



Policy Cloud
Cloud for Data-Driven Policy Management

CLOUD FOR DATA-DRIVEN POLICY MANAGEMENT

Project Number: 870675

Start Date of Project: 01/01/2020

Duration: 36 months

D7.12 DATA MARKETPLACE: SOFTWARE PROTOTYPE

Dissemination Level	PU
Due Date of Deliverable	31/10/2022, M34
Actual Submission Date	28/10/2022
Work Package	WP7, Communication, Exploitation, Standardisation, Roadmapping & Business Development
Task	T7.2
Type	Demonstrator
Approval Status	
Version	V1.0
Number of Pages	p.1 – p.87

Abstract: The deliverable D7.12 Data Marketplace: Software Prototype describes the final demonstrator of the PolicyCLOUD Data Marketplace. The latter is a unified web-based platform consisting of two (2) core services, its front-end and back-end services, offering to its users various ready-to-use solutions, by supporting different kinds of assets.

The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided "as is" without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/ her sole risk and liability. This deliverable is licensed under a Creative Commons Attribution 4.0 International License.



PolicyCloud has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 870675.

Versioning and Contribution History

Version	Date	Reason	Author
0.1	12/09/2022	ToC	Vasilis Koukos, Argyro Mavrogiorgou (UPRC)
0.2	16/09/2022	Contribution in Sections 1, 2, 4	Thanos Kiourtis (UPRC)
0.3	28/09/2022	Updates in Sections 1, 2, 3	Vasilis Koukos (UPRC), Eleftheria Kouremenou, Alexandros Raikos (UPRC)
0.4	12/10/2022	Check and revision of all Sections	Argyro Mavrogiorgou, Thanos Kiourtis (UPRC)
0.5	18/10/2022	Review	Giannis Ledakis (UBI), Panayiotis Michael (ICCS)
0.6	26/10/2022	Changes based on review comments	Vasilis Koukos, Eleftheria Kouremenou (UPRC)
0.7	27/10/2022	Quality check	Argyro Mavrogiorgou (UPRC)
1.0	28/10/2022	Submitted version	ATOS

Author List

Organisation	Name
UPRC	Vasilis Koukos
UPRC	Eleftheria Kouremenou
UPRC	Alexandros Raikos
UPRC	Argyro Mavrogiorgou
UPRC	Thanos Kiourtis

Abbreviations and Acronyms

Abbreviation/Acronym	Definition
API	Application Programming Interface
AJAX	Asynchronous JavaScript And XML
CRUD	Create Retrieve Update Delete
EOSC	European Open Science Cloud
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
REST	Representational State Transfer
UI	User Interface

Contents

Versioning and Contribution History.....	2
Author List.....	2
Abbreviations and Acronyms	2
Executive Summary	8
1 Introduction.....	9
1.1 Objective of the Deliverable	9
1.2 Structure of the Deliverable	9
1.3 Summary of Changes	9
2 Prototype Overview.....	10
2.1 Main Components	11
2.1.1 Back-end.....	11
2.1.2 Front-end.....	12
2.2 Interfaces	12
2.2.1 Back-end.....	12
2.2.1.1 Interfaces related to Users	15
2.2.1.2 Interfaces related to Solutions.....	28
2.2.1.3 Search functionality on Solutions	56
2.2.1.4 Interfaces related to Assets.....	58
2.2.1.5 Root & Other Interfaces	63
2.2.2 Front-end.....	64
2.3 Baseline Technologies and Tools.....	82
2.3.1 Back-end.....	82
2.3.2 Front-end.....	82
3 Source Code	85
3.1 Availability	85
3.1.1 Back-end.....	85
3.1.2 Front-end.....	85
3.2 Exploitation	85
3.2.1 Back-end.....	85
3.2.2 Front-end.....	85

4	Conclusion	86
	References	87

List of Tables

Table 1 – Back-end’s interfaces related to Users	15
Table 2 – Register a new user Interface	16
Table 3 – Check availability of an Email Interface	17
Table 4 – Authorize a user (Login) Interface	18
Table 5 – Verify users Interface	18
Table 6 – Resend verification code to users Interface	19
Table 7 – Get user’s information Interface	20
Table 8 – Update user’s information Interface.....	21
Table 9 – Change user’s password Interface	21
Table 10 – Reset user’s password request Interface	22
Table 11 – Reset user’s password Interface	23
Table 12 – Delete user’s account Interface	23
Table 13 – Change user’s email Interface	24
Table 14 – Verify user’s new email Interface	24
Table 15 – Revert user’s email Interface	25
Table 16 – Change user’s profile picture Interface	26
Table 17 – Remove user’s profile picture Interface	26
Table 18 – Get user’s statistics Interface	27
Table 19 – Get user’s account data Interface	27
Table 20 – Back-end’s interfaces related to Solutions.....	29
Table 21 – Get solutions’ collections Interface	30
Table 22 – Get a list with all solutions Interface.....	31
Table 23 – Get a list with all solutions from a specific collection Interface	32
Table 24 – Get a specific solution (using keyword “all”) Interface	32
Table 25 – Get a specific solution (using solution’s “collection”) Interface	33
Table 26 – Get latest solutions from all collections Interface	34
Table 27 – Get latest solutions from a specific collection Interface	34
Table 28 – Get random solutions from all collections Interface	35
Table 29 – Get random solutions from a specific collection Interface.....	35
Table 30 – Get a list with all solutions provided by a specific user (using keyword “all”) Interface	37
Table 31 – Get a list with all solutions provided by a specific user and under a specific collection (using a “collection” value) Interface	37
Table 32 – Get solutions’ statistics for front-end’s homepage Interface	38
Table 33 – Get solutions’ filtering values for front-end’s Discover page Interface	38
Table 34 – Upload / Create a new solution with random ID Interface.....	40
Table 35 – Upload / Create a new solution with given ID Interface	42
Table 36 – Update a specific solution (using keyword “all”) Interface.....	43
Table 37 – Update a specific solution (using solution’s “collection”) Interface	44
Table 38 – Delete a specific solution (using keyword “all”) Interface	44
Table 39 – Delete a specific solution (using solution’s “collection”) Interface.....	45

Table 40 – Delete all solutions Interface	46
Table 41 – Delete all solutions from a specific collection Interface	46
Table 42 – Make a review for a solution Interface	47
Table 43 – Update an existing review for a solution Interface.....	47
Table 44 – Delete a review for a solution Interface	48
Table 45 – Get a list with the reviews made by a specific user Interface	49
Table 46 – Get a list with the reviews made for a specific solution Interface	50
Table 47 – Get the cover image of a solution Interface	50
Table 48 – Change the cover image of a solution Interface	51
Table 49 – Remove the cover image of a solution Interface	51
Table 50 – Get a list with all solutions that need permission Interface	52
Table 51 – Get a list with all solutions from a specific collection that need permission Interface	53
Table 52 – Approve or reject a solution that needs permission, using keyword “all” Interface	54
Table 53 – Approve or reject a solution thats needs permission, using solution’s “collection” Interface .	54
Table 54 – Approve or reject all solutions that need permission, using keyword “all” Interface	55
Table 55 – Approve or reject all solutions that need permission under a specific collection, using a “collection” value Interface	55
Table 56 – Back-end’s search operators	57
Table 57 – Back-end’s interfaces related to Assets.....	58
Table 58 – Get a list with the stored assets Interface.....	59
Table 59 – Get a specific asset using its ID Interface.....	59
Table 60 – Upload a new asset with random ID Interface.....	60
Table 61 – Upload a new asset with given ID Interface	61
Table 62 – Update a specific asset using its ID Interface	61
Table 63 – Delete a specific asset using its ID Interface.....	62
Table 64 – Delete all assets (administrators’ action) Interface.....	62
Table 65 – Root Interface	63

List of Figures

Figure 1 – Data Marketplace architecture.....	10
Figure 2 – Data Marketplace’s layers and main functionalities.....	11
Figure 3 – Navigation bar for NON-LOGGED in users	64
Figure 4 – Navigation bar for logged in users	64
Figure 5 – Navigation bar from the Home page	64
Figure 6 – Discover’s sub-items redirect to Discover page	65
Figure 7 – Footer.....	65
Figure 8 – Home page: Upper view.....	66
Figure 9 – Home page: Middle view.....	67
Figure 10 – Home page: Lower view.....	67
Figure 11 – Sign up page	68

Figure 12 – Sign in page.....	69
Figure 13 – Account page: Overview tab.....	70
Figure 14 – Account page: Solutions tab.....	71
Figure 15 – Account page: Reviews tab.....	71
Figure 16 – Account page: Profile tab.....	72
Figure 17 – Discover page.....	72
Figure 18 – Discover page: Sorting options.....	73
Figure 19 – Solution page: View for logged in users.....	74
Figure 20 – Solution page: View for solution providers and administrators.....	74
Figure 21 – Solution page: View for non-logged in users.....	75
Figure 22 – Create page: Basic information.....	76
Figure 23 – Create page: Additional information.....	76
Figure 24 – Create page: Assets section.....	77
Figure 25 – Create page: View for non logged in users.....	77
Figure 26 – About page: Upper view.....	78
Figure 27 – About page: Bottom view.....	79
Figure 28 – Example of error message.....	79
Figure 29 – Terms and conditions page.....	80
Figure 30 – Privacy policy page.....	81
Figure 31 – Contact page.....	81
Figure 32 – Front-end access Middleware.....	83
Figure 33 – Dashboard add settings.....	83
Figure 34 – Dashboard admin view settings.....	84
Figure 35 – Token based actions.....	84

Executive Summary

This deliverable (entitled “Data Marketplace: Software Prototype”) describes the final version of the PolicyCLOUD Data Marketplace demonstrator and is a follow up of the deliverable D7.5 - “Data Marketplace: Software Prototype” where the initial demonstrator of the Data Marketplace was described.

In essence, the Data Marketplace is a unified web-based platform consisting of two (2) core services, the front-end and the back-end services, offering its users various ready-to-use solutions. More specifically, it provides to the wider research and innovation community various assets (i.e., objects/solutions) in different domains.

Into this context, the current deliverable describes an overview of the Data Marketplace architecture, detailing the main features of its core components (also described in D7.4 - Data Marketplace: Design and Open Specification, delivered in August 2021, and its updated version D7.11, delivered in August 2022), whereas all the implemented interfaces are thoroughly described accompanied by indicative examples. On top of these, the baseline technologies that have been used for the realization of the Data Marketplace are analyzed, providing detailed information on how an external user can exploit and access the Data Marketplace.

1 Introduction

Deliverables D7.4, entitled “Data Marketplace: Design and Open Specification” (delivered in August 2021) and its updated version, Deliverable D7.11 (delivered in August 2022), were about the design and the architecture of the PolicyCLOUD Data Marketplace. As described and analysed in these deliverables, the Data Marketplace is considered as a smart user-based repository of assets that aims to create a community of users who will be able, through Data Marketplace’s platform, to provide and share various ready-to-use solutions/tools to various subjects and fields of use, related to the areas of interest of PolicyCLOUD.

1.1 Objective of the Deliverable

This deliverable describes the final version of the implemented prototype of the Data Marketplace, and it is an extension of the abovementioned deliverables, as well as the deliverable D7.5, where the initial version of the implemented prototype was described. In summary, the Data Marketplace has been implemented in order to provide the means for storing, searching and retrieving several types of assets, which are the outcome of a requirements analysis that was performed during Task 7.2 and described in D7.4. It consists of a public web-based environment with many different APIs and functionalities, covering all the different requirements of the project’s stakeholders.

1.2 Structure of the Deliverable

The remainder of this deliverable describes an overview of the Data Marketplace architecture in Section 2.1, detailing the main features of its core components that are also described in the abovementioned deliverables. In Section 2.2, the final version of the implemented interfaces of the Data Marketplace’s components are described, while Section 2.3 describes the baseline technologies that have been used for the realization of the Data Marketplace. Section 3 provides some access information to the source code, and finally, Section 4 concludes with a summary of the described prototype.

1.3 Summary of Changes

This Section highlights the updates made to the previous version of this deliverable D7.5 (Data Marketplace: Software Prototype):

- Included Sub-section 1.3 (Summary of Changes).
- Updated Sub-section 2.1.1 in order to highlight the renaming of the descriptions to “solutions”.
- Updated Sub-sections 2.2.1 and 2.2.2 according to the final updates of the relative components.
- Updated Sub-section 2.3 based on the final baseline technologies (especially for the back-end).
- Updated Section 3 according to the availability and exploitation plans of the components.

2 Prototype Overview

The PolicyCLOUD Data Marketplace (<https://marketplace.policycloud.eu>) is a public web-based environment with various APIs, able to store several types of assets. It has been structured and developed having two (2) core components. The first and most important component is the back-end, which contains in a structured way the information, stores the assets offered by the Data Marketplace and implements the required functionalities. The second component is the front-end, which presents to the users the offered content (the assets and their information), allowing them to interact with the platform in an easier way (Figure 1).

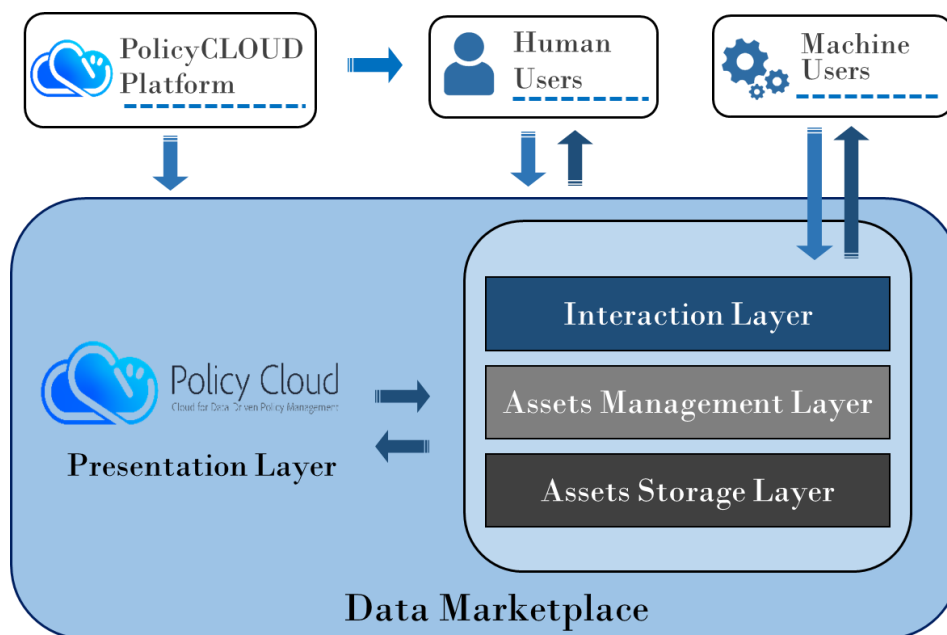


FIGURE 1 – DATA MARKETPLACE ARCHITECTURE

Generally, the Data Marketplace provides several functionalities that are mapped to different layers. The back-end includes three (3) layers (i.e., Assets Storage Layer, Assets Management Layer, and Interaction Layer), while the front-end includes one (1) layer (i.e., Presentation Layer). Hence, the platform consists of four (4) different layers (as depicted in Figure 2) that realize its capabilities. These layers of the Data Marketplace are described below:

- The Assets Storage Layer (part of the back-end) is the layer in which the platform’s offered assets are stored.
- The Assets Management Layer (part of the back-end) delivers all the needed principles and techniques for the management of the Data Marketplace’s assets.
- The Interaction Layer (part of the back-end) supports the communication between the platform and its users (i.e., human users and machine users), by providing discrete APIs for exploiting each different type of asset.

- The Presentation Layer (part of the front-end) provides the User Interface towards the different types of users that are willing to use the platform.

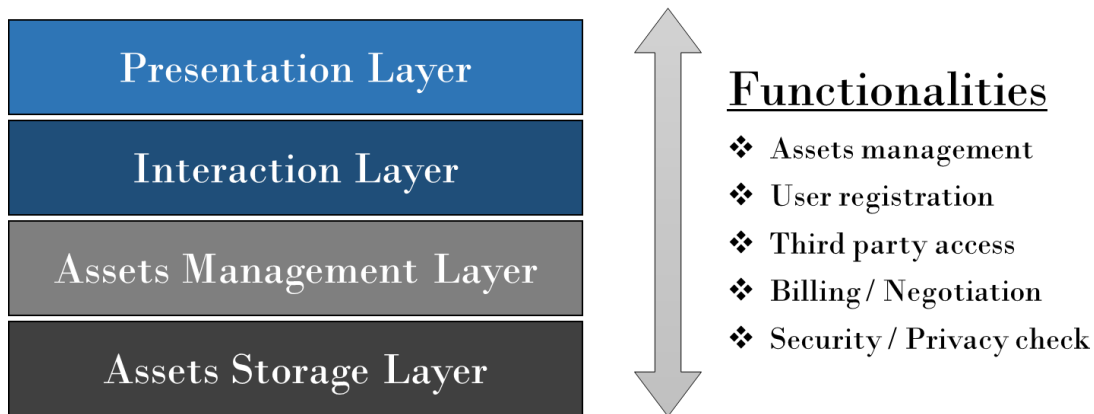


FIGURE 2 – DATA MARKETPLACE'S LAYERS AND MAIN FUNCTIONALITIES

2.1 Main Components

2.1.1 Back-end

The back-end is the main component of the Data Marketplace. It consists of three (3) different layers and implements the main functionalities for the successful management of all the existing assets. The three (3) layers are briefly described below.

The Assets Storage Layer is responsible for storing the assets that will be offered by the Data Marketplace. An essential component of this layer is the database that can store files in any format as well as additional information about the files provided. In this context, the type of the database that is used is a document-oriented NoSQL database, which stores both JSON-like documents (the format of the descriptions files that are analyzed in the Assets Management Layer) and binary files, using extended specifications (e.g., file system).

The Assets Management Layer is responsible for the entire life cycle of the assets within the platform and offers all the needed principles and techniques for their management. Specifically, this layer handles the assets from the moment they are ingested into the platform through the APIs and then stored in the database (in the Assets Storage Layer) until their final deletion from the platform. Through this layer, the Data Marketplace supports the CRUD operations and searching functionality, which are triggered by the corresponding APIs of the back-end (Assets Interaction Layer). The back-end is a REST API and receives different HTTP requests in order to either perform an operation or trigger a functionality. Moreover, there are mandatory description files for all the assets that contain metadata about the described asset (in JSON format). These description files are mandatory to make the assets searchable and retrievable by the end-users of the Data Marketplace. At this point, it should be emphasized that in this final version of the PolicyCLOUD Data Marketplace demonstrator, these description files were renamed to **“solutions”** in

order to describe the actual design flow/process of the assets management, as in the Data Marketplace it is feasible to have one description for more than one asset (i.e., 1 description for many assets - 1:N) and not only one by one (1:1). Thus, the “description of an asset” is converted into a “description of a solution” offered by the «N» assets offered by and associated with a **solution**.

The last layer, the Assets Interaction Layer, is responsible for supporting the communication between the platform and its end-users. It implements the interfaces (APIs) of the back-end (analyzed in Section 2.2.1) that will handle the back-end’s operations. As described before, these APIs receive HTTP requests that trigger the CRUD operations for both assets and description files.

2.1.2 Front-end

The front-end is the fourth layer of the platform, represented by the Presentation Layer. It is a web-based server that presents the offered assets to the users with a friendly UI. In general, the front-end converts all the interfaces of the back-end (REST API) into user friendly interfaces and provides automated forms and processes that make it easier for users to interact with the back-end and benefit from its stored assets. Therefore, it acts as an intermediate among the Data Marketplace users and the back-end, sending the respective HTTP requests to the latter and presenting its responses.

In short, the front-end allows users to register and sign in to the Data Marketplace, upload their offered assets by filling out appropriate forms whose fields are the content of the solution files of the assets (as mentioned in Section 2.1.1); search for assets according to various fields (i.e., title, asset’s type, keywords, text on their description, other metadata, etc.) that can be further filtered or even sorted by the number of views or the date they were uploaded to the Data Marketplace, etc. Also, there is a page that presents in detail the information of the solutions (and their assets), and through this page, the users are able to retrieve the real assets (i.e., the uploaded files). More details about the front-end and its supported functionalities are described in Section 2.2.2.

2.2 Interfaces

This Section provides the description and the core details of the interfaces for both components (i.e., back-end and front-end). The back-end’s subsection describes its interfaces in more technical terms, while the front-end’s subsection describes the webpages that take advantage of the back-end’s interfaces, along with their use cases.

2.2.1 Back-end

As described in Section 2.1.1, the back-end is a REST API that receives HTTP requests to trigger its designed and implemented functionalities. This Section describes the REST API endpoints that are introduced in the first version of the back-end. These APIs are categorized into three (3) main groups, namely: (i) APIs related to Users, (ii) APIs related to Solutions (previously named Descriptions), and (iii) APIs related to Assets.

One of the basic requirements set during the design of the Data Marketplace and described in the deliverable D7.4, was to become a user-based system. There are many reasons for this requirement, starting from the fact that it is a web system/server that will offer its users various types of objects (assets), and to the fact that the assets are offered by their providers/owners to all users without special restrictions. This results in intellectual property rights issues, which are resolved, allowing providers to manage their assets on their own. In case that an offered asset is not provided by its real author and just by a Data Marketplace user/provider, the providers can specify who is the real author/owner, by providing the “legal owner of the asset” information.

Thus, all users of the Data Marketplace should have their personal accounts in the system, which they will be able to manage themselves. As such requirements are very common on all websites, the Data Marketplace’s administrators are not only able to audit the accounts and perform actions that will ensure the platform’s smooth operation, but they are also able to monitor the community that will be created through the Data Marketplace.

As already described, the Data Marketplace consists of two (2) components, running two (2) different servers, both however managing the same information and data, with the storage of these data being done exclusively in the back-end. Specifically, the binaries of the provided assets and their descriptions (i.e., metadata files) are stored in the back-end, as it is done for the users’ data. In addition, both components are accessible to users by direct communication, using HTTP requests for the back-end and through web browsers for the front-end to gain access to the information stored in the back-end. Therefore, based on all these, in order to restrict the access to the information, it was decided that the back-end will be the server that will offer the authentication and authorization mechanisms to the users for the management of its content. It should be noted that the latter was decided based on the fact that the Data Marketplace will be publicly available to all interested users (either they are partners of the PolicyCLOUD consortium or third parties). As a result, since all the offered solutions will be immediately publicly available to these users, the back-end will be independent from the rest of the PolicyCLOUD components, supporting its own authentication and authorization mechanisms to manage its content.

As authorization standard, the JSON Web Token (JWT) [1] technology is used. JWT is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object in a way that can be verified and trusted because it is digitally signed. The JWT is a simple token format and because of its relatively small size, a JWT can be sent through a HTTP request either as a query parameter in the URL or inside the HTTP header, it is transmitted quickly, and it can be used very easy within the context of the HTTP. A JWT contains all the required information about an entity (e.g., information about issuer, subject, expiration time, and any other information) to avoid querying a database more than once. As described before, it is a secure approach as it is digitally signed for tamper proof and authenticity, and it can be encrypted to protect the token information using symmetric or asymmetric approach. It should be noted that by default, a JWT contains the information encoded and not encrypted (the token can be further encrypted). Some extra benefits of the JWT are that it can be used as a stateless authentication mechanism (the back-end as REST API is not able to keep users’ sessions) and finally, the fact that its content is a JSON object (as the assets’ solutions/descriptions) is makes it easier to be used and be parsed by the back-end [2].

