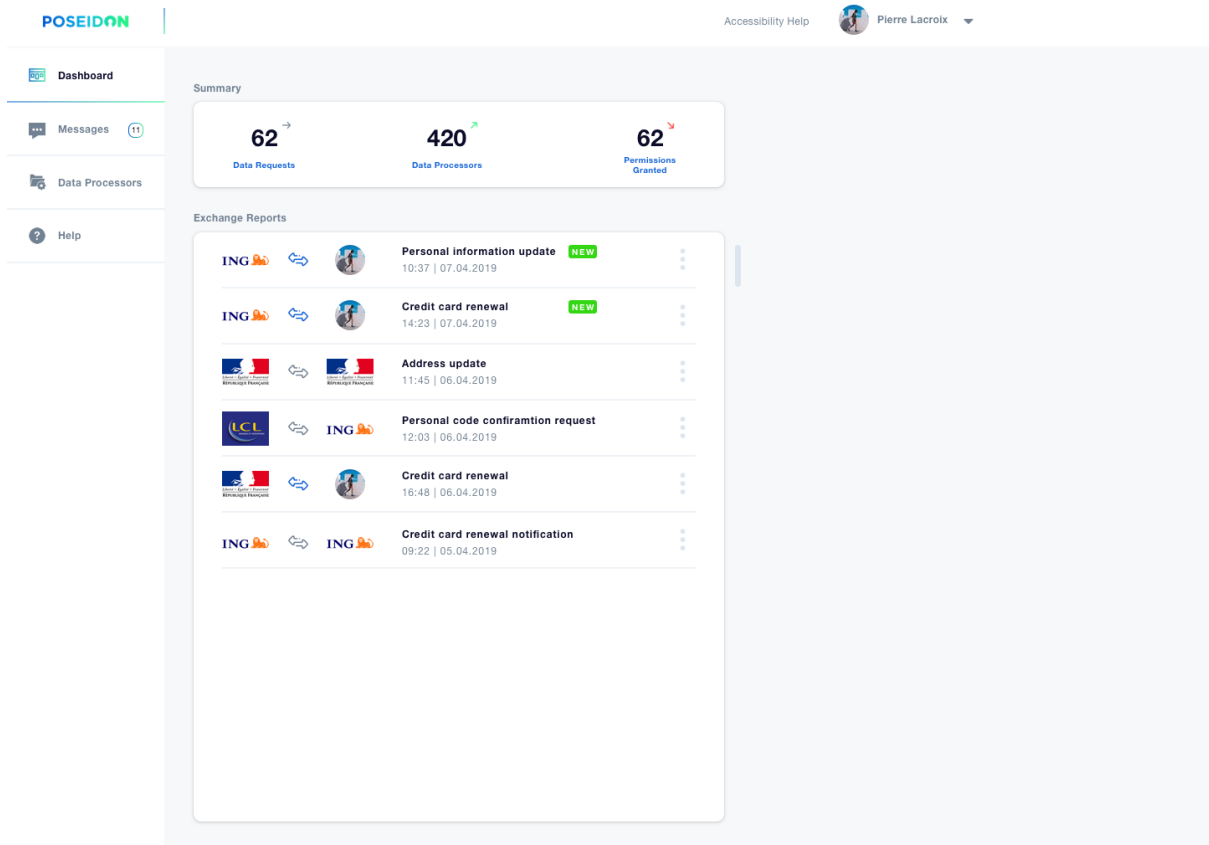


Figure 4 - Dashboard UX Mock-up

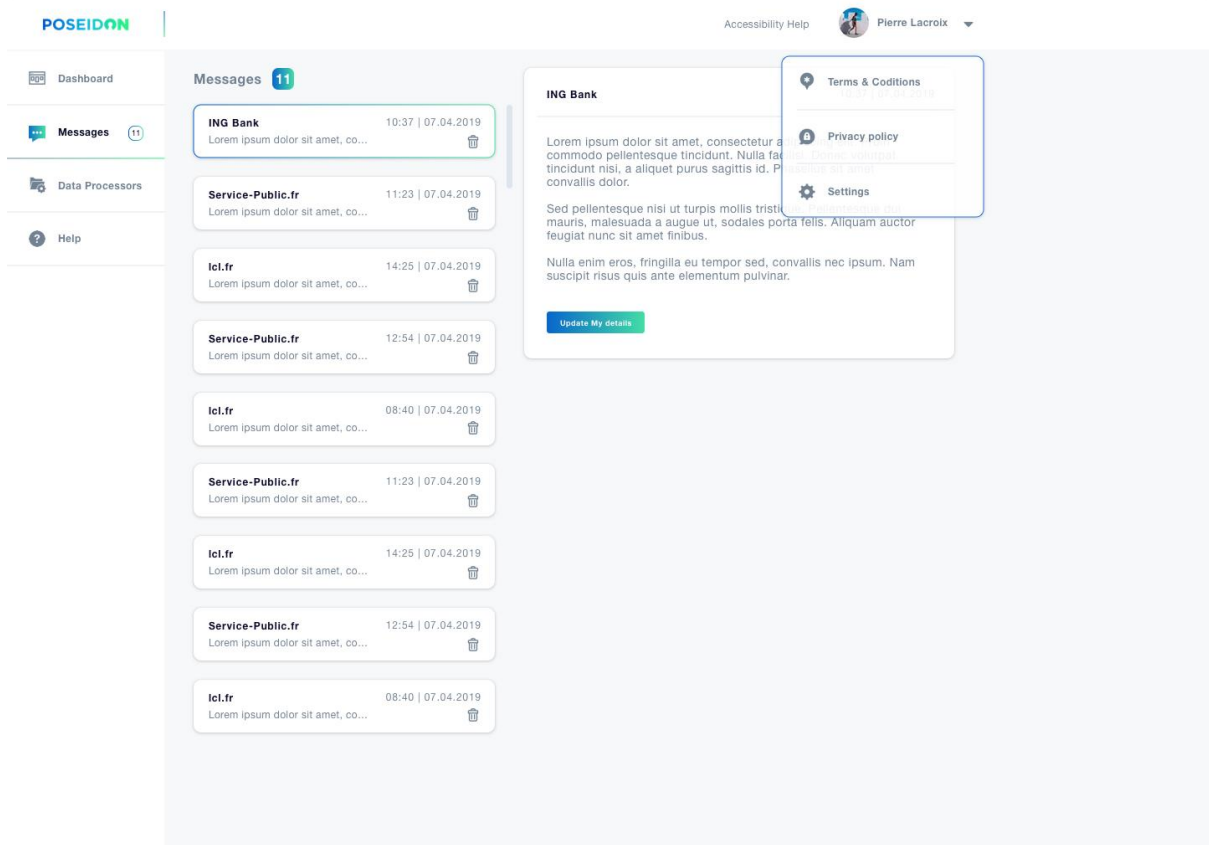


Figure 5 - Dashboard UX Mock-up

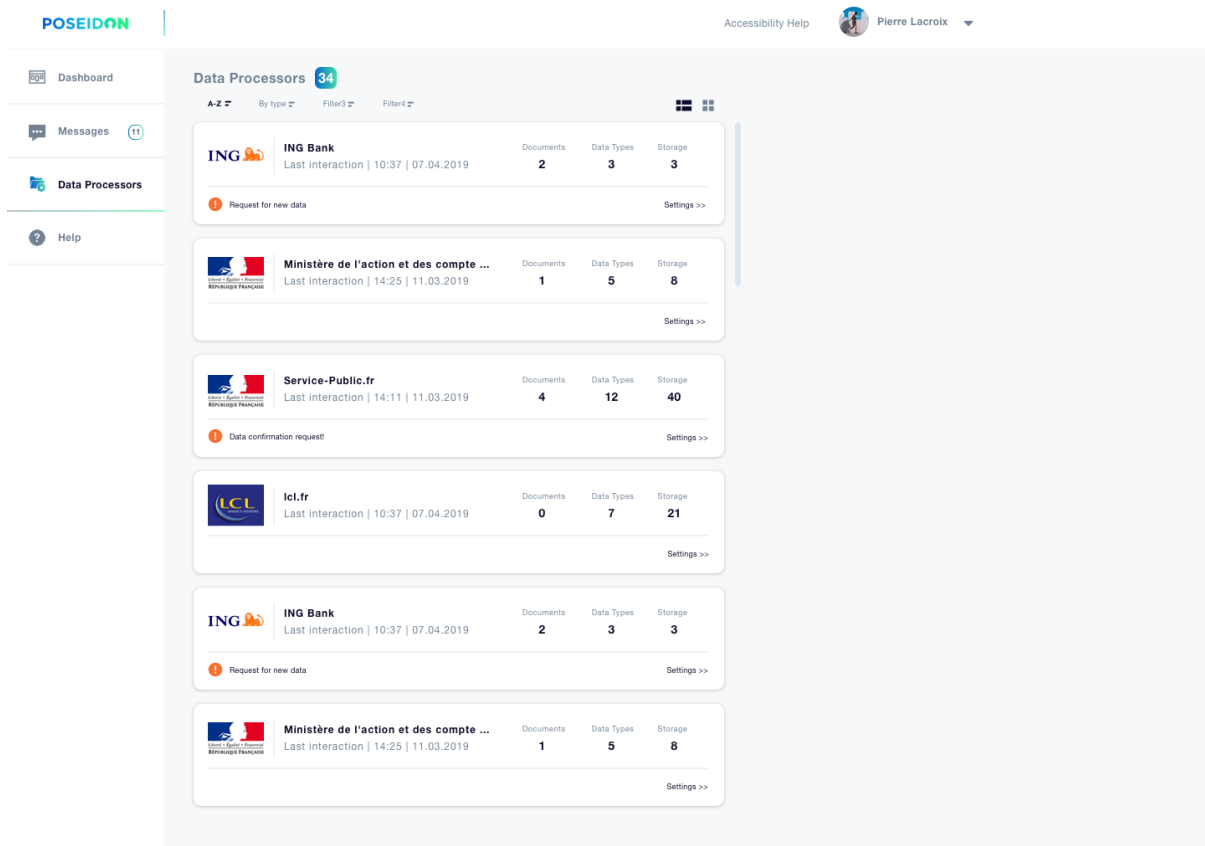


Figure 6 - Dashboard UX Mock-up

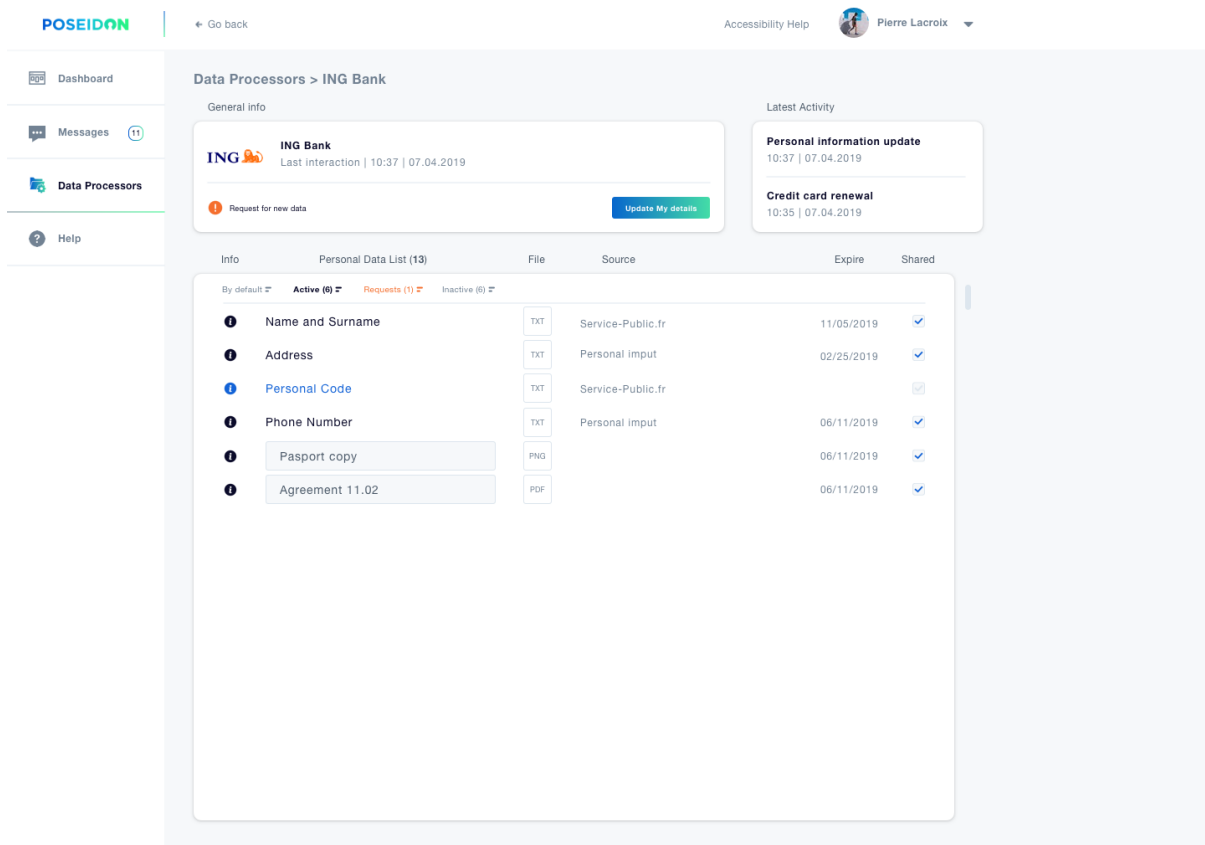


Figure 7 - Dashboard UX Mock-up

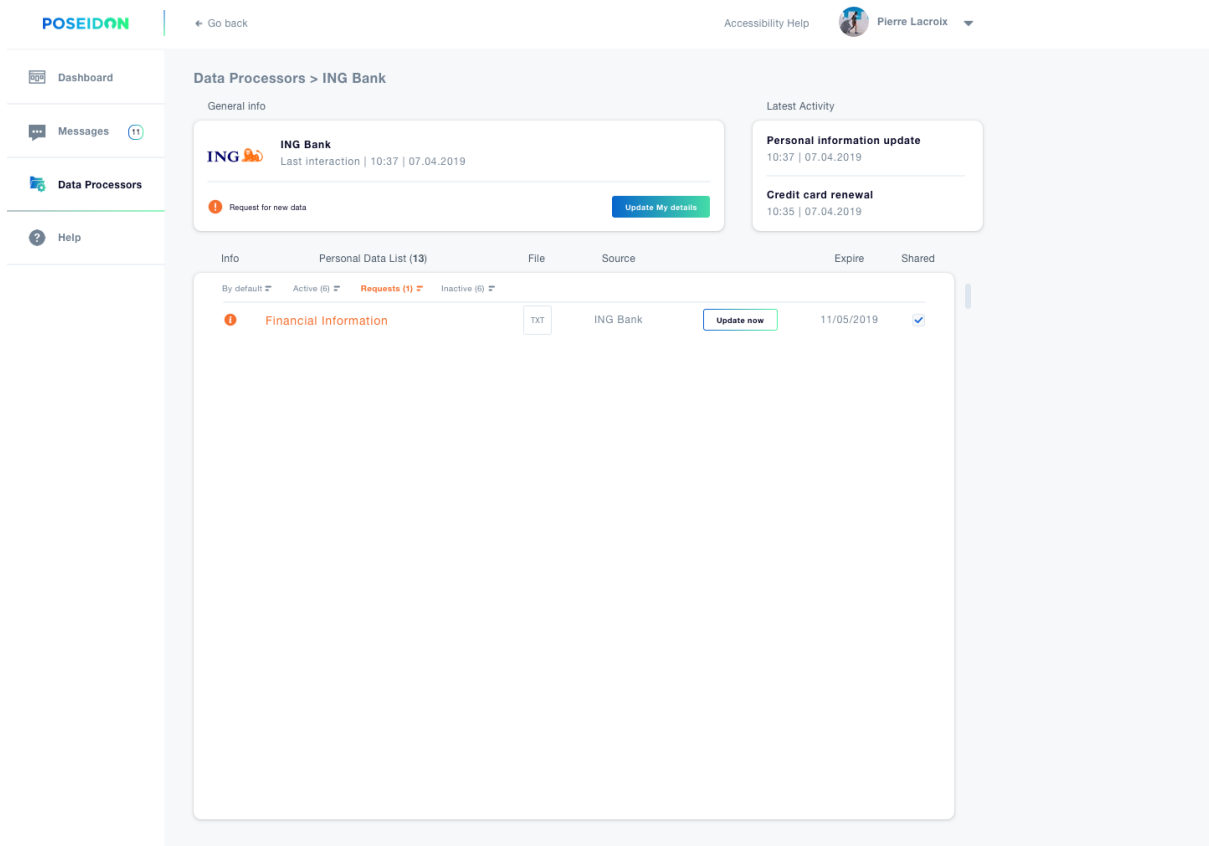


Figure 8 - Dashboard UX Mock-up

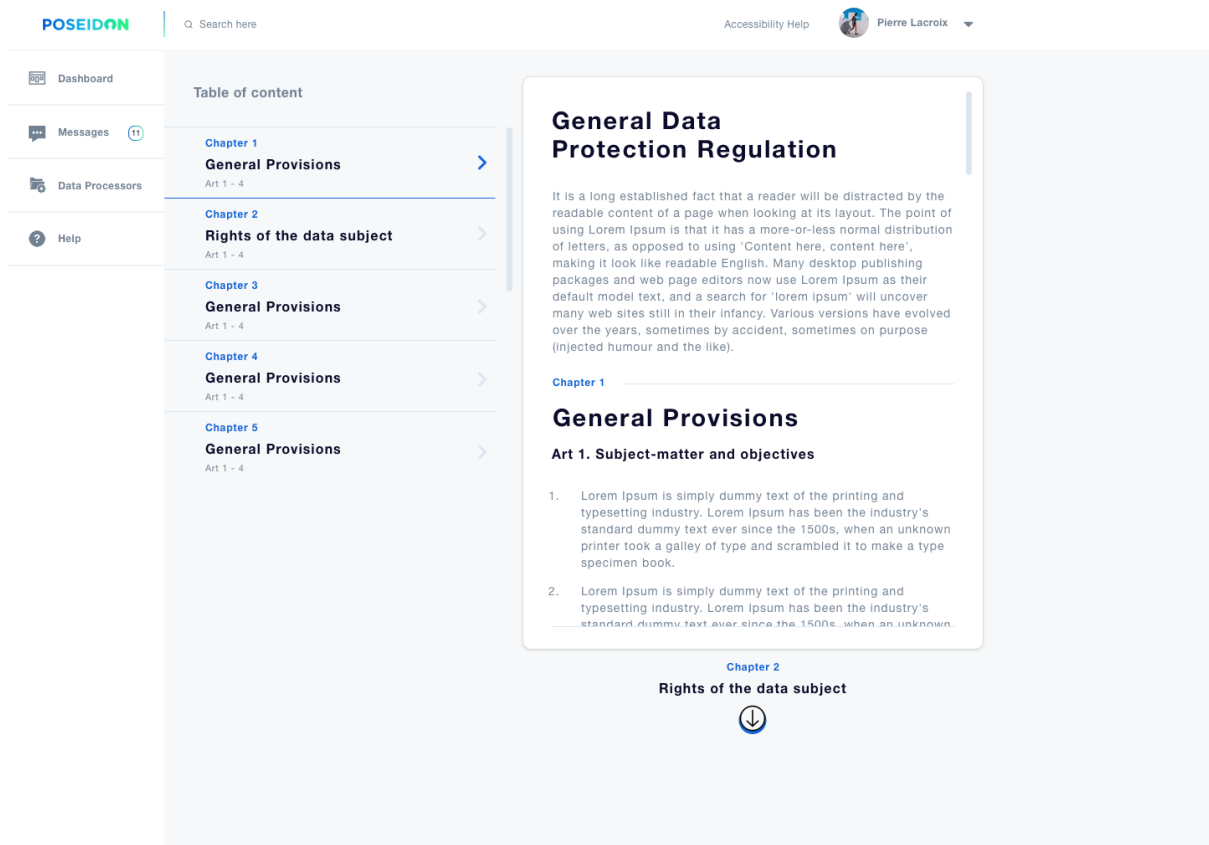


Figure 9 - Dashboard UX Mock-up

5.5 Technology

With a UX design provided, the technology stack for the front-end of the web-based dashboard can be chosen. The goals of the choices of this stack are identical to the overall goals: to be accessible, responsive, performant, scalable, secure and maintainable.