The following token is an example of a JWT for the next JSON Object, signed with a symmetric key “key”:

JSON Object: {"username": "vkoukos", "name": "Vasilis", "surname": "Koukos", "Organization": "UPRC", "exp": 1516239022}

JWT:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InZrb3Vrb3MiLCJuaW1lIjoiaWVmFzaWxpcyIsInN1cm5hbWUiOiJLb3Vrb3MiLCJpcmdhbnml6YXRpb24iOiJlVUJlIiwiaXNjaXNTE2MjM5MDIyQy31osTPhrxNFwN-moZsDFEIQq6HcOEe7svcGCGnj9IA
```

The content of a JWT can also be the key “exp” that sets an expiration time for the JWT, thus reducing its validation time, which is useful for the back-end. However, the fact that the information is not encrypted (it is simply encoded) it should not contain sensitive personal data.

The usage of the JWT in the Data Marketplace is as follows:

- The Data Marketplace restricts access to its assets and specifically to all interfaces related to its assets as well as to all HTTP requests, e.g., GET, POST, PUT, DELETE. Regarding the interfaces related to the solutions, the requests to these interfaces are restricted too, excluding however the GET HTTP request, since the solutions should be accessible to all the users (with limited content) because the Data Marketplace promotes its contents to the public.
- The users of the Data Marketplace need to register/create an account (their information will be stored in the back-end). In order to access the stored (and permitted) information, users should use an interface so as to authorize themselves using their credentials. Their authorization results to the retrieval of a JWT, which they can use in their HTTP requests to the Data Marketplace.
- The JWT contains all the necessary information of the users, along with the expiration period. The JWT is signed from the back-end with a secret key (fake JWTs are addressed from the back-end through the signature – brute force attacks are not addressed but can be limited).
- The front-end, during users’ login retrieves their JWTs and uses them on their behalf in the headers of the HTTP requests to the back-end. By decoding the JWTs, the front-end has the most important information of the users. Also, as long as the JWTs are valid (based on the expiration field), it should be kept in the users’ sessions. If a JWT expires, the user’s session must end and therefore, the user must login again in order to get access.
- The back-end, when validating a JWT, decides if a user is actually able to perform an action/access stored information (based also to other rules/restrictions/access rights).
- The expiration time of a JWT is different when users retrieve it making a request directly to the back-end instead of a request through the front-end. The reasons for this decision are: (i) the front-end users will not handle the JWTs by themselves (front-end will do), (ii) they do not have access to it, and (iii) they should have longer sessions (and more time). Unlike front-end users, the users/ services that have direct access to the back-end will be able to have a limited expiration time, as they know and handle JWT (they are also able to share it to third parties as if they were sharing their credentials).

Except for the usage of the JWT, the Data Marketplace supports authorization through Google accounts and through the KeyCloak instance of the PolicyCLOUD, for those that have such credentials.

The interfaces of the back-end are described below.

2.2.1.1 Interfaces related to Users

This group of APIs offers functionalities intended for the management of Data Marketplace’s users. The most important endpoints are those for the user registration as it is necessary for the usage of the other endpoints, and the endpoint for their authorization in order to get a JWT. For all users, except for their personal information, there exists a unique ID (usernames have been removed). The table below presents the endpoints related to users as they are in the final version of the Data Marketplace’s back-end.

Action	HTTP Method	Endpoint
Register a new user (Sign up)	POST	{HOST}/accounts/users/registration
Check availability of an email	GET	{HOST}/accounts/email/availability
Authorize a user (Login)	POST	{HOST}/accounts/users/authorization
Verify users (their email)	GET	{HOST}/accounts/users/verification/{vc}
Resend verification code to users	POST	{HOST}/accounts/users/verification/resend
Get user’s information	GET	{HOST}/accounts/users/information/{uid}
Update user’s information	PUT	{HOST}/accounts/users/information/{uid}
Change user’s password	POST	{HOST}/accounts/users/password/change
Reset user’s password request	POST	{HOST}/accounts/users/password/reset
Reset user’s password	POST	{HOST}/accounts/users/password/reset/{prc}
Delete user’s account	DELETE	{HOST}/accounts/users/delete/{uid}
Change user’s email	PUT	{HOST}/accounts/users/email/{uid}
Verify user’s new email	GET	{HOST}/accounts/users/email/change/confirm/{vc}
Revert user’s email	GET	{HOST}/accounts/users/email/change/revert/{vc}
Change user’s profile picture	PUT	{HOST}/accounts/users/image
Remove user’s profile picture	DELETE	{HOST}/accounts/users/image/{uid}
Get user’s statistics	GET	{HOST}/accounts/users/statistics/{uid}
Get user’s account data	GET	{HOST}/accounts/users/data

TABLE 1 – BACK-END’S INTERFACES RELATED TO USERS

- *{HOST}* refers to the hosting server: the domain name and the port running the back-end.
- *{uid}* refers to “user’s ID”.
- *{vc}* refers to “verification code”.
- *{prc}* refers to “password reser code”.
- Some of these actions require additional fields in the headers of the HTTP request. Example of a required field is the JWT.

Below is a more detailed description of all the developed interfaces and their corresponding actions:

Title:	Register a new user (Sign up)	
Endpoint:	{HOST}/accounts/users/registration	
HTTP Method:	POST	
Description:	<p>From this endpoint, Data Marketplace’s user registrations are made. A POST request should be submitted, and the next JSON schema must be in its body as raw data. It should be noted that a) the email must be unique and available, and b) the schema below should be exactly the same, whether there are values or not (empty strings "") – the array “social” can be empty.</p> <pre> { "account": {"password": "..."}, "info": { "name": "...", "surname": "...", "title": "...", "gender": "...", "organization": "...", "phone": "...", "email": "...", "about": "...", "social": ["...", "..."] } } </pre> <p>The headers of the request may contain the key “x-more-time” that is used only by the front-end in order to get JWTs that are valid for a longer period (greater expiration value).</p>	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json)	
Headers:	<u>Key</u>	<u>Value</u>
	x-more-time	[Restricted and available only for the front-end which use an API key] Front-end’s API key
URL Parameters:	None	
Query Parameters:	None	
Restrictions / Special Features:	None	
Successful Response:	JSON Object with a successful message and user’s JWT.	
The following is an example of the request in cURL:		
<pre> curl -request POST '{HOST}/accounts/users/registration' \ --header 'Content-Type: application/json' -header 'x-more-time: <API_KEY>' \ --data-raw '{ "account": {"password": "..."}, "info": { "name": "...", "surname": "...", "title": "...", "gender": "...", "organization": "...", "phone": "...", "email": "...", "about": "...", "social": ["...", "..."] } }' </pre>		

TABLE 2 – REGISTER A NEW USER INTERFACE

After a successful registration, the following JSON document is stored in the database:

```

{
  "_id": "...", // user's unique ID
  "account": {
    "password": "...", // user's password (hashed)
    "password_protected": "...", // parameter that determines whether the account is
    password protected or not (values 1 or 0)
    "connections": {"google": "...", ... } // object that determines if the account is

```



```

connected to any of the supported SSO services (e.g. Google, Keycloak, etc.)
"role": "user", // user's role (user or admin)
"verified": "...", // value = "1" if user is verified,
otherwise, it has a verification code to use it for user's email/account verification
"registration_datetime": "... // user's registration date
},
"info": { // info provided during user's registration
  "name": "...", "surname": "...", "title": "...", "gender": "...", "organization": "...",
  "phone": "...", "email": "...", "about": "...", "social": []
},
"profile_parameters": {
  "public_email": 0, // parameter that determines if the email will be public or not
  (values 1 or 0)
  "public_phone": 0, // parameter that determines if the phone will be public or not
  (values 1 or 0)
  "profile_image": "default_image_users" // the ID of the user's profile image,
  a default image is used for all users
}
}

```

Title:	Check availability of an email	
Endpoint:	{HOST}/accounts/email/availability	
HTTP Method:	GET	
Description:	This endpoint is used in order to check the availability of an email during the registration of the users. A GET request should be made and the key "x-email" must be included in the headers of the request.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-email	The email whose availability will be checked.
URL Parameters:	None	
Query Parameters:	None	
Restrictions / Special Features:	None	
Successful Response:	Availability status in JSON Object.	
The following is an example of the request in cURL:		
curl -request GET '{HOST}/accounts/email/availability' -header 'x-email: <value>'		

TABLE 3 – CHECK AVAILABILITY OF AN EMAIL INTERFACE

Title:	Authorize a user (Login)
Endpoint:	{HOST}/accounts/users/authorization
HTTP Method:	POST
Description:	<p>Through this endpoint, the users are authorized in order to log in to their account. The next JSON schema, containing users' credentials, must be in the body of the request as raw data.</p> <pre>{ "email": "...", "password": "..." }</pre> <p>A successful response will return the next JSON schema that contains the JWT in the key "token": <code>{ "_status": "successful", "token": "<JWT>" }</code></p> <p>The headers of the request may contain the key "x-more-time" that is used only by the front-end in order to get JWTs that are valid for a longer period (greater expiration value).</p>

Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json)	
Headers:	<u>Key</u>	<u>Value</u>
	x-more-time	[Restricted and available only for the front-end which use an API key] Front-end's API key
URL Parameters:	None	
Query Parameters:	None	
Restrictions / Special Features:	None	
Successful Response:	JSON Object with a successful message and user's JWT.	
The following is an example of the request in cURL:		
<pre>curl -request POST '{HOST}/accounts/users/authorization' \ --header 'x-more-time: <API_KEY>' -header 'Content-Type: application/json' \ --data-raw '{ "email": "...", "password": "..." }'</pre>		

TABLE 4 – AUTHORIZE A USER (LOGIN) INTERFACE

Title:	Verify users (their email)	
Endpoint:	{HOST}/accounts/users/verification/{vc}	
HTTP Method:	GET	
Description:	<p>Through this endpoint, the users can verify their account using the verification code {vc} that they received in their email during their registration. For users' convenience, the email that they receive contains a URL that directs to the front-end. It should be noted that this endpoint is also useful for all the occasions that the users' account gets locked and needs verification again (e.g., change email).</p> <p>The headers of the request may contain the key "x-more-time" that is used only by the front-end in order to get JWTs that are valid for a longer period (greater expiration value).</p>	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-more-time	[Restricted and available only for the front-end which use an API key] Front-end's API key
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	vc	The verification code that sent to user's email.
Query Parameters:	None	
Restrictions / Special Features:	Only the accounts' owners are able to verify their accounts/emails.	
Successful Response:	JSON Object with a successful message and user's JWT.	
The following is an example of the request in cURL:		
<pre>curl -request GET '{HOST}/accounts/users/verification/{vc}' -header 'x-more-time: <API_KEY>'</pre>		

TABLE 5 – VERIFY USERS INTERFACE

Title:	Resend verification code to users	
Endpoint:	{HOST}/accounts/users/verification/resend	
HTTP Method:	POST	
Description:	This endpoint is connected to the endpoint above. Its scope is to resend users' account/email verification codes. It is useful mainly for the back-end's users (those who communicate directly with the back-end) and not for those who use the front-end, because the latter has mechanisms to retrieve users' verification codes and send them to users' emails. This request requires user's JWT in the headers of the request, under the key "x-access-token", in order to authenticate the user.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	None	
Query Parameters:	None	
Restrictions / Special Features:	The endpoint is available only to accounts' owners.	
Successful Response:	JSON Object with the verification code.	
The following is an example of the request in cURL:		
<code>curl -request GET '{HOST}/accounts/users/verification/resend' -header 'x-access-token: <JWT>'</code>		

TABLE 6 – RESEND VERIFICATION CODE TO USERS INTERFACE

Title:	Get user's information	
Endpoint:	{HOST}/accounts/users/information/{uid}	
HTTP Method:	GET	
Description:	This endpoint is used in order to retrieve information about a user. A GET request should be made and the user's ID {uid} is required at the end of the endpoint. Moreover, this endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. It should be noted that the administrators and the accounts' owners are able to retrieve all users' information, while users that retrieve information of other users retrieve only public information. Private information can be users' email and phone, depending on the values of the profile parameters "public_email" and "public_phone". Below are illustrated some examples of retrieved users' information, one by an administrator/account owner (example 1) and one by a user that retrieves another user's information (example 2) - the examples present information retrieval for the same user.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	uid	The ID of the user whose information will be retrieved.
Query Parameters:	None	

Restrictions / Special Features:	The administrators and the accounts' owners are able to retrieve all users' information, while users that retrieve information of other users retrieve only public information.
Successful Response:	JSON Object with a user's information.
The following is an example of the request in cURL:	
<code>curl -request GET '{HOST}/accounts/users/information/{uid}' \</code> <code>--header 'x-access-token: <JWT>'</code>	

TABLE 7 – GET USER'S INFORMATION INTERFACE

Example 1

```
{ "_status": "successful", "result": {
  "account": {"registration_datetime": "...", "role": "user", "verified": "1"},
  "info": {"about": "...", "email": "...", "gender": "...", "name": "...", "organization": "...",
    "phone": "...", "social": [], "surname": "...", "title": "..."},
  },
  "profile_parameters": {
    "profile_image": "default_image_users",
    "public_email": 0, "public_phone": 0
  }, "id": "..."
}}
```

Example 2

```
{ "_status": "successful", "result": {
  "account": {"registration_datetime": "...", "role": "user", "verified": "1"},
  "info": {"about": "...", "gender": "...", "name": "...", "organization": "...",
    "social": [], "surname": "...", "title": "..."},
  },
  "profile_parameters": {"profile_image": "default_image_users"}, "id": "..."
}}
```

Title:	Update user's information	
Endpoint:	{HOST}/accounts/users/information/{uid}	
HTTP Method:	PUT	
Description:	<p>This endpoint handles requests for updating users' information. A PUT request should be made and the next JSON schema (it is flexible and thus may contain fewer fields – but without new fields), containing users' new information, must be in its body as raw data.</p> <pre>{ "info": { "name": "...", "surname": "...", "title": "...", "gender": "...", "organization": "...", "phone": "...", "social": ["...", "..."], "about": "..."}, "profile_parameters": {"public_email": 1, "public_phone": 0}}</pre> <p>Moreover, this endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. It should be noted that only the accounts' owners and the administrators are able to update the information of a user. The latter are not able to change the profile parameters. Also, the headers of the request may contain the key "x-more-time" that is used only by the front-end in order to get JWTs that are valid for a longer period (greater expiration value).</p>	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json)	
Headers:	<u>Key</u>	<u>Value</u>

	x-access-token	Requester's JWT
	x-more-time	[Restricted and available only for the front-end which use an API key] Front-end's API key
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	uid	The ID of the user whose information will be updated.
Query Parameters:	None	
Restrictions / Special Features:	Only the accounts' owners and the administrators are able to update the information of a user.	
Successful Response:	A successful response will return the next JSON Object that contains a new JWT in the key "token": <pre>{ "_status": "successful", "message": "The information of the user with ID '{uid}' has been updated.", "token": "<JWT>" }</pre>	
The following is an example of the request in cURL:		
<pre>curl -request PUT '{HOST}/accounts/users/information/{uid}' \ --header 'x-access-token: <JWT>' -header 'x-more-time: <API_KEY>' \ --header 'Content-Type: application/json' \ --data-raw '{"info": { "name": "...", "surname": "...", ... } }'</pre>		

TABLE 8 – UPDATE USER'S INFORMATION INTERFACE

Title:	Change user's password	
Endpoint:	{HOST}/accounts/users/password/change	
HTTP Method:	POST	
Description:	<p>This endpoint is used when the users want to change their account's password. A POST request should be made and the next JSON schema, containing users' new and old password, must be in its body as raw data. Also, this endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. It should be noted that this action is only available to accounts' owners.</p> <pre>{ "old_password": "...", "new_password": "...", "confirm_new_password": "..." }</pre>	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json)	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	None	
Query Parameters:	None	
Restrictions / Special Features:	Only available to accounts' owners. The new password must not be the same with previous password.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl -request POST '{HOST}/accounts/users/password/change' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-raw '{ "old_password": "...", "new_password": "...", "confirm_new_password": "..." }'</pre>		

TABLE 9 – CHANGE USER'S PASSWORD INTERFACE

Title:	Reset user's password request	
Endpoint:	{HOST}/accounts/users/password/reset	
HTTP Method:	POST	
Description:	<p>This endpoint handles the first step of the password reset process. The users who forgot their password have to make a password reset request first, sending a POST request to this endpoint with the next JSON schema in its body. It should be noted that it is not necessary to use both fields – at least one of the two fields is sufficient/required.</p> <pre>{ "uid": "...", "email": "..." }</pre> <p>Another important note is that this endpoint is available only through the mechanisms of the front-end that sends to the users' emails a password reset link that contains a generated password reset code. The generated password reset codes are valid only for an hour (1 hour). The password reset link redirects to a front-end's form from which the users can set their new password. After the submission of the form, the front-end uses the next interface in order to change the password of the user. The headers of the request must contain the front-end's API key.</p>	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json)	
Headers:	<u>Key</u>	<u>Value</u>
	x-api-key	Front-end's API key
URL Parameters:	None	
Query Parameters:	None	
Restrictions / Special Features:	Only available to the front-end.	
Successful Response:	JSON Object with a successful message and the password reset code in its content.	
The following is an example of the request in cURL:		
<pre>curl -request POST '{HOST}/accounts/users/password/reset' -header 'x-api-key: <API_KEY>' \ --header 'Content-Type: application/json' -data-raw '{ "uid": "...", "email": "..." }'</pre>		

TABLE 10 – RESET USER'S PASSWORD REQUEST INTERFACE

Title:	Reset user's password	
Endpoint:	{HOST}/accounts/users/password/reset/{prc}	
HTTP Method:	POST	
Description:	<p>This endpoint is connected to the above endpoint and handles the second step of the password reset process. When the users open the password reset link that they received in their email, they are redirected to a front-end's form from which they are able to set their new password. After the submission of the form, the front-end sends a request to the current interface to finish the process. The password reset code {prc} that the users received in their email must be in the request's URL and the following JSON schema should be in the body of the request.</p> <pre>{ "new_password": "...", "confirm_new_password": "..." }</pre>	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json)	
Headers:	None	
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	prc	The password reset code that the users' received in their email.

Query Parameters:	None
Restrictions / Special Features:	None
Successful Response:	JSON Object with a successful message.
The following is an example of the request in cURL:	
<pre>curl -request POST '{HOST}/accounts/users/password/reset/{prc}' \ --header 'Content-Type: application/json' \ --data-raw '{ "new_password": "...", "confirm_new_password": "..."}'</pre>	

TABLE 11 – RESET USER'S PASSWORD INTERFACE

Title:	Delete user's account	
Endpoint:	{HOST}/accounts/users/delete/{uid}	
HTTP Method:	DELETE	
Description:	<p>In order to delete an account, this endpoint should be used, making a DELETE request and providing requester's password in its body, as raw data (JSON format). The endpoint must contain the user's ID {uid} at the end of the URL.</p> <pre>{ "password": "..." }</pre> <p>The endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. This action is available to accounts' owners and to administrators who are able to delete users from the Data Marketplace. If the action is made by an administrator, the value of the field "password" in the body should be the password of administrator.</p> <p><u>An important note is that the deletion of an account has as result the deletion of all user's data, offered solutions and assets.</u></p>	
Body Data:	Raw (JSON) Data - as the above schema (Content-Type: application/json)	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	uid	The ID of the user whose account will be deleted.
Query Parameters:	None	
Restrictions / Special Features:	Only the accounts' owners and the administrators are able to delete an account/user.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl --request DELETE '{HOST}/accounts/users/delete/{uid}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-raw '{ "password": "..." }'</pre>		

TABLE 12 – DELETE USER'S ACCOUNT INTERFACE

Title:	Change user's email	
Endpoint:	{HOST}/accounts/users/email/{uid}	
HTTP Method:	PUT	
Description:	<p>This endpoint is used to update the emails of the users. This action is also possible through the endpoint for update users' information, but it is important to have the current endpoint because the email is an important field for all accounts. The next JSON schema must be in the request's body as raw data:</p> <pre>{ "new_email": "..." }</pre>	

	The endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. This action is available only to accounts owners and administrators. If the action is made by the accounts' owners, their accounts will get locked until they will verify their new email (using the endpoint for the emails' verification). In case that this action is made by an administrator, the account does not get locked. Also, the headers of the request may contain the key "x-more-time" that is used only by the front-end to get JWTs that are valid for a longer period (greater expiration value).	
Body Data:	Raw (JSON) Data - as the above schema (Content-Type: application/json)	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
	x-more-time	[Restricted and available only for the front-end which use an API key] Front-end's API key
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	uid	The ID of the user whose email will be updated.
Query Parameters:	None	
Restrictions / Special Features:	Only the accounts' owners and the administrators are able to update users' email.	
Successful Response:	JSON Object with a successful message along with a new JWT. It may contain a verification code only if the action is made by accounts' owners.	
The following is an example of the request in cURL:		
<pre>curl --request PUT '{HOST}/accounts/users/email/{uid}' \ --header 'x-access-token: <JWT>' --header 'Content-Type: application/json' \ --data-raw '{ "new_email": "..." }'</pre>		

TABLE 13 – CHANGE USER'S EMAIL INTERFACE

Title:	Verify user's new email	
Endpoint:	{HOST}/accounts/users/email/change/confirm/{vc}	
HTTP Method:	GET	
Description:	Through this endpoint, the users can verify their new email using the verification code {vc} that they received in their email after changing it through the previous interface. For users' convenience, the email that they receive contains a URL that directs to the front-end.	
Body Data:	None	
Headers:	None	
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	vc	The verification code that sent to user's email.
Query Parameters:	None	
Restrictions / Special Features:	Only the accounts' owners are able to verify their emails.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl --request GET '{HOST}/accounts/users/email/change/confirm/{vc}'</pre>		

TABLE 14 – VERIFY USER'S NEW EMAIL INTERFACE

Title:	Revert user's email	
Endpoint:	{HOST}/accounts/users/email/change/revert/{vc}	
HTTP Method:	GET	
Description:	Through this endpoint, the users can revert their old email after an email change of their account, using the verification (<u>revert</u>) code {vc} that they received in their email after changing it through the previous interfaces. For users' convenience, the email that they receive contains a URL that directs to the front-end. The account owners have a period of 14 days after changing their account email address in order to revert to their old email. Once this period passes, the owners will not be able to revert to their old email and the change to the new email address will be permanent.	
Body Data:	None	
Headers:	None	
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	vc	The verification (revert) code that sent to user's email.
Query Parameters:	None	
Restrictions / Special Features:	Only the accounts' owners are able to revert to their old emails.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<code>curl --request GET '{HOST}/accounts/users/email/change/revert/{vc}'</code>		

TABLE 15 – REVERT USER'S EMAIL INTERFACE

Title:	Change user's profile picture	
Endpoint:	{HOST}/accounts/users/image	
HTTP Method:	PUT	
Description:	All users have a default profile image from their registration and through this endpoint are able to change it. The endpoint is restricted and available only to accounts' owners and thus, the JWT of a requester must be included in the headers of the request. Also, the headers of the request may contain the key "x-more-time" that is used only by the front-end in order to get JWTs that are valid for a longer period (greater expiration value).	
Body Data:	<u>Data Type:</u>	Form Data
	<u>Key</u>	<u>Value</u>
	asset	Binary data / Path to image
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
	x-more-time	[<u>Restricted and available only for the front-end which use an API key</u>] Front-end's API key
	x-mimetype	Image's mimetype (<u>Only JPEG and PNG images are allowed</u>)
URL Parameters:	None	
Query Parameters:	None	
Restrictions / Special Features:	Available only to accounts' owners.	

Successful Response:	A successful response will return the next JSON Object that contains a new JWT in the key "token": <pre>{"_status": "successful", "message": "The profile image of the user with ID '{uid}' has been changed.", "token": "<JWT>"}</pre>
The following is an example of the request in cURL:	
<pre>curl --request POST '{HOST}/accounts/users/image' --header 'x-access-token: <JWT>' \ --header 'x-more-time: <API_KEY>' --header 'x-mimetype: <image's mimetype>' \ --form 'asset=@"<full_path_to_image>"'</pre>	

TABLE 16 – CHANGE USER'S PROFILE PICTURE INTERFACE

Title:	Remove user's profile picture	
Endpoint:	{HOST}/accounts/users/image/{uid}	
HTTP Method:	DELETE	
Description:	<p>This endpoint is used to delete users' profile images. The ID {uid} of the user whose profile image will be deleted should be in the URL. This action deletes users' images and replaces them with the default image that is used for all users.</p> <p>The endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. It should be noted that only the accounts' owners and the administrators are able to delete users' profile image. Also, the headers of the request may contain the key "x-more-time" that is used only by the front-end to get JWTs that are valid for a longer period (greater expiration value).</p>	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
	x-more-time	<u>[Restricted and available only for the front-end which use an API key]</u> Front-end's API key
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	uid	ID of user whose profile image will be deleted.
Query Parameters:	None	
Restrictions / Special Features:	Available only to accounts' owners and administrators.	
Successful Response:	A successful response will return the next JSON Object that contains a new JWT in the key "token": <pre>{"_status": "successful", "message": "The profile image of the user with ID '{uid}' has been removed.", "token": "<JWT>"}</pre>	
The following is an example of the request in cURL:		
<pre>curl --request DELETE '{HOST}/accounts/users/image/{uid}' \ --header 'x-access-token: <JWT>' --header 'x-more-time: <API_KEY>'</pre>		

TABLE 17 – REMOVE USER'S PROFILE PICTURE INTERFACE

Title:		Get user's statistics	
Endpoint:	{HOST}/accounts/users/statistics/{uid}		
HTTP Method:	GET		
Description:	This endpoint is used to get some statistics about a user whose ID {uid} is in the URL of the GET request. It is used in users' profiles where their contribution with offerings to the Data Marketplace is presented. The endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request.		
Body Data:	None		
Headers:	<u>Key</u>	<u>Value</u>	
	x-access-token	Requester's JWT	
URL Parameters:	<u>Parameter</u>	<u>Value</u>	
	uid	The ID of the user whose statistics will be retrieved.	
Query Parameters:	None		
Restrictions / Special Features:	Available to all authorized users.		
Successful Response:	JSON Object with a successful message and statistics as follows: <pre>{ "_status": "successful", "results": { "total_solutions": 0, "approved_solutions": 0, "assets_uploaded": 0, "total_links_provided": 0, "total_downloads": 0, "total_views": 0, "total_reviews": 0, "average_rating": 0 } }</pre>		
The following is an example of the request in cURL:			
<code>curl -request GET '{HOST}/accounts/users/statistics/{uid}' -header 'x-access-token: <JWT>'</code>			

TABLE 18 – GET USER'S STATISTICS INTERFACE

Title:		Get user's account data	
Endpoint:	{HOST}/accounts/users/data		
HTTP Method:	GET		
Description:	This endpoint, which is available only to accounts' owners, returns all the personalized data of the requester, returning users' information, uploaded solutions, solutions' reviews and other statistics. The endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request.		
Body Data:	None		
Headers:	<u>Key</u>	<u>Value</u>	
	x-access-token	Requester's JWT	
URL Parameters:	None		
Query Parameters:	None		
Restrictions / Special Features:	Available only to accounts' owners.		
Successful Response:	A JSON Object with users' data (as file).		
The following is an example of the request in cURL:			
<code>curl -request GET '{HOST}/accounts/users/data' -header 'x-access-token: <JWT>'</code>			

TABLE 19 – GET USER'S ACCOUNT DATA INTERFACE

2.2.1.2 Interfaces related to Solutions

This group of APIs offers functionalities intended for the management of the solutions. They support all CRUD operations as well as the search functionality. Special emphasis was placed on the APIs for the solutions' retrieval, extending them so as to get the latest solutions or even random solutions either from a specific collection (i.e., database collection) or from all the collections at once, using the keyword "all". The collections of the database as well as the Data Marketplace's offered types of assets, vary. The current list of the collections can be found at the end of Table 20, which presents the endpoints related to the Solutions as they were stated in the first version of the back-end.

Action	HTTP Method	Endpoint
Get solutions' collections	GET	{HOST}/solutions
Get a list with all solutions	GET	{HOST}/solutions/all
Get a list with all solutions from a specific collection	GET	{HOST}/solutions/{collection}
Get a specific solution (using keyword "all")	GET	{HOST}/solutions/all/{sid}
Get a specific solution (using solution's "collection")	GET	{HOST}/solutions/{collection}/{sid}
Get the latest solutions from all collections	GET	{HOST}/solutions/all/latest
Get the latest solutions from a collection	GET	{HOST}/solutions/{collection}/latest
Get random solutions from all collections	GET	{HOST}/solutions/all/random
Get random solutions from a specific collection	GET	{HOST}/solutions/{collection}/random
Get a list with all solutions provided by a specific user (using keyword "all")	GET	{HOST}/solutions/provider/{uid}/all
Get a list with all solutions provided by a specific user and under a specific collection (using a "collection" value)	GET	{HOST}/solutions/provider/{uid}/{collection}
Get solutions' statistics for front-end's homepage	GET	{HOST}/solutions/statistics/homepage
Get solutions' filtering values for front-end's Discover page	GET	{HOST}/solutions/statistics/filtering
Upload / Create a new solution with random ID	POST	{HOST}/solutions/{collection}
Upload / Create a new solution with given ID	POST	{HOST}/solutions/{collection}/{given_id}
Update a specific solution (using keyword "all")	PUT	{HOST}/solutions/all/{sid}
Update a specific solution (using solution's "collection")	PUT	{HOST}/solutions/{collection}/{sid}
Delete a specific solution (using keyword "all")	DELETE	{HOST}/solutions/all/{sid}
Delete a specific solution (using solution's "collection")	DELETE	{HOST}/solutions/{collection}/{sid}
Delete all solutions (administrators' action)	DELETE	{HOST}/solutions/all/all
Delete all solutions from a specific collection (administrators' action)	DELETE	{HOST}/solutions/{collection}/all
Make a review for a solution	POST	{HOST}/solutions/review/{sid}
Update an existing review for a solution	PUT	{HOST}/solutions/review/{sid}
Delete a review for a solution	DELETE	{HOST}/solutions/review/{sid}

Get the cover image of a solution	GET	{HOST}/solutions/image/{sid}
Change the cover image of a solution	PUT	{HOST}/solutions/image/{sid}/{img_id}
Remove the cover image of a solution	DELETE	{HOST}/solutions/image/{sid}
Get a list with the reviews made by a specific user	GET	{HOST}/solutions/review/{uid}
Get a list with the reviews made for a specific solution	GET	{HOST}/solutions/reviews/{sid}
Get a list with all solutions that need permission (administrators' action)	GET	{HOST}/solutions/permit/all
Get a list with all solutions from a specific collection that need permission (administrators' action)	GET	{HOST}/solutions/permit/{collection}
Approve or reject a solution that needs permission, using keyword "all" (administrators' action)	POST	{HOST}/solutions/permit/all/{sid}
Approve or reject a solution that needs permission, using solution's "collection" (administrators' action)	POST	{HOST}/solutions/permit/{collection}/{sid}
Approve or reject all solutions that need permission, using keyword "all" (administrators' action)	POST	{HOST}/solutions/permit/all/all
Approve or reject all solutions that need permission under a specific collection, using a "collection" value (administrators' action)	POST	{HOST}/solutions/permit/{collection}/all

TABLE 20 – BACK-END'S INTERFACES RELATED TO SOLUTIONS

- *{HOST}* refers to the hosting server: the domain name and the port running the back-end.
- *{sid}* refers to the ID of a specific solution.
- *{given_id}* is used in upload solution action, providing new solution's ID.
- As a *{collection}* can be one of the following values derived from the current types of offered assets:
{"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}
- Some of these actions require additional fields in the headers of the HTTP request. Example of a required field is the JWT.

Below is a more detailed description of all table's interfaces/actions:

Title:	Get solutions' collections
Endpoint:	{HOST}/solutions
HTTP Method:	GET
Description:	This endpoint returns a list with the sub-routes of the "solution" endpoint. More specifically, it returns the values of the {collection} that also refer to the database's collections and the types of the offered assets.
Body Data:	None
Headers:	None
URL Parameters:	None

Query Parameters:	None
Restrictions / Special Features:	None
Successful Response:	A text/plain list with the back-end's collections.
The following is an example of the request in cURL:	
<code>curl --request GET '{HOST}/solutions'</code>	

TABLE 21 – GET SOLUTIONS' COLLECTIONS INTERFACE

Title:	Get a list with all solutions					
Endpoint:	{HOST}/solutions/all					
HTTP Method:	GET					
Description:	<p>A request to this endpoint will result in the retrieval of the stored solutions from all the collections. It uses the keyword “all” instead of a specific collection, which makes the platform to retrieve solutions from all the collections at once. The solutions that return from this request are in a short schema (short description), meaning that the retrieved information is limited. An example of a solution's description in short schema is the following JSON schema:</p> <pre>{ "collection": "tools", "id": "tools_v1LZWaoQN1Fe ", "info": { "keywords": ["information"], "owner": "Vasilis Koukos", "short_desc": "This is an example", "type": "tools", "title": "Example title." }, "main_image": "default_image_assets", "metadata": { "provider": "vkoukos", "reviews": { "average_rating": 4.2, "no_reviews": 14 }, "updateDate": "...", "uploadDate": "...", "views": 35 } }</pre> <p>This endpoint can get query parameters to search for solutions that meet certain conditions. As a query parameter can be any pair of key-value, while additional search operators can be used for more advanced and enhanced search. More details about searching can be found in Section 2.2.1.3. Also, this endpoint offers some standard query parameters, as described below (Query Parameters).</p>					
Body Data:	None					
Headers:	None					
URL Parameters:	None					
Query Parameters:	<table border="1"> <thead> <tr> <th><u>Key</u></th> <th><u>Value</u></th> </tr> </thead> <tbody> <tr> <td>sortBy</td> <td> <p>[Optional] Sorts the solutions by a field – the <u>default</u> is the “<u>newest</u>” key. The value should be one of the following:</p> <p>"newest": sort by date in descending order.</p> <p>"oldest": sort by date in ascending order.</p> <p>"rating-asc": sort by average rating in ascending order.</p> <p>"rating-desc": sort by average rating in descending order.</p> <p>"views-asc": sort by the number of views in ascending order.</p> </td> </tr> </tbody> </table>	<u>Key</u>	<u>Value</u>	sortBy	<p>[Optional] Sorts the solutions by a field – the <u>default</u> is the “<u>newest</u>” key. The value should be one of the following:</p> <p>"newest": sort by date in descending order.</p> <p>"oldest": sort by date in ascending order.</p> <p>"rating-asc": sort by average rating in ascending order.</p> <p>"rating-desc": sort by average rating in descending order.</p> <p>"views-asc": sort by the number of views in ascending order.</p>	
<u>Key</u>	<u>Value</u>					
sortBy	<p>[Optional] Sorts the solutions by a field – the <u>default</u> is the “<u>newest</u>” key. The value should be one of the following:</p> <p>"newest": sort by date in descending order.</p> <p>"oldest": sort by date in ascending order.</p> <p>"rating-asc": sort by average rating in ascending order.</p> <p>"rating-desc": sort by average rating in descending order.</p> <p>"views-asc": sort by the number of views in ascending order.</p>					

		<p>"views-desc": sort by the number of views in descending order.</p> <p>"title": sort by title in ascending order.</p>
	itemsPerPage	<p>[Optional] Returns the results separated in pages (arrays) of N items. The number N is specified by the value of this key. The value N must be an integer number greater or equal to 1. If the key is not used or has a non-accepted value, the results are returned on a single page.</p>
	page	<p>[Optional] This key can only be used if the "itemsPerPage" key is also used. If it is used, it returns only the specified (by key's value) page instead of all pages created using the key "itemsPerPage". The value must be an integer number greater or equal to 1. The default value is 0, which means that all pages will be returned.</p>
	Any key to search (refer to section 2.2.1.3)	Any value to search (refer to Section 2.2.1.3).
Restrictions / Special Features:	None	
Successful Response:	A JSON Object with the results (all solutions from all collections). If the query parameter "itemsPerPage" is used, then the results contain the total number of the pages.	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/solutions/all' + curl --request GET '{HOST}/solutions/all?sortBy={value}' + curl --request GET '{HOST}/solutions/all?itemsPerPage={value}' + curl --request GET '{HOST}/solutions/all?itemsPerPage={value}&page={value}' + curl --request GET '{HOST}/solutions/all?sortBy={value}&itemsPerPage={value}' + curl --request GET '{HOST}/solutions/all?sortBy={value}&itemsPerPage={value}&page={value}'</pre>		
Example of retrieving the 10 most viewed solutions:		
<pre>+ curl --request GET '{HOST}/solutions/all?sortBy=views-desc&itemsPerPage=10&page=1'</pre>		

TABLE 22 – GET A LIST WITH ALL SOLUTIONS INTERFACE

Title:	Get a list with all solutions from a specific collection				
Endpoint:	{HOST}/solutions/{collection}				
HTTP Method:	GET				
Description:	This request is similar to the above request. The only difference between the two (2) actions is that this request retrieves solutions from a specific collection (instead of using keyword "all"). For more details, refer to the above endpoint.				
Body Data:	None				
Headers:	None				
URL Parameters:	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>collection</td> <td>Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}</td> </tr> </tbody> </table>	Parameter	Value	collection	Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}
Parameter	Value				
collection	Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}				
Query Parameters:	As in the above request.				

Restrictions / Special Features:	None
Successful Response:	A JSON Object with the results (all solutions in a specific collection).
The following is an example of the request in cURL:	
<pre>+ curl --request GET '{HOST}/solutions/{collection}' + curl --request GET \ '{HOST}/solutions/{collection}?sortBy={value}&itemsPerPage={value}&page={value}'</pre>	

TABLE 23 – GET A LIST WITH ALL SOLUTIONS FROM A SPECIFIC COLLECTION INTERFACE

Title:	Get a specific solution (using keyword “all”)					
Endpoint:	{HOST}/solutions/all/{sid}					
HTTP Method:	GET					
Description:	<p>With this request, the users are able to retrieve a specific solution. The retrieval of a specific solution is possible using its unique identification code (ID), known when uploading it. Also, the retrieval of a specific solution can be done using both keyword “all” and the name of the collection that the solution has been stored (next interface). This is feasible because the back-end ensures that the IDs are unique regardless of the collection that the solution has been stored. Moreover, the retrieval of a specific solution requires a JWT to be retrieved in its “full schema”. If requester’s JWT is missing, then the endpoint returns the short schema of the solution’s description. Example of a full schema is in the endpoint that handles the uploading of a solution. This endpoint, except for the full schema, also returns the reviews of the specified solution.</p>					
Body Data:	None					
Headers:	<table border="1"> <thead> <tr> <th>Key</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>x-access-token</td> <td>[Optional, it should be used in order to retrieve the full schema of a solution’s description] Requester’s JWT</td> </tr> </tbody> </table>	Key	Value	x-access-token	[Optional, it should be used in order to retrieve the full schema of a solution’s description] Requester’s JWT	
Key	Value					
x-access-token	[Optional, it should be used in order to retrieve the full schema of a solution’s description] Requester’s JWT					
URL Parameters:	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>sid</td> <td>The ID of the solution that will be retrieved.</td> </tr> </tbody> </table>	Parameter	Value	sid	The ID of the solution that will be retrieved.	
Parameter	Value					
sid	The ID of the solution that will be retrieved.					
Query Parameters:	None					
Restrictions / Special Features:	The full schema is available only to authorized (and verified) users, otherwise, the short schema is available to all.					
Successful Response:	A JSON Object with the solution in the results.					
The following is an example of the request in cURL:						
<pre>+ curl --request GET '{HOST}/solutions/all/{sid}' + curl --request GET '{HOST}/solutions/all/{sid}' --header 'x-access-token: <JWT>'</pre>						

TABLE 24 – GET A SPECIFIC SOLUTION (USING KEYWORD “ALL”) INTERFACE

Title:	Get a specific solution (using solution's "collection")	
Endpoint:	{HOST}/solutions/{collection}/{sid}	
HTTP Method:	GET	
Description:	This request is similar to the above request, with the difference that it uses solution's collection for the retrieval of a solution (instead of using keyword "all"). The value of the {collection} must be the collection in which the solution has been stored. More information about the endpoint can be found above.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	[Optional, it should be used in order to retrieve the full schema of a solution's description] Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}
	sid	The ID of the solution that will be retrieved.
Query Parameters:	None	
Restrictions / Special Features:	The full schema is available only to authorized (and verified) users, otherwise, the short schema is available to all.	
Successful Response:	A JSON Object with the solution in the results.	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/solutions/{collection}/{sid}' + curl --request GET '{HOST}/solutions/{collection}/{sid}' \ --header 'x-access-token: <JWT>'</pre>		

TABLE 25 – GET A SPECIFIC SOLUTION (USING SOLUTION'S "COLLECTION") INTERFACE

Title:	Get latest solutions from all collections	
Endpoint:	{HOST}/solutions/all/latest	
HTTP Method:	GET	
Description:	<p>This request is used to retrieve the most recent uploaded solutions sorted based on the date that they have been uploaded, with the most recent being on the top of the list. This request uses the keyword "all" and returns the K latest solutions from all collections. The value of K can be specified through the query parameter "max" (the default value is 20). The solutions are returned in their short schema.</p> <p>This endpoint can get query parameters to search for solutions that meet certain conditions. As a query parameter can be any pair of key-value, while additional search operators can be used for more advanced and enhanced search. More details about searching can be found in Section 2.2.1.3. Finally, the endpoint "Get a list with all solutions" can return the same results as the current, if the example at the end will be followed.</p>	
Body Data:	None	
Headers:	None	
URL Parameters:	None	
Query Parameters:	<u>Key</u>	<u>Value</u>

	max	Integer value greater than 0 – Default: 20
	Any key to search (refer to section 2.2.1.3)	Any value to search (refer to section 2.2.1.3).
Restrictions / Special Features:	None	
Successful Response:	A JSON Object with the results (latest solutions from all collections).	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/solutions/all/latest' + curl --request GET '{HOST}/solutions/all/latest?max=5'</pre>		
<u>Example of similar response by the endpoint "Get a list with all solutions":</u>		
<pre>+ curl --request GET '{HOST}/solutions/all?sortBy=newest&itemsPerPage=20&page=1'</pre>		

TABLE 26 – GET LATEST SOLUTIONS FROM ALL COLLECTIONS INTERFACE

Title:	Get latest solutions from a specific collection	
Endpoint:	{HOST}/solutions/{collection}/latest	
HTTP Method:	GET	
Description:	<p>This request is similar to the above request. It uses the value of a specific collection and not the keyword "all", which results to return sorted the K most recent solutions of the provided collection. The value of K can be specified through query parameter "max" (the default value is 20). The solutions are returned in their short schema. This endpoint can get query parameters to search for solutions that meet certain conditions. As a query parameter can be any pair of key-value, while additional search operators can be used for more advanced and enhanced search. More details about searching can be found in Section 2.2.1.3. Finally, the endpoint "Get a list with all solutions from a specific collection" can return the same results as the current, if the example at the end will be followed.</p>	
Body Data:	None	
Headers:	None	
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}
Query Parameters:	<u>Key</u>	<u>Value</u>
	max	Integer value greater than 0 – Default: 20
	Any key to search (refer to section 2.2.1.3)	Any value to search (refer to section 2.2.1.3).
Restrictions / Special Features:	None	
Successful Response:	A JSON Object with the results (latest solutions of a collection).	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/solutions/{collection}/latest' + curl --request GET '{HOST}/solutions/{collection}/latest?max=5'</pre>		
<u>Example of similar response by the endpoint "Get a list with all solutions from a specific collection":</u>		
<pre>+ curl --request GET '{HOST}/solutions/{collection}?sortBy=newest&itemsPerPage=20&page=1'</pre>		

TABLE 27 – GET LATEST SOLUTIONS FROM A SPECIFIC COLLECTION INTERFACE

Title:	Get random solutions from all collections	
Endpoint:	{HOST}/solutions/all/random	
HTTP Method:	GET	
Description:	This endpoint returns a number of random solutions from all collections (uses keyword “all”). It is useful in order to suggest and promote different solutions each time. It is also used in the home page of the Data Marketplace, where random solutions are displayed. Through the query parameter “max” it can return K solutions, where K can be specified by the users (the default value is 4). The solutions are returned in their short schema.	
Body Data:	None	
Headers:	None	
URL Parameters:	None	
Query Parameters:	<u>Key</u>	<u>Value</u>
	max	Integer value greater than 0 – Default: 20
Restrictions / Special Features:	None	
Successful Response:	A JSON Object with the results (random solutions from all collections).	
The following is an example of the request in cURL:		
+ curl --request GET '{HOST}/solutions/all/random'		
+ curl --request GET '{HOST}/solutions/all/random?max=5'		

TABLE 28 – GET RANDOM SOLUTIONS FROM ALL COLLECTIONS INTERFACE

Title:	Get random solutions from a specific collection	
Endpoint:	{HOST}/solutions/{collection}/random	
HTTP Method:	GET	
Description:	This endpoint is similar to the above endpoint. Instead of keyword “all” it uses a specific collection and thus it returns a number of K random solutions of the provided specific collection. The value of K can be specified through query parameter “max” (the default value is 4). The solutions are returned in their short schema.	
Body Data:	None	
Headers:	None	
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}
Query Parameters:	<u>Key</u>	<u>Value</u>
	max	Integer value greater than 0 – Default: 20
Restrictions / Special Features:	None	
Successful Response:	A JSON Object with the results (random solutions of a collection).	
The following is an example of the request in cURL:		
+ curl --request GET '{HOST}/solutions/{collection}/random'		
+ curl --request GET '{HOST}/solutions/{collection}/random?max=5'		

TABLE 29 – GET RANDOM SOLUTIONS FROM A SPECIFIC COLLECTION INTERFACE

Title:	Get a list with all solutions provided by a specific user (using keyword “all”)
---------------	--

Endpoint:	{HOST}/solutions/provider/{uid}/all	
HTTP Method:	GET	
Description:	<p>This request returns all the solutions that have been provided by the user whose ID {uid} is part of the request's URL. It uses the keyword "all" instead of a specific collection, which makes the platform to retrieve its provided solutions from all the collections at once. The solutions are returned in their short schema. Also, the endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. It should be noted that the accounts' owners who use this endpoint to retrieve their uploaded solutions, except for the retrieval of the approved solutions, are able to also retrieve "pending" solutions (e.g., the solutions that they uploaded and need administrators' approval). Finally, the endpoint offers some standard query parameters that specify the format of the results and are described below (Query Parameters).</p>	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	uid	The ID of the user whose offered solutions will be retrieved.
Query Parameters:	<u>Key</u>	<u>Value</u>
	sortBy	<p>[Optional] Sorts the solutions by a field – the default is the "newest" key. The value should be one of the following:</p> <p>"newest": sort by date in descending order. "oldest": sort by date in ascending order. "rating-asc": sort by average rating in ascending order. "rating-desc": sort by average rating in descending order. "views-asc": sort by the number of views in ascending order. "views-desc": sort by the number of views in descending order. "title": sort by title in ascending order.</p>
	itemsPerPage	[Optional] Returns the results separated in pages (arrays) of N items. The number N is specified by the value of this key. The value N must be an integer number greater or equal to 1. If the key is not used or has a non-accepted value, the results are returned on a single page.
	page	[Optional] This key can only be used if the "itemsPerPage" key is also used. If it is used, it returns only the specified (by key's value) page instead of all pages created using the key "itemsPerPage". The value must be an integer

	number greater or equal to 1. The default value is 0, which means that all pages will be returned.
Restrictions / Special Features:	None
Successful Response:	A JSON Object with the results (all solutions provided by a user from all collections). If the query parameter “itemsPerPage” is used, then the results contain the total number of the pages.

The following is an example of the request in cURL:

```
+ curl --request GET '{HOST}/solutions/provider/{uid}/all'
+ curl --request GET '{HOST}/solutions/provider/{uid}/all?sortBy={value}'
+ curl --request GET '{HOST}/solutions/provider/{uid}/all?itemsPerPage={value}'
+ curl --request GET '...?itemsPerPage={value}&page={value}'
+ curl --request GET '...?sortBy={value}&itemsPerPage={value}'
+ curl --request GET '...?sortBy={value}&itemsPerPage={value}&page={value}'
```

TABLE 30 – GET A LIST WITH ALL SOLUTIONS PROVIDED BY A SPECIFIC USER (USING KEYWORD “ALL”) INTERFACE

Title:	Get a list with all solutions provided by a specific user and under a specific collection (using a “collection” value)	
Endpoint:	{HOST}/solutions/provider/{uid}/{collection}	
HTTP Method:	GET	
Description:	This request is similar to the above request. The only difference between the two (2) actions is that this request retrieves all the solutions provided by a specific user and from a single/specific collection (instead of using keyword “all”). The endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. For more details, refer to the above endpoint.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester’s JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	uid	The ID of the user whose offered solutions will be retrieved.
	collection	Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}
Query Parameters:	As in the above request.	
Restrictions / Special Features:	None	
Successful Response:	A JSON Object with the results (all solutions provided by a user from a specific collection). If the query parameter “itemsPerPage” is used, then the results contain the total number of the pages.	
The following is an example of the request in cURL:		
<pre>+ curl --request GET '{HOST}/solutions/provider/{uid}/{collection}' + curl --request GET '{HOST}/solutions/provider/{uid}/{collection}?sortBy={value}' + curl --request GET '{HOST}/solutions/provider/{uid}/{collection}?itemsPerPage={value}' + curl --request GET '...?itemsPerPage={value}&page={value}' + curl --request GET '...?sortBy={value}&itemsPerPage={value}' + curl --request GET '...?sortBy={value}&itemsPerPage={value}&page={value}'</pre>		

TABLE 31 – GET A LIST WITH ALL SOLUTIONS PROVIDED BY A SPECIFIC USER AND UNDER A SPECIFIC COLLECTION (USING A “COLLECTION” VALUE) INTERFACE

Title:	Get solutions' statistics for front-end's homepage
Endpoint:	{HOST}/solutions/statistics/homepage
HTTP Method:	GET
Description:	<p>A request to this endpoint has as a result the retrieval of some statistics on stored (and approved) solutions (and collections). Briefly, the response contains:</p> <ul style="list-style-type: none"> • the total number of solutions, • the number of solutions per collection, and • the top 3 collections with the most solutions, as well as their percentages of the total number.
Body Data:	None
Headers:	None
URL Parameters:	None
Query Parameters:	None
Restrictions / Special Features:	None
Successful Response:	<p>A JSON Object with the solutions' statistics. Example of a response:</p> <pre>{ "_status": "successful", "results": { "all": { "tools": 15, "datasets": 22, "documents": 5, "other": 0, "policies": 20, "tutorials": 4, "webinars": 3}, "sum": 69, "top": [{ "collection": "datasets", "solutions": 22, "percentage": 0.32}, { "collection": "policies", "solutions": 20, "percentage": 0.29}, { "collection": "tools", "solutions": 15, "percentage": 0.22}] } }</pre>
The following is an example of the request in cURL:	
<code>curl --request GET '{HOST}/solutions/statistics/homepage'</code>	

TABLE 32 – GET SOLUTIONS' STATISTICS FOR FRONT-END'S HOMEPAGE INTERFACE

Title:	Get solutions' filtering values for front-end's Discover page
Endpoint:	{HOST}/solutions/statistics/filtering
HTTP Method:	GET
Description:	<p>A request to this endpoint has as a result the retrieval of some filtering values on stored (and approved) solutions that are used in the Discover page of the front-end (filtering sidebar).</p>
Body Data:	None
Headers:	None
URL Parameters:	None
Query Parameters:	None
Restrictions / Special Features:	None
Successful Response:	A JSON Object with the filtering values.
The following is an example of the request in cURL:	
<code>curl --request GET '{HOST}/solutions/statistics/filtering'</code>	

TABLE 33 – GET SOLUTIONS' FILTERING VALUES FOR FRONT-END'S DISCOVER PAGE INTERFACE

Title:	Upload / Create a new solution with random ID	
Endpoint:	{HOST}/solutions/{collection}	
HTTP Method:	POST	
Description:	<p>Through this POST request, the users can upload their solutions. It requires users/providers to specify the collection in which the solution will be stored. Also, the providers should include their JWTs in the headers of the request because the endpoint is available only to authorized (and verified) users. An important note is that all the new solutions uploaded to the Data Marketplace must be approved by an administrator before they can be made available to other users. Moreover, the administrators can upload a solution on behalf of other users, adding the key "x-provider" in the headers of the request. The body of the request must contain the description of the solution as raw data in JSON format. The schema of the solutions' content varies, and it is flexible to be extended. The JSON schema below, presents the required fields of a solution's description.</p> <pre> { "title": "<title of the solution>", "description ": "<description of the solution and its assets>", "type": "<type of the solution (same as its collection value)>", "comments": "<a private field that is shown only when the full schema is retrieved (only by authorized users) - useful for provider's private comments>", "keywords": ["<keyword 1>", ...], "owner ": "<organization / author / etc.>", } </pre> <p>The front-end has appropriate forms that build such solutions.</p>	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json). It should be noted that the solutions can also be uploaded from binary files that contain the above JSON schema (a curl example can be found below).	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
	x-provider	[Optional & only for administrators] The ID of the provider user in case that the solution is uploaded by an administrator and not by the provider.
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}
Query Parameters:	None	
Restrictions / Special Features:	Available to all authorized (and verified) users. The administrators can upload a solution on behalf of other users.	
Successful Response:	JSON Object with the new solution's ID in its content.	
The following is an example of the request in cURL:		
<pre> curl -request POST '{HOST}/solutions/{collection}' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-raw '{ "title": "<title of the solution>", "description": "<description of the solution and its assets>", "type": "<type of the solution (same as its collection value)>", } </pre>		

```

"comments": "<a private field that is shown only when the full schema is retrieved
              (only by authorized users) - useful for private comments>",
"keywords": ["<keyword 1>", ... ], "owner ": "<organization / author / etc.>",
}'

```

Example of uploading a solution through binary data/file:

```

curl -request POST '{HOST}/solutions/{collection}' -header 'x-access-token: <JWT>' \
--header 'Content-Type: application/json' -data-binary '@<path_to_json_file>'

```

TABLE 34 – UPLOAD / CREATE A NEW SOLUTION WITH RANDOM ID INTERFACE

Below are some examples of the stored solutions' schema:

Example 1 – Newly uploaded solution with no assets

```

{
  "id": "others_P8fYOAX67HkK-8fpe1TlB-KuR4-Zsck",
  "info": {"comments": "Private comment.", "contact": "Vasilis Koukos, email",
          "description": "This is an example of description.",
          "keywords": [ "testing", "documentation" ], "owner": "UPRC",
          "title": "Example.", "type": "others"},
  "main_image": "default_image_assets",
  "metadata": {"approved": 1, //0 for pending / 1 for approved
              "last_updated_by": "vkoukos", "md5": "<md5 hash of the solution's data>",
              "provider": "vkoukos", "reviews": {"average_rating": 3.2, "no_reviews": 5},
              "updateDate": "2021-10-11 13:50:48.420Z", "uploadDate": "2021-10-11 13:50:48.420Z",
              "version": 1, //the version of the solution - increases when updating
              "views": 8},
  "assets": {"files": [], //list with the uploaded files for this solution
            "images": [], //list with the uploaded images for this solution
            "videos": [] //list with the uploaded videos for this solution
            },
  "links": [], //list with the external links added to this solution
}

```

Example 2 – Solution with uploaded file

```

{
  "id": "others_P8fYOAX67HkK-8fpe1TlB-KuR4-Zsck",
  ...
  "assets": {
    "files": [{
      "verified": 0, //0 for pending / 1 for approved
      "downloads": 3, //number of downloads of the file
      "filename": "kmeans.py", "id": "80F7MjRTIxvb-7qIKRAjv-IJ3p-b3vL", //file's ID
      "md5": "...", "size": "7.92 KB", "updateDate": "Thu, 14 Oct 2021 13:56:52 GMT",
      "version": 1 //the version of the file - increases when updating
    }],
    "images": [], "videos": []
  },
  "links": []
}

```


Example 3 – Retrieved solution (full schema)