For the front-end business logic to be maintainable, a framework is a necessity. The added benefit is that a well-known framework also makes the code more transferable to other engineers, simply by the fact that it forces the project to be structured in a certain way. For the PoSeID-on web-based dashboard React.js is chosen. At the time of writing, it is one of the most popular front-end JavaScript frameworks available¹. Using a popular choice ensures that developers have a large body of documentation to source from during development, and makes sure that the software can easily be supported by other developers than the original authors. Added to the React framework, React Router is added for declarative routes.

Much of the business logic can be expressed as a state machine. Therefore, Redux is introduced to make the development and management of state machines much easier.

For testing, Jest and Cypress are used for unit and integration tests, respectively.

¹ <https://2018.stateofjs.com/front-end-frameworks/overview/>

6 Integration Approach

To have all the components that make up the web-based dashboard integrate nicely into the PoSeID-on platform as a whole, an overarching integration approach is needed. The main body of work describing the integration efforts is found in Work Package 5. This chapter merely describes the ways in which the web-based dashboard aims to meet WP5's requirements.

Integration is done on several levels. Firstly, components need to be packaged as reusable parts. The PoSeID-on integrated platform uses OCI containers for that. Secondly, these reusable parts need to be configured somehow. Lastly, the components need to be able to communicate with the rest of the PoSeID-on platform using well-defined protocols.

6.1 Packaging

The web-based dashboard components are built using two language runtimes; Python and Node.js. Python is used for the authentication server and the back-end. Node.js is used for the front-end.

Both language runtimes get their own OCI base container image, inheriting from the common PoSeID-on base image, in order to centralise all common dependencies between components. This helps with centrally rolling (security) updates across the entire platform.

Every component gets its own git repository, containing the source code and accompanying documentation:

- *poseidon-dashboard*
The dashboard back-end.
- *poseidon-frontend*
The dashboard front-end.
- *poseidon-auth*
The authentication server.

Versioning of container images follows the git branching model. This is different from what is normally done with container images. Normally, new releases of software get a new version number, a new container image is built, tagged with the version number and end users have to update their configuration. The "latest" tag is often misunderstood and is prone to mistakes. Therefore, the git branch name is used as the tag. End users do not need any configuration changes to update to the latest stable container image; they only need to pull updates and restart their containers.

Container images can be automatically built and published to the image registry out of the source code repositories by simply using the supplied Makefiles.

Common code shared between components is shared using Python modules, installed through git. At the time of writing, there are two Python modules written that are shared between containers:

- *poseidon-protocols*
Responsible for the message bus communication: (de)serialisation, cryptography, verification and protocol-level business logic.