```
{
  "id": "others_P8fYOAX67HkK-8fpelTlB-KuR4-Zsck",
  "info": {"comments": "Private comment.", "contact": "Vasilis Koukos, email",
    "description": "This is an example of description.",
    "keywords": [ "testing", "documentation" ] }, "owner": "UPRC",
    "title": "Example.", "type": "others"},
  "main_image": "default_image_assets",
  "metadata": {"approved": 1, last_updated_by": "vkoukos", "md5": "<md5 hash of solution's data>",
    "provider": "vkoukos", "reviews": {"average_rating": 3.2, "no_reviews": 5},
    "updateDate": "2021-10-11 13:50:48.420Z", "uploadDate": "2021-10-11 13:50:48.420Z",
    "version": 1, "views": 8},
  "assets": {
    "files": [{
      "verified": 0, "downloads": 3, filename: "kmeans.py",
      "id": "80F7MjRTIxb-7qIKRAjv-IJ3p-b3vL", "md5": "...", "size": "7.92 KB",
      "updateDate": "Thu, 14 Oct 2021 13:56:52 GMT", "version": 1
    }], "images": [], "videos": []
  },
  "links": [],
  "reviews": [
    {
      "comment": "Very good!", "solution_version": 1, "rating": 4, "uid": "user_1",
      "review_version": 1, "updated_review_date": "2021-10-14 16:02:05.484Z",
    }, {
      "comment": "Needs improvement...", "solution_version": 1, "rating": 2, "uid": "user_2",
      "review_version": 2, "updated_review_date": "2021-10-15 11:06:03.334Z",
    }, {
      "comment": "Not bad.", "solution_version": 1, "rating": 3, "uid": "user_3",
      "review_version": 1, "updated_review_date": "2021-10-15 13:30:00.209Z",
    }, {
      "comment": "Thank you for this!!!", "solution_version": 1, "rating": 5, "uid": "user_4",
      "review_version": 1, "updated_review_date": "2021-10-18 10:12:49.956Z",
    }, {
      "comment": "Good idea but does not perform well for big data.",
      "solution_version": 1, "rating": 2, "review_version": 1,
      "updated_review_date": "2021-10-18 14:53:13.410Z", "uid": "user_5"
    }
  ]
}
```

Title: Upload / Create a new solution with given ID	
Endpoint:	{HOST}/solutions/{collection}/{given_id}
HTTP Method:	POST
Description:	This endpoint is similar to the above. The only difference is that through the current endpoint, the users are able to specify the ID of the new solution, providing it at the end of the endpoint {given_id}. Currently, this endpoint can be used only by the administrators.
Body Data:	Raw (JSON) Data – as the schema of the previous endpoint (Content-Type: application/json). It should be noted that the solutions can also be uploaded from binary files that contain the JSON schema of the previous endpoint (example in curl can be found at the end of the interface).
Headers:	<u>Key</u>
	<u>Value</u>
	x-access-token Requester's JWT

	x-provider	[Optional & only for administrators] The ID of the provider in case that the solution is uploaded by an administrator and not by the provider.
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}
	given_id	The ID to be given to the new solution.
Query Parameters:	None	
Restrictions / Special Features:	Available only to administrators. The administrators are able to upload a solution on behalf of other users.	
Successful Response:	JSON Object with the new solution's ID in its content.	
The following is an example of the request in cURL:		
<pre>curl -request POST '{HOST}/solutions/{collection}/{given_id}' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-raw '{ "title": "<title of the solution>", "description ": "<description of the solution and its assets>", "type": "<type of the solution (same as its collection value)>", "comments": "<a private field that is shown only when the full schema is retrieved (only by authorized users) - useful for private comments>", "keywords": ["<keyword 1>", ...], "owner ": "<organization / author / etc.>", }'</pre>		
<p>Example of uploading a solution through binary data/file:</p> <pre>curl -request POST '{HOST}/solutions/{collection}/{given_id}' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-binary '@<path_to_json_file>'</pre>		

TABLE 35 – UPLOAD / CREATE A NEW SOLUTION WITH GIVEN ID INTERFACE

Title:	Update a specific solution (using keyword "all")
Endpoint:	{HOST}/solutions/all/{sid}
HTTP Method:	PUT
Description:	<p>With this endpoint, the providers of the solutions can update the solutions' contents. It requires the ID of the solution at the end of the endpoint and the body of the request should contain the next JSON schema as raw data.</p> <pre>{ "title": "<title of the solution>", "description ": "<description of the solution ans its assets>", "type": "<type of the solution (same as its collection value)>", "comments": "<a private field that is shown only when the full schema is retrieved (only by authorized users) - useful for provider's private comments>", "keywords": ["<keyword 1>", ...], "owner ": "<organization / author / etc.>", }</pre> <p>It should be noted that this endpoint uses the keyword "all" (the solutions are already stored in the Data Marketplace, thus the platform knows the collections in which they have been stored). Also, this action is only available to the creators /providers of the solutions and to administrators who can update any solution. Thus, the JWT of a requester should be included in the headers of the request. An important note is that all updated solutions get</p>

	locked and must be approved again by an administrator to be available again to other users.	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json). The solutions can also be updated from binary files that contain the above JSON schema (curl example can be found at the end of the interface).	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution that will be updated.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers/creators of the solutions and for the administrators who can update any solution.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl -request PUT '{HOST}/solutions/all/{sid}' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-raw '{ "title": "<title of the solution>", "description ": "<description of the solution and its assets>", "type": "<type of the solution (same as its collection value)>", "comments": "<a private field that is shown only when the full schema is retrieved (only by authorized users) - useful for private comments>", "keywords": ["<keyword 1>", ...], "owner ": "<organization / author / etc.>", }'</pre>		
<u>Example of uploading a solution through binary data/file:</u>		
<pre>curl -request POST '{HOST}/solutions/all/{sid}' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-binary '@<path to json file>'</pre>		

TABLE 36 – UPDATE A SPECIFIC SOLUTION (USING KEYWORD “ALL”) INTERFACE

Title:	Update a specific solution (using solution's “collection”)	
Endpoint:	{HOST}/solutions/{collection}/{sid}	
HTTP Method:	PUT	
Description:	This PUT request is similar to the previous. The only difference is that instead of using keyword “all” it uses the collection in which the solution has been stored during its creation. The endpoint is restricted and available only to solutions' providers/creators and to administrators who can update any solution. Thus, the JWT of a requester must be included in the headers of the request. More information about the endpoint can be found on the above endpoint.	
Body Data:	Raw (JSON) Data – as the schema of the previous endpoint (Content-Type: application/json). It should be noted that the solutions can also be uploaded from binary files that contain the JSON schema of the previous endpoint (example in curl can be found at the end of the interface).	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {“tools”, “policies”, “datasets”, “webinars”, “tutorials”, “documents”, “other”}
	sid	The ID of the solution that will be updated.

Query Parameters:	None
Restrictions / Special Features:	Available only for the providers/creators of the solutions and for the administrators who can update any solution.
Successful Response:	JSON Object with a successful message.
The following is an example of the request in cURL:	
<pre>curl -request PUT '{HOST}/solutions/{collection}/{sid}' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-raw '{ "title": "<title of the solution>", "description ": "<description of the solution and its assets>", "type": "<type of the solution (same as collection's value)>", "comments": "<a private field that is shown only when the full schema is retrieved (only by authorized users) - useful for private comments>", "keywords": ["<keyword 1>", ...], "owner ": "<organization / author / etc.>", }'</pre>	
Example of uploading a solution through binary data/file:	
<pre>curl -request POST '{HOST}/solutions/{collection}/{sid}' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-binary '@<path_to_json_file>'</pre>	

TABLE 37 – UPDATE A SPECIFIC SOLUTION (USING SOLUTION'S "COLLECTION") INTERFACE

Title:	Delete a specific solution (using keyword "all")	
Endpoint:	{HOST}/solutions/all/{sid}	
HTTP Method:	DELETE	
Description:	<p>A DELETE request to this endpoint has as a result the deletion of a specific solution, using its ID. The endpoint is restricted and available only to solutions' providers/creators and to administrators who can delete any solution. Thus, the JWT of a requester must be included in the headers of the request. It should be noted that this endpoint uses the keyword "all" instead of solution's collection (the solutions are already stored in the Data Marketplace, thus the platform knows the collections in which have been stored). For security reasons, the requesters should provide their password in the body of their request, as raw data (JSON schema):</p> <pre>{ "password": "..." }</pre> <p>If the action is made by an administrator, the field "password" should be the password of the administrator.</p>	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json).	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution that will be deleted.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers/creators of the solutions and for the administrators who can delete any solution.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl -request DELETE '{HOST}/solutions/all/{sid}' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-raw '{ "password": "..." }'</pre>		

TABLE 38 – DELETE A SPECIFIC SOLUTION (USING KEYWORD "ALL") INTERFACE

Title:	Delete a specific solution (using solution's "collection")	
Endpoint:	{HOST}/solutions/{collection}/{sid}	
HTTP Method:	DELETE	
Description:	This request is similar to the previous. The only difference is that instead of using keyword "all" it uses the collection in which the solution has been stored during its creation. The endpoint is restricted and available only to solutions' providers/creators and to administrators who can delete any solution. Thus, the JWT of a requester must be included in the headers of the request. More information about the endpoint can be found on the above endpoint.	
Body Data:	Raw (JSON) Data – as the schema of the previous endpoint (Content-Type: application/json).	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: { "tools", "policies", "datasets", "webinars", "tutorials", "documents", "other" }
	sid	The ID of the solution that will be deleted.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers/creators of the solutions and for the administrators who can delete any solution.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl -request DELETE '{HOST}/solutions/{collection}/{sid}' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-raw '{ "password": "..." }'</pre>		

TABLE 39 – DELETE A SPECIFIC SOLUTION (USING SOLUTION'S "COLLECTION") INTERFACE

Title:	Delete all solutions (administrators' action)	
Endpoint:	{HOST}/solutions/all/all	
HTTP Method:	DELETE	
Description:	This endpoint is available only to the administrators, who can delete all the existing solutions from all the collections (the keyword "all" is used instead of a specific collection). The endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. For security reasons, the requesters should provide their password in the body of their request, as raw data (JSON schema): <pre>{ "password": "..." }</pre>	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json).	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	None	
Query Parameters:	None	
Restrictions / Special Features:	Currently, it is available only to the "superuser" (master admin) of the Data Marketplace.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		

```
curl -request DELETE '{HOST}/solutions/all/all' -header 'x-access-token: <JWT>' \
--header 'Content-Type: application/json' -data-raw '{ "password": "..." }'
```

TABLE 40 – DELETE ALL SOLUTIONS INTERFACE

Title:	Delete all solutions from a specific collection (administrators' action)	
Endpoint:	{HOST}/solutions/{collection}/all	
HTTP Method:	DELETE	
Description:	This endpoint is similar to the above. It is available only to the administrators who can delete all the solutions from a specific collection. The endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. For security reasons, the requesters should provide their password in the body of their request, as raw data (JSON schema): <code>{ "password": "..." }</code>	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json).	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}
Query Parameters:	None	
Restrictions / Special Features:	Currently, it is available only to the "superuser" (master admin) of the Data Marketplace.	
Successful Response:	JSON Object with a successful message.	

The following is an example of the request in cURL:

```
curl -request DELETE '{HOST}/solutions/{collection}/all' -header 'x-access-token: <JWT>' \
--header 'Content-Type: application/json' -data-raw '{ "password": "..." }'
```

TABLE 41 – DELETE ALL SOLUTIONS FROM A SPECIFIC COLLECTION INTERFACE

Title:	Make a review for a solution	
Endpoint:	{HOST}/solutions/review/{sid}	
HTTP Method:	POST	
Description:	This endpoint is used to make a review for a solution, whose ID is included in the request's URL. The endpoint is available to all registered and verified users whose JWT is required in the headers of the request. A review consists of a rating (integer value between 1 and 5) and a comment (free text). The users can make a review for a specific solution only once, but they can update it via the next endpoint, whilst the providers/creators of solution cannot make a review on their solutions. The next JSON schema must be in request's body, as raw data: <code>{"rating": <value>, "comment": "..."}</code> After the successful submission of a review, the average rating of the reviewed solution is recalculated.	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json).	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution for which the review of a user will be made.

Query Parameters:	None
Restrictions / Special Features:	Available to all registered and verified users. The providers/creators are not able to make a review for their solutions.
Successful Response:	JSON Object with a successful message.
The following is an example of the request in cURL:	
<pre>curl -request POST '{HOST}/solutions/review/{sid}' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-raw '{"rating": <value>, "comment": "..."}'</pre>	

TABLE 42 – MAKE A REVIEW FOR A SOLUTION INTERFACE

After the successful submission of a review, the following JSON document is stored in the database:

```
{
  "_id": "<review's ID>", "rating": <integer value between 1 and 5>,
  "comment": "...", "title": "<solution's title>", "sid": "<solution's ID>",
  "collection": "<solution's collection>",
  "uid": "<the ID of the user who made the review>",
  "reviewer": <the fullname of the user who made the review>,
  "initial_review_date": "<the date of the initial review>",
  "updated_review_date": "<date of the last review>",
  "solution_version": <solution's version when the review made>,
  "review_version": <version of the current review>,
  "provider": <solution's provider ID>
}
```

Title:	Update an existing review for a solution	
Endpoint:	{HOST}/solutions/review/{sid}	
HTTP Method:	PUT	
Description:	<p>This endpoint is used in order to update a review that made for a specific solution. The ID of the solution should be included in the URL of the request. The endpoint is available to all registered and verified users whose JWT is required in the headers of the request. A prerequisite for this action is that users have already made a review for the specific solution.</p> <p>A review consists of a rating (integer value between 1 and 5) and a comment. The next JSON schema should be in the body of the request, as raw data:</p> <pre>{"rating": <value>, "comment": "..."}'</pre> <p>After the successful submission of an updated review, the average rating of the reviewed solution is recalculated.</p>	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json).	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution for which the review of a user will be updated.
Query Parameters:	None	
Restrictions / Special Features:	Available to all registered and verified users. A prerequisite for this action is that users have already made a review for the specific solution.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl -request PUT '{HOST}/solutions/review/{sid}' \ --header 'x-access-token: <JWT>' -header 'Content-Type: application/json' \ --data-raw '{"rating": <value>, "comment": "..."}'</pre>		

TABLE 43 – UPDATE AN EXISTING REVIEW FOR A SOLUTION INTERFACE

Title:	Delete a review for a solution	
Endpoint:	{HOST}/solutions/review/{sid}	
HTTP Method:	DELETE	
Description:	<p>This endpoint is used to delete a review that made for a specific solution. The ID of the solution should be included in the URL of the request. The endpoint is available to all registered and verified users whose JWT is required in the headers of the request. A prerequisite for this action is that users have already made a review for the specific solution. It should be noted that the administrators are able to delete reviews that made from other users, providing the ID of the reviewer in the headers of the request. After the successful deletion of a review, the average rating of the solution is recalculated.</p>	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
	x-uid	[Optional & only for administrators] The ID of the user whose review on the specified solution will be deleted. It is used by administrators in order to specify the reviewer.
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution for which the review of a user will be deleted.
Query Parameters:	None	
Restrictions / Special Features:	Available to all registered and verified users. A prerequisite for this action is that users have already made a review for the specific solution. The administrators are able to delete reviews that made from other users.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>+ curl -request DELETE '{HOST}/solutions/review/{sid}' --header 'x-access-token: <JWT>' + curl -request DELETE '{HOST}/solutions/review/{sid}' \ --header 'x-access-token: <JWT>' -header 'x-uid: <value>'</pre>		

TABLE 44 – DELETE A REVIEW FOR A SOLUTION INTERFACE

Title:	Get a list with the reviews made by a specific user	
Endpoint:	{HOST}/solutions/review/{uid}	
HTTP Method:	GET	
Description:	<p>This request returns all the reviews made by a specific user whose ID {uid} is part of the request's URL (the schema of the reviews can be found in the "Make a review for a solution" interface). The endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. Finally, the endpoint offers some standard query parameters that specify the format of the results and are described below (Query Parameters).</p>	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	uid	The ID of the user whose reviews will be retrieved.
Query Parameters:	<u>Key</u>	<u>Value</u>

	sortBy	[Optional] Sorts the reviews by a field – the <u>default</u> is the “newest” key. The value should be one of the following: “newest”: sort by review date in descending order. “oldest”: sort by review date in ascending order. “rating-asc”: sort by user’s rating in ascending order. “rating-desc”: sort by user’s rating in descending order. “title”: sort by solution’s title in ascending order.
	itemsPerPage	[Optional] Returns the results separated in pages (arrays) of N items. The number N is specified by the value of this key. The value N must be an integer number greater or equal to 1. If the key is not used or has a non-accepted value, the results are returned on a single page.
	Page	[Optional] This key can only be used if the “itemsPerPage” key is also used. If it is used, it returns only the specified (by key’s value) page instead of all pages created using the key “itemsPerPage”. The value must be an integer number greater or equal to 1. The default value is 0, which means that all pages will be returned.
Restrictions / Special Features:	Available to all registered and verified users.	
Successful Response:	JSON Object with the reviews made by a specific user.	
The following is an example of the request in cURL:		
<pre>+ curl -request GET '{HOST}/solutions/review/{uid}' -header 'x-access-token: <JWT>' + curl -request GET '...?sortBy={value}' ... + curl -request GET '...?itemsPerPage={value}' ... + curl -request GET '...?itemsPerPage={value}&page={value}' ... + curl -request GET '...?sortBy={value}&itemsPerPage={value}' ... + curl -request GET '...?sortBy={value}&itemsPerPage={value}&page={value}' ...</pre>		

TABLE 45 – GET A LIST WITH THE REVIEWS MADE BY A SPECIFIC USER INTERFACE

Title:	Get a list with the reviews made for a specific solution	
Endpoint:	{HOST}/solutions/reviews/{sid}	
HTTP Method:	GET	
Description:	This request returns all the reviews made for a specific solution whose ID {sid} is part of the request’s URL (the schema of the reviews can be found in the “Make a review for a solution” interface). The endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. Finally, the endpoint offers some standard query parameters that specify the format of the results and are described below (Query Parameters).	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester’s JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution whose reviews will be retrieved.
Query Parameters:	<u>Key</u>	<u>Value</u>
	sortBy	[Optional] Sorts the reviews by a field – the <u>default</u> is the “newest” key. The value should be one of the following:

		<p>“newest”: sort by review date in descending order.</p> <p>“oldest”: sort by review date in ascending order.</p> <p>“rating-asc”: sort by user’s rating in ascending order.</p> <p>“rating-desc”: sort by user’s rating in descending order.</p> <p>“title”: sort by solution’s title in ascending order.</p>
	itemsPerPage	[Optional] Returns the results separated in pages (arrays) of N items. The number N is specified by the value of this key. The value N must be an integer number greater or equal to 1. If the key is not used or has a non-accepted value, the results are returned on a single page.
	Page	[Optional] This key can only be used if the “itemsPerPage” key is also used. If it is used, it returns only the specified (by key’s value) page instead of all pages created using the key “itemsPerPage”. The value must be an integer number greater or equal to 1. The default value is 0, which means that all pages will be returned.
Restrictions / Special Features:	Available to all registered and verified users.	
Successful Response:	JSON Object with the reviews made by a specific user.	
The following is an example of the request in cURL:		
<pre>+ curl -request GET '{HOST}/solutions/reviews/{sid}' -header 'x-access-token: <JWT>' + curl -request GET '...?sortBy={value}' ... + curl -request GET '...?itemsPerPage={value}' ... + curl -request GET '...?itemsPerPage={value}&page={value}' ... + curl -request GET '...?sortBy={value}&itemsPerPage={value}' ... + curl -request GET '...?sortBy={value}&itemsPerPage={value}&page={value}' ...</pre>		

TABLE 46 – GET A LIST WITH THE REVIEWS MADE FOR A SPECIFIC SOLUTION INTERFACE

Title:	Get the cover image of a solution	
Endpoint:	{HOST}/solutions/image/{sid}	
HTTP Method:	GET	
Description:	This interface can be used in order to retrieve the cover image of a specific solution, using its ID {sid}, which is part of the request’s URL.	
Body Data:	None	
Headers:	None	
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution whose cover image will be retrieved.
Query Parameters:	None	
Restrictions / Special Features:	None	
Successful Response:	The cover image of the solution (returned as image).	
The following is an example of the request in cURL:		
<pre>curl --request GET '{HOST}/solutions/image/{sid}'</pre>		

TABLE 47 – GET THE COVER IMAGE OF A SOLUTION INTERFACE

Title:	Change the cover image of a solution	
Endpoint:	{HOST}/solutions/image/{sid}/{img_id}	
HTTP Method:	PUT	
Description:	This interface can be used in order to change the cover image of a specific solution, using the ID of the solution {sid}, which is part of the request's URL and the ID of the image {img_id} that will replace the current cover image of the specified solution. It should be noted that the image should be uploaded as 'image' asset type for the specified solution in order to be used as the cover image, whose ID is the {img_id} value that is part of the request's URL.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution whose cover image will be retrieved.
	img_id	The ID of the image that will replace the current cover image – the image must be an asset of the specified solution.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers/creators of the solutions and for the administrators who can update any solution.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<code>curl --request PUT '{HOST}/solutions/image/{sid}/{img_id}' --header 'x-access-token: {JWT}'</code>		

TABLE 48 – CHANGE THE COVER IMAGE OF A SOLUTION INTERFACE

Title:	Remove the cover image of a solution	
Endpoint:	{HOST}/solutions/image/{sid}	
HTTP Method:	DELETE	
Description:	This interface can be used to remove the cover image of a specific solution, using the ID of the solution {sid} that is part of the request's URL. This action removes the current cover image which is replaced by the default cover image of all solutions. Though, this action will not delete the image from the assets lists.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution whose cover image will be removed.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers/creators of the solutions and for the administrators who can update any solution.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<code>curl --request DELETE '{HOST}/solutions/image/{sid}' --header 'x-access-token: {JWT}'</code>		

TABLE 49 – REMOVE THE COVER IMAGE OF A SOLUTION INTERFACE

Title:	Get a list with all solutions that need permission (administrators' action)	
Endpoint:	{HOST}/solutions/permit/all	
HTTP Method:	GET	
Description:	This endpoint returns the solutions from all the collections (since the keyword "all" is used) that need permission before they become available to the Data Marketplace's users. A solution needs permission either when it is uploaded or after it has been updated by the users. Moreover, the endpoint is only available to administrators and thus, the JWT of a requester is required in the headers of the request. Finally, the endpoint offers some standard query parameters that specify the format of the results and are described below (Query Parameters).	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	None	
Query Parameters:	<u>Key</u>	<u>Value</u>
	sortBy	[Optional] Sorts the solutions by a field – the <u>default</u> is the "newest" key. The value should be one of the following: "newest": sort by date in descending order. "oldest": sort by date in ascending order. "title": sort by title in ascending order.
	itemsPerPage	[Optional] Returns the results separated in pages (arrays) of N items. The number N is specified by the value of this key. The value N must be an integer number greater or equal to 1. If the key is not used or has a non-accepted value, the results are returned on a single page.
	Page	[Optional] This key can only be used if the "itemsPerPage" key is also used. If it is used, it returns the specified (by key's value) page instead of all pages created using the key "itemsPerPage". The value must be an integer greater or equal to 1. The default value is 0, meaning that all pages will be returned.
Restrictions / Special Features:	Available only to the administrators.	
Successful Response:	JSON Object with the solutions (from all collections) that need permission in its content.	
The following is an example of the request in cURL:		
<pre>+ curl -request GET '{HOST}/solutions/permit/all' -header 'x-access-token: <JWT>' + curl -request GET '...?sortBy={value}' ... + curl -request GET '...?itemsPerPage={value}' ... + curl -request GET '...?itemsPerPage={value}&page={value}' ... + curl -request GET '...?sortBy={value}&itemsPerPage={value}' ... + curl -request GET '...?sortBy={value}&itemsPerPage={value}&page={value}' ...</pre>		

TABLE 50 – GET A LIST WITH ALL SOLUTIONS THAT NEED PERMISSION INTERFACE

Title:	Get a list with all solutions from a specific collection that need permission (administrators' action)
---------------	--

Endpoint:	{HOST}/solutions/permit/{collection}	
HTTP Method:	GET	
Description:	This request is similar to the above request. The only difference between the two (2) actions is that the current request retrieves the solutions that need permission from a specific collection (uses specific {collection} value instead of the keyword "all"). For more details, refer to the above endpoint.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}
Query Parameters:	As in the above request.	
Restrictions / Special Features:	Available only to the administrators.	
Successful Response:	JSON Object with the solutions (from a specific collection) that need permission in its content.	
The following is an example of the request in cURL:		
<pre>+ curl -request GET '{HOST}/solutions/permit/{collection}' -header 'x-access-token: <JWT>' + curl -request GET '...?sortBy={value}' ... + curl -request GET '...?itemsPerPage={value}' ... + curl -request GET '...?itemsPerPage={value}&page={value}' ... + curl -request GET '...?sortBy={value}&itemsPerPage={value}' ... + curl -request GET '...?sortBy={value}&itemsPerPage={value}&page={value}' ...</pre>		

TABLE 51 – GET A LIST WITH ALL SOLUTIONS FROM A SPECIFIC COLLECTION THAT NEED PERMISSION INTERFACE

Title:	Approve or reject a specific solution that need permission, using keyword "all" (administrators' action)	
Endpoint:	{HOST}/solutions/permit/all/{sid}	
HTTP Method:	POST	
Description:	This endpoint is used by administrators to approve or reject a specific solution (using its ID) that needs administrators' permission. The endpoint is restricted and available only to administrators and thus, the requesters must provide their JWTs in the headers of the request. Also, this endpoint uses the keyword "all" and not the collection in which a specific solution is stored, as the next endpoint does. An important parameter/key that must be included in the request's headers is the "x-permission" key that should contain the text "approve" for the solution's approval, otherwise the text "disapprove" for its rejection. A rejection/ disapproval of a solution results to the deletion of the solution and all its assets.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
	x-permission	Valid values: {"approve", "disapprove"}
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution that will be approved or rejected.

Query Parameters:	None
Restrictions / Special Features:	Available only to the administrators.
Successful Response:	JSON Object with a successful message.
The following is an example of the request in cURL:	
<pre>curl -request POST '{HOST}/solutions/permit/all/{sid}' \ --header 'x-access-token: <JWT>' -header 'x-permission: <value>'</pre>	

TABLE 52 – APPROVE OR REJECT A SOLUTION THAT NEEDS PERMISSION, USING KEYWORD “ALL” INTERFACE

Title:	Approve or reject a specific solution that need permission, using solution’s “collection” (administrators’ action)	
Endpoint:	{HOST}/solutions/permit/{collection}/{sid}	
HTTP Method:	POST	
Description:	This request is similar to the above request. The only difference between the two (2) actions is that the current request uses the value of the {collection} in which a specific solution is stored. For more details, refer to the above endpoint.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester’s JWT
	x-permission	Valid values: {"approve", "disapprove"}
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {"tools", "policies", "datasets", "webinars", "tutorials", "documents", "other"}
	sid	The ID of the solution that will be approved or rejected.
Query Parameters:	None	
Restrictions / Special Features:	Available only to the administrators.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl -request POST '{HOST}/solutions/permit/{collection}/{sid}' \ --header 'x-access-token: <JWT>' -header 'x-permission: <value>'</pre>		

TABLE 53 – APPROVE OR REJECT A SOLUTION THATS NEEDS PERMISSION, USING SOLUTION’S “COLLECTION” INTERFACE

Title:	Approve or reject all solutions that need permission, using keyword “all” (administrators’ action)
Endpoint:	{HOST}/solutions/permit/all/all
HTTP Method:	POST
Description:	This endpoint is used by the administrators to approve or reject all the stored solutions (from all the collections, since keyword “all” is used) that need administrators’ permission. The endpoint is restricted and available only to administrators and thus, the requesters’ must provide their JWTs in the headers of the request. An important parameter/key that must be included in the headers of the request is the “x-permission” key that should have as a value the text “approve” for the solutions to be approved,

	otherwise the text “disapprove” to be rejected. A rejection/disapproval of the solutions has as a result the deletion of the solutions and all of their assets.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester’s JWT
	x-permission	Valid values: {“approve”, “disapprove”}
URL Parameters:	None	
Query Parameters:	None	
Restrictions / Special Features:	Currently, it is available only to the “superuser” (master admin) of the Data Marketplace.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl -request POST \{HOST}/solutions/permit/all/all ` \ --header 'x-access-token: <JWT>' -header 'x-permission: <value>'</pre>		

TABLE 54 – APPROVE OR REJECT ALL SOLUTIONS THAT NEED PERMISSION, USING KEYWORD “ALL” INTERFACE

Title:	Approve or reject all solutions that need permission under a specific collection, using a “collection” value (administrators’ action)	
Endpoint:	{HOST}/solutions/permit/{collection}/all	
HTTP Method:	POST	
Description:	This request is similar to the above request. The only difference is that the administrators, using the current endpoint, are able to approve or reject all the solutions of a specific {collection}. The endpoint is restricted and available only to administrators and thus, the requesters’ must provide their JWTs in the headers of the request. An important parameter/key that must be included in the headers of the request is the “x-permission” key that should have as a value the text “approve” for the solutions to be approved, otherwise the text “disapprove” to be rejected. A rejection/disapproval of the solutions has as a result the deletion of the solutions and all of their assets.	
Body Data:	None	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester’s JWT
	x-permission	Valid values: {“approve”, “disapprove”}
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	collection	Valid values: {“tools”, “policies”, “datasets”, “webinars”, “tutorials”, “documents”, “other”}
Query Parameters:	None	
Restrictions / Special Features:	Currently it is available only to the “superuser” (master admin) of the Data Marketplace	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl -request POST \{HOST}/solutions/permit/{collections}/all ` \ --header 'x-access-token: <JWT>' -header 'x-permission: <value>'</pre>		

TABLE 55 – APPROVE OR REJECT ALL SOLUTIONS THAT NEED PERMISSION UNDER A SPECIFIC COLLECTION, USING A “COLLECTION” VALUE INTERFACE

2.2.1.3 Search functionality on Solutions

The search functionality is a vital requirement for most services in order to reduce the number of objects returned by a query. Thus, the back-end's endpoints that retrieve multiple solutions simultaneously, support some relative query filters. These filters enable the users of the Data Marketplace to search for assets, based on various parameters from the content of the stored solutions.

More specifically, the interfaces of the back-end that return lists of assets, support additional query parameters with any key-value pair. Query parameters are a defined set of parameters attached to the end of a URL and are used in order to help search specific content or actions based on the data being passed. In order to append query parameters to the end of a URL, a question mark "?" is added to the end of the URL, followed immediately by a pair of a key and a value, separated by an equal symbol "=". Moreover, a URL can have multiple parameters, by adding an ampersand symbol "&" between each pair of key-value.

In the context of the Data Marketplace and the solutions, the keys added to the URLs as query parameters must be valid, in the sense that they exist as fields in the solutions and their search has a real value. Below are some valid syntaxes for advanced search with additional query parameters. The examples use the "Get a list with all solutions" interface.