- *poseidon-token-manipulator*
Responsible for the authentication token on the web-based dashboard, used by end users (data subjects) to authenticate themselves.

6.2 Configuration

Following 12-factor standards and the choice for Kubernetes, all configuration of the software making up the components happens through environment variables. From the get-go all variables, including secrets, are supplied through the runtime environment.

This means that no secrets or default configuration can be found in the source code repositories. However, there will be some documentation about what environment variables will be used and what the values should be.

Configuration about how the application should be started is done through the Dockerfile, which is part of the OCI container image.

The final part of the configuration, the deployment settings, are supplied using Kubernetes configuration files in the yaml format. For development purposes, patches using *kustomization* are applied that make the components behave in a more contained way.

6.3 Communication

This chapter only describes the communication between the web-based dashboard and the rest of the PoSeID-on platform. Communication between the dashboard components or between the dashboard and end users is described in other chapters.

For communication with the rest of the platform, the web-based dashboard accepts messages for data subjects, and it sends messages to other components on behalf of the data subjects. All the communication flows through the message bus.

The message bus protocol specifies that the name of the AMQP queue is derived from the recipient's public key. However, having a queue for each data subject would run into scaling issues. Therefore, the dashboard listens on a single queue named *dashboard* on behalf of all data subjects. As specified before, the queue will be consumed into a Redis cache immediately on message arrival.

The only component that is not accessed by the message bus is the blockchain API. The communication to this API can not go through the message bus for the simple reason that this API supplies the credentials needed to even start communicating over the bus. For communication with the blockchain API, a separate library still needs to be developed. At the moment of writing this document, the blockchain API is not available yet in the integrated platform. When the integration has taken place, the functionality provided by the blockchain can be provided to the web-based dashboard.

6.4 Data Storage

The web-based dashboard is designed with privacy in mind. All data not stored is data that can not be leaked, either accidentally or stolen. Especially personally identifiable information (PII) is of a very sensitive nature and should therefore not be stored if it is not needed.

The main "database" the web-based dashboard uses is reached through the message bus and data processor API; all PII is stored on the data processor's systems and transported, end-to-end encrypted, to the dashboard upon request of the data subject. The end-to-end encryption

also extends to the temporary storage into the Redis key-value store of these messages. Only the data subject can decrypt these. At rest, the PII contained in these messages is fully encrypted. No PII is ever stored permanently on the web-based dashboard.

Permissions are stored on the blockchain and are, just like PII, retrieved upon request by the data subject. No permissions are permanently stored on the web-based dashboard.

7 Table of Figures

Figure 1 - Dashboard Architecture.....	12
Figure 2 - Authentication Flow Chart.....	13
Figure 3 - Back-end Functional Schematic.....	16
Figure 4 - Dashboard UX Mock-up.....	27
Figure 5 - Dashboard UX Mock-up.....	27
Figure 6 - Dashboard UX Mock-up.....	28
Figure 7 - Dashboard UX Mock-up.....	28
Figure 8 - Dashboard UX Mock-up.....	29
Figure 9 - Dashboard UX Mock-up.....	29

Annex I – Terms and Conditions

The following Terms and Conditions (hereinafter: the “Terms”) govern the relations between PoSeID-on project and its provided services through the site www.poseidon-h2020.eu (hereinafter “the Platform”) and the users (individuals, private and public organizations) (hereinafter: the “User” individually and collectively the “Users”). PoSeID-on Platform is a project financed by the European Commission within the European Union’s Horizon 2020 program under Grant Agreement n° 786713. The PoSeID-on project has been created by a Consortium participated by several parties, from different European countries, such as SMEs, private/public IT providers, research centers and universities. PoSeID-on is a Platform aimed for the protection and control of personal data and secured information by means of a privacy enhanced dashboard (PED) to safeguard data subjects’ rights and support public and private organizations ensuring them GDPR compliance for the data management and processing. PoSeID-on reserves the right to modify at any time these Terms and Conditions. In this case, PoSeID-on will notify the modified Terms to the User by sending an email to the address provided when registered, or on the first access to the site and will ask the User to accept the new modified version of the Terms.

Registration and execution of Contract

For the access and use of PoSeID-on’s services the Users shall register on the Platform by completing the appropriate registration form. To ensure registration, the Users shall previously read and accept these Terms and Conditions. In case the User does not intend to accept these Terms, he should not continue with the registration, access and use of the Platform. The access to the Platform and reserved areas, for which registration is required, is free. These Terms will be effective upon the Users’ acceptance and will govern the relations with PoSeID-on for the all execution of the contract.