```
Single key: `{HOST}/solutions/all?<key_name>=<value>'
```

```
Multiple keys: `{HOST}/solutions/all?<key_1>=<value>&<key_2>=<value>&...'
```

Moreover, the Python programming language that is used by the back-end (as described in Section 2.3.1), enables access to nested fields of dictionary/JSON object using a dot "." between a key at the first level of the hierarchy and a key at the second level (this applies to all levels, up to the lowest level). Thus, the next example is also a valid schema of a query:

```
For keys in lower hierarchical level:
```

```
`{HOST}/solutions/all?<key_level_1>.<key_level_2>.<...>.<key_level_n>=<value>'
```

To sum up, given the above syntaxes of a valid query and the JSON Object/solution of "Example 1" in the interface of "Table 34 – Upload / Create a new solution with random ID Interface", the following search example request in cURL returns the solutions that in their title contain the value "machine learning" and their provider is the user with ID "vkoukos":

```
curl -request GET  
{HOST}/solutions/all?info.title=machine%20learning&metadata.provider=vkoukos'
```

It should be noted that the value "%20" is the ASCII Encoding Reference of the space character.

Except for these, the back-end supports advanced searching using some operators that extend the keys of the query parameters, using a dot "." between the keys and the operators. Below are the supported operators along with a description for their usage.

Operator	Usage	Example
<i>eq</i>	Full title: equal This operator performs an equality search and has exactly the same use with the equality symbol “=”. It applies to both texts (strings) and numbers.	<key>.eq=<value>
<i>ne</i>	Full title: not equal This operator performs a non-equality search. It applies to both texts (strings) and numbers.	<key>.ne=<value>
<i>gt</i>	Full title: greater than This operator performs searching for a key with a value greater than the provided. It applies to both texts (strings) and numbers.	<key>.gt=<value>
<i>gte</i>	Full title: greater than or equal This operator performs searching for a key with a value greater than or equal to the provided. It applies to both texts (strings) and numbers.	<key>.gte=<value>
<i>lt</i>	Full title: less than This operator performs searching for a key with a value less than to the provided. It applies to both texts (strings) and numbers.	<key>.lt=<value>
<i>lte</i>	Full title: less than or equal This operator performs searching for a key with a value less than or equal to the provided. It applies to both texts (strings) and numbers.	<key>.lte=<value>
<i>in</i>	Full title: in (equal to one of the values) This operator performs searching for a key with a value equal to one of the provided values. The <value> may have multiple values separated by a comma “,”. It applies to both texts (strings) and numbers.	<key>.in=<value_1>, <value_2>
<i>nin</i>	Full title: not in (not equal to any of the value) This operator performs searching for a key with a value not equal to any of the provided values. The <value> may have multiple values separated by a comma “,”. It applies to both texts (strings) and numbers.	<key>.nin=<value_1>, <value_2>

TABLE 56 – BACK-END’S SEARCH OPERATORS

Below are some examples of the operators’ use.

```
eg: '{HOST}/solutions/all?metadata.provider.eq=vkoukos'
ne: '{HOST}/solutions/all?metadata.version.ne=1'
gt: '{HOST}/solutions/all?metadata.views.gt=100'
gte: '{HOST}/solutions/all?info.type.gte=datasets'
lt: '{HOST}/solutions/all?metadata.uploadDate.lt=2021-10-15'
lte: '{HOST}/solutions/all?metadata.reviews.no_reviews.lte=20'
in: '{HOST}/solutions/all?info.title.in=machine,learning,algorithm'
nin: '{HOST}/solutions/all?info.keywords.nin=poverty,crime'
```

Furthermore, the back-end’s search mechanism uses a ranking system for the results. More specifically, for each solution in the results, it maintains a score resulting from the points it receives for each search argument.

In an equality search (using “=” symbol or “eq” operator) for a specific key, the points that a solution receives can be one of the following:

- **5:** if the values are exactly equal (same) and case sensitive.
- **4:** if the values are equal (same) but not case sensitive.
- **3:** if the values are similar (e.g., the first value contains the second value but are not the same) and case sensitive.
- **2:** if the values are similar but not case sensitive.
- **0:** if the values do not match.

The other operators just receive 1 if the conditions match (“true”). The operator “in” uses the operator “eq” (or the symbol “=”) for each value in its “array” and thus, it has the same score system.

Finally, the operator “nin” uses the operator “ne” for each value in its “array”.

2.2.1.4 Interfaces related to Assets

This group of APIs offers functionalities intended for the management of the assets. They support all CRUD operations for the assets that are stored in the back-end. Table 57 presents the endpoints related to Assets as they are in the first version of the Data Marketplace’s back-end.

Action	HTTP Method	Endpoint
Get a list with the stored assets	GET	{HOST}/assets
Get a specific asset, using its ID	GET	{HOST}/assets/{asset_id}
Upload a new asset with random ID, linked to a specific solution	POST	{HOST}/assets/{sid}
Upload a new asset with given ID, linked to a specific solution	POST	{HOST}/assets/{sid}/{given_asset_id}
Update a specific asset, using its ID	PUT	{HOST}/assets/{asset_id}
Delete a specific asset, using its ID	DELETE	{HOST}/assets/{asset_id}
Delete all assets (administrators’ action)	DELETE	{HOST}/assets/all

TABLE 57 – BACK-END’S INTERFACES RELATED TO ASSETS

- *{HOST}* refers to the hosting server: the domain name and the port running the back-end.
- *{asset_id}* refers to the ID of a specific asset.
- *{given_asset_id}* is used in the upload asset action, providing the new asset’s ID.
- *{sid}* refers to the ID of the solution with which the new asset will be linked to.
- Most of these actions require additional fields in the headers of the HTTP request. Example of a required field is the JWT.

Below is a more detailed description of all the provided interfaces and their corresponding actions:

Title:		Get a list with the stored assets	
Endpoint:	{HOST}/assets		
HTTP Method:	GET		
Description:	A request to this endpoint will result in the retrieval of a list with the stored assets and some additional information of them.		
Body Data:	None		
Headers:	<u>Key</u>	<u>Value</u>	
	x-access-token	Requester's JWT	
URL Parameters:	None		
Query Parameters:	None		
Restrictions / Special Features:	Available only to administrators.		
Successful Response:	Results in JSON Object		
The following is an example of the request in cURL:			
<code>curl -request GET '{HOST}/assets' -header 'x-access-token: <JWT>'</code>			

TABLE 58 – GET A LIST WITH THE STORED ASSETS INTERFACE

Title:		Get a specific asset, using its ID	
Endpoint:	{HOST}/assets/{asset_id}		
HTTP Method:	GET		
Description:	This endpoint is used to retrieve a specific stored asset. For its retrieval, the usage of the asset's ID is necessary. Also, this endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request.		
Body Data:	None		
Headers:	<u>Key</u>	<u>Value</u>	
	x-access-token	Requester's JWT	
URL Parameters:	<u>Parameter</u>	<u>Value</u>	
	asset_id	The ID of the asset that will be retrieved.	
Query Parameters:	None		
Restrictions / Special Features:	Available to all authorized (and verified) users.		
Successful Response:	Binary data		
The following is an example of the request in cURL:			
<code>curl -request GET '{HOST}/assets/{asset_id}' -header 'x-access-token: <JWT>'</code>			

TABLE 59 – GET A SPECIFIC ASSET USING ITS ID INTERFACE

Title:		Upload a new asset with random ID, linked to a specific solution	
Endpoint:	{HOST}/assets/{sid}		
HTTP Method:	POST		
Description:	Via this endpoint the users can upload their assets. It requires to add at the end of the endpoint the solution's ID with which is going to be linked. It is also needed to add to the headers of the request the JWT of the provider and the asset's filename., whilst the assets must be uploaded as form-data with the key "asset".		

Body Data:	<u>Data Type:</u>	Form Data
	<u>Key</u>	<u>Value</u>
	asset	Binary data / Path to file
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
	x-filename	New asset's filename.
	x-provider	[Optional & only for administrators] The ID of the provider user in case that the asset is uploaded by an administrator and not by the provider.
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution with which the new asset is going to be linked.
Query Parameters:	None	
Restrictions / Special Features:	Available for the solutions' providers with which the assets will be connected, and also for the administrators who can upload assets on behalf of the providers.	
Successful Response:	JSON Object with the new asset's ID in its content.	
The following is an example of the request in cURL:		
<pre>curl -request POST '{HOST}/assets/{sid}' \ --header 'x-access-token: <JWT>' -header 'x-filename: <value>' \ --form 'asset=@"<full_path_to_asset>"'</pre>		

TABLE 60 – UPLOAD A NEW ASSET WITH RANDOM ID INTERFACE

Title:	Upload a new asset with given ID, linked to a specific solution	
Endpoint:	{HOST}/assets/{sid}/{given_asset_id}	
HTTP Method:	POST	
Description:	This endpoint is similar to the previous. The difference is that with the current endpoint it is possible to specify the ID of the new asset, providing it at the end of the endpoint {given_asset_id}.	
Body Data:	<u>Data Type:</u>	Form Data
	<u>Key</u>	<u>Value</u>
	asset	Binary data / Path to file
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
	x-filename	New asset's filename.
	x-provider	[Optional & only for administrators] The ID of the provider user in case that the asset is uploaded by an administrator and not by the provider.
URL Parameters:	<u>Parameter</u>	<u>Value</u>
	sid	The ID of the solution with which the new asset is going to be linked.
	Given_asset_id	The ID to be given to the new asset.
Query Parameters:	None	
Restrictions / Special Features:	Available only for administrators whether they upload an asset for their solutions or upload an asset on behalf of the providers.	
Successful Response:	JSON Object with the new asset's ID in its content.	

The following is an example of the request in cURL:

```
curl -request POST '{HOST}/assets/{sid}/{given_asset_id}' \
--header 'x-access-token: <JWT>' -header 'x-filename: <value>' \
--form 'asset=@'<full_path_to_asset>'"
```

TABLE 61 – UPLOAD A NEW ASSET WITH GIVEN ID INTERFACE

Title:		Update a specific asset, using its ID
Endpoint:	{HOST}/assets/{asset_id}	
HTTP Method:	PUT	
Description:	With this PUT request, it is possible to update an already stored asset. The asset's ID that should be at the end of the endpoint, determines which asset should be replaced by the new asset. As in the uploading, the asset should be uploaded as form-data with the key "asset" and the headers of the request should contain provider's JWT. It should be noted that the users can only update the assets provided by themselves (except for administrators).	
Body Data:	<u>Data Type:</u>	Form Data
	<u>Key</u>	<u>Value</u>
Headers:	asset	Binary data / Path to file
	<u>Key</u>	<u>Value</u>
URL Parameters:	x-access-token	Requester's JWT
	x-filename	[Optional] Asset's new filename.
Query Parameters:	<u>Parameter</u>	<u>Value</u>
	asset_id	The ID of the asset that will be updated.
Restrictions / Special Features:	Available only for the providers of the solutions / assets and for the administrators who can update stored assets on behalf of the providers.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl -request PUT '{HOST}/assets/{asset_id}' \ --header 'x-access-token: <JWT>' -header 'x-filename: <value>' \ --form 'asset=@'<full_path_to_asset>'"</pre>		

TABLE 62 – UPDATE A SPECIFIC ASSET USING ITS ID INTERFACE

Title:		Delete a specific asset, using its ID
Endpoint:	{HOST}/assets/{asset_id}	
HTTP Method:	DELETE	
Description:	A request to this endpoint has as a result the deletion of a specific asset, by using its ID to find it. This endpoint is restricted and thus, the JWT of a requester must be included in the headers of the request. Note that an asset can be deleted only by its provider and the administrators. For security reasons, the requesters should provide their password in the body of their request, as raw data (JSON schema): <pre>{ "password": "..." }</pre> If the action is made by an administrator, the field "password" should be the password of the administrator.	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json).	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT

URL Parameters:	<u>Parameter</u>	<u>Value</u>
	asset_id	The ID of the asset that will be deleted.
Query Parameters:	None	
Restrictions / Special Features:	Available only for the providers of the solutions / assets and for the administrators who can delete any stored assets.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl -request DELETE '{HOST}/assets/{asset_id}' -header 'x-access-token: <JWT>' \ --header 'Content-Type: application/json' -data-raw '{ "password": "..."}'</pre>		

TABLE 63 – DELETE A SPECIFIC ASSET USING ITS ID INTERFACE

Title:	Delete all assets (administrators' action)	
Endpoint:	{HOST}/assets/all	
HTTP Method:	DELETE	
Description:	<p>This request is similar to the above request, with the difference that it deletes all the assets, as it uses the keyword "all". Again, it is necessary the usage of the requester's JWT, and it is only available to administrators. For security reasons, the requesters should provide their password in the body of their request, as raw data (JSON schema):</p> <pre>{ "password": "..."} </pre>	
Body Data:	Raw (JSON) Data – as the above schema (Content-Type: application/json).	
Headers:	<u>Key</u>	<u>Value</u>
	x-access-token	Requester's JWT
URL Parameters:	None	
Query Parameters:	None	
Restrictions / Special Features:	Currently, it is available only to the "superuser" (master admin) of the Data Marketplace.	
Successful Response:	JSON Object with a successful message.	
The following is an example of the request in cURL:		
<pre>curl --request DELETE '{HOST}/assets/all' --header 'x-access-token: <JWT>' \ --header 'Content-Type: application/json' --data-raw '{ "password": "..."}'</pre>		

TABLE 64 – DELETE ALL ASSETS (ADMINISTRATORS' ACTION) INTERFACE

2.2.1.5 Root & Other Interfaces

One endpoint that was not mentioned is that of the back-end’s root interface, which presents a roadmap of the main back-end’s interfaces. The latter is described below:

Title:	Root interface
Endpoint:	{HOST}
HTTP Method:	GET
Description:	This endpoint returns a list with the back-end’s interfaces that are available to be used by all users. It acts as a roadmap, providing the interfaces along with short information about the functionalities that they trigger. The structure of the information follows a tree approach.
Body Data:	None
Headers:	None
URL Parameters:	None
Query Parameters:	None
Restrictions / Special Features:	None
Successful Response:	Back-end’s roadmap in text/plain.
The following is an example of the request in cURL:	
<code>curl -request GET '{HOST}'</code>	

TABLE 65 – ROOT INTERFACE

Except for the already described interfaces, the back-end provides the following restricted interfaces that will not be described, since they are mostly related only with the platform’s administrators. In short, the interfaces’ titles are:

- Get a list of the administrators,
- Add a new administrator,
- Remove an administrator,
- Get system’s backup,
- Restore a backup,
- Get a list of all users,
- Get a list with system metrics / report,
- Get the manual of the front-end.

2.2.2 Front-end

In this Section, the most important pages of the Data Marketplace’s front-end are presented and analyzed in detail. For each page, there is a relevant description of the displayed content as well as related images (screenshots) from their final view. At this point, it should be noted that through the front-end and back-end, a detailed manual about the front-end pages and the actions that can be performed on them is offered to the users who can refer to it to resolve any issues (the manual is offered through the website of the Data Marketplace – link: <https://mdb.policycloud.eu/manual>). Finally, all the pages of the front-end that exploit the multiple benefits of WordPress, have a responsive web design for all devices as they can adjust accordingly to the users’ screens.

Navigation bar: The Navigation bar is common to all the pages, being located at the top of each page and contains the appropriate items that redirect to the core pages of the front-end. Depending on whether a user is logged in or not, the users can see different items/pages on the bar (also called “menu”). Figures 3 and 4 illustrate the navigation bar that is displayed to non-logged in and logged in users, respectively.



FIGURE 3 – NAVIGATION BAR FOR NON-LOGGED IN USERS

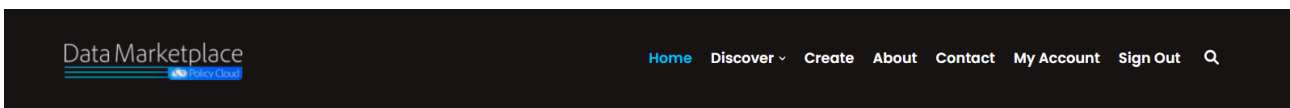


FIGURE 4 – NAVIGATION BAR FOR LOGGED IN USERS

The “Discover” item in the navigation bar is a drop-down list that contains a variety of sub-items that are displayed on mouse over, and redirect the user to the Discover page, applying filtering based on the type of the Data Marketplace’s offered solutions. Figure 5 presents the navigation bar along with the Discover’s drop-down list from the Data Marketplace’s Home page.

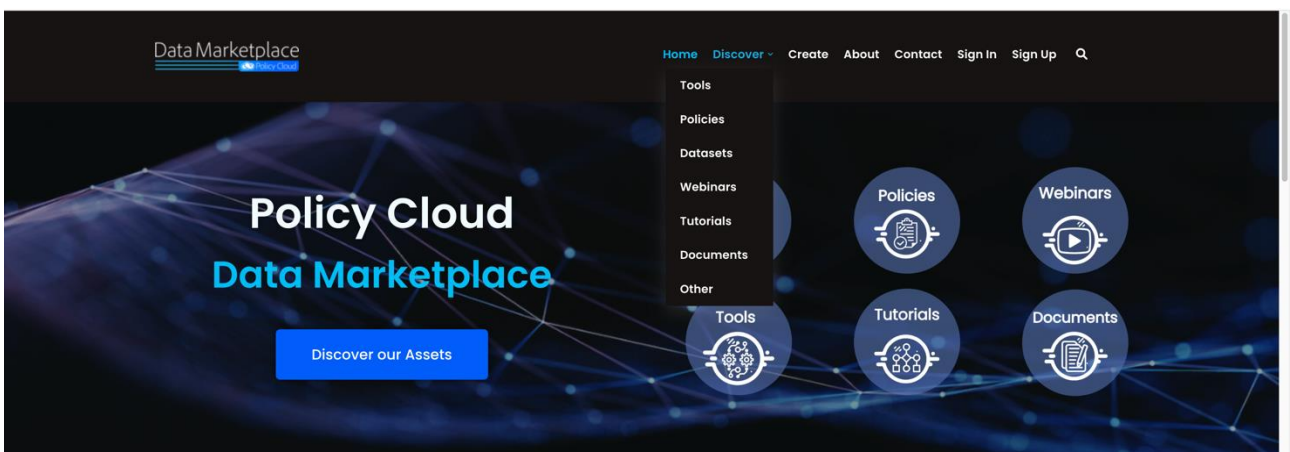


FIGURE 5 – NAVIGATION BAR FROM THE HOME PAGE

In the following example, a user searches for solutions categorized as “Tools”. By hovering the mouse on the “Discover” drop-down list, the main solutions’ types are displayed and with a click on the “Tools” item, the user is redirected to the Discover page that presents the stored solutions categorized as “Tools”, applying the respective filtering (Figure 6).

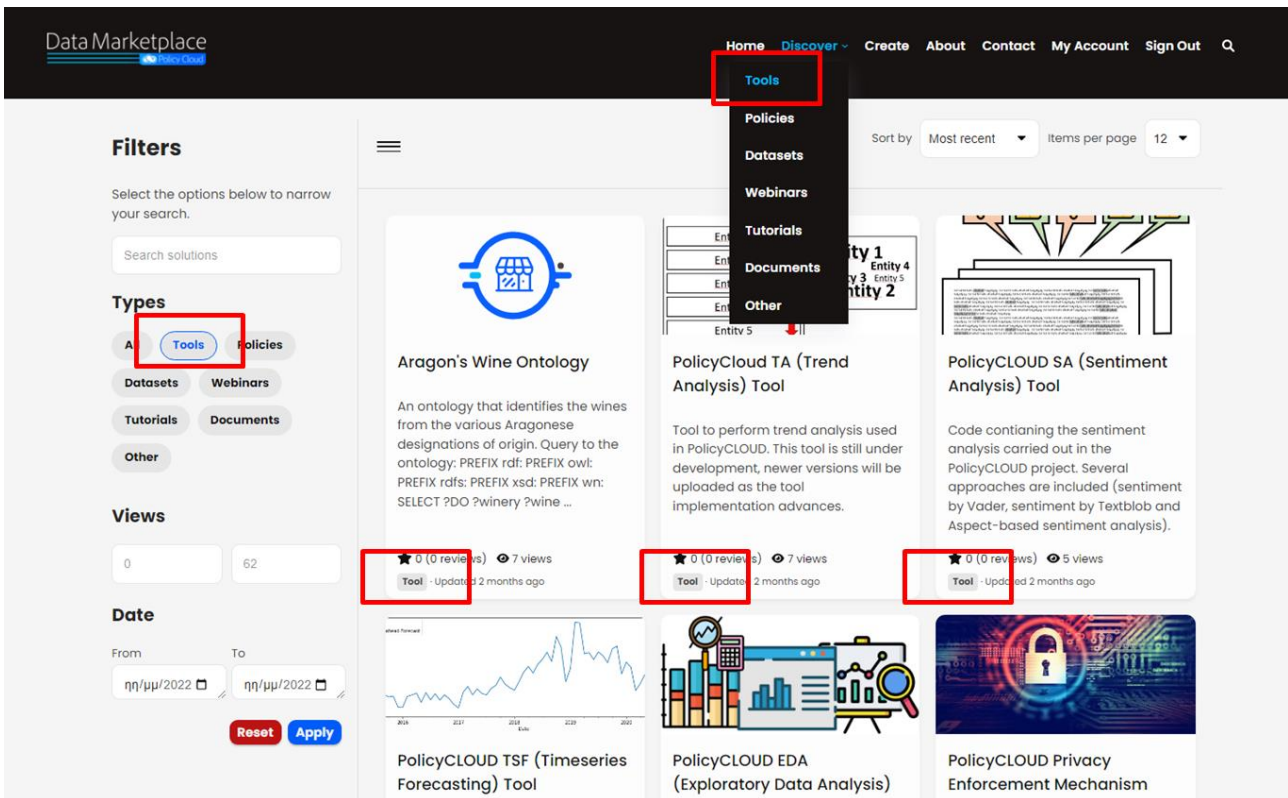


FIGURE 6 – DISCOVER'S SUB-ITEMS REDIRECT TO DISCOVER PAGE

Footer: Except for the navigation bar, another common section to all the pages is the footer, which is located at the bottom of each page and contains information about the project's main site, social media, copyrights issues (i.e., copyright warning, reference to project's funding by EC, etc.), as well as references to terms and conditions for the website's usage along with the privacy policies (Figure 7).

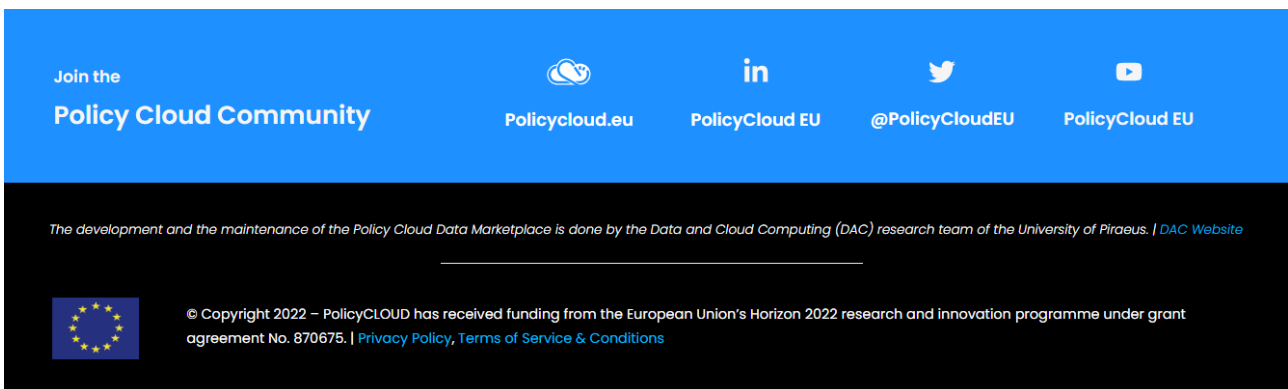


FIGURE 7 – FOOTER

Home page: The Home page illustrates some of the most popular and latest solutions provided into the Data Marketplace, along with some random suggested solutions, which are differentiating every time that a user reloads the Home page (Figure 9). What is more, the Home page depicts some relevant statistics about the supported collections and the offered assets. If a user tries to log in, the front-end sends an AJAX request to the corresponding interface of the back-end to get a valid token (JWT). If the response does not contain a successful message, the front-end presents to the user a corresponding error. In the case of a successful log in, the user is redirected to his/her account page. There is also a button that when it is pressed by the users, it redirects them to the Discover page. It should be noted that the basic assets' categories are shown in circles in the beginning of the Home page, where every circle category is interactive and if a user clicks on in, he/she is redirected in the Discover page, which illustrates the existing solutions of the corresponding chosen category. Also, the Home page contains a background image from the main PolicyCLOUD website, being designed in such a way to be consistent with it (Figure 8). Finally, in the final bottom section of the Home page, there exists an image with a link that redirects the user to the Create page, encourage the user to sign in and create a new solution to the Data Marketplace (Figure 10).

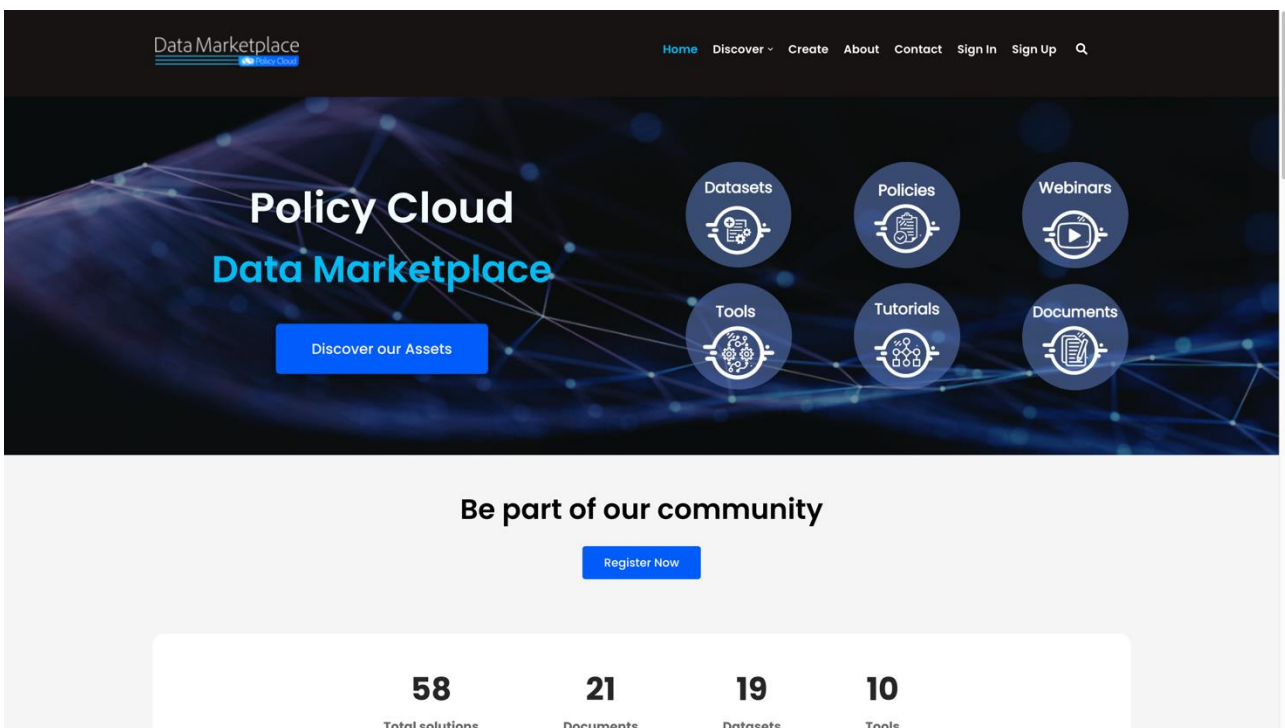


FIGURE 8 – HOME PAGE: UPPER VIEW

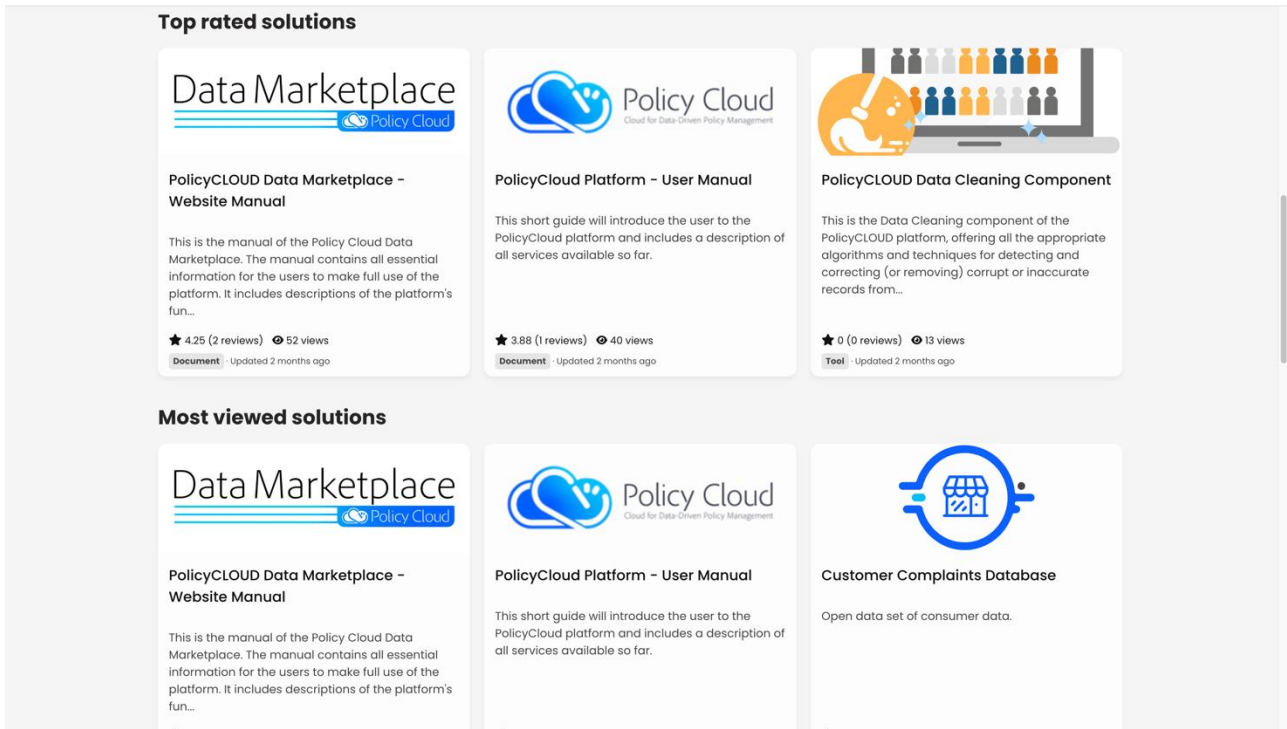


FIGURE 9 – HOME PAGE: MIDDLE VIEW

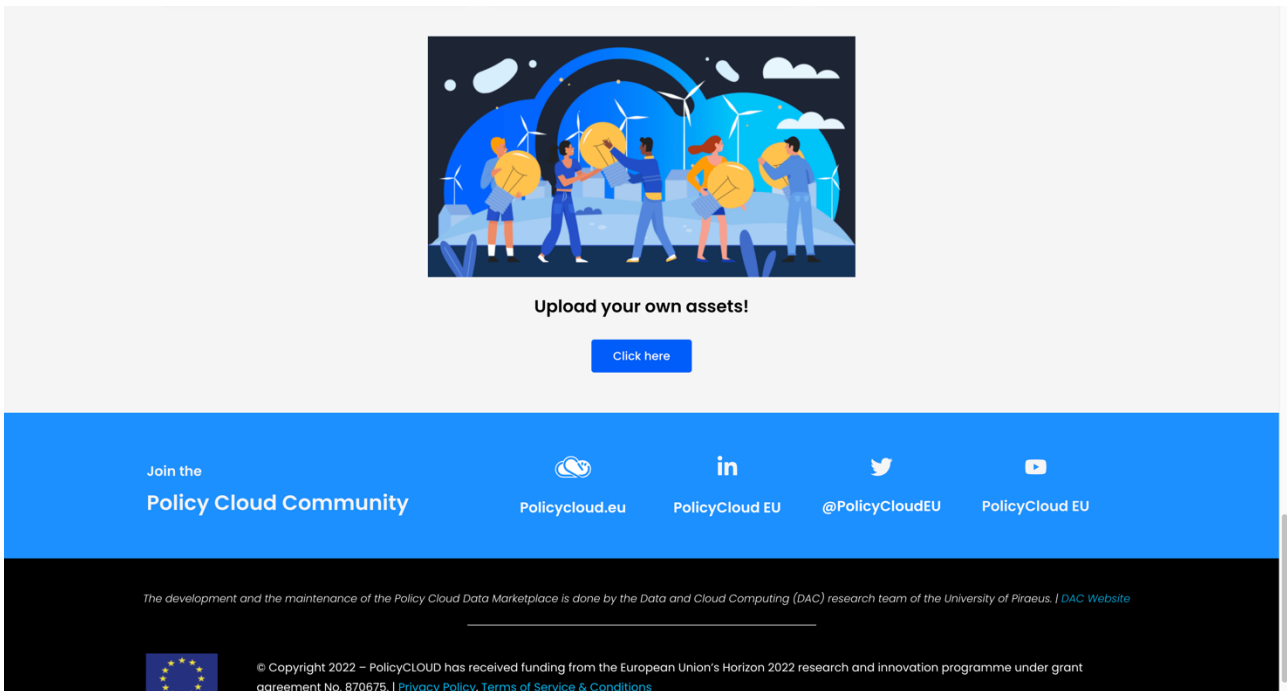
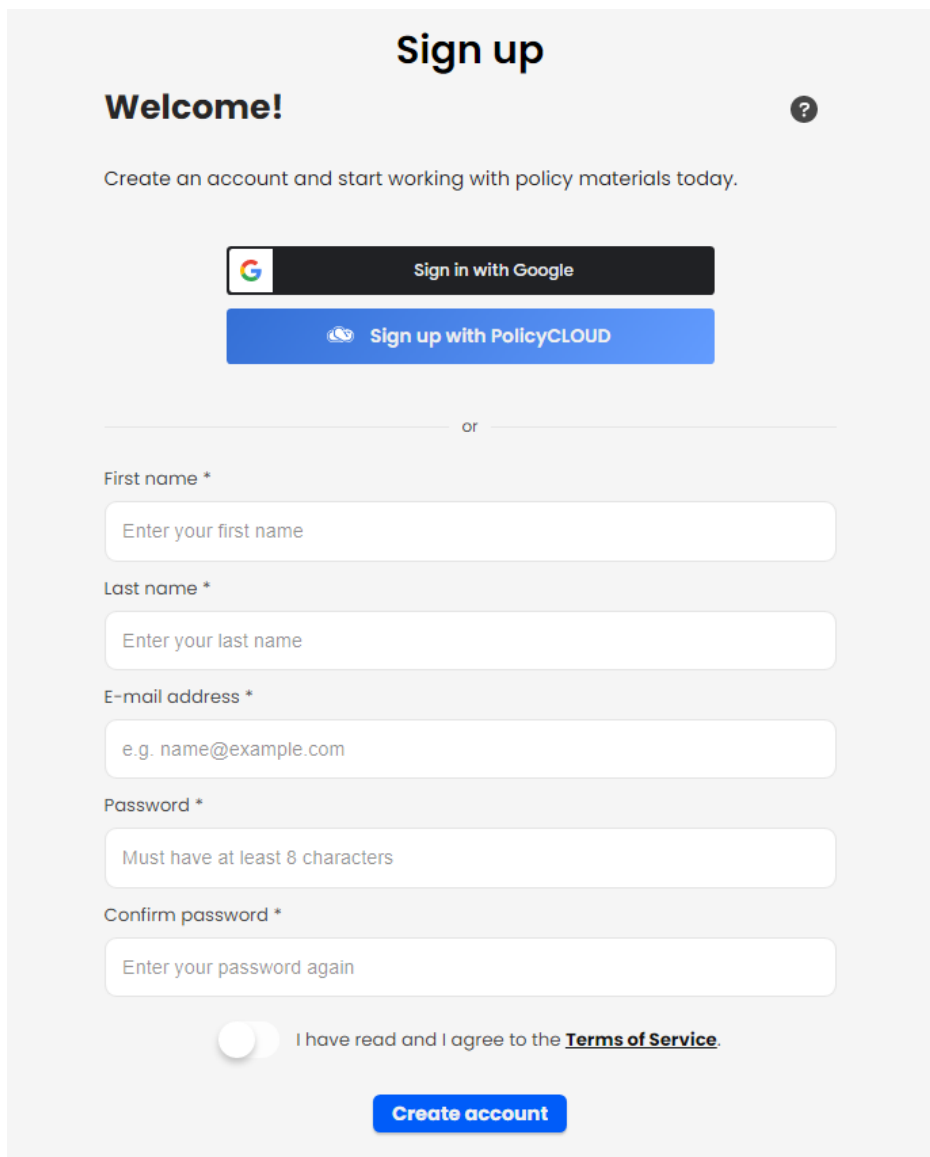


FIGURE 10 – HOME PAGE: LOWER VIEW

Sign up page: The Sign up page contains a form that a user must fill in and submit in order to register and access all the Data Marketplace’s content. As depicted in Figure 11, there are three (3) different ways for the users to sign up: 1) through their Google account, 2) through their credentials provided in the context of KeyCloak that is integrated into the overall PolicyCLOUD platform, and 3) by filling in the form of the Sign up page with their basic information. Specifically, the users must fill in their first and last name, their e-mail address, and a password for their account.



The screenshot shows a 'Sign up' page with the following elements:

- Header:** 'Sign up' title, 'Welcome!' subtitle, and a help icon (?)
- Text:** 'Create an account and start working with policy materials today.'
- Buttons:** 'Sign in with Google' (black) and 'Sign up with PolicyCLOUD' (blue)
- Separator:** 'or' text between two horizontal lines
- Form Fields:**
 - 'First name *' with placeholder 'Enter your first name'
 - 'Last name *' with placeholder 'Enter your last name'
 - 'E-mail address *' with placeholder 'e.g. name@example.com'
 - 'Password *' with placeholder 'Must have at least 8 characters'
 - 'Confirm password *' with placeholder 'Enter your password again'
- Terms:** A switch button and text 'I have read and I agree to the [Terms of Service](#)'
- Final Button:** 'Create account' (blue)

FIGURE 11 – SIGN UP PAGE

At the bottom of the Sign up page there is also a switch button that determines whether the users have read and accept the Terms and Conditions for the usage of the PolicyCLOUD Data Marketplace platform. By clicking the “Terms of Service” text/link, the users are redirected to the corresponding page, in order to be informed about the Data Marketplace terms of use before their registration.

Sign in page: The Sign in page consists of a simple form in which the users must insert their credentials and depending on whether the users are indeed registered users of the Data Marketplace or not, they are redirected to the Account page or they get an error message, respectively. To this context, there are supported three (3) different ways for the users to sign in (similarly with the Sign up process), where the first refers to the usage of their Google accounts, the second one refers to their credentials for the PolicyCLOUD instance of KeyCloak, and the third one refers to the provided form of the Sign in page with their Data Marketplace credentials.

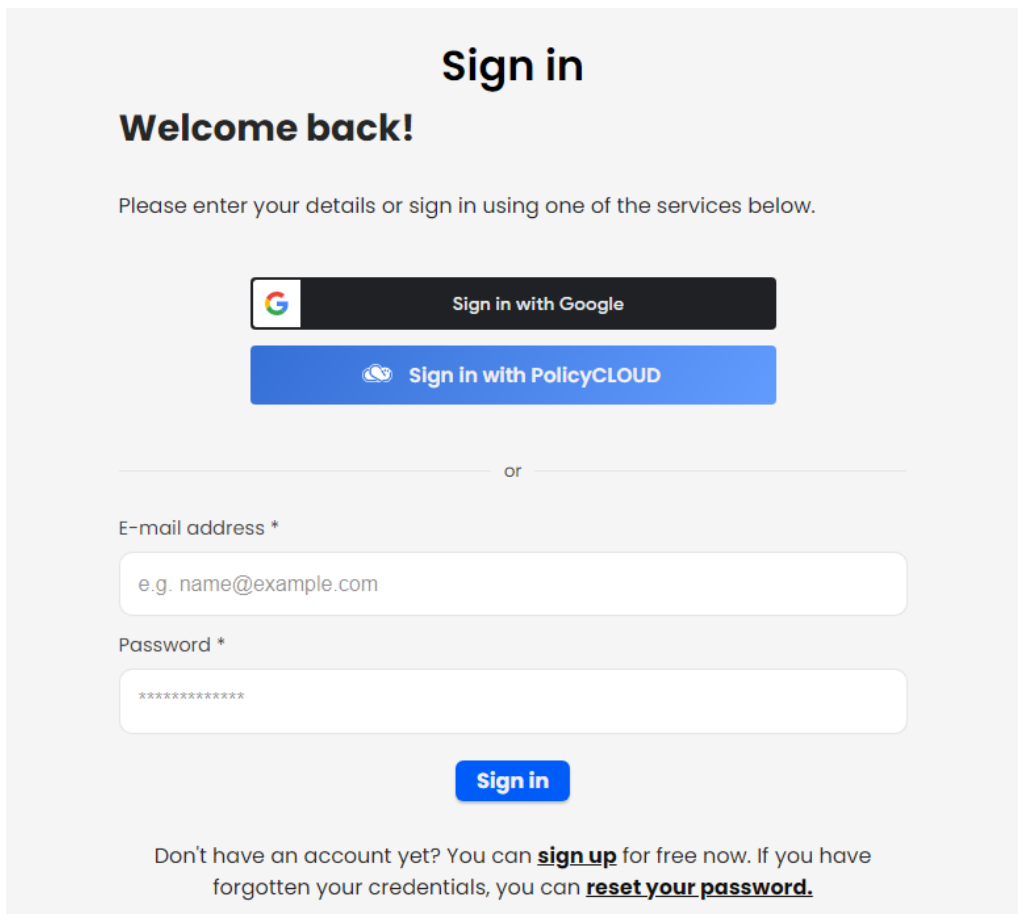


FIGURE 12 – SIGN IN PAGE

Account Page: In the Account page, the logged in users are able to see the various details of their profiles. More specifically, the Overview tab displays the overview of the users along with some statistics for their contributions to the Data Marketplace. The Solutions tab presents the solutions offered/created by the users, while the Reviews tab displays the reviews made by the users. Finally, there is the Profile tab that presents the personal and the account details of the users.