Terms Duration and Termination for withdrawal

These Terms is for an indefinite period, subject to the possible withdrawal of either party, operated to pursuant to the following Section. The Users or PoSeID-on might at any time ask for the cancellation withdrawing the acceptance of the Terms with either party. PoSeID-on may withdraw notifying the User by the email address provided at registration and this withdrawal will be effective in _____ (____) days after notification; the Users can exercise the above mentioned right by sending notice by email to info@poseidon-h2020.eu and the withdrawal shall be effective in _____ days (____) after notification.

Services and using the Platform

By accessing to PoSeID-on Platform, the Users will be able to use the Privacy Enhancing Dashboard (PED) for personal data protection, aimed to safeguard the rights of data subjects as well as support public and private organizations in data management and processing while ensuring the GDPR compliance. The PED allows the data subjects to maintain the control of their own data by having a concise, transparent, intelligible and ease access, as well as tracking, control and management of their Personality Identifiable Information (PII) processed by public or private organizations, acting as data controllers and/or data providers. In this way the Users will be able to make conscious decisions about who can process their own data by enabling, restricting or revoking permissions in accordance to the data minimization principle, as well as to be alerted in case of privacy exposure. Paralleling, PoSeID-on Platform will also represent a valuable instrument supporting private and public organizations processing personal data ensuring compliance of their data protection and control procedures with the EU regulations, as required by the GDPR.

The Privacy Enhancing Dashboard (PED) contains a series of security mechanisms in order to protect the Personally Identifiable Information (PII) and the communication between the data subjects and data controllers/processors (private and public organizations). The Permissioned Blockchain will through the Risk Management Module and the Personal Data Analyzer check the legitimacy of data processing and the data exchanges between the different parties by alerting the data subjects in case of aberration leading to the breach of their fundamental rights and freedom; indeed, PoSeID-on Platform through the Risk Management Module will advise data subjects for any kind of privacy threats or risk exposure by sending warnings or alarms, while, the Personal Data Analyzer will alarm the data subjects to identify anomalous patterns of transactions carried out by historical data analysis.

Paralleling for supporting the public and private organizations, the Privacy Enhanced Dashboard will enforce their data protection and control procedures by the implementation of the following tools:

- the Permissioned Blockchain and Smart Contracts will enable public and private organizations to have a contextual guarantee of accountability, transparency and compliance with the EU data protection Regulation. Indeed, through the implementation of Smart Contracts on the Permissioned Blockchain, PoSeID-on will ensure the quality of personal data collected that will be managed directly by the data subject in order to be accurate and up to date. The system has been conceived in a way that, the specific Smart

Contract will contain the reference to just the User's personal data necessary for that specific transaction, in compliance with the data minimization principle;

- cloud, access management according to eIDAS (electronic Identification, Authentication and trust Services, the EU regulation on electronic identification and trust services for electronic transactions in the internal market) and privacy management.

Moreover, PoSeID-on Platform can also be used as a set of open source tools and toolkits, that can be separated deployed by the Users according to their specific needs. This will mitigate the TCO (Total Cost Ownership) for public agencies and SMEs facing budget constraints. All the available tools can be in this way utilized:

- the Privacy Enhanced Dashboard as an ICT integrated prototype also provided with an innovative web-based dashboard for data subjects with a user-friendly interface. It can be used by organizations that want to integrate their procedures with a GDPR compliant tool;
- open source components or API, as interoperable ICT components that can be integrated in any public or private ICT architecture. In this way, the organizations can integrate every single component of the Privacy Enhanced Dashboard in their own systems achieving a high technological development and competitiveness;
- cloud-based Privacy Enhanced Dashboard as Service (PEDaaS). This service can be used by the organizations that do not have their own blockchain and/or cloud or they don't want to afford, for any reason, the cost of managing the GDPR compliant tools. In this way, they can access the PoSeID-on cloud service and use the Privacy Enhanced Dashboard to monitor and control the data processing.

Obligations and Responsibilities of the User

The use of the available material and services provided by the Platform is under the exclusive responsibility, control and discretion of the User. The User acknowledges and agrees that, it is his responsibility to use the services provided by the Platform in compliance with any rule of these Terms and Conditions and according to the regulations in force and agrees not to hold PoSeID-on responsible for any damage or loss that may be caused by such activities.