Regarding the Overview tab, the statistics that the page illustrates refer to the following: (i) the total number of solutions offered by the users, (ii) the number of the approved solutions, (iii) the number of total assets that users' solutions contain along with the total number of their downloads, (iv) the total

number of views and reviews that the users' solutions have, and (v) the average rating of all user's solutions (Figure 13).

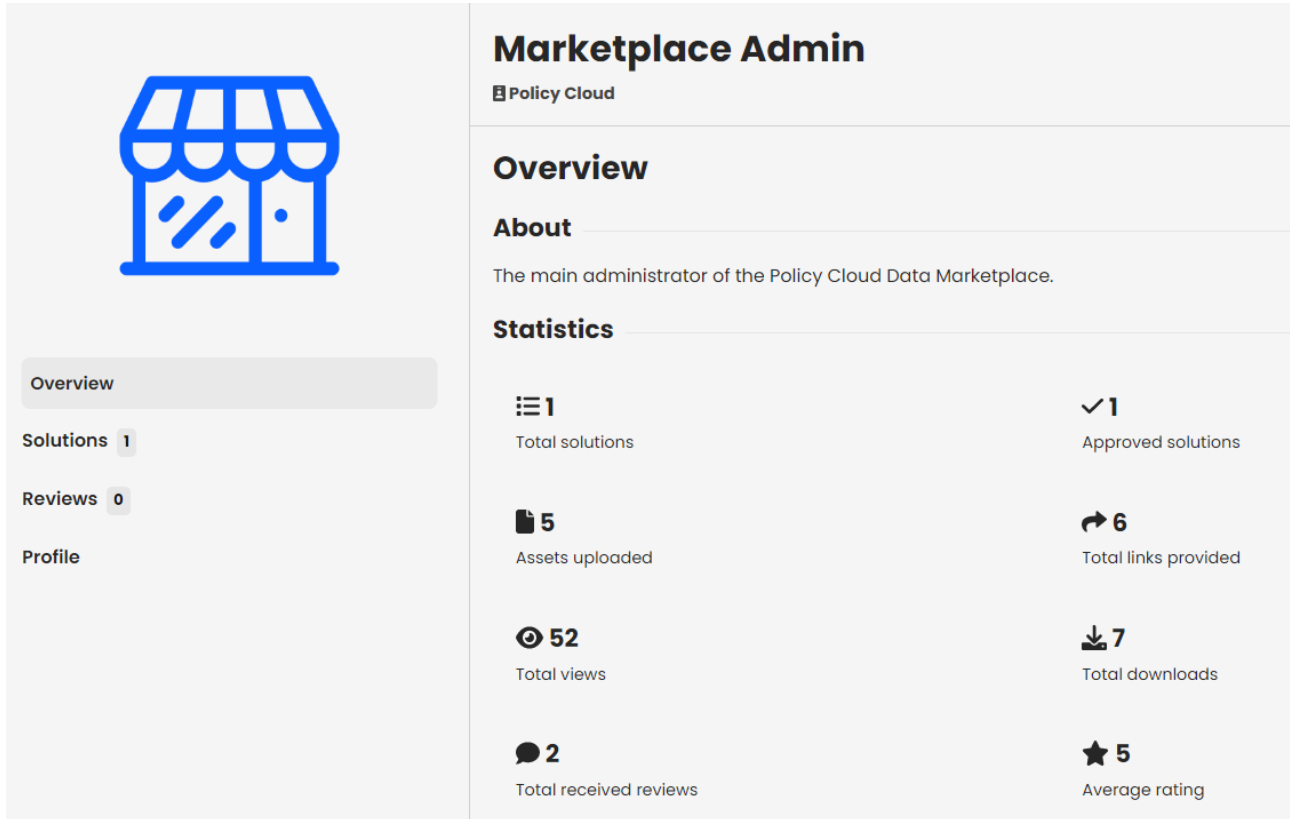


FIGURE 13 – ACCOUNT PAGE: OVERVIEW TAB

As for the Solutions tab, it displays the offered solutions of the users (Figure 14). If the Solutions tab is accessed by a visitor/other logged in user and not the account owner or an administrator, it displays only the approved solutions. Moreover, the Solutions tab supports both filtering and sorting options as in the Discover page, in order to present the solutions based on the users' preferences.

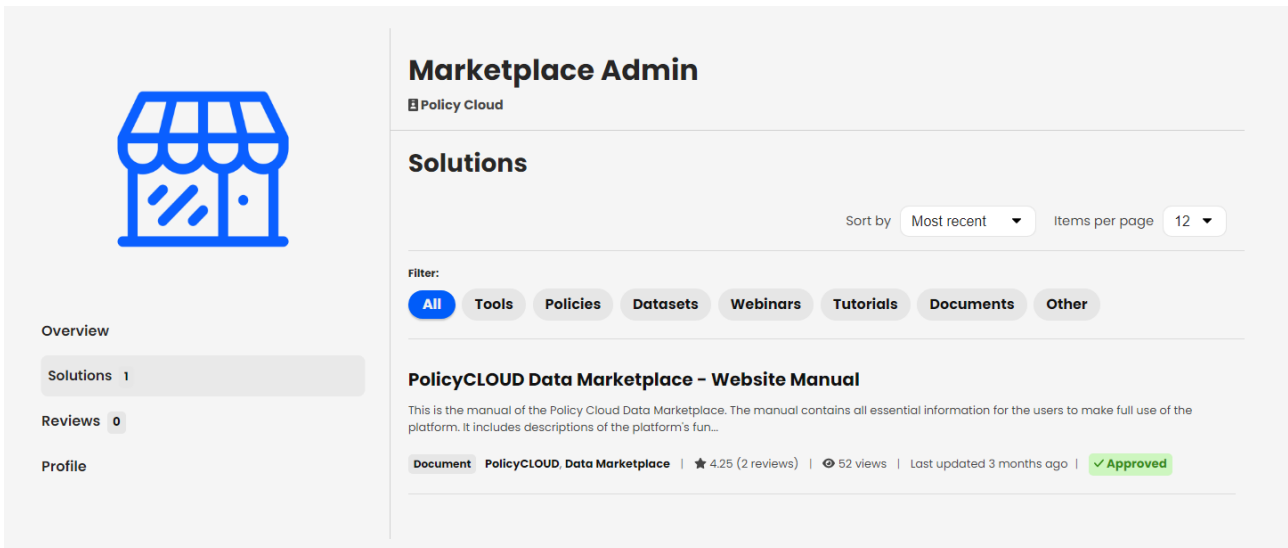


FIGURE 14 – ACCOUNT PAGE: SOLUTIONS TAB

Similarly to the Solutions tab, the Reviews tab displays the reviews made by the user, indicating the rating as well as the comment that the user has provided to the chosen solutions (Figure 15). This tab contains the same filtering and sorting options with the Solutions tab, having however the difference that the Reviews tab presents the same content whether the user is in visitor's mode or not.

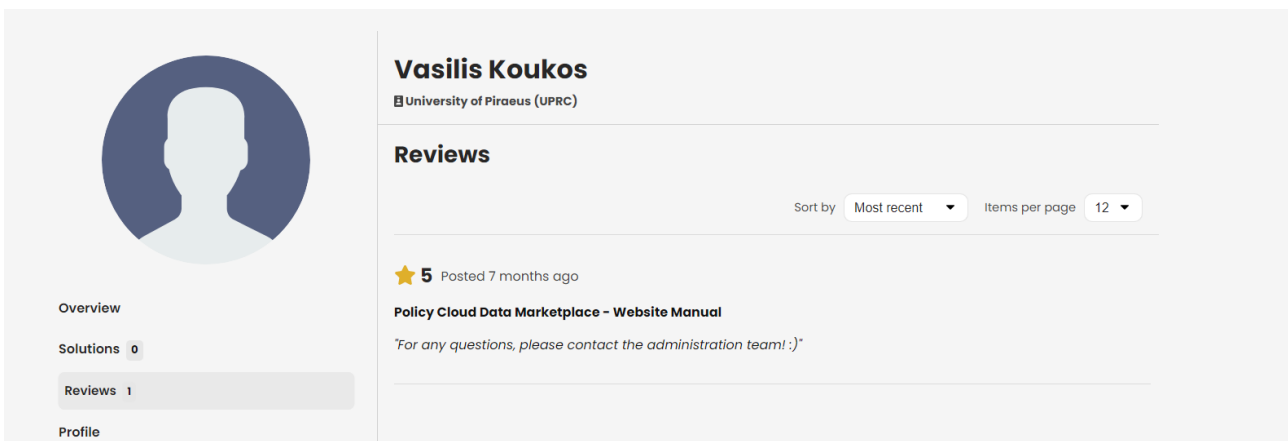
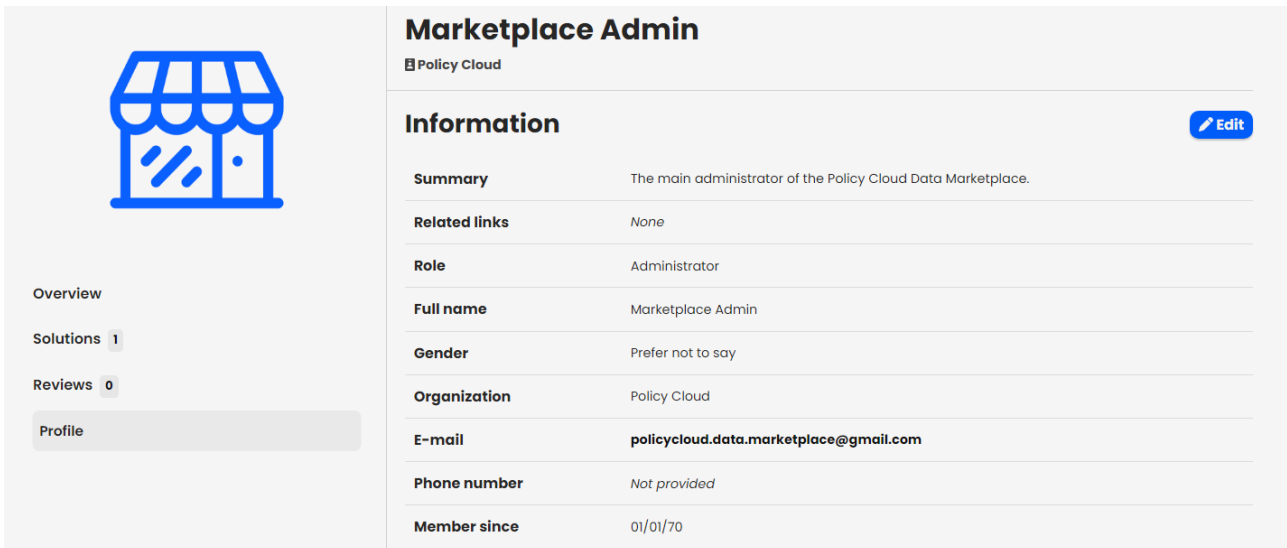


FIGURE 15 – ACCOUNT PAGE: REVIEWS TAB

Regarding the Profile tab, the users can see the personal and the account information/details of the displayed user (most of the presented information was provided by the users during their registration) (Figure 16). Moreover, this tab enables the users to edit/update their personal information (e.g., Full name, Gender, Organization, Phone number, etc.), request a copy of their data, as well as handle their account details (e.g., change their password or email, delete their account, etc.) and the connections to other services (e.g., Google account).



Marketplace Admin
Policy Cloud

Information Edit

Summary	The main administrator of the Policy Cloud Data Marketplace.
Related links	None
Role	Administrator
Full name	Marketplace Admin
Gender	Prefer not to say
Organization	Policy Cloud
E-mail	polycycloud.data.marketplace@gmail.com
Phone number	Not provided
Member since	01/01/70

Overview

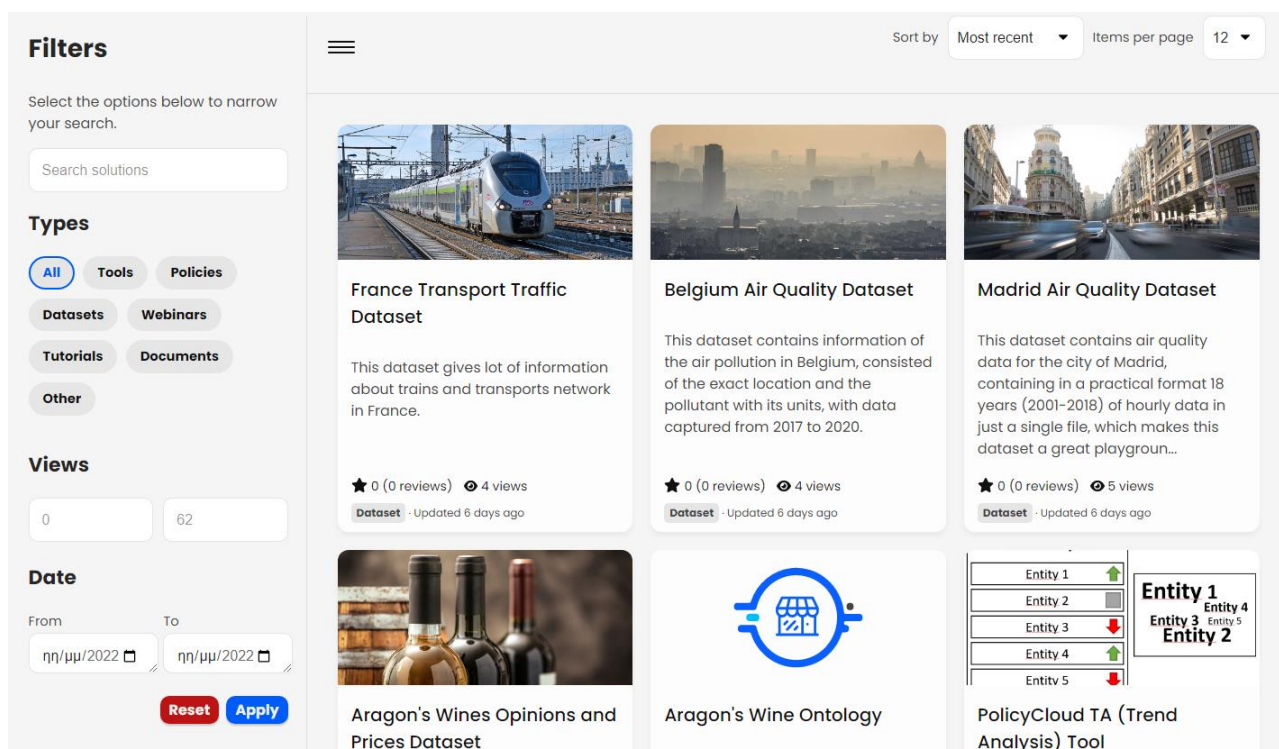
Solutions 1

Reviews 0

Profile

FIGURE 16 – ACCOUNT PAGE: PROFILE TAB

Discover page: In this page all the approved solutions are displayed, illustrating the main image of each solution and its basic information, including its title, short description, last updated date, number of views and reviews, average ratings and solution type (Figure 17). On the left side of the page, through the provided sidebar, the users can search for solutions based on their title by filling in the search bar. They can also filter the results based on a specific solution type, as well as the number of views and the creation/update dates of the solutions. On the upper-right side of the page, there is a button for sorting the results and selecting the desired number of the items (i.e., solutions) that will be presented per page.



Filters

Select the options below to narrow your search.

Search solutions

Types

All (selected) Tools Policies Datasets Webinars Tutorials Documents Other

Views

0 62

Date

From: 01/01/2022 To: 01/01/2022

Reset Apply

Sort by: Most recent Items per page: 12

France Transport Traffic Dataset

This dataset gives lot of information about trains and transports network in France.

★ 0 (0 reviews) 👁 4 views

Dataset · Updated 6 days ago

Belgium Air Quality Dataset

This dataset contains information of the air pollution in Belgium, consisted of the exact location and the pollutant with its units, with data captured from 2017 to 2020.

★ 0 (0 reviews) 👁 4 views

Dataset · Updated 6 days ago

Madrid Air Quality Dataset

This dataset contains air quality data for the city of Madrid, containing in a practical format 18 years (2001-2018) of hourly data in just a single file, which makes this dataset a great playground...

★ 0 (0 reviews) 👁 5 views

Dataset · Updated 6 days ago

Aragon's Wines Opinions and Prices Dataset

Aragon's Wine Ontology

PolicyCloud TA (Trend Analysis) Tool

Entity 1 Entity 2 Entity 3 Entity 4 Entity 5

Entity 1 Entity 2 Entity 3 Entity 4 Entity 5

FIGURE 17 – DISCOVER PAGE

As for the sorting options, the users can sort the results by their average rating, their number of views, their creation/update date, alphabetically based on their title and (after a search) based on the most relevant results.

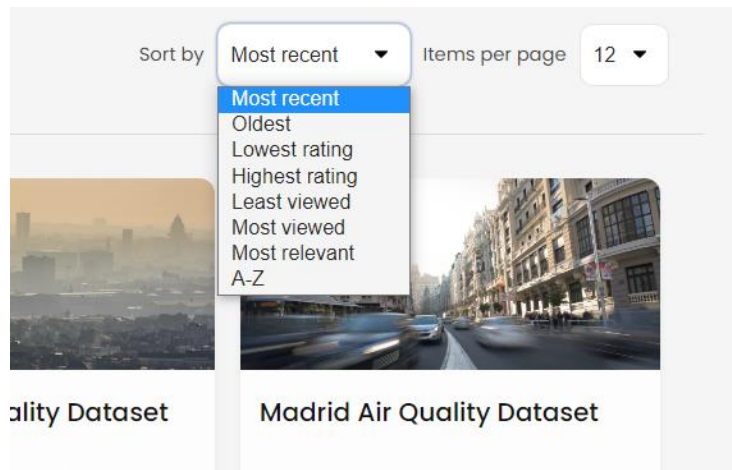


FIGURE 18 – DISCOVER PAGE: SORTING OPTIONS

When the users click on a solution, they are redirected to the specific Solution page. To this point it should be noted that the Discover page has the same view for both logged in and non-logged in users. It communicates with the back-end via the REST API and loads the information dynamically. In addition, through its responsive mode and in relation to the type of the device that the user possesses, the display is adapted accordingly, while the side bar appears only as a drop-down menu.

Solution page: The Solution page displays the main information of a solution, where a logged in user has different views than a non-logged in user. All the users can view the title of the solution, the owner, the type of the solution, the solution keywords, the average ratings, the number of reviews and the number of views. In the case of logged in users, the latter have access to the full description of the solution, the gallery with images and videos of the solution, whilst they are also able to download all the offered assets/files that are part of the solution. In case of non-logged in/authorized users, the latter do not have access to the assets and the gallery, whereas they can only see a short description of the solution. If the users are the providers or administrators, they can update the information that is displayed, while all the provided solutions need a permission from an administrator in order to be publicly displayed to all the users. Based on this, if a solution is approved, an indication “approved” is displayed, otherwise, an indication “pending” is displayed.

Logged in User View: A logged in user sees the entire content of a solution, the image gallery, and has access to the assets, as depicted in Figure 19.

PolicyCloud Data Cleaning Component

Argyro Mavrogiorgou © UPRC Radicalization, Winery, Smart city ★ 0 (0 reviews) 17 views Last updated 3 months ago **Tool**

Assets

Files

- Data_Cleaning_Architecture.pdf v1 - 353.81 KB - 3 months ago

Images

- Architecture.png v1 - 153.48 KB - 3 months ago
- cover.png v1 - 78.77 KB - 3 months ago

Description

This is the Data Cleaning component of the PolicyCloud platform, offering all the appropriate algorithms and techniques for detecting and correcting (or removing) corrupt or inaccurate records from all the collected data, correspondingly.

Gallery

Reviews

No reviews yet.

FIGURE 19 – SOLUTION PAGE: VIEW FOR LOGGED IN USERS

Owner View: If the user is the provider, both an edit and a delete button appear in the page, which enable the editing of the information, and the management of the solutions' assets (i.e., upload, update, delete). Every change made to the information of the solution has to be approved by an administrator in order to be displayed. If the change has been approved, a green button "approved" appears in the page that is only visible to the owner (Figure 20).

PolicyCloud Data Cleaning Component ✓ Approved

Argyro Mavrogiorgou © UPRC Radicalization, Winery, Smart city ★ 0 (0 reviews) 17 views Last updated 3 months ago **Tool** **Edit**

Assets

Files

- Data_Cleaning_Architecture.pdf v1 - 353.81 KB - 3 months ago

Images

- Architecture.png v1 - 153.48 KB - 3 months ago
- cover.png v1 - 78.77 KB - 3 months ago

Description

This is the Data Cleaning component of the PolicyCloud platform, offering all the appropriate algorithms and techniques for detecting and correcting (or removing) corrupt or inaccurate records from all the collected data, correspondingly.

Gallery

Reviews

No reviews yet.

FIGURE 20 – SOLUTION PAGE: VIEW FOR SOLUTION PROVIDERS AND ADMINISTRATORS

Non-logged in User View: A non-logged in user is able to see only the basic information of a solution, which contains a short description and metadata such as the solution’s owner, number of views and reviews, average rating, and others (Figure 21).

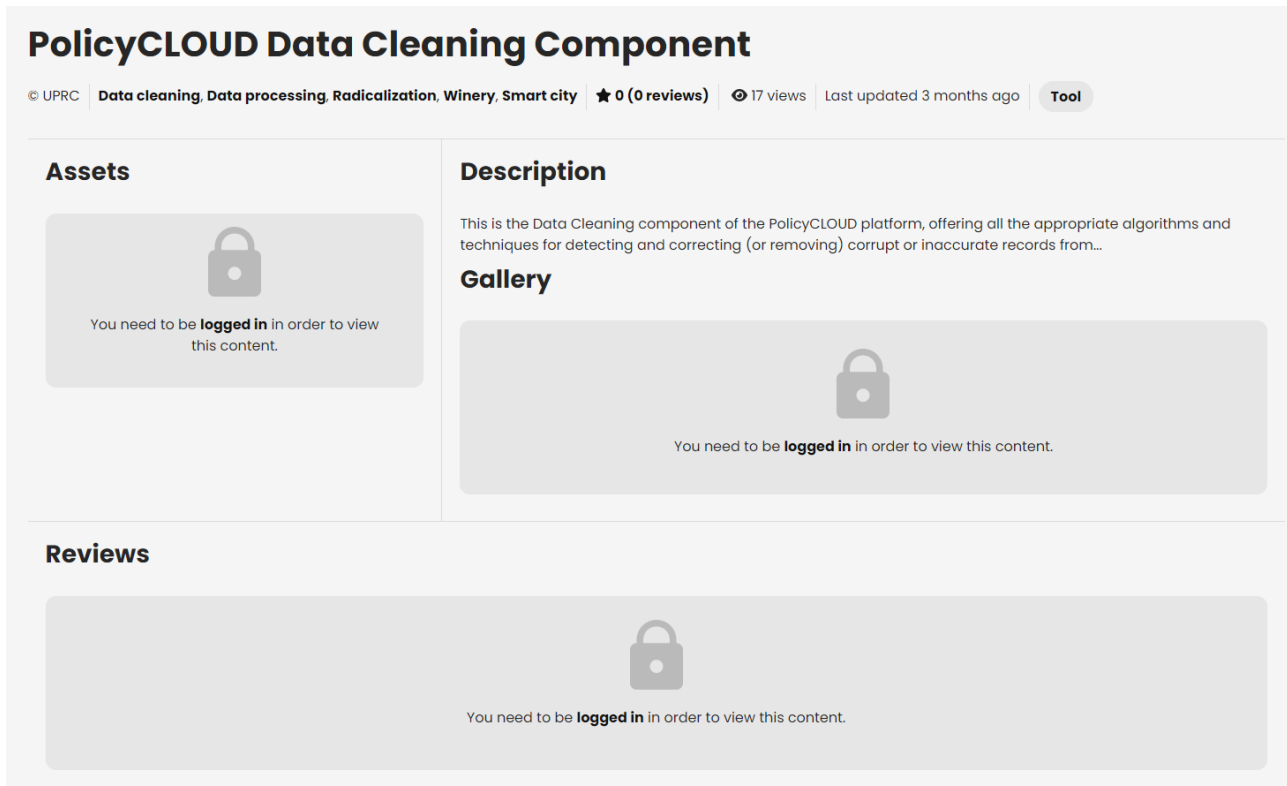
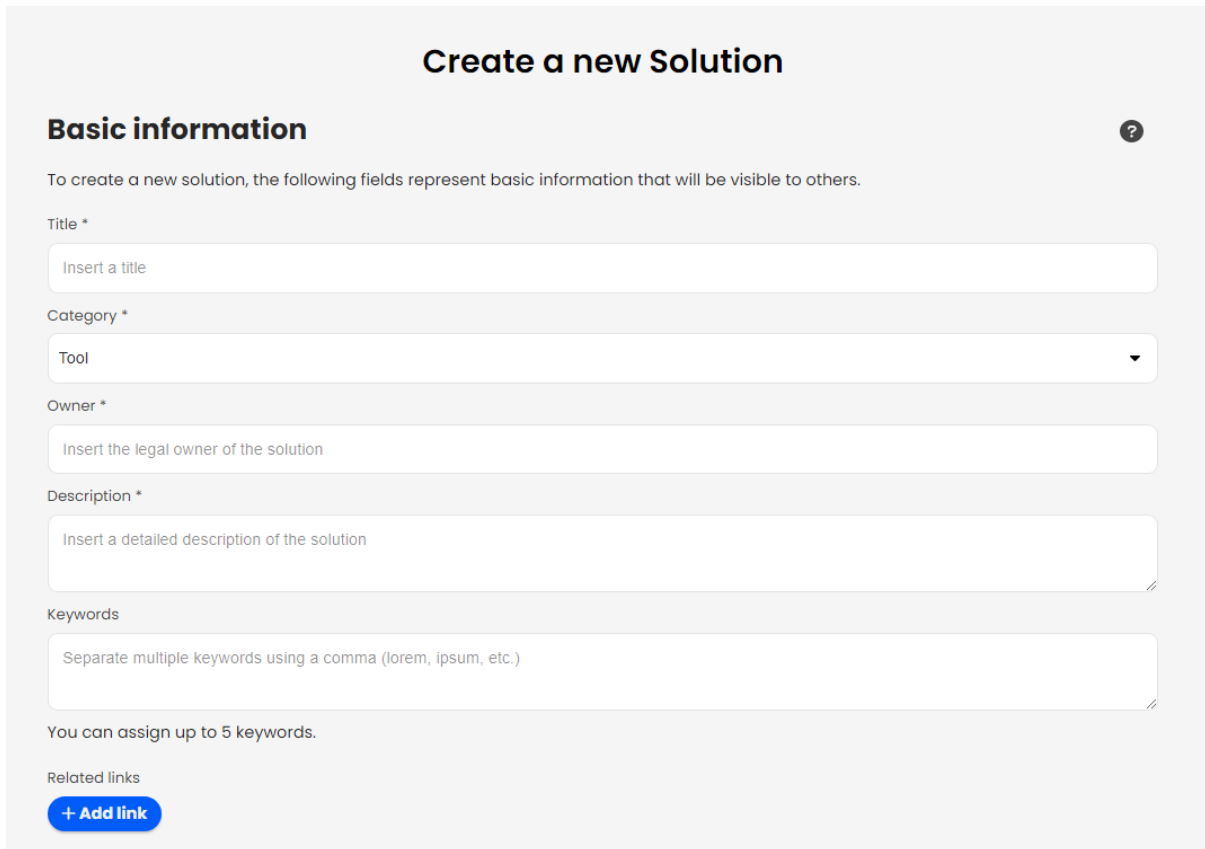


FIGURE 21 – SOLUTION PAGE: VIEW FOR NON-LOGGED IN USERS

Create page: The Create page can be only accessed by all registered and logged in users who can create a new solution to the Data Marketplace by filling in the form that the page contains. This page is only available to registered users who can find it either from the navigation bar or from their account page (i.e., in the Solution tab by pressing the Create button). More specifically, this page contains a form that is divided into three (3) discrete parts. The first part (Figure 22) includes several basic information about the new offered solution such as the solution’s title, category/type, detailed description about the offered content, owner of the solution, some optional related links for external information or assets, and some optional keywords that can distinguish the solution from other solutions and can enhance the search functionality.



Create a new Solution

Basic information ?

To create a new solution, the following fields represent basic information that will be visible to others.

Title *

Insert a title

Category *

Tool

Owner *

Insert the legal owner of the solution

Description *

Insert a detailed description of the solution

Keywords

Separate multiple keywords using a comma (lorem, ipsum, etc.)

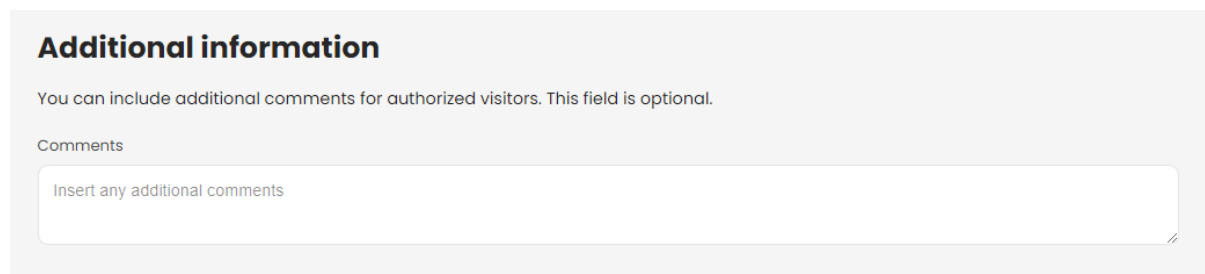
You can assign up to 5 keywords.

Related links

[+ Add link](#)

FIGURE 22 – CREATE PAGE: BASIC INFORMATION

The second part of the form (Figure 23) has one (1) additional field, the “Comments” field, which is not mandatory and can be used by the provider/creator of the solution in order to add comments that are displayed only to logged in users. This field can be useful in case of sensitive information (e.g., contact details, promotional/discount codes).



Additional information

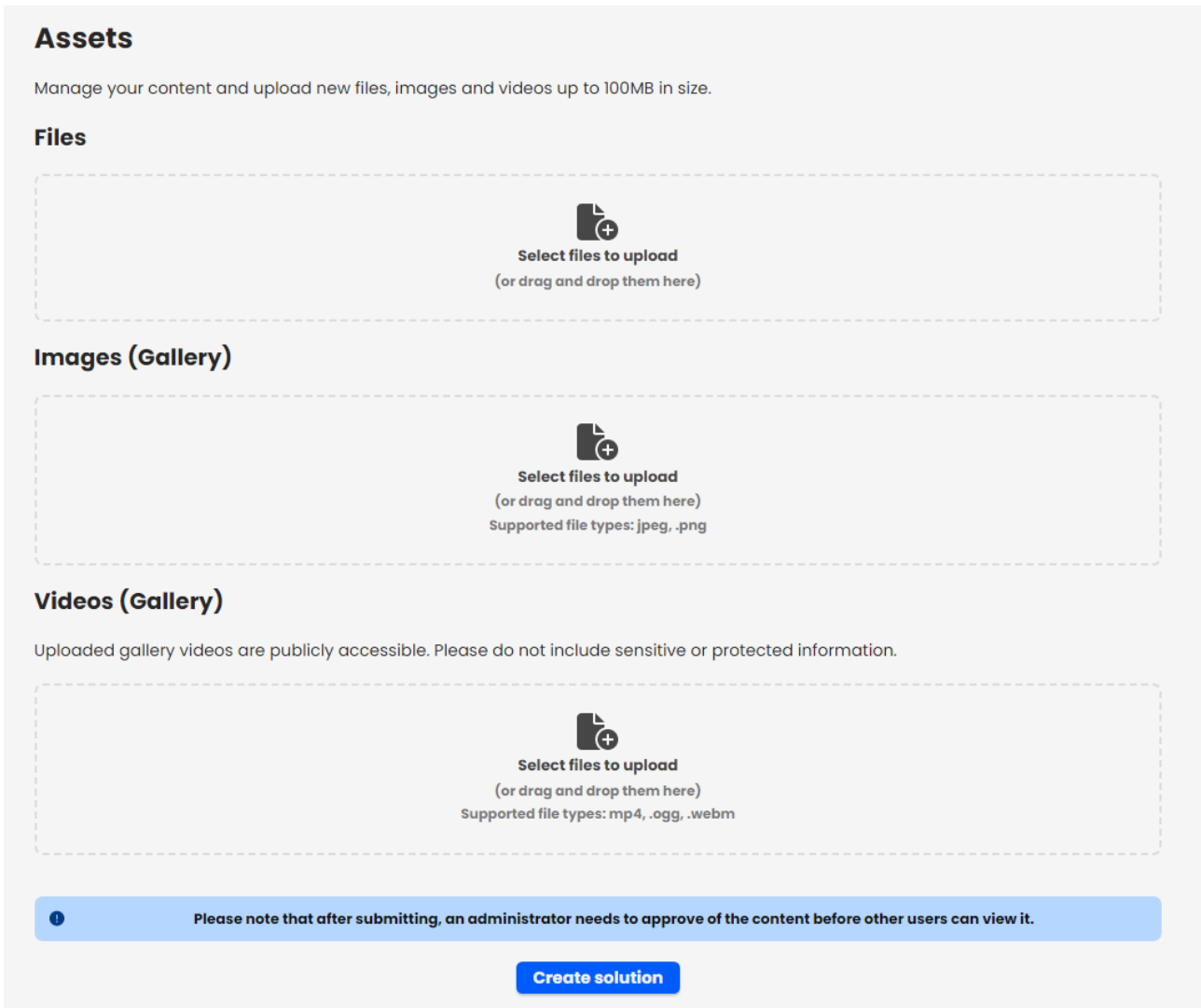
You can include additional comments for authorized visitors. This field is optional.

Comments

Insert any additional comments

FIGURE 23 – CREATE PAGE: ADDITIONAL INFORMATION

The third part of the form (Figure 24) refers to the Assets’ section, where the first assets of the solution can be uploaded. More specifically, from this area, the provider of the solution can upload files, images, or videos for the solution. It is noted that this form is used only for the creation of a solution, since the users are able to update their solutions and add, update or delete the assets through the Solution page, at a later stage.



Assets

Manage your content and upload new files, images and videos up to 100MB in size.

Files

Select files to upload
(or drag and drop them here)

Images (Gallery)

Select files to upload
(or drag and drop them here)
Supported file types: jpeg, .png

Videos (Gallery)

Uploaded gallery videos are publicly accessible. Please do not include sensitive or protected information.

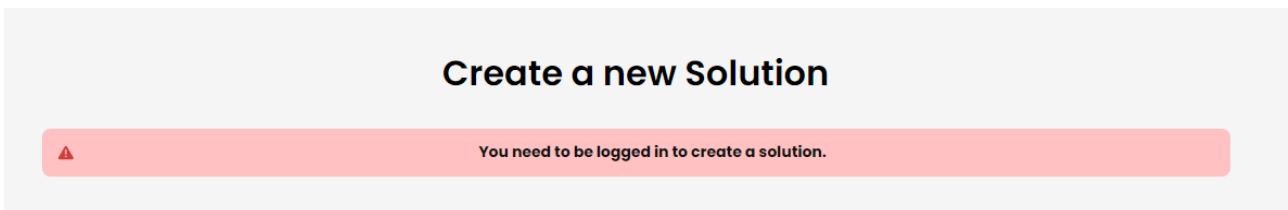
Select files to upload
(or drag and drop them here)
Supported file types: mp4, .ogg, .webm

Please note that after submitting, an administrator needs to approve of the content before other users can view it.

[Create solution](#)

FIGURE 24 – CREATE PAGE: ASSETS SECTION

Finally, when non logged in users try to access the Create page, a warning message that encourages them to sign in is displayed in their screen (Figure 25).



Create a new Solution

You need to be logged in to create a solution.

FIGURE 25 – CREATE PAGE: VIEW FOR NON LOGGED IN USERS

About page: The About page is a public page that provides information about the PolicyCLOUD Data Marketplace, including its structure, scope, and offerings. The scope of this page is to guide users' navigation to the website. In addition, at the top of the page, there is a prompt button to "Register" to the Data Marketplace that links to the sign-up page. Moreover, the page displays the PolicyCLOUD Data Marketplace's basic flow, as well as describes the purpose of the PolicyCLOUD project in general, and specific scope of the PolicyCLOUD Data Marketplace (Figure 26).

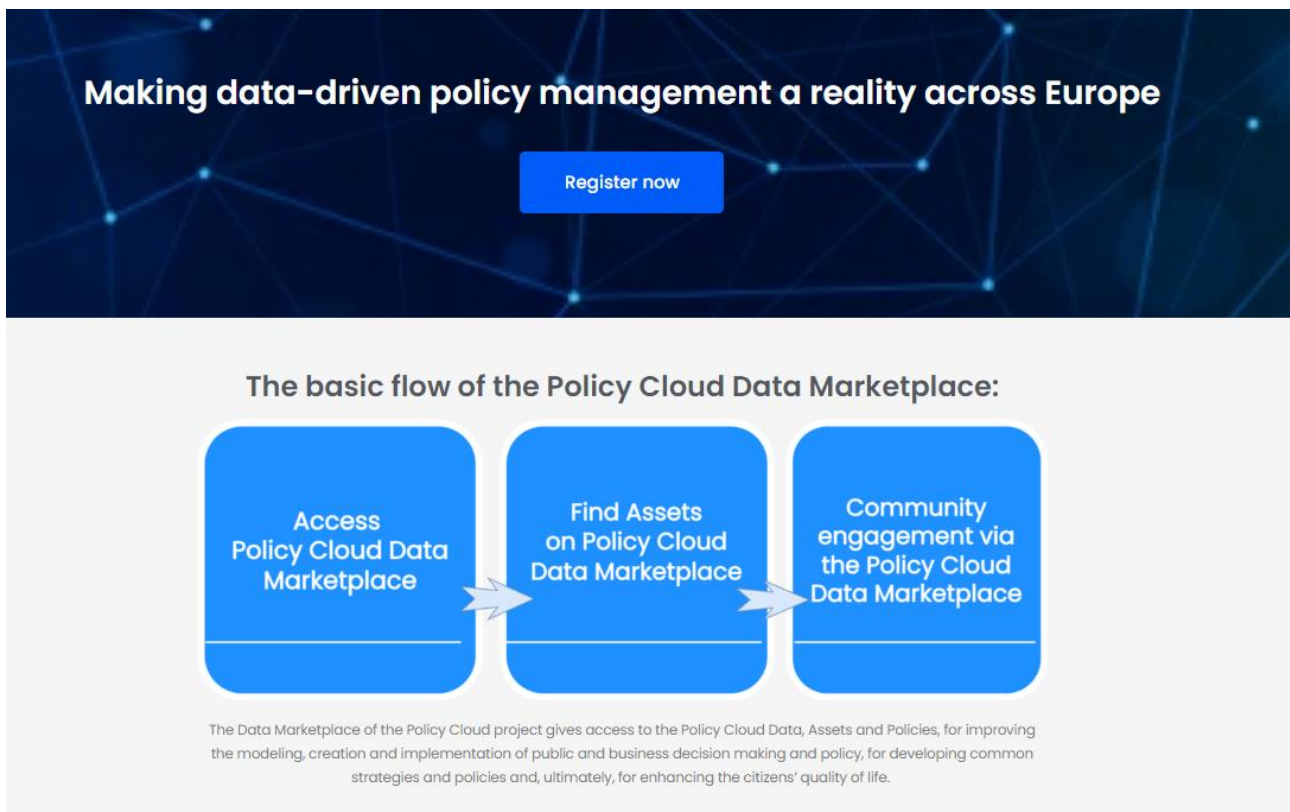
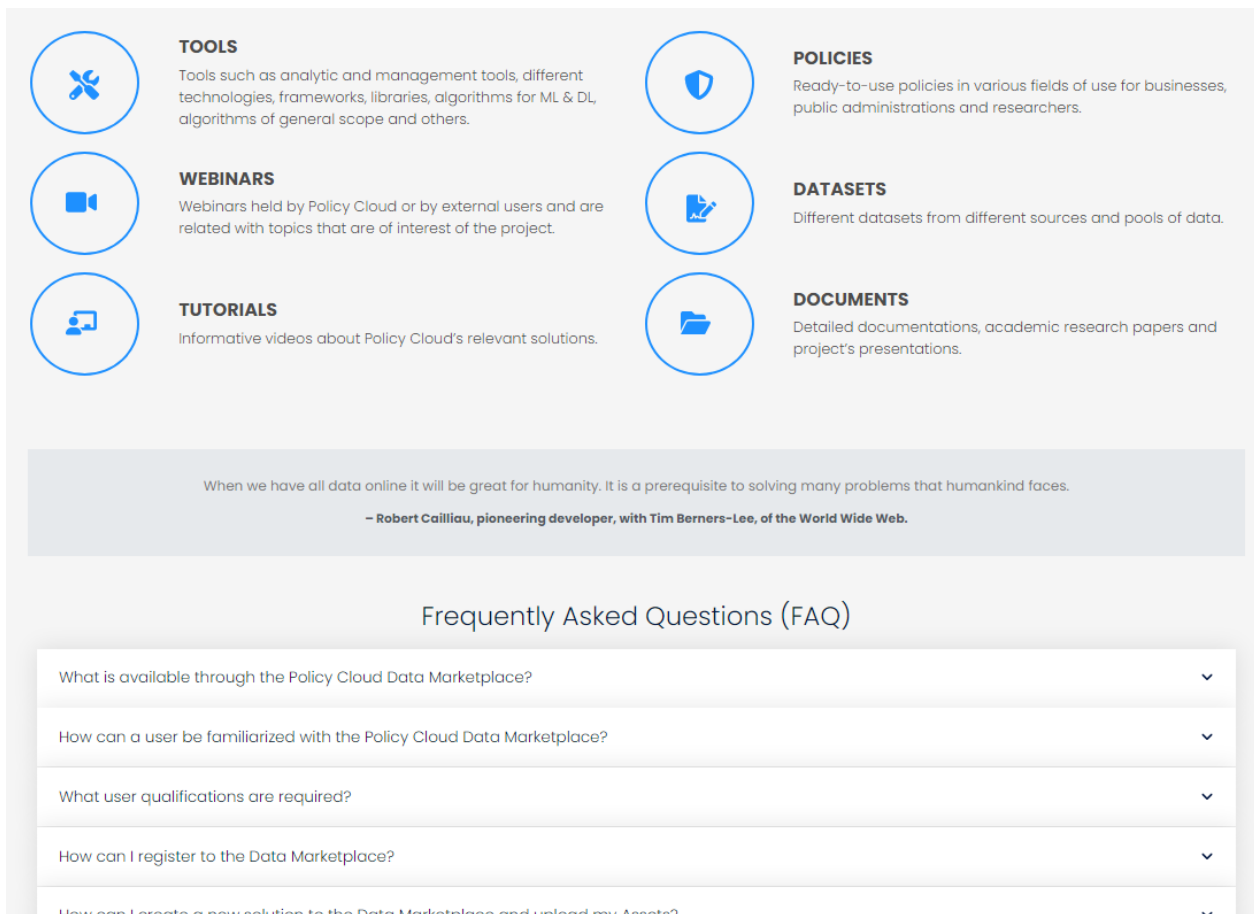


FIGURE 26 – ABOUT PAGE: UPPER VIEW

Furthermore, the About page includes a description for the types of solutions that the PolicyCLOUD Data Marketplace supports, assisting the users in determining which category to search, in accordance with their specific needs. Finally, the About page has a list of frequently asked questions for answering to users' potential concerns (Figure 27).



TOOLS
Tools such as analytic and management tools, different technologies, frameworks, libraries, algorithms for ML & DL, algorithms of general scope and others.

WEBINARS
Webinars held by Policy Cloud or by external users and are related with topics that are of interest of the project.

TUTORIALS
Informative videos about Policy Cloud's relevant solutions.

POLICIES
Ready-to-use policies in various fields of use for businesses, public administrations and researchers.

DATASETS
Different datasets from different sources and pools of data.

DOCUMENTS
Detailed documentations, academic research papers and project's presentations.

When we have all data online it will be great for humanity. It is a prerequisite to solving many problems that humankind faces.
– Robert Cailliau, pioneering developer, with Tim Berners-Lee, of the World Wide Web.

Frequently Asked Questions (FAQ)

- What is available through the Policy Cloud Data Marketplace? ▾
- How can a user be familiarized with the Policy Cloud Data Marketplace? ▾
- What user qualifications are required? ▾
- How can I register to the Data Marketplace? ▾
- How can I create a new solution to the Data Marketplace and upload my Assets? ▾

FIGURE 27 – ABOUT PAGE: BOTTOM VIEW

Error Messages: In case of an error, a red bar appears with the appropriate message.

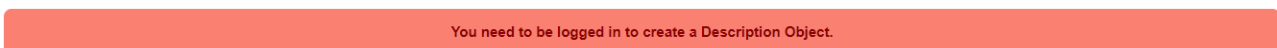


FIGURE 28 – EXAMPLE OF ERROR MESSAGE

Terms and Condition page: The “Terms and Conditions” agreement governs the contractual relationship between a service provider and its users. Towards this direction, this page is intended to provide this agreement to the PolicyCLOUD Data Marketplace users (Figure 29).

Terms and Conditions

By indicating your acceptance of the PolicyCLOUD Data Marketplace Terms and Conditions (when creating a registered account), you accept that your use of the Service must abide by all of the terms and conditions laid down below (“T&Cs”).

If you do not agree to the T&Cs, you may not use the Service.

The T&Cs are enforceable like any written agreement signed by you.

✓ 1. DEFINITIONS
✓ 2. AGREEMENT
✓ 3. MODIFICATIONS TO THE AGREEMENT
✓ 4. SERVICE DESCRIPTION
– 5. ACCESS TO THE SERVICE
5.1 The Authorised Users may access and use the Service in accordance with the T&Cs and the Documentation.
5.2 Without prejudice to Section 6 (Contractors and Affiliates), use of and access to the Service are permitted only to the User.
✓ 6. CONTRACTORS AND AFFILIATES

FIGURE 29 – TERMS AND CONDITIONS PAGE

Privacy Policy page: In privacy legislation, a privacy policy is a declaration or legal document that reveals the ways that a party collects, uses, discloses, and maintains a customer’s or a client’s personal data. This page provides information for such issues in the context of the PolicyCLOUD Data Marketplace (Figure 30).

Privacy Policy

The Policycloud project consortium (“**PolicyCLOUD**”), according to the provisions of Regulation EU No. 2016/679 (“General Data Protection Regulation” – “**GDPR**”), provides the following information on the processing of the personal data (the “**Data**”) of the end users of the PolicyCLOUD Data Marketplace platform (the “**Platform**”).

✓ **Who we are and what Data we process?**

– **Why do we process the Data and on which legal basis?**

PolicyCLOUD uses the Data for the following purposes:

- Creation and management of the end user account and operation of the Platform. The use of the Data is necessary to allow the creation and management of the account of the end user and to allow their use of the Platform. The legal basis of the processing is the provision of the services related to the Platform, according with art. 6, par. 1, let. b) GDPR.
- Ensuring the security and the proper use of the Platform. PolicyCLOUD processes the Data of the end user to the extent necessary to guarantee the security of the Platform, as required also by relevant legal provisions on information security. The legal basis of this processing is both compliance with legal requirements, according with art. 6, par. 1, let. c) GDPR, and the legitimate interest of PolicyCLOUD and the organization using the Platform to protect the Platform and the information included in the same Platform, according with art. 6, par. 1, let. f) GDPR.

FIGURE 30 – PRIVACY POLICY PAGE

Contact page: The Contact page contains a form through which the users can communicate with the administrators of the PolicyCLOUD Data Marketplace (Figure 31).

Contact us

Whether you have a question about the content of Data Marketplace or the team of EU experts behind Policy Cloud, our team is ready to answer all your questions.

First Name *

Last Name *

Email *

Organisation

Message *

Privacy Policies & Cookies *

Accept

FIGURE 31 – CONTACT PAGE

2.3 Baseline Technologies and Tools

The following sub-sections are describing the baseline technologies that both the back-end and the front-end of the Data Marketplace exploit in order to implement its capabilities and functionalities.

2.3.1 Back-end

The back-end is the core base of the Data Marketplace and it has been developed using a variety of technologies/tools. First of all, its components are containerized in Docker images [3] that, among others, offer more efficient management and maintenance, enabling continuous updates and integration. Python [4] is used as the programming language along with the Flask framework [5], which is a Web Server Gateway Interface (WSGI) developed in Python, and implements RESTful APIs to handle the respective HTTP requests.

The offered assets are stored in a MongoDB No-SQL database [6] that is used in combination with the hosting operating system (OS) for storing and retrieving large files/objects, of any format. Moreover, Gunicorn [7], a Python WSGI HTTP Server for UNIX, is utilized with NGINX [8], an open-source high-performance HTTP web server and reverse proxy, since Flask is not optimum for production mode, and thus, both tools extend the Flask framework in order to enable access to multiple users at the same time.

2.3.2 Front-end

The front-end has been implemented using various web technologies (HTML, CSS, Bootstrap, PHP, JavaScript, jQuery) and it is functional using PHP and JavaScript technologies. It also exploits WordPress [9] and various plugins of it, in order to manage the content that is presented. More specifically, for the implementation of the front-end, the following tools were used:

- **WordPress:** A major part of the platform was designed with customized code based on the architecture logic of WordPress. A minor part was introduced manually, by utilizing the Elementor editor of WordPress [10].
- **Elementor:** Utilized at various stages of design, mainly for the header.

Except for these, a **custom-made plugin** with the name “**PolicyMS Plugin**” was implemented, for the connection between the front-end and the back-end, as well as for the correct display of the assets’ information. The main methods of the plugin are called from WordPress with hooks, and by placing short code names of methods on each page, for each interaction with the back-end.

The plugin contains authentication methods, checks if the user is valid, connects to the back-end with post request, creates the user’s token and returns the JSON response to the WordPress page. To be more specific, when a user tries to log in, after filling in the login form, the information from the browser is sent by AJAX request to the WordPress custom-made functions, checking if the values are empty. The login information is then sent, by post request, to the back-end API for verification. The back-end API returns the JSON response with user’s information and the user’s token or error, while a WordPress’s encrypted security token (nonce) is created. If the token is valid, the information from the database (the dynamic

content) is displayed with HTML, jQuery, PHP in the browser and the encrypted token is temporarily stored in the browser storage. The aforementioned process is also depicted in Figure 32.

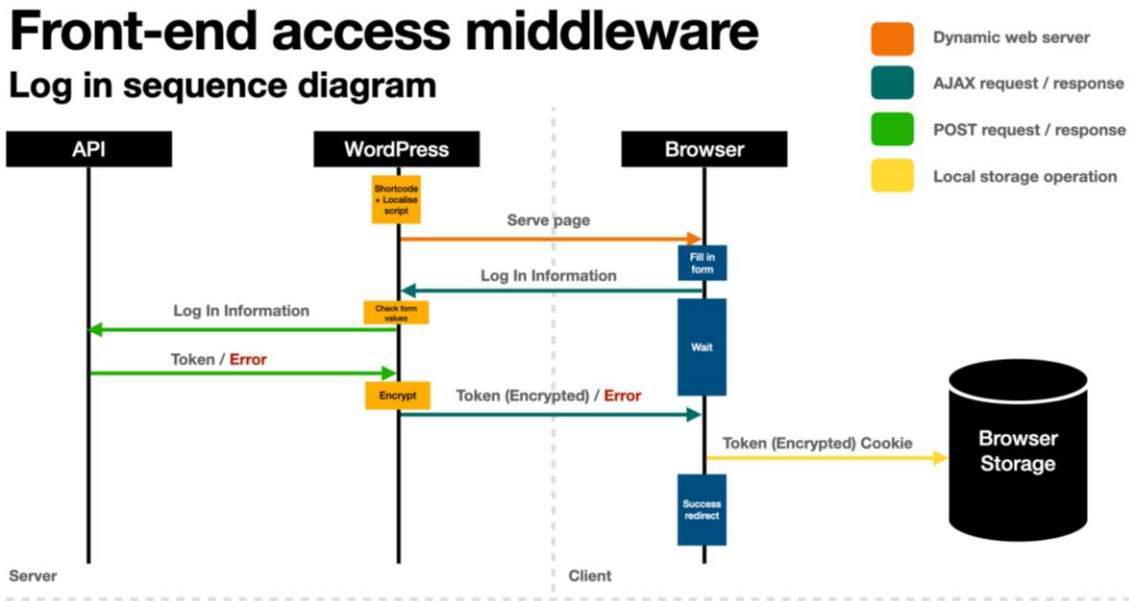


FIGURE 32 – FRONT-END ACCESS MIDDLEWARE

The admin class is responsible for the extensions that are added to the WordPress dashboard, to which administrators have access (Figure 33).



FIGURE 33 – DASHBOARD ADD SETTINGS

With the `add_admin_settings ()` method, the administrator adds a field to their menu to save the key with which the system will communicate with the back-end. The key is valid until it expires, after one month, for security reasons.

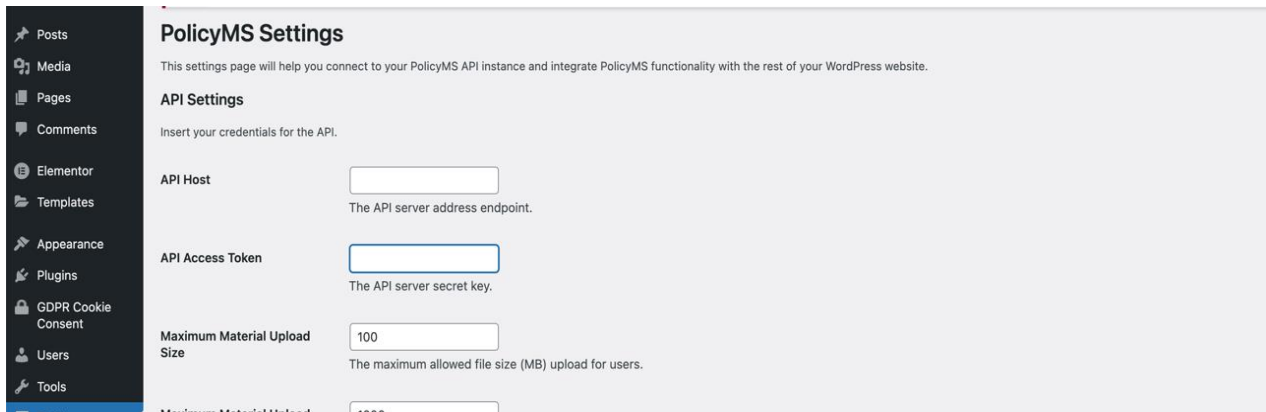


FIGURE 34 – DASHBOARD ADMIN VIEW SETTINGS

When a user tries to an Access Display Asset’s information’s page, such as the Discover page, the WordPress functions browser sends a post request to the back-end API that returns the response with JSON assets information or an error message. If the assets information is valid, it is displayed to the browser through dynamic content with HTML, jQuery and PHP. If the token exists, it is stored in a cookie in the browser storage. The aforementioned process is also depicted in Figure 35.

Read description objects

Authenticated (token-based) actions

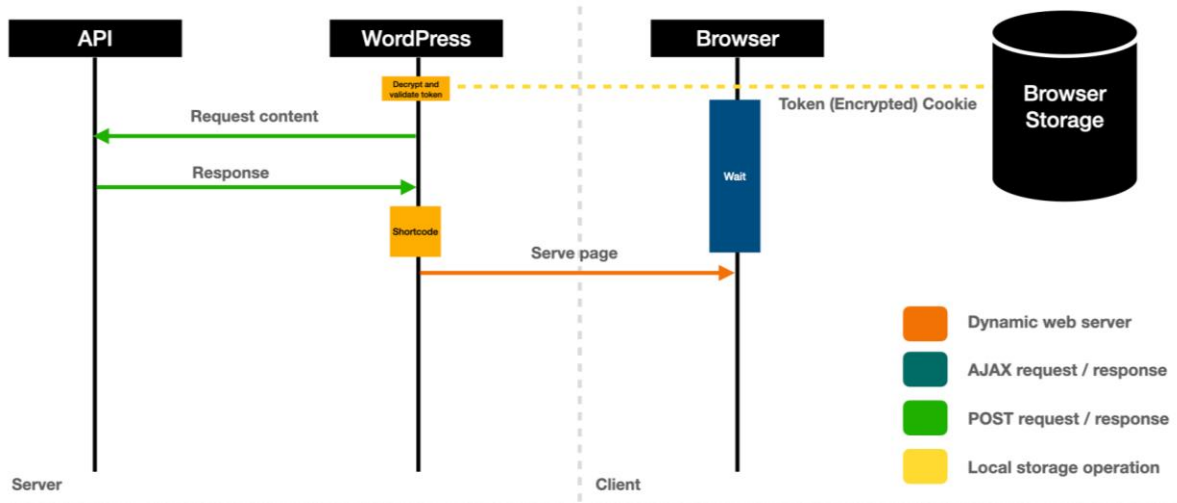


FIGURE 35 – TOKEN BASED ACTIONS

3 Source Code

3.1 Availability

This Section provides information with regards to the actual code repositories of the Data Marketplace.

3.1.1 Back-end

The software prototype of the Data Marketplace's back-end is provided in PolicyCLOUD's GitLab repository, but it will remain private for the consortium partners who have participated in the development process.

3.1.2 Front-end

The software prototype of the Data Marketplace's front-end is provided in PolicyCLOUD's GitLab repository, but it will remain private for the consortium partners who have participated in the development process.

3.2 Exploitation

This Section provides information about where the components of the Data Marketplace are deployed and how they can be accessed and run.

3.2.1 Back-end

As described in Section 3.1.1, the source code of the Data Marketplace's back-end is currently in a private GitLab repository, being exploited for private use and reuse in external research projects, as well as for private education purposes (e.g., MSc programmes, student hackathons).

3.2.2 Front-end

As described in Section 3.1.2, the source code of the Data Marketplace's front-end is currently in a private GitLab repository, being exploited for private use and reuse in external research projects, as well as for private education purposes (e.g., MSc programmes, student hackathons).

4 Conclusion

This deliverable described and analysed the final version of the implemented prototype of the Data Marketplace based on the design and the architecture specifications described in Section 2.1 in short, and in deliverables D7.4 and D7.11 in deeper detail.

Moreover, the interfaces of the components have been updated respectively using examples of HTTP requests that trigger specific actions of the Data Marketplace. In terms of the front-end, the final version of Data Marketplace's web pages was presented along with some descriptions about them.

Finally, the baseline technologies and tools that are used in the Data Marketplace's components were reported, specifying the status of both the availability and the potential for exploitation of the implemented source codes.

As for the next steps of the development of the PolicyCLOUD Data Marketplace, the latter will be enhanced with additional features and functionalities (e.g., automated updates with latest versions of domain-specific data assets, plagiarism and copyrights' detection tools for data assets that already exist in other platforms), and these will be offered with public access capabilities. The public access additions will be able to be efficiently integrated with other marketplace platforms upon specific guidelines with proper documentation. However, it should be noted that this will be an upcoming enhancement that is not planned to be provided within the context of the PolicyCLOUD project.

References

- [1] JSON Web Tokens (JWT), Homepage, <https://jwt.io>
- [2] Auth0, JSON Web Tokens, <https://auth0.com/docs/security/tokens/json-web-tokens>
- [3] Docker, Homepage, <https://www.docker.com>
- [4] Python, Homepage, <https://www.python.org>
- [5] The Pallets Projects, Flask, <https://palletsprojects.com/p/flask>
- [6] MongoDB, Homepage, <https://www.mongodb.com>
- [7] Gunicorn, Homepage, <https://gunicorn.org>
- [8] NGINX, Homepage, <https://www.nginx.com>
- [9] WordPress, Homepage, <https://wordpress.com>
- [10] Elementor, Homepage, <https://elementor.com>
- [11] cURL, Homepage, <https://curl.se>