In case of breach of this Terms and in the event the behavior of the User does not comply with these Terms or with the policy of PoSeID-on, PoSeID-on reserves the right to limit, suspend, interrupt the account of the User without prior notice or, in the most serious cases, forbid the access to the Platform and its services and take legal action against him. The User is responsible for the confidentiality, truthfulness and completeness of his login credentials (username and password) while registering and to check its updating and

accuracy for the entire duration of use of the Platform's services. PoSeID-on cannot be held responsible for the truthfulness of the information nor for any mistake deriving from inaccuracies and/or incorrect of the inserted data by the User of the Platform. The User shall prevent the use of his login credentials by third parties and report immediately to PoSeID-on any loss of his exclusive control over this information, taking note of the fact that, in the absence of such notification, he is liable for any action caused by third parties. The User undertakes not to use the Platform to transmit, distribute or provide third parties any type of content that in any way can:

- harass, threaten or abuse other Users of the Platform;
- encourage other Users to breach the Platform's rules or other legal rules;
- impersonate members of the PoSeID-on's Platform staff or other Users using a similar identity or any other tool or device;
- perform spam actions such as carry out publications that may damage who receives them or to carry out commercial, advertising or illegal proposals;
- induce or incite to criminal, degrading, defamatory, violent or discriminatory actions due to sex, age, illness, creed etc.;
- encourage involvement in actions that are dangerous, risky may damage people's health or the psychological status;
- send viruses that could damage or interrupt the normal use of the network, the system or the computer and the PoSeID-on's Platform tools;
- breach material protected by third parties copyright, or by PoSeID-on's copyright without a prior written authorization;
- cause difficulties in the correct provision of the Platform's services.

Warranties and Limitations of liability of the Platform

PoSeID-on undertakes to use the personal data provided by the Users during the registration in full compliance with the current EU Regulation on the personal data protection. PoSeID-on will take provisional and protective measures to ensure the confidentiality and integrity of the Users' personal data processed during the entire use of the Platform's services, but PoSeID-on will not be responsible for any abusive use or for the loss, alteration or dissemination of information, due to causes not attributable to it.

PoSeID-on will make every reasonable effort to ensure continued access to the Users without interruption to the contents and services, but may not, under any circumstances, be held responsible if one or more of the services are temporarily inaccessible. By way of example, PoSeID-on is not liable for the interruption of services as a result of force majeure

or malfunction of services due to the incorrect functioning of telephone lines, electricity and global and/or national networks. PoSeID-on cannot be held responsible for direct or indirect damages suffered by the Users as a result of the use of the Platform. PoSeID-on undertakes to keep all the contents on the Platform duly updated and without mistakes or omissions but cannot always guarantee it. Consequently, PoSeID-on is not responsible for any mistake or omission that may appear in the contents of the Platform, or that may be due to its inaccuracies, completeness or updating. Once the mistakes or the omissions have been highlighted, PoSeID-on undertakes to correct them as quickly as possible.

PoSeID-on is not responsible and cannot monitor or verify the contents of other websites that can be accessed through the hypertext links found on the Platform, as it cannot recommend their use under any circumstances. PoSeID-on assumes responsibility not to publish on the Platform material that does not meet the requirements established by these Terms and undertakes to remove, as soon as possible, any information or false and misleading content that could damage the Users. The Platform allows the Users the option to report any information or content that does not meet with these Terms and that is therefore inappropriate by writing to the following email address _____. In any case, PoSeID-on reserves the right to remove the contents that it deems to be non-compliant with its own policies.

Intellectual Property Rights

Except where otherwise stated, all the material available on the Platform such as images, photographs, videos, texts are owned by the legitimate copyright holder PoSeID-On and protected by the Italian and European copyright laws. Therefore, it is forbidden to reproduce any material of the Platform, even in part, without a prior PoSeID-on's written authorization.

Any text and material taken from other sources or any trademark of third parties utilized on the Platform, for which prior authorization for the publication has been given, are also owned by the legitimate copyright holders and protected by the copyright law.

Any use of the material in violation of the above-mentioned limits will be objective of legal action by PoSeID-On.

Applicable Law and Jurisdiction Disputes

This Agreement is governed by the Italian Law.

The eventual nullity and/or invalidity, in whole or in part, of one or more items, does not affect the other Articles contained in this contract which, consequently, shall therefore be regarded as fully valid and effective.

Disputes which may arise between the parties relating to the interpretation, execution and validity of this contract shall be to the exclusive jurisdiction of the Court of Rome.

Privacy

The Information provided by the User represent his personal data according to the EU Regulation 2016/679 on the Personal Data Protection.

With reference to the Regulation, PoSeID-on is the Data Controller of the personal data provided by the User in relation to all the aspects connected to the use of the Platform.

PoSeID-on has provided a policy privacy on the processing of personal data, available at the address _____/privacy-policy, which must be accepted by the User before proceeding with the registration and the use of the Platform.

✓ Accept

✓ Dont Accept

Approval of the specific clauses

Pursuant to and for the purposes of articles 1341 and 1342 of the Italian Civil code, the User declares to have read and specifically accepted the following clauses: art. 2 - duration of contract and termination for withdrawal, art. 4 – obligations and responsibilities of the User; art. 5 – warranties and limitations of liability of the Platform; art. 7 - applicable law and jurisdiction disputes.

✓ Accept

✓ Dont Accept